EPSON

EPSON RC+ 7.0 Option

Part Feeding 7.0

Introduction & Software

Rev.10 ENM238S5890F

Original instructions

EPSON RC+ 7.0 Option   Part Feeding 7.0   Introduction & Software Rev.10

EPSON RC+ 7.0 Option

Part Feeding 7.0
Introduction & Software

Rev.10

FOREWORD

Thank you for purchasing our robot system.
This manual contains the information necessary for the correct use of the EPSON RC+ PartFeeding option.
Please carefully read this manual and other related manuals before installing the robot system.
Keep this manual handy for easy access at all times.

The robot system and its optional parts are shipped to our customers only after being subjected to the strictest quality controls, tests, and inspections to certify its compliance with our high performance standards. Please note that the basic performance of the product will not be exhibited if our robot system is used outside of the usage conditions and product specifications described in the manuals.

This manual describes possible dangers and consequences that we can foresee. Be sure to comply with safety precautions on this manual to use our robot system safety and correctly.

TRADEMARKS

TRADEMARK NOTATION IN THIS MANUAL

Microsoft® Windows® 8 operating system
Microsoft® Windows® 10 operating system
Microsoft® Windows® 11 operating system
Throughout this manual, Windows 8, Windows 10 and Windows 11 refer to above respective operating systems. In some cases, Windows refers generically to Windows 8, Windows 10 and Windows 11.

NOTICE

MANUFACTURER

SEIKO EPSON CORPORATION

CONTACT INFORMATION

Contact information is described in “SUPPLIERS” in the first pages of the following manual:
Robot System Safety Manual Read this manual first

Introduction

Troubleshooting

Introduction

1. Introduction

1.1 Overview of Part Feeding

The Part Feeding option for EPSON RC+ 7.0 (“Part Feeding option” hereinafter) can be used for easily developing a system in which parts are separated by a part feeder and a robot picks up the parts from the feeder.

1.1.1 Background

Types of manufacturing are becoming more diversified in response to the growing trends of shorter product life and multi-product diversification, “Just-in-time” manufacturing (small lots with short turnarounds), and similar factors. In contrast, factors such as continuously increasing wages, persons moving away from production site areas, and the aging of the worker population make it increasingly difficult to perform manufacturing, resulting in the issue of how to achieve the flexibility necessary to respond to diversification.

Let’s look at part feeding as one element in creating automated production lines.
Part feeding is actually a key element for production equipment because you cannot increase productivity beyond the part feeding capacity. Currently, the main method used for part feeding is the comparatively inexpensive vibrating bowl feeder which provides great diversity of use.
However, it is necessary to create bowls corresponding to the fed parts and, moreover, to manually replace bowls for each part type, which requires the coordination of specialized technical personnel for such purposes. These requirements make it difficult to respond to diversification and short turnaround times due to the necessity for advanced engineering capabilities and experience.

One device that resolves some of the shortcomings of a bowl feeder is the “intelligent parts feeder” (“feeder” hereinafter) that was first put in use a few years ago. This device allows you to easily handle various types of fed parts just by changing the feeder settings. There are also products that combine this feeder with image processing that is capable of identifying parts and a robot that handles the recognized parts.
However, these types of products require users to specify settings for themselves for feeder adjustment and operation, image processing, and robot parts handling because individual operations of the feeder can only be performed from the robot system. Time and experience are necessary for comprehensively studying and designing part feeding. For example, the cycle time greatly varies depending on the number of parts fed and the position where the robot picks up parts. Failure to provide a proper design will prevent the feeder from operating at full functionality and a high-performance level even if, for example, a high-performance/high-functioning feeder is used, and thereby prevent any improvement to the cycle time. These current feeder products cannot just be used right away by any person.

1.1.2 Merits of the Part Feeding Option

The Part Feeding option is a product that brings innovation to these current part feeders. Introducing this option provides the following merits.

- Complete Integration of Feeder, Vision System, and Robot
  All the necessary elements for part feeding are provided in a single package. The complete integration of the feeder, robot, and vision system requires less labor time at introduction in comparison with having to prepare and evaluate each of these separately.

- Reduced Labor Time for Device Development/Startup
  Feeder and vision system operations are performed automatically so that the user does not have to perform any programming. The robot operation needs to be described by the user, but this can be simply done by writing in a template code. The feeder, vision system, and robot operations are subject to automatic simultaneous control so that device development can be performed with minimal programming. It is simple to integrate into whatever environment the user is currently using.
  Feeder, robot, and vision system settings can be easily specified by using the GUI.

- Reduced Downtime and Running Costs
  The Part Feeding option supports a great variety of parts. There is no need to change equipment each time you change parts, thereby reducing production downtime.
  There is also no need for newly producing equipment to reduce long-term running costs.

1.1.3 Functions of the Part Feeding Option

Use of the following representative functions allows you to readily achieve full feeder functionality and performance to create a highly efficient part feeding system.

- Feeder and Communication I/F

  Communication programs for specifying feeder settings and performing control are incorporated into this system.
  You do not have to perform any programming for communication.

- Automatic Feeder Adjustment Function

  This system includes a function that automatically adjusts feeder parameters (such as vibration amplitude and time) according to the parts fed. Feeder adjustment can be performed easily even by persons with no knowledge of feeders just by performing a few simple procedures.

- Feeder Control Algorithms

  Algorithms for controlling the feeder are included in the system. These algorithms take into account reducing as much as possible the time required by the robot to pick parts in order to provide efficient operation with no effort on the part of the user.

- Cycle Time Log Output Function for Ascertaining Robot and Feeder Operating Status

  This system includes a function to output the robot, feeder, and vision system operating times to a file. This can be used for more efficient operation by changing the parameter settings, and then obtaining the log to analyze operation. The Controller must be connected to a PC with the EPSON RC+ installed in order to use this function.

- Multi-feeder operation

  Up to 4 feeders can be connected to and controlled by a single controller. For T/VT series, up to 2 feeders can be controlled. When you want to link multiple feeder operations, you can control them with a single controller, so it makes programming easier.

- Multi-part operation

  Up to four different parts can be processed in a single feeder at the same time. The number of feeders can be reduced, which results in lower costs and space savings. In multi-part operation, up to two robots can be used per feeder.

- Purge operation

  This system includes a built-in purge operation for parts on the feeder. When you want to automatically eject a part from the feeder by switching parts, or when you want to use it for damaged parts or overfeeding.
  The IF-80 includes a platform for the purge operation and a container for storing the ejected parts. (Optional)
  The IF-240, IF-380 and IF-530, the ejection mechanism is provided by the customer. This includes a door that opens and closes when the part is ejected, a mechanism to open and close the door, a container to store the ejected part etc.

1.2 Required Basic Knowledge of EPSON RC+ 7.0

The Part Feeding option operates within the EPSON RC+ 7.0 environment.

Knowledge of the EPSON RC+ 7.0 development environment, Epson's robots, and EPSON RC+ 7.0 Option Vision Guide 7.0 is required for using the Part Feeding option. The information in this manual is for persons with knowledge of the following items.

- General knowledge of the EPSON RC+ 7.0 project management and procedures for use
- Procedures for creating and editing SPEL+ programs using the EPSON RC+ 7.0
- Procedures for executing SPEL+ programs from the Run window
- Basic language structure, functions, and procedures for use of SPEL+
- Vision Guide 7.0 functions and procedures for use

1.3 Related Manuals

Refer to the following related manuals along with the Part Feeding option manuals for using the Part Feeding option.

*"EPSON RC+ 7.0 Option Part Feeding 7.0 IF-*** "*

***: Feeder robots (IF-80, IF-240, or IF-380/530)

This manual contains information on using each feeder robots.

"EPSON RC+ 7.0 Option Part Feeding 7.0 Hopper IF240, 380, 530"

This manual contains information on using hopper.

EPSON RC+7.0 User's Guide

This manual contains information on using the EPSON RC+ Robot Control System.

SPEL$^+$ Language Reference Manual

This manual contains a complete description of all commands for the SPEL$^+$ language.

Each Robot Manual

Each robot manual contains information on our robots.

1.4 Symbols Used in this Manual

In this manual, important matters are shown with the symbols below. Each symbol is described below.

⚠ WARNING	This symbol indicates that a danger of possible serious injury or death exists if the associated instructions are not followed properly.
⚠ WARNING	This symbol indicates that a danger of possible harm to people caused by electric shock exists if the associated instructions are not followed properly.
⚠ CAUTION	This symbol indicates that a danger of possible harm to people or physical damage to equipment and facilities exists if the associated instructions are not followed properly.

NOTE ☞	The “NOTE” sections describe important information to be followed for operating the Robot system.
TIP ☞	The “TIP” sections describe hints for easier or alternative operations.

2. Safety

Be sure to read this manual before use and operate the product accordingly in the proper manner.
After reading this manual, store it in a readily accessible location and refer to it whenever you have any questions or doubts.

2.1 Safety Precautions

⚠ WARNING	■ Do not use this product for the purpose of ensuring safety. ■ Use this product according to the use conditions indicated in this manual. Use of this product in an environment that does not satisfy the usage conditions can not only reduce product service life but may also result in serious safety problems.

⚠ CAUTION	■ Purchase the feeder from an authorized distributor. ■ Purchase the camera and camera cable from an authorized distributor. Components not purchased from an authorized distributor are not subject to warranty.

2.2 Robot Safety

Make safety the top priority when operating a robot and other automated equipment. The Controller and EPSON RC+ 7.0 are equipped with many safety functions. Be sure to use the various safety functions such as emergency stop and safety door input. Use these safety functions when designing robot cells.

For safety information and guidelines, refer to the *Safety* section in the *EPSON RC+ 7.0 Users Guide*.

2.3 Vision System Safety

For vision system safety, refer to section *2.2 Safety Precautions* in *Vision Guide 7.0 Hardware & Setup* manual.

2.4 Feeder Safety

For feeder safety, refer to following either manual of section “*1.2 Safety precautions*”.
EPSON RC+ 7.0 Option Part Feeding 7.0 IF-80
EPSON RC+ 7.0 Option Part Feeding 7.0 IF-240
EPSON RC+ 7.0 Option Part Feeding 7.0 IF-380, IF-530

2.5 Hopper Safety

For hopper safety, refer to the following sections.
EPSON RC+ 7.0 Option Part Feeding 7.0 Hopper IF240, 380, 530
“1.1 Safety Instructions for Hopper”
「1.2 Safety Instructions for Hopper Controller」

3. Definition of Terms

The following terms are used in the manual as defined below.

Hardware

Term	Explanation
Feeder	Equipment that vibrates bulk parts to separate them for easy transfer to by the robot.
Platform	Component part of the feeder that serves as a parts-receiving tray.
Parts	Parts handled by the robot. Provided by the user.
Add-in feeding	Feeding method in which parts are added to the feeder from the hopper in order to constantly maintain an optimal number of parts in the feeder.
Run-out feeding	Feeding method in which parts are fed from the hopper to the feeder only after all the parts in the feeder have been fed.
Pick from anywhere	All parts that can be picked among the separated parts in the feeder are picked.
Pick from region	Parts within the specified regions of the separated parts in the feeder are picked. With the Part Feeding option, you can select from four regions defined in the feeder from which to perform picking.
User lighting	Lighting provided by the user. Use this lighting if there are parts that cannot be identified by the feeder backlight or you want to identify the orientation of a part (such as front/back).
Hopper	Equipment that supplies parts to the feeder platform.
Purging	To purge the parts remaining on the feeder.
Purging Gate	The gate that opens and closes to purge the parts remaining on the feeder. It is connected to the feeder, and opening and closing can be controlled by commands. It can be used to switch part types for high-mix low-volume production and to purge defective parts.
Feeder calibration	Procedures for adjusting the feeder parameters so that parts are moved properly in the feeder.
Multi-feeder	Multiple feeders can be connected to a single controller. Up to four feeders can be supported with this option.
Multi-part	Process multiple types of parts in a single feeder at the same time. Up to four different parts can be supported with this option.

Software

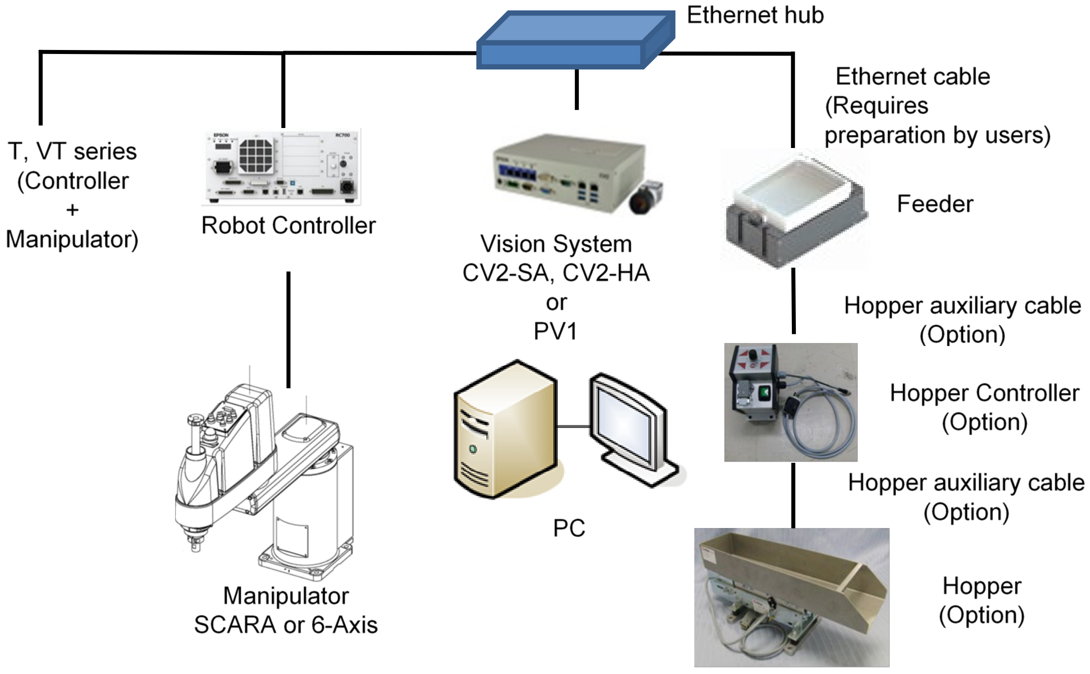
Term	Explanation
Pick	Refers to the gripping of a part in the feeder by the robot.
Place	Refers to when a part gripped by the robot is dropped or placed at the specified position.
Part Feeding process	This automatic process is built into the Part Feeding option to provide automated vision system and feeder operation, and invoke robot operations.
Callback function	SPEL$^+$ function invoked by the Part Feeding process under the specified conditions. Function content is indicated by the user. The processing specific to the user's device is described (example: picking and placing of parts by the robot).
Part coordinates queue	Used for obtaining coordinates for parts in the feeder. Coordinates are defined in the local coordinate system.
UPM	Unit per minute Number of parts handled by robot per minute.
Active part	Main part of a multi-part operation. The feeder operation parameters of this part are used.

4. System Overview

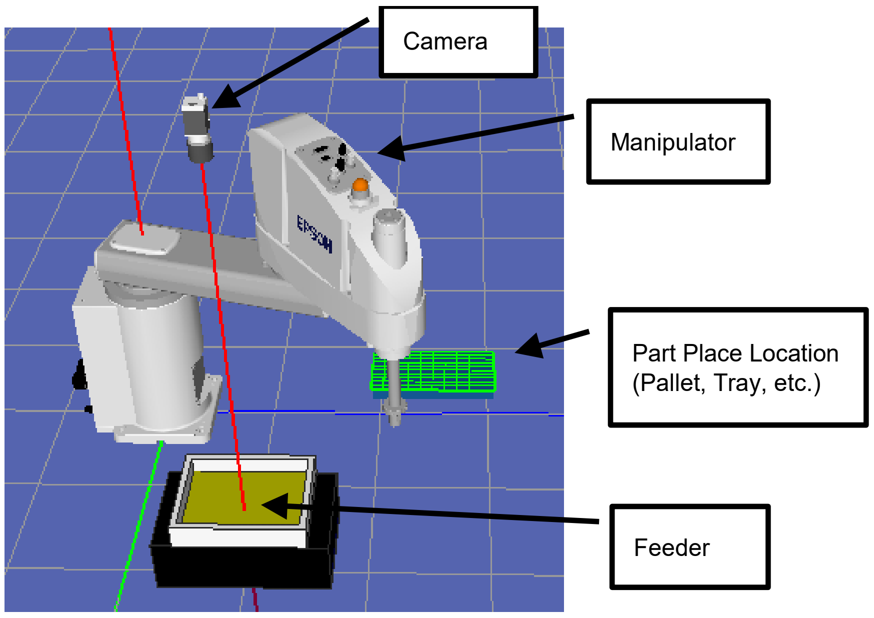
Use the Part Feeding option to easily create a system for picking and placing parts.
This section explains how to configure a system.

4.1 Overall Configuration

The figure below shows an example configuration of a system using the Part Feeding option. Feeder IF-80 has built-in hopper .



The figure shows an example layout around the Manipulator.



4.2 Feeder

Use of a feeder is necessary for this system. Be sure to prepare one to use.

The IF-80, IF-240, IF-380 or IF-530 can be used. No other feeder type is supported.

Multiple feeders can be controlled by one Controller.

Use a feeder purchased from Epson.
Use of a feeder not purchased from Epson can result in inability to connect with the Controller and failure to achieve full functionality.

4.3 Robot

Use of a robot is necessary for this system. Be sure to prepare one to use.

4.3.1 Manipulator

A SCARA or 6-axis Manipulator that can be connected to an RC700 series or RC90 series Controller or T/VT series can be used. X5 series and PG are not supported.

One Manipulator can only be controlled by one Controller. You cannot use one Controller for two or more Manipulators.

4.3.2 End Effector

An end effector is used to grasp parts. There are various types such as those using suction by vacuum/pressure and those using a chuck mechanism. Select one appropriate to your parts and needs.

Epson does not manufacture or distribute any end effectors. Be sure to prepare proper hands for use.

4.4 Vision System

A vision system is necessary for this system. Be sure to prepare one to use.

4.4.1 Vision System

You can use any of the following vision systems.

- PC Vision PV1 (For required specifications, refer to *4.6. PC*.)

- Compact Vision CV2-SA, CV2-HA (Firmware Ver. 3.0.0.0 or later)

CV1 or CV2-S/H/L is not supported.

Vision systems of other manufacturers are not supported.

4.4.2 Camera

You can use any cameras that can be connected to the vision system.

Be sure to provide one camera to find parts in the feeder. The camera will be installed either as a fixed downward camera or a mobile camera mounted to a movable axis of the robot.

You can also add other cameras such as one facing upward for correcting the position of grasped parts or one for positioning parts in their place position.

4.5 Lighting

Lighting is necessary for accurately identifying parts on the platform.

You can use either or both of the following lighting methods.

- Backlight incorporated into feeder
- User's own custom lighting (I/O control, Ethernet control, etc.)

4.6 PC

A PC is required for the following tasks.

- Viewing/Editing of Part Feeding option settings
- Feeder calibration
- SPEL+ programming
- Log retrieval

The Part Feeding process operations can be performed without connecting a PC.
Required PC specifications are as follows.

- When using CV: Must be possible to install the EPSON RC+
- When using PV: Per *4.3.1 System Requirements* of the *Vision Guide 7.0 Hardware & Setup* manual

4.7 Hopper

The hopper supplies parts to the feeder. It is an optional device.

Hoppers purchased from Epson can be connected to the feeder.

Connect any hopper you provide to the Controller via I/O or Ethernet.

Feeder IF-80 has built-in hopper.

5. Hardware

5.1 Check Included Items

Depending on your specific purchase order, the following items are included for the Part Feeding option.

After the items arrive, please check that all items are included and there is no damage to the items.

- Part Feeding license document (There are cases in which it may be shipped separately.)
- Feeder with embedded Backlight
- Platform
- Power cable for the feeder
- Ethernet cable

Optional items are as follows:

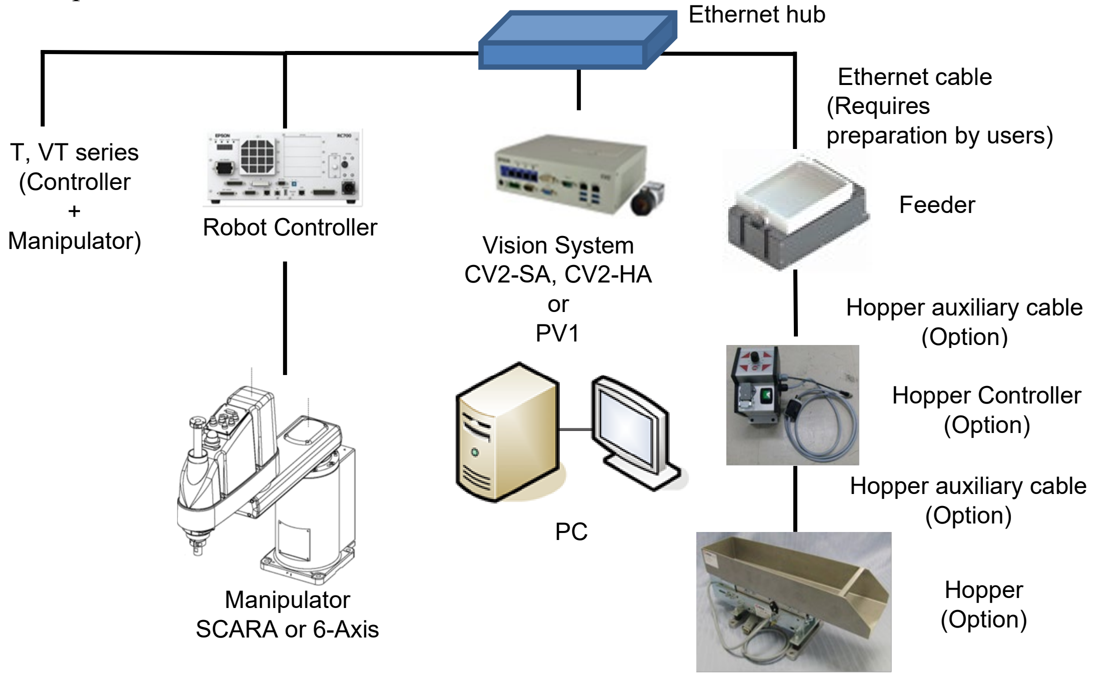
- Hopper
- Hopper controller
- Hopper attached cable
- Others Optional items

5.2 System Configuration

5.2.1 Configuration Example

The illustration below shows the system configuration using the PartFeeding option.
Feeder IF-80 has built-in hopper .

¥



Up to four feeders can be installed in one controller.
For T/VT series, up to two feeders can be controlled at the same time.

In the case of multi-part operation, up to two manipulators can be installed per feeder.

Ethernet should be connected using Ethernet hub.
Use the following vision system.

Compact Vision CV2A series	CV2-HA (Firmware Ver.3.0.0.0 or later)
	CV2-SA (Firmware Ver.3.0.0.0 or later)
PC Vision PV1	

CV1 or CV2-S/H/L is not supported.

Vision systems of other manufacturers are not supported.

5.2.2 Considerations for Configuration Selection

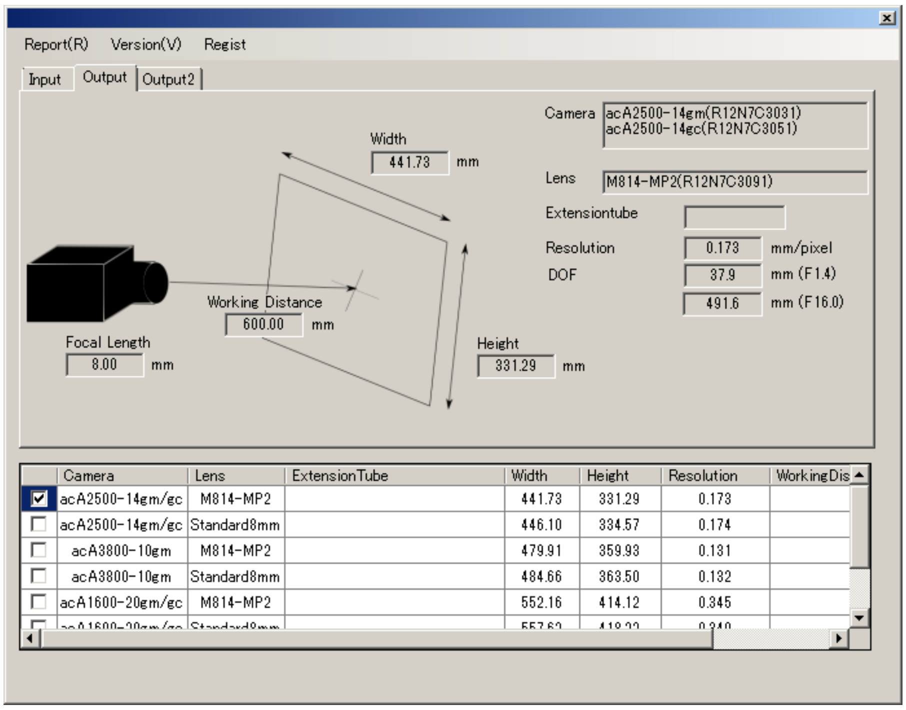
- Available manipulators for the Part Feeding option are SCARA and 6-axis robot series. This option does not support PG and X5 series robots.
- When using PV (PC Vision), make sure not to perform the feeder's communication and the camera's communication by the same network port. If the feeder's communication and the camera's communication are performed simultaneously, it may affect imaging or feeder operations. We recommend adding the network card to PC or using the network card with multi-port and connect the camera and the network port directly.
- No error occurs even if you connect more than one robot controller to a feeder. However, pay attention to IP address assignment and be careful not to connect more than one robot controller to a feeder when you set the network.
- Up to two hoppers can be installed in one feeder. However, it is not possible to control two feeders at the same time.
- It is possible to display CAD data of the feeder or hopper on Simulator. The CAD data file is in following folder.
  C: \EpsonRC70\Simulator\CAD\PartFeeder

5.2.3 Select a Camera Lens

Select a lens and an extension tube based on field of view and working distance by using the Vision Guide attached camera select tool.

Camera select tool:

C:\EpsonRC70\Tools\CamSelectTool\CamSelectTool.exe



Camera: acA2500-14gm(R12N7C3031) acA2500-14gc(R12N7C3051)

Lens: M814-MP2(R12N7C3091)

Extensiontube:

Resolution: 0.173 mm/pixel

DOF: 37.9 mm (F1.4)

491.6 mm (F16.0)

Width: 441.73 mm

Height: 331.29 mm

Working Distance: 600.00 mm

Focal Length: 8.00 mm

	Camera	Lens	ExtensionTube	Width	Height	Resolution	WorkingDis
✓	acA2500-14gm/gc	M814-MP2		441.73	331.29	0.173	
☐	acA2500-14gm/gc	Standard8mm		446.10	334.57	0.174	
☐	acA3800-10gm	M814-MP2		479.91	359.93	0.131	
☐	acA3800-10gm	Standard8mm		484.66	363.50	0.132	
☐	acA1600-20gm/gc	M814-MP2		552.16	414.12	0.345	

5.3 Installation and Adjustment

5.3.1 Manipulator and Controller

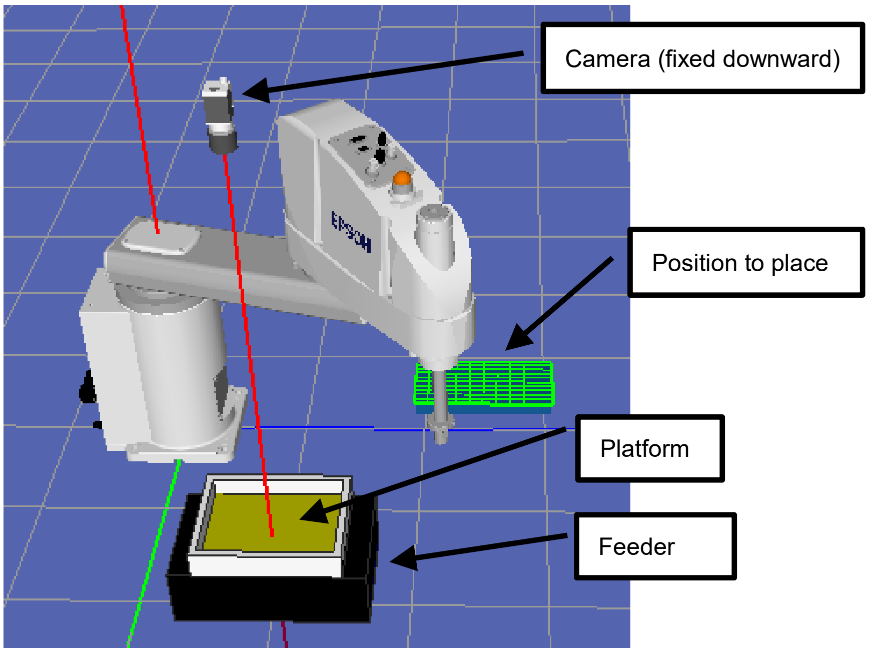
For installation of manipulator, refer to each manipulator's manual. Install the manipulator properly according to *Robot System Safety manual.*

Please make or prepare a robot hand (gripper).

For installation of the controller, refer to the controller manual.

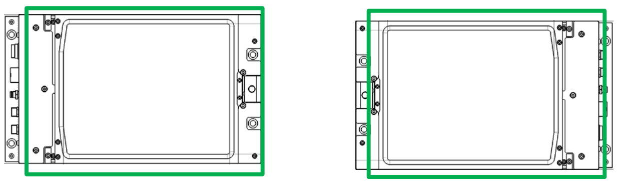
5.3.2 Camera and Lens

Install the downward facing camera so that it can display the entire feeder platform. We recommend using a fixed downward camera. Support for a mobile camera is also provided.

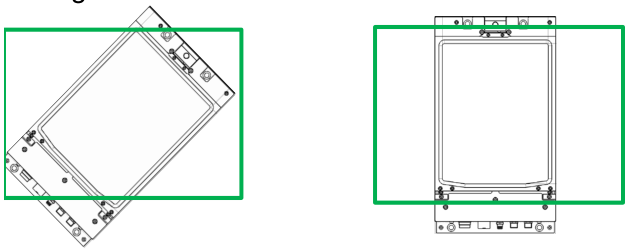


The feeder's longitudinal direction and the horizontal direction of the camera's field of view (green squares below) need to be matched. Otherwise, the PartFeeding system will not work properly. You can set the feeder regardless of its direction.

Correct

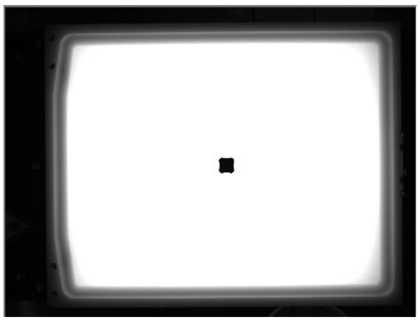


Wrong



When using a mobile camera, teach (create a point data) a position where the feeder's longitudinal direction and the horizontal direction of the camera's field of view match as shown above. Then, set it as the imaging position.

Adjust focus and aperture of the camera lens so that the camera can recognize the part clearly and the brightness of the platform will be uniform when taking the part image.



5.3.3 Feeder and Hopper

Please note the following points of installation.

- Mount on a horizontal surface.
- Mount on a solid underground.
- Fasten tightly with four M6 screws.

If the installation surface is not flat and level, or stiffness of the mounting base is low, parts will not be dispersed fully. The number of the parts that can be picked up decrease and it may result in lowering the cycle time.
For details on the installation, refer to the following each feeder manuals.
*EPSON RC+ 7.0 Option Part Feeding 7.0 IF-*** "3. Environment and Installation"*

****: Feeder robot (IF-80, IF-240, or IF-380, IF-530)*

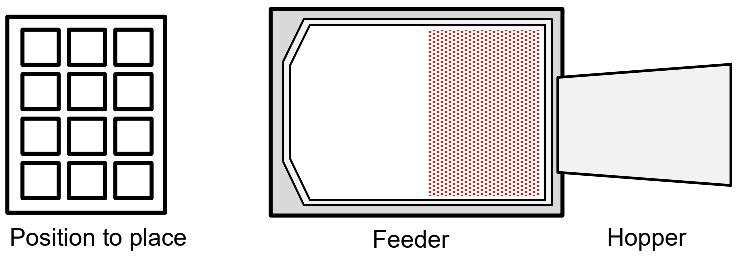
Install the hopper on the flat and level surface. Securely fix it on the base with high stiffness.
Be careful not to set the hopper over the feeder platform. By making the platform access as wide as possible, the number of the parts that can be picked up increase and the cycle time improves.
For more details on installation or adjustment, refer to following manual.

EPSON RC+ 7.0 Option Part Feeding 7.0 Hopper IF240, 380, 530

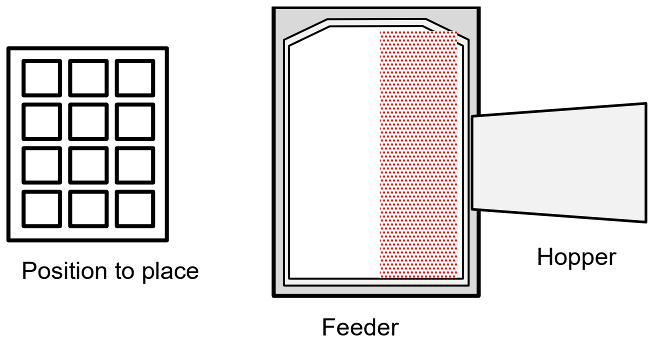
「2.4 Names of each Part and Outer Dimension, Mounting Dimension」

Set the hopper so that the position to feed parts on the feeder faces to the position to place. Feed the parts to the red shaded area in the following examples. By setting the hopper as shown below, you can use the parallel feeding function and the cycle time is improved.

Alignment example 1

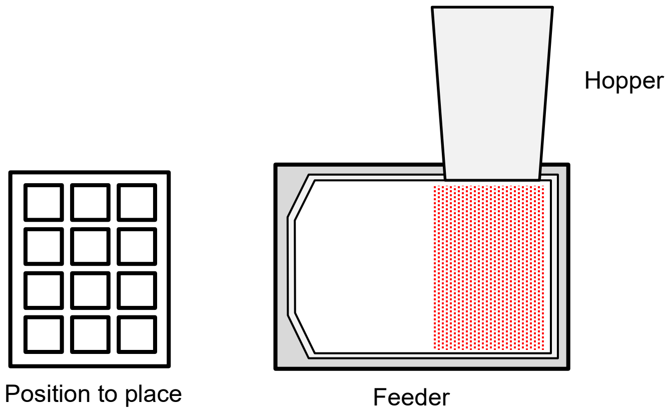


Alignment example 2



If a fed part rolls into the area where the robot picks up the parts, it may result in a bad effect on the pick-up operation. To prevent this, the following alignment is effective.

Alignment example 3



When feeding the parts from the hopper, feeding the proper amount to the proper position is important. Therefore, please consider the following:

- To adjust the part feeding amount, make a dam structure on the hopper.
- To control the part feeding position, make a guide at the end of the hopper.

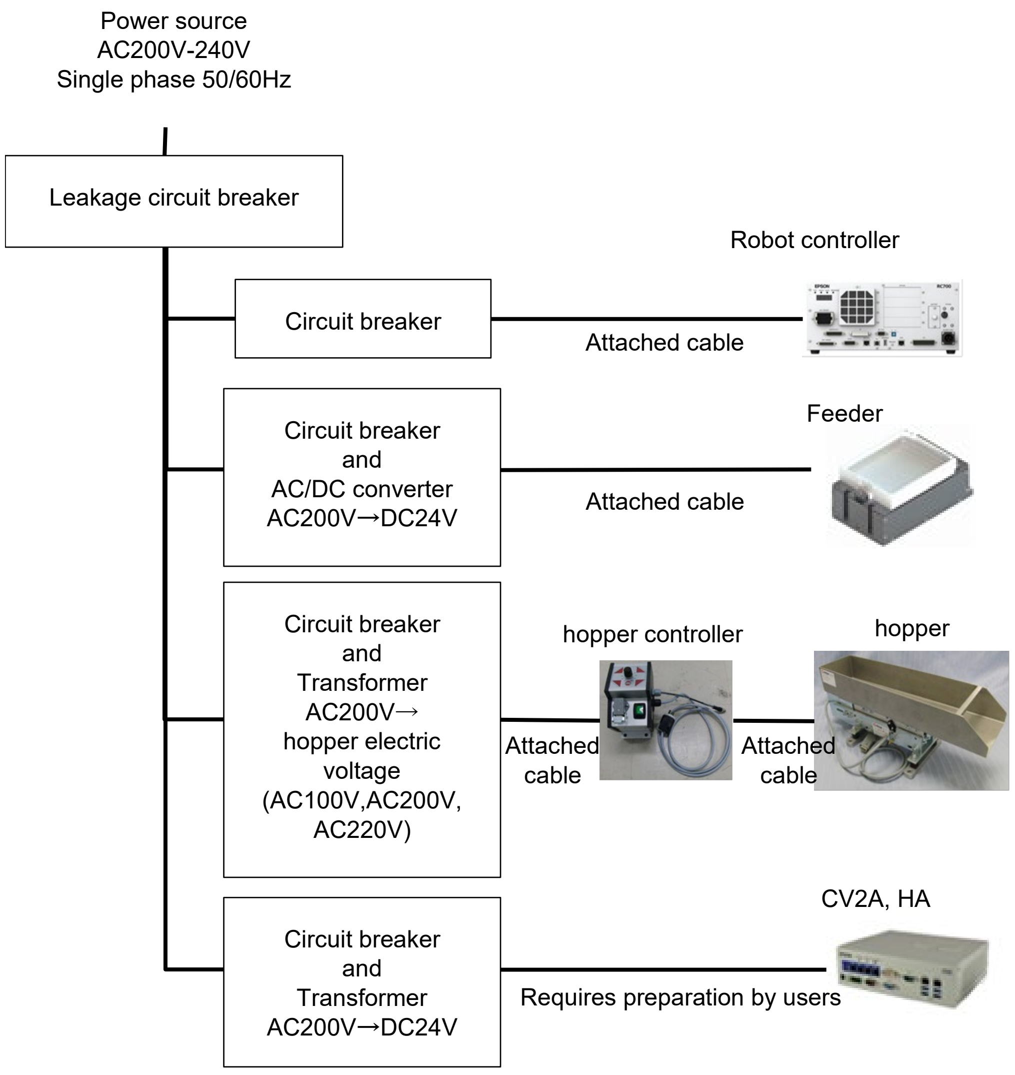
5.4 Electric Wiring

5.4.1 Cautions for Power Supply

The following describes cautions for the power supply to the robot controller, CV2A, feeder, and hopper.

According to *JIS B 9960-1(IEC 60204-1) Safety of machinery - Electrical equipment of machines - 5.1 Incoming supply conductor terminations*, we recommend connecting electric equipment of the machine designed by the user to a single power source. When using a different power source from the input power source on the particular part such as a feeder or a hopper, we recommend supplying the power from an electric transformer or a converter inside the electric equipment of the machine.

When connecting to single AC200V power source, the following is an example of electronic equipment which is designed by user. For more details, refer to *JIS B 9960-1(IEC 60204-1) Safety of machinery -Electrical equipment of machines-*.

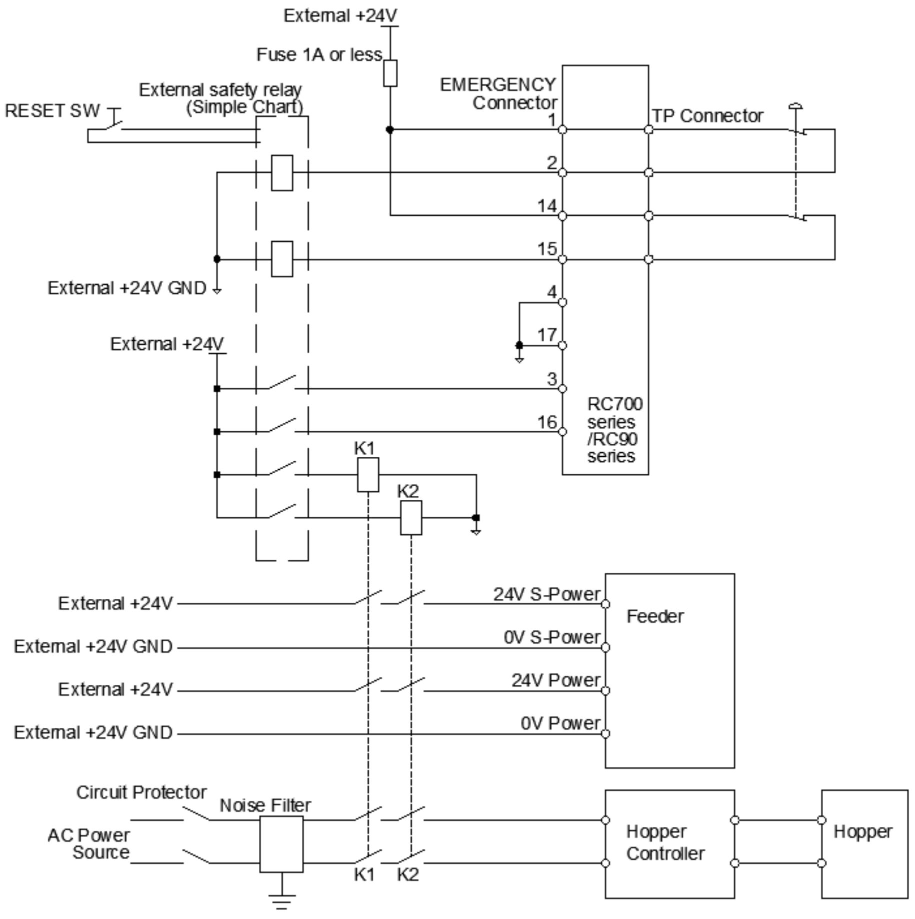


5.4.2 Power Wiring for the Feeder

For details on wiring, refer to following manual.
*EPSON RC+ 7.0 Option Part Feeding 7.0 IF-*** "3.6 Connecting Cable"*

****: Feeder robot (IF-80, IF-240, or IF-380, IF-530)*

We recommend designing a circuit which turns OFF the power of the feeder and the hopper by external safety relay when holding down the emergency stop switch. Refer to *Controller Manual "Emergency" - Circuit Diagrams*.
Reference diagram is as follows:



5.4.3 Power wiring for the Hopper

For more details on wiring, refer to following manual.
EPSON RC+ 7.0 Option Part Feeding 7.0 Hopper IF240, 380, 530

"1.2.3 Safety Instructions of AC Power Cable"

5.4.4 Robot Wiring

Refer to manuals of each robot and controller to perform the wiring.

5.4.5 Camera Wiring

Refer to the following manual to perform wiring.
EPSON RC+ 7.0 Option Vision Guide Hardware

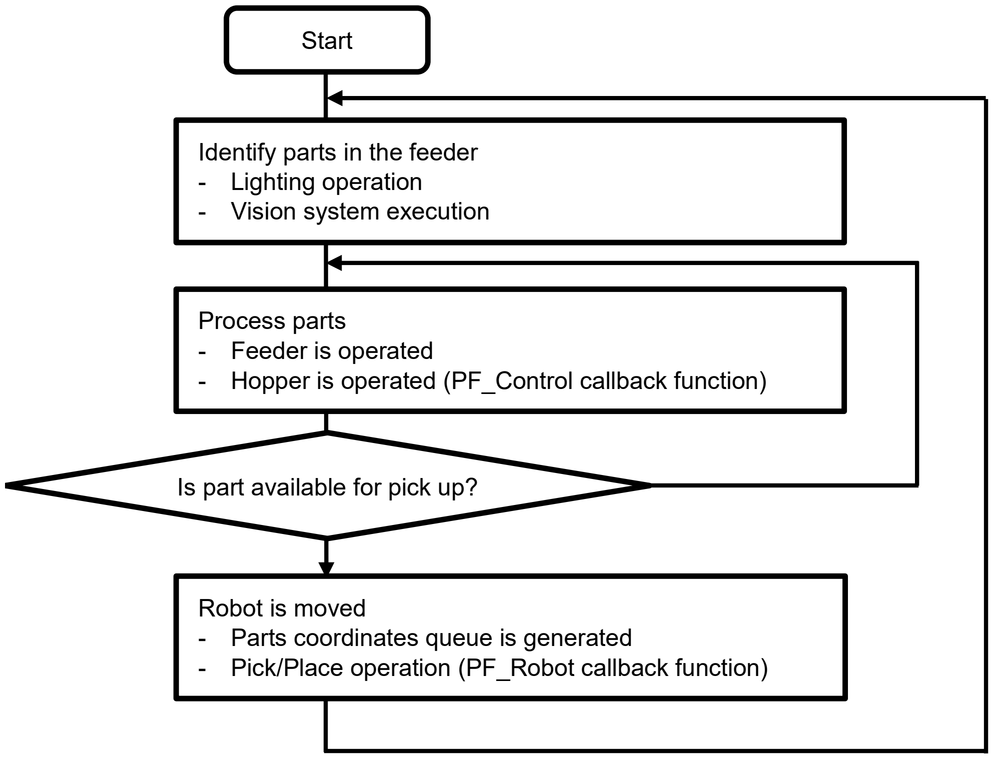
6. Operation Overview

This section provides an overview of the operations of the Part Feeding option.

6.1 Part Feeding Process

The Part Feeding process is the process of operations incorporated into the Part Feeding system in which the vision system and feeder operations are automatically controlled. You can start the Part Feeding process by executing the PF_Start command from your program.

The Part Feeding process is as shown below.


Start
Identify parts in the feeder
- Lighting operation
- Vision system execution
Process parts
- Feeder is operated
- Hopper is operated (PF_Control callback function)
Is part available for pick up?
Robot is moved
- Parts coordinates queue is generated
- Pick/Place operation (PF_Robot callback function)

1. Identify parts in the feeder
   Use the vision system to check the quantity and distribution of parts on the platform.
2. Process parts
   The feeder is controlled and parts are moved in order to make it easier for them to be grasped by the robot. When the parts are low in quantity or there are none remaining, the PF_Control callback function is invoked to supply parts from the hopper.
3. Robot is moved
   A parts coordinates queue (list of coordinates of parts in the feeder) is generated. The PF_Robot callback function is invoked to perform pick/place of parts.

The Part Feeding process can be stopped by invoking the PF_Stop command from your program.

6.2 Supplying Parts to the Feeder

The following methods are used for supplying parts to the feeder.

- Using the hopper
- Manually supply

6.2.1 Quantities of Supplied Parts

The quantity of parts supplied to the platform is an important element in determining operation cycle time.

- An excessively large quantity of parts negatively affects the cycle time due to parts overlapping and the need to vibrate the feeder numerous times.
- An excessively small quantity of parts also negatively affect the cycle time because you have to supply parts to the platform numerous times.

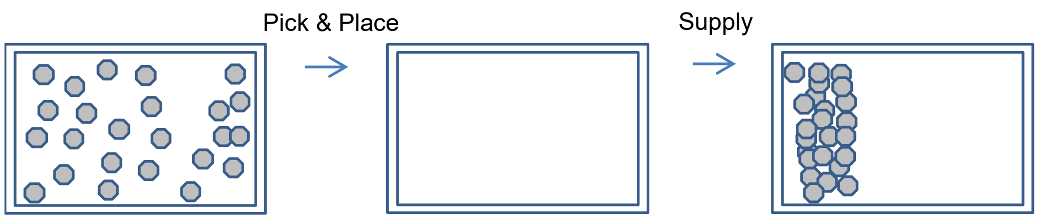
Accordingly, there is an ideal quantity of parts to be supplied (quantity of parts on the platform after feeder operation). This quantity can be calculated during feeder calibration.

The three methods (timing-related) described below can be used for supplying parts to the feeder.

1. Run-Out Feeding

Parts are supplied from the hopper to the feeder only after all the parts in the feeder have been fed.

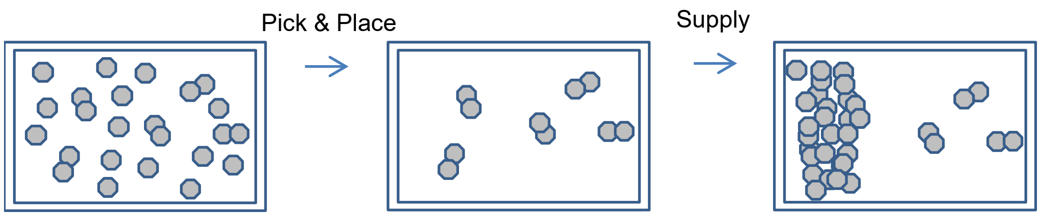
We recommend using this method for parts that are sensitive to vibration (easily affected by vibration) because retention time of parts in the feeder maintains fairly uniform. However, this method increases the cycle time because it reduces the average quantity of parts that can be retrieved by a robot during one feeder operation.



2. Add-In Feeding

Parts are additionally supplied from the hopper when all the parts in the feeder that can be retrieved are depleted.

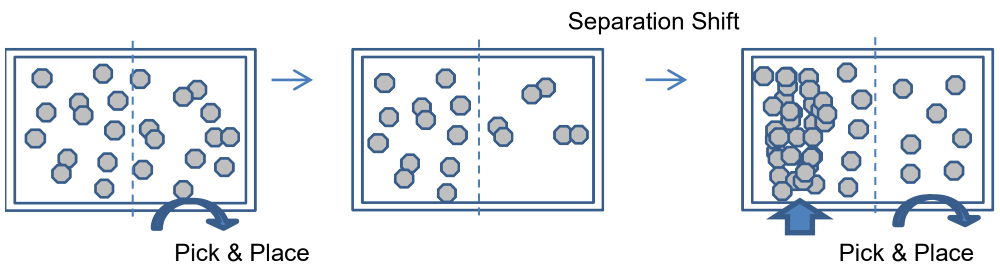
Cycle time is shortened because the average quantity of parts that can be retrieved by a robot during one feeder operation is relatively large, thereby improving efficiency. However, this method cannot be used for parts that are sensitive to vibration because the parts retention time can be long.



3. Parallel Feeding

This method is used together with a function for specifying the pick locations of parts.

While the robot is picking parts, parts are added to the region on the opposite side of the parts picking position. This method decreases the cycle time because the average quantity of parts that can be retrieved by a robot during one feeder operation is increased.
Cycle time is further reduced because both the hopper and robot can operate in parallel (simultaneously). However, this method cannot be used for parts that are sensitive to vibration because some parts remain in the feeder for a long period of time. It is necessary to write your own program so that the hopper and robot can operate in parallel.



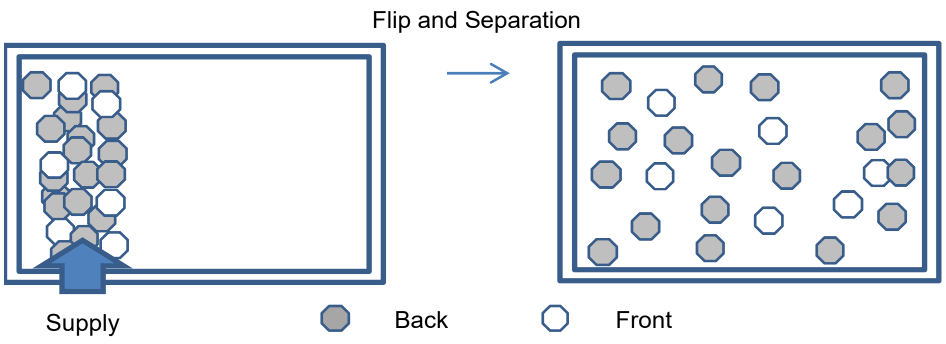
6.3 Feeder Operation

The Part Feeding option is executed by selecting feeder operation automatically depending on the conditions of the parts in the feeder. This makes it easier for the robot to grasp parts.

Feeder operations are described below. Explanatory drawing shows the outline.
It might be different from actual movement.

6.3.1 Flip and Separation

Parts are spaced out and flipped appropriately so that they can be easily grasped by the robot.

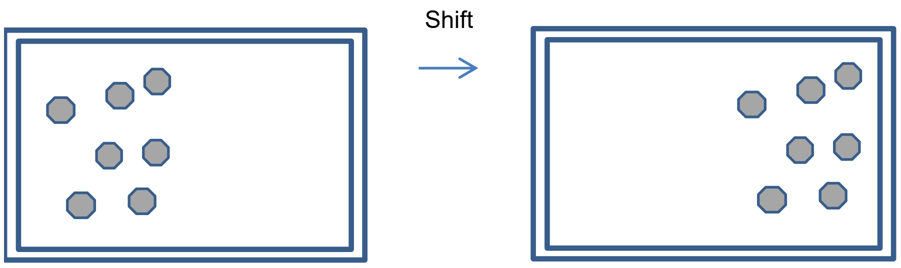


There may be an operation to move parts to the center before flip and separation.
This is called centering.

6.3.2 Shift

Moves all parts in one direction while maintaining the spacing (distribution) of the parts.

Moving parts closer to the place position reduces the robot movement distance and improves the cycle time.



Shift can be performed either forwards (closer to the pick position) or backwards (further from the pick position).

6.4 Part Pick Positions on Platform

The following two positions are available for parts picked by the robot.

To determine which position, pick all or pick some, is most efficient for your system, we recommends first performing actual operation and retrieving the log because the efficiency of the position used depends on your equipment configuration such as the parts, end effector, and hopper you are using.

6.4.1 Pick From Anywhere

Picking is performed for all parts on the entire surface of the platform.

Select this if parts are large for the size of the platform (e.g. approx. 2 cm^2 or more for the IF-240).

6.4.2 Pick From Region

Picking is performed in a region near the place position.

Using this method, shifting of parts to the picking region is performed automatically according to the distribution of parts. Robot operation in parallel with parts supply is possible within any region where picking is not performed (this is performed by user's own program). Use of this function generally reduces the robot cycle time in comparison with the pick all method.

6.5 Preventing End Effector and Platform Interference

To prevent physical interference of the end effector by the platform, it is necessary to specify the region where it is possible to pick parts in the feeder so that it falls within the outer circumference of the platform. That distance can be easily specified using the Part Feeding option. Parts that are detected to be too close to the feeder tray are not added to the part queue for pick up.

7. Parts

This section explains the parts that can be used with the Part Feeding option.

NOTE

Epson has created a system for evaluating whether your parts are compatible with this Part Feeding option at our authorized distributors. For details, please contact an authorized distributor.

7.1 Conditions for Usable Parts

Parts that can be used with the feeder are subject to the conditions indicated below.

7.1.1 Vision System Compatibility

It is necessary that parts can be correctly identified by the vision system.

- It may not be possible to identify the shape of parts made of transparent plastic because light passes through them. Such cases can possibly be resolved by changing the lighting to outside the visible spectrum or by using reflected lighting.
- It might not be possible to identify parts by front/back due to their shape. Such cases can possibly be resolved by adding reflected lighting.

7.1.2 Size and Weight

Larger part size reduces the quantity of parts that fit on the platform (when spread out across the platform with no overlapping). If this quantity is small, the number of feeder operations increases resulting in a relative reduction of the robot operating time, thereby negatively affecting cycle time. The ideal quantity of parts is 50 pieces or more to be loaded to the feeder.

The gross weight of parts (Weight of one part × Quantity of parts that can fit on the platform with no overlapping) must be less than the load capacity of the feeder. Exceeding this weight overloads the feeder, negatively affecting the ability to separate parts and the cycle time, as well as reducing the service life of the feeder.

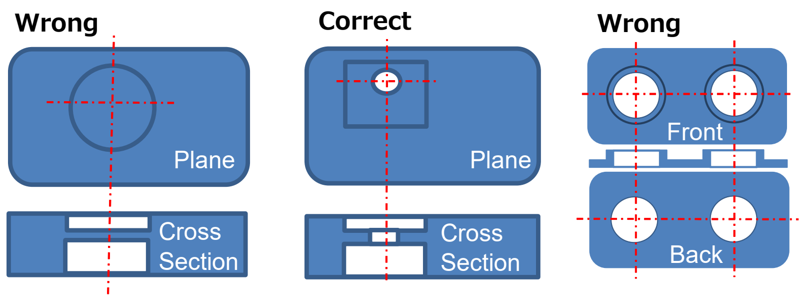
For the feeder load capacity, refer to the corresponding feeder manual.

7.1.3 Materials and Characteristics

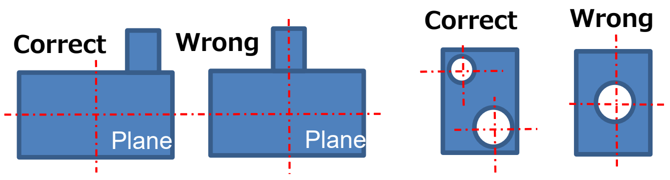
- Parts made of flexible or light material are difficult to use.
  Examples: Paper and fibrous materials
- Parts easily damaged or deformed by vibrated parts that generate dust when rubbed are difficult to use.
  Examples: Parts made of hardened powder or that are painted
- Parts that are sticky or leaking liquid are difficult to use.
  Examples: Food products

7.1.4 Parts Shape and Other

- It is difficult to pick spherical parts because they do not remain still in the feeder.
  Examples: Bearing balls
- Parts that easily become entangled are difficult to separate.
  Examples: Coil Springs
- It may not be possible to identify front/back of parts with different cross-sectional shapes made of material that is not transparent by using transmitted light. The following are some examples of such parts.

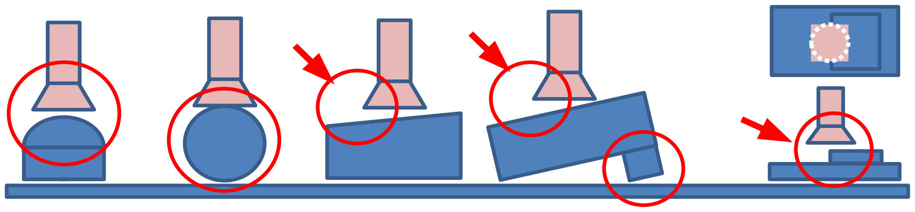


Parts with different cross-sectional shapes cannot be selected



Parts with different external shape of plane surfaces can be selected by front/back

- The best part to use when picking up parts by use of a suction system is one that has the suction surface that is parallel to the feeder base where the suction pad can move straight downward when picking up the part, and that secures the surface area for the suction pad. The following are some examples of such parts.

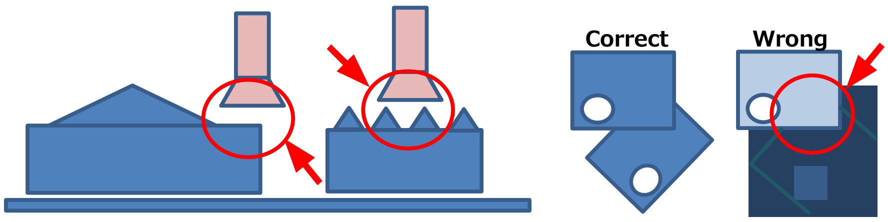


Ball

Parts that prevent the suction surface being parallel to the feeder base

If suction position is an uneven surface

- If using a suction method for feeding, it is best if the parts do not have any differences in their center of gravity. The following are some examples of such parts.



Parts that prevent a sufficient suction surface

Parts that cannot ensure a smooth surface (If using suction for feeding parts)

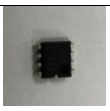
Parts where light passes through from below

- If used in an assembly process, it is best to provide some measure to prevent the part position from becoming misaligned after being picked up. This measure can consist of making guide holes in the parts and placing a pin on the end effector to prevent the position from becoming misaligned. Other measures consist of installing an upward-facing fixed camera to align part position after picking.

7.2 Examples of Parts

Some examples of parts that can be used with the Part Feeding option are shown below.

Parts No. 1 to 3 are appropriate for the IF-240.
No. 4 and 5 are not appropriate for IF-240 because they are too large and heavy.
No.6 and 7 are not appropriate for IF-240 or IF-380 or IF-530 because they are too small and light.

No	Photo	Characteristic	Size [mm]	Weight [g]	Comment
1		Metal press part	10 × 10 × 0.2	0.088	IF-240 is suitable
2		Metal press part	11 × 5.5 × 0.2	0.029	IF-240 is suitable
3		Plastic part	10 × 9 × 2.1	0.127	IF-240 is suitable
4		Nylon connector	21 × 29.9 × 21	7.1	IF-380 is suitable
5		Long nut	36 × 11 × 9.5	14.0	IF-380 or IF-530 are suitable
6		IC	5 × 4.4 × 1.5	0.082	IF-80 is suitable
7		Metal Bush	ø4 × 1	0.102	IF-80 is suitable

Note that No. 1 and 2 cannot be identified front/back by use of backlight only.

No. 3 can be identified front/back by use of backlight only.

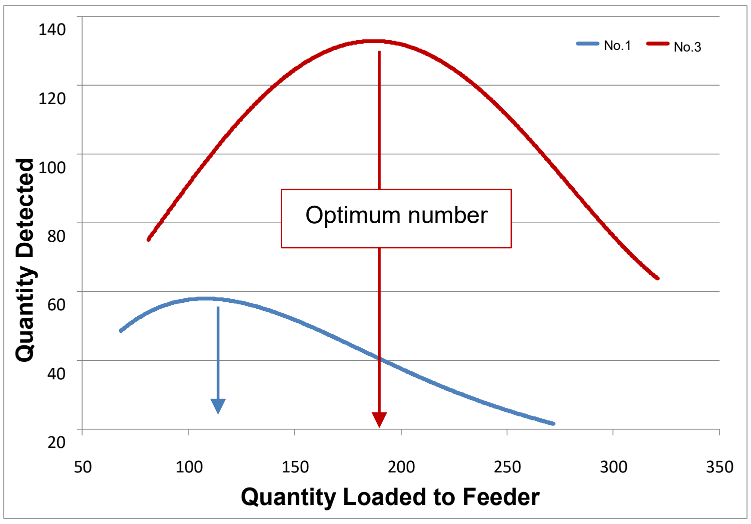
7.2.1 Relation of the Quantity of Parts Loaded to the Feeder and the Quantity Detected by Image Processing

The relation between the quantity of parts loaded to the feeder and the quantity detected by image processing is upward-convex shaped.

It is not possible to detect all parts in the feeder if the parts are such that they contact neighboring parts and overlap each other when loaded. The graph varies depending on how easily the parts contact neighboring ones or how easily they become overlapped. You can use the Part Feeding option to apply the optimal quantity of parts to load by use of calibration based on testing performed by Epson.

The graph shows the relation between the quantity of parts loaded to the feeder and the quantity detected by image processing for parts No. 1 and 3.

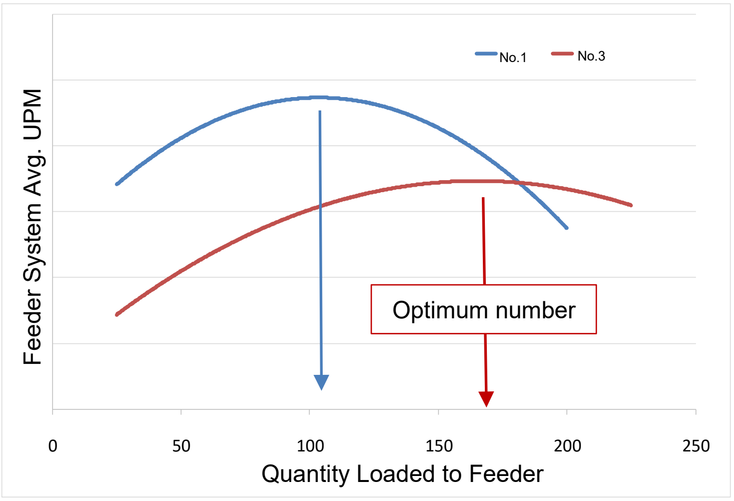
Be sure to pay attention to the facts that not all parts loaded to the feeder can be detected, and that the quantity detected changes depending on the quantity loaded.



7.2.2 Relation of Quantity Loaded to Feeder and Average UPM

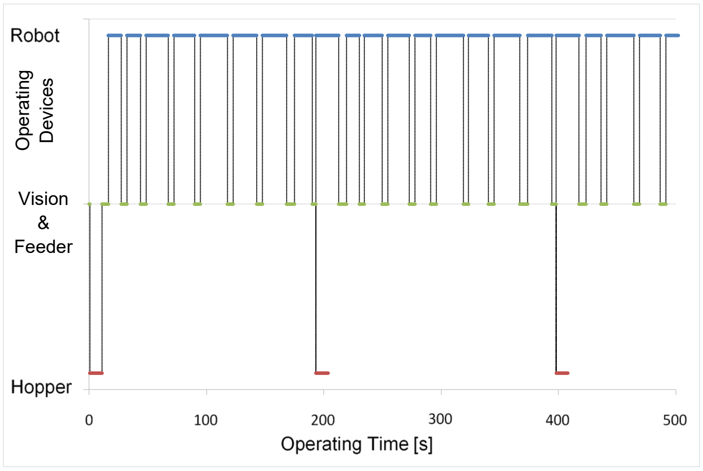
The average quantity of parts picked up for a certain amount of time when performing pick-up operation is the average Units Per Minute (UPM). UPM is number of parts handled by robot per minute.

The graph shows the relation between the quantity of parts loaded to the feeder and the average UPM for parts No. 1 and 3. Both the graphs for parts No. 1 and 3 are upward-convex shaped. Values for the vertical axis are not indicated because the average UPM varies depending on the robot speed, acceleration, and movement amount. In addition to robot operating conditions, it is important to keep in mind that the UPM changes depending on the quantity of parts loaded to the feeder and that there is an optimal quantity of parts to load to the feeder to increase the UPM.



7.2.3 Relation Between Feeder Operation and UPM

The operation timing of each operating device (Robot, Vision & Feeder, and Hopper) is plotted in the figure below with time plotted along the horizontal axis. When operation starts, Vision & Feeder operation detect there are no parts in the feeder, resulting in the hopper being moved to load parts to the feeder.



After this, the feeder operates, parts are separated, detection is performed by the vision system, and the robot then picks up the parts. When all parts that can be picked up are gone, parts are once again separated and detected by Vision & Feeder operation. After that, the robot operates again.

The quantity of parts in the feeder decreases as Vision & Feeder operation and Robot operation are repeated. The hopper is operated to feed parts according to the timing for hopper operation corresponding to the specified threshold value. The graph presupposes parallel feeding operation.

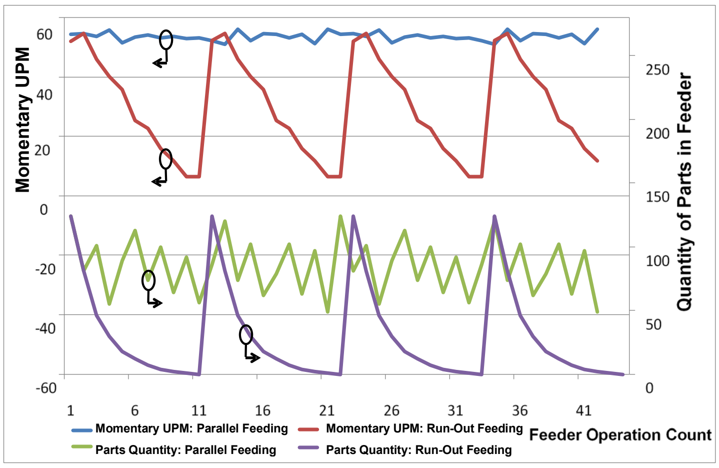
UPM is zero (= 0) during Vision & Feeder operation because Vision & Feeder operation and Robot operation are repeated. The UPM while the robot is operating is a value larger than the average UPM indicated in section *6.2.2*. Note that the average UPM is the hourly average of the momentary UPM (= 0) while the Vision & Feeder are operating and the momentary UPM while the robot is operating.

Also note that the length of the blue lines of Robot operation in the figure are not uniform. This is because the quantity of parts that can be picked up differs depending on the separation status of parts in the feeder and because repeating pick-up operation reduces the quantity of parts in the feeder and the parts that can be picked up.

It is necessary to maintain a uniform quantity of parts in the feeder in order to stably perform part feeding.

7.2.4 Relation Between Quantity of Parts in Feeder and Hopper Operation

For stable part feeding, the graph below shows the momentary UPM and quantity of parts in the feeder at each feeder operation when performing run-out feeding from the hopper and when performing parallel feeding of an optimal 180 parts with 90 parts loaded in the hopper.



In run-out feeding, parts are not loaded from the hopper until the parts run out, resulting in the momentary UPM gradually decreasing, and when there are no more parts, parts are then supplied from the hopper and the momentary UPM returns to the original value.

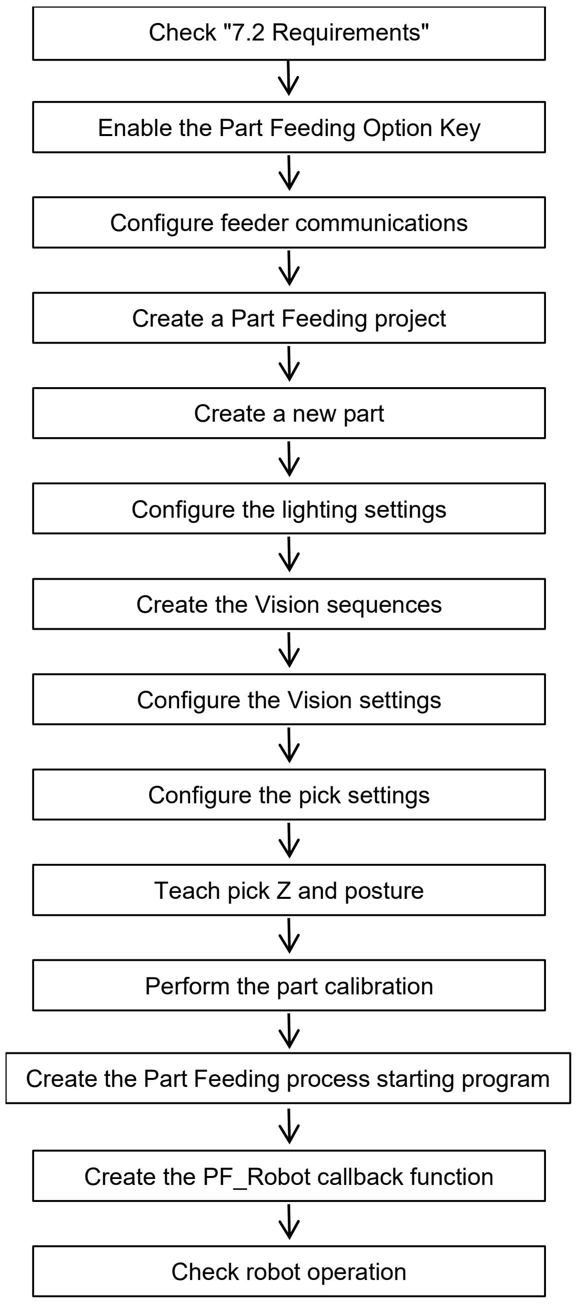
In parallel feeding, the hopper operates when the feeder operates two to four times, and this reduces variations in the momentary UPM as the quantity of parts in the feeder does not drop below the lower limit.

8. Let’s Use the Part Feeding Option

Let’s configure the system that picks and place parts, using the Part Feeding option.

8.1 Workflow

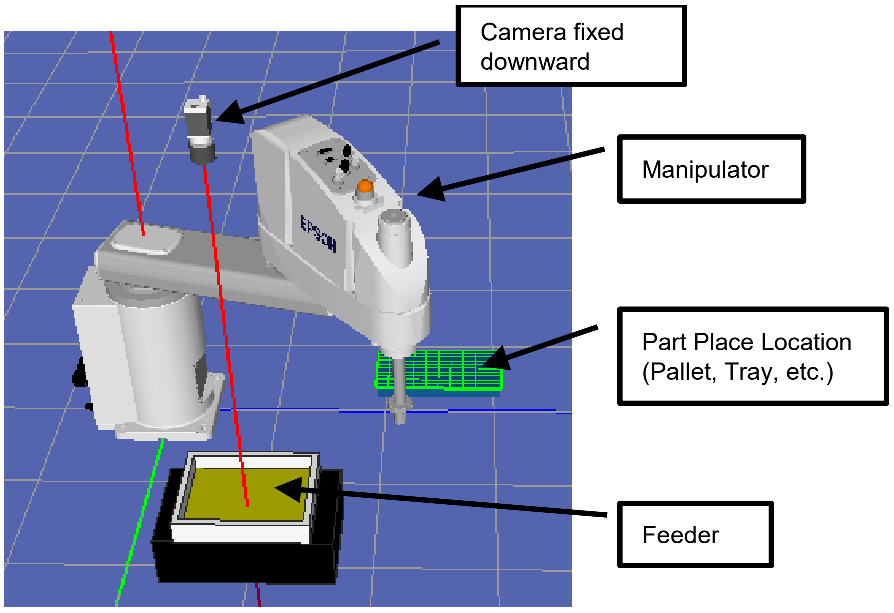
The workflow is as follows.


Check "7.2 Requirements"
Enable the Part Feeding Option Key
Configure feeder communications
Create a Part Feeding project
Create a new part
Configure the lighting settings
Create the Vision sequences
Configure the Vision settings
Configure the pick settings
Teach pick Z and posture
Perform the part calibration
Create the Part Feeding process starting program
Create the PF_Robot callback function
Check robot operation

8.2 Requirements

8.2.1 Device Configuration

- A SCARA robot is used as the manipulator.
  The required operation for 6-axis robots stays the same as that for SCARA robots.
  A hand suited for the parts is connected.
- A camera fixed downward is used.
- The feeder's backlight is used.
- A hopper is not used.



8.2.2 Connection and Adjustment

- EPSON RC+ is connected with the controller.
- The manipulator is connected with the controller.
- The feeder is connected with the controller.
- Vision Guide (PV or CV) is connected with the controller.
- The manipulator, camera, and the feeder are installed correctly.
- The adjustments of position, focus, and brightness have been completed.

8.2.3 Part

- The part ID is "1".
- The part is simply shaped and its surface features are the same on both sides.
  Parts are detected by the Vision's Blob object (detection using area values).

8.2.4 Settings

- The manipulator is registered in the system settings.
- The calibration for the Vision Guide has been completed.
- The tool setting for a robot has been completed.

8.2.5 Others

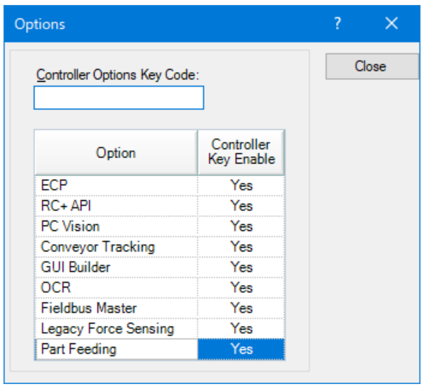
Handling of the errors described in the template code for the callback functions is implemented.

8.3 Enable the Part Feeding Options Key

To use the Part Feeding functions, the Option key for Part Feeding must be enabled. The Option key must be purchased from your Epson dealer.

(1) In EPSON RC+ 7.0, select Menu-[Setup], then select [Options] to open the Options dialog box.

(2) Copy and paste or write down the [Controller Options Key Code].



Option	Controller Key Enable
ECP	Yes
RC+ API	Yes
PC Vision	Yes
Conveyor Tracking	Yes
GUI Builder	Yes
OCR	Yes
Fieldbus Master	Yes
Legacy Force Sensing	Yes
Part Feeding	Yes

(3) Call your distributor to purchase the enable key code for the desired option.

(4) You will receive a code to enable the option from your distributor.

(5) Select the option to be enabled and select the <Enable> button.

(6) Enter the code you received from your distributor.

NOTE

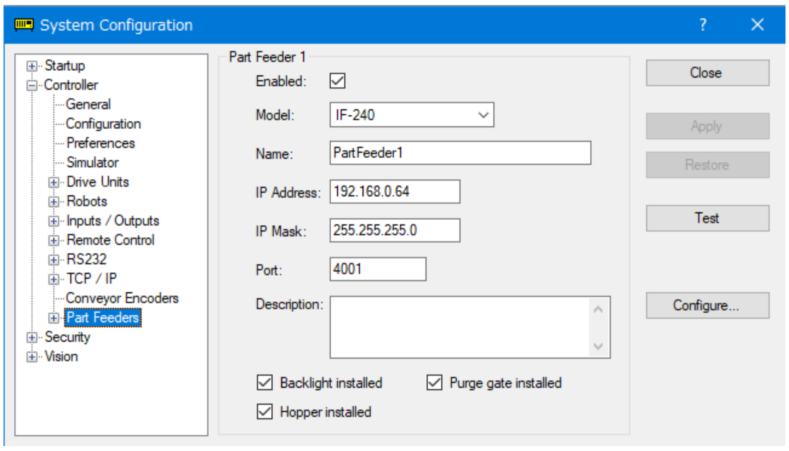
The key code is case sensitive.

(7) Click the <OK> button.

8.4 Configure Feeder Communications

(1) In EPSON RC+ 7.0, select Menu-[Setup] to open the [System Configuration] dialog box.

(2) In the tree, select [Controller]-[Part Feeders]-[Feeder 1].


System Configuration

Part Feeder 1

Enabled: ☑

Model: IF-240

Name: PartFeeder1

IP Address: 192.168.0.64

IP Mask: 255.255.255.0

Port: 4001

Description:

☑ Backlight installed ☑ Purge gate installed

☑ Hopper installed

(3) Change the settings for the following items.

Items	How to make settings
Enabled	Check the box.
Model	Select a feeder type which connected to the controller.
Name	Enter feeder name
IP Address	Enter 192.168.0.64
IP Mask	Enter 255.255.255.0
Backlight installed	Check the box.

(4) Click the <Apply> button when the settings for all the items have been completed.

TIP

When the IP address of the feeder is changed, refer to the following section.

Software 2.1.1 Part Feeding Page

TIP

Though 4 feeders can be configure to the T/VT series controller, the number of feeders that can be controlled at the same time is up to 2.

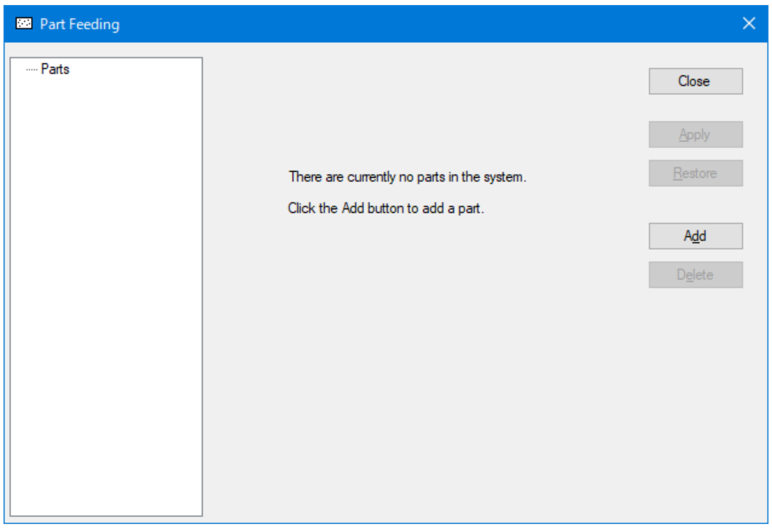
8.5 Create a Project for Part Feeding

In EPSON RC+ 7.0, select Menu-[Project]-[New] and create a new project.

Otherwise, open an existing project and make a copy of it.

8.6 Create a New Part

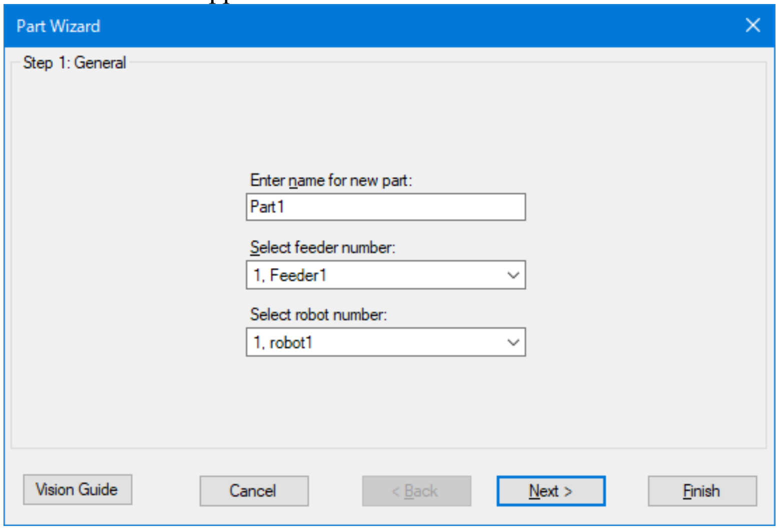
(1) In EPSON RC+ 7.0, Select Menu-[Tool] to open the [Part Feeding] dialog box.
Click the <Add> button.



There are currently no parts in the system.

Click the Add button to add a part.

(2) The Part Wizard appears.



Enter name for new part: Part1

Select feeder number: 1, Feeder1

Select robot number: 1, robot1

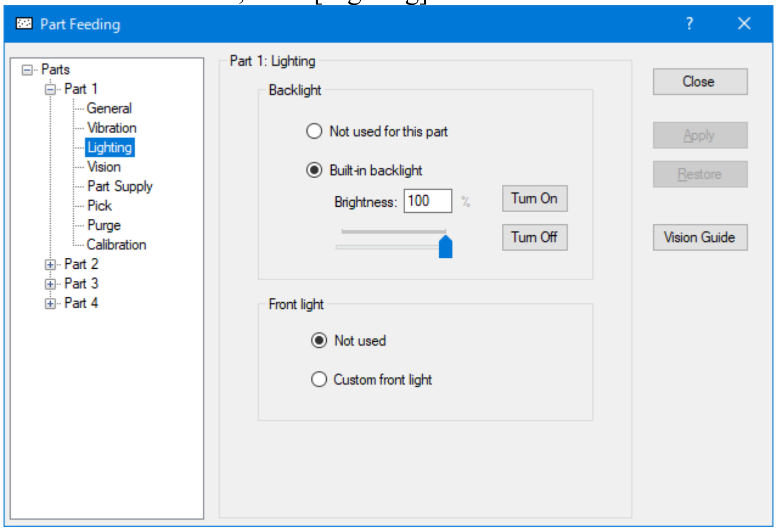
(3) Follow the steps of the Part Wizard to complete the part setting. For details, refer to the *Software 2.2 Part Wizard.*
The Part Wizard configures the basic part setup. For this tutorial, exit the wizard configure the settings.

TIP

When the first part is created in a project, a program file named PartFeeding.prg and an include file named PartFeeding.inc are added to the project automatically.

8.7 Configure the Lighting Settings

(1) In the tree on the left, click [Lighting].



Part 1: Lighting

Backlight

Not used for this part [✓ Built-in backlight]

Brightness: 100 %

Front light

[✓ Not used] Custom front light

(2) Click the <Turn On> button to turn on the backlight of the feeder.

8.8 Create the Vision Sequences

This section describes how to create two Vision sequences.

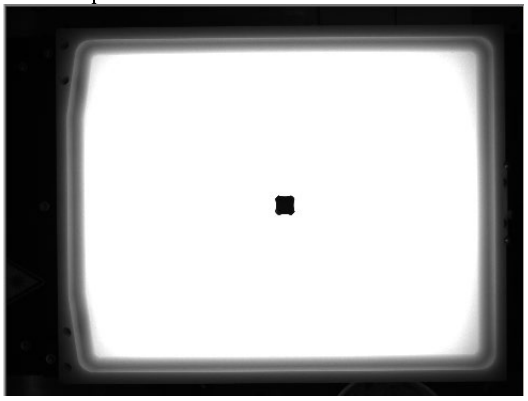
Be sure to create a Vision sequence for part detection and another Vision sequence for feeder calibration.

8.8.1 Creating a Vision sequence for Part Detection

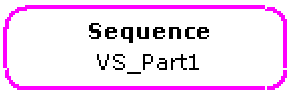
Create a Vision sequence for part detection.

(1) Click the <Vision Guide>button to display the Vision Guide dialog.

(2) Place a part on the feeder.



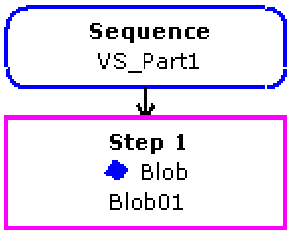
(3) Create a new Vision sequence. Set the property as follows.



Sequence
VS_Part1

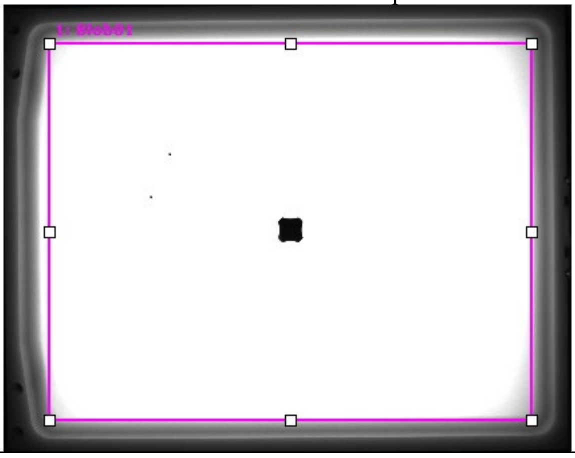
Properties	How to make settings
Name	Set an adequate name. (Example: VS_Part1)
Calibration	Make settings for Vision calibration.
ExposureTime	Make the settings, bearing the following in mind. - The part is clearly identifiable. - The brightness of the center of the platform and that of the surrounding area are almost the same.

(4) Add a Blob object. Make the settings for the properties as follows.



Sequence
VS_Part1

Step 1
Blob
Blob01

Properties	How to make settings
SearchWindow	Set the detection area to the entire platform. 
NumberToFind	Set to "All."
MaxArea	Set a value as 1.3 times as the part's Area value.
MinArea	Set a value as 0.7 times as the part's Area value.
ThresholdHigh	Make the setting so that the part will be certainly detected and the dark areas outside the platform will not be mistakenly detected.

TIP

If the part is not detected correctly, readjust the other properties and the properties of the Vision sequences.

(5) On completion of the settings, click the <Run> button and check that the part is identifiable correctly and the background is not mistakenly detected.

(6) Select Vision Guide menu-[File]-<Save> button to save the settings.

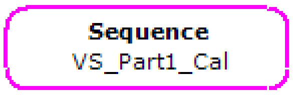
TIP

For details on creating for a Vision sequence for part detection, refer to the following section.

Software 6.3. Parts Detection Vision Sequence

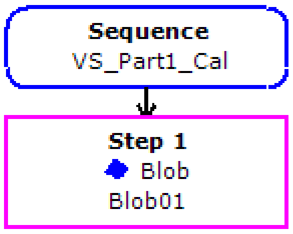
8.8.2 Create a Vision Sequence for Feeder Calibration

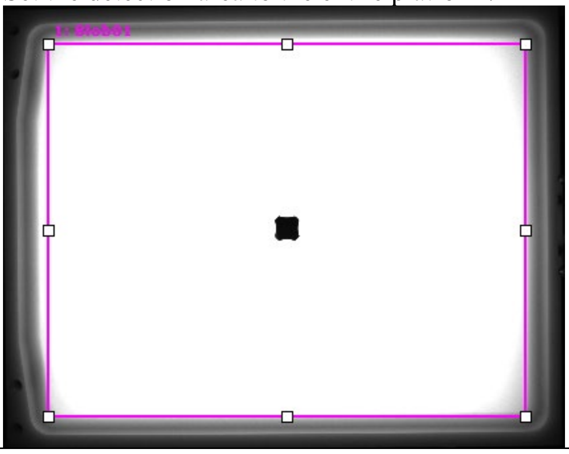
(1) To create a new Vision sequence, set the properties as follows.



Properties	How to make settings
Name	Set an adequate name. (Example: VS_Part1_Cal)
Calibration	Set this for Vision.
ExposureTime	Set this to identify the part clearly.

(2) Add a Blob object. Set the properties as follows.



Properties	How to make settings
SearchWindow	Set the detection area to the entire platform. 
MaxArea	Leave as the default value.
MinArea	Set to around 0.9 times that of the parts area. Use the following procedure to confirm the size of the parts area. 1. Turn on the feeder backlight 2. Place several parts on the platform so that they do not overlap 3. Run the vision sequence 4. The parts area is the average area for Blob results
NumberToFind	Set to "All."
ThresholdHigh	Make the setting so that the part will be certainly detected and the dark areas around the platform will not be mistakenly detected.

TIP

If the part is not detected correctly, readjust the properties and the properties of the Vision sequences.

(3) When you have finished the settings, click the <Run> button and check that the part is identifiable correctly and the background is not mistakenly detected.

(4) When you have confirmed the settings are correctly made, select Vision Guide menu - [File] - <Save> button to save the settings.

(5) Close the Vision Guide screen.

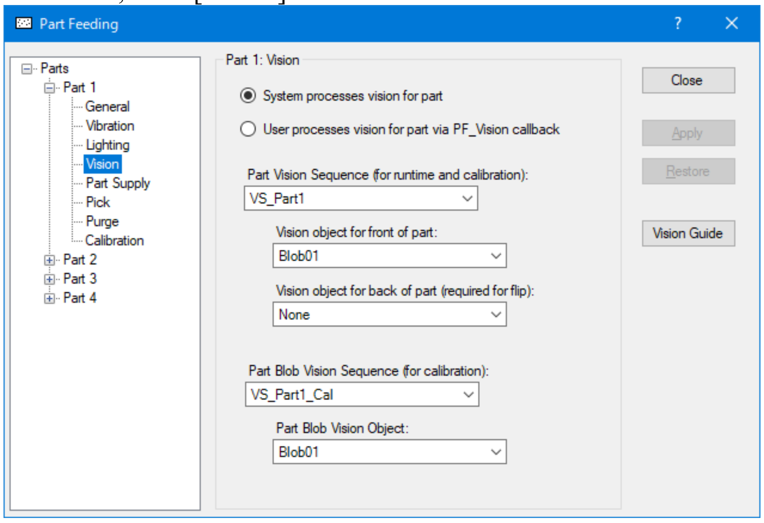
TIP

For details on creating for a Vision sequence for Feeder Calibration, refer to the following section.

Software 6.2. Feeder Calibration Vision Sequence

8.9 Configure the Vision Settings

(1) In the tree, click [Vision].



Part 1: Vision

● System processes vision for part

○ User processes vision for part via PF_Vision callback

Part Vision Sequence (for runtime and calibration): VS_Part1

Vision object for front of part: Blob01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): VS_Part1_Cal

Part Blob Vision Object: Blob01

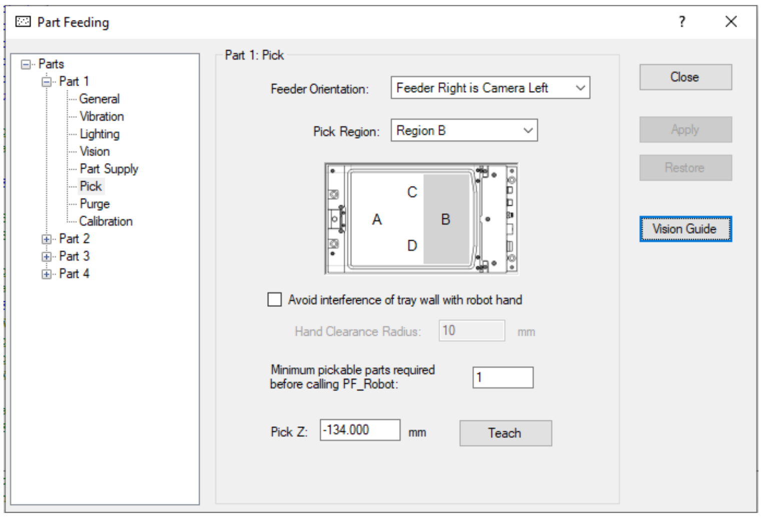
(2) Change the settings for the following items.

Items	How to make settings
Part Vision Sequence	Specify the Vision sequence created in the following section. *7.8.1 Creating a Vision sequence for part detection.*
Vision object for front of part	Specify the Blob object included in the Vision sequence created in the following section. *7.8.1 Creating a Vision sequence for part detection.*
Part Blob Vision Sequence	Specify the Vision sequence created in the following section. *7.8.2. Creating Vision sequence for feeder calibration.*
Part Blob Vision Object	Specify the Blob object included in the Vision sequence created in the following section. *7.8.2. Creating Vision sequence for feeder calibration.*

(3) Click the <Apply> button.

8.10 Configure the Pick Settings

(1) In the tree, click [Pick].



Feeder Orientation: Feeder Right is Camera Left

Pick Region: Region B

☐ Avoid interference of tray wall with robot hand

Hand Clearance Radius: 10 mm

Minimum pickable parts required before calling PF_Robot: 1

Pick Z: -134.000 mm

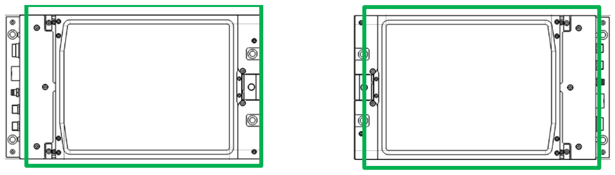
(2) Change the settings for the following items.

Items	How to make settings
Feeder Orientation	Select the orientation of the feeder installation viewed from the camera. It is important to set this correctly so the system knows the orientation of the feeder with respect to the camera.
Pick Region	Select a position closest from the placing position. Select "B" here. The selectable items vary depending on the feeder.

(3) Click the <Teach> button.

TIP

Feeder Orientation: Choose either direction.



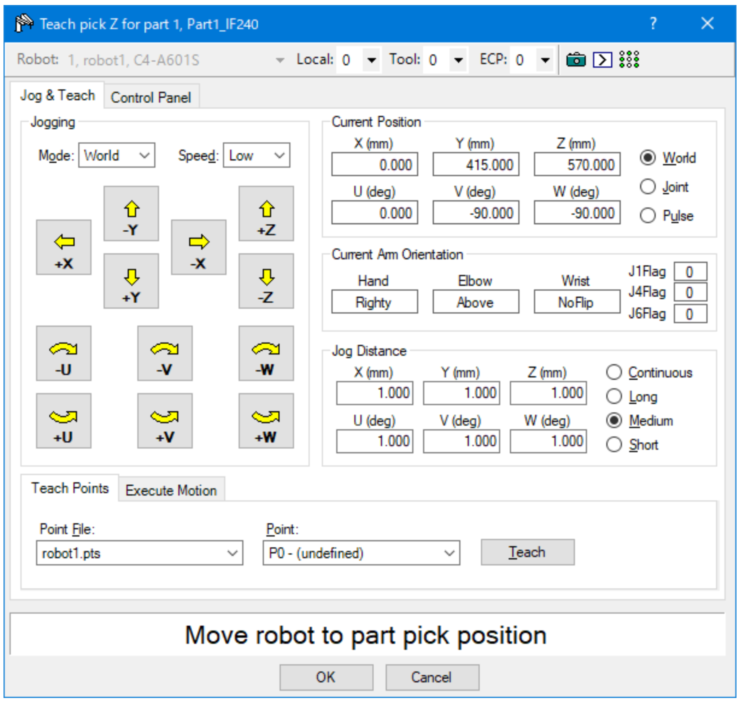
Select "Pick Region" closest from the placing position.

For the IF-240, the Pick Region can be selected from A to D.

For the IF-530, the Pick Region can be selected from A to B.

For the IF-80, the Pick Region "Anywhere" can only be selected.

8.11 Teach Pick Z and Posture


Teach pick Z for part 1, Part1_IF240

Robot: 1, robot1, C4-A601S Local: 0 Tool: 0 ECP: 0

Jog & Teach | Control Panel

Jogging

Mode: World Speed: Low

Current Position

X (mm)	Y (mm)	Z (mm)
0.000	415.000	570.000
U (deg)	V (deg)	W (deg)
0.000	-90.000	-90.000

[✓ World] Joint Pulse

Current Arm Orientation

Hand	Elbow	Wrist
Righty	Above	NoFlip

J1Flag 0
J4Flag 0
J6Flag 0

Jog Distance

X (mm)	Y (mm)	Z (mm)
1.000	1.000	1.000
U (deg)	V (deg)	W (deg)
1.000	1.000	1.000

Continuous Long [✓ Medium] Short

Teach Points | Execute Motion

Point File: robot1.pts

Point: P0 - (undefined)

Move robot to part pick position

(1) Place a part on the feeder.

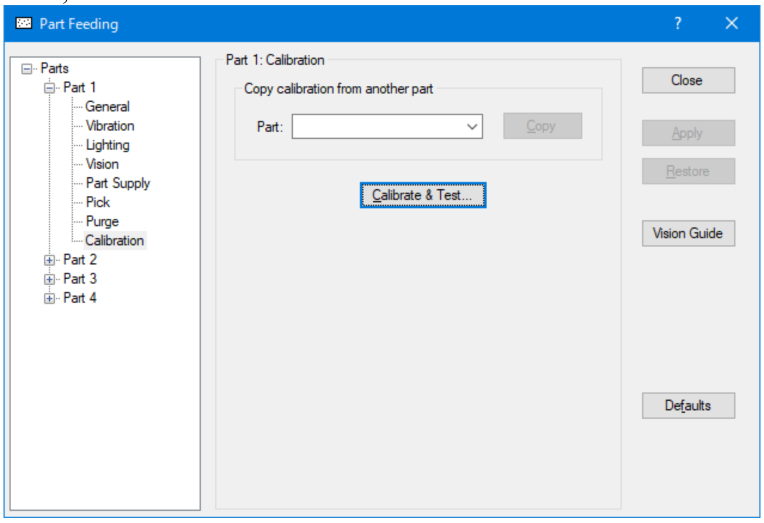
(2) Move the robot by the Jog operation, etc. and bring the hand into contact with the part on the feeder. In case of a hand with chuck mechanism, hold the part.

(3) Click the <OK> button.
The Z coordinate will be saved.

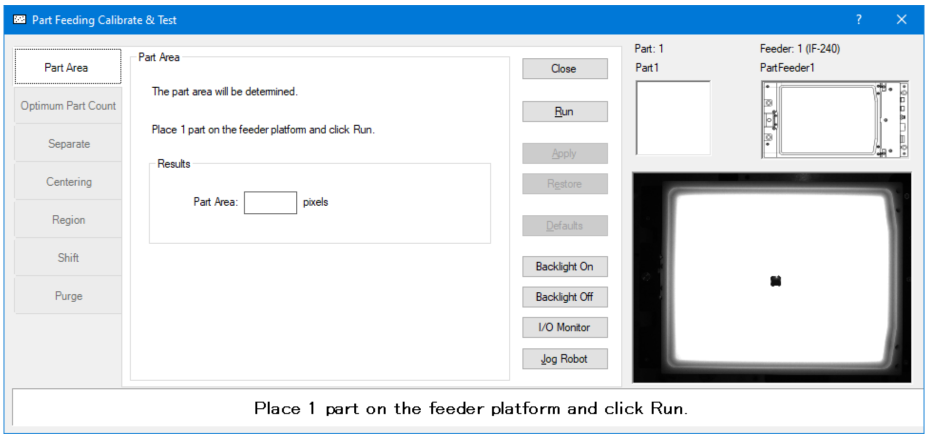
(4) In the [Pick] dialog box, click the <Apply> button.

8.12 Calibration & Test

(1) In the tree, click "Calibration".
Then, click the < Calibrate & Test > button.



(2) The [Calibrate & Test] wizard starts.
Select [Part Area] tab.



The part area will be determined.

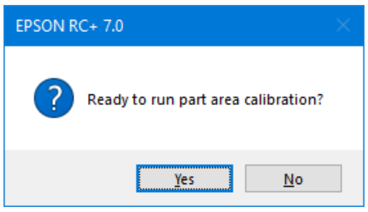
Place 1 part on the feeder platform and click Run.

Place 1 part on the feeder platform and click Run.

(3) Put one part in the center of the platform.

(4) Click < Run > button.

(5) Next message displayed.
Click <Yes> button.



Ready to run part area calibration?

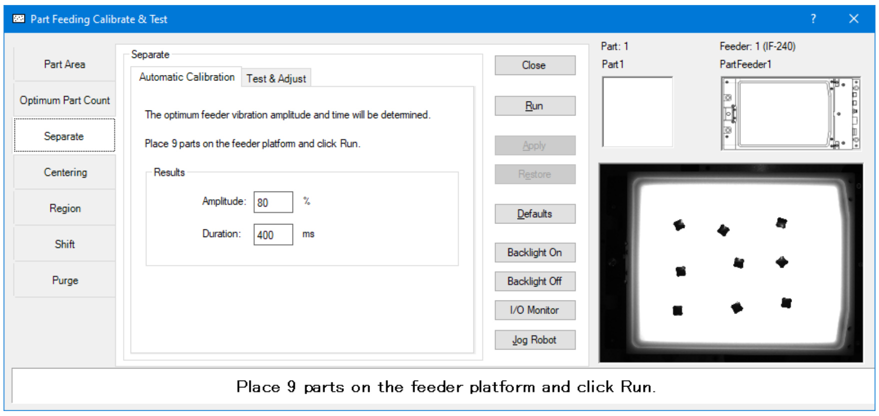
(6) Check the value displayed in the "Part Area"

(7) Click < Apply > button.

NOTE

Following both Calibration (8) [Separate] ([Flip & Separate] when flip is enabled) and (14) [Region] are optional. However, recommend running calibration.

(8) Select [Flip & Separation] tab.


Part Feeding Calibrate & Test

Separate

Automatic Calibration | Test & Adjust

The optimum feeder vibration amplitude and time will be determined.

Place 9 parts on the feeder platform and click Run.

Results

Amplitude: 80 %

Duration: 400 ms

Part: 1 Part1

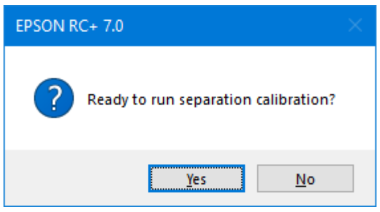
Feeder: 1 (IF-240) PartFeeder1

Place 9 parts on the feeder platform and click Run.

(9) Put the same number of parts deisplyed, on the platform.

(10) Click the < Run > button.

(11) Next message displayed.
Click <Yes> button.

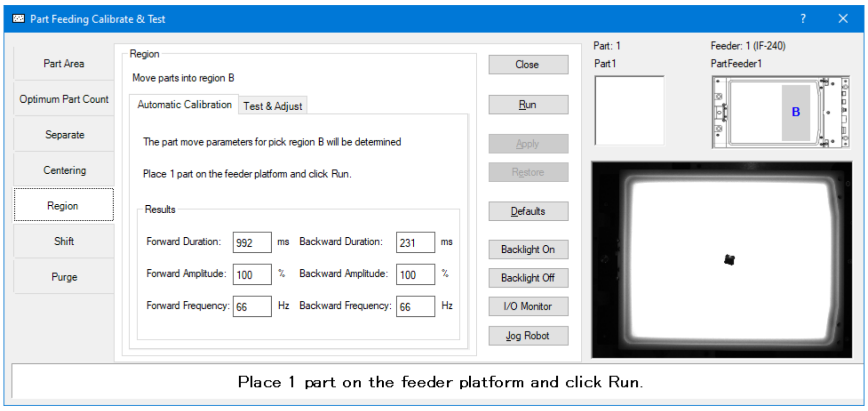

EPSON RC+ 7.0

Ready to run separation calibration?

(12) After several times of running feeder vibration and vision processing, check each value of [Results] has been updated.

(13) Click the < Apply > button.

(14) Select [Region] tab.


Part Feeding Calibrate & Test

Region

Move parts into region B

Automatic Calibration | Test & Adjust

The part move parameters for pick region B will be determined

Place 1 part on the feeder platform and click Run.

Results

Forward Duration:	992	ms	Backward Duration:	231	ms
Forward Amplitude:	100	%	Backward Amplitude:	100	%
Forward Frequency:	66	Hz	Backward Frequency:	66	Hz

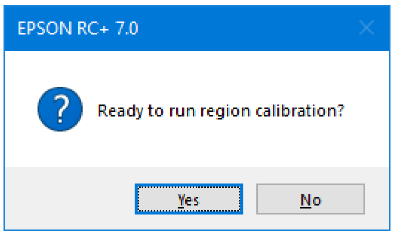
Part: 1 Part1

Feeder: 1 (IF-240) PartFeeder1

Place 1 part on the feeder platform and click Run.

(15) Change the number of input parts to one.

(16) Next message displayed.
Click the < Run > button.


EPSON RC+ 7.0

Ready to run region calibration?

(17) After several times of running feeder vibration and vision processing, check each value of [Results] has been updated.

(18) Click the < Apply > button.

8.13 Create the Part Feeding Process Starting Program

Write the program code to prepare the robot for pick & place of the part in the PartFeeding process starting program. The basic examples of the content to write are as follows.

1. If using a multi-robot system, change the current robot to the one for pick & place of parts.
   Use the Robot command.
2. Add statements to set the speed, acceleration, power mode, etc. for the robot.
   Use "Speed," "Accel," "Power," and so on.
   LimZ is set in consideration of Z coordinates to which teaching is done by *7.11 Teach pick Z and posture* and depth of platform (28mm), Z direction margin when platform's jumping over, height of Z direction of unit set up around feeder.
3. Turn the robot motor on.
   Use "Motor."
4. Move the robot to the position where imaging by Vision is possible
   (When using a camera fixed downward, move it to a position where the camera would not interfere with imaging by the vision system.)
   Use "Home," etc.
5. Add statements to initialize the customer's system devices, e.g. hopper, customized lightings.
6. To obtain a log, add the PF_InitLog statement.
7. Add the PF_Start statement which specifies the part ID of the part you want to use.
   Running PF_Start is executed in task 32 and immediately returns control to the caller.

The following is a sample program.
This program creates the following function in a program file main.prg
(The description for 5, 6 is omitted.)

```
Function test

  Robot 1
  Motor On
  Speed 100
  Accel 100, 100
  Power High
  LimZ -80.0
  Home

  PF_Start(1)

Fend
```

8.14 Create the PF_Robot Callback Function

Write the processing in which the robot starts picking and placing of the part in the PF_Robot callback function. The concrete example of the content to write is as follows. (Steps 1 through 7 are repeated in a loop.)

1. Obtain the coordinate of the part on the platform through the part coordinate que.
   Use "PF_QueGet."
2. Move the robot to the position where the part is placed.
   Use "Jump", and so on (in case of a SCARA robot).
3. Hold the part by turn on the adsorption function, and so on.
4. Move the robot to where to place the part.
   Use "Jump," and so on (in case of a SCARA robot).
5. Release the part by turning of the adsorption function, etc.
6. Delete one of the data among the part coordinate que.
   Use "PF_QueRemove."
7. Check if the stop request is issued. If it is issued, leave the loop.
   Use "PF_IsStopRequested."

The following is a sample PF Robot call back function.
As the handling of a loop and steps 1, 6, 7 are described in automatically generated PartFeeding.prg, this sample describes the handling of the rest of the steps.

The labels used in this program are as follows:

IO label: Chuck (adsorption of parts), UnVacumm (release of parts)
Point label: PlacePos (Place coordinate)

```
Function PF_Robot(partID As Integer) As Integer

   Do While PF_QueLen(partID) > 0

         ' Pick
         P0 = PF_QueGet(partID)
         Jump P0 ! Wait 0.1; Off UnVacumm !
         On Chuck
         Wait 0.1

         ' Place
         Jump PlacePos
         Off Chuck
         On UnVacumm
         Wait 0.1

         ' Deque
         PF_QueRemove partID

         ' Check Cycle stop
         If PF_IsStopRequested = True Then
              Exit Do
         EndIf

   Loop
   Off UnVacumm

Fend
```

8.15 Check Robot Operation

Start the main program to check the robot operation.

NOTE ☞ In the first test, operate the robot at low power and low speed to thoroughly examine if it operates as intended.

1. Build the project.
2. Start the Run window.
3. Run the test function.
4. Confirm that the robot picks and place the parts in the correct movements.

This completes the building of a system that picks and places the parts.

Software

1. Introduction

This section provides an overview of the EPSON RC+ 7.0 Part Feeding 7.0 software.

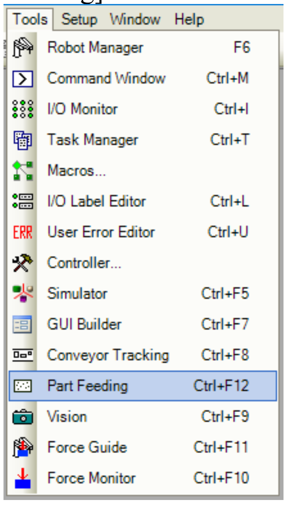
1.1 Part Feeding Software Configuration

The Part Feeding software mainly consists of the following three components.

- Part Feeding Window
- Part Feeding SPEL+ Commands
- Part Feeding Callback Functions

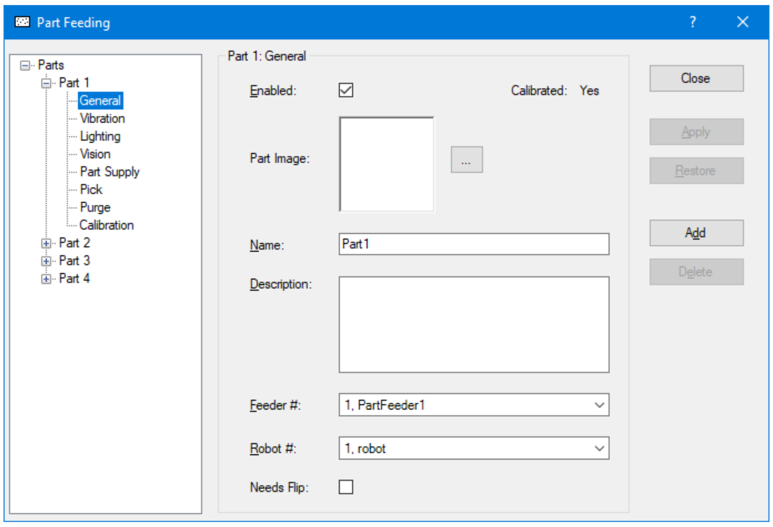
1.1.1 Part Feeding Window

The Part Feeding dialog is opened by selecting EPSON RC+ 7.0-Menu-[Tools]-[Part Feeding].



Here you can perform the following actions.

1. Configure part parameters.



Part 1: General

Enabled: ☑ Calibrated: Yes

Part Image: ...

Name: Part1

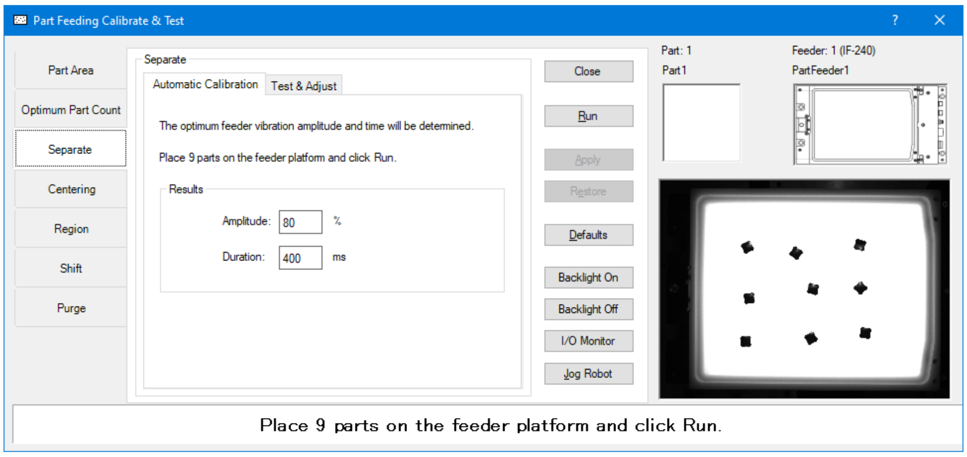
Description:

Feeder #: 1, PartFeeder1

Robot #: 1, robot

Needs Flip: ☐

2. Calibrate the feeder and manual adjustment/test.
   Anyone can easily calibrate by following the on-screen instructions.
   Manual adjustment / test allows fine adjustment of feeder parameters and can be easily tested with the touch of a button.



The optimum feeder vibration amplitude and time will be determined.

Place 9 parts on the feeder platform and click Run.

Results

Amplitude: 80 %

Duration: 400 ms

Place 9 parts on the feeder platform and click Run.

NOTE

The Part Feeding window will not appear if EPSON RC+ is not connected to the Controller. An error will occur when attempting to open the Part Feeding window using a virtual controller or when offline.

1.1.2 Part Feeding SPEL+ Commands

Part Feeding SPEL commands are SPEL commands provided to allow users to execute and control the Part Feeding option from a user-prepared program. The following are some typical examples of such commands.

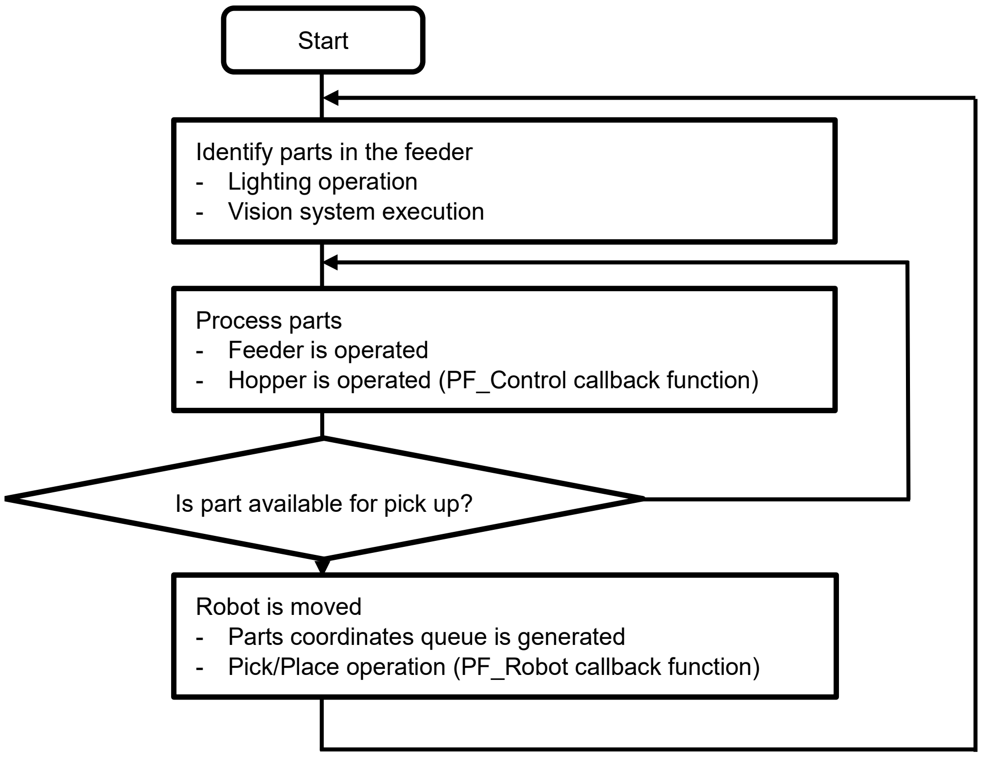
Command	Description/Application
PF_Start	Starts the Part Feeding process
PF_Stop	Issues a Part Feeding process stop request
PF_Abort	Forces the Part Feeding process to stop
PF_QueGet function	Returns coordinates data registered to the parts coordinates queue (list of parts coordinates on the feeder)
PF_InitLog	Specify the output destination path for log files
PF_Center	Run the feeder centering operation
PF_Flip	Run the feeder flip operation
PF_Shift	Run the feeder shift operation

1.1.3 Part Feeding Process

The Part Feeding process is a process of operations incorporated into the Part Feeding system to automatically perform vision and feeder control.

The Part Feeding process begins with the PF_Start command. The process is stopped using a PF_Stop or other such command.

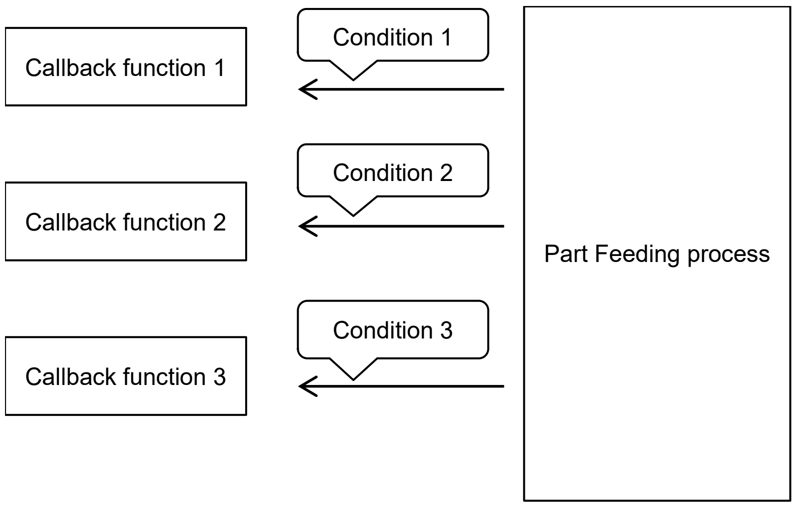
An overview of the Part Feeding process is provided below.


Start
Identify parts in the feeder
- Lighting operation
- Vision system execution
Process parts
- Feeder is operated
- Hopper is operated (PF_Control callback function)
Is part available for pick up?
Robot is moved
- Parts coordinates queue is generated
- Pick/Place operation (PF_Robot callback function)

1. Identify parts in the feeder
   Use the vision system to check the quantity and distribution of parts on the platform.
2. Process parts
   The feeder is controlled and parts are moved in order to make it easier for them to be grasped by the robot. When the parts are low in quantity or there are none remaining, the PF_Control callback function is invoked to supply parts from the hopper.
3. Robot is moved
   A parts coordinates queue (list of coordinates of parts in the feeder) is generated.
   The PF_Robot callback function (see the next section) is invoked to perform pick/place of parts.

1.1.4 Part Feeding Callback Functions

Callback functions are SPEL+ functions called back from the Part Feeding process when certain conditions are met. Callback functions are automatically generated within a project when parts are added. Callback functions are to be modified by the user based on the system in use.



For example, let’s assume the user sets a robot command to pick and place parts to the PF_Robot callback function. If the robot can pick up parts scattered randomly across the feeder, the Part Feeding process will then call the PF_Robot callback function. Therefore, it is important to note that callback functions are not called using user-prepared functions. Rather, they are automatically called up from the Part Feeding process.

The following is a list of callback functions with a description of each.

Function name	Description
PF_Robot	Picks up and places parts
PF_Control	Controls user device (hopper, user lighting)
PF_Status	Processes errors
PF_MobileCam	Moves robot while using the mobile camera
PF_Vision	Performs user-defined vision processing
PF_Feeder	Customer's own feeder operation
PF_CycleStop	Performs processing when a stop command is issued

The following lists the conditions upon which callback functions are called.

Condition	Function name
Able to pick up parts	PF_Robot
Parts no longer found	PF_Status
Turn on user lighting	PF_Control
Mobile camera is moved to image capture position	PF_MobileCam
An error occurs	PF_Status

For more details on callback functions, refer to *4. Part Feeding Callback Functions*.

1.2 Part Feeding Projects

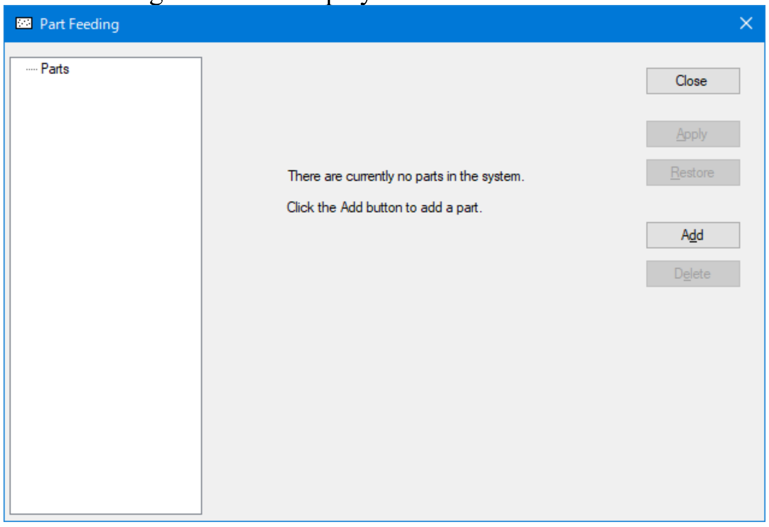
This section describes SPEL projects for Part Feeding in detail.

1.2.1 Applying the Part Feeding Option to a Project

To apply the Part Feeding option to a project, follow the procedure below.

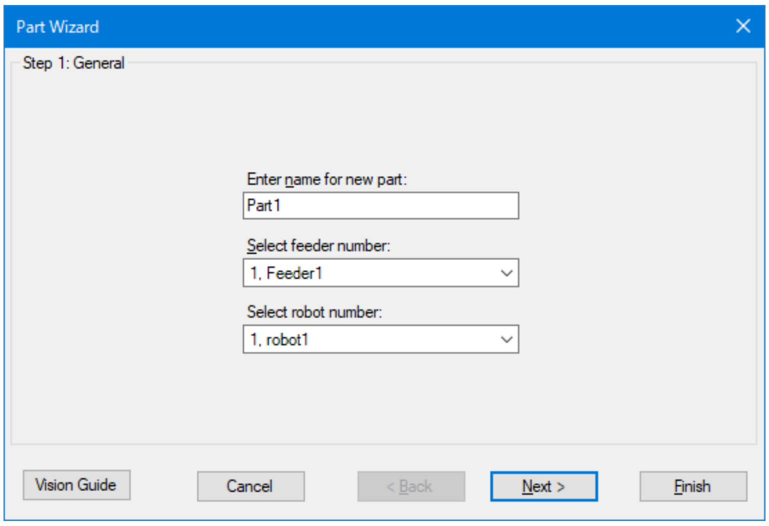
(1) Open the existing project you wish to apply the option to.
Or, create a new project.

(2) Open EPSON RC+ 7.0-Menu-[Tools]-[Part Feeding].
The following window is displayed.



There are currently no parts in the system.
Click the Add button to add a part.

(3) Click the <Add> button.
Run the Part Wizard.



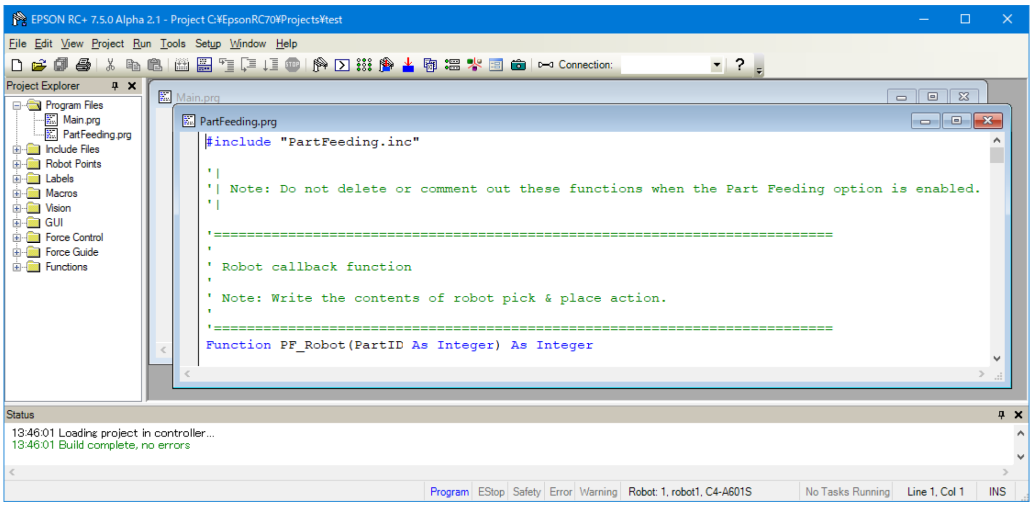
Enter name for new part: Part1
Select feeder number: 1, Feeder1
Select robot number: 1, robot1

(4) Follow the steps of the Part Wizard to complete the part setting. For details, refer to the *Software 2.2 Part Wizard.*
This will create one part.
When this happens, a Part Feeding option-specific program file will be added to the project. Refer to *1.2.2 Creating a Project* for more details.

1.2.2 Creating a Project

This section describes the components added when enabling the Part Feeding option.

The following program file will be added when adding new parts on the Part Feeding window.


EPSON RC+ 7.5.0 Alpha 2.1 - Project C:¥EpsonRC70¥Projects¥test
File Edit View Project Run Tools Setup Window Help
Connection:
Project Explorer
Program Files
Main.prg
PartFeeding.prg
Include Files
Robot Points
Labels
Macros
Vision
GUI
Force Control
Force Guide
Functions
Main.prg
PartFeeding.prg
Status
Program EStop Safety Error Warning Robot: 1, robot1, C4-A601S No Tasks Running Line 1, Col 1 INS

```
#include "PartFeeding.inc"

'|
'| Note: Do not delete or comment out these functions when the Part Feeding option is enabled.
'|

'==========================================================================
'
' Robot callback function
'
' Note: Write the contents of robot pick & place action.
'
'==========================================================================
Function PF_Robot(PartID As Integer) As Integer
```

```
13:46:01 Loading project in controller...
13:46:01 Build complete, no errors
```

PartFeeding.prg
This file includes a template code for callback functions.

PartFeeding.inc
This file includes constants used when programming with the Part Feeding option.

NOTE

Do not exclude or delete these files from the project. Doing so will cause a build error.

NOTE

With the following language settings, the comments in the program will be in the same language as the language setting.

English

Japanese

German

Spanish

For all other language settings, the comments will be in English.

1.2.3 Configuration Files

The following files will be added as project configuration files.

PF file
A file that includes part setting details. This file is named [project name].pf.

NOTE

Do not directly edit this file in a text editor. Doing so will render the file unreadable, and cause errors to occur.

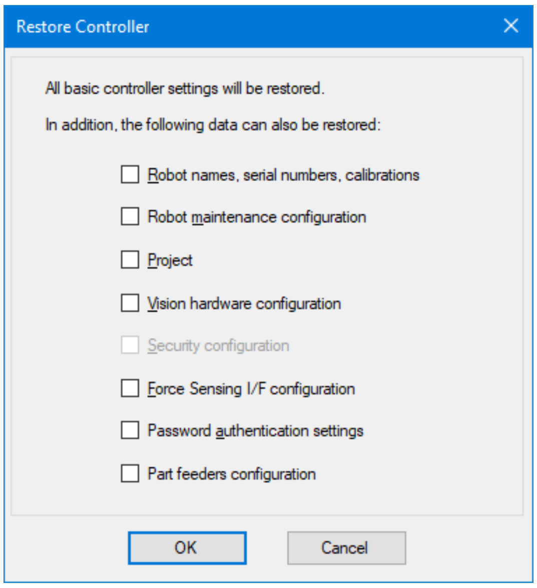
1.2.4 Importing Files

PF files can be imported. Use this to copy parts information to other projects.
Refer to *2.5.1 [Import] (File Menu)* for more details.

1.2.5 Backing up/Restoring the Controller

To back up Controller settings, navigate to EPSON RC+ 7.0-Menu-[Tools]-[Controller] and select [Backup Controller]. This will also back up Part Feeding option data.

To restore Controller settings from a backup, navigate to EPSON RC+ 7.0-Menu-[Tools]-[Controller] and select [Restore Controller]. At this time, Part Feeding option data is also restored when put check in "Part feeders configuration" in following wizard.


Restore Controller

All basic controller settings will be restored.

In addition, the following data can also be restored:

- ☐ Robot names, serial numbers, calibrations
- ☐ Robot maintenance configuration
- ☐ Project
- ☐ Vision hardware configuration
- ☐ Security configuration
- ☐ Force Sensing I/F configuration
- ☐ Password authentication settings
- ☐ Part feeders configuration

1.3 SPEL Programming

1.3.1 Programming Overview

This section briefly describes the overall Parts Feeding programming code. Similar information will be presented in more detail in forthcoming sections of the manual.

For most applications, Parts Feeding is handled by the system automatically. The basic SPEL code for Parts Feeding is created for you. The code consists of SPEL+ functions called "callbacks".

Callback functions are user functions called by the system at runtime and are used to let your program know the following:

PF_Robot function	:	When parts are available to be picked from the feeder
PF_Status function	:	When status has changed
PF_Vision function	:	When user vision processing is required
PF_Feeder function	:	When doing unique feeder operations
PF_Control function	:	When something needs to be controlled, such as hopper, front light, etc.
PF_MobileCam function	:	When using a mobile camera and the robot needs to move the camera above the feeder to search for parts
PF_CycleStop function	:	When the Part Feeding operation is stopped

Some modification to the callbacks will be required to handle the specific requirements of your application.

Most applications will only require use of the PF_Robot and PF_Status callbacks.

Here is a brief overview of each of these callback functions:

PF_Robot callback Function

PF_Robot will be automatically executed when the feeding / vision operation has been completed and parts are available to be picked. In general, this is the location in code where you will add instructions for the pick and place operation. A simple Parts Feeding application only requires adding a few lines of code to handle the robot's pick and place motion and gripper (mechanism grasp parts) actuation. When the PF_Robot callback ends, the PF_Status callback will start.

PF_Status callback Function

PF_Status will be automatically executed after each callback function completes. PF_Status tells the system how to proceed based upon the callback's return value or PF_Status will notify the operator that an error condition has occurred. The error could be a system level error (like a robot over-torque error) or a Parts Feeding error (like the quantity of parts supplied by the hopper are too little or too much). PF_Status gives you the ability to decide how to handle error conditions. A simple Parts Feeding application does not require modification to the PF_Status code.

For more complex applications, the user may need to handle the vision processing themselves. In this case you will need to use the PF_Vision callback. Here is a brief overview of the PF_Vision callback.

PF_Vision callback Function

When the customer performs the vision processing of the parts feeder instead of the vision processing prepared by the system, write the processing required for the PF_Vision callback function. You will need to execute VRun to acquire an image from the camera and process the sequence that finds the parts, get the vision RobotXYU results, and add then add the robot coordinate data into the Part coordinates queue. A simple Parts Feeding application that processes vision by the system will not require you to modify the PF_Vision callback.

In rare situations, you may need to use your own lighting rather than the built-in feeder backlight. The PF_Control callback is required to control your own lighting. There might also be situations where you want to use your own hopper. Whether you are using Epson's hopper or your own, you will need to control the hopper from the PF_Control callback. Here is a brief overview of the PF_Control callback.

PF_Control callback Function

PF_Control will be automatically executed if the system needs to turn on the hopper or if the system needs to turn on/off user supplied lighting. When using the Epson supplied hopper, you will need to uncomment the PF_OutputONOFF statements in the PF_Control callback.

PF_MobileCam callback Function

PF_MobileCam will be automatically executed if the camera is mobile mounted onto the robot rather than fixed mounted downward above the feeder. PF_MobileCam requests the robot to move the camera over the feeder so that the system can process images. PF_MobileCam also notifies the robot that it is ok to move away from the feeder when parts have been found and added to the Parts Feeding queue. A simple Parts Feeding application that uses a Fixed Downward camera does not require you to modify the PF_MobileCam callback.

PF_CycleStop callback Function

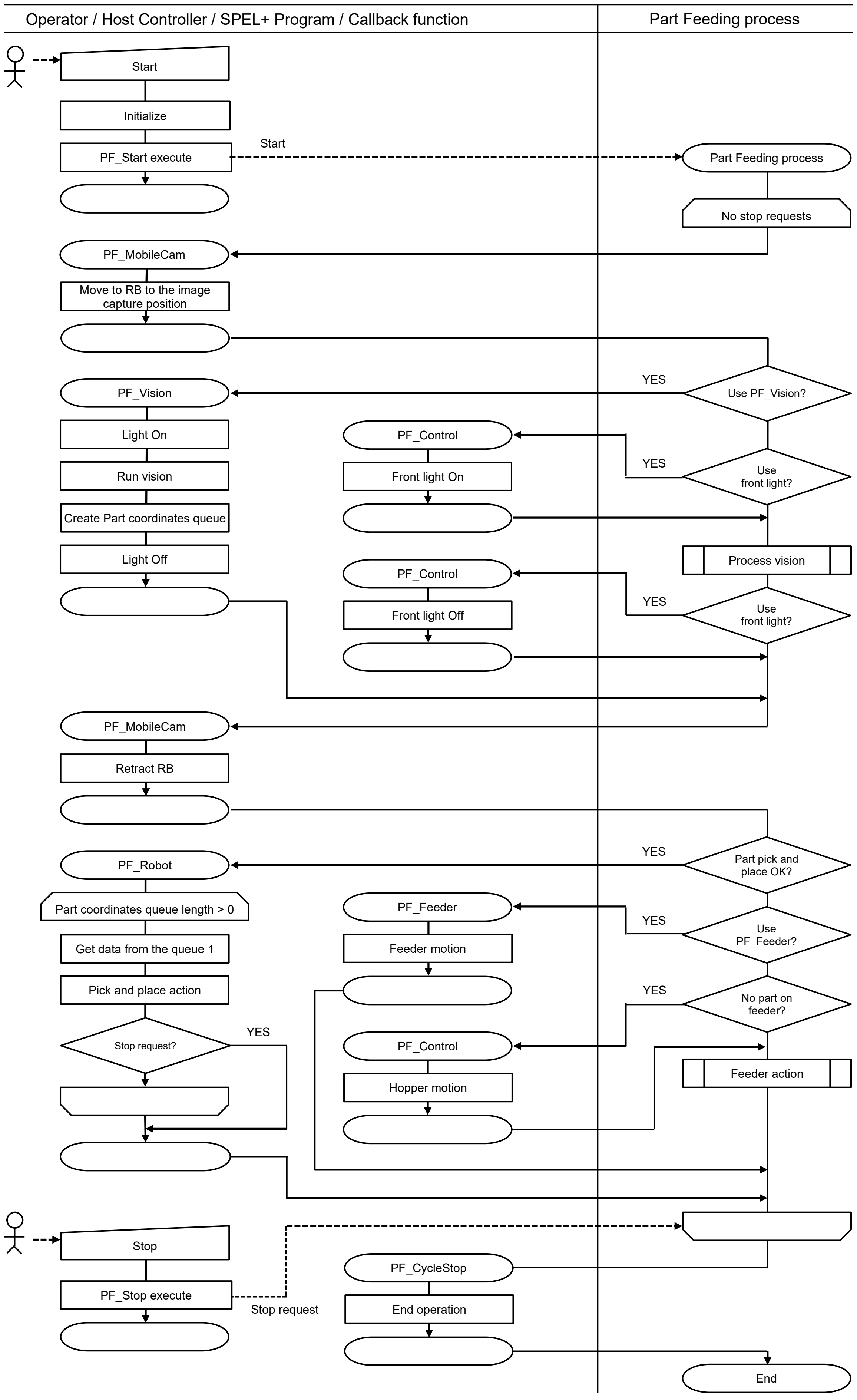
PF_CycleStop will be automatically started if the PF_Stop statement is executed. This callback allows you to finalize any operations upon termination (like turning on/off outputs or moving the robot to a safe location). A simple Parts Feeding application does not require modification to the PF_CycleStop code.

PF_ Feeder callback Function

When your device or part cannot be processed well by automatic feeder control, you can write the feeder operation in SPEL code using the PF_Flip command, PF_Shift command, etc. in the PF_Feeder callback function. In general, there is no need to write a PF_Feeder callback function.

All Part Feeding callback functions require you to return a value. To return a value from a function, you assign a value to the function's name (for example, PF_Robot = PF_CALLBACK_SUCCESS where PF_CALLBACK_SUCCESS is a predefined constant which has a value of 0). Normally the return value will indicate that the operation has completed successfully (constant PF_CALLBACK_SUCCESS).
PF_Status tells the system how to proceed after any of the callback functions have completed or after an error has occurred. The system needs to know what you want it to do – continue, exit, restart etc…. For example, you may want to set the return value to constant PF_EXIT to terminate the Parts Feeding operation after an error. For a simple Parts Feeding application, you can use the callback return values that are automatically generated in the code for you.
Refer to *Software 4. Part Feeding Callback Functions* for a detailed explanation of each of the callbacks.

The next page is an overview of the pick and place program.


Operator / Host Controller / SPEL+ Program / Callback function
Part Feeding process
Start
Initialize
PF_Start execute
Start
Part Feeding process
No stop requests
PF_MobileCam
Move to RB to the image capture position
YES
Use PF_Vision?
PF_Vision
Light On
Run vision
Create Part coordinates queue
Light Off
PF_Control
Front light On
YES
Use front light?
Process vision
PF_Control
Front light Off
YES
Use front light?
PF_MobileCam
Retract RB
YES
Part pick and place OK?
PF_Robot
Part coordinates queue length > 0
Get data from the queue 1
Pick and place action
Stop request?
YES
PF_Feeder
Feeder motion
YES
Use PF_Feeder?
YES
No part on feeder?
PF_Control
Hopper motion
Feeder action
Stop
PF_Stop execute
Stop request
PF_CycleStop
End operation
End

1.3.2 Starting the Part Feeding Process

Requirements for starting the Part Feeding process are as described below.

1. Move the robot to a position where the vision system can capture images.
   (For fixed downward-facing cameras, move the robot to where it will not interfere with imaging)
2. Run the PF_Start command to start the Part Feeding process.

The following is an example program.

```
Function StartPickPlace()

    Home
    PF_InitLog 1, "C:\log.csv", True
    PF_Start 1

End function
```

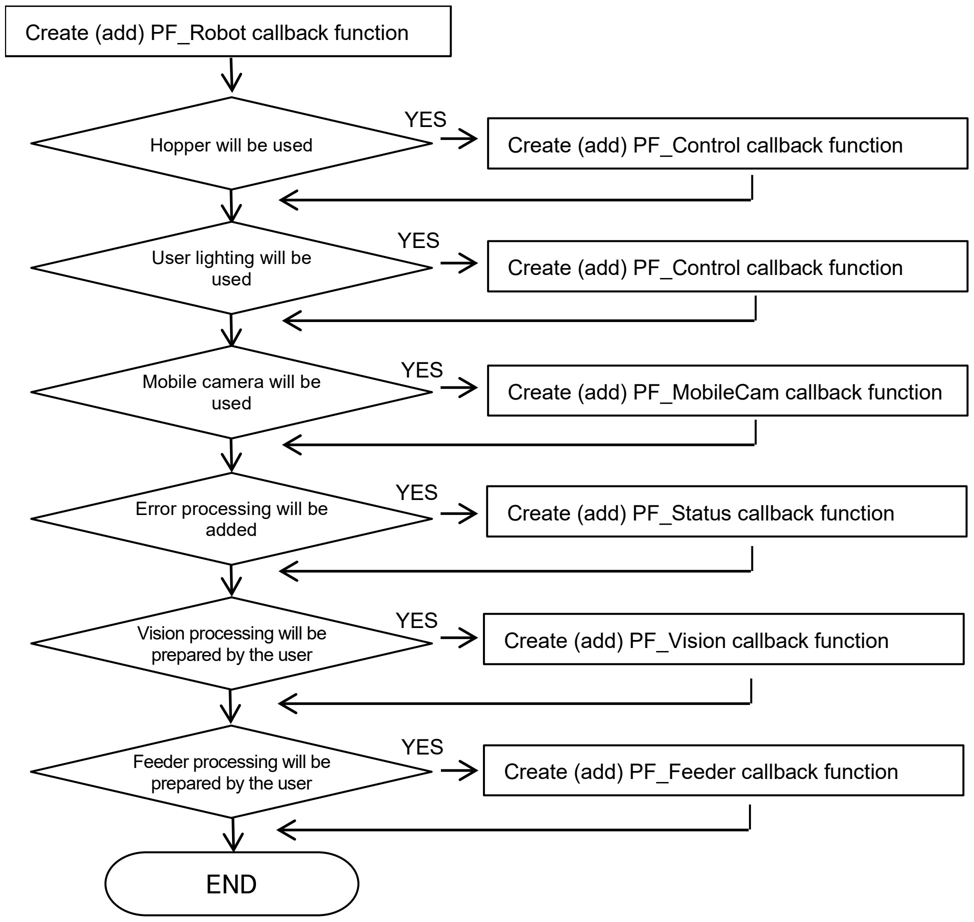
NOTE ☞

The following processes are written to be performed when starting up the system.

- Speed, acceleration, power mode and other settings relating to the robot used to pick up and place parts
- Hopper settings, user lighting settings, end effector attitude initialization, etc.

1.3.3 Pick and Place Processing

The following describes the programming process used for pick and place processing.


Create (add) PF_Robot callback function
Hopper will be used
YES
Create (add) PF_Control callback function
User lighting will be used
YES
Create (add) PF_Control callback function
Mobile camera will be used
YES
Create (add) PF_MobileCam callback function
Error processing will be added
YES
Create (add) PF_Status callback function
Vision processing will be prepared by the user
YES
Create (add) PF_Vision callback function
Feeder processing will be prepared by the user
YES
Create (add) PF_Feeder callback function
END

A PF_Robot callback function must be created (added).
Other callback functions are used based on user system specification requirements.

1. PF_Robot callback function

   Write a command for the robot to pick and place parts to the PF_Robot callback function. A specific example is provided below. (Repeat steps 1 through 7 on a loop)

   (1) Retrieve the coordinates of parts on the platform from the parts coordinates queue.
   Use PF_QueGet.

   (2) Move the robot to the part position.
   Use Jump or similar commands. (For SCARA robots)

   (3) Turn suction on and grip the part.

   (4) Move the robot to the position where the part will be placed.
   Use Jump or similar commands. (For SCARA robots)

   (5) Turn suction off, or release the part using some other method.

   (6) Delete one data entry in the parts coordinates queue.
   Use PF_QueRemove.

(7) Check whether a stop command has been issued. Discontinue the loop if issued. Use PF_IsStopRequested.

For more programming examples, refer to the following examples.

4. Part Feeding Callback Functions PF_Robot and *9. Application Programming Examples*

2. Supply parts to the feeder (PF_Control) *Optional

Program commands to use a hopper to supply parts to the feeder. Use the PF_Control callback function (PartFeeding.prg). Program details are provided below.

(1) Write the hopper command (supply parts) when no parts are on the platform. When doing so, the Control parameter for the PF_Control callback function is PF_CONTROL_SUPPLY_FIRST.

(2) Write the hopper command (supply parts) to add parts to the platform. When doing so, the Control parameter for the PF_Control callback function is PF_CONTROL_SUPPLY.

For more programming examples, refer to *4. Part Feeding Callback Functions PF_Robot*.

3. Control a user lighting (PF_Control) *Optional

Program commands to use a user lighting. Use the PF_Control callback function (PartFeeding.prg). Program details are provided below.

(1) Write a command to turn on the user lighting. When doing so, the Control parameter for the PF_Control callback function is PF_CONTROL_LIGHT_ON.

(2) Write a command to turn off the user lighting. When doing so, the Control parameter for the PF_Control callback function is PF_CONTROL_LIGHT_OFF.

For more programming examples, refer to *4. Part Feeding Callback Functions PF_Robot*.

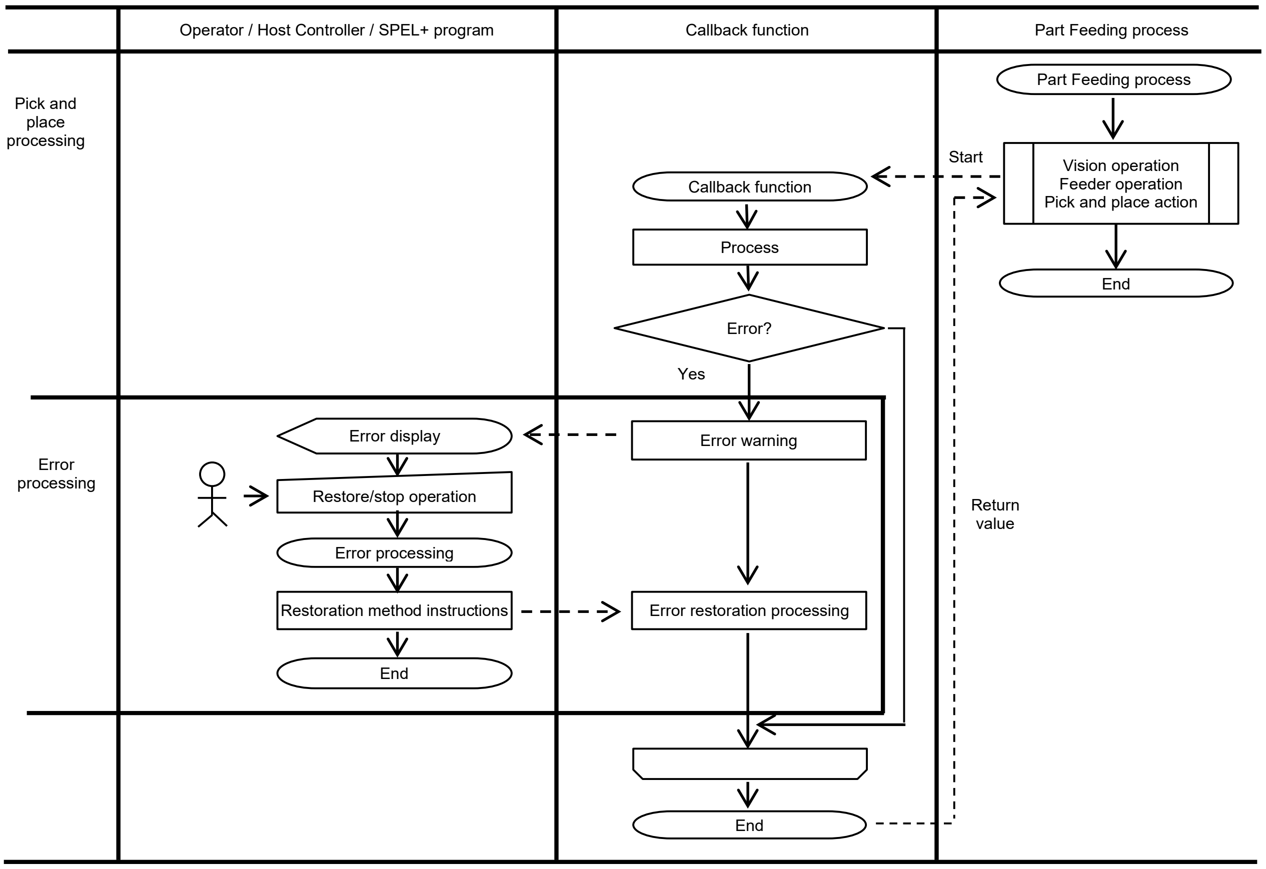
NOTE

Write both commands to turn the user lighting on and off.
When only the turn on command is added, the vision system may have difficulty recognizing parts, resulting in errors.

1.3.4 Processing Errors

This section describes how to process errors that occur while using the Part Feeding option.

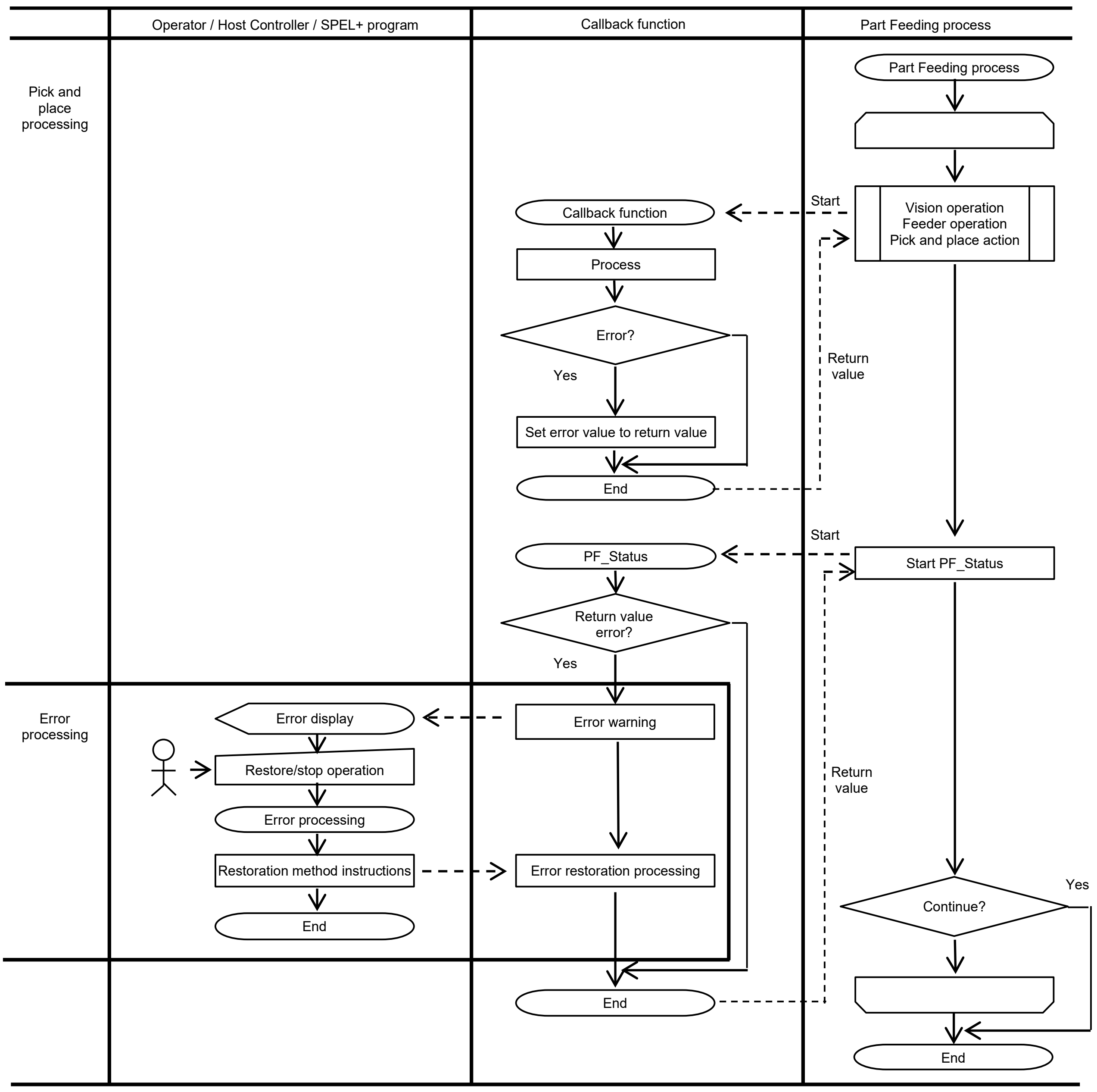
1. Processing errors occurring in the callback function



Detect and process errors occurring within the callback function inside the function itself. Use this error processing method if you wish to proceed without returning control to the Part Feeding process.

Example: A parts suction error occurs with the PF_Robot callback function, initiating a retry process

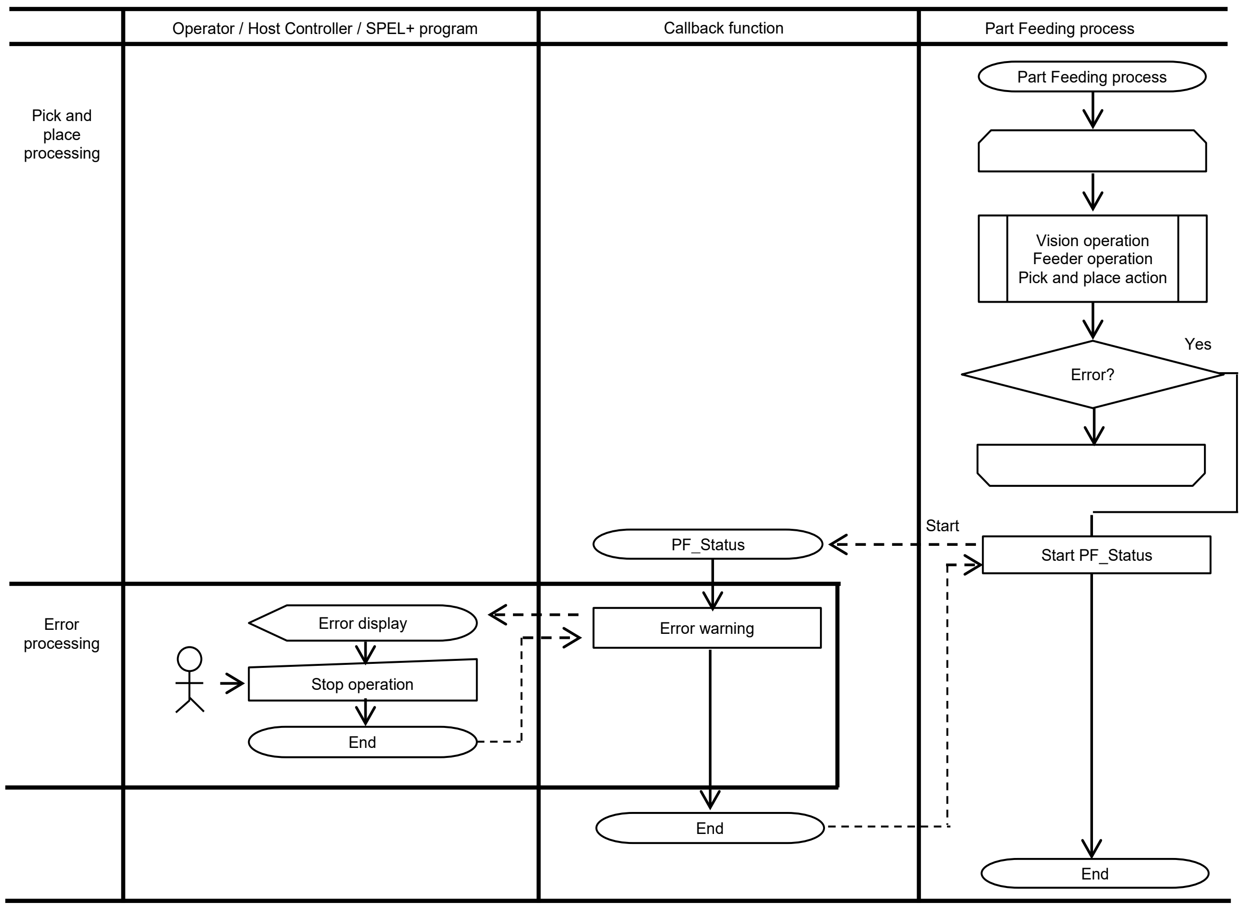
2. Processing errors occurring in the PF_Status callback function



Set a value other than PF_CALLBACK_SUCCESS to the return value (user error 8000-8999) if an error occurs in the callback function.
The Part Feeding process will set this value to the Status parameter of the PF_Status callback function at startup.
Use this error processing method to share error processes.

For more details, refer to *4. Part Feeding Callback Functions PF_Status*.

3. Processing errors occurring inside the Part Feeding process



An error may occur inside the Part Feeding process due to incomplete Part Feeding parameter settings (unspecified PF_Start parameter), incomplete vision system settings, or other deficiencies.
The Part Feeding process will set PF_STATUS_ERROR to the Status parameter at startup. The Part Feeding process will terminate once the PF_Status function ends.

For more details, refer to *4. Part Feeding Callback Functions PF_Status* and *9.6 Error Handling.*

1.3.5 End Processing

Use one of the following methods to terminate a Part Feeding process from a user-prepared program.

1. Execute a PF_Stop command
   Execute a PF_Stop command from a user-prepared program.
   The Part Feeding process will end the callback function currently being processed, and will terminate once the PF_CycleStop callback function has ended.
   For more details, refer to *3. Part Feeding SPEL+ Command Reference PF_Stop*.

2. Execute a PF_Abort command
   Execute a PF_Abort command from a user-prepared program.
   This immediately terminates the Part Feeding process. When this happens, the callback function currently being processed will end immediately.
   For more details, refer to *3. Part Feeding SPEL+ Command Reference PF_Abort*.

NOTE

Feeder and hopper vibration will stop when the Safeguard is opened or Pause is executed. The vibration will not resume after the Safeguard is closed and Continue is executed.

1.3.6 Functions used by Part Feeding process

The Part Feeding process occupies the following functions while running.
Do not set your program to use these functions while the Part Feeding process is running.

Feeder Number	Task	Timer	SyncLock
1	32	63	63
2	31	62	62
3	30	61	61
4	29	60	60
System (reserved)	28	-	59

These functions may be used in following conditions.

- When the Part Feeding processing is stopped
- When not using multiple feeders
  (e.g. when only one feeder is connected, the functions occupied by feeder number 2 or later can be used.)

NOTE

The Reserved System Task and SyncLock should never be used.

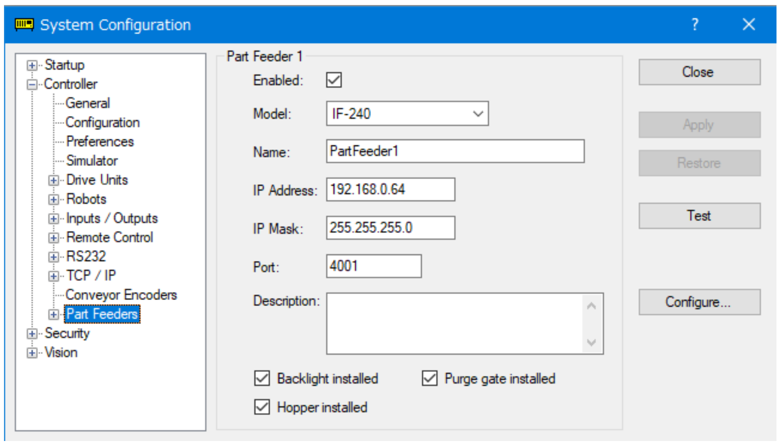
2. Part Feeding GUI

This section describes the EPSON RC+ 7.0 Part Feeding 7.0 GUI in detail.

2.1 System Configuration

Configure settings to connect the feeder to the Controller in EPSON RC+ 7.0-Menu-[Setup]-[System Configuration].

2.1.1 Part Feeding Page



Part Feeder 1
Enabled: ☑
Model: IF-240
Name: PartFeeder1
IP Address: 192.168.0.64
IP Mask: 255.255.255.0
Port: 4001
Description:
☑ Backlight installed ☑ Purge gate installed
☑ Hopper installed

Item	Description
Enabled	Select this check box to enable the feeder.
Model	Select the feeder model.
Name	Set a name of your choice. (Half-width alphanumeric characters and underscores only. Up to 32 characters in length)
IP Address	Enter the IP Address currently set to the feeder. Default IP Address 192.168.0.64
IP Mask	Enter the IP Subnet Mask currently set to the feeder. Default Subnet Mask 255.255.255.0
Port	Enter the Port number currently set to the feeder. Default Port number 4001
Backlight Installed	Select this check box if a backlight has been installed inside the feeder.
Hopper Installed	Select this check box to use a hopper for the feeder option.
Purge Gate Installed	If connecting the purge gate of the feeder option, check the check box. (IF-240, IF-380 and IF-530 only)
Description	Write a description (comments) for the feeder. This field is optional. (Up to 256 characters in length)

NOTE

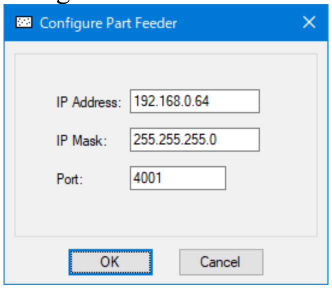
To change network settings for the feeder, click the <Configure> button described in the next section.

NOTE

An error will occur when connecting to the feeder if an IP Address, IP Mask or Port that is different from current feeder settings is entered. Note that the error will not occur when the <Apply> button is pressed.

NOTE

The “Purge gate installed” setting influences vibration parameters. If you are using a Purge Gate, check the “Purge gate installed” checkbox prior to adding new Parts in the Part Feeding dialog. If the checkbox is checked after adding new Parts, the feeder will not perform properly.

Button	Description
Close	Closes the dialog.
Apply	Applies changes.
Restore	Undoes changes.
Test	Tests the feeder's communication
Configure	Changes feeder network settings.  Configure Part Feeder IP Address: 192.168.0.64 IP Mask: 255.255.255.0 Port: 4001 IP Address Sets a new IP address for the feeder. IP Mask Sets a new subnet mask for the feeder. Port Sets a new port number for the feeder. Once settings are complete, click the <OK> button.

CAUTION

- Set a private IP address (see below) for the feeder before use.

Class A:	10.0.0.0	-	10.255.255.255
Class B:	172.16.0.0	-	172.31.255.255
Class C:	192.168.0.0	-	192.168.255.255

- When setting a global IP address for the feeder, be sure to understand risks such as unauthorized access before use.

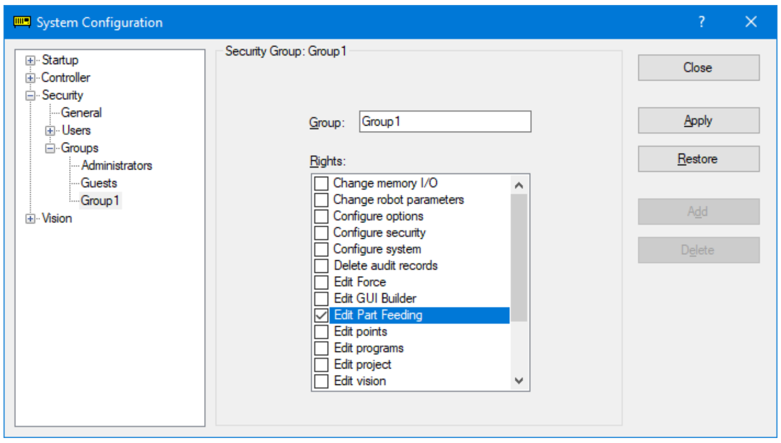
NOTE

The controller and the feeder can only communicate if they are in the same network segment.
You will not be able to connect to the feeder if network settings are configured for a different subnet that is used for the Controller. If this is the case, you will need to change the IP Address and IP Mask for the Controller.

These settings are found in EPSON RC+ 7.0-Menu-[Setup]-[System Configuration]-[Controller]-[Configuration].
Refer to the following document for more details.
EPSON RC+ User's Guide "4.3.3 Ethernet communications"

2.1.2 Security Page

Configure security settings to restrict users from viewing or editing Part Feeding option settings.



Security Group: Group1

Group: Group1

Rights:

- [ ] Change memory I/O
- [ ] Change robot parameters
- [ ] Configure options
- [ ] Configure security
- [ ] Configure system
- [ ] Delete audit records
- [ ] Edit Force
- [ ] Edit GUI Builder
- [x] Edit Part Feeding
- [ ] Edit points
- [ ] Edit programs
- [ ] Edit project
- [ ] Edit vision

When the [Edit Part Feeding] check box is selected, users belonging to the corresponding group are able to open the Part Feeding window.

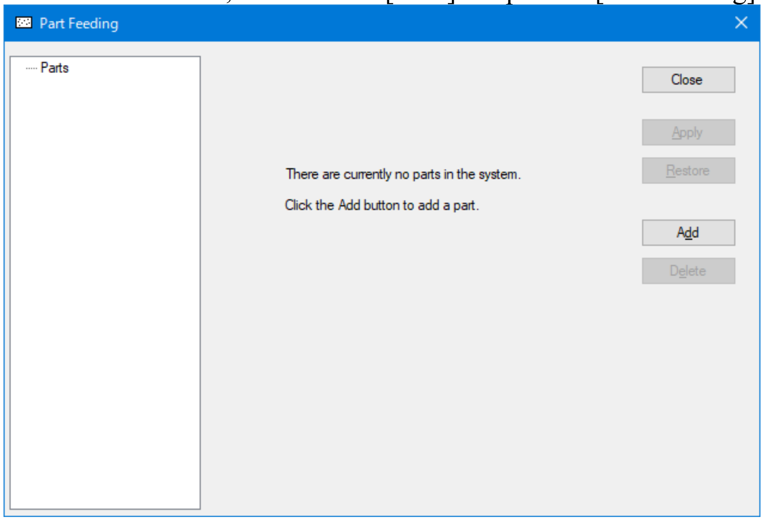
When the Edit Part Feeding check box is not selected, an error will occur when users belonging to the corresponding group attempt to open the Part Feeding window.

Refer to the following document for more details.
EPSON RC+ User's Guide 15. Security

2.2 Part Wizard

2.2.1 Add a new part

(1) In EPSON RC+ 7.0, select Menu-[Tool] to open the [Part Feeding] dialog box.



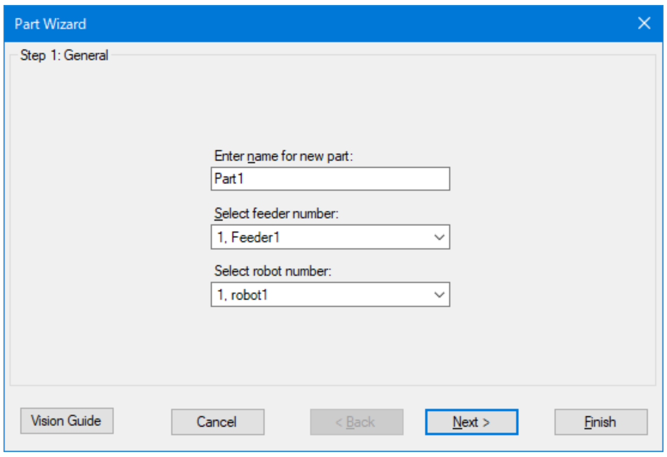
There are currently no parts in the system.

Click the Add button to add a part.

(2) Click the <Add> button. Run the Part Wizard.

2.2.2 General

This screen is used for configure general settings.



Enter name for new part: Part1

Select feeder number: 1, Feeder1

Select robot number: 1, robot1

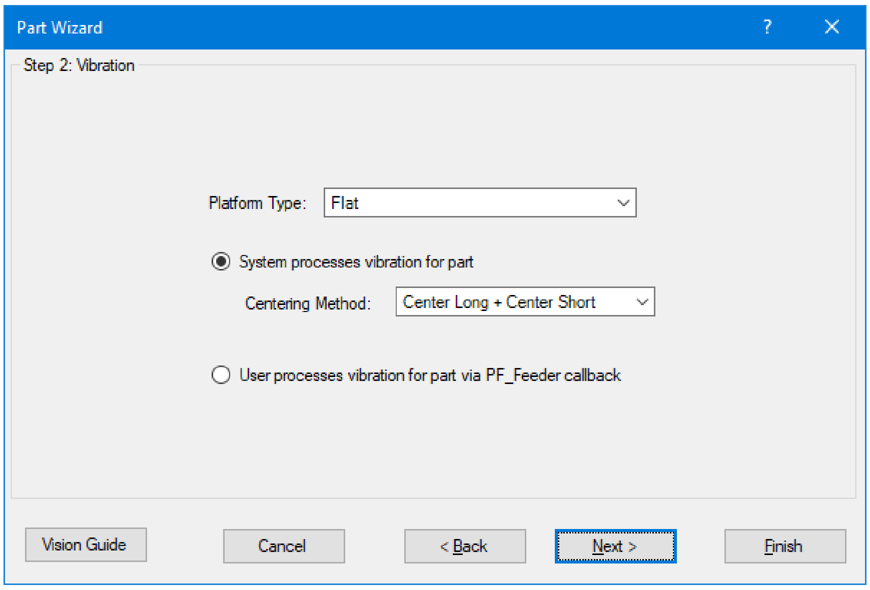
Item	Description
Enter name for new part	Write the name of the part. (Half-width alphanumeric characters and underscores only. Up to 32 characters in length)
Select feeder number	Select the feeder number used with this part. You can check the feeder number on the System Configuration screen.
Select robot number	Select the robot number.

This table describes the basic operation of the part wizard.
The part wizard is controlled using the five buttons at the bottom of the window.

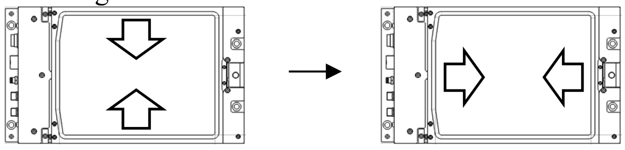
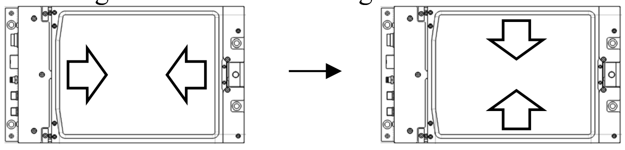
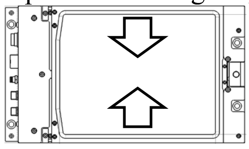
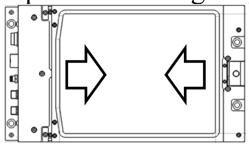
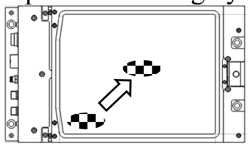
Button	Description
Vision Guide	Displays the Vision Guide screen. Use this screen to create vision sequences.
Cancel	Stops the Part Wizard.
Back	Returns to the previous STEP.
Next	Proceeds to the next STEP.
Finish	Finish the Part Wizard. If you click the <Finish> button in the middle of the wizard, the default values will be applied to the subsequent settings.

2.2.3 Vibration

Setting feeder vibration.


Part Wizard
Step 2: Vibration
Platform Type: Flat
● System processes vibration for part
Centering Method: Center Long + Center Short
○ User processes vibration for part via PF_Feeder callback

Item	Description
Platform Type	Set the platform type. a) A standard platform that can be purchased from Epson Flat : Flat platform Anti-rolling: Platform with anti-rolling processing Anti-stick : Platform with anti-stick processing b) Custom platforms must be designed and fabricated by the customer. Grooves : Platform with grooves for vertical parts Holes : Platform with holes for vertical parts Pockets : Platform with holes to align parts
System processes vibration for part	Controlling feeder by system. When selecting above b), this item can not be used.
User process vibration for part via PF_Feeder callback	Using PF_Feeder callback function.

Item	Description
Centering Method	When < System processes vibration for part > is selected, select the type of part centering operation. (operation to center and evenly distribute parts when the parts distribution is highly biased, such as when parts are put in.) No centering: No centering operated. Long axis centering+ Short axis centering: Operate centering in the direction of long axis then, operate centering in the direction of short axis.  Short axis centering+ Long axis centering: Operate centering in the direction of short axis then, operate centering in the direction of long axis.  Long axis centering: Operate centering only in the direction of long axis.  Short axis centering: Operate centering only in the direction of short axis.  Center by Shift: Operate centering by shift 

NOTE

The best centering method depends on the type of parts and the position of the hopper. The most effective method is one of the following methods.

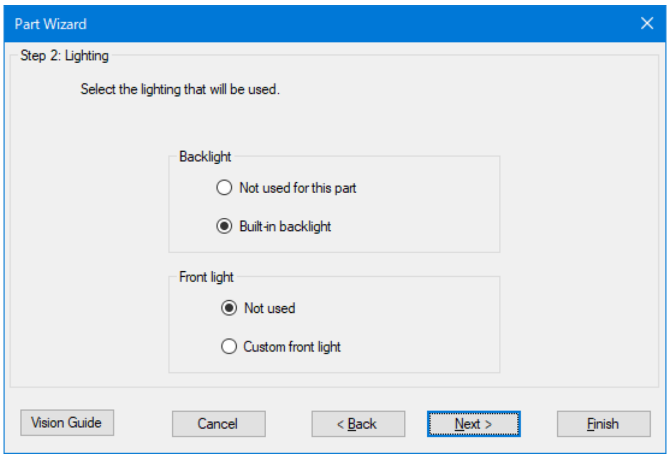
Long axis centering+ Short axis centering

Short axis centering+ Long axis centering

However, the feeder operating time will be longer compared to other centerings (or "no centering"). It is effective to select the one with the parts properly distributed and the shortest centering time.

2.2.4 Lighting

This screen is used to configure lighting settings.


Part Wizard

Step 2: Lighting

Select the lighting that will be used.

Backlight

Not used for this part

[✓ Built-in backlight]

Front light

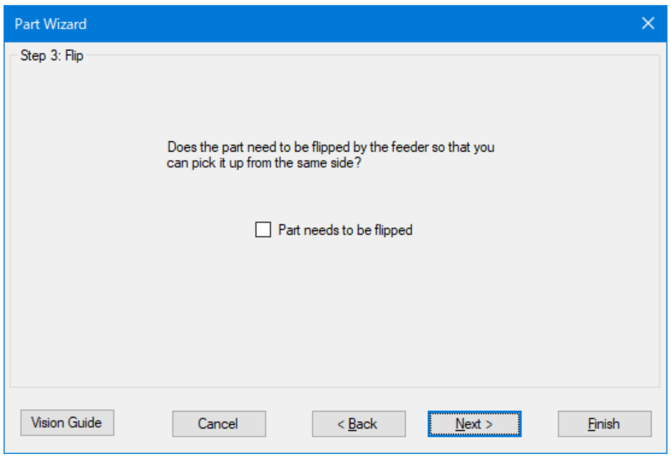
[✓ Not used]

Custom front light

Item		Description
Backlight		Set the backlight control method.
	Not used for this part	Select this to not use the feeder backlight during vision image capturing. This item is optional.
	Built-in Backlight	Select this to use the feeder backlight during vision image capturing. This option cannot be selected if the backlight is not installed.
Front light		Set the optional front light control method.
	Not used	Do not use a front light.
	Custom front light	Call the PF_Control callback function to control user-set custom lighting. This item is optional. For more information on the PF_Control callback function, refer to *4. Part Feeding Callback Function - PF_Control*.

2.2.5 Flip

This screen is used to configure whether the part needs to be flipped or not. If a part needs to be picked up from a particular side, then “Part needs to be flipped” should be checked.



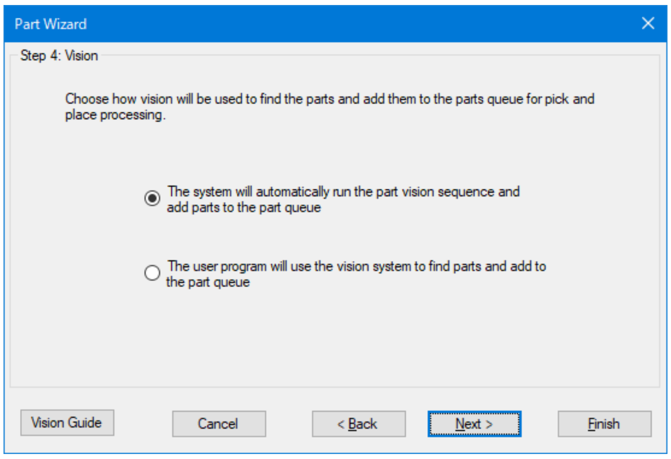
Does the part need to be flipped by the feeder so that you can pick it up from the same side?

☐ Part needs to be flipped

Item	Description
Part needs to be flipped	Select this check box for parts that require the front and back sides to be orientated properly. This enables a flip action used to change orientation at pick up. This may increase the number of parts that can be retrieved in a single feeder motion, improving cycle times.

2.2.6 Vision

This screen is used to configure vision settings.



Choose how vision will be used to find the parts and add them to the parts queue for pick and place processing.

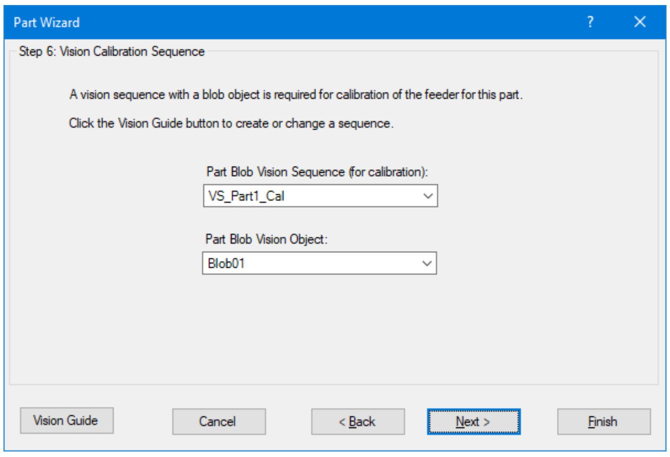
● The system will automatically run the part vision sequence and add parts to the part queue

○ The user program will use the vision system to find parts and add to the part queue

Item	Description
The system will automatically run the part vision sequence and add parts to the part queue	The system automatically performs vision processing. This setting is selected under normal circumstances.
The user program will use the vision system to find parts and add to the part queue	This item is optional. This enables the PF_Vision callback functions, and allows users to customize vision operations. For more information on the PF_Vision callback function, refer to *4. Part Feeding Callback Functions PF_Vision.*

2.2.7 Vision Calibration Sequence

This screen is used to configure vision settings.



A vision sequence with a blob object is required for calibration of the feeder for this part.

Click the Vision Guide button to create or change a sequence.

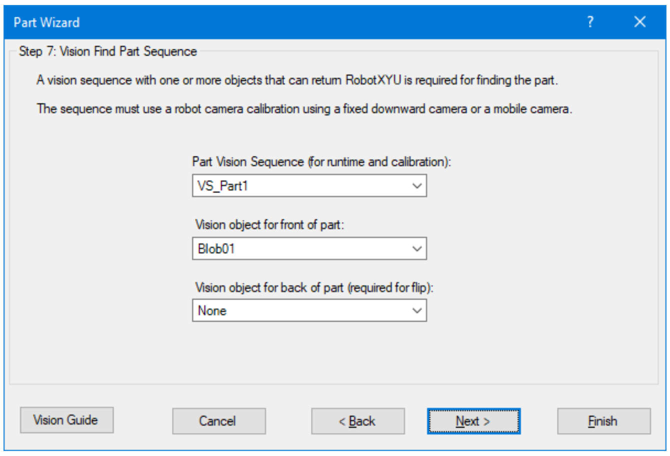
Part Blob Vision Sequence (for calibration): VS_Part1_Cal

Part Blob Vision Object: Blob01

Item	Description
Part Blob Vision Sequence	Required setting: This field is mandatory. Select the name of the vision sequence used for feeder calibration.
Part Blob Vision Object	Required setting: This field is mandatory. Select the name of the vision object used to detect parts while using the feeder backlight during feeder calibration. Only Blob can be selected.

2.2.8 Vision Find Part Sequence

This screen is used to configure vision settings.


Part Wizard

Step 7: Vision Find Part Sequence

A vision sequence with one or more objects that can return RobotXYU is required for finding the part.

The sequence must use a robot camera calibration using a fixed downward camera or a mobile camera.

Part Vision Sequence (for runtime and calibration): VS_Part1

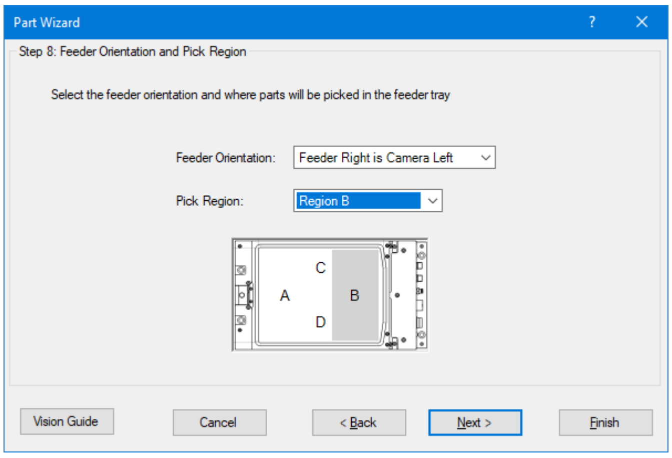
Vision object for front of part: Blob01

Vision object for back of part (required for flip): None

Item	Description
Part Vision Sequence	Required setting: This field is mandatory. Select the name of the vision sequence used to detect parts. Only sequence with robot calibrations can be selected.
Vision Object for front of part	Required setting: This field is mandatory. Select the name of the vision object used to detect parts for picking up. Only objects that return RobotXYU results are displayed.
Vision Object for back of part	This field is mandatory when flip is required (refer to *2.3.1. General)*. Select the name of the vision object used to detect parts that cannot be picked up because they are facing down. Only objects that return RobotXYU results are displayed.

2.2.9 Feeder Orientation and Pick Region

This screen configures settings relating to the feeder layout and part pick up.


Part Wizard

Step 8: Feeder Orientation and Pick Region

Select the feeder orientation and where parts will be picked in the feeder tray

Feeder Orientation: Feeder Right is Camera Left

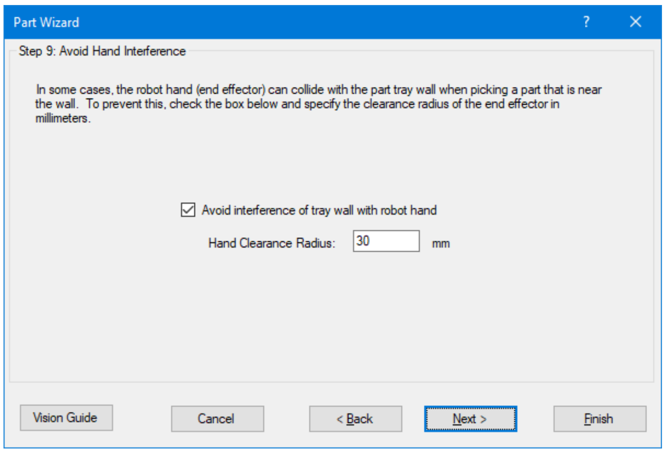
Pick Region: Region B

Item	Description
Feeder Orientation	Sets the feeder orientation when seen from the camera.

Pick Region	Sets the region for picking up parts when seen from the camera.
	Select either of Anywhere, Region A, B, C or D.
	The region that can be set differs depending on feeder model.
	IF-80: Anywhere
	IF-240: Anywhere, Region A, Region B, Region C, Region D
	IF-380 & IF-530: Anywhere, Region A, Region B

2.2.10 Avoid Hand Interference

This screen is used to configure Avoid Hand Interference settings.


Part Wizard

Step 9: Avoid Hand Interference

In some cases, the robot hand (end effector) can collide with the part tray wall when picking a part that is near the wall. To prevent this, check the box below and specify the clearance radius of the end effector in millimeters.

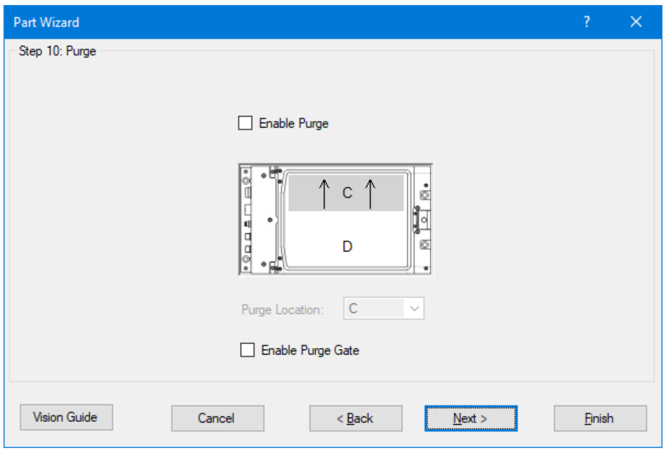
☑ Avoid interference of tray wall with robot hand

Hand Clearance Radius: 30 mm

Item	Description
Avoid interference with robot hand	Select this check box to enable a function used to avoid interference between the end effector and the platform.
Hand Clearance Radius	Parts within this distance from the circumference of the platform (set in the vision search window) will not be included in the coordinates queue. Units are in mm.

2.2.11 Purge

This screen is used to configure purge.

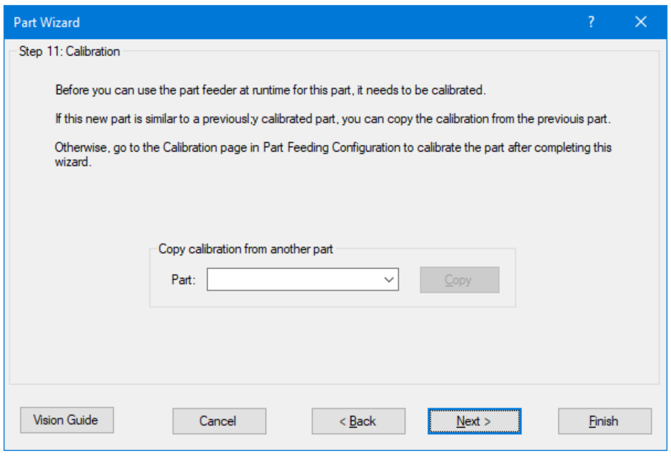


Item	Description
Enable Purge	Select this check box to enable purge.
Enable Purge Gate	Select this check box to enable Purge Gate.

Purge Location	Sets the location for purge. Parts will move (shift) to arrow direction during the purge operation. Select either of A, C or D.
	The location that can be set differs depending on feeder model.
	IF-80: A
	IF-240, 380, 530: C, D

2.2.12 Feeder Calibration

This screen is used to configure feeder calibration.



Before you can use the part feeder at runtime for this part, it needs to be calibrated.

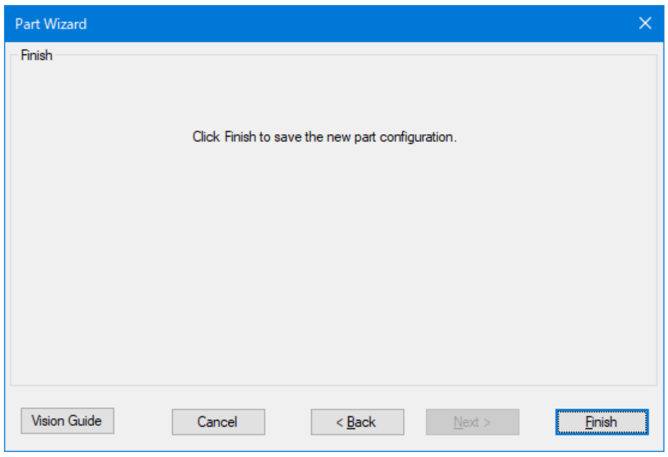
If this new part is similar to a previousl;y calibrated part, you can copy the calibration from the previouis part.

Otherwise, go to the Calibration page in Part Feeding Configuration to calibrate the part after completing this wizard.

Item		Description
Copy calibration from another part		Optional. Copies calibration results from another part to the part selected. Use this feature if the new part is similar to a previously calibrated part.
	Part	Specify the part to copy calibration results from.
	Copy	Copy calibration results.

2.2.13 Finish

Click the <Finish> button to proceed.



Click Finish to save the new part configuration.

2.3 Part Feeding Dialog

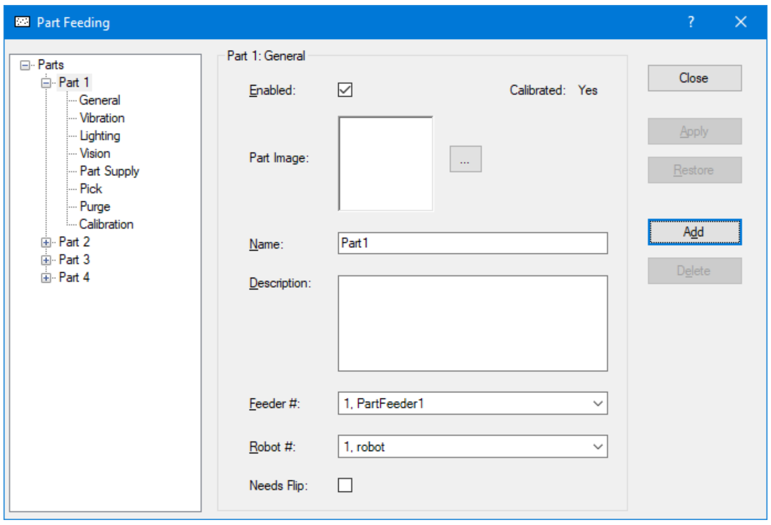
You can configure Part Feeding settings, calibration, adjustment and testing the feeder in the EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] dialog.

NOTE

Connect the EPSON RC+ to the Controller.
The Part Feeding window will not appear if EPSON RC+ is not connected to a Controller. An error will occur when attempting to open the Part Feeding window using a virtual controller or when offline or if the controller firmware version does not support Part Feeding. The Part Feeding license must also be enabled in the controller.

2.3.1 General

This screen is used to configure general settings.



Part 1: General

Enabled: ☑ Calibrated: Yes

Part Image: ...

Name: Part1

Description:

Feeder #: 1, PartFeeder1

Robot #: 1, robot

Needs Flip: ☐

Item	Description
Enabled	Select this check box to enable the part. An error will occur if a disabled part is specified when running the PF_Start command.
Part image	Available to registrate parts image. Click <…> button to select a file of part image.
Calibration	If No, feeder calibration is required. If Yes, feeder calibration has been completed.
Name	Write the name of the part. (Half-width alphanumeric characters and underscores only. Up to 16 characters in length)
Description	Write a description (comments) for the part. This field is optional. (Up to 256 characters in length)
Feeder #	Select the feeder number used with this part. You can check the feeder number on the System Configuration screen.
Robot #	Select the robot number.
Needs Flip	Select this check box for parts that require the front and back sides to be orientated properly. This enables a flip action used to change orientation at pick up. This may increase the number of parts that can be retrieved in a single feeder motion, improving cycle times.

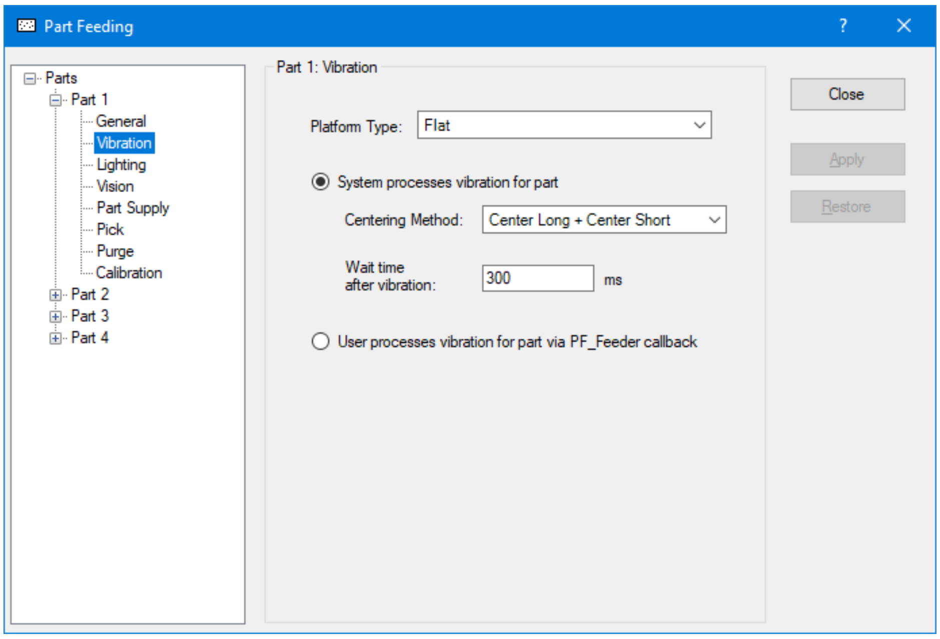
Button	Description
Close	Closes the screen.
Apply	Applies changes.
Restore	Undoes changes.
Add	Adds parts to the tree. Up to 32 types of parts can be added.
Delete	Deletes parts from the tree. Only parts at the bottom of the tree can be deleted. As parts in the middle of the tree cannot be deleted, deselect the [Enabled] check box to disable them instead.

NOTE

Delete all parts and build cause a build error. This is because Part Feeding commands and functions are no longer available. In this case, comment out the line.

2.3.2 Vibration

Setting feeder vibration.



Part 1: Vibration

Platform Type: Flat

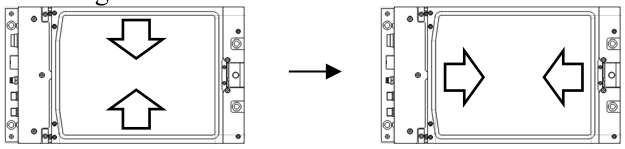
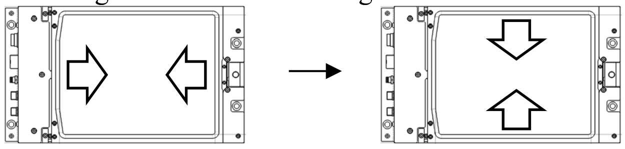
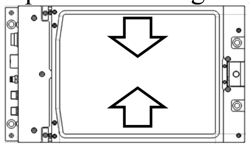
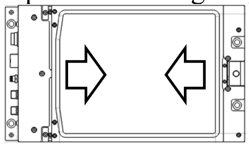
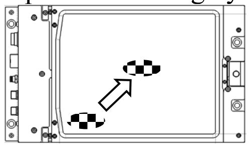
(●) System processes vibration for part

Centering Method: Center Long + Center Short

Wait time after vibration: 300 ms

( ) User processes vibration for part via PF_Feeder callback

Item	Description
Platform	Set the platform type. a) A standard platform that can be purchased from Epson Flat : Flat platform Anti-rolling: Platform with anti-rolling processing Anti-stick : Platform with anti-stick processing b) Custom platforms must be designed and fabricated by the customer. Grooves : Platform with grooves for vertical parts Holes : Platform with holes for vertical parts Pockets : Platform with holes to align parts
System process vibration for part	Controlling feeder by system. When selecting above b), this item can not be used.
User processes vibration for part	Using PF_Feeder callback function.

Item	Description
Centering Method	When < User processes vibration for part > is selected, select the type of part centering operation.(operation to center and evenly distribute parts when the parts distribution is highly biased, such as when parts are put in.) No centering: No centering operated. Long axis centering+ Short axis centering: Operate centering in the direction of long axis then, operate centering in the direction of short axis.  Short axis centering+ Long axis centering: Operate centering in the direction of short axis then, operate centering in the direction of long axis.  Long axis centering: Operate centering only in the direction of long axis.  Short axis centering: Operate centering only in the direction of short axis.  Center by Shift: Operate centering by shift 
Wait time after vibration	Specify a standby time to wait after the feeder stops vibrating before the vision system starts image capturing. Units are in milliseconds. Try increasing this value if the vision system has difficulty identifying parts, or if position aberrations occur when the robot grips onto parts on the feeder.

NOTE

The best centering method depends on the type of parts and the position of the hopper. The most effective method is one of the following methods.

Long axis centering+ Short axis centering

Short axis centering+ Long axis centering

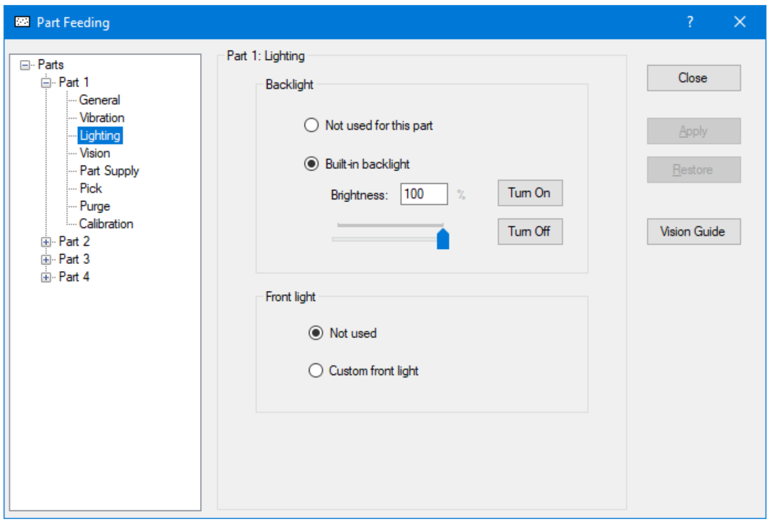
However, the feeder operating time will be longer compered to other centerings (or “no centering”). It is effective to select the one with the parts properly distributed and the shortest centering time.

NOTE ☞ There is no automatic calibration for “Center by Shift”.
(Refer to *2.4 Calibration & Test*.)

Button	Description
Close	Closes the screen.
Apply	Applies changes.
Restore	Undo changes.

2.3.3 Lighting

This page is used to configure lighting settings.



Part 1: Lighting

Backlight

- Not used for this part
- [✓ Built-in backlight]

Brightness: 100 %

Front light

- [✓ Not used]
- Custom front light

Item		Description
Backlight		Set the backlight control method.
	Not used for this part	Select this to not use the feeder backlight during vision image capturing. This item is optional.
	Built-in Backlight	Select this to use the feeder backlight during vision image capturing. This option cannot be selected if the backlight is not installed.
Turn On		Turns the feeder backlight on.
Turn Off		Turns the feeder backlight off.
Brightness		Sets the brightness of the feeder backlight. Set a value between 0 and 100%.
Front light		Set the optional front light control method.
	Not used	Do not use a front light.
	Custom front light	Call the PF_Control callback function to control user-set custom lighting. This item is optional. For more information on the PF_Control callback function, refer to 4. Part Feeding Callback Function - PF_Control.

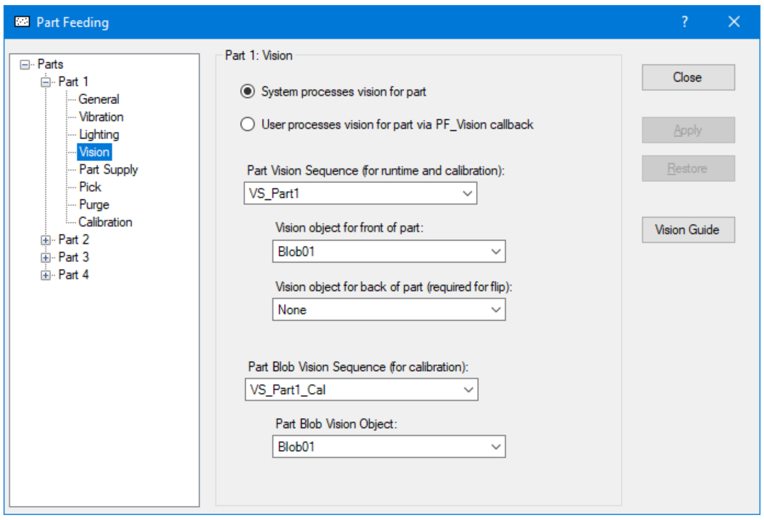
NOTE

Although the percentage of brightness can be set from 0 to 100%, the minimum, physical intensity may be limited depending upon the feeder model. Refer to [PF_BacklightBrightness] in *3. Part Feeding SPEL+ Command Reference*.

Button	Description
Close	Closes the screen.
Apply	Applies changes.
Restore	Undoes changes.
Vision Guide	Displays the Vision Guide screen. Use this screen to create vision sequences.

2.3.4 Vision

This screen is used to configure vision settings.
For more information on creating vision sequences, refer to *6. Vision Sequences Used With the Part Feeding Option.*



Part 1: Vision

(●) System processes vision for part
( ) User processes vision for part via PF_Vision callback

Part Vision Sequence (for runtime and calibration): VS_Part1

Vision object for front of part: Blob01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): VS_Part1_Cal

Part Blob Vision Object: Blob01

Item	Description
System processes vision for part	The system automatically performs vision processing. This setting is selected under normal circumstances.
User processes vision for part via PF_Vision callback	This item is optional. This enables the PF_Vision callback functions, and allows users to customize vision operations. For more information on the PF_Vision callback function, refer to *4. Part Feeding Callback Functions PF_Vision*.
Part Vision Sequence	Required setting: This field is mandatory. Select the name of the vision sequence used to detect parts Only sequences with robot calibrations can be selected
Vision Object for front of part	Required setting: This field is mandatory. Select the name of the vision object used to detect parts for picking up. Any object that supports multiple RobotXYU results can be selected.
Vision Object for back of part	This field is mandatory when performing a flip (refer to *2.3.1. General*), but it can also be specified when picking from front and back is required Select the name of the vision object used to detect parts that cannot be picked up because they are facing down. Any object that supports multiple RobotXYU results can be selected.
Part Blob Vision Sequence	Required setting: This field is mandatory. Select the name of the vision sequence used for feeder calibration. Only sequences with robot calibrations can be selected
Part Blob Vision Object	Required setting: This field is mandatory. Select the name of the vision object used to detect parts while using the feeder backlight during feeder calibration. Only Blob can be selected.

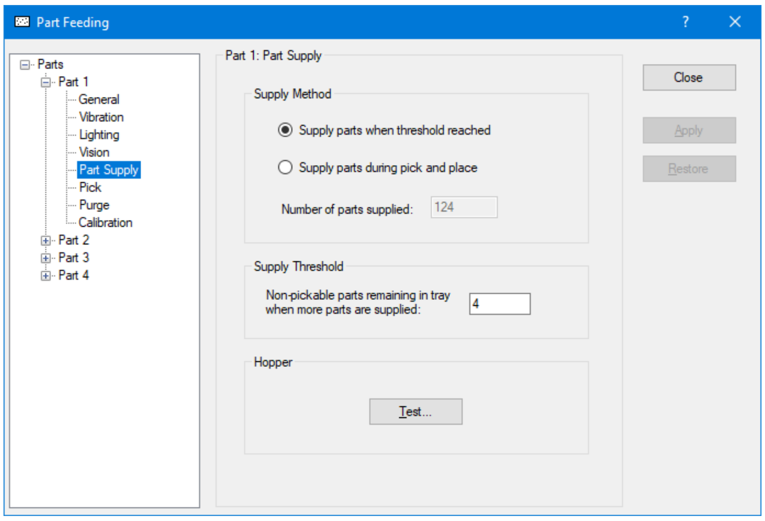
NOTE

All required settings must be configured. An error will occur during calibration or process operation if required settings have not been configured.

Button	Description
Close	Closes the dialog.
Apply	Applies changes. This is unavailable when non-optional fields have not been set.
Restore	Undoes changes.
Vision Guide	Displays the Vision Guide dialog. Use this screen to create vision sequences.

2.3.5 Part Supply

This screen is used to set the part supply method to the feeder.
A user-created program must be embedded into the PF_Control callback function to supply parts from a hopper. For more information on the PF_Control callback function, refer to *4. Part Feeding Callback Functions PF_Control.*



Part 1: Part Supply

Supply Method

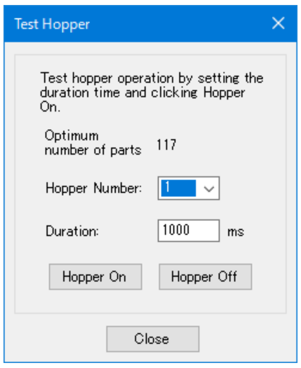
(●) Supply parts when threshold reached
( ) Supply parts during pick and place

Number of parts supplied: 124

Supply Threshold

Non-pickable parts remaining in tray when more parts are supplied: 4

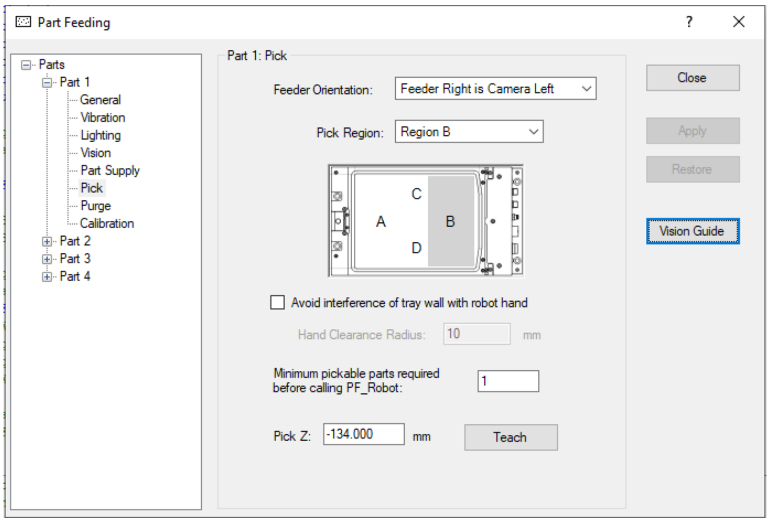
Hopper

Item	Description
Supply parts when threshold reached	Supply parts when the number of parts reaches the threshold.
Supply parts during pick and place	Add parts to optimize the number of parts on the feeder. This reduces robot cycle times compared to completely depleting all parts.
Parts remaining in tray when more parts are supplied.	When the number of non-pickable parts remaining in the tray is below this value, then more parts will be supplied. The default value is 4. If 0 is used, then all detected parts must be picked before more parts are supplied.
Test	Tests the hopper.  Test Hopper Test hopper operation by setting the duration time and clicking Hopper On. Optimum number of parts 117 Hopper Number: 1 Duration: 1000 ms Hopper On / Hopper Off / Close Hopper Number: Output terminal number of Feeder (It cannot be select when IF-80) Hopper On: Starts the hopper. Hopper Off: Stops the hopper. Duration: Specify the hopper operating time. Increase the length of time to increase the number of parts supplied.

Button	Description
Close	Closes the screen.
Apply	Applies changes.
Restore	Undoes changes.

2.3.6 Pick

This page configures settings relating to the feeder layout and part pick up.



Part 1: Pick

Feeder Orientation: Feeder Right is Camera Left

Pick Region: Region B

☐ Avoid interference of tray wall with robot hand

Hand Clearance Radius: 10 mm

Minimum pickable parts required before calling PF_Robot: 1

Pick Z: -134.000 mm

Item	Description
Feeder Orientation	Sets the feeder orientation as viewed by the camera.

Pick Region	Sets the region for picking up parts when seen from the camera. Select either of Anywhere, Region A, B, C or D. Note that when Anywhere is not selected, then Shift calibration is required.
	IF-80: Anywhere
	IF-240: Anywhere, Region A, Region B, Region C, Region D
	IF-380 & IF-530: Anywhere, Region A, Region B

Avoid interface with robot hand	Select this check box to enable a function used to avoid interference between the end effector and the platform tray walls.
Hand Clearance Radius	Parts within this distance from the circumference of the platform tray (set in the vision search window) will not be included in the coordinates queue. Units are in mm.
Minimum pickable parts required before calling PF_Robot	Normally the system will call the PF_Robot callback even when only 1 pickable part is found. The default value is 1. In general, this is the best setting because feeder vibration can take a long time to perform and there is no guarantee that more pickable parts will be found after the vibration. Regardless of this setting, the part queue is always loaded with the actual number of parts that were found by vision. The system determines how to vibrate based upon the quantity and distribution of parts. If the "Minimum pickable parts required before calling PF_Robot" is not satisfied, the system will perform an appropriate action. In some cases, system performance can be improved if there are a minimum number of pickable parts on the feeder before PF_Robot is called.
Teach	Opens the Teach dialog to allow teaching of the Z coordinate at part pick up. When using a 6-axis robot, the V and W orientations are taught in addition to the Z coordinate. Refer to *2.3.7 Teach Window* for more details.
Pick Z	This is the Z coordinate (local coordinates) at part pick up. The Z coordinate used for teaching will be shown here. Enter a new value to manually change this. Units are in mm.

NOTE

Check that interference does not occur between the end effector and the platform when "Avoid interface with robot hand" is enabled and the Hand radius has been adjusted.

NOTE

When "User processes vision for part via PF_Vision callback" is selected on the Vision screen and "Avoid interface with robot hand" is enabled, it is important to note that parts within the Hand radius distance from the platform edge will be included in the parts coordinates queue.

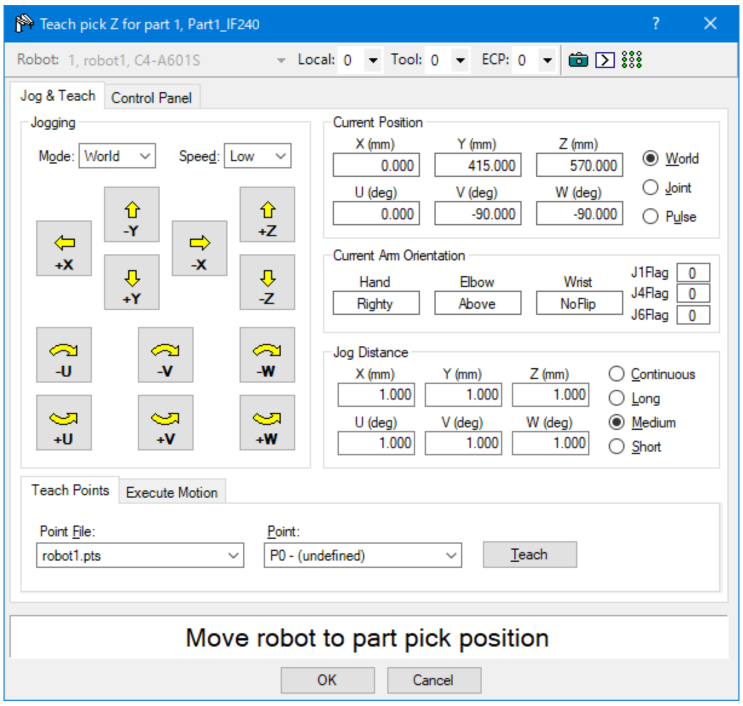
TIP

While V and W positional orientation values will not appear on the dialog when using a 6-axis robot, these values will be stored in internal memory.

Button	Description
Close	Closes the screen.
Apply	Applies changes.
Restore	Undo changes.

2.3.7 Teach Window

This dialog is used to teach the robot Z coordinate when picking up parts.



Teach pick Z for part 1, Part1_IF240

Robot: 1, robot1, C4-A601S Local: 0 Tool: 0 ECP: 0

Jog & Teach | Control Panel

Jogging

Mode: World Speed: Low

Current Position

X (mm)	Y (mm)	Z (mm)
0.000	415.000	570.000
U (deg)	V (deg)	W (deg)
0.000	-90.000	-90.000

[✓ World] Joint Pulse

Current Arm Orientation

Hand	Elbow	Wrist
Righty	Above	NoFlip

J1Flag 0
J4Flag 0
J6Flag 0

Jog Distance

X (mm)	Y (mm)	Z (mm)
1.000	1.000	1.000
U (deg)	V (deg)	W (deg)
1.000	1.000	1.000

Continuous Long [✓ Medium] Short

Teach Points | Execute Motion

Point File: robot1.pts Point: P0 - (undefined) Teach

Move robot to part pick position

OK Cancel

(1) Place a part on the platform.

(2) Use the Jog buttons (+X, –X, +Y, –Y) to align the robot into the Z coordinate and orientation used to pick up the part.

(3) Click the <OK> button.
Register the Z coordinate, V orientation and W orientation.

NOTE

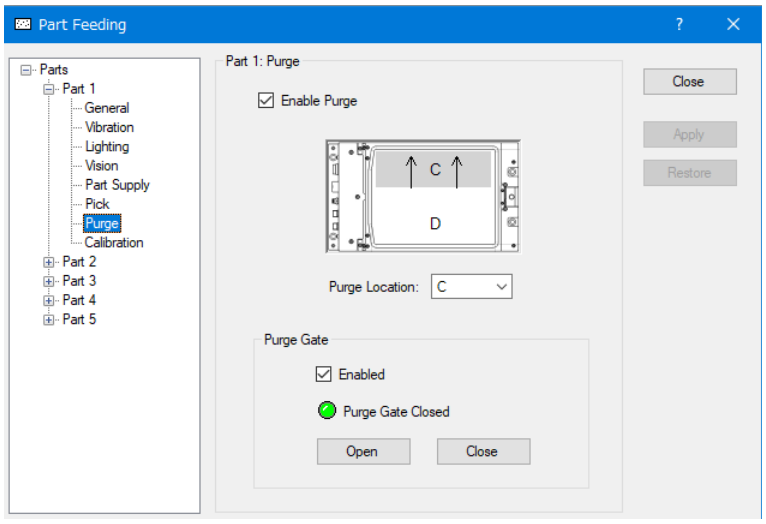
The robot selected here is the one that is referred to by the vision calibration set for the part detection vision sequence.

NOTE

If "User processes vision for part via PF_Vision callback" is selected on the Vision screen, you will need to program Z coordinate processing within the PF_Vision callback function. For more details, refer to *4. Part Feeding Callback Functions PF_Vision*.

2.3.8 Purge

This screen configures settings relating to purge.



Part 1: Purge

☑ Enable Purge

Purge Location: C

Purge Gate

☑ Enabled

● Purge Gate Closed

Item		Description
Enable Purge		Select this check box for enabling purge.

Purge Location		Sets the location for purge. Parts will move (shift) to arrow direction during the purge operation. Select either of A, C or D.
		The location that can be set differs depending on feeder model.
		IF-80: A
		IF-240, 380, 530: C, D

Purge Gate		Set the Purge Gate settings.
	Enabled	Select this check box for enabling the purge gate for the part. By default, the Purge Gate Enabled check box will be checked if the Purge Gate Installed check box was checked in the EPSON RC+ 7.0-Menu-[Setup]-[System Configuration]-[Controller]-[Part Feeders].
	Purge Gate Closed	Status indicator for the Purge Gate sensor Closed : Green, Not closed : Gray
	Open / Close	Allows opening and closing of the Purge Gate

NOTE

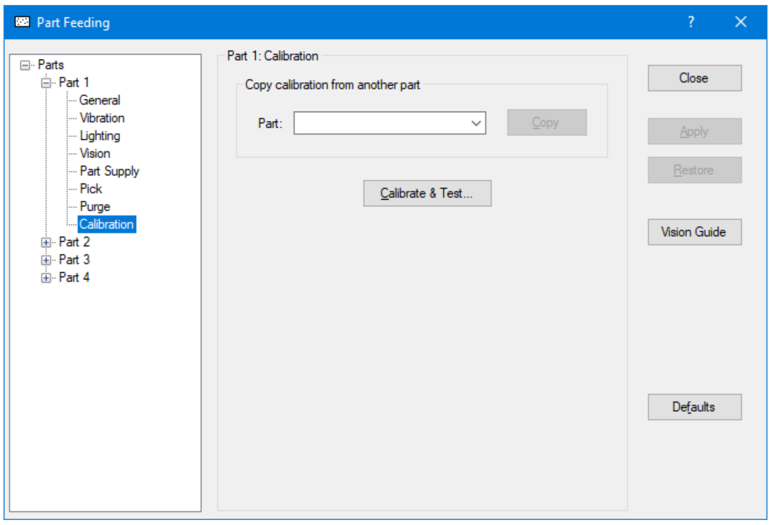
The Purge Gate groupbox will only be visible if the "Purge Gate Installed" check box was checked in the EPSON RC+ 7.0-Menu-[Setup]-[System Configuration]-[Controller]-[Part Feeders].

NOTE

If you try to move to another tab when the Purge Gate is not closed, the dialog "Purge gate will now be closed. Continue?" is displayed. Press OK to close the Purge Gate.

2.3.9 Calibration

This page describes feeder calibration, parameter editing, and testing.



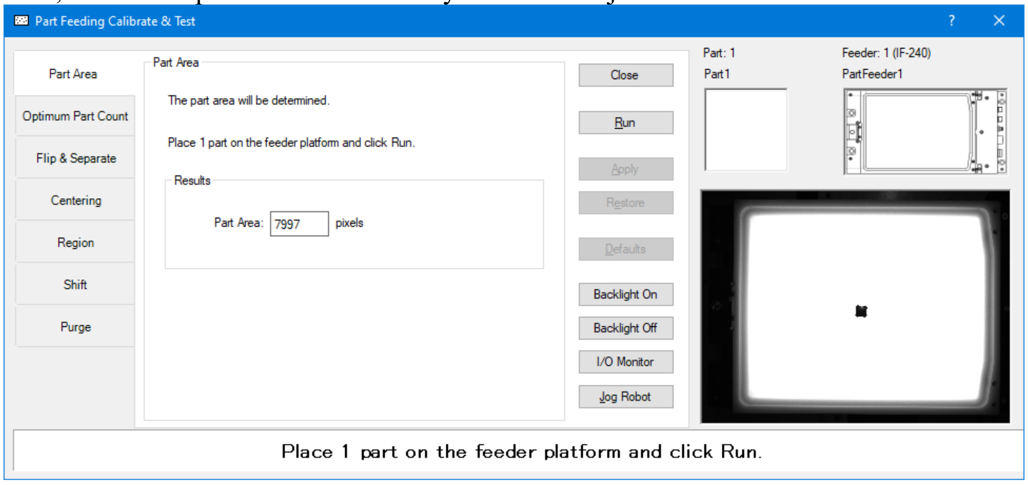
Item		Description
Copy calibration results from another part		Copies calibration results from another part to the part selected.
	Part	Specify the part to copy calibration results from.
	Copy	Copy calibration results.
Calibration&Test		Operates feeder calibration, parameter editing, and testing.

2.4 Calibration&Test

Calibration and testing allow you to:

- Feeder calibration for a part
- Adjusting feeder parameters for a part
- Feeder operation test

First, select the operation of the feeder you want to adjust from the tab on the left.



Part Area

The part area will be determined.

Place 1 part on the feeder platform and click Run.

Results

Part Area: 7997 pixels

Part: 1
Part1

Feeder: 1 (IF-240)
PartFeeder1

Place 1 part on the feeder platform and click Run.

1. Description of each tab

Tab	Description	Calibration Required/Optional
Part Area	Perform a part area calibration.	Required
Optimum Part Count	Adjust the optimal number of parts.	Optional
Separate	Adjust and test of Separate (part distribution).	Optional
Flip & Separate	Adjust and test of Flip (Re-orient the parts) and Separate (part distribution). This is displayed when flip is enabled. (see *"2.3.1 General"*.)	Optional
Centering	Adjust and test centering	Optional
Region	Adjusts and tests the pick region (the movement of the part when picking from a specific region). On the pick screen (see *2.3.6 Pick*), select A, B, C, or D areas for the part region.	Optional
Shift	Adjust and test shifting (movement of the parts).	Optional
Purge	Adjust and test the purge (discharging the part from the feeder). (Only IF-80) The Purge tab appears when Enable Purge is checked on the Purge page (see *2.3.8 Purge*)	Optional
Hopper	Test the IF-80 hopper. Displayed when the feeder is IF-80.	(None)

NOTE

When registering a new part, you cannot select anything other than the [Part Area] tab and the [Hopper] tab.

First, select the [Part Area] tab and perform the calibration.

If you have selected a “Pick Region”, it is strongly recommended that you perform the Automatic Region Calibration.

If you are using an IF-80 and you enabled purge, it is strongly recommended that you perform the Automatic Region Calibration.

If calibration is not performed for optional calibrations, then the default values will be used.

2. Description of each button

Button	Description
Close	Close the window.
Run / Abort	Start calibrating or testing the current screen. During execution, the display changes to “Abort”. Clicking the <Abort> button interrupts the currently running behavior.
Apply	Save your changes.
Undo	Undoes the changed value.
Defaults	Returns the value of the selected screen to its default value.
Backlight On/Off	Turn the feeder backlight on or off.
I/O Monitor	Start the I/O monitor. Used to control custom lighting, etc.
Jog Robot	Open the window jog the robot. Used to move a mobile camera to an image-capturing position.

NOTE

In the case of a system with a safeguard, when the <Run> button has been clicked and the safeguard is opened, the feeder will stop. The wizard screen does not change.

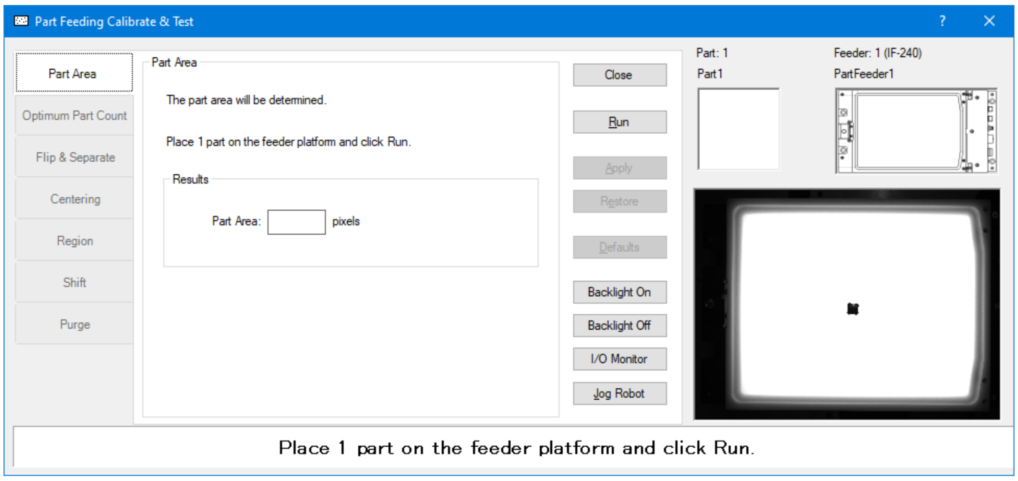
In this case, click the <Abort> button to stop calibration then close the safeguard and start calibration again.

3. Preparation

- Prepare the parts to be calibrated for feeder calibration. A specific quantity of parts will be used. The quantity could be determined by referring *2.4.2 Optimal Part Count.*
- When using a mobile camera:
  Click the <Jog Robot> button to move the robot to the image-capturing position.
- When you want to use custom lighting controlled by I/O:
  Click the <I/O Monitor> button that appears in the wizard to turn on the light.
- When adding a new part:
  Perform the “Part Area” calibration before any other calibration can be performed.
  Click the [Part Area] tab and <Run> the calibration.

2.4.1 Part Area

Calibrate the part area (the number of pixels per part).



Part Area

The part area will be determined.

Place 1 part on the feeder platform and click Run.

Results

Part Area: [ ] pixels

Place 1 part on the feeder platform and click Run.

1. Description of display items

Item	Description
Part Area	Number of pixels in the part area

2. Calibration

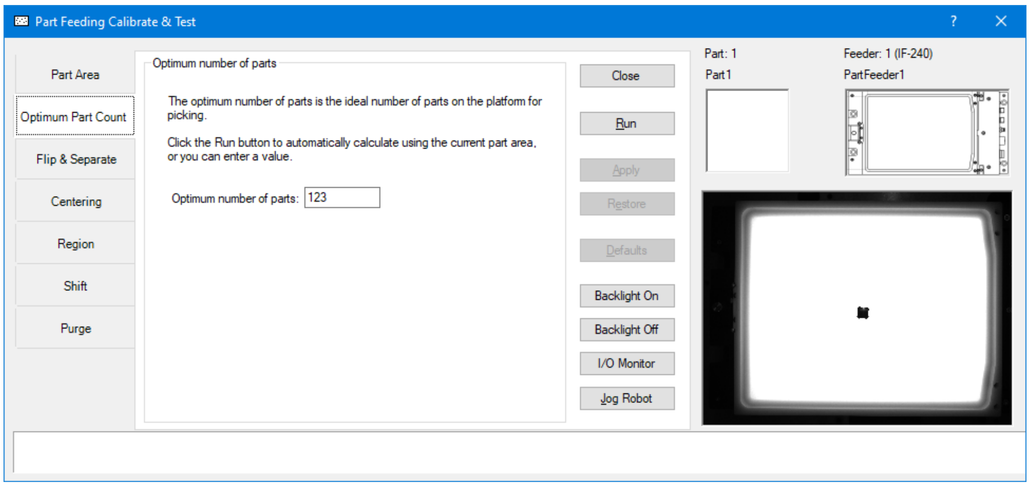
(1) Place one part on the platform.

(2) Click the <Run> button.
The number of pixels in the part area is measured.

(3) Click the <Apply> button.
Result saved.

2.4.2 Optimal Part Count

Calculate the optimal number of parts that should run on the feeder.



Optimum number of parts

The optimum number of parts is the ideal number of parts on the platform for picking.

Click the Run button to automatically calculate using the current part area, or you can enter a value.

Optimum number of parts: 123

1. Description of display items

Item	Description
Optimal number of parts	The calculated value of the optimal number of input parts. This value can be manually edited.

2. Guidelines for adjustment

This parameter is calculated on the assumption that the part is simple geometric shape like a square or circle. Therefore, if the part has an elongated shape or a hollow part in the center, set a smaller value than the calibration.
When used with a hopper, this value determines whether a part needs to be added or not. If you feel that the number of parts being provided by the hopper is too little, then increase this value.

3. Calibration

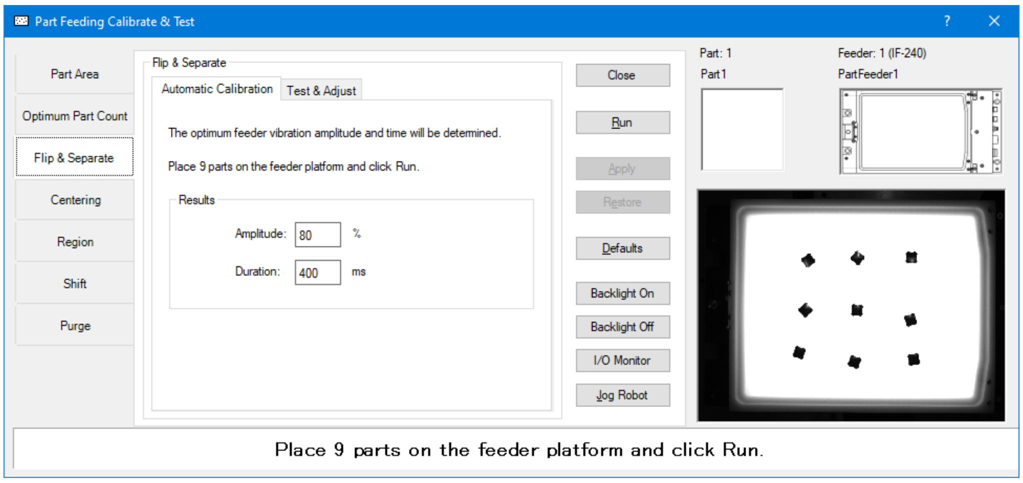
(1) Click the <Run> button.
The optimal number of parts is calculated.

(2) Click the <Apply> button
The result saved.

2.4.3 Flip & Separate - Automatic Calibration

To perform the flip and separate calibration, select the [Automatic Calibration] tab on the screen below.

This item is displayed as “Sparate” when Flip is disabled. (See *“2.3.1 General”*)



The optimum feeder vibration amplitude and time will be determined.

Place 9 parts on the feeder platform and click Run.

Results

Amplitude: 80 %

Duration: 400 ms

Place 9 parts on the feeder platform and click Run.

NOTE

The [Automatic Calibration] tab will not appear if the platform type (see *2.3.2 Vibration*) is set to “Grooves”, “Holes”, or “Pockets”.

1. Description of display items

Item	Description
Amplitude	Displays the strength of the amplitude of the vibration after the Automatic Calibration. Unit: %
Duration	Displays the duration time of the vibration after the Automatic Calibration. Unit: ms

2. Calibration

(1) Place the number of parts displayed in the instruction.

(2) Click the <Run> button.

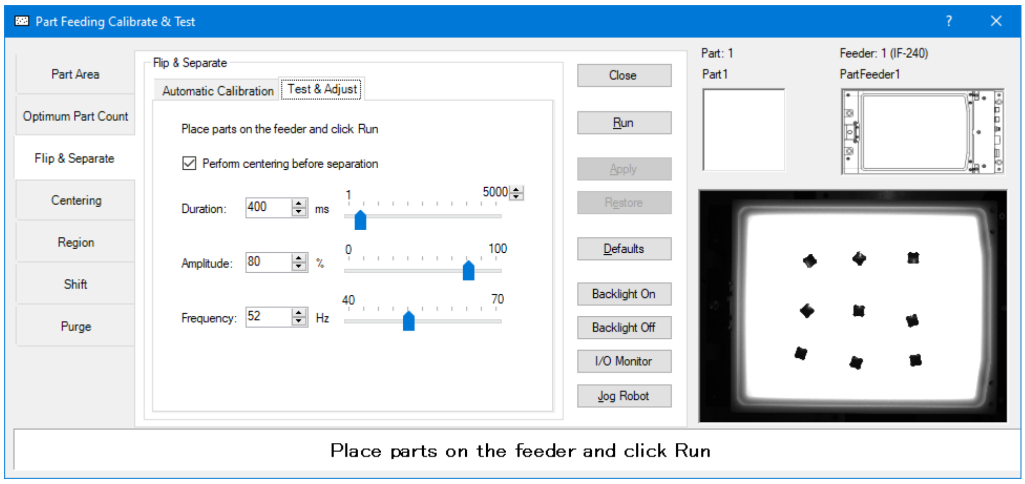
Feeder starts vibrating and the optimal Amplitude and Duration will be determined. The process can take several minutes to perform depending upon the physical characteristics of the part (weight, size, material, surface friction etc…).

(3) Click the <Apply> button

The result saved.

2.4.4 Flip & Separate - Test & Adjust

Test and adjust the flip and separate parameters.
This item is displayed as "Sparate" when Flip is disabled. (See *"2.3.1 General"*)
[Test & Adjust] tab on the screen below.



Part: 1
Part1
Feeder: 1 (IF-240)
PartFeeder1

Place parts on the feeder and click Run

☑ Perform centering before separation

Duration: 400 ms (1 – 5000)

Amplitude: 80 % (0 – 100)

Frequency: 52 Hz (40 – 70)

Place parts on the feeder and click Run

1. Description of display items

Item	Description
Perform centering before separation	If checked, perform the centering operation before the separation operation during the test operation. This applies only to test operation.
Amplitude	Set the strength of the amplitude of the vibration. Unit: %
Duration	Set the duration of the vibration. Unit: ms
Frequency	Set the frequency of the vibration. Unit: Hz

2. Guidelines for adjustment

The objective is to have the parts evenly distributed across the platform.
Adjust the amplitude and frequency so that the part can disperse or reorient itself as quickly as possible and do not fly off the platform.

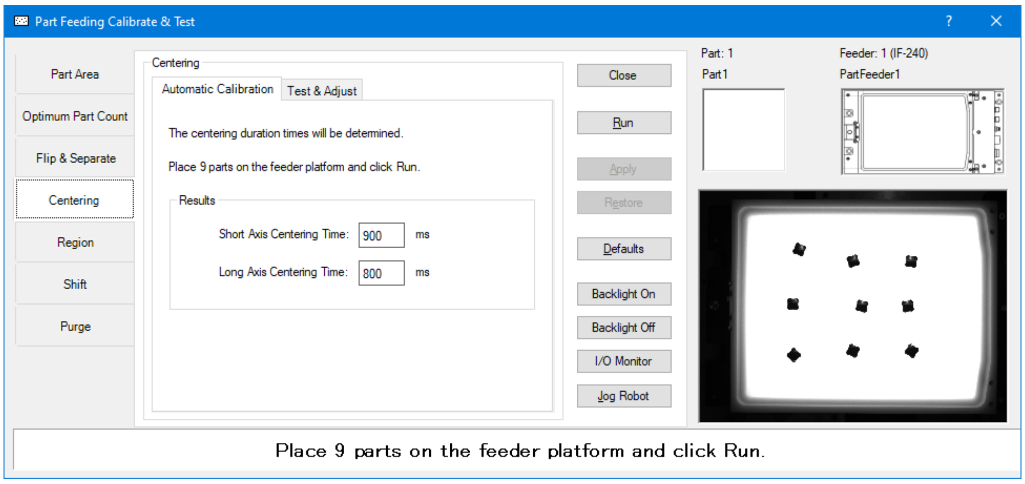
Set the vibration time to the shortest time that the part can be fully dispersed or reoriented.

3. Test

(1) Put in an appropriate number of parts (e.g., optimal number of parts).
(2) Click the <Run> button.
(3) Check the operation.
Adjust the parameters as needed and click the <Run> button again.
(For more detail, see *2.4.14 How to adjust feeder parameters*.)

2.4.5 Centering - Automatic Calibration

Perform centering calibration.
Select the [Automatic Calibration] tab on the screen below.



The centering duration times will be determined.

Place 9 parts on the feeder platform and click Run.

Results

Short Axis Centering Time: 900 ms

Long Axis Centering Time: 800 ms

Place 9 parts on the feeder platform and click Run.

NOTE

The [Automatic Calibration] tab will not appear if the platform type (see *2.3.2 Vibration*) is set to “Grooves”, “Holes”, or “Pockets”.

The [Automatic Calibration] tab is not displayed for the IF-80.

1. Description of display items

Item	Description
Short axis centering time	The duration of the short axis centering is displayed. Unit: ms
Long axis centering time	The duration of the long axis centering is displayed. Unit: ms

2. Calibration

(1) Place the number of parts displayed in the instruction.

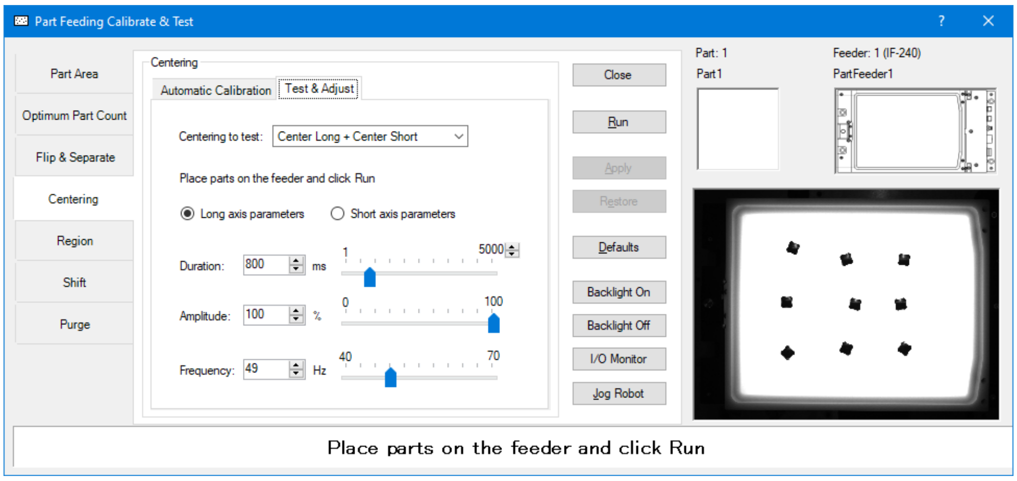
(2) Click the <Run> button.
Feeder starts vibrating and the optimal centering times will be determined.

(3) Click the <Apply> button
The result saved.

2.4.6 Centering - Test & Adjust

Test and adjust the centering parameters.

Select the [Test & Adjust] tab on the screen below.



Centering to test: Center Long + Center Short

Place parts on the feeder and click Run

(● Long axis parameters) ○ Short axis parameters

Duration: 800 ms (1 – 5000)

Amplitude: 100 % (0 – 100)

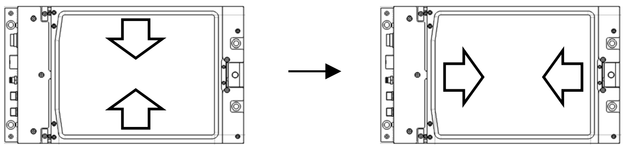
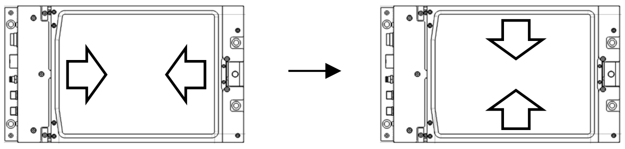
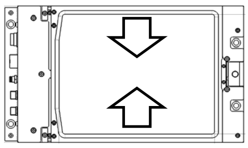
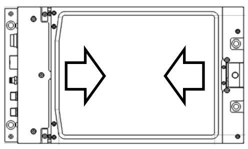
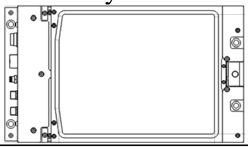
Frequency: 49 Hz (40 – 70)

Part: 1
Part1

Feeder: 1 (IF-240)
PartFeeder1

Place parts on the feeder and click Run

1. Description of display items

Item	Description
Centering to test	Select the type of centering behavior you want to test. Long Axis + Short Axis  Short Axis + Long Axis  Long Axis  Short Axis  Center By Shift 
Long axis parameters Short axis parameters	Choose which centering axis you want to adjust. It is not displayed when "Center By Shift" is selected.
Amplitude	Set the strength of the amplitude of the vibration. Unit: % It is not displayed when "Center By Shift" is selected.

Item	Description
Duration	Set the duration of the vibration. Unit: ms It is not displayed when "Center By Shift" is selected.
Frequency	Set the frequency of the vibration. Unit: Hz It is not displayed when "Center By Shift" is selected.

2. Guidelines for adjustment

Adjust the amplitude and frequency so that the parts concentrate in the specified direction as quickly as possible.
Set the vibration time to the time it takes centering to complete.

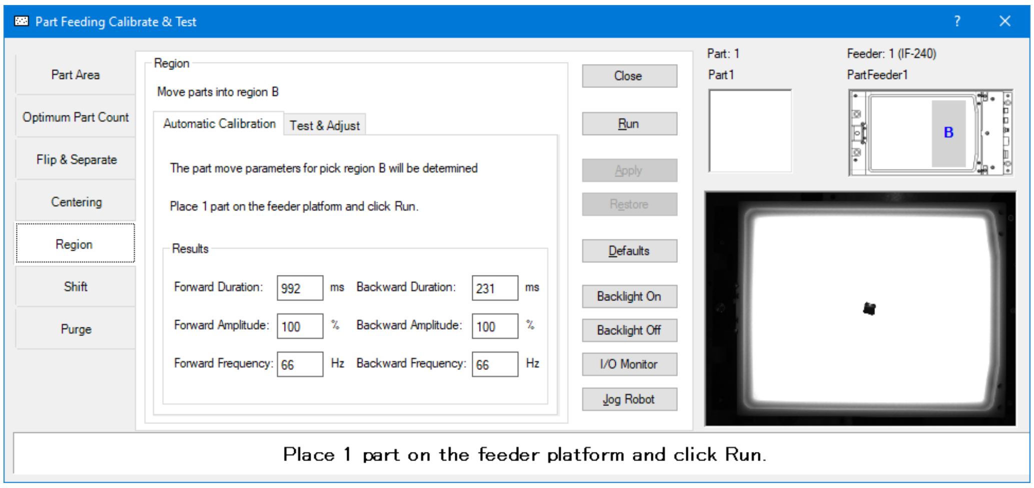
3. Test

(1) Put in an appropriate number of parts (e.g., optimal number of input parts).
(2) Click the <Run> button.
(3) Check the operation.
Adjust the parameters as needed and click the <Run> button again.
(For more detail, see *2.4.14 How to adjust feeder parameters*.)

2.4.7 Region - Automatic Calibration

Performs the calibration of the pick region shifting.
This item is displayed when “Pick region” selected. (See *2.3.6 Pick*)

Select the [Automatic Calibration] tab on the screen below.



Part: 1
Part1
Feeder: 1 (IF-240)
PartFeeder1

Region
Move parts into region B

The part move parameters for pick region B will be determined

Place 1 part on the feeder platform and click Run.

Results

Forward Duration:	992 ms	Backward Duration:	231 ms
Forward Amplitude:	100 %	Backward Amplitude:	100 %
Forward Frequency:	66 Hz	Backward Frequency:	66 Hz

Place 1 part on the feeder platform and click Run.

NOTE The [Automatic Calibration] tab will not appear if the platform type (see *2.3.2 Vibration*) is set to “Grooves”, “Holes”, or “Pockets”.

1.Description of display items

Item	Description
Forward Duration	Displays the time that is required to shift parts into the pick region. Unit: ms
Forward Amplitude	Displays the strength of the amplitude of the vibration. Unit: %
Forward Frequency	Displays the frequency of the vibration. Unit: Hz
Back Duration	Displays the time required to shift parts that have accumulated against the platform wall back into the pick region. Unit: ms
Back Amplitude	Displays the strength of the amplitude of the vibration. Unit: %
Back Frequency	Displays the frequency of the vibration. Unit: Hz

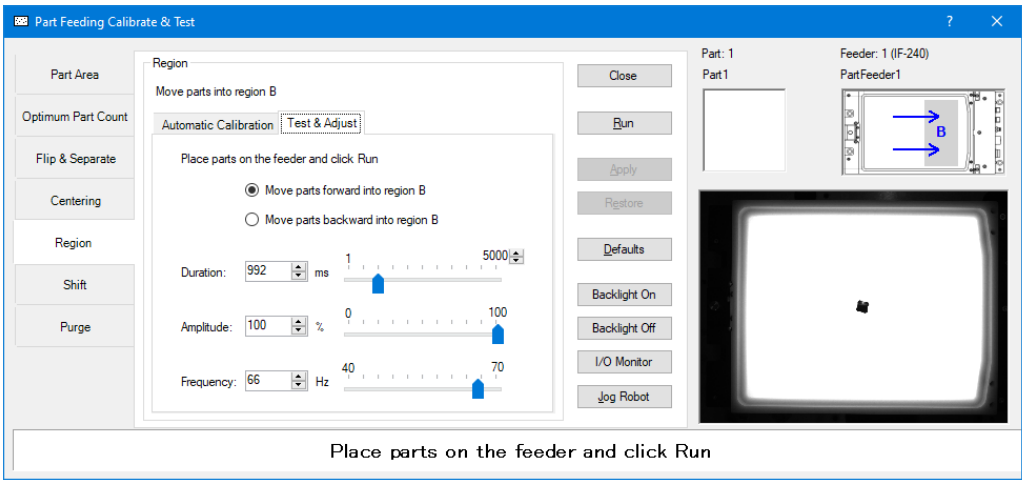
2. Calibration

(1) Place one part in the platform as displayed in the instruction
(2) Click the <Run> button.
Feeder starts vibrating and the optimal shift durations and amplitudes will be determined.
(3) Click the <Apply> button.
The result saved.

2.4.8 Region - Test & Adjust

Test and adjust the parameters for pick region shifting.
This item is displayed when "Pick region" selected. (See *2.3.6 Pick*)

Select the "Test & Adjust" tab on the screen below.



Move parts into region B

Place parts on the feeder and click Run

(●) Move parts forward into region B
( ) Move parts backward into region B

Duration: 992 ms (1 – 5000)
Amplitude: 100 % (0 – 100)
Frequency: 66 Hz (40 – 70)

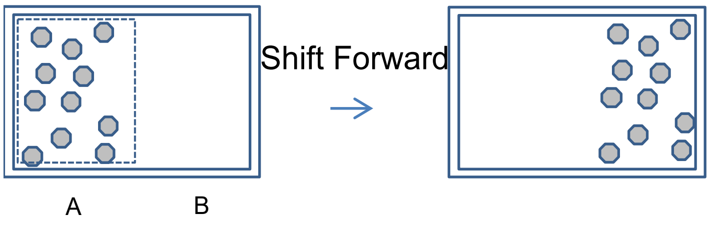
Place parts on the feeder and click Run

1. Description of display items

Item	Description
Move parts forward into region x Move parts backward into region x	Select a shifting direction. "x" is the pick region (either A, B, C, or D) selected on the Pick screen (see *2.3.6 Pick*). The shift direction will vary depending on the feeder installation direction. The actual shift direction is displayed in the upper right corner of the screen.
Amplitude	Set the strength of the amplitude of the vibration. Unit: %
Duration	Set the duration of the vibration. Unit: ms
Frequency	Set the frequency of the vibration. Unit: Hz

2. Guidelines for adjustment the Shift Forward and test

The Shift Forward is the process of moving parts into the specified pick region.
When the number of parts in the pick region decreases, the Shift Forward operation moves parts from the standby area into the pick region.
The figure below shows the ideal result of the previous shift operation to region B.

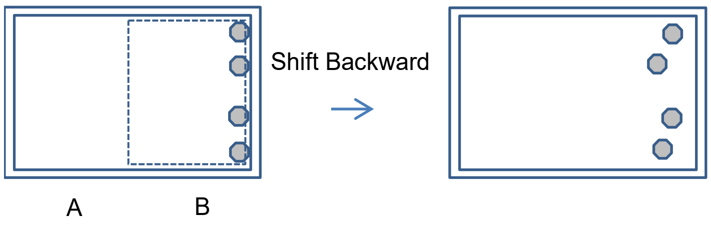


Adjust the duration time so that the part moves the proper distance (half the length of the platform's edge). If the part moves too far, they will get stuck at the edge of the platform. If the part is not moved enough, the area will not be supplied with enough parts, resulting in inefficiency.

Adjust the amplitude and frequency so that the parts move properly (keep the distance of each parts before the movement). If the part does not move smoothly, they will touch or overlap each other, resulting in low efficiency.

(1) Put an appropriate number of parts (e.g. 1/2 of the optimal number of parts) into the displayed region.
(2) Click the <Run> button.
(3) Check the operation.
Adjust the parameters as necessary and click the <Run> button again.
(For more details, see *2.4.14 How to adjust feeder parameters*.)

3. Guidelines for adjustment the Shift Backward and test
The Shift Backward operation moves parts that have accumulated against the platform wall back into the pick region.
The figure below shows the ideal result of the Shift Backward operation to region B.



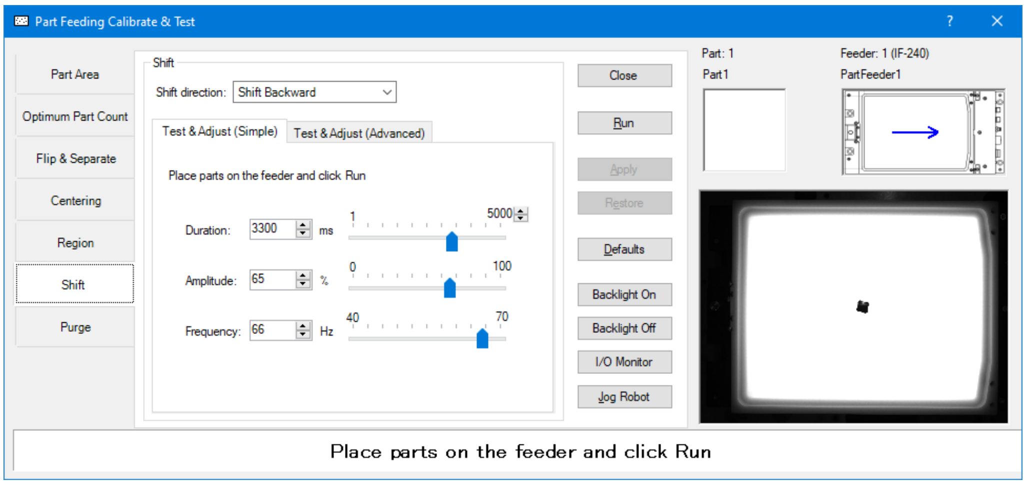
Adjust the duration time (and amplitude or frequency) so that the part moves an appropriate distance (enough so that the part near the corner of the platform returns to the pick region). Moving the part too far will cause the part to move out of the region, which is inefficient.

(1) Put an appropriate number of parts (e.g. 4 pcs) into the corner of the displayed region. (near the edge the platform)
(2) Click the <Run> button.
(3) Check the operation.
Adjust the parameters as necessary and click the <Run> button again.
(For more detail, see *2.4.14 How to adjust feeder parameters*.)

2.4.9 Shift - Test & Adjust (Simple)

Test and adjust the shift parameters.

Select the [Test & Adjust (Simple)] tab on the screen below.



Shift

Shift direction: Shift Backward

Place parts on the feeder and click Run

Duration: 3300 ms (1 – 5000)

Amplitude: 65 % (0 – 100)

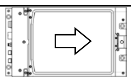
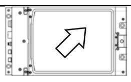
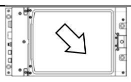
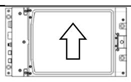
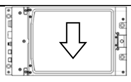
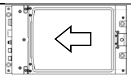
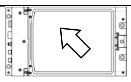
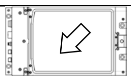
Frequency: 66 Hz (40 – 70)

Part: 1
Part1

Feeder: 1 (IF-240)
PartFeeder1

Place parts on the feeder and click Run

1. Description of display items

Item	Description
Shift direction	Select the direction of the shift you want to test. | Shift direction | Direction to move parts | | Forward |  | | Forward Left |  | | Forward Right |  | | Left |  | | Right |  | | Backward |  | | Backward Left |  | | Backward Right |  | The shift direction will vary depending on the feeder installation direction. The actual shift direction is displayed in the upper right corner of the screen.
Amplitude	Set the strength of the amplitude of the vibration. Unit: %
Duration	Set the duration of the vibration. Unit: ms
Frequency	Set the frequency of the vibration. Unit: Hz

2. Guidelines for adjusting

Adjust the amplitude and frequency so that the part moves as quickly and smoothly as possible in the specified direction.
Set the duration time to the time it takes to complete the intended motion.

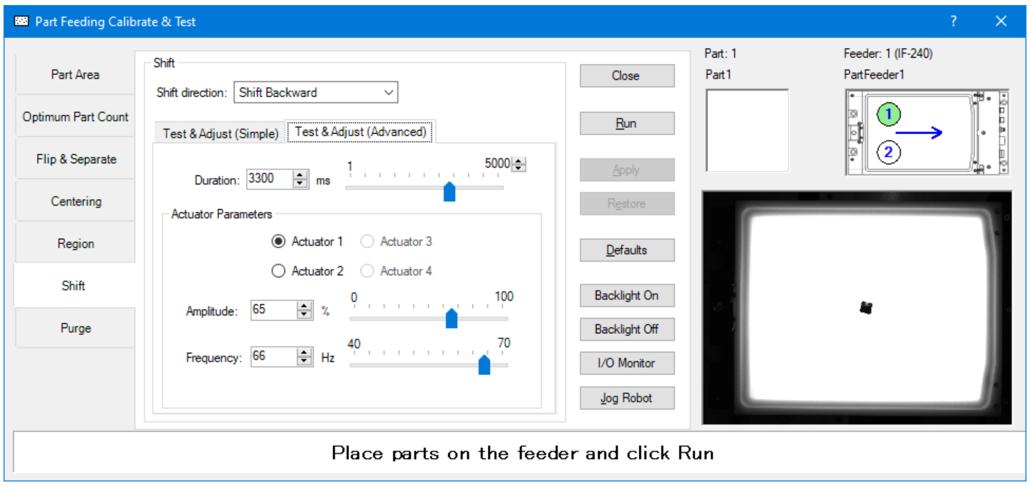
3. Test

(1) Put an appropriate number of part (e.g., 4 pcs).
(2) Click the <Run> button.
(3) Check the operation.
Adjust the parameters as necessary and click the <Run> button again.
(For more detail, see 2.4.14 How to adjust feeder parameters.)

2.4.10 Shift - Test & Adjust (Advanced)

Test and adjust the shift parameters.

Select the [Test & Adjust (Advanced)] tab on the screen below.


Part Feeding Calibrate & Test

Part Area
Optimum Part Count
Flip & Separate
Centering
Region
Shift
Purge

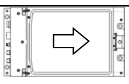
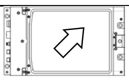
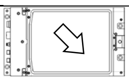
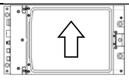
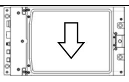
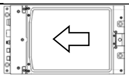
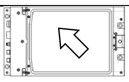
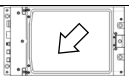
Shift
Shift direction: Shift Backward
Test & Adjust (Simple) | Test & Adjust (Advanced)
Duration: 3300 ms (1 – 5000)
Actuator Parameters
(●) Actuator 1 ( ) Actuator 3
( ) Actuator 2 ( ) Actuator 4
Amplitude: 65 % (0 – 100)
Frequency: 66 Hz (40 – 70)

Close
Run
Apply
Restore
Defaults
Backlight On
Backlight Off
I/O Monitor
Jog Robot

Part: 1
Part1
Feeder: 1 (IF-240)
PartFeeder1

Place parts on the feeder and click Run

1. Description of display items

Item	Description
Shift direction	Select the direction of the shift you want to test. | Shift direction | Direction to move parts | | Forward |  | | Forward Left |  | | Forward Right |  | | Left |  | | Right |  | | Backward |  | | Backward Left |  | | Backward Right |  | The shift direction will vary depending on the feeder installation direction. The actual shift direction is displayed in the upper right corner of the screen.
Duration	Set the duration of the vibration. This value is common to all actuators. Unit: ms
Actuator 1 Actuator 2 Actuator 3 Actuator 4	Select the actuator (the vibrating element inside the feeder) to be set. The position of the actuator will be displayed in the upper right corner of the screen.
Amplitude	Set the strength of the amplitude of the vibration. Unit: %

Item	Description
Frequency	Set the frequency of the vibration. This value is common to all actuators. Unit: Hz

2. Guidelines for adjustment

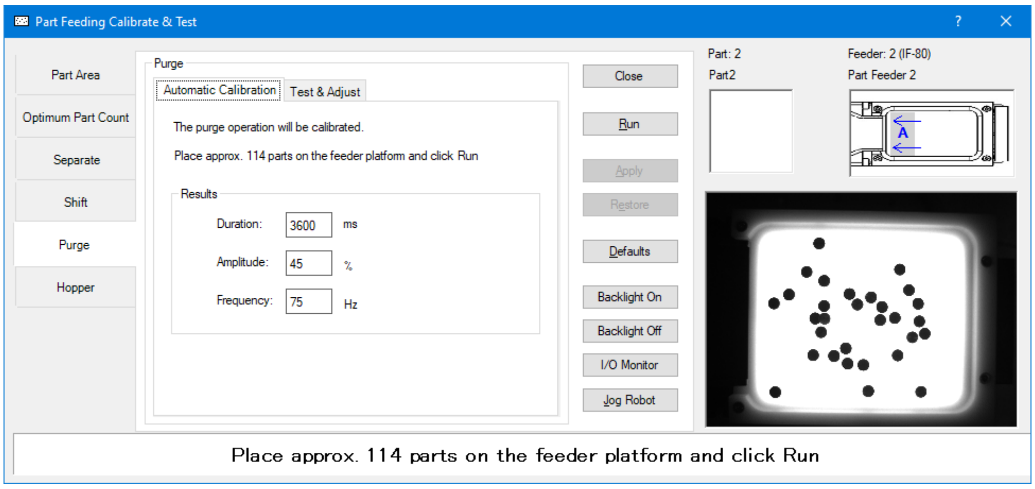
Adjust the amplitude and frequency so that the part moves as quickly and smoothly as possible in the specified direction.
Set the duration time to the time it takes to complete the intended motion.

3. Test

(1) Put an appropriate number of part (e.g., 4 pcs).
(2) Click the <Run> button.
(3) Check the operation.
Adjust the parameters as necessary and click the <Run> button again.
(For more detail, see *2.4.14 How to adjust feeder parameters*.)

2.4.11 Purge - Automatic Calibration (for IF-80)

To perform the purge calibration (IF-80 feeder only), select the [Automatic Calibration] tab on the screen below.
This item is displayed when Purge enabled. (See *2.3.8 Purge*)



The purge operation will be calibrated.

Place approx. 114 parts on the feeder platform and click Run

Results

Duration: 3600 ms

Amplitude: 45 %

Frequency: 75 Hz

Place approx. 114 parts on the feeder platform and click Run

NOTE

The [Automatic Calibration] tab will not appear if the platform type (see *2.3.2 Vibration*) is set to “Grooves”, “Holes”, or “Pockets”.

1. Description of display items

Item	Description
Amplitude	Set the strength of the amplitude of the vibration. Unit: %
Duration	Set the duration of the vibration. Unit: ms
Frequency	Set the frequency of the vibration. Unit: Hz

2. Calibration

(1) Empty the purge box (optional).

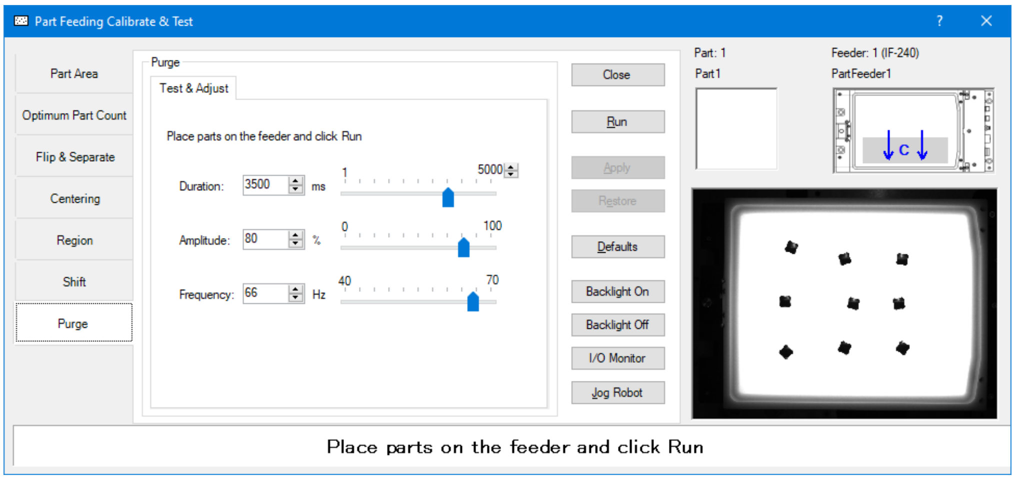
(2) Place the number of parts displayed in the instruction.

(3) Click the <Run> button.
Feeder starts vibrating, the optimal Amplitude, Duration and Frequency will be determined.
The process can take several minutes to perform depending upon the physical characteristics of the part (weight, size, material, surface friction etc…).

(4) Click the <Apply> button.
The result saved.

2.4.12 Purge - Test & Adjust

Test and adjust the purge parameters.
This item is displayed when Purge enabled. (See *2.3.8 Purge*)



Place parts on the feeder and click Run

Duration: 3500 ms (1 – 5000)

Amplitude: 80 % (0 – 100)

Frequency: 66 Hz (40 – 70)

Place parts on the feeder and click Run

1. Description of display items

Item		Description
Amplitude		Set the strength of the amplitude of the vibration. Unit: %
Duration		Set the duration of the vibration. Unit: ms
Frequency		Set the frequency of the vibration. Unit: Hz
Purge Gate		Operate the Purge Gate.
	Purge Gate Closed	Status indicator for the Purge Gate sensor Closed : Green, Not closed : Gray
	Open / Close	Allows opening and closing of the Purge Gate

NOTE

Even if the vibration amplitude is set to 0%, the feeder will still vibrate slightly. This is a specification.

NOTE

Purge Gate Closed will only be visible if the "Purge Gate Installed" check box was checked in the EPSON RC+ 7.0-Menu-[Setup]-[System Configuration]-[Controller]-[Part Feeders].

NOTE

If you try to move to another tab when the Purge Gate is not closed, the dialog "Purge gate will now be closed. Continue?" is displayed. Press OK to close the Purge Gate.

2. Guidelines for adjustment

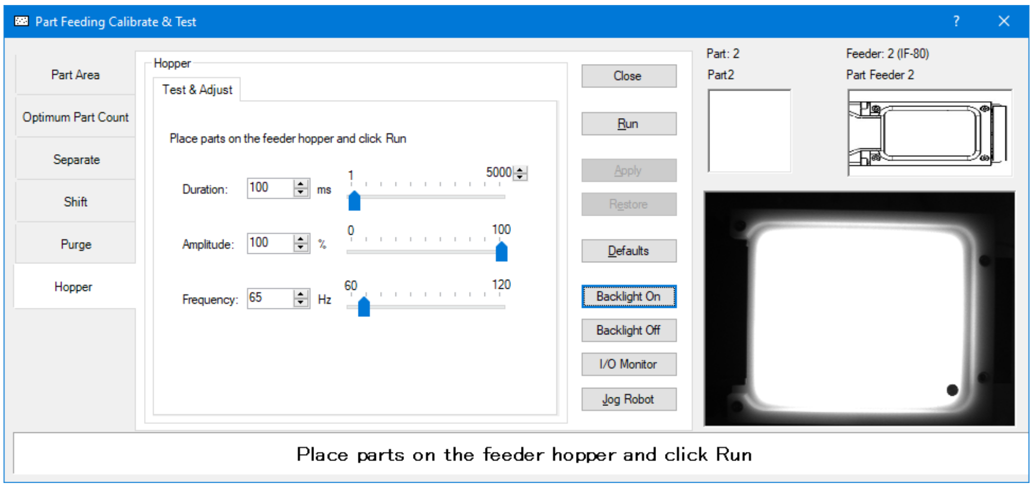
Adjust the amplitude and frequency so that the part is ejected from the feeder as quickly as possible.
Set the vibration time to the time it takes for the parts to be completely ejected.

3. Test

(1) Put the appropriate number of parts (e.g., the optimum number of parts).
(2) Open the purge gate (provided by the customer).
In the case of the IF-80, empty the purge box (optional).
(3) Check the operation.
Adjust the parameters as necessary and click the <Run> button again.
(For more detail, see *2.4.14 How to adjust feeder parameters*.)

2.4.13 Hopper - Test & Adjust (for IF-80)

Adjust the hopper parameters (IF-80 feeder only).



Part: 2
Part2
Feeder: 2 (IF-80)
Part Feeder 2

Place parts on the feeder hopper and click Run

Duration: 100 ms (1 – 5000)
Amplitude: 100 % (0 – 100)
Frequency: 65 Hz (60 – 120)

Place parts on the feeder hopper and click Run

1. Description of display items

Item	Description
Amplitude	Set the strength of the amplitude of the vibration. Unit: %
Duration	Set the duration of the vibration. Unit: ms
Frequency	Set the frequency of the vibration. Unit: Hz

2. Guidelines for adjustment

Adjust the amplitude and frequency so that the part moves smoothly.
Adjust the amount and condition of the parts in the hopper so that the parts are fed at a constant speed. In some cases, a divider plate can be attached to the hopper outlet to adjust the speed.

3. Test

(1) Load the appropriate number of parts (e.g., 5 to 10 times the optimal number of parts) into the hopper.
(2) Empty the purge box (optional).
(3) Click the <Run> button.
(4) Check the operation.
Adjust the parameters as necessary, return the part to the hopper, and click the <Run> button again.

2.4.14 How to adjust feeder parameters

NOTE ☞

Each parameter is pre-set to the appropreate value.
Therefore, you usually do not need to adjust these parameters manually.

1. Vibration amplitude

Setting a higher value speeds up the movement of the part. This reduces the time it takes to distribute and move parts and reduces cycle times.
On the other hand, if you increase the value too much, parts may jump out of the platform. The preferred method is to set a minimum value that is determined to be sufficient to move or distribute the part.

2. Vibration duration

Setting a longer time increases the amount of variance and movement of the part. This has the effect of increasing the number of parts that can be obtained in a per feeder operation. On the other hand, if you increase the value too much, the cycle time will be longer. The preferred method is to set a minimum value that is determined to be sufficient to move or distribute the part.

3. Vibration frequency

This value is preset to the resonant frequency, which is based on the weight of the platform and the spring constant of the feeder. Therefore, when working with heavy parts (such as metal), setting a value that is a little smaller than the set value may improve operation. In addition, changing the frequency when using a custom platform may improve the behavior of the part.
If you change this value, side effects such as changes in the movement behavior of the part may occur, so be careful when changing the value.

If the frequency is changed, the vibration noise produced may become louder. This is not a fault.

If you are concerned about the vibration noise, change the frequency by a few Hz from the frequency at which the noise is loudest (= the resonance frequency).

2.5 [File] Menu

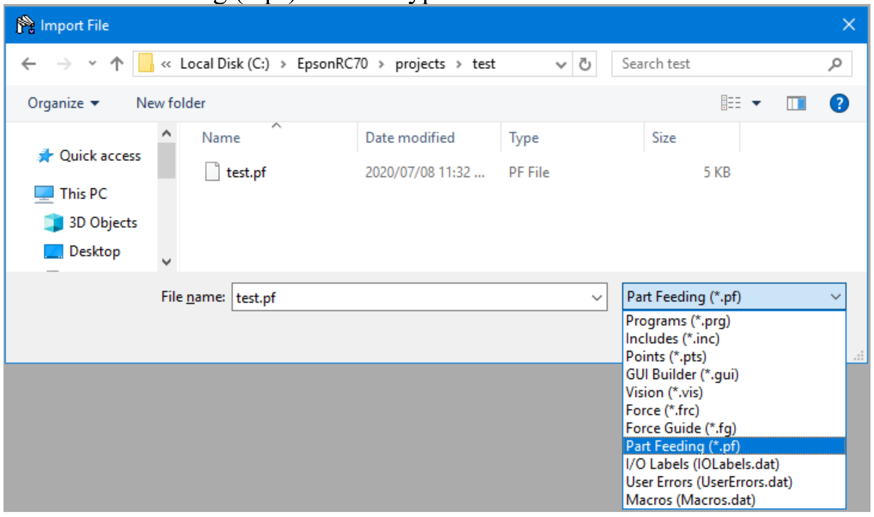
You can work with Part Feeding files in the EPSON RC+ 7.0 - Menu - [File].

2.5.1 [Import] (File Menu)

You can import parts from other EPSON RC+ 7.0 projects.

(1) Select EPSON RC+ 7.0 - Menu - [File] - [Import].

(2) Select “PartFeeding (*.pf)” for file type.

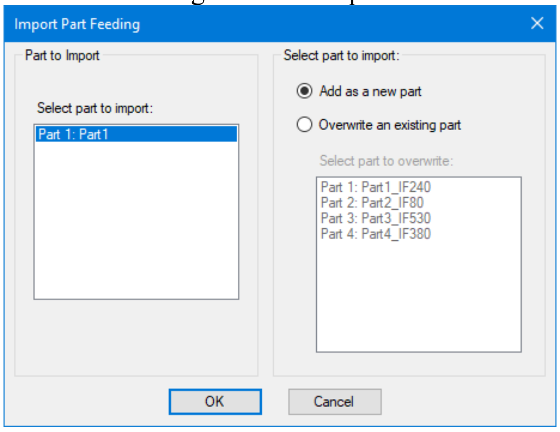


test.pf 2020/07/08 11:32 ... PF File 5 KB

File name: test.pf

Part Feeding (*.pf)

(3) Use the following screen to import files.



Item	Description
Select part to import	Selects the part to import.
Add as a new part	Select this to add the part as a new part.
Overwrite an existing part	Select this to overwrite existing part data.
Select part to overwrite	Select the part to overwrite when “Overwrite an existing part” is selected.

3. Part Feeding SPEL$^+$ Command Reference

This section provides a description of Part Feeding SPEL$^+$ commands that can be used from user-created SPEL$^+$ programs. The following is the list of commands.

Command/Function	Description/Application
PF_AccessFeeder	Get a lock for robot accessing feeder
PF_ActivePart	Switching the active part when operating multiple parts
PF_Abort	Forces the Part Feeding process to stop
PF_Backlight	Used to control backlight on and off at runtime when the vision callback is used
PF_BacklightBrightness	Sets the backlight brightness at runtime
PF_Center	Operate Centering behavior
PF_CenterByShift	Operate Centering behavior by shift
PF_Flip	Operate flip behavior
PF_Info function	Retrieves Part Feeding properties
PF_InitLog	Enables log file output and specifies the destination path
PF_IsStopRequested function	Returns whether the PF_Stop command has been issued
PF_Name$ function	Retrieves the part name from the part ID
PF_Number function	Retrieves the part ID from the part name
PF_Purge function	Starts purge motion
PF_Output	Turns the feeder output terminal on or off Used when controlling two hoppers.
PF_OutputOnOff	Turns the feeder output terminal on or off Used when controlling an individual hopper.
PF_PurgeGate	Controls the opening and closing of the purge gate
PF_PurgeGateStatus	Get the status of the purge gate close sensor.
PF_QtyAdjHopperTime function	Returns the estimated amount of time that is required to supply the optimal number of parts.
PF_QueAdd	Adds data (point data, part orientation, user data) to the coordinates queue
PF_QueAutoRemove	Configures the auto remove function for the coordinates queue
PF_QueAutoRemove function	Returns the status of the auto remove function for the coordinates queue
PF_QueGet function	Retrieves point data from the coordinates queue
PF_QueLen function	Returns the data quantity registered to the coordinates queue
PF_QueList	Displays the coordinates queue data list
PF_QuePartOrient	Resets and displays the part orientation (integer) registered to the coordinates queue
PF_QuePartOrient function	Returns the part orientation registered to the coordinates queue
PF_QueRemove	Deletes data in the coordinates queue
PF_QueSort	Sets and displays the coordinates queue sorting method
PF_QueSort function	Returns the coordinates queue sorting method
PF_QueUserData	Resets and displays the user data (real) registered to the coordinates queue
PF_QueUserData function	Returns the user data (real) registered to the coordinates queue
PF_ReleaseFeeder	Release the lock getting by PF_AccessFeeder
PF_Shift	Operate shift behavior
PF_Start	Starts the Part Feeding process
PF_Stop	Issues a Part Feeding process stop request

PF_Abort

Forces the Part Feeding process to stop for the specified part.

Syntax

PF_Abort *part ID*

Parameters

part ID	Specify the part ID (integer number from 1 to 32).

Return Values

None

Description

Immediately aborts the Part Feeding process for the specified part.
Unlike PF_Stop, this aborts the callback function in progress.
Nothing will occur when using this function when the Part Feeding process has not been started.
When operating in multi-part, any PartID set in PF_Start can be specified.
Cannot be executed from a virtual controller or command window.

Example

```
PF_Abort 1
```

PF_AccessFeeder

PF_AccessFeeder locks access to a feeder to prevent potential collisions on a multi-robot / one feeder system. This statement is required when two robots are sharing the same feeder at the same time. PF_AccessFeeder gets a lock for the robot accessing a feeder. When the lock has already been acquired, PF_AccessFeeder pauses the task until the lock is released or until the specified timeout (optional) is reached. After the robot finishes using a feeder, it must release the lock using the PF_ReleaseFeeder statement in order to relinquish the feeder to the other robot (please refer to the PF_ReleaseFeeder Statement for more information).

Example:
Task 1 executes PF_AccessFeeder 1, then Task 1 continues.
Task 2 executes PF_AccessFeeder 1, then Task 2 is paused.
Task 1 executes PF_ReleaseFeeder 1, then Task 2 is resumed.

This command is used for exclusive control in a multi-robot configuration.

Syntax

PF_AccessFeeder *feeder number / feeder name [, timeout]*

Parameters

feeder number	Specify the feeder number (integer number) as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.
timeout	Specify the timeout (in seconds) for waiting for the lock to be released as an expression or a number (real number). Optional.

Return Values

None

Description

When timeout is specified, the result can be determined by the return value of the TW function (see EPSON RC+ 7.0 SPEL+ Language – TW Function).
False - The lock was released or the lock was successfully acquired.
True - Timeout has been reached .
The behavior of the feeder is unaffected by the acquisition/release of locks.
At the end of the task, the locks acquired in that task are automatically released.
If the PF_AccessFeeder is executed twice in a row for the same feeder in the same task, an error occurs.
Cannot be executed from a virtual controller or command window.

Example

Here is an example of picking a part on one feeder with two robots.
Robot 1 gets the part on the feeder and moves to point place1.
When robot 1 reaches 50% of the movement path, robot 2 moves to get the part.

```
Function Main

  MemOff PartsToPick

Motor On

  PF_Start 1

  Xqt Robot1PickPlace
  Xqt Robot2PickPlace
Fend
```

```
Function Robot1PickPlace
  Robot 1
  Do
    If MemSw(PartsToPick) = On
      If PF_QueLen(1) > 0 Then
        PF_AccessFeeder 1
        P0 = PF_QueGet(1)
        PF_QueRemove 1
        Jump P0 /R
        On 5
        Wait .5
        Jump place ! D30; PF_ReleaseFeeder 1 !
        Off 5
        Wait .25
      Else
        MemOff PartsToPick
      EndIf
    EndIf
      Wait 0.1
  Loop
Fend

Function Robot2PickPlace
    Robot 2
    Do
     If MemSw(PartsToPick) = On Then
      If PF_QueLen(2) > 0 Then
       PF_AccessFeeder 1
       P0 = PF_QueGet(2)
       PF_QueRemove (2)
       Jump P0 /R
       On 5
       Wait .5
       Jump place ! D30; PF_ReleaseFeeder 1 !
       Off 5
       Wait .25
      Else
       MemOff PartsToPick
      EndIf
     EndIf
     Wait 0.1
    Loop
Fend

Function PF_Robot(PartID As Integer) As Integer
    Select PartID
     Case 1
      MemOn PartsToPick
      Wait MemSw(PartsToPick) = Off
     Case 2
      MemOn PartsToPick
      Wait MemSw(PartsToPick) = Off
    Send
    PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

PF_ActivePart

Switch the active part during multi-part operation. The PF_ActivePart Statement tells the system what part is currently desired. The system will vibrate or supply parts so that the PF_ActivePart is available for the robot to pick up.

Syntax

PF_ActivePart *part ID*

Parameters

part ID	Specify the part ID (integer number from 1 to 32).

Return Values

None

Description

In the case of multi-part operation, the first Part ID in PF_Start Statement is the initial Active Part. When a different part is desired, you can use this command to switch the Active Part. The system will feed (use the correct vibration settings, supply parts from the hopper etc…) for the Part ID that was specified in the PF_ActivePart Statement.
The ActivePart is normally set prior to exiting the PF_Robot callback so that the feeding action will be specific to the desired part.
If no Part Feeding operation has started, this command has no effect.
When it is not a multi-part operation (i.e., when only one ID is specified at the time of PF_Start execution), this command has no effect.
When you specify a Part ID that was not used in the multi-part operation, this command has no effect.
Cannot be executed from a virtual controller or command window.

Example

This example illustrates how to alternate between Parts 1 and 2 using the PF_ActivePart statement.

```
Function PF_Robot(PartID As Integer) As Integer
  Select PartID
    Case 1
       If PF_QueLen(1) > 0 Then
           MemOn PartsToPick1
           Wait MemSw(PartsToPick1) = Off
           PF_ActivePart 2  'Switch to Part 2
       Else
           PF_ActivePart 1  'Part 1 is still needed
       EndIf
     Case 2
       If PF_QueLen(2) > 0 Then
          MemOn PartsToPick2
          Wait MemSw(PartsToPick2) = Off
          PF_ActivePart 1 'Switch to Part 1
      Else
          PF_ActivePart 2 'Part 2 is still needed
     EndIf
  Send
    PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

PF_Backlight

Turns the built-in backlight on or off.

Syntax

PF_Backlight *feeder number* | *feeder name*, On | Off

Parameters

feeder number	Specify the feeder number (integer number from 1 to 4) as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.
On/Off	Set On/Off.

Return Values

None

Description

When the system is controlling vision, the backlight is automatically turned on and off as needed. Use this command to control the built-in backlight on status when the system is not controlling vision.

In the case of IF-80, the backlight turns off automatically after 30 seconds. (The time changes depend to the brightness) If the backlight turns off automatically, PF_Backlight Off command must be executed before the backlight can be turned on again.

Cannot be executed from a virtual controller or command window.

Example

```
PF_Backlight 1, On
```

PF_BacklightBrightness

Sets the brightness for the built-in backlight.

Syntax

PF_BacklightBrightness *feeder number* | *feeder name*, brightness

Parameters

feeder number	Specify the feeder number (integer number from 1 to 4) as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.
brightness	The percentage of brightness (integer number) from 0 to 100%.

Return Values

None

Description

Normally, the built-in backlight brightness to be used for a part is set in the Part Feeding Configuration dialog. If changes to brightness are required at runtime, you can use this command to change it.
Cannot be executed from a virtual controller or command window.

NOTE

☞ Although the percentage of brightness can be set from 0 to 100%, the minimum, physical intensity may be limited depending upon the feeder model. The physical brightness is limited because even illumination cannot be achieved below certain brightness settings.

Feeder Model	Brightness Setting	Physical Brightness Intensity
IF-80	No limitation	No limitation
IF-240	<=25%	25%
IF-380	<=1%	1%
IF-530	<=1%	1%

If you would like 0% backlight brightness, use the PF_Backlight statement to turn off thebacklight.

Example

```
PF_BacklightBrightness 1, 80
```

PF_Center

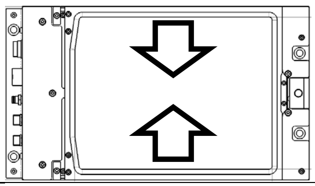
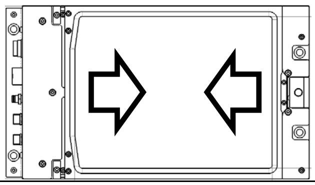
Perform the feeder centering operation.
Available with IF-240, 380 and 530. Not available on IF-80.

Syntax

PF_Center Part ID, Direction [, Duration]

Parameters

Part ID	Specifies the part ID (integer value 1 to 32).
Direction	Specifies the direction of the centering.

Direction	Value (defined by PartFeeding.inc)	Direction to move parts
Long axis	PF_CENTER_LONG_AXIS	
Short axis	PF_CENTER_SHORT_AXIS	

Duration	Specifies the operating time (integer value 1 to 30000 in milliseconds). When omitted, the value calculated by feeder calibration is used. When integer value is -1, the operation is same as omitted one.

Return Values

None

Description

Performs the centering operation of the IF series feeder.
PF_Center is used in the following cases:
- The parts need to be centered before separation
- Align the direction of the long and thin parts.

This command can be used directly within your functions. It can also be executed inside any part feeding callback functions when PF_Start has been executed.

It cannot be run under the following conditions:
- When executed in a user function
  The feeder specified in this command (specified by the Part ID) is used in the PartFeeding process (PF_Start command) (Error 7733)
- When executed in a callback function
  Specified part ID is not set for the PF_Start command (Error 7733)
- When executed from a virtual controller or command window. (Error 2582)

This command uses SyncLock for internal processing. For details, please refer to *1.3.6 Functions used by Part Feeding process* .

Example

```
PF_Center 1,1
```

PF_CenterByShift

Perform centering by shifting.

Syntax

PF_CenterByShift Part ID

Parameters

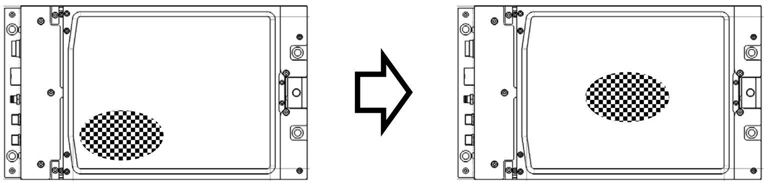
Part ID	Specifies the part ID (integer value 1 to 32).

Return values

None

Description

Perform centering by shifting.



This command is executed in following situation.
- Collect parts in the center before distributing
- Collect parts in the center when hand interferes with platform

When this command is executed, the Part Blob vision sequence is run. The combined center of gravity of all parts on the feeder is calculated. The combined center of gravity is shifted to the center of the feeder. If the combined center of gravity is already near the center of the feeder then no shifting will be performed.

It cannot be run under the following conditions.
- When executed in a user function:
  The feeder specified in this command (specified by the Part ID) is used in the PartFeeding process (PF_Start command) (Error 7733)
- When executed in a callback function:
  Specified part ID is not set for the PF_Start command (Error 7733)
- When executed from a virtual controller or command window. (Error 2582)

This command uses SyncLock for internal processing. For details, please refer to *1.3.6 Functions used in PartFeeding process* .

Example

```
PF_CenterByShift 1
```

PF_Flip

Perform the flip operation.

Syntax

PF_ Flip Part ID [, Duration]

Parameters

Part ID	Specifies the part ID (integer value 1 to 32).
Duration	Specifies the operating time (integer value 1 to 30000 in milliseconds). When omitted, the value calculated by feeder calibration is used. When integer value is -1, the operation is same as omitted one.

Return value

None

Description

Performs the Flip operation.
PF_Flip is used in the following cases.
- Dispersing parts (by long operation time)
- Re-orient the parts (by short operation time)

It cannot be run under the following conditions.
- When executed in a user function:
  The feeder specified in this command (specified by the Part ID) is used in the PartFeeding process (PF_Start command) (Error 7733)
- When executed in a callback function:
  Specified part ID is not set for the PF_Start command (Error 7733)
- When executed from a virtual controller or command window. (Error 2582)

This command uses SyncLock for internal processing. For details, please refer to *1.3.6 Functions used in PartFeeding process* .

Example

```
PF_Flip 1,500
```

PF_Info Function

Retrieves part properties.

Syntax

(1) PF_Info(*part ID*, *property ID*)
(2) PF_Info(*part name*, *property ID*)

Parameters

part ID	Specify the part ID (integer number 1 to 32).
part name	Specify the part name (character string).
property ID	Specify the property ID. Properties that can be retrieved are as follows.

Property ID	Details
PF_INFO_ID_FEEDER_CALIB_CORRECT_MAXNUM	Results of calibration for the optimum number of parts Used to calculate the number of parts to feed.
PF_INFO_ID_FEEDER_NO	Returns the feeder number used by the part.
PF_INFO_ID_ROBOT_NO	Returns the robot number used by the part.

Return Values

Returns the value of the property specified.

Description

Returns part property values. Use this value to customize system behavior within the callback function.
Cannot be executed from a virtual controller or command window.

Example

Refer to *4. Part Feeding Callback Functions PF_Control* for more details.

PF_InitLog

Enables Part Feeding log file output and specifies the destination path.
This should be executed before PF_Start.
To output log files, the Controller must be connected to the PC on which RC+ is installed.
For more information on Part Feeding log files, refer to *5. Part Feeding Log File*.

Syntax

PF_InitLog *part ID*, *output path*, *append*

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
output path	Specify the log file output destination path (file path on the PC + file name).
append	Set to True to append data when the specified output path exists. Set to False to overwrite the file.

Return Values

None

Description

Run this function from the same task as the task running PF_Start. A log will not be created when this function is run from a different task.
Nothing will occur when using this function if the Part Feeding process has not been started.
A log will not be created when running this function while the Controller is not connected to the PC. Note that this will not cause an error to occur.
An error will occur when running PF_Start and the path does not exist, or when the system fails to write data to the file. Note that an error will not occur when running this function.
Cannot be executed from a virtual controller or command window.
User vibration commands (PF_Center, PF_CenterByShift, PF_Flip, PF_Shift) will be logged when they are executed inside of part feeding callback functions and PF_Start has been executed.
If user vibration commands are executed when PF_Start is not running then the vibrations will not be logged.
This command uses Timer for internal processing. For details, please refer to *1.3.6 Functions used in PartFeeding process* .

Example

Refer to the usage example provided for PF_Start.

PF_IsStopRequested Function

Checks whether a Part Feeding process end request has been issued (whether PF_Stop has been executed).
This is normally used within a callback function.

Syntax

PF_IsStopRequested(*part ID)*

Parameters

part ID	Specify the part ID (integer number from 1 to 32).

Return Values

True is returned when PF_Stop has been called while the Part Feeding process is running.
False is returned in all other circumstances.

Description

This is used to determine whether a Part Feeding end request has been issued within a callback function. When running loop processing, etc., code the program so that this function is called for each loop, discontinuing the loop and ending the callback function when an end request is issued.

In the case of multi-part operation, you can specify one of the part IDs specified by PF_Start.
For example, if you run PF_Start 1, 2, 3, 4, then PF_Stop 2 is executed, you can check exit request with this function by using Part ID1,2,3 or 4.
Cannot be executed from a virtual controller or command window.

Example

Refer to *4. Part Feeding Callback Functions PF_Robot* for more details.

PF_Name$ Function

Returns the part name from a part ID.

Syntax

PF_Name$(*part ID*)

Parameters

part ID	Specify the part ID (integer number from 1 to 32).

Return Values

Returns the name of the specified part ID as a character string.

Description

This will return “” (blank string) if the specified part ID is invalid.
Cannot be executed from a virtual controller or command window.

Example

```
Print PF_Name$(1)
```

PF_Number Function

Returns a part ID from a part name.

Syntax

PF_Number(*part name*)

Parameters

Part name	Specify the part name (character string).

Return Values

Returns the part ID (integer number from 1 to 32) for the specified part name.

Description

Returns −1 if the corresponding part name does not exist.
If multiple parts with the same name exist, the part with the smallest ID will be retrieved.
Cannot be executed from a virtual controller or command window.

Example

```
Print "Part1 part number = ", PF_Number("Part1")
```

PF_Output

Turns the feeder output terminal on and off.
When two hoppers are connected to a feeder and parts are fed from each hopper, the ON time of each hopper can be specified.

Syntax

PF_Output *feeder number/feeder name*, *Output 1 duration, Output 2 duration*

Parameters

feeder number	Specify the feeder number (integer number from 1 to 4) as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.
Output 1 duration	Specifies the ON time (integer value). 1 - 30000[ms] can be specified. 0 means that the device remains OFF.
Output 2 duration	Specifies the ON time (integer value). 1 - 30000[ms] can be specified. 0 means that the device remains OFF.

Return Values

None

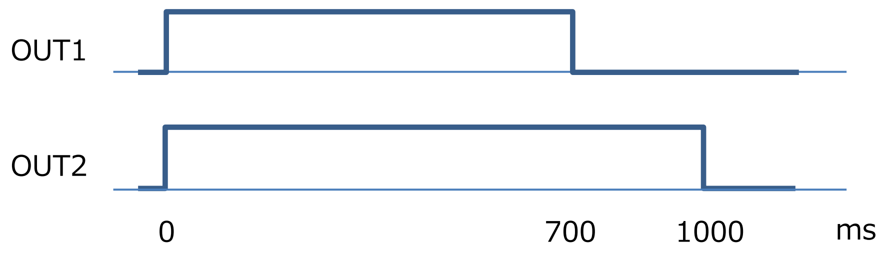
Description

Control will return immediately after this command is executed. If you wish to wait for the hopper operation to finish, use the Wait command immediately after executing this command.
If this command is executed during feeder vibration or hopper operation, those operations will be stopped and the operation specified by this command will be started.
If you want to execute feeder vibration and hopper operation at the same time, please use the PF_OutputOnOff command.
When the PF_Abort command or PF_Stop command is executed, it will turn OFF.
Cannot be executed from the virtual controller or command window.
If “Hopper installed” is not checked in [System Configuration] - [Controller] - [Parts Feeder], error 2584 (Purge Gate is not valid.) will occur.
Cannot be used with IF-80 (error 2589 “Action command call that the feeder cannot execute” occurred), please use PF_OutputOnOff command.

Example

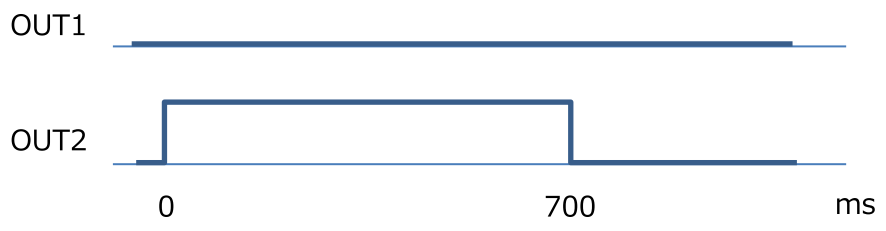
```
PF_Output 1, 700, 1000
```

The output from the output terminal will be as follows:



```
PF_Output 1, 0, 700
```

The output from the output terminal will be as follows:



PF_OutputOnOff

Turns the feeder output terminal on and off.
Set this command to On to supply parts from a hopper. Set the value to Off to stop supplying parts from a hopper.
The IF-80 has a built-in hopper. Optional hoppers are available for other feeder models.

Syntax

(1) PF_OutputONOFF *feeder number*/*feeder name*, On, *output number* [, *duration*] [, *wait*]
(2) PF_OutputONOFF *feeder number*/*feeder name*, Off

Parameters

feeder number	Specify the feeder number (integer number from 1 to 4) as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.
On/Off	Set On/Off.
output number	Specify the output destination. This can be specified when the command is set to On. 1: Out terminal 1 2: Out terminal 2
duration	Specify the duration to turn On. Specify an integer value between 1 - 30000 [ms]. Setting this to 0 or omitting this value will set this command to always remain On.
wait	Specify whether waiting the operation completed or not. Only for IF-80. 0: Does not wait for hopper operation to be completed. 1: Wait for hopper operation to be completed.

Return Values

None

Description

Control will return immediately after running this command (including the case where the wait setting is 0 for IF-80). Use the Wait command as shown in the example to wait for the hopper to complete its actions.
Output terminals 1 and 2 cannot both be turned On at the same time. For example, turning terminal 1 On will turn terminal 2 Off.
When set to Off, both terminals 1 and 2 will be Off.
Output will turn Off when the PF_Abort command or the PF_Stop command is run.
If “Hopper installed” is not checked in [System Configuration]-[Controller]-[Parts Feeder], error 2584 (Purge Gate is not valid.) will occur.
Cannot be executed from a virtual controller or command window.
When a purge gate is used with the IF-240, Out pin 2 is used to control the purge gate. Therefore, setting the output number to 2 will not change the output of Out 2, although it can be executed.

Example

Refer to *4. Part Feeding Callback Functions PF_Control* for more details.

PF_PurgeGate

Controls the opening and closing of the purge gate (optional).

Syntax

PF_PurgeGate *feeder number*/*feeder name*, On/Off

Parameters

feeder number	Specify the feeder number (integer number from 1 to 4) as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.
On/Off	Set On(1) / Off(0). When On is specified, the purge gate opens. When Off is specified, the purge gate closes.

Return Values

None

Description

Control will be returned immediately after this command is executed.
If you wish to wait for the hopper operation to finish, please refer to the Example of PF_PurgeGateStatus.
If “Purge gate installed” is not checked in [System Configuration] - [Controller] - [Parts Feeder], error 2593 (Feeder purge output is not valid) will occur.
This function cannot be executed from the virtual controller or command window.
PF_Stop, Abort, emergency stop and safeguard open does not immediately stop the purge gate open and close.

Example 1

Wait for the purge gate to open, and then perform the next operation.

```
PF_PurgeGate 1, On
Wait 5.0

' Next action
```

Example 2

Wait for the purge gate to close, and then perform the next operation.
Note: If the purge gate becomes overloaded during the closing operation (e.g. a part gets caught), it will automatically switch to the opening operation. The following program is an example of using PF_PurgeGateStatus to detect this.

```
Integer looplim
looplim = 50
PF_PurgeGate 1, Off
Do While PF_PurgeGateStatus(1) = On And looplim > 0
  Wait 0.1
  looplim = looplim -1
Loop
If looplim <= 0 Then
   ' Error Purge gate cannot close
EndIf

' Next action
```

PF_PurgeGateStatus Function

Get the close sensor status of the purge gate (optional).

Syntax

PF_PurgeGateStatus(*feeder number*/*feeder name*)

Parameters

feeder number	Specify the feeder number (integer number from 1 to 4) as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.

Return Values

False if the close sensor is On (the gate is closed).
True if the close sensor is Off (the gate is not closed).

Description

The purge gate has a close sensor, but no open sensor. In other words, when the purge gate is even slightly open, this function returns True.
Control will be returned immediately after this command is executed.
If "Purge gate installed" is not checked in [System Configuration] - [Controller] - [Parts Feeder], error 2593 (Feeder purge output is not valid) will occur.
This function cannot be executed from the virtual controller or command window.

Example

Refer to the Example of PF_PurgeGate.

PF_Purge Function

Purging (ejecting the parts on the feeder) is performed.
When the purge operation is performed (without success or not), the coordinate queue of the part specified by the part ID is cleared.

Syntax

PF_Purge *partID, type[, duration[, remain[, retry]]]*

Parameters

partID Specify the part ID (integer number from 1 to 32).

type Specify the process.

Value (defined by PartFeeding.inc)	Details
PF_PURGETYPE_NOVISION	Without vision feedback. Not count the number of remaining parts.
PF_PURGETYPE_VISION	With vision feedback. Counts the number of remaining parts using vision.

duration Specifies the purge time (in milliseconds)
This can be omitted. When omitted, the value set in “2.4.12 Purge - Test & Adjust” is used.
When integer value is -1, the operation is same as omitted one.

remain Specifies the number of parts to retry purging operation.
This can be omitted.
When the operation type is PF_PURGETYPE_NOVISION, operation is not going to be changed regardless of settings. When the operation type is PF_PURGETYPE_NOVISION, retry is operated until remain is 0 if remain omitted.

retry Specifies the maximum number of retries to achieve that *remain* number of parts.
This can be omitted.
When the operation type is PF_PURGETYPE_NOVISION, operation is not going to be changed regardless of settings. When the operation type is PF_PURGETYPE_VISION, if the number of retries is omitted then purging will be performed until the “remain” number of parts are remaining on the platform or all parts have been removed from the platform (in the case that “remain” and “retry” have been omitted).

Return Values

Returns True when purging has completed successfully. Returns False if the number of parts remaining on the feeder exceeds the *remain* value within the specified number of *retries*.

Description

The IF-80 has an available purge option (which consists of a special platform and a removable purge box).

The IF-240, IF-380 and IF-530 have an available Purge Gate option. The Purge Gate installation is enabled from [Setup] – [System Configuration] – [Controller] – [Part Feeding] – [Feeder]. Each part that uses a feeder with the Purge Gate installed, can either use or not use the Purge Gate. If the Purge Gate is used for the part then the opening, the closing and the sensor detection will be automatically handled by the PF_Purge statement.
The customer can also make their own purge door mechanism. In that case, the PF_Purge statement will simply shift parts off of the feeder platform.
When the *type* is “PF_PURGETYPE_VISION”, the Part Blob Sequence is used to approximate the number of parts on the platform.
To enable/disable purging and to determine the direction of the purge, refer to *2.2.11 Purge*.
The IF-80 must be calibrated for purging. Set up in *2.4.11 Purge - Automatic Calibration (for IF-80)*.
When the purge is disabled and this function is called, an error occurs and the PF_Startus callback function will be called with PF_STATUS_PURGENOTENABLED.
Cannot be executed from a virtual controller or command window.

In EPSON RC+ 7.5.0, the *remain* and *retry* parameters were not optional even if the *type* = 1. For EPSON RC+ 7.5.0, specify a value of “0” for *remain* and *retry* when *type* =1.

Examples

Example 1:

Purge Part #1 using vision feedback. Each purge duration is 1500 msec. 3 parts can remain on the platform. 5 retries will be attempted.

```
Boolean purgeStatus

purgeStatus = PF_Purge(1, PF_PURGETYPE_NOVISION, 1500, 3, 5)
Print purgeStatus
```

Example 2:

Purge Part #1 without vision feedback. Purge for 2000 msec.

```
PF_Purge 1, PF_PURGETYPE_VISION, 2000
```

PF_QtyAdjHopperTime Function

Returns the estimated amount of hopper operation time that is required to supply the optimal number of parts.

Syntax

PF_QtyAdjHopperTime(*partID*, *SupplyQty*, *SupplyTime)*

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
SupplyQty	Specify the quantity of parts (number of parts) that are supplied in the *SupplyTime.*
SupplyTime	Specify the hopper operating time in milliseconds that is required to supply the quantity of parts (number of parts) specified by *SupplyQty*.

Return Values

Returns the estimated amount of hopper operation time (ms).
The return value will be between 1 - 30000 [ms].
In the case of error, the return value will be 1.

Description

PF_QtyAdjHopperTime uses vision, to calculate the hopper operation time required to ensure that the platform has the optimal number of parts (determined by feeder calibration). The return value is used as the duration parameter for the PF_OutputOnOff statement.
The SupplyQty and SupplyTime are determined by the developer (using the Test Hopper button on the Part Feeding Supply page). These values indicate how many parts are supplied in a given amount of time.

In the case of multi-part operation, the calculated operating time assumes that an equal quantity of each part type is optimal. If PF_QtyAdjHopperTime is executed while parts are running on the feeder, then the Part Blob vision sequence (for the supplied PartID) is run as well as each individual Part Sequence (for each part in the PF_Start grouping). If PF_QtyAdjHopperTime is executed when no parts are running on the feeder, then only the Part Blob Sequence (for the supplied PartID) is used.

This command cannot be executed from a virtual controller or command window.
This command cannot be used with mobile cameras.
If the User processes vision (i.e., the PF_Vision callback function is used) for any part running on the feeder then PF_QtyAdjHopperTime will only use the Part Blob Sequence to determine the hopper operation time for the part specified by the Part ID parameter.
Please note that if “Supply parts during pick and place” is selected (see “2.3.5 Part Supply”) and this command is executed while the robot arm is in the camera’s field of view, the arm will be recognized as a part and an incorrect hopper operation time may be returned.

Example

This example uses the PF_Control callback to turn on the optional hopper for the estimated amount of time that is necessary to supply the optimal quantity of parts. The hopper is adjusted to deliver 10 parts per second (1000ms).

```
Function PF_Control(PartID As Integer, Control As Integer) As Integer
  Integer hopperOnTime

  Select Control
    'Request for part supply (add up to optimum number)
    Case PF_CONTROL_SUPPLY
      hopperOnTime = PF_QtyAdjHopperTime(PartID, 10, 1000)
      PF_OutputONOFF 1, On, 1, hopperOnTime
      Wait hopperOnTime / 1000
  Send

  PF_Control = PF_CALLBACK_SUCCESS
Fend
```

PF_QueAdd

Adds data (point data, part orientation, user data) to the parts coordinates queue.

Syntax

PF_QueAdd *part ID*, *point data* [, *part orient* [, *user data*]]

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
point data	Specify point data.
part orient	Optional. Integer expression used to register the part orientation along with the point data. PF_PARTORIENT_FRONT=1 The part faces up. PF_PARTORIENT_BACK=2 The part faces down.
user data	Optional. Real expression used to register the user data along with the point data.

Return Values

None

Description

This is used to register data (point data, part orientation, user data) to the parts coordinates queue.
Under normal circumstances, PF_QueAdd is not needed as the parts coordinates queue is generated automatically within the Part Feeding process. This is used when using unique vision processes (using the PF_Vision callback function).
Data is added to the end of the parts coordinates queue for the specified part ID.
However, data will be registered according to the sorting method set should one be applied using PF_QueSort.
Up to 1,000 data items can be retained in a parts coordinates queue.

Example

Refer to *4. Part Feeding Callback Functions PF_Vision* for more details.

PF_QueAutoRemove

Configures the auto remove function for the parts coordinates queue specified.

Syntax

PF_QueAutoRemove *part ID*, {True | False}

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
True | False	False: Disables the auto remove function (default). True: Enables the auto remove function.

Return Values

None

Description

Configures the auto remove function for the parts coordinates queue.
When the auto remove function is enabled, point data is automatically deleted from the parts coordinates queue when using PF_QueGet to retrieve point data from the parts coordinates queue.
Point data is not deleted when the auto remove function is disabled. Use PF_QueRemove to delete point data.
The auto remove function can be enabled/disabled for each part ID.

Example

```
PF_QueAutoRemove 1, True
```

PF_QueAutoRemove Function

Returns the status of the auto remove function set for the parts coordinates queue.

Syntax

PF_QueAutoRemove (*part ID*)

Parameters

part ID	Specify the part ID (integer number from 1 to 32).

Return Values

This is returned as "True" if the parts coordinates queue auto remove function is enabled, and "False" if it is disabled.

Description

Refer to the description for PF_QueAutoRemove.

Example

```
Boolean autoremove
autoremove = PF_QueAutoRemove (1)
```

PF_QueGet Function

Returns point data from the parts coordinates queue.

Syntax

PF_QueGet (*part ID* [, *index*])

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
index	Specify the index of the queue data to retrieve as an integer number. The initial index value is 0. This can be omitted.

Return Values

Returns the point data.

Description

PF_QueGet retrieves point data from the parts coordinates queue. If the index is omitted, this function will retrieve the initial queue data entry. If an index is specified, point data for the index specified will be returned.
Point data is deleted by executing PF_QueGet when PF_QueAutoRemove is used to enable the auto remove function for queue data.
Point data is not deleted when the auto remove function is disabled. Use PF_QueRemove to delete point data.

Example

Refer to *4. Part Feeding Callback Functions PF_Robot* for more details.

PF_QueLen Function

Returns the amount of parts coordinates queue data registered to the parts coordinates queue.

Syntax

PF_QueLen (*part ID*)

Parameters

part ID	Specify the part ID (integer number from 1 to 32).

Return Values

Returns the number of valid parts coordinates queue data entries registered as an integer number.

Description

Returns the current number of registered entries of parts coordinates queue data.
This can also be used as a Wait command parameter.

Example

Refer to *4. Part Feeding Callback Functions PF_Robot* for more details.

PF_QueList

Displays a list of parts coordinates queue data (point data) for the parts coordinates queue specified.

Syntax

PF_QueList *part ID*, [*display count*]

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
display count	Specify the number of displayed data items as an integer number. This can be omitted. All queue data will be displayed when this parameter is omitted.

Description

This can only be used on the Command window.

Example

Command window usage example

```
> PF_QueList 1
Queue 0    = XY(     1.000,    1.000,    0.000,    0.000 )  /R /0  ( 1 ) (   0.000)
Queue 1    = XY(     3.000,    1.000,    0.000,    0.000 )  /R /0  ( 1 ) (   2.000)
Queue 2    = XY(     4.000,    1.000,    0.000,    0.000 )  /R /0  ( 1 ) (   3.000)
Queue 3    = XY(     5.000,    1.000,    0.000,    0.000 )  /R /0  ( 2 ) (   4.000)
Queue 4    = XY(     6.000,    1.000,    0.000,    0.000 )  /R /0  ( 2 ) (   5.000)
```

PF_QuePartOrient

Resets and displays the part orientation (integer number) registered to the parts coordinates queue.

Syntax

PF_QuePartOrient *part ID* [, *index* [, *part orient*]]

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
index	Integer expression that represents the index of the parts coordinates queue data. (the beginning of the index number is 0). Optional when executing from the command window.
part orient	Integer expression that represents the part orientation to be set again. This can be omitted when executed from the command window. If omitted, the current part orientation (integer expression) is displayed. PF_PARTORIENT_FRONT=1 The part faces up. PF_PARTORIENT_BACK=2 The part faces down.

Description

Resets and displays the part orientation currently registered to the parts coordinates queue.

If the Sort method is specified by PF_Sort, the order of the parts coordinates queue data is changed according to the specified Sort method.

QUE_SORT_POS_PARTORIENT : Part orientation (integer expression) ascending order
QUE_SORT_INV_PARTORIENT : Part orientation (integer expression) descending order

See Also

PF_QuePartOrient Function

Example

```
PF_QuePartOrient 1, 1, PF_PARTORIENT_FRONT
```

PF_QuePartOrient Function

Returns the part orientation (integer value) registered to the parts coordinates queue.

Syntax

PF_QuePartOrient (*part ID* [, *index*])

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
index	Optional. Integer expression that represents the index of the parts coordinates queue data. (the first index number is 0).

Return Values

Returns part orient (integer value).

Description

PF_QuePartOrient retrieves the part orientation from the parts coordinates queue. If the index is omitted, this function will retrieve the initial queue data entry. If an index is specified, part orientation for the index specified will be returned.

Example

Refer to *4. Part Feeding Callback Functions PF_Robot* for more details.

PF_QueRemove

Deletes parts coordinates queue data (point data) from the parts coordinates queue specified.

Syntax

PF_QueRemove *part ID* [, *index* | All]

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
index	Specify the index of the parts coordinates queue data to delete as an integer number. The initial index value is 0. This can be omitted. Specify All to delete all data items.

Return Values

None

Description

Deletes data (point data) from the parts coordinates queue.
This is used to delete data used from the parts coordinates queue.

Example

Refer to *4. Part Feeding Callback Functions PF_Robot* for more details.

PF_QueSort

Sets and displays the sorting method applied to the parts coordinates queue specified.

Syntax

PF_QueSort *part ID* [,*sort method*]

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
sort method	Set the sorting method using integer numbers (0 to 8), or with the constants described below. This can be omitted when running from the Command window. When omitted, the current sorting method in use will be displayed.

Constant	Value	Description
QUE_SORT_NONE	0	Do not sort (use the parts coordinates queue registration order)
QUE_SORT_POS_X	1	Sort in ascending order by the X coordinate
QUE_SORT_INV_X	2	Sort in descending order by the X coordinate
QUE_SORT_POS_Y	3	Sort in ascending order by the Y coordinate
QUE_SORT_INV_Y	4	Sort in descending order by the Y coordinate
QUE_SORT_POS_USER	5	Sort in ascending order by the user data.
QUE_SORT_INV_USER	6	Sort in descending order by the user data.
QUE_SORT_POS_PARTORIENT	7	Sort in ascending order by the part orientation.
QUE_SORT_INV_PARTORIENT	8	Sort in descending order by the part orientation.

Return Values

None

Description

Sets the sorting method applied to the parts coordinates queue. Point data added using PF_QueAdd will be registered to the parts coordinates queue according to the sorting method set.
Therefore, you must run PF_QueSort before using PF_QueAdd. Note that data will not be resorted if this function is used after the data has already been added.

Example

```
PF_QueSort 1, QUE_SORT_POS_X
```

PF_QueSort Function

Returns the sorting method set to the parts coordinates queue specified.

Syntax

PF_QueSort (*part ID*)

Parameters

part ID	Specify the part ID (integer number 1 to 32).

Return Values

Returns the sorting method set to the parts coordinates queue as an integer number.

Value	Description
0	Do not sort (use the parts coordinates queue registration order)
1	Sort in ascending order by the X coordinate
2	Sort in descending order by the X coordinate
3	Sort in ascending order by the Y coordinate
4	Sort in descending order by the Y coordinate
5	Sort in ascending order by the user data.
6	Sort in descending order by the user data.
7	Sort in ascending order by the part orientation.
8	Sort in descending order by the part orientation.

Example

```
Integer quesort

quesort = PF_QueSort(1)
```

PF_QueUserData

Resets and displays the user data (real number) registered to the parts coordinates queue.

Syntax

PF_QueUserData *part ID* [, *index* [, *user data*]]

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
index	Integer expression that represents the index of the parts coordinates queue data. (the beginning of the index number is 0). Optional when executing from the command window.
user data	Real expression that represents the user data to be set again. This can be omitted when executed from the command window. If omitted, the current user data (real expression) is displayed.

Description

Resets and displays the user data currently registered to the parts coordinates queue.

If the Sort method is specified by PF_Sort, the order of the parts coordinates queue data is changed according to the specified Sort method.

QUE_SORT_POS_USER : User data (real expression) ascending order
QUE_SORT_INV_USER : User data (real expression) descending order

See Also

PF_QueUserData Function

Example

```
Real r
r = 0.1
PF_QueUserData 1, 1, r
```

PF_QueUserData Function

Returns the user data (real value) registered to the parts coordinates queue

Syntax

PF_QueUserData (*part ID* [, *index*])

Parameters

part ID	Specify the part ID (integer number from 1 to 32).
index	Optional. Integer expression that represents the index of the parts coordinates queue data. (the first index number is 0).

Return Values

Returns user data (real value).

Description

PF_QueUserData retrieves the user data from the parts coordinates queue. If the index is omitted, this function will retrieve the initial queue data entry. If an index is specified, part orientation for the specified index will be returned.

Example

```
Real r
r = PF_QueUserData(1, 1)
```

PF_ReleaseFeeder

Release the lock acquired by PF_AccessFeeder. PF_AccessFeeder & PF_ReleaseFeeder lock and unlock access to a feeder to prevent potential collisions on a multi-robot / one feeder system. These commands are required when two robots are sharing the same feeder at the same time.

Example:
Task 1 executes PF_AccessFeeder 1, then Task 1 continues.
Task 2 executes PF_AccessFeeder 1, then Task 2 is paused.
Task 1 executes PF_ReleaseFeeder 1, then Task 2 is resumed.

This command can be written in parallel processing (!...!) in motion commands.

Syntax

Format 1: F_ReleaseFeeder *feeder number/ feeder name*
Format 2: *MotionCommand* ! *[Dn;]* PF_ReleaseFeeder *feeder number/ feeder name* !

Parameters

feeder number	Specify the feeder number as an expression or a numerical value.
feeder name	Specify the feeder name as a character string.
MotionCommand	Any of Arc, Arc3, Go, Jump, Jump3, Jump3CP, Move, BGo, BMove, TGo, and TMove.
Dn	Specify where in the path of the robot's movement to start parallel processing in %. (See *EPSON RC+ 7.0 SPEL+ Language Reference ! Parallel Processing*)

Return Values

None

Description

The lock can only be unlocked within the same task.
Cannot be executed from a virtual controller or command window.

Example

This is an example of getting the last part on the feeder and when 50% of the path to the place position is reached, control is returned to PF_Start to start vision imaging and feeder operation.

```
Function main
  Motor On
  PF_Start(1)
Fend

Function PF_Robot(partID As Integer)
  Xqt Task_PF_Robot(partID)
  Wait 1.0
  AccessFeeder 1
  ReleaseFeeder 1
  PF_Robot = PF_SUCCESS
Fend

Function Task_PF_Robot(partID As Integer)
  AccessFeeder 1

  Integer i
  For i = 1 to numToPick
    pick = PF_GetQue(PartID)
    PF_QueRemove(PartID)
    Jump pick
    On gripperOutput
    Wait .1
    If i < numToPick And PF_QueLen(PartID) > 0 Then
      Jump place
    Else
   ' Last part, so release the feeder at 50%
      Jump place ! D50; PF_ReleaseFeeder 1!
    EndIf
    Off gripperOutput
    Wait .1
  Next i
Fend
```

PF_Shift

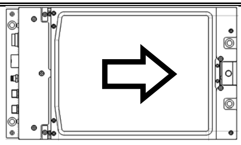
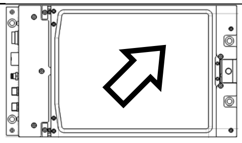
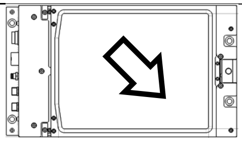
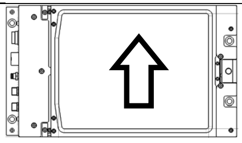
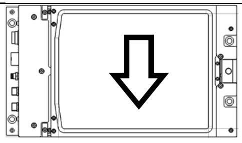
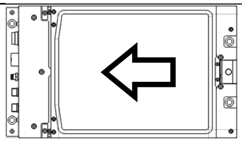
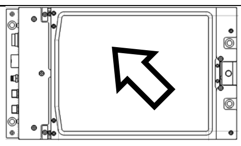
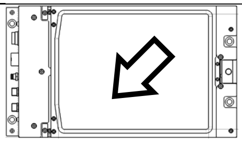
Perform a shift operation.

Syntax

PF_ Shift Part ID, Direction, [, Duration]

Parameters

Part ID	Specifies the part ID (integer value 1 to 32).
Direction	Specifies the direction of shift.

Direction	Value (defined by PartFeeding.inc)	Work
Forward	PF_SHIFT_FORWARD	
Forward Left	PF_SHIFT_FORWARD_LEFT	
Forward Right	PF_SHIFT_FORWARD_RIGHT	
Left	PF_SHIFT_LEFT	
Right	PF_SHIFT_RIGHT	
Backward	PF_SHIFT_BACKWARD	
Backward Left	PF_SHIFT_BACKWARD_LEFT	
Backward Right	PF_SHIFT_BACKWARD_RIGHT	

Duration Specifies the shift operating time. (in milliseconds)
This can be omitted. When omitted, the value in the Calibrate & Test dialog (See *2.4.9 Shift - Test & Adjust (Simple),* or *2.4.10 Shift - Test & Adjust (Advanced)*) will be used.
When integer value is -1, the operation is same as omitted one.

Return value

None

Description

Performs the shifting operation of IF series.

Shifting operation is used in the following cases.

- Move the part closer to the place position to increase the pick and place efficiency of the robot.
- When a using custom platform, dropping parts into grooves or holes to make them stand up or align them.

It cannot be run under the following conditions.

- When executed in a user function, the feeder specified in this command (specified by the Part ID) is used in the PartFeeding process (PF_Start command) (Error 7733)
- When executed in a callback function, specified part ID is not set for the PF_Start command (Error 7733)
- When executed from a virtual controller or command window. (Error 2582)

This command uses SyncLock for internal processing. For details, please refer to *1.3.6 Functions used in PartFeeding process* .

Use cases

```
PF_Shift 1, PF_SHIFT_FORWARD, 500
```

PF_Start

Starts the Part Feeding process for a specified part.

Syntax

PF_Start *part ID1[, part ID2 [, part ID3 [, part ID4]]]*

Parameters

part ID	Specify the part ID (integer number from 1 to 32). When taking a variable as an argument and do not specify the part ID, set the value to 0.

Return Values

None

Description

Perform the following before starting PF_Start (refer to the usage example).
- Select the robot in use
- Configure robot settings (Power, Speed, Accel, etc.)
- Turn the motors on
- Run PF_InitLog when outputting a log

Running PF_Start generates a new task (task number 32) and the part feeding process is started.
The control returns to the caller without waiting the end of the part feeding process.

In the following conditions, an error occurs and the Status callback function will be run.
The Part Feeding process will not start.

Condition	Status callback function Status parameter value
The part ID is not exist Try to start a multi-part operation with parts from different feeders Duplicate part IDs are set	PF_STATUS_BAD_ID
Part parameter settings are invalid (Enabled check box not selected, etc.)	PF_STATUS_BAD_PARAMETER
Feeder calibration not complete	PF_STATUS_CAL_NOT_COMPLETE
The part is disabled	PF_STATUS_PARTNOTENABLED
The feeder is in use	PF_STATUS_FEEDERINUSE_ERROR
An system error occurred	PF_STATUS_ERROR

Multi-feeder operation :

The Part Feeding process can be run simultaneously on parts that belong to different feeders.
For example, if part 1 belongs to feeder 1 and part 2 belongs to feeder 2, you can run PF_Start 1 at first, then PF_Start 2 can be executed.
PF_Start creates a task for each feeder. The Task Number that is used depends upon the Feeder Number.

Feeder Number	Task
1	32
2	31
3	30
4	29

Each callback function (PF_Robot, PF_Control, PF_Status. PF_Vision, PF_MobileCam) is executed in the same task as PF_Start creates.

For T/VT series controller, up to two feeders can be controlled at the same time. If using three or more feeders, “the error 7731: The maximum number of simultaneous feeders for the controller type has been exceeded.” occurs.

Multi-part operation:

When you want to run the multi-part operation, you should specify multiple part IDs as arguments. Up to four part IDs can be specified.
In this case, the feeder vibration is performed the Part ID (active part) specified by the first argument of PF_Start. You can use the PF_ActivePart command to switch the active part.
Only parts that belong to the same feeder can be specified in a multi-part operation. If a part from a different feeder is specified and PF_Start is executed, an error occurs and the PF_Status callback function is called with PF_STATUS_BAD_ID.
Up to two robots can share a single feeder at the same time. If you attempt to PF_Start a grouping of parts (i.e., PF_Start 1, 2, 5) and the parts are assigned to more than 2 robots, an error will occur.

Other notes:

While the Part Feeding process is running on a feeder, another Part feeding process cannot be executed for the same feeder.
For example, if part 1 belongs to feeder 1 and part 2 belongs to feeder 1, run PF_Start 1 and then PF_Start 2, an error occurs and the and the PF_Status callback function is called with PF_STATUS_FEEDERINUSE_ERROR. At this time PF_Start 1 continues processing with no error.
PF_Start must be executed from a normal task. If PF_Start is executed from the background task, an error will occur.
Cannot be executed from a virtual controller or command window.

Example

```
Robot 1
Motor On
Power High
Speed 100
Accel 100, 100
LimZ -80.0

PF_InitLog 1, "C:\log.csv", True
PF_Start 1
```

PF_Stop

Issues a Part Feeding process end request.
This will wait for running callback functions to finish.
Once complete, the PF_CycleStop callback function will run and the process will stop.

Syntax

PF_Stop *part ID*

Parameters

part ID	Specify the part ID (integer number from 1 to 32).

Return Values

None

Description

Stops the Part Feeding process for the specified part.
Unlike the PF_Abort command, PF_Stop will wait for running callback functions to finish.
Once callback functions are complete, the PF_CycleStop callback function will run.
Nothing will occur when using this function when the Part Feeding process has not been started.
When using PF_Stop in a multi-part operation, any of the partID that specified with PF_Start can stop the Part Feeding process.
You cannot execute PF_Start immediately after executing PF_Stop. If PF_Start is executed before the PF_CycleStop callback function has completed, a PF_STATUS_FEEDERINUSE_ERROR will occur.
This is because you are trying to run a new part on the feeder before the current part has completed.
To correct this situation, add code like to the following.

```
PF_Stop 1        'For this example, Part 1 is running on Feeder 1 which uses task 32
TaskWait 32      'Wait for the current part to finish
PF_Start 2       'Now you can start a new part
```

Cannot be executed from a virtual controller or command window.

Example

```
PF_Stop 1
```

4. Part Feeding Callback Functions

Callback functions are SPEL functions that are automatically called from the PF_Start command process when predetermined conditions are met.

Program files (PartFeeding.prg, PartFeeding.inc) that include templates for each callback function will be added to the project when registering the first part as a new entry. Users describe the required commands in SPEL based on the specifications of the user's system setup in use.

Function	Description / Application
PF_Robot	Syntax for robot behavior (pick, place).
PF_Control	Syntax for hopper, user lighting operations.
PF_Status	Syntax for error processing.
PF_MobileCam	Syntax for moving/retracting the mobile camera to the image capture position.
PF_Vision	Syntax for unique vision processing (e.g.: determining parts based on multiple vision sequence results).
PF_Feeder	Syntax for unique feeder operation (e.g. Processing parts by platform customer manufactured.
PF_CycleStop	Syntax for processing after PF_Stop has been executed.

4.1 Common Items

The robot number inside callback functions is the robot number specified in *2.2.2 General.*

Callback functions are run as normal tasks while the Part Feeding process is running.
The task number used starts at 32 and decrements for each additional feeder.

Do not delete PartFeeding.prg and PartFeeding.inc. Further, do not delete or comment out callback functions not in use. Doing so will result in a build error.

Do not call callback functions from user code while the Part Feeding process is running. Doing so may have unexpected consequences. Callback functions can be executed individually for testing purposes.

PF_Robot

Describe robot behavior for retrieving parts from the feeder and placing them in a designated location.
Make sure this function is written.

Syntax

```
Function PF_Robot(part ID As Integer) As Integer
  ' Robot movement
Fend
```

Parameters

part ID	The part ID (integer number from 1 to 32) goes here. When multi-part operation, the active part goes here.

Return Values

To control the PartFeeding process after the PF_Robot function exits, set the following values.
(Each constant is defined in PartFeeding.inc)

PF_CALLBACK_SUCCESS
Normally, you should specify this value.
When there is no data in the part coordinate queue (in the case of multi-part operation, there is no data of the active part), the PartFeeding process continues.
When there is data left in the queue (in case of multipart operation, there is some data of the active part), the PF_Robot callback function is restarted with no changing the contents of the part coordinate queue.

PF_CALLBACK_RESTART
The PartFeeding process continues and regenerates the part coordinate queue, regardless of data remaining or not of the queue. This can be used when you want to get the part coordinate with each pick for picking with high accuracy.

PF_CALLBACK_RESTART_ACTIVEPART
The PartFeeding process continues and regenerates the part coordinate queue for the active part only, regardless of data remaining or not of the queue. This can be useful to speed up of the vision process by omitting other than the active parts.

User defined error numbers (8000 - 8999)
This is set when you want to invoke the PF_Status callback function to handle errors.
The set value is passed as an argument to the PF_Status callback function.

Description

This function should contain the following operations:

1. Get the coordinates of the part from the coordinate queue (using PF_QueGet function).
2. Move the robot to the part coordinates on the feeder
3. Grasping a part by operating a hand, etc.
4. Move the robot to the part placement coordinates.
5. Releasing a part by operating a hand, etc.
6. Delete a coordinate queue (using the PF_QueRemove command)

Normally steps 1 -6 (listed above) are executed in a loop until all the coordinate queue data has been removed.
The list of coordinates for parts detected in the most recent vision process are added to the coordinates queue. Coordinates are defined in the local coordinate system. In the case of multi-part operation, a coordinate queue is generated for each part ID.
The Robot # used in the PF_Robot callback function needs to match the robot # that was selected for the part. In the case of multi-robot, you can use Robot statements to switch the robot # (refer to *EPSON RC+ 7.0 SPEL+ Language Reference Robot*).

CAUTION	■ To acquire part ID using the parameters is recommended. ■ Be careful not to change to wrong robot number when changing the robot number in the callback function in a multi-robot system.

Program Example

The following is an example of a simple program written to use a vacuum end effector to process all parts.
A vacuum sensor is used to monitor suction when picking up parts.
Three attempts are made when suction cannot be applied. User error code 8001 is returned as a return value if suction still cannot be applied.

```
'
' ** IO Label (Output) **
' O_VacOn   Suction
' O_VacRelease   Release
'
' ** IO Label (Input) **
' I_VacSensed     Suction sensor
'
' ** Point **
' PlacePos    Parts place position
'
' ** User Error **
' 8001 Suction timeout occurred
'
Function PF_Robot(partID As Integer) As Integer

   Byte retry

   ' Process entire coordinates queue, or loop until a stop request is issued
   Do While PF_QueLen(partID) > 0 And PF_IsStopRequested(partID) = False

      ' Move the robot to pick up position
      Jump PF_QueGet(partID)

      ' Vacuum ON & suction check (retry three times)
      On O_Adsorb
      retry = 3
      Do While retry > 0
         Wait Sw(I_VacSensed) = On, 0.5
         If TW = True Then
            ' If timed out, stop and reapply suction
            Off O_VacOn
            Wait 0.1
            On O_VacOn
         EndIf
      Loop
      If TW = True Then
         ' Terminate with error as suction still cannot be applied after reattempts
         ' Perform error processing according to Status callback function
         PF_Robot = 8001
         Exit Function
      EndIf

      ' Move the robot to place position
      Jump PlacePos

      ' Vacuum OFF & vacuum break
      Off O_VacOn
      On O_VacRelease
      Wait 0.1
      Off O_VacRelease

      ' Delete one data entry in the coordinates queue
      PF_QueRemove partID

   Loop

   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

PF_Control

PF_Control is called when the system needs the hopper or user lighting to turn on or off. This callback function is used to control the following devices installed in the system:

- Hopper
- User lighting

When using user lighting, select "Custom front light" in EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] - [Lighting].

Syntax

```
Function PF_Control(part ID As Integer, Control As Integer) As Integer
  ' (Hopper operations)
  ' (External lighting operations)
Fend
```

Parameters

part ID	The part ID (integer number from 1 to 32) goes here. When multi-part operation, the active part goes here.
Control	The type of operation (integer number) goes here.

Operation	Value (defined in PartFeeding.inc)
Hopper operation (when the feeder contains no parts)	PF_CONTROL_SUPPLY_FIRST
Hopper operation (when the feeder contains parts, and parts can be added)	PF_CONTROL_SUPPLY
User lighting On	PF_CONTROL_LIGHT_ON
User lighting Off	PF_CONTROL_LIGHT_OFF

Return Values

Under normal circumstances, set the PF_CALLBACK_SUCCESS constant (defined in PartFeeding.inc). When an error occurs, set a user-defined error number (8000 - 8999). This value is passed back as a PF_Status callback function parameter.

Description

Use the Select...Send descriptor text to describe processes by device.

Hopper operation (when the feeder contains no parts)
There are no parts in the feeder. Move the hopper to supply parts to the feeder. You can optimize robot movement by setting the number of parts supplied to the value obtained when calibrating for the optimum number of parts (retrieved with the PF_Info command).

Hopper operation (when the feeder contains parts, and parts can be added)
There are parts in the feeder, and more parts can be added. Adding parts at this timing increases the number of parts that can be detected with the vision system, and improves robot operation efficiency. As an example, assume the number of parts supplied is equal to the number of parts removed by the robot.

User lighting On
This indicates the timing to turn user lighting on for vision imaging. Operate the user lighting to turn it on.

User lighting Off
This indicates the timing to turn user lighting off when vision imaging has ended. Operate the user lighting to turn it off.
Describe the following processes in user-prepared code. Have these run before PF_Start.

- Connect to the hopper, and adjust settings, etc.
- Connect to the external lighting, and adjust settings (brightness), etc.

Program Example

The following program example shows how to control a hopper and external lighting.
The hopper is connected to OUT terminal 1 on the IF-240 (feeder number = 1), and is controlled with the PF_OutputONOFF command.
The hopper has been configured to supply 30 parts every second while in operation.
In this example, assume that the "Number of part supplies" is set to 60.
Assume that the external lighting is set to a standard IO.

```
' ** IO Label (Output) **
' O_FrontLight     External lighting

Function PF_Control(partID As Integer, Control As Integer) As Integer

  Integer duration, numPart

  numPart = PF_Info(partID, PF_INFO_ID_FEEDER_CALIB_CORRECT_MAXNUM)

  Select Control

      ' Hopper operation.  When the feeder contains no parts
      Case PF_CONTROL_SUPPLY_FIRST
          duration = numPart / 1000.0 * 30
          PF_OutputOnOff 1, On, 1, duration

      ' Hopper operation.  When adding parts to the feeder
      Case PF_CONTROL_SUPPLY
          duration = 2000
          PF_OutputONOFF 1, On, 1, duration

      ' User lighting On
      Case PF_CONTROL_LIGHT_ON
          On O_FrontLight

      ' User lighting Off
      Case PF_CONTROL_LIGHT_OFF
          Off O_FrontLight

  Send

  PF_Control = PF_CALLBACK_SUCCESS

Fend
```

PF_Status

Describes processes (mainly error processes) based on the callback function return value, and the system status (parts not supplied, etc.).

Syntax

```
Function PF_Status(part ID As Integer, Status As Integer) As Integer
  ' (Error processing)
Fend
```

Parameters

part ID	The part ID (integer number from 1 to 32) goes here. When multi-part operation, the active part goes here.
status	The status goes here. This is either the callback function return value or a value set by the system. (see Description)

Return Values

If the return value is not specified, it is equal that PF_EXIT is specified, and the PartFeeding process is terminated.

PF_CONTINUE
Continues the PartFeeding process. Normally this value should be specified.

PF_EXIT
Terminate the Part Feeding process. When this value is specified, the Part Feeding process will terminated. If this value is specified, the device should be returned to the initial state (i.e., returning the robot to the home position, returning the robot to the motor is off, etc.).

PF_RESTART
Restart the Part Feeding process from vision.

PF_RESTART_ACTIVEPART
Restart the Part Feeding process from vision. Only acquire an image and load the queue for the Active Part.

Description

This function runs after other callback functions (except the PF_CycleStop function and the PF_Status function).

The return value for the callback function just performed or the error that occurred in the Part Feeding process is set to the Status parameter. Describes processes based on these Status values.

PF_CALLBACK_SUCCESS
Callback function completed successfully. Normally nothing to be processed.

PF_CALLBACK_RESTART
PF_Robot callback function finished successfully and the return value is PF_CALLBACK_RESTART. Normally nothing to be processed.

PF_CALLBACK_RESTART_ACTIVEPART
PF_Robot callback function finished successfully and the return value is PF_CALLBACK_RESTART_ACTIVEPART. Normally nothing to be processed.

PF_STATUS_NOPART
This status indicates that parts are not being supplied from the hopper. Describes the operation to supply parts to the hopper. Normally nothing to be processed.

PF_STATUS_TOOMANYPART
This status indicates that too many parts are being supplied from the hopper, and that parts cannot be picked up. Describe the operation to remove parts from the hopper using an operator call, etc. Review hopper settings if this status occurs regularly.

PF_STATUS_BAD_ID
The part ID specified when running the PF_Start command is invalid. Make sure that you have specified the registered part ID correctly.
You tried to start with multiple feeders during a multi-part operation. Please review the settings for each part.
This immediately terminates the Part Feeding process.

PF_STATUS_BAD_PARAMETER
The part parameter specified when running the PF_Start command is invalid.
This immediately terminates the Part Feeding process.

PF_STATUS_CAL_NOT_COMPLETE
The part specified when running the PF_Start command has not completed the feeder calibration process. This immediately terminates the Part Feeding process. Run the calibration using the information provided in *2.3.9 Calibration* as a reference.

PF_STATUS_WRONGPART
The parts remaining on the feeder could not be detected. Check that the part vision sequence can detect the parts properly, or check for different types of parts or damaged parts. This Status value occurs after multiple attempts have been made to singulate the parts and the Part Blob sequence sees that there is something inside the Pick Region but the Part sequence is unable to identify it as a Front or Back part.

PF_STATUS_PARTBLOB_ERROR
The vision sequence or object for part blob detection is not valid. This immediately terminates the Part Feeding process. Check the part blob sequence and object that is used for the part.

PF_STATUS_PARTSEQ_ERROR
The vision sequence or object(s) used for part detection are not valid. This immediately terminates the Part Feeding process. Check the part sequence and object(s) that are used to detect the part.

PF_STATUS_ERROR
An error (system error) occurred while running the PF_Start command. This immediately terminates the Part Feeding process. Check that the vision sequence set in *2.3.3. Vision* functions properly. Debug callback functions individually to verify whether they function properly. Error 7730 "The maximum number of robots per feeder has been exceeded." can also occur when attempting to share a feeder with more than 2 robots.

Operation process processes can be specified with the Status callback function return value.
PF_EXIT is set whenever a return value is not specified. This will terminate the Part Feeding process. Make sure to set a return value to avoid this.

PF_STATUS_FEEDERINUSE_ERROR
The Part Feeding process was launched multiple times for the same feeder. The Feeding process is terminated immediately. Part Feeding process that is already running will continue.
Recheck the program.

PF_STATUS_PARTNOTENABLED
The part is disabled.
EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] - [General], make sure Enabled is checked.

PF_STATUS_PURGENOTENABLED
The PF_Purge function has been executed, although purging has been disabled.
EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] - [Purge], make sure Enabled is checked.

Note

1. Ensure that errors do not occur inside the PF_Status callback. If errors occur inside PF_Status, then PF_Status could be called recursively and error processing may not complete.

2. It is strongly recommended not to execute Feeder control commands (PF_Center, PF_CenterByShift, PF_Flip, PF_Shift) inside the PF_Status function.

3. PF_Status is responsible for telling the system how to proceed after completing the previous callback function (i.e., how to proceed after PF_Robot, PF_Vision, PF_Control, PF_MobileCam, PF_Feeder). This is accomplished by setting the PF_Status return value to one of the following return values - PF_CONTINUE or PF_RESTART or PF_RESTART_ACTIVEPART or PF_EXIT.
See the example program below for details.

Program Example

The following program describes error processing.
The user error referred to in this example is the suction timeout error referred to in the program example used for the Robot callback function.
If this error occurs, the robot motors are turned off.

```
' ** User Error **
' 8001 Suction timeout occurred

Function PF_Status(PartID As Integer, Status As Integer) As Integer

  Select Status

      Case PF_CALLBACK_SUCCESS
          '  Success (do nothing under normal circumstances)
      Case PF_CALLBACK_RESTART
          ' Restart from vision
          PF_Status = PF_RESTART
          Exit Function

      Case PF_CALLBACK_RESTART_ACTIVEPART
          ' Restart from vision –
          ' only acquire image and the load queue for Active Part
          PF_Status = PF_RESTART_ACTIVEPART
          Exit Function

      Case PF_STATUS_NOPART
          '  No parts in hopper
          MsgBox "Hopper empty."

      Case PF_STATUS_TOOMANYPART
          '  Too many parts in feeder
          MsgBox "Too many parts on Feeder."

      Case PF_STATUS_BAD_ID
          '  The specified part ID does not exist
          MsgBox "Bad PartID."

      Case PF_STATUS_BAD_PARAMETER
          '  Invalid part parameter
          MsgBox "Bad parameter."

      Case PF_STATUS_CAL_NOT_COMPLETE
          '  Calibration not complete
          MsgBox "Calibration incomplete."
      Case PF_STATUS_WRONGPART
         ' There may be a wrong part on the feeder platform.
         MsgBox “Wrong Part.”

      Case PF_STATUS_ERROR
         ' Error
         MsgBox "Error!! (code: " + Str$(Err) + " )  " + ErrMsg$(Err)

      Case PF_STATUS_PARTBLOB_ERROR
         ' Part Blob vision error
         MsgBox "Part Blob vision error."

      Case PF_STATUS_PARTSEQ_ERROR
         ' Part Sequence vision error
         MsgBox "Part Sequence vision error."

      Case PF_STATUS_FEEDERINUSE_ERROR
         ' Feeder is already in use
         MsgBox "Feeder is already in use."

      Case PF_STATUS_PARTNOTENABLED
         ' Part is disabled
         MsgBox “Part is disabled.”

      Case PF_STATUS_PURGENOTENABLED
         ' Purge is disabled.
         MsgBox "Purge is disabled."

      Case 8001
         ' Example: Suction timeout
         MsgBox "Vacuum Error!!"
         Motor Off
         PF_Status = PF_EXIT
         Exit Function

   Send

   PF_Status = PF_CONTINUE

 Fend
```

PF_MobileCam

Describe the process used to move the robot to the image capture position, and to retreat back after image capturing, when using a mobile camera. This function will always run, regardless of whether or not a mobile camera is in use.

Syntax

```
Function PF_MobileCam(part ID As Integer, Action As Integer) As Integer
  ' (Move the robot to image capture position)
  ' (Retract the robot)
Fend
```

Parameters

part ID	The part ID (integer number from 1 to 32) goes here. When multi-part operation, the active part goes here.
Action	The type of movement goes here.

Type of movement	Constant (defined in PartFeeding.inc)
Move the robot to the image capture position (and perform vision imaging and feeder operations after this)	PF_MOBILECAM_BEFORE
Retract the robot (and run the PF_Robot callback function after this)	PF_MOBILECAM_AFTER

Return Values

Under normal circumstances, set the PF_CALLBACK_SUCCESS constant (defined in PartFeeding.inc).
To end this function to perform error processing, set a user-defined error number (8000 - 8999). This value is passed back as a PF_Status callback function parameter.

Description

This function runs before image capturing, and before the PF_Robot callback function is called.
After describing this function, test whether the robot can travel to the assigned destination points safely.

Program Example

The following example describes a program used to move a mobile camera to the image capture position and retract it.
Positions are defined as point data, and labeled as VisionPos for the image capture position, and HomePos for the retraction position.

```
' ** Point **
' VisionPos   Mobile camera image capture position
' HomePos     Mobile camera retraction position
'
Function PF_MobileCam(partID As Integer, Action As Integer) As Integer

  Select Action

      Case PF_MOBILECAM_BEFORE '  Move the robot to image capture position
          Jump VisionPos
      Case PF_MOBILECAM_AFTER '  Retract the robot
          Jump HomePos
  Send

  MobileCam = PF_CALLBACK_SUCCESS
Fend
```

PF_Vision

This is used when the user wishes to program image capture processes (light control, using vision system, SPEL program processing). This is also used when it is difficult to detect parts and determine orientation (side facing, etc.) with vision system alone. To use this function, select “User processes vision for part via PF_Vision callback” in EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] - [Vision].

Syntax

```
Function PF_Vision(part ID As Integer, ByRef NumBack As Integer) As Integer
  ' (Vision processing)
Fend
```

Parameters

part ID	The part ID (integer number from 1 to 32) goes here. When multi-part operation, the active part goes here.
numBack	Assigns the number of parts that cannot be picked up (facing down, etc.). (ByRef is specified) This value is used by the Part Feeding process to determine whether flipping is required. Enter “0” if the part does not have defined sides.

Return Values

To continue processing, set the return value to PF_CALLBACK_SUCCESS.
To perform error processing, set the return value to a user-defined error number (8000 - 8999). This value is passed back as a PF_Status callback function parameter.

Description

This function is used when using combinations of multiple vision sequences and external lighting controls to detect parts coordinates. The following processes are to be described by the user.

- User lighting controls
- Vision processing (running the VGet statement)
- Parts coordinates queue management (initialization and registering point data)

Program Example

The following example describes a program detecting parts using two vision sequences, VSeq1 (including one Geom object) and VSeq2 (including one Geom object).
VSeq1 is used with the user lighting 1. This can detect the position of parts, but cannot determine their orientation. VSeq2 is used with the user lighting 2, and can only detect parts facing up.
The coordinates of detected parts (local number 1 in this example) are added to the coordinates queue with the PF_QueAdd command. When doing so, the value obtained in *2.3.6 Teach Window* is assigned as the Z coordinate.
Further, in order to enable flipping, first enable flipping in *2.3.1 General*, specify VSeq2 for “Part Vision Sequence” in *2.3.3. Vision*, turn on the user lighting 2, and then perform the flip calibration as described in *2.3.7 Calibration*.

```
' ** IO Label (Output) **
' O_FrontLight1   User lighting 1
' O_FrontLight2   User lighting 2

Function PF_Vision(partID As Integer, ByRef NumBack As Integer) As
Integer

  Boolean found
  Integer i, numFound
  Real PX_X, PX_Y, PX_U
  Real RB_X, RB_Y, RB_U, RB_Z

  ' Pick Z coordinate
```

```
    RB_Z = -80.0

    ' Initialize coordinates queue
    PF_QueRemove partID, All

    ' Turn on the user lighting 1
    On O_FrontLight1

    ' Detect part (run UsrVisionSeq1)
    VRun VSeq1
    VGet VSeq1.Geom01.NumberFound, numFound

    ' Turn off the user lighting 1
    Off O_FrontLight1
    ' Turn on the user lighting 2
    On O_FrontLight2

    NumBack = 0
    For i = 1 To numFound
        ' Retrieve XY pixel coordinates for part detected with VSeq1.Geom01
        ' Set the VSeq2.Geom01 search window to cover this part
        ' Part size = 100 x 100 pxl
        VGet VSeq1.Geom01.PixelXYU(i), found, PX_X, PX_Y, PX_U
        VSet VSeq2.Geom01.SearchWinLeft, (PX_X - 50)
        VSet VSeq2.Geom01.SearchWinTop, (PX_Y - 50)

        ' Run VSeq2
        VRun VSeq2
        VGet VSeq2.Geom01.Found, found

        If found = True Then
          ' If found, register to the coordinates queue as the part is facing up
          ' Local number is 1
          VGet VSeq1.Geom01.RobotXYU(i), found, RB_X, RB_Y, RB_U
          PF_QueAdd partID, XY(RB_X, RB_Y, RB_Z, RB_U) /1

        Else
          ' If not found, part is facing down
          NumBack = NumBack + 1

        EndIf

    Next

    ' Turn off the external lighting 2
    Off O_FrontLight2

    PF_Vision = PF_CALLBACK_SUCCESS

  Fend
```

PF_Feeder

Describes what actions are recommended with feeder control commands (PF_CenterByShift, PF_Center, PF_Flip, and PF_Shift commands) if the user processes the feeder vibration rather than the system.

Syntax

```
Function PF_Feeder(PartID As Integer, NumFrontParts As Integer, NumBackParts As Integer,
state As Integer) As Integer
   ' (Part Judgement)
   ' (Vibration Action)
Fend
```

Parameters

Part ID	Specifies the part ID (integer value 1 to 32). When multi-part operation, the active part goes here.
NumFrontParts	The number of Front parts that were found by vision and loaded into the part queue.
NumBackParts	The number of Back parts that were found by vision.
State	The state of parts, or the recommended action that has been determined by the System goes here. One of the values from the table below will be provided.

State / Recommended action	Constant (defined in PartFeeding.inc)	Notes
OK to pick	PF_FEEDER_PICKOK	Parts are available to pick
Supply more parts	PF_FEEDER_SUPPLY	Hopper needs to supply more parts when the Supply method is [supply parts during pick and place] (see *2.3.5 Part Supply*)
Parts are spread out but need to be flipped	PF_FEEDER_FLIP	Flip the parts
Shift parts into pick region	PF_FEEDER_SHIFT	Shift forward into region
Center, Flip and Separate	PF_FEEDER_CENTER_FLIP	Center, Flip and Separate
Hopper is empty	PF_FEEDER_HOPPER_EMPTY	No part in the hopper. Notify the operator and supply parts in the hopper
Parts have gathered against the platform wall	PF_FEEDER_SHIFT_BACKWARD	Shift parts back into pick region
Hopper Supply, Center, Flip and Separate	PF_FEEDER_SUPPLY_CENTER_FLIP	Hopper needs to supply more parts and the parts need to be centered, flipped and separated
Too many parts	PF_FEEDER_TOO_MANY	There are too many parts on the tray. Notify the operator.
Wrong part	PF_FEEDER_WRONGPART	There may be a wrong part on the feeder platform. Notify the operator.
Plate Type is Stuctured	PF_FEEDER_UNKNOWN	The system does not know how to handle vibration for a custom, structured plate (see *2.3.2 Vibration*)

Return Values

Specify the following values based on processes you want the system to perform after PF_Feeder ends.

Constant (defined in PartFeeding.inc)	Part Feeding process operations
PF_CALLBACK_SUCCESS	Specify this when the part is ready to be picked and no more feeder operations are necessary. The system will call the PF_Robot callback function. Note: The part coordinate queue will not be regenerated. If you return this value after a feeder operation, there will be a difference between the part coordinates on the feeder and the data in the part coordinate queue.
PF_CALLBACK_RESTART	Specify this after the feeder operation has been performed. The system will regenerate all the part coordinate queues and call the PF_Feeder callback function again.
PF_CALLBACK_RESTART_ACTIVEPART	Specify this after the feeder operation has been performed. The system will regenerate the part coordinate queue for the active part only and call the PF_Feeder callback function again.

Description

This function allows the user to handle the feeding judgement and action. Normally, the system will process the vibration for parts. When "User processes vibration for part via PF_Feeder callback" in [PartFeeding] – [Part] – [Virbation] is selected, the user decides how to feed the parts. The PF_Feeder callback can be used to solve difficult applications where the system vibration method is not performing well.
For example, PF_Feeder cannot determine what to do with a custom platform (a special platform with Holes, Slots or Pockets). Instead, you can write your own feeder processing in the PF_Feeder callback function.

The NumFrontParts and NumBackParts parameters can be used to help determine the appropriate vibration action. For example, if NumFrontParts > 0 then parts are available to be picked and the system can continue without any vibration.
The State parameter is the system's recommended action or the state of parts. It may be helpful in determining how to best vibrate the feeder.

The PF_Feeder callback must return one of the values shown in the table above. The return value tells the system how to proceed.

Program Example 1:

The following example illustrates how to perform part judgement and vibration action via the PF_Feeder callback using the System's recommended State parameter.

```
Function PF_Feeder(PartID As Integer, NumFrontParts As Integer,
NumBackParts As Integer, state As Integer) As Integer

  Select state

      Case PF_FEEDER_PICKOK
          ' Call PF_Robot because there are parts ready to pick
          PF_Feeder = PF_CALLBACK_SUCCESS

      Case PF_FEEDER_SUPPLY
          ' Need to supply more parts
          PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)
          ' Shift forward and then Flip
          PF_Shift PartID, PF_SHIFT_FORWARD, 500
          PF_Flip PartID
          ' Restart and re-acquire images
          PF_Feeder = PF_CALLBACK_RESTART

      Case PF_FEEDER_FLIP
          ' Flip the parts
          PF_Flip PartID
          ' Restart and re-acquire images
          PF_Feeder = PF_CALLBACK_RESTART

      Case PF_FEEDER_CENTER_FLIP
          ' Center, Flip and Separate the parts
          PF_Center PartID, PF_CENTER_LONG_AXIS, 900
          PF_Center PartID, PF_CENTER_SHORT_AXIS
          PF_Flip PartID
          ' Restart and re-acquire images
          PF_Feeder = PF_CALLBACK_RESTART

      Case PF_FEEDER_TOO_MANY
          ' Notify operator that there are too many parts on the feeder
          PFStatusReturnVal = PF_Status(PartID, PF_STATUS_TOOMANYPART)
          ' Restart and re-acquire images
          PF_Feeder = PF_CALLBACK_RESTART

  Send

Fend
```

Program Example 2:

The following example illustrates how to perform part judgement and vibration action via the PF_Feeder callback using the NumFrontParts and NumBackParts parameters.

```
Function PF_Feeder(PartID As Integer, NumFrontParts As Integer,
NumBackParts As Integer, state As Integer) As Integer
  Integer PFControlReturnVal

  Select True

      Case NumFrontParts = 0 And NumBackParts <> 0
          PF_CenterByShift PartID
          PF_Flip PartID
          ' Restart and re-acquire images
```

```
            PF_Feeder = PF_CALLBACK_RESTART

        Case NumFrontParts = 0 And NumBackParts = 0
            PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)
            ‘ Center, Flip and Separate
            PF_Center 1, PF_CENTER_LONG_AXIS, 900
            PF_Center 1, PF_CENTER_SHORT_AXIS, 700
            PF_Flip 1, 600
            PF_Feeder = PF_CALLBACK_RESTART 're-acquire images

        Default
            ' Call PF_Robot because there are parts ready to pick
            PF_Feeder = PF_CALLBACK_SUCCESS

    Send

  Fend
```

PF_CycleStop

Describe processes after the PF_Stop function has been executed.

Syntax

```
Function PF_CycleStop(part ID As Integer)
  ' (Processes when stopped)
Fend
```

Parameters

part ID	The part ID (integer number from 1 to 32) goes here. When multi-part operation, the active part goes here.

Return Values

None

Description

This function runs after the callback function in progress stops running once PF_Stop has been executed. System processes are described here. Processes to be described include returning the robot to its home position and turning off the robot motors.

Program Example

The following program executes processes to return the robot to its home position and turn off the motors when the PF_Stop command is issued.

```
Function PF_CycleStop(partID As Integer)

  Home
  Motor Off

Fend
```

5. Part Feeding Log File

5.1 Summary

The Part Feeding log file is a log file that records the following actions performed as part of the process of operations, and the operation time and results.

- Vision processing
- Feeder control
- Callback function operation (Robot, Status)

The Part Feeding log file can be used to perform the following.

- Check the cycle time for part pick up.
- Check the number of parts that can be picked up with each attempt in order to adjust the hopper feed amount.
- Check and make improvements to actions taking up an undue amount of time based on operation processing times.

Note that the Controller must be connected to the PC to use this function.

5.2 Enabling the Log Function

Connect the Controller to the PC.

Program PF_InitLog to run before PF_Start.

For more details, refer to *3. Part Feeding SPEL+ Command Reference PF_InitLog*.

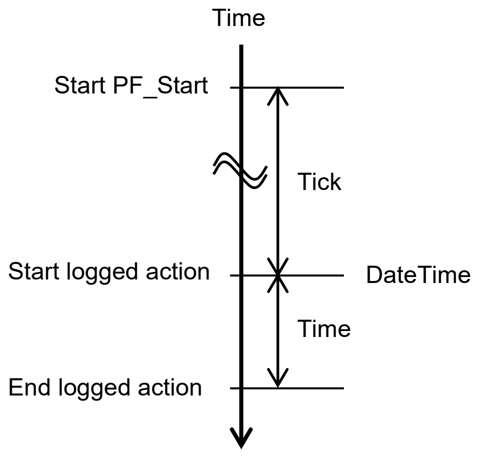
5.3 Log File Format

5.3.1 Common Items

Log files are in CSV format. File names are specified as a PF_InitLog parameter.
The following data is recorded in chronological order to a single log file. The "Data" fields vary depending upon the log Type. All other fields are common.

Column	Column name	Format	Details
1	DateTime	Character string	Time action started (yyyy/mm/dd hh:MM:ss)
2	Tick	Real value	Time elapsed since PF_Start started running [seconds] (s.sss)
3	Time	Real value	Processing time [seconds] (s.sss)
4	Type	Character string	Operation Type
5	ID	Integer number	Part ID
6	Data1	Varies depending on the data type (Type).	
7	Data2		
8	Data3		
9	PartName	Character string	Name of the part
10	RobotNo	Integer number	Robot number assigned to the part
11	FeederNo	Integer number	Feeder number assigned to the part
12	Project	Character string	EPSON RC+ project name

The relationship between the DateTime reading and the Tick, Time reading is as shown in the diagram below.



5.3.2 Vision Sequence Log

This log records the time required by the Part Feeding process to process the vision sequence specified by the Part Vision Sequence described in *2.3.3. Vision*, the number of parts detected facing up, and the number of parts facing down.

Column	Column name	Format	Details
4	Type	Character string	Log type (“UserVision”)
5	ID	Integer number	Part ID
6	NumFront	Integer number	Number of parts facing up
7	NumBack	Integer number	Number of parts facing down

5.3.3 System Vision Sequence Log

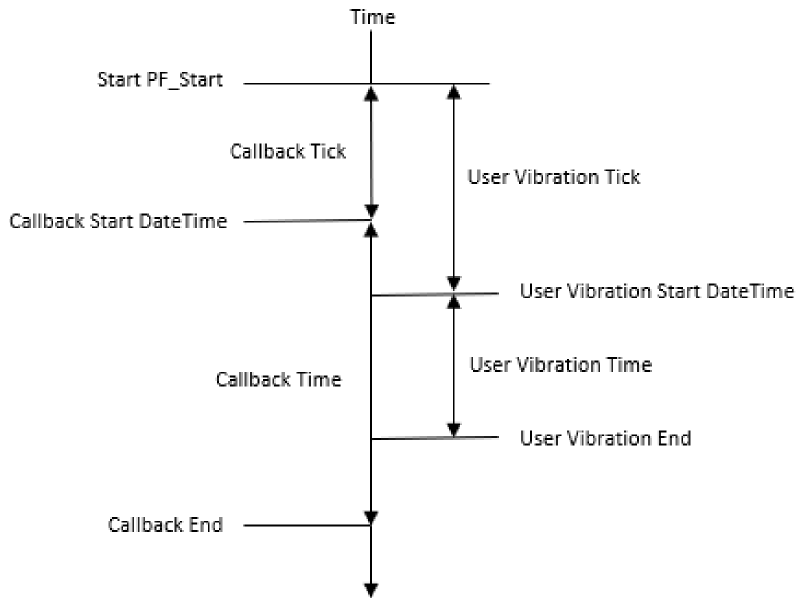
This log records the time required by the Part Feeding process to process the vision sequence generated internally to detect the distribution of parts, etc.

Column	Column name	Format	Details
4	Type	Character string	Log type (“SystemVision”)
5	ID	Integer number	Part ID

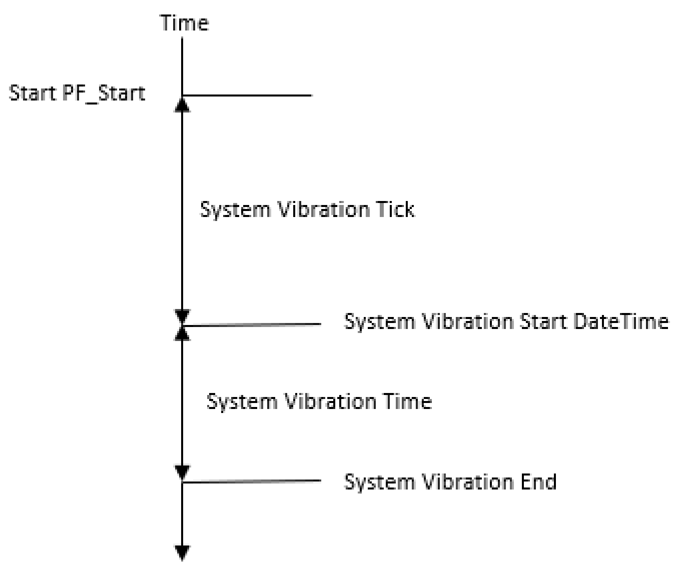
5.3.4 Vibration Log

This log records types of feeder vibration actions performed by the System or by the User.

Column	Column name	Format	Details
4	Type	Character string	Operation type: Separation — Separation Centering — Centering Shift — Shift BackShift — Back shift Flip — Flip CenterByShift — Center by shift Purge — Purge QtyAdjHopperTime — QtyAdjHopperTime
5	ID	Integer number	Part ID
6	Callback Name	Character string	Name of the callback (or System) where the vibration was executed: System — Feeder Robot — Vision Control — MobileCam CycleStop — Status



Timing Diagram: User Vibration executed inside a Callback



Timing Diagram: System Vibration

NOTE The Data1 column in the log file will show "System" if the vibration is performed by the system. Otherwise, Data1 will show the name of the Callback where the User executed the vibration.

5.3.5 PF_Robot Callback Function Log

This log records the number of parts processed with the PF_Robot callback function.

Column	Column name	Format	Details
4	Type	Character string	Operation type ("Robot")
5	ID	Integer number	Part ID
6	Num	Integer number	Number of parts processed (Active parts only) (Number of registered entries in the coordinates queue before calling - Number of registered entries in the coordinates queue after calling)

5.3.6 PF_MobileCam Callback Function Log

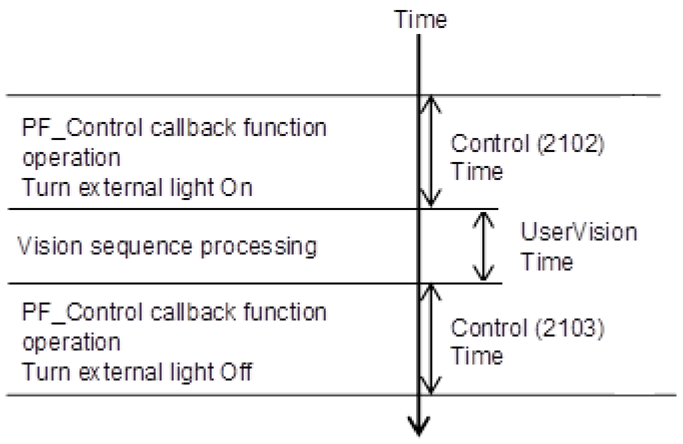
This log records the PF_MobileCam callback function action type.

Column	Column name	Format	Details
4	Type	Character string	Operation type ("MobileCam")
5	ID	Integer number	Part ID
6	Action	Integer number	Move the robot to image capture position 2001 Retract the robot 2002

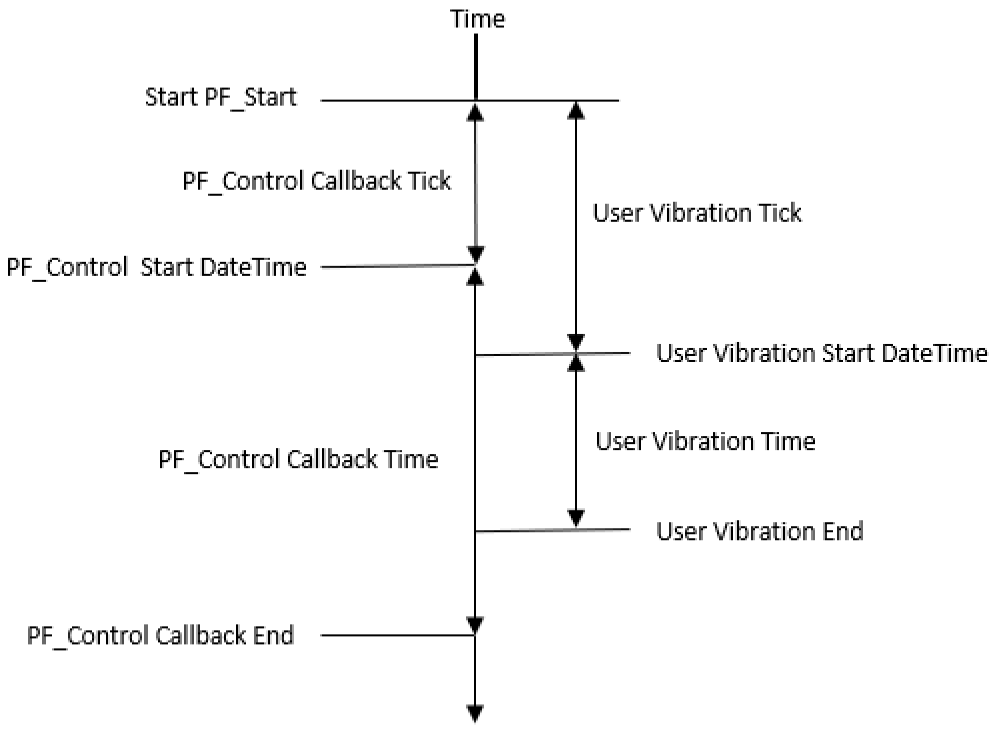
5.3.7 PF_Control Callback Function Log

This log records the PF_Control callback function action type.

Column	Column name	Format	Details
4	Type	Character string	Operation type ("Control")
5	ID	Integer number	Part ID
6	Action	Integer number	Hopper operation (no parts) 2100 Hopper operation (add parts) 2101 User lighting On 2102 User lighting Off 2103


Time
PF_Control callback function operation Turn external light On
Control (2102) Time
Vision sequence processing
UserVision Time
PF_Control callback function operation Turn external light Off
Control (2103) Time

Timing Diagram: Vision with Front Light



Timing Diagram: User Vibration executed inside PF_Control Callback

5.3.8 PF_Status Callback Function Log

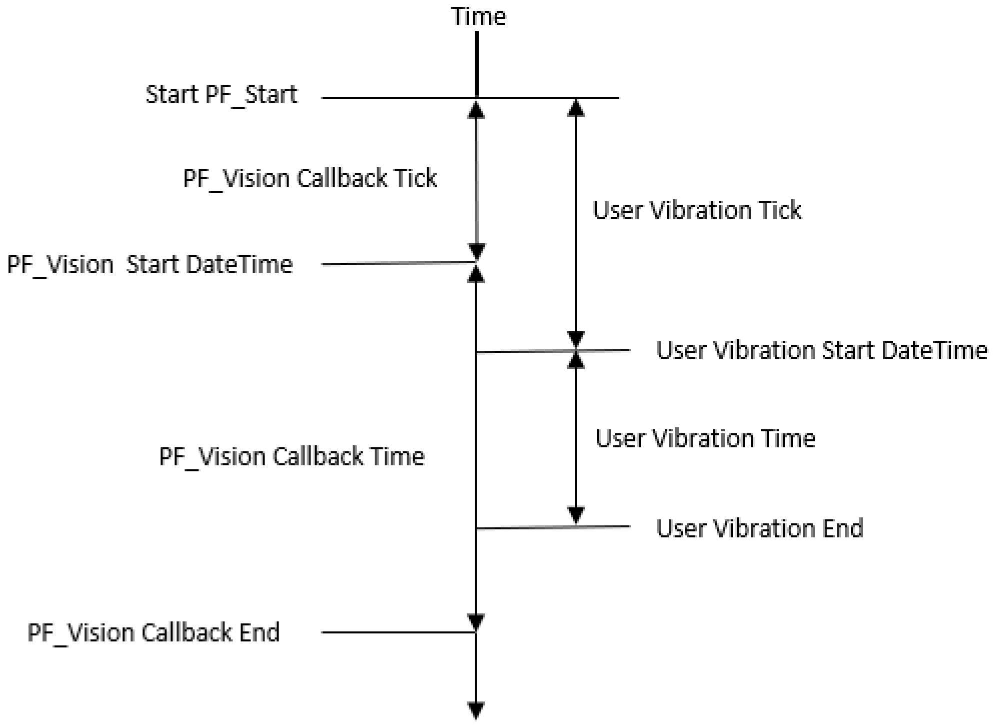
This log records the PF_Status callback function status type.

Column	Column name	Format	Details
4	Type	Character string	Operation type ("Status")
5	ID	Integer number	Part ID
6	Status	Integer number	The status or user error occurred Normal 0 Parts not supplied 2200 Excessive parts 2201 Invalid ID 2202 Invalid parameter 2203 Calibration not complete 2204 System error 2205 Part cannot detect 2206 Part blob sequence failure 2207 Part vision sequence failure 2208 Feeder in use 2209 Part disabled 2210 Purge disabled 2211 User error 8000 - 8999

5.3.9 PF_Vision Callback Function Log

This log records the number of parts detected with the PF_Vision callback function that are facing up/down.

Column	Column name	Format	Details
4	Type	Character string	Operation type ("VisionCallback")
5	ID	Integer number	Part ID
6	NumFront	Integer number	Number of parts detected that are facing up
7	NumBack	Integer number	Number of parts detected that are facing down

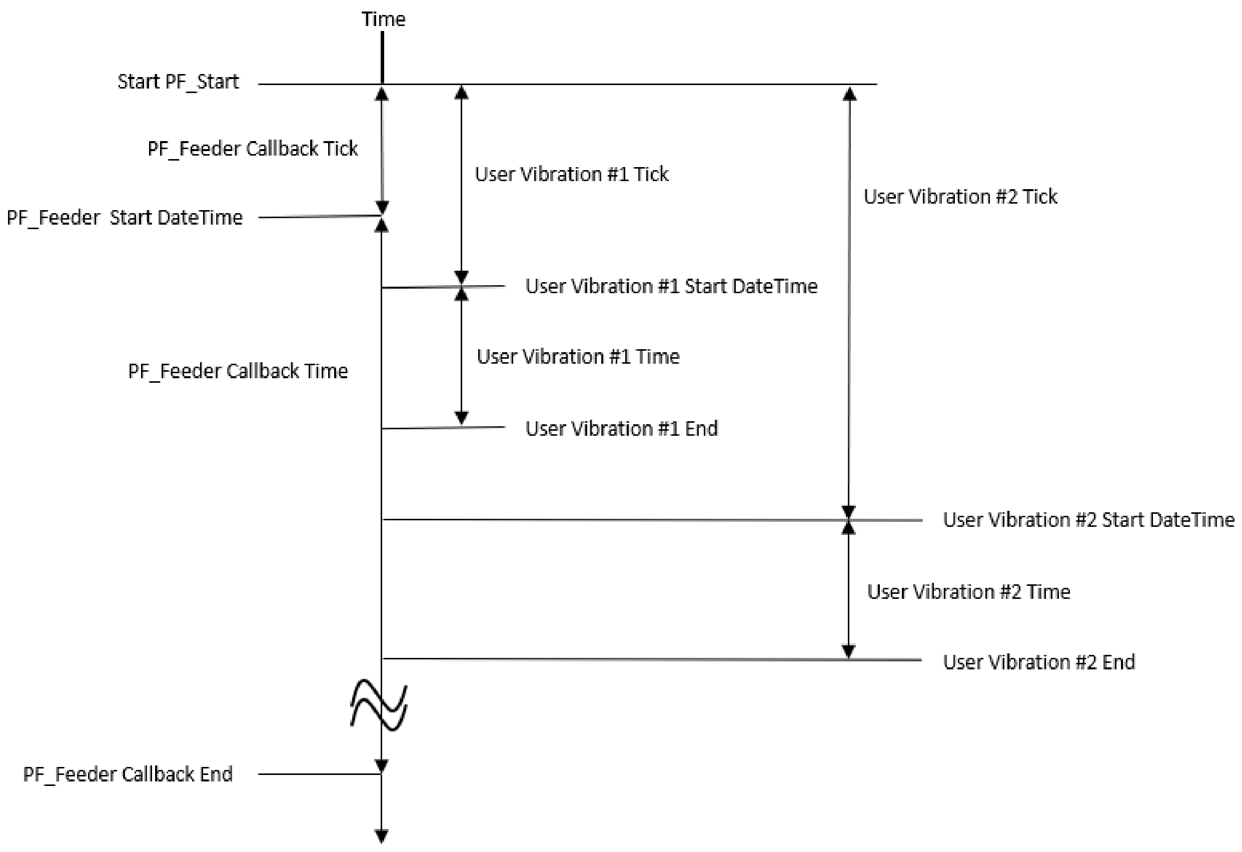


Timing Diagram: User Vibration executed inside PF_Vision Callback

5.3.10 PF_Feeder Callback Function Log

This log records the PF_Feeder callback function state type.

Column	Column name	Format	Details
4	Type	Character string	Operation type ("Feeder")
5	ID	Integer number	Part ID
6	State	Integer number	Recommended Action: Plate Type is Structured 0 OK to pick 1 Supply more parts 2 Flip 3 Shift 4 Center & Flip 5 Hopper is empty 6 Shift Backwards 7 Hopper supply, Center & Flip 8 Too many parts 9 Wrong part 10



Timing Diagram: User Vibrations executed inside PF_Feeder Callback

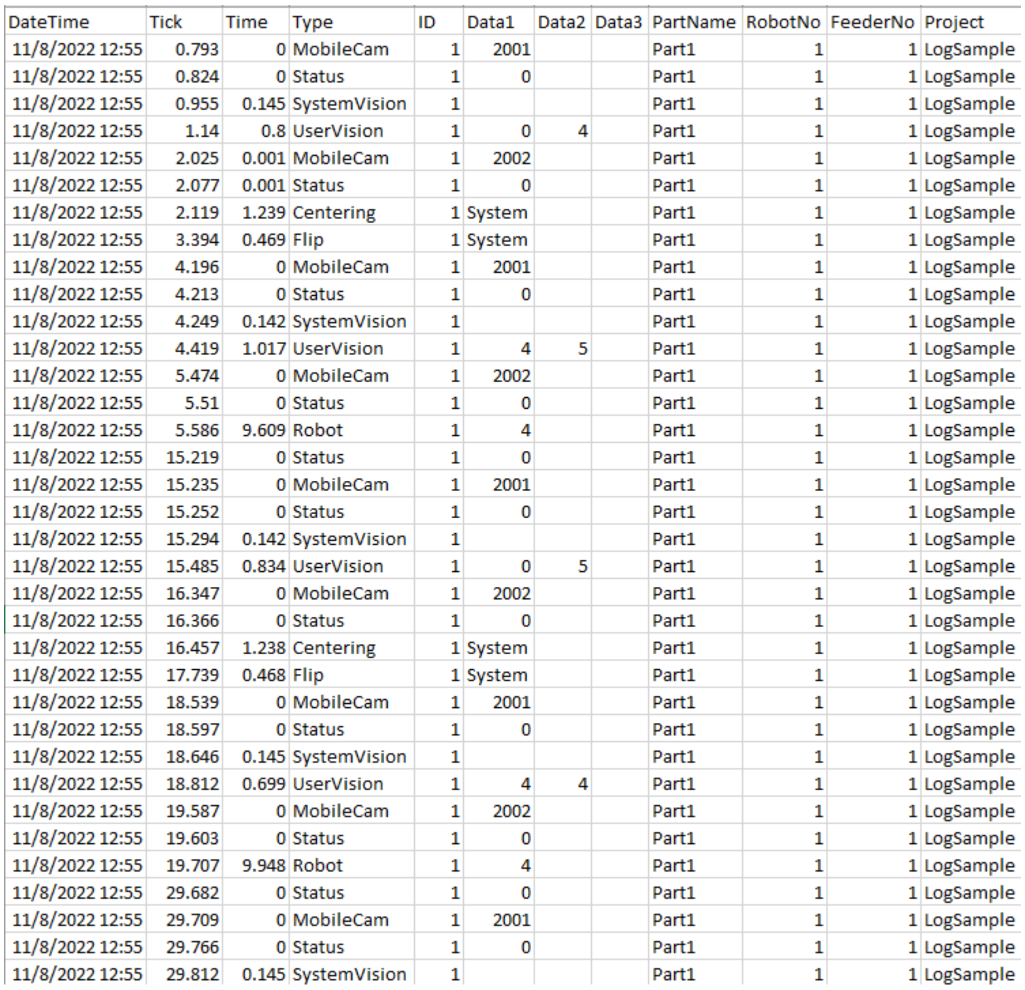
5.3.11 PF_CycleStop Callback Function Log

This log records the PF_CycleStop callback function state type.

Column	Column name	Format	Details
4	Type	Character string	Log type (“CycleStop”)
5	ID	Integer number	Part ID

5.4 Log Sample

The following is a sample of a Part Feeding log.



DateTime	Tick	Time	Type	ID	Data1	Data2	Data3	PartName	RobotNo	FeederNo	Project
11/8/2022 12:55	0.793	0	MobileCam	1	2001			Part1	1	1	LogSample
11/8/2022 12:55	0.824	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	0.955	0.145	SystemVision	1				Part1	1	1	LogSample
11/8/2022 12:55	1.14	0.8	UserVision	1	0	4		Part1	1	1	LogSample
11/8/2022 12:55	2.025	0.001	MobileCam	1	2002			Part1	1	1	LogSample
11/8/2022 12:55	2.077	0.001	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	2.119	1.239	Centering	1	System			Part1	1	1	LogSample
11/8/2022 12:55	3.394	0.469	Flip	1	System			Part1	1	1	LogSample
11/8/2022 12:55	4.196	0	MobileCam	1	2001			Part1	1	1	LogSample
11/8/2022 12:55	4.213	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	4.249	0.142	SystemVision	1				Part1	1	1	LogSample
11/8/2022 12:55	4.419	1.017	UserVision	1	4	5		Part1	1	1	LogSample
11/8/2022 12:55	5.474	0	MobileCam	1	2002			Part1	1	1	LogSample
11/8/2022 12:55	5.51	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	5.586	9.609	Robot	1	4			Part1	1	1	LogSample
11/8/2022 12:55	15.219	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	15.235	0	MobileCam	1	2001			Part1	1	1	LogSample
11/8/2022 12:55	15.252	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	15.294	0.142	SystemVision	1				Part1	1	1	LogSample
11/8/2022 12:55	15.485	0.834	UserVision	1	0	5		Part1	1	1	LogSample
11/8/2022 12:55	16.347	0	MobileCam	1	2002			Part1	1	1	LogSample
11/8/2022 12:55	16.366	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	16.457	1.238	Centering	1	System			Part1	1	1	LogSample
11/8/2022 12:55	17.739	0.468	Flip	1	System			Part1	1	1	LogSample
11/8/2022 12:55	18.539	0	MobileCam	1	2001			Part1	1	1	LogSample
11/8/2022 12:55	18.597	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	18.646	0.145	SystemVision	1				Part1	1	1	LogSample
11/8/2022 12:55	18.812	0.699	UserVision	1	4	4		Part1	1	1	LogSample
11/8/2022 12:55	19.587	0	MobileCam	1	2002			Part1	1	1	LogSample
11/8/2022 12:55	19.603	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	19.707	9.948	Robot	1	4			Part1	1	1	LogSample
11/8/2022 12:55	29.682	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	29.709	0	MobileCam	1	2001			Part1	1	1	LogSample
11/8/2022 12:55	29.766	0	Status	1	0			Part1	1	1	LogSample
11/8/2022 12:55	29.812	0.145	SystemVision	1				Part1	1	1	LogSample

6. Vision Sequences Used With the Part Feeding Option

The following two vision sequences must be created in order to use Part Feeding.

- Feeder calibration vision sequence
- Parts detection vision sequence

For more details on the Vision Guide, vision sequences and objects, refer to the *Vision Guide Software* manual.

Vision property details are explained in the *Vision Guide Properties and Results Reference* manual.

6.1 Vision Calibration

Perform vision calibration for the feeder platform surface.

For more information on how to perform vision calibration, refer to the following reference materials.

Vision Guide 7.0 (Ver. 7.4) Software 7. Vision Calibration

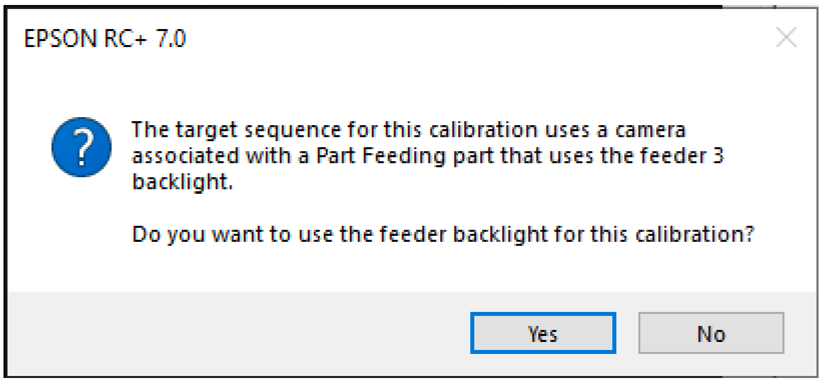
The following table list important properties for vision calibration.

Property	Configuration method
Camera	Set the camera number.
CameraOrientation	Set the camera fixing method. Fixed downward facing camera - Fixed downward Mobile camera - Mobile J4 or Mobile J6
RobotLocal	Specify the robot local number.
RobotNumber	Specify the robot number. Match this number to the robot number set in *2.2.2 General*.
RobotTool	Specify the robot tool number.

If there are one or more feeders with backlights:

NOTE

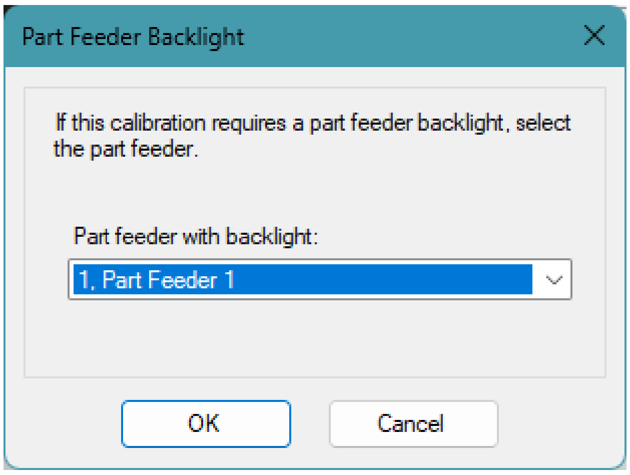
If the Calibration's TargetSequence Camera is the same as a part sequence, a message like the following will be displayed:


EPSON RC+ 7.0

The target sequence for this calibration uses a camera associated with a Part Feeding part that uses the feeder 3 backlight.

Do you want to use the feeder backlight for this calibration?

If the calibration's TargetSequence Camera is not used by any part sequences, then a dialog like the following will be displayed:



If this calibration requires a part feeder backlight, select the part feeder.

Part feeder with backlight: 1, Part Feeder 1

6.2 Part Vision Sequence

A parts detection vision sequence is used to detect parts and retrieve part coordinates. Create a vision sequence as outlined below.

NOTE

Create a separate parts detection vision sequence to the feeder calibration vision sequence.

6.2.1 Simple Parts

The following is an example of a vision sequence created to detect parts that simply need to be gripped without needing to take into account orientation (how the part is facing) and the degree of rotation.

Vision sequence

Configure the following properties. Make sure to configure or check the following. Configure other properties as required.

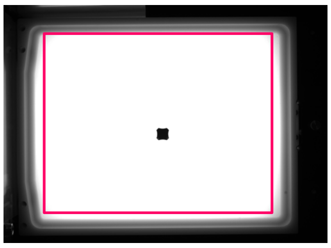
Property	Configuration method
Calibration	Configure vision calibration for the camera used for capturing images of the feeder. Specify the same calibration as that specified for the calibration vision sequence. Vision calibration must be completed.
Camera	Set the camera number for the camera used for capturing images of the feeder. Specify the same camera as that specified in the calibration vision sequence.
ExposureTime	Adjust this while the feeder backlight is on so that parts can be captured clearly. Additionally, ensure that the area surrounding the feeder does not appear darkened out.

Vision objects

Register one of the following objects for which robot coordinates can be retrieved.

- ArcFinder
- ArcInspector
- Blob
- BoxFinder
- ColorMatch
- CornerFinder
- Correlation
- DefectFinder
- Edge
- Geometric
- LineInspector
- Point
- Polar

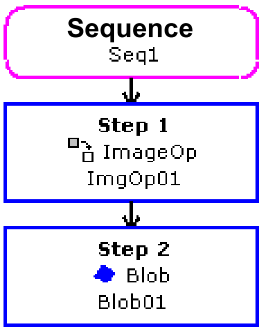
Configure the following properties for the object. Make sure to configure or check the following. Configure other properties as required based on the object type.

Property	Configuration method
NumberToFind	Set to All.
SearchWindow	Match to the inner circumference of the platform. Additionally, ensure that darkened areas surrounding the platform do not enter the search window. 

TIP

This can be used with other objects.

Example: Using an ImageOp object for binarization processing

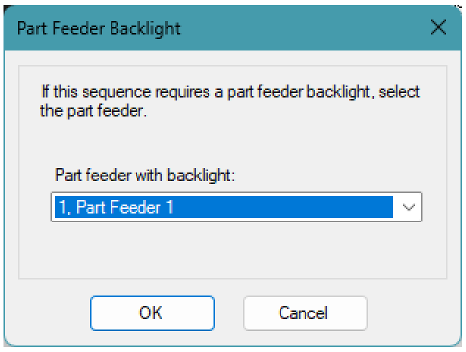


NOTE

When a new sequence is created and one or more part feeders with backlights exist, then a dialog is displayed asking the user to select a feeder with a backlight if desired. When running a vision sequence or object from the EPSON RC+ Vision Guide window, this selection will allow the backlight to be turned on automatically.

If the feeder with a backlight is an IF-80, then the backlight will turn off after the image is grabbed and the video will be temporarily frozen. Freezing the image allows the user to see the image even though the backlight has been turned off. The user can click on the video area to switch back to live video.

This is mainly for the case where a new project has been created but no part feeding parts have been added.


Part Feeder Backlight

If this sequence requires a part feeder backlight, select the part feeder.

Part feeder with backlight:

1, Part Feeder 1

6.2.2 Parts With Sides

Configure settings as follows to retrieve one side of parts with multiple sides.

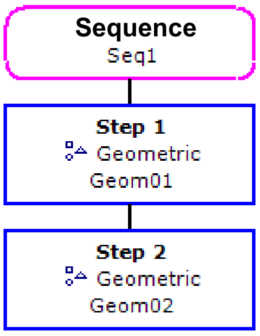
Vision sequence

Create a vision sequence in the same manner as *6.3.1 Simple Parts* 6.2.1 Simple Parts.

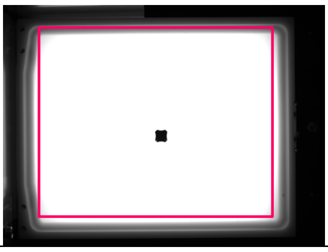
Vision objects

1. Register an object detecting the front side of parts (e.g.: Geometric).
2. Register an object detecting the back side of parts (e.g.: Geometric).

E.g.: Create two Geometric objects to determine which side a part is facing.
(Geom01 to detect the front side of parts, Geom02 to detect the back side of parts)



The Geometric property settings example is as follows.

Property	Configuration method
Accept	Set to ensure that parts are detected.
NumberToFind	Set to All.
SearchWindow	Match to the inner circumference of the platform. 
ModelWindow	Teach parts along their contour.

TIP

The Timeout property setting can be important for Geometric and Correlation vision objects. The Timeout property should be set just higher than the normal time it takes to run the sequence, otherwise it will sometimes take 2000 ms (default timeout).

Example:

Geometric with NumberToFind = 9
A lot of parts in the image
VRun
➔ Time to find is 75 ms. for this example.
Set NumberToFind = All
VRun
➔ Time is about 2000 ms (the timeout value)
Set Timeout to 100 ms
VRun
➔ Time is about 100 ms (the timeout value)

So, if Timeout is not set correctly, the application may take for up to 2000 ms (default) when the Geometric or Correlation runs. For time critical applications, it is important to set Timeout correctly.

6.2.3 Parts that Require Gripper Clearance

Some parts detected using pattern matches (Geometric, Correlation) may partially overlap, hindering the robot from picking up parts. Configure the following settings to exclude such parts using the clearance checking feature of Vision Guide.

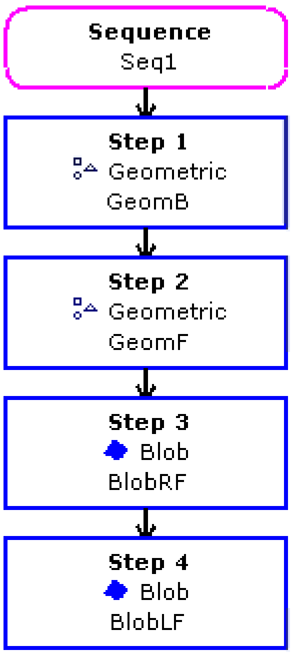
Vision sequence

Create a vision sequence in the same manner as *6.3.1. Simple Parts*.

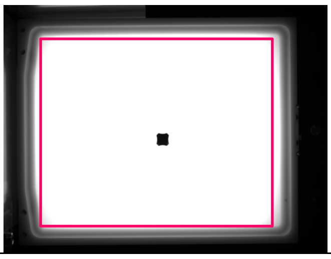
Vision objects

1. Add two objects for detecting parts. (E.g.: Geometric)
   Example: Create GeomF for front detection and GeomB for back detection.

2. Add one or more objects to check clearance for the object that detects the part.
   Example: Create Blob objects with CheckClearanceFor set to the Geometric object to determine overlapping parts.



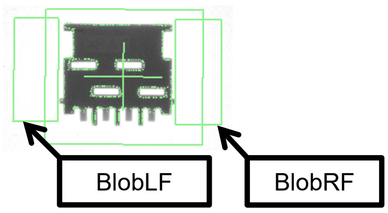
Geometric property settings example

Property	Settings example
Accept	Set to ensure that parts are detected.
NumberToFind	Set to All.
SearchWin	Match to the inner circumference of the platform. 
ModelWin	Teach parts along their contour.
Name	GeomB : Name of the object that detects the back side GeomF : Name of the object that detects the front side

Blob property settings example

Property	Settings example
CheckClearanceFor	GeomF Name of the object that detects the front side
ClearanceCondition	NotFound
FailColor	LightGreen
Name	BlobRF : Detect space for right finger BlobLF : Detect space for left finger
PassColor	Red
SearchWin Type	RotatedRectangle
ThresholdHigh	192

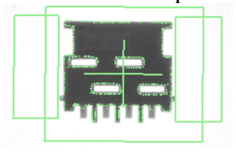
In this example, only the front side is checked for clearance.



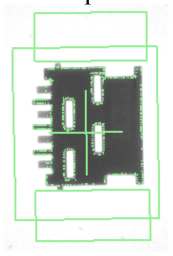
TIP

(1) One part is put on the center of the feeder.

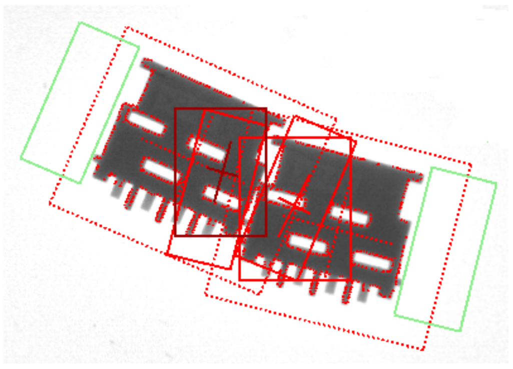
(2) Run Vision Sequence.



(3) Rotate parts 90°. Run Vision Sequence again.



Check to rotate BlobRF and BlobLF SearchWindow in accordance with the rotation of the parts. Detection result when there is a space for end effector finger. Detection result when there is no space for the end effector finger.



6.2.4 Special Vision Configurations

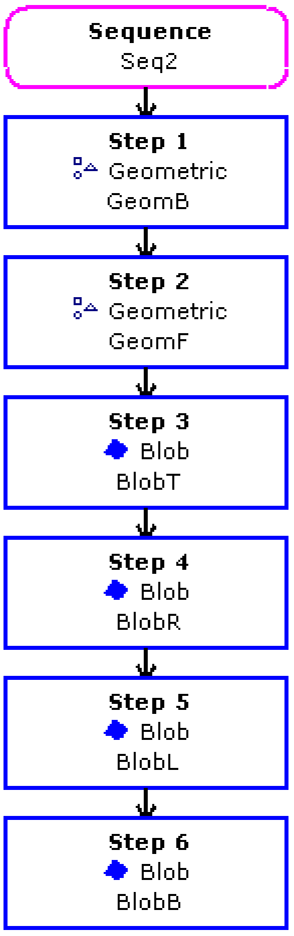
When using two or more vision sequences, or multiple lighting sources to detect parts, you can enable the vision callback function to describe all lighting controls and part detection processes.

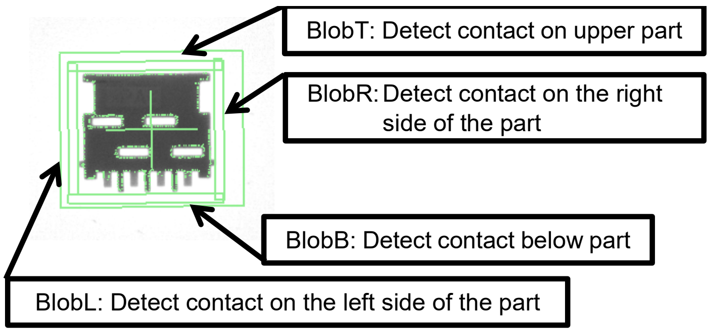
Refer to *4. Part Feeding Callback Functions PF_Vision* for more details.

6.2.5 Example of not picking up when contacting with the next parts

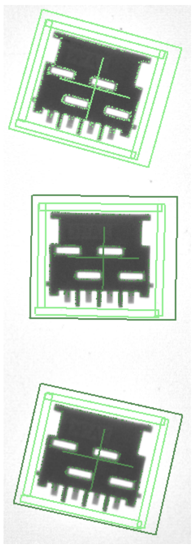
This section is an example of not making to the picking up object detecting the contact and a space with the next parts little.

The vision sequence is made as well as *6.2.3. Parts that Require Gripper Clearance*.

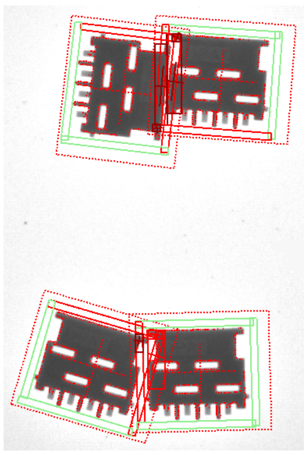



BlobT: Detect contact on upper part
BlobR: Detect contact on the right side of the part
BlobB: Detect contact below part
BlobL: Detect contact on the left side of the part

No contacting with the next parts.

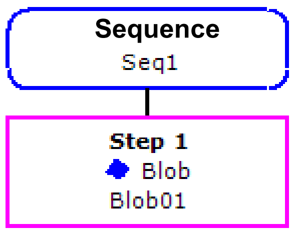


Contacting with the next parts.



6.3 Part Blob Vision Sequence

Create a vision sequence to use for feeder calibration.



NOTE

Create a separate feeder calibration vision sequence to the parts detection vision sequence.

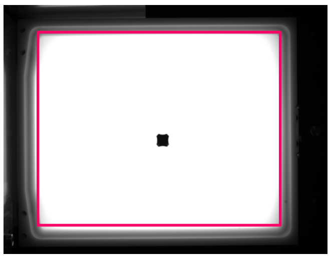
6.3.1 Vision Sequence

Configure the following properties. Make sure to configure or check the following.
Leave all other properties as their default values.

Property	Configuration method
Calibration	Configure the vision calibration created in *6.1. Vision Calibration*. Vision calibration must be completed (the CalComplete result must be True).
Camera	Set the camera number for the camera used for capturing images of the feeder.
ExposureTime	Adjust this while the feeder backlight is on so that parts can be captured properly. Additionally, ensure that the area surrounding the platform does not appear darkened out. Values need to be fine-tuned for parts that readily transmit light.

6.3.2 Vision Objects

Configure the following properties. Make sure to configure or check the following.
Leave all other properties as their default values.

Property	Configuration method
MaxArea	Leave as the default value (width × height of the camera).
MinArea	Set to around 0.9 times that of the parts area. Use the following procedure to confirm the size of the parts area. 1. Turn on the feeder backlight 2. Place several parts on the platform so that they do not overlap 3. Run the vision sequence 4. The parts area is the average area for Blob results
NumberToFind	Set to "All".
SearchWin	Set as close to the inner circumference of the platform as is possible. Additionally, ensure that darkened areas surrounding the platform do not enter the search window.  When rotating the search window (Select [Rotated Rectangle] in [Type]), the rotation angle should be set within +/-45 deg.
ThresholdColor	Set to Black.
ThresholdHigh	Set so that parts can be detected, and that darkened areas on the platform's periphery are not detected. Values need to be fine-tuned for parts that readily transmit light.
ThresholdLow	Set to "0".

7. How to Adjust the Hopper

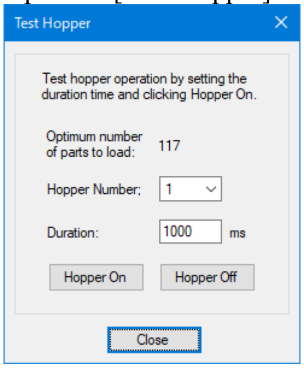
The following is an example of how to adjust the hopper.

7.1 How to Adjust

This is an example of a hopper adjustment for the IF-240, 380 and 530.

It is assumed that the installation and connection of the hopper and feeder have been completed, and feeder calibration is finished.

1. Open the [Test Hopper] dialog (see *2.3.5 Parts Supply*).



Test hopper operation by setting the duration time and clicking Hopper On.

Optimum number of parts to load: 117

Hopper Number: 1

Duration: 1000 ms

Hopper number	Select the OUT terminal 1 or 2.
Duration	Enter a duration of 10000 (10 sec) (for reference).

2. Insert parts into the hopper.
   Approx. quantity of parts to be put in: 5 to 10 times the [Optimum number of parts to load]
   The optimal number of parts is shown in the [Test Hopper] dialog.

3. Click the <Hopper On> button.
   Adjust the volume of the hopper controller to make the part move smoothly.
   When the feeder stops during adjustment, click the <Hopper On> button again.

4. Adjust the amount or condition of parts into the hopper so that feed quantity of the parts is constant.
   In some cases, it is a good idea to attach a divider to the hopper outlet.

5. Repeat steps 2-4 to adjust.

6. Modify the program.
   Refer to following command example in *3. Part Feeding SPEL+ Command Reference*.

   PF_OutputOnOff
   PF_Output
   PF_QtyAdjHopperTime

7.2 How to Adjust the IF-80 Hopper

The IF-80 has a built-in hopper. The following is an example of how to adjust this hopper. It is assumed that the installation and connection of the hopper and feeder have been completed.

1. Run the hopper caolbration. (see *2.4.13 Hopper - Test & Adjust (for IF-80)*)
2. Modify the program.
   Refer to following command example in *3. Part Feeding SPEL+ Command Reference*.

   PF_OutputOnOff
   PF_QtyAdjHopperTime

8. Errors that Occur While Using EPSON RC+

Message	Cause/solution
There are no cameras in the system. You can add cameras from [Setup] – [System Configuration] – [Vision].	A camera has not been registered to the system. Register the camera.
Part Feeding is not supported for virtual controllers.	The Part Feeding option does not support virtual controllers. Connect to the Controller.
Part Feeding Option is not installed or enabled. Check [Setup] – [Options].	The Part Feeding option has not been enabled. This option requires a separate fee. Please purchase an option key code from one of our distributors and perform the setup process.
There are no part feeders enabled in System Configuration.	A feeder has not been registered to the system. Alternatively, the feeder has not been enabled. Register or enable the feeder.
The calibration vision sequence has not been specified.	The calibration vision sequence has not been specified. Specify the calibration vision sequence on the Vision page of the Part Feeding dialog.
No more parts can be added.	Up to 32 part types can be registered on a single project. Either delete parts that are not in use, or overwrite parameters on parts not in use to keep using them.
Invalid vision calibration for calibration vision sequence. A robot camera calibration must be specified for the sequence.	Vision calibration has not been configured for the parts detection vision sequence or the feeder calibration vision sequence. Configure a valid vision calibration for Calibration property in each vision sequence.
Calibration Error: Too many parts.	The number of parts to feed is too large during feeder calibration. Or, the part is falsely detected due to improper vision settings. Input the correct number of parts displayed on the screen. Alternatively, review vision settings to make changes.
Calibration Error: Not enough parts.	The number of parts to feed is too small during feeder calibration. Or, the part is not detected due to improper vision settings. Input the correct number of parts displayed on the screen. Alternatively, review vision settings to make changes.
Part could not be found.	No parts have been feeded during feeder calibration. Or, the part is not detected due to improper vision settings. Input the correct number of parts displayed on the screen. Alternatively, review vision settings to make changes.
Failed to open as client for the Ethernet port.	1) Feeder connection disconnected. Check that the Ethernet connection between the feeder and the Controller is functioning normally (have cables become disconnected, is there a hub failure or a lack of power supply to the hub, etc.). Check the power supply to the feeder. 2) Settings for communication with feeder (Feeder model, IP address, Subnet mask, Port) is incorrect. Review settings to make changes.
Cannot connect with part feeder x	Feeder connection disconnected. Check that the Ethernet connection between the feeder and the Controller is functioning normally (have cables become disconnected, is there a hub failure or a lack of power supply to the hub, etc.). Check the power supply to the feeder.
An undefined function was specified.	Close the Part Feeding window and try rebuilding the project.

Message	Cause/solution
The controller could not connect with the part feeder using the current settings.	Settings for communication with feeder (Feeder model, IP address, Subnet mask, Port) is incorrect. Review settings to make changes.
To support Part Feeding, this version of EPSON RC+ 7.0 requires that the controller firmware must be x or greater.	The controller version does not match the RC + version. Update the firmware of the controller.
Part Feeding Part x was configured for use with part feeder x, model x, but feeder x in the controller is model x. The feeder model for the part will be changed and the part will need to be re-calibrated. Continue?	Unconformity arised with feeder setting of part because feeder model has been changed at system settings. Change feeder model correctly at system settings or operate calibration again for the parts.
The part feeder firmware version (x) is not compatible with this version of EPSON RC+ 7.0. The part feeder firmware version for model x must be x or greater.	1) Part Feeding is not supporting this firmware version of this feeder. Update the firmware of the feeder. 2) Other manufacturers feeder is connected. Connect the feeder purchased from us.
One or more Part Feeding parts uses a Compact Vision unit with firmware x. Compact Vision firmware version must be x or greater for use with Part Feeding.	1) When using CV with Part Feeding, CV firmware version needs to be 3.0.0.0 or greater. Update firmware version of CV. 2) When using CV with Part Feeding, use CV2-SA, CV2-HA. CV1 or CV2-S/H/L is not supported.

9. Application Programming Examples

This chapter contains application programming examples for various scenarios.

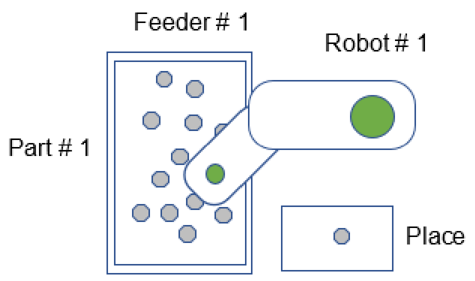
9.1 One Robot Per Feeder & One Part Per Feeder

9.1.1 Program Example 1.1

Example Type:
Using the PF_Robot Callback for motion

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

Parts are removed from the feeder and placed into a box. When all the parts have been removed, the Control callback will request more parts. When the operator or hopper replenishes the feeder with more parts, the cycle continues. If the PF_Stop command is executed, the current cycle ends and then the application will terminate. (PF_Stop command is not used in the following sample code.)

Sample Code

<u>**Main.prg**</u>

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

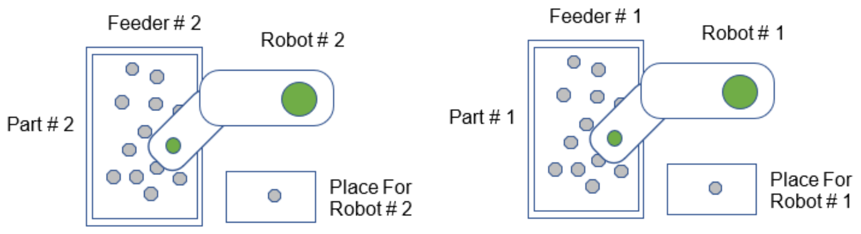
9.1.2 Program Example 1.2

Example Type:

Multiple robot system – 1 robot per feeder & 1 part per feeder

Configuration

Number of Robots: 2. Robot 1 is connected to the Control Unit and Robot 2 is using a Drive Unit
Number of Feeders: 2
Number of Parts Types on each Feeder: 1
Number of Placement Positions: 1 per robot
Camera Orientation: Each Feeder has its own Fixed Downward Camera



Description

Both robots independently pick and place parts. The robots do not share feeders or placement position. The robot work envelopes do not overlap. Both robots have points that are labeled “Park”, “Pick” and “Place” in their respective point files.

Sample Code

Main.prg

```
Function main
    Robot 1
    If Motor = Off Then Motor On
    Power Low
    Jump Park

    Robot 2
    If Motor = Off Then Motor On
    Power Low
    Jump Park

    PF_Start 1
    PF_Start 2
Fend
```

PartFeeding.prg

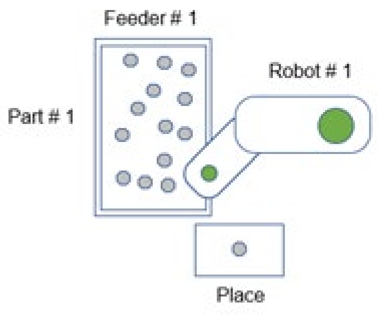
```
Function PF_Robot(PartID As Integer) As Integer
   Integer gripperOutput

   Select PartID
      Case 1
         Robot 1
         gripperOutput =1
           Case 2
         Robot 2
         gripperOutput =2
   Send

   Do While PF_QueLen(PartID) > 0
      Pick = PF_QueGet(PartID)
      Jump Pick
      On gripperOutput; Wait 0.2;
      Jump Place
      Off gripperOutput; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

9.1.3 Program Example 1.3

Example Type:
Parallel processing vision & vibration with robot motion

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera



Description

The robot picks up a Part #1 from the parts feeder and places it into a fixture. This continues until all the "Front" parts are removed from the Feeder. When the last available part is being placed (90% of the way through the motion), the feeder will vibrate, the hopper will replenish the feeder if necessary, etc
This example demonstrates how the feeder action can occur in parallel with the robot motion to optimize throughput.

Sample Code

Main.prg

```
Function main
   MemOff PartsToPick
   Off Gripper

   Robot 1
   If Motor = Off Then Motor On
   Power Low
   Jump Park

   PF_Start(1)

   Xqt RobotPickPlace
Fend
```

```
Function RobotPickPlace
   DO
      Wait MemSw(PartsToPick) = On
      If PF_QueLen(1) > 0 Then
      Do
            Pick = PF_QueGet(1)
            PF_QueRemove 1
            Jump Pick
            On Gripper; Wait 0.2
            If PF_QueLen(1) = 0 Then
               Jump Place ! D90; MemOff PartsToPick!
               Off Gripper; Wait 0.2
               Exit Do
            Else
               Jump Place
               Off Gripper; Wait 0.2
            EndIf
         Loop
      Else
         MemOff PartsToPick
      EndIf
   Loop
Fend
```

PartFeeding.prg

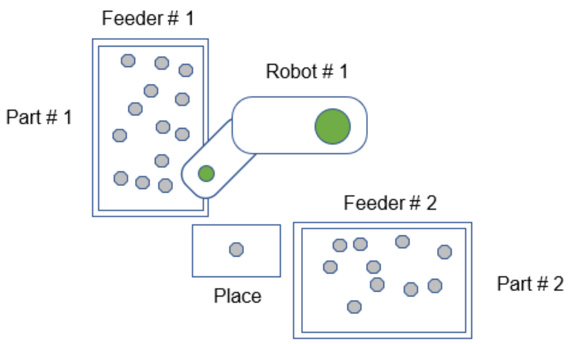
```
Function PF_Robot(PartID As Integer) As Integer
   MemOn PartsToPick
   Wait MemSw(PartsToPick) = Off

   PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

9.1.4 Program Example 1.4

Example Type:
1 Robot, 2 Feeders and 1 Part Per Feeder

Configuration

Number of Robots: 1
Number of Feeders: 2
Number of Parts Types on each Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Each Feeder has its own Fixed Downward Camera



Description

The robot picks up a Part #1 from Feeder #1 and places it into a box. This continues until all the "Front" parts are removed from Feeder #1. Then the robot picks up each "Front" part from Feeder #2 and places them into the box. The Cameras and Feeders work in parallel with one another.

Sample Code

Main.prg

```
Function main
    MemOff PartsToPick1; MemOff PartsToPick2
    Off Gripper

    Robot 1
    If Motor = Off Then Motor On
    Power Low
    Jump Park

    PF_Start(1)
    PF_Start(2)

    Xqt rbt1
Fend
```

```
Function rbt1
   Do
      Call RobotPickPlace(1)
      Call RobotPickPlace(2)
   Loop
Fend

Function RobotPickPlace(PartID As Integer)
   Integer partsToPickMembit,

     Select PartID
      Case 1
         partsToPickMembit = IONumber("PartsToPick1")
           Case 2
         partsToPickMembit = IONumber("PartsToPick2")
   Send

   Wait MemSw(partsToPickMembit) = On
     If PF_QueLen(PartID) > 0 Then

      Do
         Pick = PF_QueGet(PartID)
         PF_QueRemove PartID
         Jump Pick
         On Gripper; Wait 0.2
         If PF_QueLen(PartID) = 0 Then
            Jump Place ! D90; MemOff partsToPickMembit !
            Off Gripper; Wait 0.2
            Exit Do
         Else
            Jump Place
            Off Gripper; Wait 0.2
         EndIf

         If PF_IsStopRequested(PartID) = True Then
            MemOff partsToPickMembit
                    Exit Do
         EndIf
      Loop
   Else
      MemOff partsToPickMembit
   EndIf
Fend
```

PartFeeding.prg

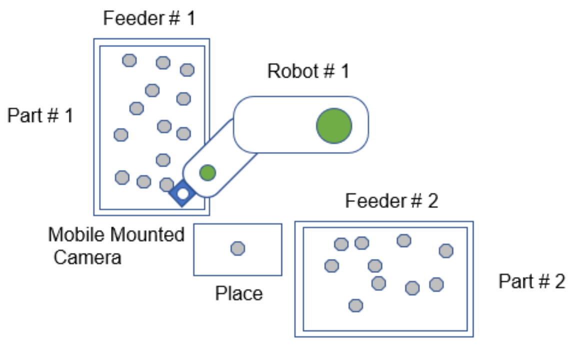
```
Function PF_Robot(PartID As Integer) As Integer

   Select PartID
      Case 1
         MemOn PartsToPick1
         Wait MemSw(PartsToPick1) = Off
      Case 2
         MemOn PartsToPick2
         Wait MemSw(PartsToPick2) = Off
   Send

   PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

9.1.5 Program Example 1.5

Example Type:
1 Robot, 2 Feeders and 1 Part Per Feeder – Mobile Mounted Camera

Configuration

Number of Robots: 1
Number of Feeders: 2
Number of Parts Types on each Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Each Feeder will use the Mobile Mounted Camera on the robot
(there are no Fixed Downward cameras for this example)



Description

The robot moves the Mobile Mounted camera over Feeder #1 and takes a picture. The robot picks up each part from Feeder #1 and places it into a box. This continues until all the "Front" parts are removed from Feeder #1. Then the robot moves the Mobile Mounted camera over Feeder #2 and takes a picture. The robot picks up each correctly oriented part from Feeder #2 and places them into the box.

NOTE ☞ When using a mobile camera with multiple feeders which have backlights, use separate vision sequences for parts associated with each feeder.

Sample Code

<u>Main.prg</u>

```
Function main
    MemOff PartsToPick1; MemOff PartsToPick2;
    MemOff mobileCamBefore1; MemOff mobileCamAfter1
    MemOff mobileCamBefore2; MemOff mobileCamAfter2
    MemOff mobileCamInPos1; MemOff mobileCamInPos2

    Robot 1
    If Motor = Off Then Motor On
    Power Low
    Jump Park

    PF_Start(1)
    PF_Start(2)
    Xqt rbt1
Fend
```

```
Function rbt1

   Do
      Wait MemSw(mobileCamBefore1) = On
      Jump MobileCamShotFeeder1; MemOn mobileCamInPos1
      Wait MemSw(mobileCamAfter1) = On
      MemOff mobileCamInPos1

      Call RobotPickPlace(1)

      Wait MemSw(mobileCamBefore2) = On
      Jump MobileCamShotFeeder2; MemOn mobileCamInPos2
      Wait MemSw(mobileCamAfter2) = On
      MemOff mobileCamInPos2

      Call RobotPickPlace(2)
   Loop
Fend

Function RobotPickPlace(PartID As Integer)
   Integer partsToPickMembit, partCnt, gripperOutput, toolNum

     Select PartID
      Case 1
         partsToPickMembit = IONumber("PartsToPick1")
      Case 2
         partsToPickMembit = IONumber("PartsToPick2")
   Send

   Wait MemSw(partsToPickMembit) = On
   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   MemOff partsToPickMembit
Fend
```

PartFeeding.prg

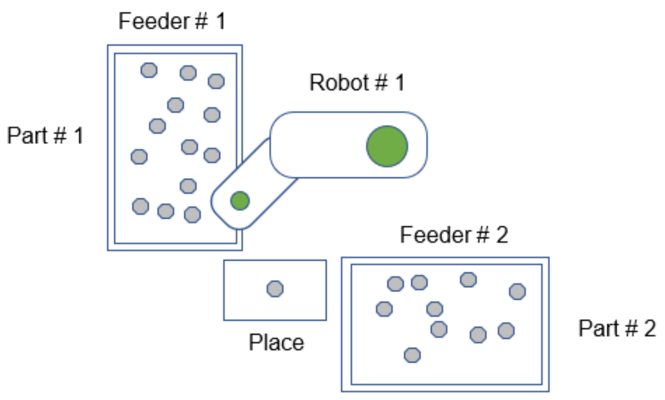
```
Function PF_Robot(PartID As Integer) As Integer

   Select PartID
      Case 1
         MemOn PartsToPick1
         Wait MemSw(PartsToPick1) = Off
      Case 2
         MemOn PartsToPick2
         Wait MemSw(PartsToPick2) = Off
   Send

   PF_Robot = PF_CALLBACK_SUCCESS
Fend

Function PF_MobileCam(PartID As Integer, Action As Integer) As Integer
Integer mobileCamBeforeMembit, mobileCamAfterMembit, mobileCamInPosMembit

   Select PartID
      Case 1
         mobileCamBeforeMembit = IONumber("mobileCamBefore1")
         mobileCamAfterMembit = IONumber("mobileCamAfter1")
         mobileCamInPosMembit = IONumber("mobileCamInPos1")
      Case 2
         mobileCamBeforeMembit = IONumber("mobileCamBefore2")
         mobileCamAfterMembit = IONumber("mobileCamAfter2")
         mobileCamInPosMembit = IONumber("mobileCamInPos2")
   Send

   Select Action
      Case PF_MOBILECAM_BEFORE
         ' Request for robot move to camera position
         MemOff mobileCamAfterMembit
         MemOn mobileCamBeforeMembit
         Wait MemSw(mobileCamInPosMembit) = On
      Case PF_MOBILECAM_AFTER
         ' Request for robot move after part vision acquisition
         MemOff mobileCamBeforeMembit
         MemOn mobileCamAfterMembit
         Wait MemSw(mobileCamInPosMembit) = Off
   Send

   PF_MobileCam = PF_CALLBACK_SUCCESS
Fend
```

9.1.6 Program Example 1.6

Example Type:
Specifying the number of parts to pick

Configuration

Number of Robots: 1
Number of Feeders: 2
Number of Parts Types on each Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Each Feeder has its own Fixed Downward Camera



Description

The robot picks up four Part #1 from Feeder #1 and place them individually. The robot will then pick up five Part #2 from Feeder #2 and place them. The Cameras and Feeders work in parallel with one another. This example illustrates how the robot can pick up a specific number of parts from each feeder. The sample code also supports picking all available parts by setting the numToPick parameter to ALL_AVAILABLE. If “Front” parts remain on either feeder after picking the desired quantity, the feeder will not vibrate unnecessarily.

Sample Code

Main.prg

```
#define ALL_AVAILABLE -1

Function main
   MemOff PartsToPick1; MemOff PartsToPick2
   Off Gripper

   Robot 1
   If Motor = Off Then Motor On
   Power Low
   Jump Park

   PF_Start(1)
   PF_Start(2)

   Xqt rbt1
Fend
```

```
Function rbt1

      Do

       Call RobotPickPlace(1, 4)  'part 1...pick & place 4 times
       Call RobotPickPlace(2, 5)  'part 2...pick & place 5 times
   Loop
Fend

Function RobotPickPlace(PartID As Integer, numToPick As Integer)
   Integer partsToPickMembit, partCnt,

      Select PartID
       Case 1
          partsToPickMembit = IONumber("PartsToPick1")
       Case 2
          partsToPickMembit = IONumber("PartsToPick2")
   Send

   partCnt = 0
   Do
      Wait MemSw(partsToPickMembit) = On
      If PF_QueLen(PartID) > 0 Then
         Pick = PF_QueGet(PartID)
         PF_QueRemove PartID
         Jump Pick
         On Gripper; Wait 0.2
         partCnt = partCnt + 1
         If PF_QueLen(PartID) = 0 Then
            Jump Place ! D90; MemOff partsToPickMembit !
            Off Gripper; Wait 0.2
            If (partCnt = numToPick) Or (numToPick = ALL_AVAILABLE) Then
               Exit Do
            EndIf
         Else
            Jump Place
            Off Gripper; Wait 0.2
            If (partCnt = numToPick) Then
               Exit Do
            EndIf
         EndIf
      Else
         MemOff partsToPickMembit
      EndIf
      If PF_IsStopRequested(PartID) = True Then
         MemOff partsToPickMembit
         Exit Do
      EndIf
   Loop
Fend
```

PartFeeding.prg

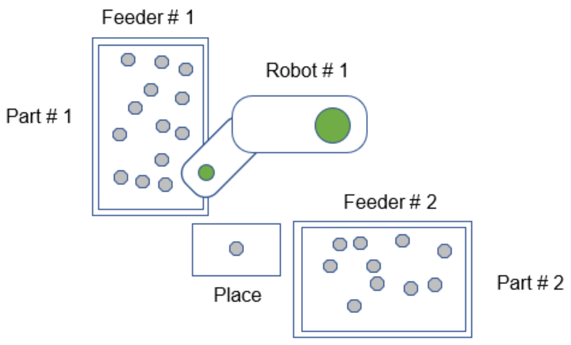
```
Function PF_Robot(PartID As Integer) As Integer

   Select PartID
      Case 1
         MemOn PartsToPick1
         Wait MemSw(PartsToPick1) = Off
      Case 2
         MemOn PartsToPick2
         Wait MemSw(PartsToPick2) = Off
   Send

   PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

9.1.7 Program Example 1.7

Example Type:
Ensuring that the Robot matches the Robot # that was used for the Part

Configuration

Number of Feeders: More than 1
Number of Parts Types on each Feeder: 1
Camera Orientation: Each Feeder has its own Fixed Downward Camera



Description

When using multiple feeders, it is common that the robot makes motion in a multitask rather than inside the PF_Robot callback. In other words, the PF_Robot callback simply loads part locations (points) into the Part Feeding Queue. A robot motion task uses the points in the queue. When the code is structured in this fashion, it is recommended that the robot motion task verifies that current robot # matches the robot # that was selected for the part. This additional check will ensure that the robot does not use the wrong points and cannot crash the robot.

Sample Code

<u>**Main.prg**</u>

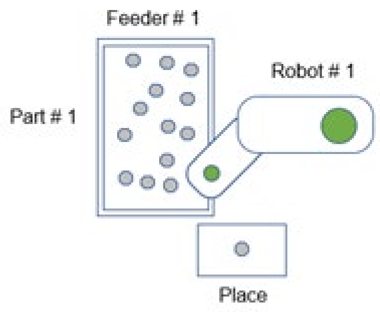
```
Function RobotPickPlace(PartID As Integer, numToPick As Integer)

    If PF_Info(PartID, PF_INFO_ID_ROBOT_NO) <> Robot Then
        Print "Robot does not match the robot # for the current part"
        Quit All
    EndIf

    'Robot Motion Code

Fend
```

9.1.8 Program Example 1.8

Example Type:
Acquiring a new image and loading the queue after every pick

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera



Description

The robot picks up a Part #1 from the parts feeder and places it into a fixture. After each pick and place operation, a new image will be acquired and the queue will be reloaded. For this example, we are concerned that surrounding parts may be disturbed during pick up. The PF_Robot callback return value "PF_CALLBACK_RESTART" will force vision to re-run for all parts and all part queues will be reloaded. This method is not efficient but "PF_CALLBACK_RESTART" can be useful in certain circumstances where acquiring an image every cycle can improve performance accuracy. The vision and feeder vibration occurs in parallel with robot motion.

Sample Code

Main.prg

```
Function main
    MemOff PartsToPick
    Off Gripper

    Robot 1
    If Motor = Off Then Motor On
    Power Low
    Jump Park

    PF_Start(1)

    Xqt RobotPickPlace
Fend
```

```
Function RobotPickPlace
   Do
      Wait MemSw(PartsToPick) = On
      If PF_QueLen(1) > 0 Then
         Pick = PF_QueGet(1)
         PF_QueRemove 1
         Jump Pick
         On Gripper; Wait 0.2
         Jump Place ! D90; MemOff PartsToPick!
         Off Gripper; Wait 0.2
      Else
         MemOff PartsToPick
      EndIf
   Loop
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   MemOn PartsToPick
   Wait MemSw(PartsToPick) = Off

   PF_Robot = PF_CALLBACK_RESTART
Fend
```

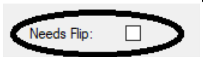
9.1.9 Program Example 1.9

Example Type:
Sorting a Part by Front and Back orientation

Configuration

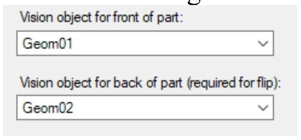
Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 2 (one for the Front side and one for the Back side)
Camera Orientation: Fixed Downward Camera over Feeder #1

Part 1 General Page:

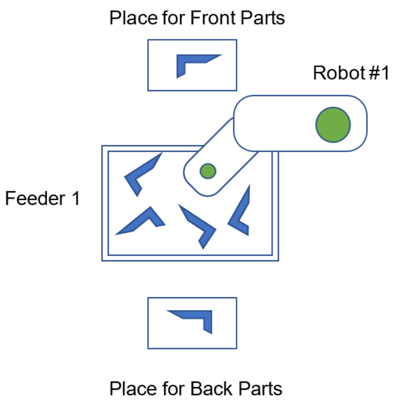


Needs Flip: ☐

Part 1 Vision Page:



Vision object for front of part:
Geom01
Vision object for back of part (required for flip):
Geom02



Description

For this application, the robot will sort the parts based upon their Front and Back orientation. Front side parts will be placed at a point labeled "PlaceFront" and Back side parts will be placed in another point labeled "PlaceBack". The pick order does not matter for this application – the robot will pick / place as many Front parts as it can and then pick / place as many Back parts as it can. When "Needs Flip" is uncheck but vision objects have been selected in "Vision object for front of part" and "Vision object for back of part", the system will load both the Front & Back parts into the coordinate queue and set the Part Orientation value accordingly. Remember that the "Needs Flip" setting tells the system that you want parts in a certain orientation. By Unchecking this setting, you are telling the system that you want both orientations.

In this case, the Front parts will automatically have their orientation data (in the part coordinate queue) set to constant "PF_PARTORIENT_FRONT". The Back parts will automatically have their orientations set to constant "PF_PARTORIENT_BACK".

Sample Code

Main.prg

```
Function main
   If Not Motor = On Then
      Motor On
   EndIf
   Power Low
   Jump park
   PF_Start 1
Fend
```

PartFeeding.prg

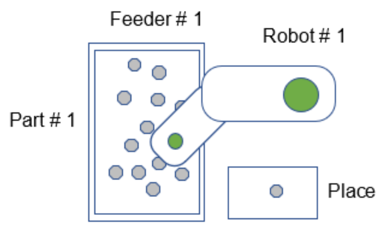
```
Function PF_Robot(PartID As Integer) As Integer
   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      If PF_QuePartOrient(PartID) = PF_PARTORIENT_FRONT Then
         Jump PlaceFront
      ElseIf PF_QuePartOrient(PartID) = PF_PARTORIENT_BACK Then
         Jump PlaceBack
      EndIf
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

9.1.10 Program Example 1.10

Example Type:
Using the PF_Vision Callback

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

This example illustrates how to use the PF_Vision callback to acquire an image and load the parts coordinate queue with the vision results. To use this function, select "User processes vision for part via PF_Vision callback" in EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] - [Vision].

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

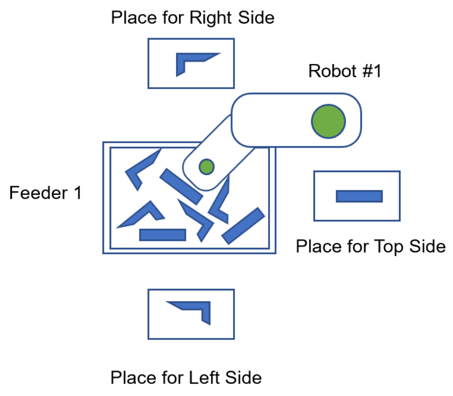
```
Function PF_Robot(PartID As Integer) As Integer
   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend

Function PF_Vision(PartID As Integer, ByRef numBack As Integer) As Integer
   Boolean found
   Integer i, numFront
   Real RB_X, RB_Y, RB_U, RB_Z

   ' Pick Z coordinate
   RB_Z = -132.0

   ' Initialize coordinates queue
   PF_QueRemove PartID, All

    PF_Backlight 1, On

   ' Detect the parts
   VRun UsrVisionSeq

   PF_Backlight 1, Off

   VGet UsrVisionSeq.Geom01.NumberFound, numFront 'Front Parts
   VGet UsrVisionSeq.Geom02.NumberFound, numBack  'Back Parts
   If numFront <> 0 Then
      For i = 1 To numFront
         VGet UsrVisionSeq.Geom01.RobotXYU(i), found, RB_X, RB_Y, RB_U
         If found Then
            PF_QueAdd PartID, XY(RB_X, RB_Y, RB_Z, RB_U)
         EndIf
      Next
   EndIf

   PF_Vision = PF_CALLBACK_SUCCESS

Fend
```

9.1.11 Program Example 1.11

Example Type:
Picking a Multi-sided Part

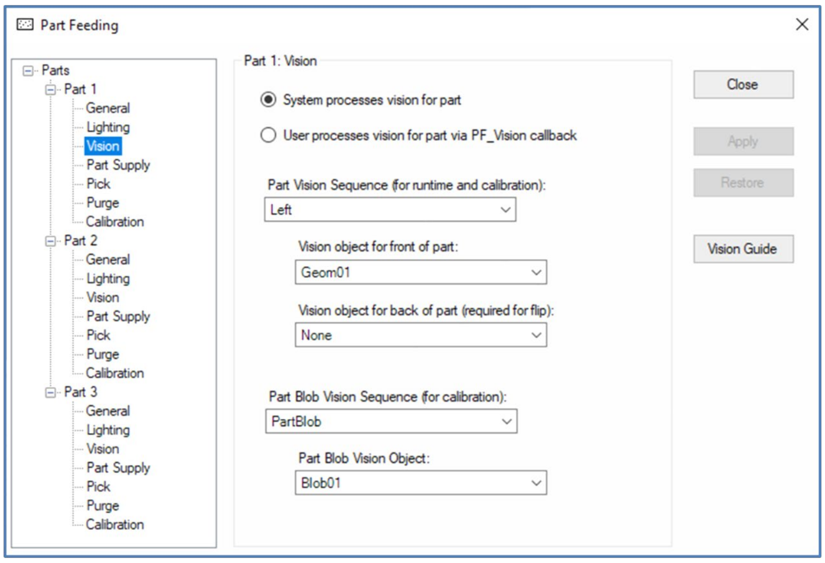
Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Unique Sides on the Part: 3
Number of Placement Positions: 3 (one placement position for each side of the part)
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

For this example, there is 1 physical part type on the feeder. The part has 3 unique sides (Right side, Left side and Top side). The robot can pick up the part in any one of the 3 orientations. From the perspective of the vision system, each side is a different part (even though it is actually 1 physical part). Part Feeding supports 4 unique parts running on the same feeder at the same time. Consequently, we are going to create a separate Part for each side. We will make 3 Part Vision sequences and name them “Left”, “Right” and “Top”. Each vision sequence will use a Geometric object to locate the part in its specific orientation. In the Part Feeding dialog, we will create 3 Parts - Part 1, 2 and 3. Part 1 will locate the part in the “Left” side orientation. Part 2 will locate the part in the “Right” side orientation. Part 3 will locate the part in the “Top” side orientation. “Needs Flip” will be unchecked for all 3 Parts. All 3 parts will use the same Part Blob Vision Sequence called “PartBlob”.



Part 1: Vision

(●) System processes vision for part
( ) User processes vision for part via PF_Vision callback

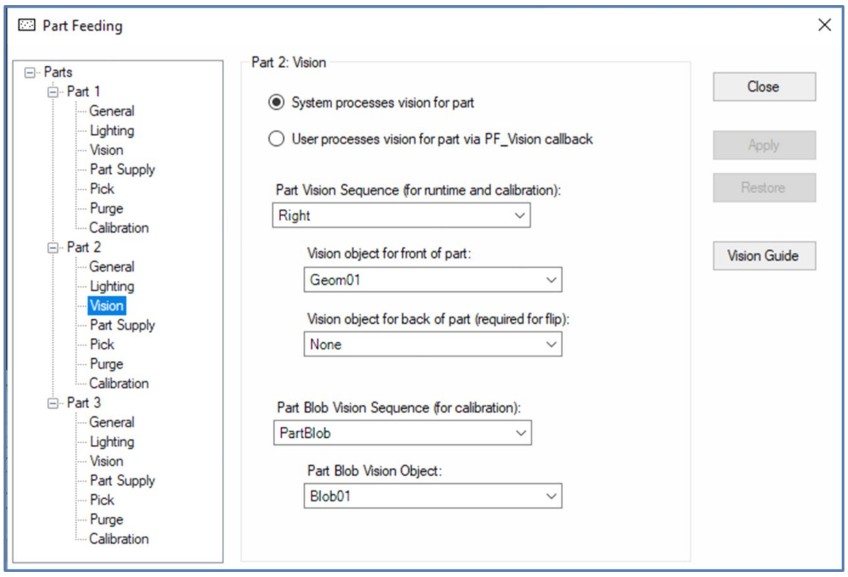
Part Vision Sequence (for runtime and calibration): Left

Vision object for front of part: Geom01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): PartBlob

Part Blob Vision Object: Blob01



Part 2: Vision

(●) System processes vision for part
( ) User processes vision for part via PF_Vision callback

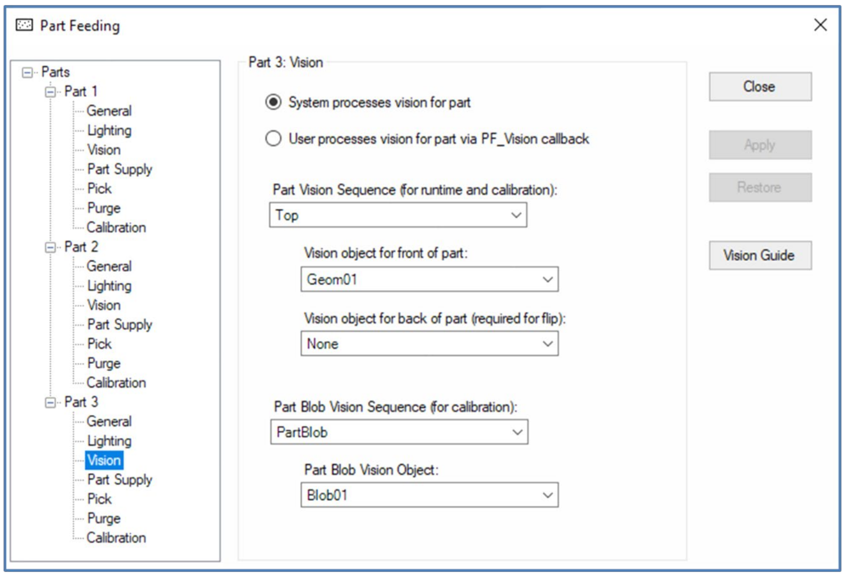
Part Vision Sequence (for runtime and calibration): Right

Vision object for front of part: Geom01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): PartBlob

Part Blob Vision Object: Blob01



Part 3: Vision

(●) System processes vision for part
( ) User processes vision for part via PF_Vision callback

Part Vision Sequence (for runtime and calibration): Top

Vision object for front of part: Geom01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): PartBlob

Part Blob Vision Object: Blob01

Teach the Pick Z for each Part (the pick height may be different for each side of the part). Left side parts will be placed at a point labeled “PlaceLeft”. Right side parts will be placed at a point labeled “PlaceRight”. Top side parts will be placed at a point labeled “PlaceTop”. The robot will pick / place all the parts in the Left orientation, then pick / place all the parts in the Right orientation and lastly it will pick / place all the parts in the Top orientation.

Sample Code

Main.prg

```
Function main
   If Not Motor = On Then Motor On
   Power Low
   Jump park
   PF_Start 1,2,3
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer

   Do While PF_QueLen(PartID) > 0
      P10 = PF_QueGet(PartID)
      Jump P10
      On Vacuum; Wait 0.2
      Select PartID
         Case 1
            Jump PlaceLeft
         Case 2
            Jump PlaceRight
         Case 3
            Jump PlaceTop
      Send
      Off Vacuum; Wait 0.2
      PF_QueRemove PartID
   Loop

   PartID = PartID + 1
   If PartID > 3 Then PartID = 1
   PF_ActivePart PartID
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

NOTE

This example illustrates a simple method of handling a multi-side part by using the Multi-Part functionality. Another method is for the “User to Process Vision” via the PF_Vision callback. When using PF_QueAdd, User Data can be included with the part coordinates. The user data could be a numerical value that represents the part orientation (i.e., 1 = Left, 2=Right, 3=Top). The PF_QueUserData function would be used to get the orientation value so that the part can be placed in the correct location. The code would also need to account for height differences in the part sides.

9.1.12 Program Example 1.12

Example Type:
Using the PF_Vision Callback and the Part Sequence uses Multi-Search

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1

Description

The PF_Vision callback is used when the Part Sequence vision processing is difficult (i.e., the application requires multiple vison sequences to detect the part or several different lighting controls are required etc…).

This example illustrates how to use the PF_Vision callback to acquire an image and load the parts coordinate queue with the vision results. To use this function, select “User processes vision for part via PF_Vision callback” in EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] - [Vision].

This example uses multi-search for the Part Sequence. Multi-search is a feature where an object will search for each result of the CenterPointObject property or a Frame obgject. In multi-search, the found results may not be found sequentially. PF_Vision shows how to iterate through all the results to find the found results.

Example of multi-search with CenterPointObject:

1. Create an object to find multiple parts, such as a Blob
2. Create another object, such as a Polar, that will use the Blob as its CenterPointObject.
3. Set the CenterPntObjResult property to All.
4. Run the sequence. You will see an instance of the Polar object for each Blob result that was found.

Example of multi-search with Frame:

1. Create an object to find multiple parts, such as a Blob.
2. Create a Frame object and set the OriginPoint property to the Blob.
3. Set the OriginPntObjResult property to All.
4. Create another object, such as a Polar, that will use the Frame.
5. Set the FrameResult property to All.
6. Run the sequence. You will see an instance of the Frame object for each Blob result that was found, and an instance of the Polar object for each Frame result.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer

   Do While PF_QueLen(PartID) > 0
      P10 = PF_QueGet(PartID)

      Select PF_QuePartOrient(PartID)
         Case PF_PARTORIENT_FRONT ' Front
            Tool 1 ' Tool to pick Front
            ' Pick
            Jump P10 ! D90; On Vacuum !
            Wait 0.1
            ' Place
            Jump FrontPlace
            Off Vacuum
            Wait 0.1
         Case PF_PARTORIENT_BACK ' Back
            Tool 2 ' Tool to pick Back
            ' Pick
            Jump P10 ! D90; On Vacuum !
            Wait 0.1
            ' Place
            Jump BackPlace
            Off Vacuum
            Wait 0.1
      Send

      PF_QueRemove PartID

      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf

   Loop

   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

```
    Boolean found
    Real x, y, z, u

    z = -170   ' Set the pick Z coordinate to the desired height for your application

    PF_QueRemove 1, All

    PF_Backlight 1, On
    VRun FindPart

    VGet FindPart.Front.NumberFound, numFront
    VGet FindPart.Front.NumberOfResults, numResults
    count = 0
    For i = 1 To numResults
       VGet FindPart.Front.RobotXYU(i), found, x, y, u
       If found Then
          count = count + 1
          P999 = XY(x, y, z, u) /R
          PF_QueAdd 1, P999, PF_PARTORIENT_FRONT
       EndIf
       If count = numFront Then
          Exit For
       EndIf
    Next

    VGet FindPart.Back.NumberFound, numBack
    VGet FindPart.Back.NumberOfResults, numResults
    count = 0
    For i = 1 To numResults
       VGet FindPart.Back.RobotXYU(i), found, x, y, u
       If found Then
          count = count + 1
          P999 = XY(x, y, z, u) /R
          PF_QueAdd 1, P999, PF_PARTORIENT_BACK
       EndIf
       If count = numBack Then
          Exit For
       EndIf
    Next

    PF_Backlight 1, Off
    PF_Vision = PF_CALLBACK_SUCCESS

Fend
```

9.2 One Robot – Multiple Parts

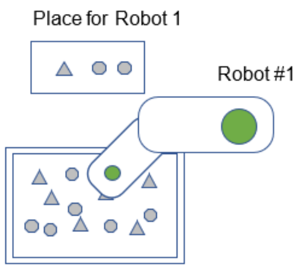
9.2.1 Program Example 2.1

Example Type:

1 Robot and Multiple Parts - Parallel processing vision & vibration with robot motion

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on Feeder: 2
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera



Description

There are two unique (physically different) parts. The robot will continuously pick and place two of Part #1 and then pick and place one of Part #2. This is accomplished by alternating "PF_ActivePart". For this application, the pick order matter (for example, in the case of a part assembly). When the last part has been placed, the "PF_ActivePart" is changed to feed the desired part and signal the system to vibration and acquire images if necessary. This is accomplished in parallel with the robot motion (30% of the way to "place").

Sample Code

Main.prg

```
Function Main
   Integer numToPick1, numToPick2, i

   Robot 1
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park

   MemOff PartsToPick1
   MemOff PartsToPick2
   numToPick1 = 2
   numToPick2 = 1

   PF_Start 1,2


   Do
      i = 0
      Do
```

```
         Wait MemSw(PartsToPick1) = On
         P0 = PF_QueGet(1)
         PF_QueRemove (1)
         Jump P0 /R
         On Gripper
         Wait .25
         i = i + 1
         If i < numToPick1 And PF_QueLen(1) > 0 Then
            Jump place
         Else
            'Last part or no more parts available to pick
            If i = numToPick1 Then
               PF_ActivePart 2
            EndIf
            Jump place ! D30; MemOff PartsToPick1 !
         EndIf
         Off Gripper
         Wait .25
       Loop Until i = numToPick1
      i = 0
      Do
         Wait MemSw(PartsToPick2) = On
         P0 = PF_QueGet(2)
         PF_QueRemove (2)
         Jump P0 /R
         On Gripper
         Wait .25
         i = i + 1
         If i < numToPick2 And PF_QueLen(2) > 0 Then
            Jump place
         Else
            'Last part or no more parts available to pick
            If i = numToPick2 Then
               PF_ActivePart 1
            EndIf
            Jump place ! D30; MemOff PartsToPick2 !
         EndIf
         Off Gripper
         Wait .25
      Loop Until i = numToPick2
   Loop
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Select PartID
      Case 1
         MemOn PartsToPick1
         Wait MemSw(PartsToPick1) = Off
      Case 2
         MemOn PartsToPick2
         Wait MemSw(PartsToPick2) = Off
   Send

   PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

9.3 Two Robots – One Part

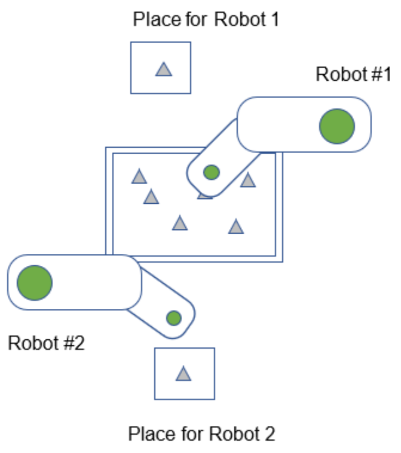
9.3.1 Program Example 3.1

Example Type:

2 Robots & 1 Physical Part Type – Motion in PF_Robot callback – Picking in a specific order

Configuration

Number of Robots: 2
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 2
Camera Orientation: Fixed Downward Camera



Description

There are two robots and one feeder. There is only 1 physical part type. Because each robot has its own camera calibration, there are two logical Parts – Part 1 for Robot 1 and Part 2 for Robot 2. The robots will take turns picking from the feeder. The pick order matters for this application. The alternating pick order is accomplished with "PF_ActivePart". Robot motion is performed inside the PF_Robot callback. This example does not have parallel processing of the feeder and robot motion. The code is simple but not efficient. Each robot has a point labeled "park" and a point labeled "place".
The key concept of this example is the PF_Robot callback return value "PF_CALLBACK_RESTART_ACTIVEPART". This return value allows multiple robots to use the same feeder without the potential of the part coordinates in both Part's queues. The return value forces a new image to be acquired for only the PF_ActivePart and only the PF_ActivePart's queue is loaded.

Sample Code

Main.prg

```
Function Main
   Robot 1
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park
   Robot 2
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park

   PF_Start 1, 2
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   If PF_QueLen(PartID) > 0 Then
      Select PartID
         Case 1
            Robot 1
            P0 = PF_QueGet(1)
            PF_QueRemove (1)
            Jump P0 /R
            On rbt1Gripper
            Wait .25
            Jump place
            Off rbt1Gripper
            Wait .25
            PF_ActivePart 2
         Case 2
            Robot 2
            P0 = PF_QueGet(2)
            PF_QueRemove (2)
            Jump P0 /L
            On rbt2Gripper
            Wait .25
            Jump place
            Off rbt2Gripper
            Wait .25
            PF_ActivePart 1
      Send

   EndIf

   PF_Robot = PF_CALLBACK_RESTART_ACTIVEPART

Fend
```

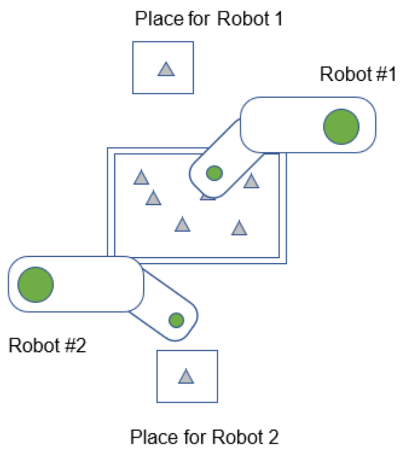
9.3.2 Program Example 3.2

Example Type:

2 Robots & 1 Physical Part Type – motion in separate tasks – pick order does not matter – Picking in a specific order

Configuration

Number of Robots: 2
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 2
Camera Orientation: Fixed Downward Camera



Description

There are two robots and one feeder. There is only 1 physical part type. Because each robot has its own camera calibration, there are two logical Parts – Part 1 for Robot 1 and Part 2 for Robot 2. Pick order does not matter - first come, first served. What makes this example different is that vision is acquired for each part every cycle. This may be helpful if you are concerned that surrounding parts may be disturbed during pick up. The PF_Robot callback return value “PF_CALLBACK_RESTART” will force vision to re-run for all parts and all part queues will be reloaded.
This method is not efficient but “PF_CALLBACK_RESTART” can be useful in certain circumstances.

Sample Code

Main.prg

```
Function Main
   Robot 1
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park
   Robot 2
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park
   MemOff PartsToPick
   PF_Start 1,2
   Xqt Robot1PickPlace
   Xqt Robot2PickPlace
Fend

Function Robot1PickPlace
   Robot 1

   Do
      PF_AccessFeeder(1)
      Wait MemSw(PartsToPick) = On
      If PF_QueLen(1) > 0 Then
         P0 = PF_QueGet(1)
         PF_QueRemove (1)
         Jump P0 /R
         On 5
         Wait .5
         Jump place ! D30; MemOff PartsToPick; PF_ReleaseFeeder 1!
         Off 5
         Wait .25
      Else
         MemOff PartsToPick; PF_ReleaseFeeder 1
      EndIf
   Loop
Fend

Function Robot2PickPlace
   Robot 2

   Do
      PF_AccessFeeder(1)
      Wait MemSw(PartsToPick) = On
      If PF_QueLen(2) > 0 Then
         P0 = PF_QueGet(2)
         PF_QueRemove (2)
         Jump P0 /L
         On 2
         Wait .5
         Jump place ! D30; MemOff PartsToPick; PF_ReleaseFeeder 1!
         Off 2
         Wait .25
      Else
         MemOff PartsToPick; PF_ReleaseFeeder 1
      EndIf
   Loop
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   MemOn PartsToPick
   Wait MemSw(PartsToPick) = Off


   PF_Robot = PF_CALLBACK_RESTART 'Force vision and vibration to refresh
Fend
```

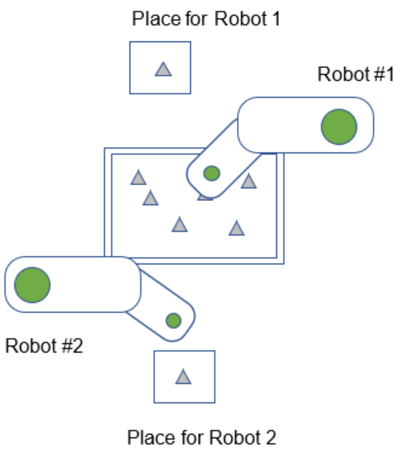
9.3.3 Program Example 3.3

Example Type:

2 Robots & 1 Physical Part Type – motion in separate tasks – First Come, First Served - Simulated process delay

Configuration

Number of Robots: 2
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 2
Camera Orientation: Fixed Downward Camera



Description

There are two robots and one feeder. There is only 1 physical part type. Because each robot has its own camera calibration, there are two logical Parts – Part 1 for Robot 1 and Part 2 for Robot 2. For this example, each robot has a variable process time (simulated by a random wait time). Each robot is busy performing some other operation after picking up a part from the feeder. Memory bits "Rbt1Complete" and "Rbt2Complete" are used to signal when the robot has finished its secondary operation and it is now ready to pickup another part from the feeder. The PF_Robot callback will return the value "PF_CALLBACK_RESTART_ACTIVEPART" if the desired part (PF_ActivePart) is not the same as the current part (i.e., the other robot wants to pick up a part). This will prevent duplication of points in the robot queues. A new image will be acquired for the PF_ActivePart and only the PF_ActivePart's queue will be loaded. However, if the next part is the same as the current part (i.e, the same robot is going to pick from the feeder) then the PF_Robot callback return value will be "PF_CALLBACK_SUCCESS". PF_AccessFeeder and PF_ReleaseFeeder ensure that the robots will not collide when accessing the feeder.

Sample Code

Main.prg

```
Function Main
   Robot 1
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump place
   Robot 2
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump place
   MemOff PartsToPick1
   MemOff PartsToPick2

   PF_Start 1,2
   Xqt Robot1PickPlace
   Xqt Robot2PickPlace
Fend

Function Robot1PickPlace
   Integer randomTime

   Robot 1
   MemOn Rbt1Complete

   Do
      Wait MemSw(PartsToPick1) = On
      PF_AccessFeeder(1)
      MemOff Rbt1Complete
      P0 = PF_QueGet(1)
      PF_QueRemove (1)
      Jump P0 /R
      On rbt1Gripper
      Wait .25
      Jump place ! D30; MemOff PartsToPick1; PF_ReleaseFeeder 1!
      Off rbt1Gripper
      Wait .25
      'Test long process time - robot is doing something else
      Randomize
      randomTime = Int(Rnd(9)) + 1
      Wait randomTime
      MemOn Rbt1Complete
   Loop
Fend
```

```
Function Robot2PickPlace
   Integer randomTime

   Robot 2
   MemOn Rbt2Complete

   Do
      Wait MemSw(PartsToPick2) = On
      PF_AccessFeeder(1)
      MemOff Rbt2Complete
      P0 = PF_QueGet(2)
      PF_QueRemove (2)
      Jump P0 /L
      On rbt2Gripper
      Wait .25
      Jump place ! D30; MemOff PartsToPick2; PF_ReleaseFeeder 1!
      Off rbt2Gripper
      Wait .25
      'Test long process time - robot is doing something else
      Randomize
      randomTime = Int(Rnd(9)) + 1
      Wait randomTime
      MemOn Rbt2Complete
   Loop
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Integer nextPart

   Select PartID
      Case 1
         MemOn PartsToPick1
         Wait MemSw(PartsToPick1) = Off
      Case 2
         MemOn PartsToPick2
         Wait MemSw(PartsToPick2) = Off
   Send

   Wait MemSw(Rbt1Complete) = On Or MemSw(Rbt2Complete) = On
   If MemSw(Rbt1Complete) = On Then
      nextPart = 1
   ElseIf MemSw(Rbt2Complete) = On Then
      nextPart = 2
   EndIf

   PF_ActivePart nextPart

   If nextPart = PartID Then
      'Same part so no need to re-acquire an image and reload the queue
      PF_Robot = PF_CALLBACK_SUCCESS
   Else
      'Restart from vision -
      'Acquire image and load queue for only the Active Part
      PF_Robot = PF_CALLBACK_RESTART_ACTIVEPART
   EndIf

Fend
```

9.4 Two Robots – Multiple Parts

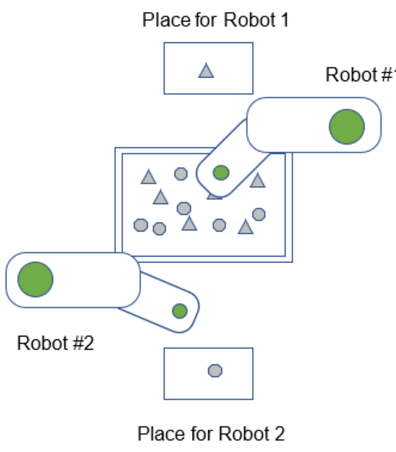
9.4.1 Program Example 4.1

Example Type:

2 Robots, 1 Feeder, Multiple Parts – Motion in PF_Robot callback – Picking in a specific order

Configuration

Number of Robots: 2
Number of Feeders: 1
Number of Parts Types on Feeder: 2
Number of Placement Positions: 2
Camera Orientation: Fixed Downward Camera



Description

There are two robots and one feeder. Each robot picks up a unique (physically different) part. The robots will take turns picking from the feeder. The pick order matters for this application. The alternating pick order is accomplished with “PF_ActivePart”. Robot motion is performed inside the PF_Robot callback. This example does not have parallel processing of the feeder and robot motion. The code is simple but not efficient. Robot 1 is picking and placing Part #1. Robot 2 is picking and placing Part #2. Each robot has a point labeled “park” and a point labeled “place”.

Sample Code

Main.prg

```
Function Main
   Robot 1
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park
   Robot 2
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park

   PF_Start 1, 2
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   If PF_QueLen(PartID) > 0 Then
      Select PartID
         Case 1
            Robot 1
            P0 = PF_QueGet(1)
            PF_QueRemove (1)
            Jump P0 /R
            On rbt1Gripper
            Wait .25
            Jump place
            Off rbt1Gripper
            Wait .25
            PF_ActivePart 2
         Case 2
            Robot 2
            P0 = PF_QueGet(2)
            PF_QueRemove (2)
            Jump P0 /L
            On rbt2Gripper
            Wait .25
            Jump place
            Off rbt2Gripper
            Wait .25
            PF_ActivePart 1
      Send

   EndIf

   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

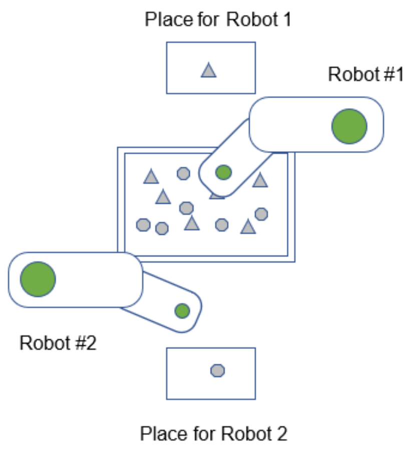
9.4.2 Program Example 4.2

Example Type:

2 Robots, 1 Feeder, Multiple Parts – Motion in separate tasks – Picking in a specific order

Configuration

Number of Robots: 2
Number of Feeders: 1
Number of Parts Types on Feeder: 2
Number of Placement Positions: 2
Camera Orientation: Fixed Downward Camera



Description

There are two robots and one feeder. Each robot picks up a unique (physically different) part. The robots will take turns picking from the feeder. This is accomplished by alternating "PF_ActivePart". If one of the robots does not have parts to pick then the other robot is allowed to continue picking from the feeder until no more parts are available for the robot. Robot 1 is picking and placing Part #1. Robot 2 is picking and placing Part #2. Each robot has a point labeled "park" and a point labeled "place". "PF_AccessFeeder" & "PF_ReleaseFeeder" are used to prevent both robots from attempting to access the feeder at the same time. When either robot has moved 30% of the way to its place position, the other robot is allowed to access the feeder.

Sample Code

Main.prg

```
Function Main
   Robot 1
   Motor On
   Power High
   Speed 50
   Accel 50, 50
   Jump park
   Robot 2
   Motor On
   Power Low
   Speed 50
   Accel 50, 50
   Jump park
   MemOff PartsToPick1
   MemOff PartsToPick2

   PF_Start 1, 2
   Xqt Robot1PickPlace
   Xqt Robot2PickPlace
Fend

Function Robot1PickPlace
   Robot 1
   Do
      Wait MemSw(PartsToPick1) = On
      PF_AccessFeeder 1
      P0 = PF_QueGet(1)
      PF_QueRemove (1)
      Jump P0 /R
      On 5
      Wait .5
      Jump place ! D30; MemOff PartsToPick1; PF_ReleaseFeeder 1 !
      Off 5
      Wait .25
   Loop
Fend

Function Robot2PickPlace
   Robot 2
   Do
      Wait MemSw(PartsToPick2) = On
      PF_AccessFeeder 1
      P0 = PF_QueGet(2)
      PF_QueRemove (2)
      Jump P0 /L
      On 2
      Wait .5
      Jump place ! D30; MemOff PartsToPick2; PF_ReleaseFeeder 1 !
      Off 2
      Wait .25
   Loop
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Select PartID
      Case 1
         If PF_QueLen(1) > 0 Then
            MemOn PartsToPick1
            Wait MemSw(PartsToPick1) = Off
            PF_ActivePart 2
         Else
            PF_ActivePart 1
         EndIf
      Case 2
         If PF_QueLen(2) > 0 Then
            MemOn PartsToPick2
            Wait MemSw(PartsToPick2) = Off
            PF_ActivePart 1
         Else
            PF_ActivePart 2
         EndIf
   Send
   PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

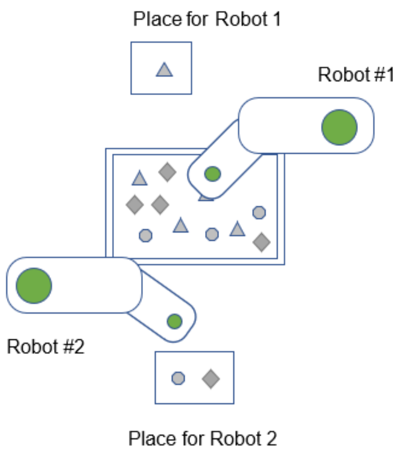
9.4.3 Program Example 4.3

Example Type:

2 Robots, 1 Feeder, Multiple Parts – Motion in separate tasks – Picking in a specific order

Configuration

Number of Robots: 2
Number of Feeders: 1
Number of Parts Types on Feeder: 3
Number of Placement Positions: 2
Camera Orientation: Fixed Downward Camera



Description

There are two robots and one feeder. Each robot will pick a different parts. Robot 1 will pick and place one of Part 1. Robot 2 will then pick and place one Part 4 and one Part 5. The pick order matters for this application. The alternating pick order is accomplished with “PF_ActivePart”. Robot motion is performed in parallel with the feeder vibration.

Sample Code

Main.prg

```
Function Main
    Robot 1
    Motor On
    Power High
    Speed 50
    Accel 50, 50
    Jump park
    Robot 2
    Motor On
    Power High
    Speed 50
    Accel 50, 50
    Jump park
    MemOff PartsToPick1
    MemOff PartsToPick4
    MemOff PartsToPick5

    PF_Start 1,4,5
    Xqt Robot1PickPlace
```

```
   Xqt Robot2PickPlace
Fend

Function Robot1PickPlace
   Robot 1
   Do
      Wait MemSw(PartsToPick1) = On
      PF_AccessFeeder(1)
      P0 = PF_QueGet(1)
      PF_QueRemove (1)
      Jump P0 /R
      On rbt1Gripper; Wait .25
      Jump place ! D30; MemOff PartsToPick1; PF_ReleaseFeeder 1!
      Off rbt1Gripper
      Wait .25
   Loop
Fend

Function Robot2PickPlace
   Robot 2
   Do
      Wait MemSw(PartsToPick4) = On
      PF_AccessFeeder(1)
      P0 = PF_QueGet(4)
      PF_QueRemove (4)
      Jump P0 /L
      On rbt2Gripper; Wait .25
      Jump place ! D30; MemOff PartsToPick4!
      Off rbt2Gripper; Wait .25
      Wait MemSw(PartsToPick5) = On
      P0 = PF_QueGet(5)
      PF_QueRemove (5)
      Jump P0 /L
      On rbt2Gripper; Wait .25
      Jump place ! D30; MemOff PartsToPick5; PF_ReleaseFeeder 1!
      Off rbt2Gripper; Wait .25
   Loop
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Select PartID
      Case 1
         MemOn PartsToPick1
         Wait MemSw(PartsToPick1) = Off
         PF_ActivePart 4
      Case 4
         MemOn PartsToPick4
         Wait MemSw(PartsToPick4) = Off
         PF_ActivePart 5
      Case 5
         MemOn PartsToPick5
         Wait MemSw(PartsToPick5) = Off
         PF_ActivePart 1
   Send
   PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

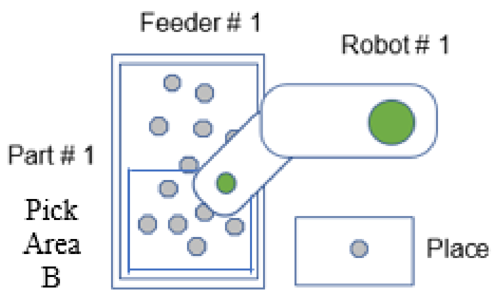
9.5 User Processes Vibration for Part via PF_Feeder Callback

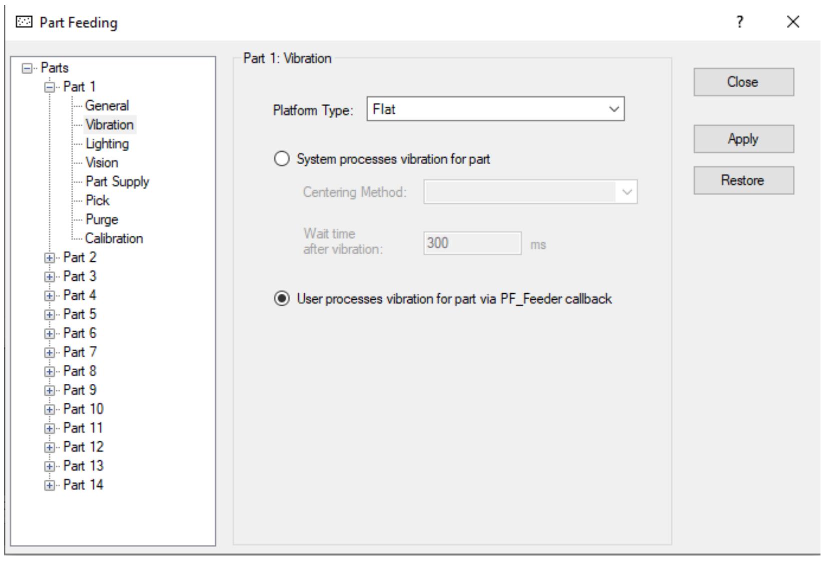
9.5.1 Program Example 5.1

Example Type:
Flat Platform – User Processes vibration for Part via PF_Feeder Callback

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 1
Platform Type: Flat
Pick Area: Pick Region B
Camera Orientation: Fixed Downward Camera





Part 1: Vibration

Platform Type: Flat

System processes vibration for part

Centering Method:

Wait time after vibration: 300 ms

[✓ User processes vibration for part via PF_Feeder callback]

Description

For this example, a standard Flat Platform is being used. Normally <System processes vibration for part> in Menu-[Tool]-[PartFeeing]-[Part]-[Vibration] would be selected for a Flat plate. For this example, we will demonstrate how you can select <User processes vibration for part via PF_Feeder callback> to handle the vibration action yourself.
The user's vibration code will be performed inside the PF_Feeder callback. <User processes vibration for part via PF_Feeder callback> is required when you want a different vibration strategy than what the system can provided. When using a custom platform (i.e, holes, slots or pockets), you must handle the vibration yourself via the PF_Feeder callback (refer to *Example 5.2* for details).

Even if the <User processes vibration for part via PF_Feeder callback> is selected (for Standard Flat, Anit-stick and Anti-roll platforms), the system will make the determination of how to best vibrate the part in different situations. The part judgement is provided to the PF_Feeder callback using a parameter called "state". The different states are defined with constants in the "PartFeeding.inc" file. For example, the constant "PF_FEEDER_PICKOK" means that parts are available to be picked up by the robot. As another example, the constant "PF_FEEDER_FLIP" is passed to the PF_Feeder callback when the system has determined that the best action is to Flip the parts. It is entirely up to the user whether to use the "state" recommendation or not.

Conceptually, the user can recreate the System Processing via the PF_Feeder state and the appropriate vibration statements. Once again, normally the <System processes vibration for part> would be selected for a Flat tray. That said, this example demonstrates how to mimic the System Processing using the PF_Feeder callback and vibration commands.

Sample Code

Main.prg

```
Function main
   If Not Motor = On Then
      Motor On
   EndIf
   Power Low
   Jump park
   PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

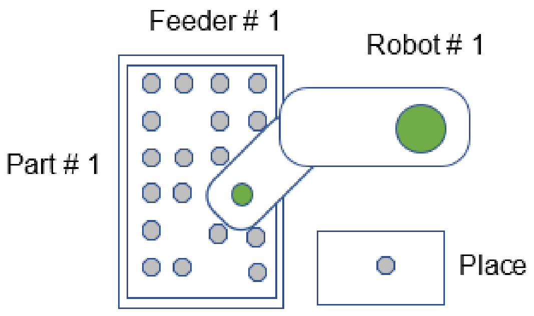
```
Function PF_Feeder(PartID As Integer, NumFrontParts As Integer,
NumBackParts As Integer, state As Integer) As Integer

Select state

‘ OK to Pick
      Case PF_FEEDER_PICKOK
         ' Call PF_Robot because there are parts ready to pick
         PF_Feeder = PF_CALLBACK_SUCCESS

‘ Supply more parts
      Case PF_FEEDER_SUPPLY
         PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)
         PF_Feeder = PF_CALLBACK_RESTART ‘ Restart and re-acquire images

      ‘ Parts are spread out but need to be flipped
      Case PF_FEEDER_FLIP
         PF_Flip PartID
‘ Restart and re-acquire images
         PF_Feeder = PF_CALLBACK_RESTART

      ‘ Shift parts into pick region
Case PF_FEEDER_SHIFT
PF_Shift PartID, PF_SHIFT_FORWARD
         PF_Feeder = PF_CALLBACK_RESTART ‘ Restart and re-acquire images

‘ Center the parts
      Case PF_FEEDER_CENTER
         PF_Center PartID, PF_CENTER_LONG_AXIS
         PF_Center PartID, PF_CENTER_SHORT_AXIS
         PF_Feeder = PF_CALLBACK_RESTART ‘ Restart and re-acquire images

‘ Center, Flip and Separate
      Case PF_FEEDER_CENTER_FLIP
         PF_Center PartID, PF_CENTER_LONG_AXIS
         PF_Center PartID, PF_CENTER_SHORT_AXIS
         PF_Flip PartID
         PF_Feeder = PF_CALLBACK_RESTART ‘ Restart and re-acquire images

‘ Hopper is empty
Case PF_FEEDER_HOPPER_EMPTY
         PFStatusReturnVal = PF_Status(PartID, PF_STATUS_NOPART)
         PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)
   ‘ Center, Flip and Separate
         PF_Center PartID, PF_CENTER_LONG_AXIS
         PF_Center PartID, PF_CENTER_SHORT_AXIS
         PF_Flip PartID
         PF_Feeder = PF_CALLBACK_RESTART ‘ Restart and re-acquire images

‘ Parts have gathered against the platform wall
Case PF_FEEDER_SHIFT_BACKWARDS
   PF_Shift PartID, PF_SHIFT_BACKWARD
   PF_Feeder = PF_CALLBACK_RESTART

‘ Hopper Supply, Center, Flip and Separate
Case PF_FEEDER_SUPPLY_CENTER_FLIP
   PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY)
         PF_Center PartID, PF_CENTER_LONG_AXIS
         PF_Center PartID, PF_CENTER_SHORT_AXIS
   PF_Flip PartID
   PF_Feeder = PF_CALLBACK_RESTART ' Restart and re-acquire images

' Too many parts
Case PF_FEEDER_TOO_MANY
   PFStatusReturnVal = PF_Status(PartID, PF_STATUS_TOOMANYPART)
   PF_Feeder = PF_CALLBACK_RESTART  ' Restart and re-acquire images

' Wrong part
Case PF_FEEDER_WRONGPART
   PFStatusReturnVal = PF_Status(PartID, PF_STATUS_WRONGPART)
   PF_Feeder = PF_CALLBACK_RESTART  ' Restart and re-acquire images

   Send

Fend
```

9.5.2 Program Example 5.2

Example Type:
Custom Platform with Holes – User Processes vibration for Part via PF_Feeder Callback

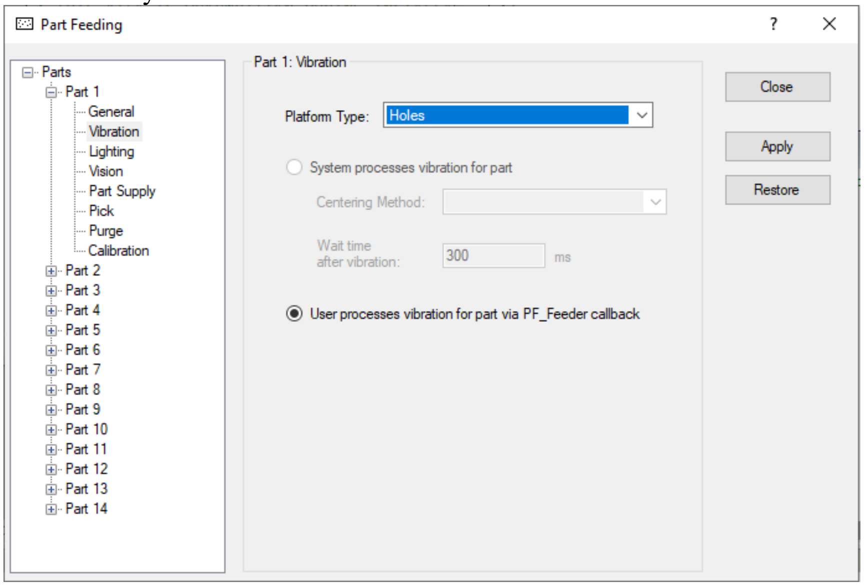
Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 1
Platform Type: Holes
Pick Area: Anywhere
Camera Orientation: Fixed Downward Camera



Description

Because this is a custom platform, the <User processes vibration for part via PF_Feeder callback> is automatically selected.



Part 1: Vibration

Platform Type: Holes

System processes vibration for part

Centering Method:

Wait time after vibration: 300 ms

[✓ User processes vibration for part via PF_Feeder callback]

After vision acquires an image and the part queue is loaded, the PF_Feeder callback is called. The user's code must judge how to vibrate the feeder inside the PF_Feeder callback. The quantity of "front" and "back" parts are provided to the PF_Feeder callback. The parameters are called "NumFrontParts" and "NumBackParts". For this example, if the NumFrontParts is greater than 0 then no vibration is required since parts are available to be picked up by the robot. In that case, the callback return value is "PF_CALLBACK_SUCCESS". This return value tells the system to go ahead and call the PF_Robot callback.

If the NumFrontParts equals 0 then the sample code VRun's the Part Blob sequence to determine if there is a clump of parts or whether there are no parts at all. If the Part Blob sequence does not find any parts, then the hopper is turned on. If the Part Blob sequence finds something then the feeder Flips, Shifts Forward and then Shifts backward so that parts can fall into the holes.
After parts have been vibrated on the feeder, the system must re-acquire vision images (the location of the parts has changed due to vibration). This is accomplished by setting the return value to "PF_CALLBACK_RESTART". "PF_CALLBACK_RESTART" will restart the Part Feeding process from the beginning, re-aquire new images, reload the part queue and then call PF_Feeder once again to determine if any further action is required.

TIP

☞ A Flip, a long duration Shift Forward and a short duration Shift Backward is the typical feeding strategy for Custom Platforms.

NOTE

☞ The constant PF_FEEDER_UNKNOWN is passed to PF_Feeder when the Platform Type is Holes, Slots or Pockets. In the case of Custom Plates, the system has no knowledge of how the plate is machined and consequently, the system can not properly determine how to best feed the parts.

Sample Code

Main.prg

```
Function main
   If Not Motor = On Then
      Motor On
   EndIf
   Power Low
   Jump park
   PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

```
Function PF_Feeder(PartID As Integer, NumFrontParts As Integer,
NumBackParts As Integer, state As Integer) As Integer

   ' Example for Structured Platform with holes state = PF_FEEDER_UNKNOWN

    Integer PFControlReturnVal
    Integer numFound

   Select True

    ‘ OK to Pick
    Case NumFrontParts > 0
           ' Call PF_Robot because there are parts ready to pick
        PF_Feeder = PF_CALLBACK_SUCCESS   ‘

    ‘ No Front parts were found but there are Back parts
    Case NumFrontParts = 0 And NumBackParts <> 0

        ‘ Flip, long Shift Forward and short Shift Backward
        PF_Flip PartID, 500
        PF_Shift PartID, PF_SHIFT_FORWARD, 1000
        PF_Shift PartID, PF_SHIFT_BACKWARD, 300

        PF_Feeder = PF_CALLBACK_RESTART ‘ Restart and re-acquire images

    ‘ There are no Front or Back parts found
    ‘ Either there is a clump of parts or there are no parts on the tray
    ‘ Acquire an image from the Part Blob sequence to make a determination
    Case NumFrontParts = 0 And NumBackParts = 0

    PF_Backlight 1, On   ‘ Backlight on
        VRun PartBlob    ‘ Acquire Image
    PF_Backlight 1, Off ‘Backlight off
        VGet PartBlob.Blob01.NumberFound, numFound ‘ Were any Blobs found?

        If numFound > 0 Then   ' Clump of parts found

            ‘ Flip, long Shift Forward and short Shift Backward
            PF_Flip PartID, 500
            PF_Shift PartID, PF_SHIFT_FORWARD, 1000
            PF_Shift PartID, PF_SHIFT_BACKWARD, 300

        Else   ‘ No parts found

            ‘ Call the Control callback to supply more parts
            PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)
            EndIf

        PF_Feeder = PF_CALLBACK_RESTART ‘ Restart and re-acquire images

   Send

Fend
```

9.6 Error Handling

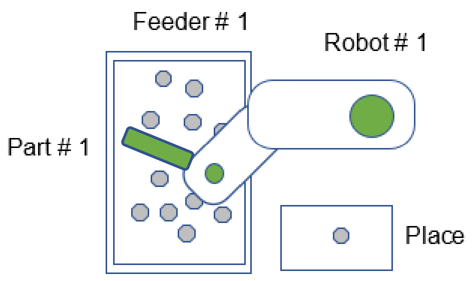
This section describes how to process errors in different use case scenarios.

9.6.1 Program Example 6.1

Example Type:
Handling a potential error condition inside the Callback Function

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

This example detects and handles a potential error condition inside a callback function so that the error does not occur inside the main Part Feeding process loop. For this example, the PF_Vision callback is used to acquire an image and load the part coordinate queue with the vision results. The robot has a large tool offset. In some instances, the Tool cannot align with the part angle (determined by vision) because the robot would have to travel outside its work envelope. If left unhandled, this condition would result in a "coordinate conversion" error. This sample code checks whether the robot can pick up the part with the Tool at the vision angle prior to loading the coordinates into the part queue. This is achieved with the TargetOK statement.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

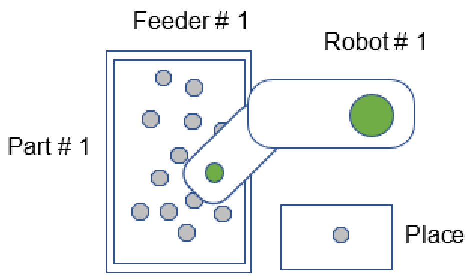
```
Function PF_Robot(PartID As Integer) As Integer

   ' Tool 1 will be used to pick up the part
   Tool 1

   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend

Function PF_Vision(PartID As Integer, ByRef numBack As Integer) As Integer
   Boolean found
   Integer i, numFront
   Real RB_X, RB_Y, RB_U, RB_Z

   ' Tool 1 will be used to pick up the part
   Tool 1

   ' Pick Z coordinate
   RB_Z = -132.0

   ' Initialize coordinates queue
   PF_QueRemove PartID, All
    PF_Backlight 1, On
   ' Detect the parts
   VRun UsrVisionSeq
   PF_Backlight 1, Off

   VGet UsrVisionSeq.Geom01.NumberFound, numFront 'Front Parts
   VGet UsrVisionSeq.Geom02.NumberFound, numBack  'Back Parts
   If numFront <> 0 Then
      For i = 1 To numFront
         VGet UsrVisionSeq.Geom01.RobotXYU(i), found, RB_X, RB_Y, RB_U
         If found Then
            If TargetOK(XY(RB_X, RB_Y, RB_Z, RB_U)) Then
               PF_QueAdd PartID, XY(RB_X, RB_Y, RB_Z, RB_U)
            EndIf
         EndIf
      Next
   EndIf

   PF_Vision = PF_CALLBACK_SUCCESS

Fend
```

9.6.2 Program Example 6.2

Example Type:
Handling a process error inside the Callback Function

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

For this example, the PF_Vision callback is used to acquire an image and load the part coordinate queue with the vision results. In this example, a minimum number of pickable parts must be on the feeder tray to load the part queue. If the part queue is not loaded after 3 attempts, a message is displayed asking the operator whether to continue trying to find parts or Stop.

Sample Code

Main.prg

```
Function main
   If Not Motor = On Then
      Motor On
   EndIf
   Power Low
   Jump park
   PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer

   ' Tool 1 will be used to pick up the part
   Tool 1

   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

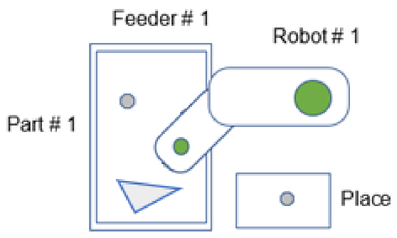
```
Function PF_Vision(PartID As Integer, ByRef numBack As Integer) As Integer
   Boolean found
   Integer i, numFront
   Real RB_X, RB_Y, RB_U, RB_Z
   Integer RetryCount
   String msg$
   Integer mFlags, answer

   ' Pick Z coordinate
   RB_Z = -132.0

   ' Initialize coordinates queue
   PF_QueRemove PartID, All
   RetryCount=0

   Do
      PF_Backlight 1, On
      ' Detect the parts
      VRun UsrVisionSeq
      PF_Backlight 1, Off

      VGet UsrVisionSeq.Geom01.NumberFound, numFront 'Front Parts
      VGet UsrVisionSeq.Geom02.NumberFound, numBack  'Back Parts
      If numFront >= 5 Then 'Min number of parts = 5 for this example
         For i = 1 To numFront
         VGet UsrVisionSeq.Geom01.RobotXYU(i), found, RB_X, RB_Y, RB_U
            If found Then
               PF_QueAdd PartID, XY(RB_X, RB_Y, RB_Z, RB_U)
            EndIf
         Next
         Exit Do
      Else
         If RetryCount < 3 Then
            PF_Center 1, PF_CENTER_LONG_AXIS
            PF_Center 1, PF_CENTER_SHORT_AXIS
            PF_Flip 1,500
            RetryCount = RetryCount + 1
         Else
            msg$ = PF_Name$(PartID) + CRLF + CRLF
            msg$ = msg$ + "Min Number of Parts Cannot be Loaded." + CRLF
            msg$ = msg$ + "Do you want to Continue trying?"
            mFlags = MB_YESNO + MB_ICONQUESTION
            MsgBox msg$, mFlags, "Minimum Number Parts", answer
            If answer = IDNO Then
               PF_Stop(PartID)
               Exit Do
            Else
               RetryCount=0
            EndIf
         EndIf
      EndIf
   Loop

   PF_Vision = PF_CALLBACK_SUCCESS

Fend
```

9.6.3 Program Example 6.3

Example Type:
Handling a Status Error in the PF_Status Callback

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

For this example, the last part in the tray is detected as a wrong part. The tray has the Purge Gate option installed and enabled. The detected “wrong part” will be Purged from the tray, new parts will be fed from the hopper and the parts will be centered & flipped.

Sample Code

PartFeeding.prg

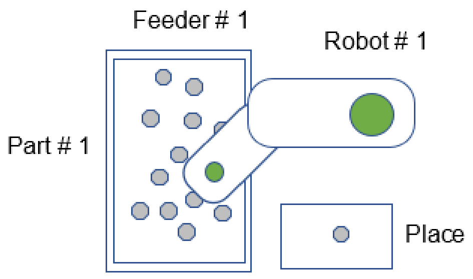
```
Function PF_Status(PartID As Integer, Status As Integer) As Integer

    Select Status

' Other Status Cases have been removed from this sample code for simplicity

        Case PF_STATUS_WRONGPART
            ' There may be a wrong part on the feeder platform.
            ' Purge Part 1 without vision feedback. Purge duration is default.
            ' The Purge Gate automatically opens and closes
            PF_Purge 1, PF_PURGETYPE_NOVISION
            ' Turn on the hopper for 3 sec
            PF_OutputOnOff 1, On, 1, 3000
            Wait 3.0
            PF_Center 1, PF_CENTER_LONG_AXIS
            PF_Center 1, PF_CENTER_SHORT_AXIS
            PF_Flip 1,500

' Other Status Cases have been removed from this sample code for simplicity

    Send

    PF_Status = PF_CONTINUE
Fend
```

9.6.4 Program Example 6.4

Example Type:
Handling a User Error inside the PF_Status Callback

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

The robot has a vacuum cup gripper with a vacuum switch to detect that the part has been properly picked up. If the vacuum sensor fails to detect the part, the robot will attempt to re-pick the part. After 3 unsuccessful attempts, a User Error (8000) is generated. The User Error is sent to the PF_Status callback by setting the PF_Robot return value to the User Error number. The PF_Status call back displays a message box allowing the operator to Continue or Exit the application.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer

   Integer PickRetryCount ' Pick Retry Count

   Do While PF_QueLen(PartID) > 0

      ' Get position of part to be picked
      P10 = PF_QueGet(PartID)

      PickRetryCount = 0
      Do
         Jump P10
         On Vacuum
         Wait Sw(VacOn), 0.5     ' 0.5 second timeout on Vacuum switch
         If TW = False Then  ' Vacuum successful
            Exit Do          ' Exit Do Loop and place the part
         EndIf
         Off Vacuum
         PickRetryCount = PickRetryCount + 1 ' Increment retry count
         If PickRetryCount = 3 Then
            ' Vacuum retries were not successful
            Jump park
            PF_QueRemove PartID
            PF_Robot = 8000  ' Set the return value to user error 8000
            ' PF_Status callback will be called with status value 8000
            Exit Function
         EndIf
      Loop

      ' Part detected in vacuum gripper
      Jump place
      Off Vacuum
      Wait 0.25

      ' Deque
      PF_QueRemove PartID

      'Check Cycle stop
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf

   Loop

   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

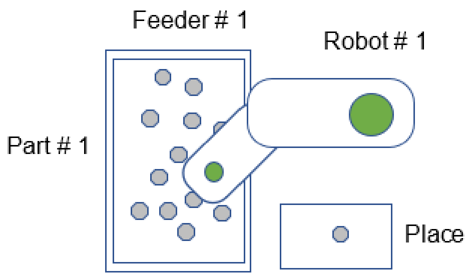
```
Function PF_Status(PartID As Integer, Status As Integer) As Integer

    String msg$
    Integer mFlags, answer

    Select Status

' Other Status Cases have been removed from this sample code for simplicity

        Case 8000 ' User Error 8000 occured.

            msg$ = PF_Name$(PartID) + CRLF + CRLF
            msg$ = msg$ + "Vacuum Pick error has occurred." + CRLF
            msg$ = msg$ + "Do you want to Continue?"
            mFlags = MB_YESNO + MB_ICONQUESTION
            MsgBox msg$, mFlags, "Vacuum Pick Error", answer
            If answer = IDNO Then
                PF_Status = PF_EXIT
            Else
                PF_Status = PF_CONTINUE
            EndIf

            Exit Function
    Send

Fend
```

9.6.5 Program Example 6.5

Example Type:
Handling a Controller Error inside a Part Feeding Callback

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1



Description

Normally when a controller error occurs, the Part Feeding Process automatically sets the constant PF_STATUS_ERROR to the Status parameter and the PF_Status function will be called. This happens without any additional user code. PF_Status typically prints or displays the error number and message. The Part Feeding process will terminate once the PF_Status function ends.

In this example, however, we want to handle a specific controller error inside the PF_Robot callback. All other controller errors will be sent to the PF_Status callback with the Status parameter set to PF_STATUS_ERROR.

In this case, the gripper's electrical and pneumatic lines prevent the U axis (SCARA robot) from rotating the full +/-360 degrees. The Joint 4 motion range has been limited inside the EPSONRC+ Robot Manager. Limiting the Joint 4 motion range prevents the electrical and pneumatic lines from being damaged. If a part on the feeder would require Joint 4 to rotate beyond its motion range, an Error 4001 "Arm reached the limit of motion range" will occur. The error handler removes the part from the queue and the robot will resume picking up all the remaining parts. To prevent the vision system from re-acquiring an image of the same rejected parts and re-adding them to the queue, a PF_Flip is executed after all parts have been picked and the queue is empty.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer

   Integer errNum

   OnErr GoTo ehandle  ' Error Handler

retry:
   Do While PF_QueLen(PartID) > 0

      ' Get position of part to be picked
      P10 = PF_QueGet(PartID)

      ' Error 4001 can occur if the part's angle
      ' causes Joint 4 to rotate beyond its motion range
      Jump P10

      On gripper
      Wait 0.25
      Jump place
      Off gripper
      Wait 0.25

      'Deque
      PF_QueRemove PartID

      'Check Cycle stop
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf

   Loop

   PF_Flip PartID

   PF_Robot = PF_CALLBACK_SUCCESS
   Exit Function

ehandle:

   errNum = Err
   If errNum = 4001 Then   ' Example of Handled error
      Print "Error 4001: Arm reached the limit of motion range"
      PF_QueRemove PartID ' Remove the part from the queue
      EResume retry ' Continue picking the remaining parts in the queue
   Else
      ' Other unhandled errors
      ' PF_Status is called with the PF_STATUS_ERROR status parameter
      PF_Robot = PF_STATUS_ERROR
   EndIf
Fend
```

```
Function PF_Status(PartID As Integer, Status As Integer) As Integer

    Select Status

' Other Status Cases have been removed from this sample code for simplicity

        Case PF_STATUS_ERROR ' Error.
            msg$ = PF_Name$(PartID) + CRLF
            msg$ = msg$ + "Error!! (code: " + Str$(Err) + " )  " + ErrMsg$(Err)
            MsgBox msg$, MB_ICONSTOP

' Other Status Cases have been removed from this sample code for simplicity

    Send

    If Status = PF_STATUS_ERROR Then
        ' A controller error occurred. Terminate the Part Feeding Process.
        PF_Status = PF_EXIT
    Else
        ' Otherwise Continue running the Part Feeding Process.
        PF_Status = PF_CONTINUE
    EndIf

Fend
```

9.7 Multiple Cameras

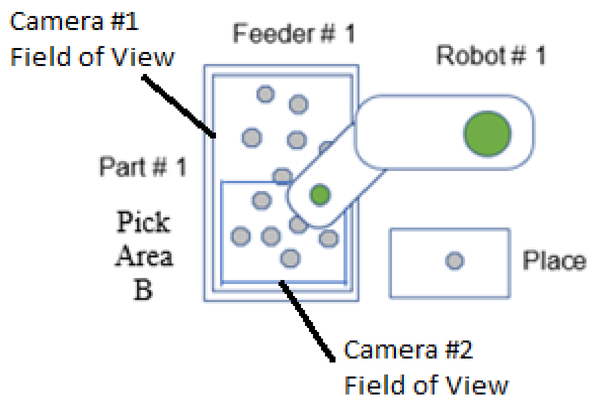
This section describes how improve pick accuracy by using multiple cameras.

9.7.1 Program Example 7.1

Example Type:
Using Multiple Fixed Downward Cameras for Improved Pick Region Accuracy

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 1
Platform Type: Flat
Pick Area: Pick Region B
Camera Orientations:
Camera #1: Fixed Downward. Field of View is the size of the entire feeder tray. Used by the Part Blob Sequence.
Camera #2 - Fixed Downward. Field of View is the same size as Pick Region B (1/2 of the feeder tray). Used by the Part Sequence.



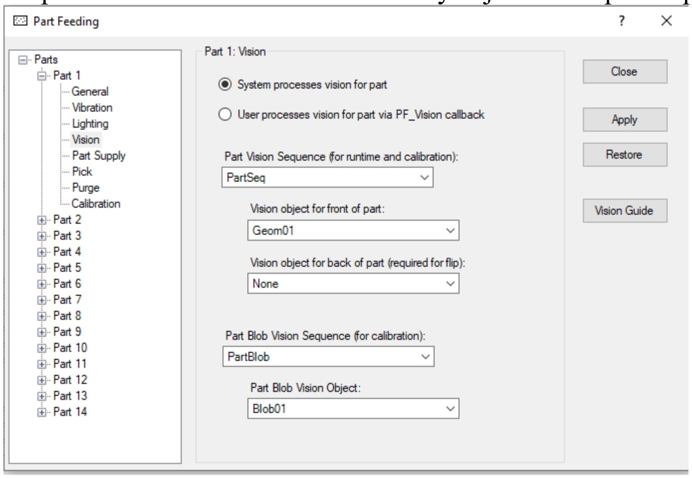
Description

Camera #1 has a field of view that covers the entire feeder tray. The Part Blob sequence uses Camera #1. The Part Blob sequence determines how to vibrate the feeder based upon the number of parts and the distribution of parts on the entire tray. Camera #2's field of view is the same area as Pick Region B. To fill as much of the field of view as possible, Camera #2 should be rotated 90 degrees relative to Camera #1 (i.e., the cameras are orthogonal to each other). Camera #2 is used for the Part Sequence. The parts that are found by the Part Sequence are used to load the part feeding queue. Because the field of view is half the size of Camera #1's field of view, the mm/pixel resolution can be significantly improved. Additionally, since Camera #1 is only used for making a judgement on how to vibrate the feeder, it can have a lower resolution than Camera #2. For example, Camera #1 could have a 640 x 480 resolution and Camera #2 could have a 5472 x 3648 resolution. By reducing the field of view and increasing the camera resolution, the robot's pick accuracy will be improved.

Flip & Separate Auto Calibration uses the Part Sequence to verify that an acceptable number of pickable parts were found. If a small field of view camera is used for the Part Sequence, then the Flip & Separate Auto Calibration will not work properly without additional setup steps.

You can do one of the following.

1. Skip the Auto Calibration and manually adjust the Flip & Separate calibration parameters.



Part Feeding

Part 1: Vision

[✓ System processes vision for part]

User processes vision for part via PF_Vision callback

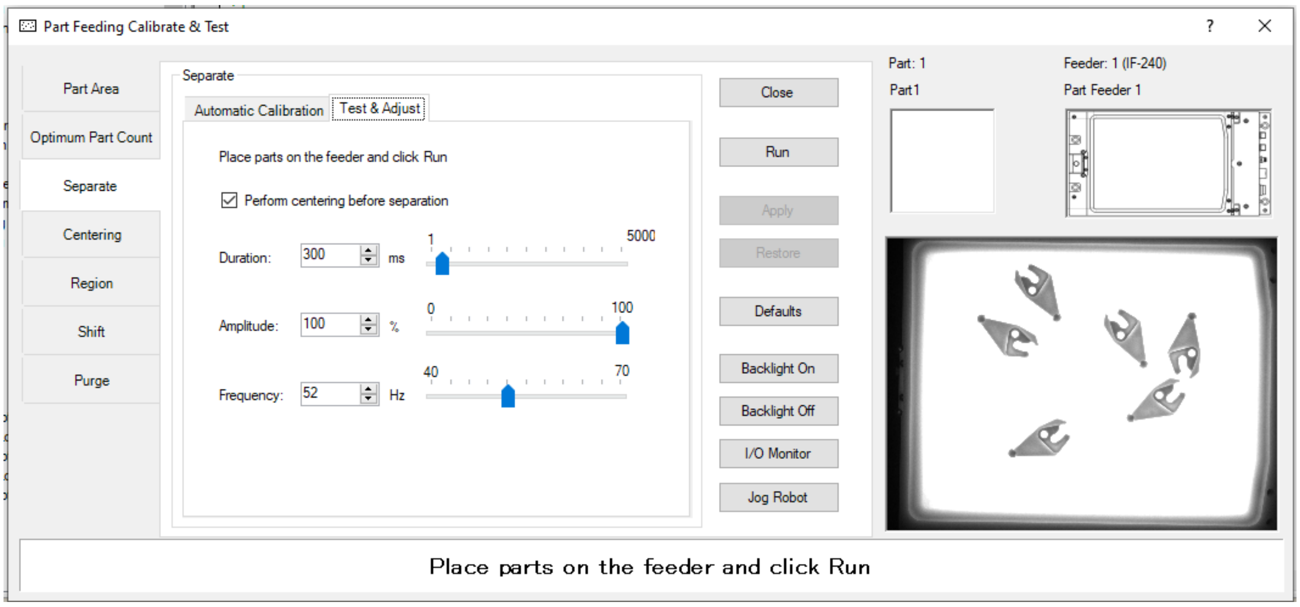
Part Vision Sequence (for runtime and calibration): PartSeq

Vision object for front of part: Geom01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): PartBlob

Part Blob Vision Object: Blob01



Part Feeding Calibrate & Test

Separate

Automatic Calibration | Test & Adjust

Place parts on the feeder and click Run

[✓] Perform centering before separation

Duration: 300 ms (1–5000)

Amplitude: 100 % (0–100)

Frequency: 52 Hz (40–70)

Part: 1

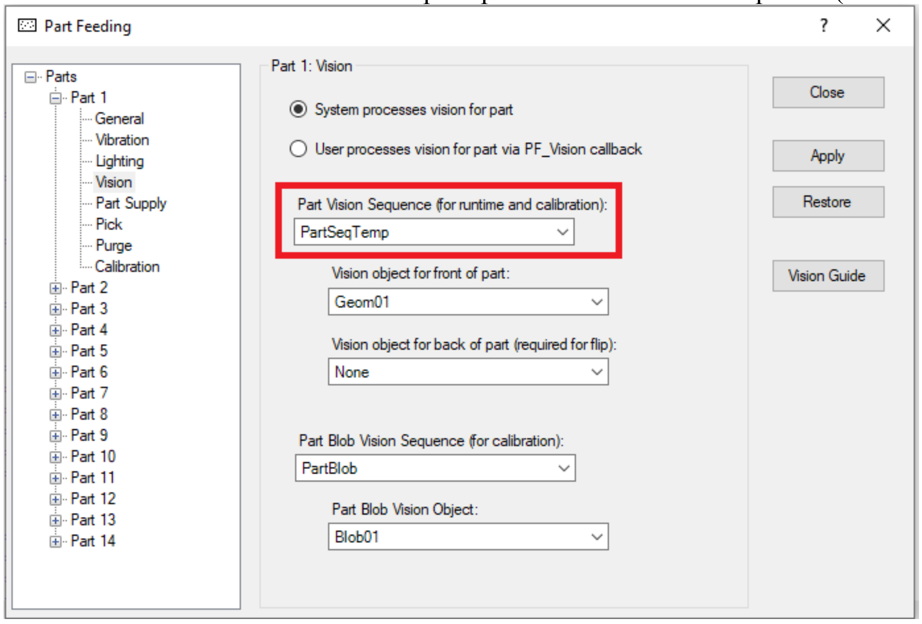
Part1

Feeder: 1 (IF-240)

Part Feeder 1

Place parts on the feeder and click Run

2. Make a Part Sequence that uses Camera #1 (large field of view) just for the Auto Calibration. For this example, the Part Sequence that uses Camera #2 (small field of view) is called "PartSeq". The temporary Part Sequence that uses Camera #1 (large field of view) is called "PartSeqTemp". "PartSeqTemp" uses a Geometric object to find the part. "PartSeqTemp" will only be used temporary to calibrate the feeder.Select "PartSeqTemp" as the Part Vision Sequence (shown below).


Part Feeding

Parts
Part 1
General
Vibration
Lighting
Vision
Part Supply
Pick
Purge
Calibration
Part 2
Part 3
Part 4
Part 5
Part 6
Part 7
Part 8
Part 9
Part 10
Part 11
Part 12
Part 13
Part 14

Part 1: Vision

(●) System processes vision for part
( ) User processes vision for part via PF_Vision callback

Part Vision Sequence (for runtime and calibration): PartSeqTemp

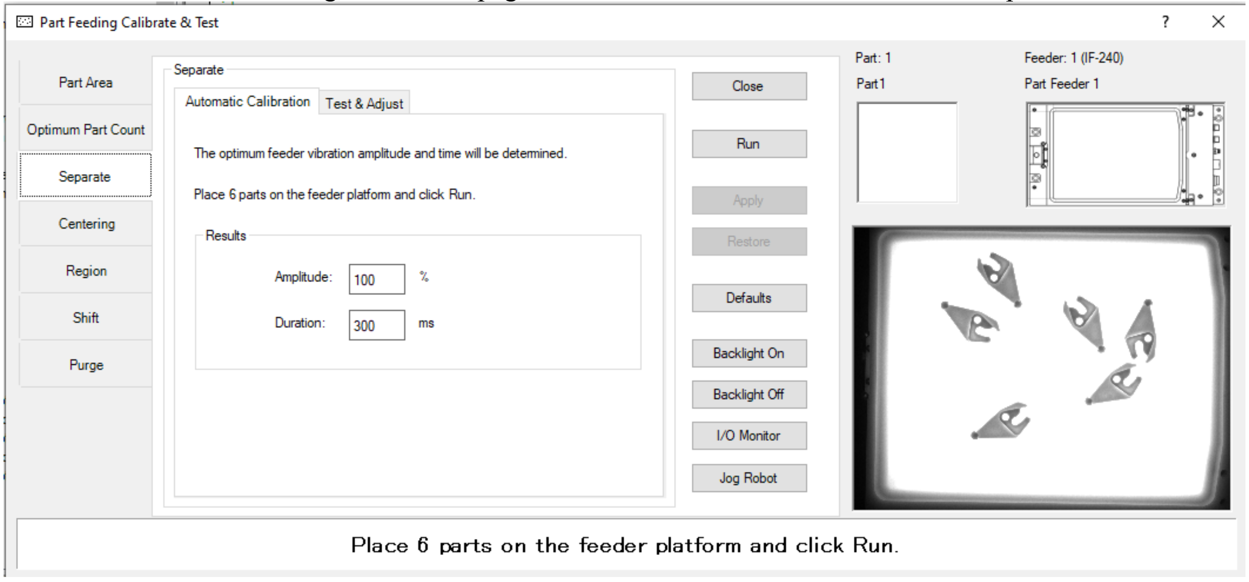
Vision object for front of part: Geom01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): PartBlob

Part Blob Vision Object: Blob01

3. Go to the Part Feeding Calibration page and Run the Automatic Calibration for Separation.


Part Feeding Calibrate & Test

Separate

Automatic Calibration | Test & Adjust

The optimum feeder vibration amplitude and time will be determined.

Place 6 parts on the feeder platform and click Run.

Results

Amplitude: 100 %

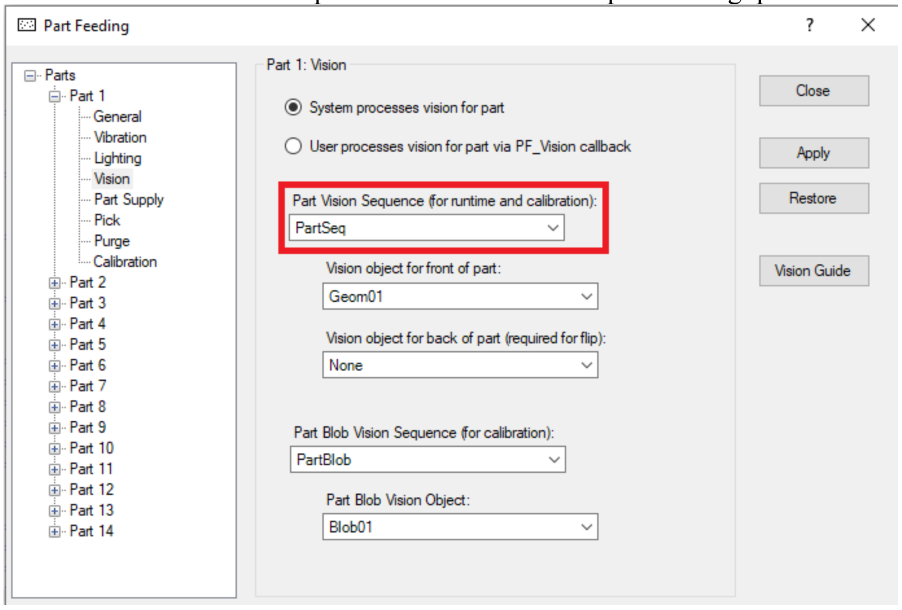
Duration: 300 ms

Part: 1
Part 1

Feeder: 1 (IF-240)
Part Feeder 1

Place 6 parts on the feeder platform and click Run.

4. After all the desired calibrations have been completed, close the Calibration & Test dialog. Change the Part Vision Sequence to “PartSeq” (which uses the small field of view Camera #2). At runtime, the vision results from “PartSeq” will be used to load the part feeding queue.



Part 1: Vision

(●) System processes vision for part
( ) User processes vision for part via PF_Vision callback

Part Vision Sequence (for runtime and calibration): PartSeq

Vision object for front of part: Geom01

Vision object for back of part (required for flip): None

Part Blob Vision Sequence (for calibration): PartBlob

Part Blob Vision Object: Blob01

No special part feeding code is required.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
    Do While PF_QueLen(PartID) > 0
        P0 = PF_QueGet(PartID)
        Jump P0
        On Gripper; Wait 0.2;
        Jump Place
        Off Gripper; Wait 0.2;
        PF_QueRemove PartID
        If PF_IsStopRequested(PartID) = True Then
            Exit Do
        EndIf
    Loop
    PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

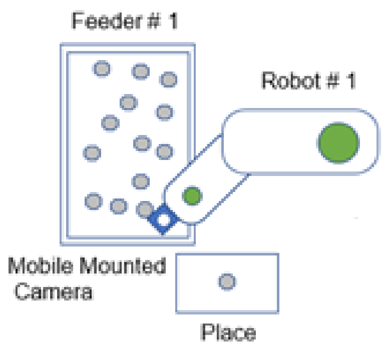
9.7.2 Program Example 7.2

Example Type:

Using both a Fixed Downward Camera and a Mobile Mounted Camera to Improve Pick Accuracy

Configuration

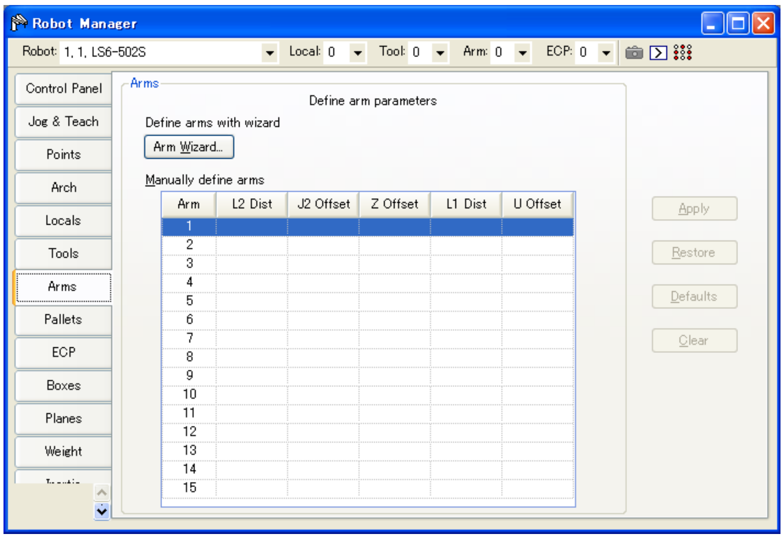
Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 1
Platform Type: Flat
Pick Area: Anywhere
Camera Orientations:
Camera #1: Fixed Downward. Field of View is the size of the entire feeder tray. Camera #1 is used by the Part Blob Sequence and the Part Sequence.
Camera #2 – Mobile Mounted onto Joint 2. Field of View is slightly larger than the part itself. Camera #2 acquires a secondary image of the part prior to pick up.



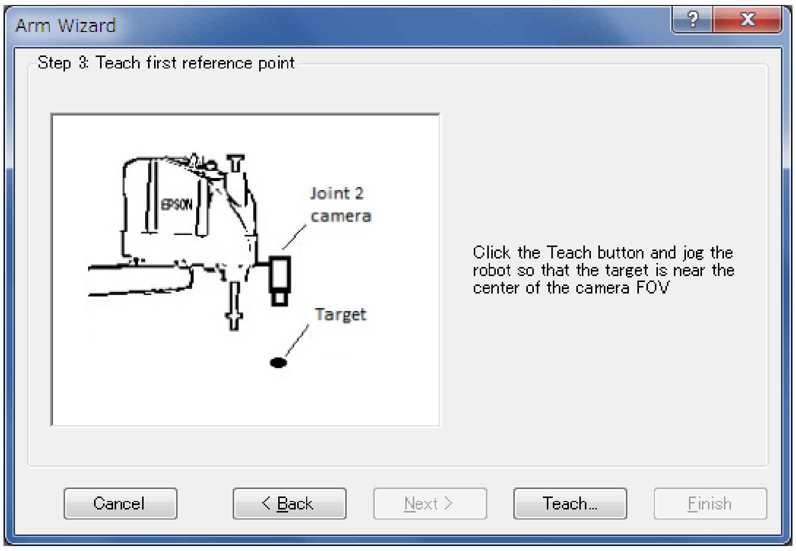
Description

An “Arm” will be defined for the Mobile Joint #2 camera (SCARA robot only). If you are using a Six Axis robot then a Tool (rather than an Arm) can be defined for a Mobile Joint #6 camera. Instead of commanding the gripper to go directly to the part queue location, Camera #2 will be driven over the top of the part. An additional vision sequence will be run to determine a more precise pick position for the part. The Mobile camera has a much smaller field of view than the Fixed Downward camera and as a result, the pick accuracy will be improved. Because of the additional motion required to position the camera over the part and the additional vision acquisition, the overall cycle time will be longer.

To automatically define the Arm, go to the EPSONRC+|Tools|Robot Manager. Select the [Arms] tab in the Robot Manager. For this example, select Arm 1 and click the Arm Wizard button. Go through each step of the Arm Wizard. Additional information about the Arm Wizard can be found in the section in the Vision Guide 7.0 Software Manual called “Arm Setting of Camera Installation Position”.



Arm	L2 Dist	J2 Offset	Z Offset	L1 Dist	U Offset
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					



Click the Teach button and jog the robot so that the target is near the center of the camera FOV

Calibrate the mobile mounted camera and create a vision sequence that can locate a single part on the feeder. This sequence will be VRun from within the PF_Robot callback. For this example, the sequence is called “MobileCam”. “MobileCam” will not be selected in the Part Feeding dialog. The Part Blob Sequence and Part Sequence use Camera #1 and are selected in the Part Feeding dialog as normal.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

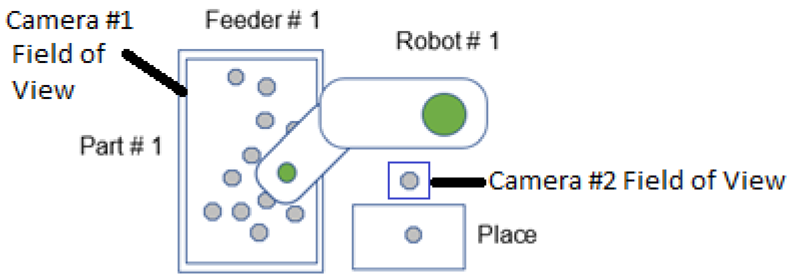
```
Function PF_Robot(PartID As Integer) As Integer

   Boolean found
   Real x, y, u

   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Arm 1 'Select the Arm that is defined for the Mobile Camera
      Jump P0:Z(0)  'Position the Mobile camera over the part
      Vrun MobileCam
      VGet MobileCam.Geom01.RobotXYU, found, x, y, u
      Arm 0 'Select default robot arm
      If found then
         Jump XY(x, y, PICKZ, u) /R
         On Gripper; Wait 0.2;
         Jump Place
         Off Gripper; Wait 0.2;
      EndIf
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

9.7.3 Program Example 7.3

Example Type:
Using Multiple Fixed Cameras for Improved Pick Accuracy

Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on Feeder: 1
Number of Placement Positions: 1
Platform Type: Flat
Pick Area: Anywhere
Camera Orientations:
Camera #1: Fixed Downward. Field of View is the size of the entire feeder tray. Used by the Part Blob Sequence and the Part Sequence
Camera #2 - Fixed Upward. Field of View is slightly larger than the part. This camera is used to create a tool offset for the part held in the robot's gripper.



Description

Camera #1 is Fixed Downward over the feeder tray. Camera #2 is Fixed Upward. Camera #1 has a field of view that covers the entire feeder tray. The Part Blob Sequence amd Part Sequence uses Camera #1. Camera #2's field of view should slightly larger than the part itself. After the robot picks up the part from the feeder, the robot will hold the part over the Camera #2. The Camera #2 is used to dynamically create a Tool for the part being held in the gripper. The robot will then place the part in the newly defined Tool. The Tool offsets compensate for inaccuracy in the pick-up from the feeder. Camera #2 is only used in the PF_Robot callback code. It is not selected on the EPSON RC+ 7.0-Menu-[Tools]-[Part Feeding]-[Vision]. The sample code uses the RobotToolXYU result from the Camera #2 to determine the tool offsets. The PF_Robot function first runs a sequence to locate the part in the gripper. Then the tool offsets are retrieved using VGet RobotToolXYU, and the tool is defined using TLSet. The robot then places the part using the new Tool.

This example requires the Fixed Upward camera to be calibrated. For details on how to calibrate a Fixed Upward camera, refer to the section of the Vision Guide 7.0 Software Manual call *"7.6.3 Calibration Procedure: Fixed Upward Camera"*.

Sample Code

Main.prg

```
Function main
   If Not Motor = On Then
      Motor On
   EndIf
   Power Low
   Jump park
   PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Boolean found
   Real xTool, yTool, uTool

   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Tool 0 ‘ Select the correct Tool number for the Gripper
      Jump P0
      On Gripper; Wait 0.2;
      Jump upCam
      VRun findPartInGripper
      VGet findPartInGripper.Geom01.RobotToolXYU, found, xTool, yTool,
   uTool
      If found Then
         TlSet 1, XY(xTool, yTool, 0, 0)
         Tool 1
         Jump place
      Else
         Jump reject   ‘ Part not found in gripper – reject part
      EndIf

      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

9.8 Improving Vision Results

This section describes how improve vision results.

9.8.1 Program Example 8.1

Example Type:
Using Image Buffers and ImageOp SubtractAbs

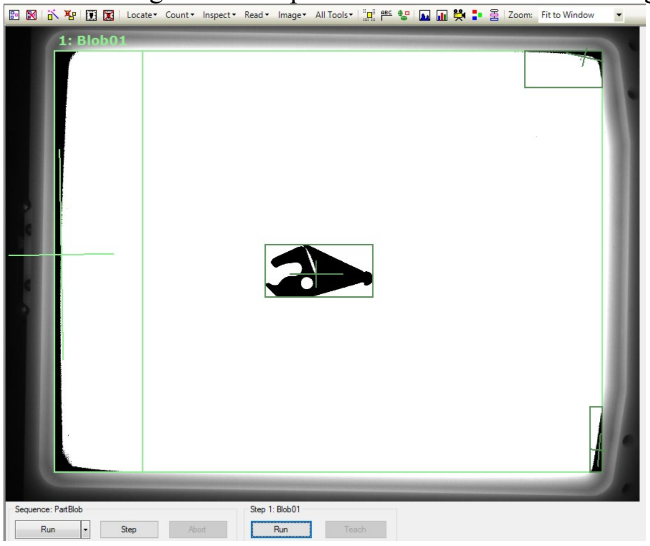
Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1

Description

Even with the backlight turned on, the corners of the tray can have shadows. If the Part Blob search window includes the corners and if the Blob's thresholds are not properly adjusted, then the shadows can be mistakenly identified as parts. The Part Blob Sequence is used to detect individual parts or an accumulation of parts. If the Part Blob sees the shadows as parts, then the system will make a bad decision on how to vibrate or the PF_Control callback may not turn on the hopper since the system thinks that there are sufficient parts in the tray.

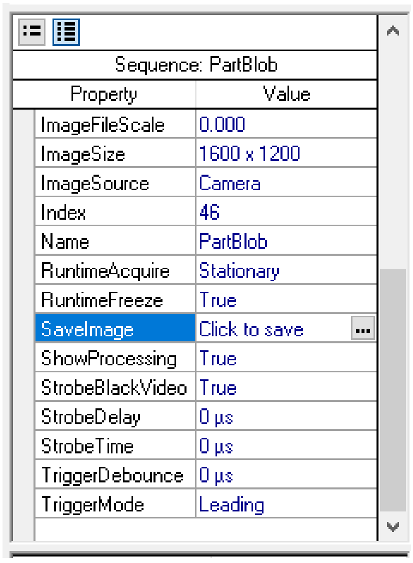
The following is an example of how the corner shadowing can be misidentified as parts.



An ImageOp vision object can be added to the Part Blob sequence to help fix this issue. The ImageOp will use the SubtractAbs operation. SubtractAbs outputs the difference between two image buffers. For this example, the ImageBuffer1 property will be an image file of the empty feeder and the ImageBuffer2 property will be the image acquired by the camera (value "0" is the camera's image buffer). When the image buffers are subtracted, the feeder will effectively be removed from the image.

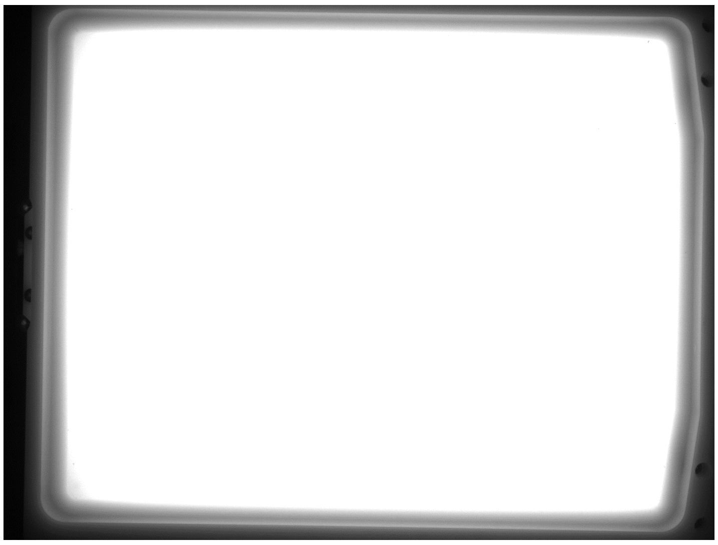
Here are the steps.

1. Create a new vision sequence and call it "Part Blob".
2. Turn on the feeder's backlight and remove any parts in the feeder tray.
3. Click on the "Click to Save" button on the Part Blob's SaveImage property. For this example, we will name the file "Empty IF-240". The image of the file is shown below.

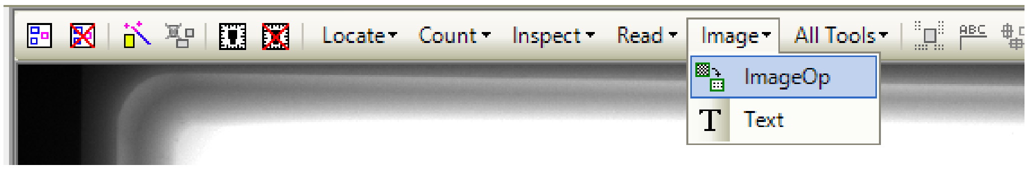


Sequence: PartBlob

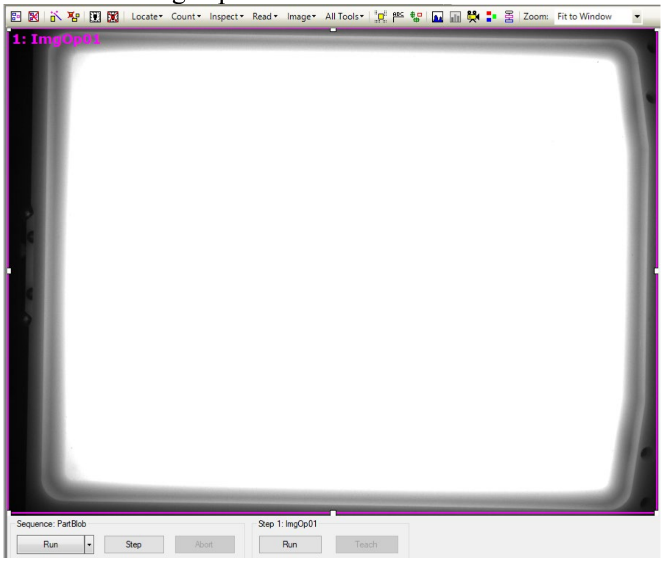
Property	Value
ImageFileScale	0.000
ImageSize	1600 x 1200
ImageSource	Camera
Index	46
Name	PartBlob
RuntimeAcquire	Stationary
RuntimeFreeze	True
SaveImage	Click to save
ShowProcessing	True
StrobeBlackVideo	True
StrobeDelay	0 µs
StrobeTime	0 µs
TriggerDebounce	0 µs
TriggerMode	Leading



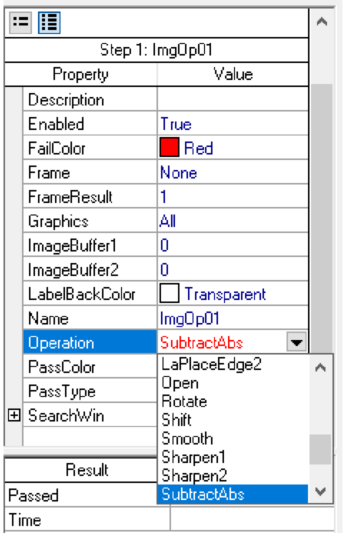
4. Drag and drop an ImageOp vision object from the Vision Guide toolbar.



5. Resize the ImageOp to be the entire camera field of view.



6. Change the ImageOp's Operation property to "SubtractAbs".



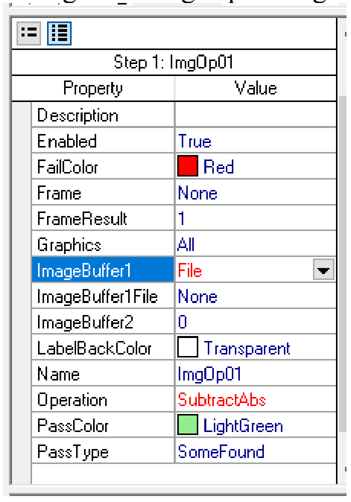
Step 1: ImgOp01

Property	Value
Description	
Enabled	True
FailColor	🟥 Red
Frame	None
FrameResult	1
Graphics	All
ImageBuffer1	0
ImageBuffer2	0
LabelBackColor	☐ Transparent
Name	ImgOp01
Operation	SubtractAbs
PassColor	
PassType	
⊞ SearchWin	

LaPlaceEdge2
Open
Rotate
Shift
Smooth
Sharpen1
Sharpen2
SubtractAbs

Result	
Passed	
Time	

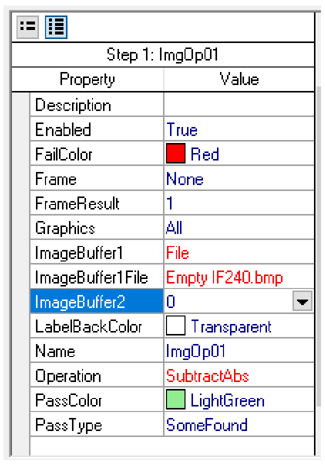
7. Change the ImageOp's ImageBuffer1 property to "File.



Step 1: ImgOp01

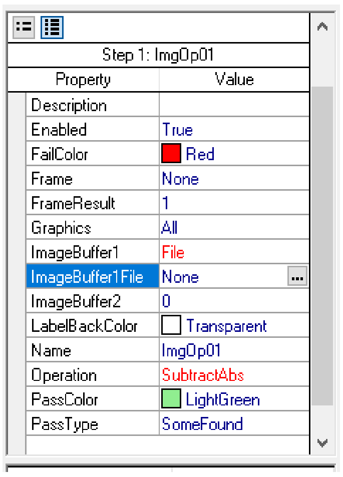
Property	Value
Description	
Enabled	True
FailColor	🟥 Red
Frame	None
FrameResult	1
Graphics	All
ImageBuffer1	File
ImageBuffer1File	None
ImageBuffer2	0
LabelBackColor	☐ Transparent
Name	ImgOp01
Operation	SubtractAbs
PassColor	🟩 LightGreen
PassType	SomeFound

8. Click the ImageBufferFile button and navigate to the file which was previously saved as "Empty IF-240".



Step 1: ImgOp01

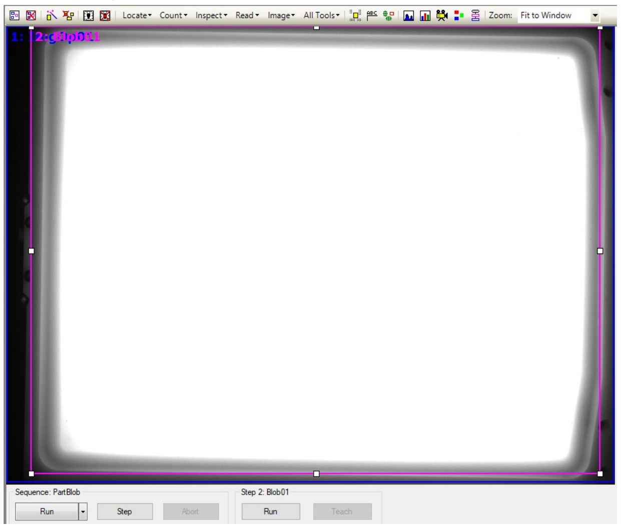
Property	Value
Description	
Enabled	True
FailColor	🟥 Red
Frame	None
FrameResult	1
Graphics	All
ImageBuffer1	File
ImageBuffer1File	Empty IF240.bmp
ImageBuffer2	0
LabelBackColor	☐ Transparent
Name	ImgOp01
Operation	SubtractAbs
PassColor	🟩 LightGreen
PassType	SomeFound

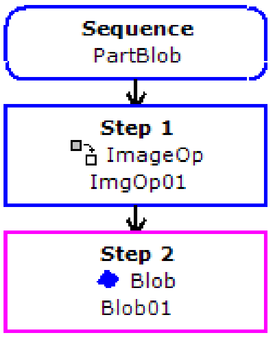


Step 1: ImgOp01

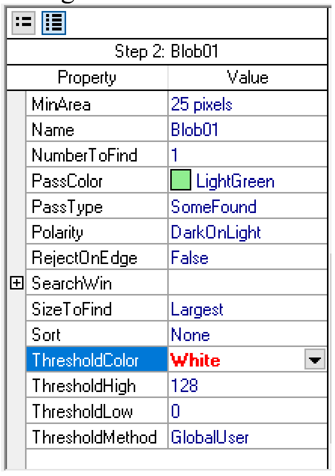
Property	Value
Description	
Enabled	True
FailColor	🟥 Red
Frame	None
FrameResult	1
Graphics	All
ImageBuffer1	File
ImageBuffer1File	None ...
ImageBuffer2	0
LabelBackColor	☐ Transparent
Name	ImgOp01
Operation	SubtractAbs
PassColor	🟩 LightGreen
PassType	SomeFound

9. Add a Blob from the Vision Guide toolbar. Resize the Blob's Search Window to be larger than the feeder tray.





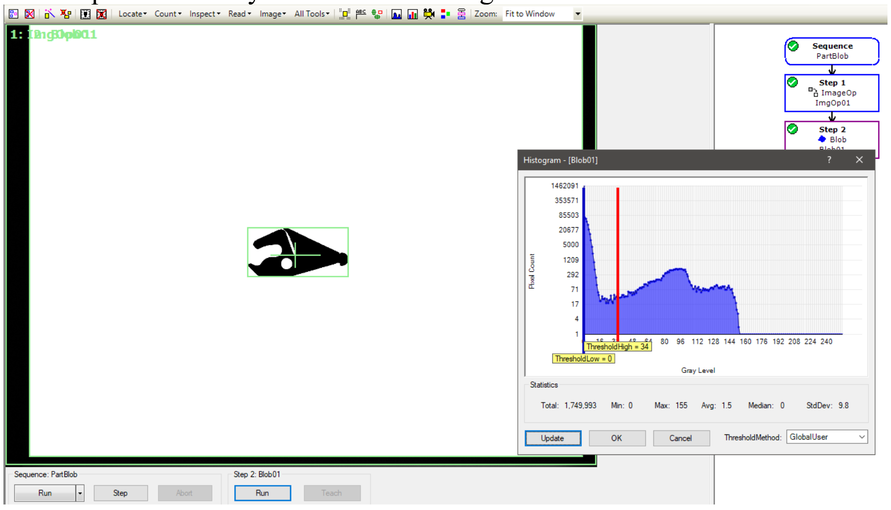
10. Change the Blob's ThresholdColor property to "White".



Step 2: Blob01

Property	Value
MinArea	25 pixels
Name	Blob01
NumberToFind	1
PassColor	🟩 LightGreen
PassType	SomeFound
Polarity	DarkOnLight
RejectOnEdge	False
⊞ SearchWin	
SizeToFind	Largest
Sort	None
ThresholdColor	White
ThresholdHigh	128
ThresholdLow	0
ThresholdMethod	GlobalUser

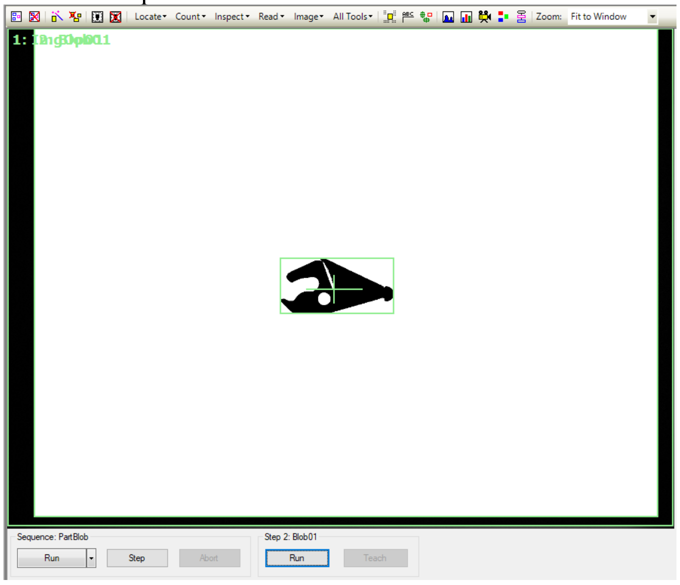
11. Place a part on the tray and click the Histogram button on the Vision Guide toolbar.



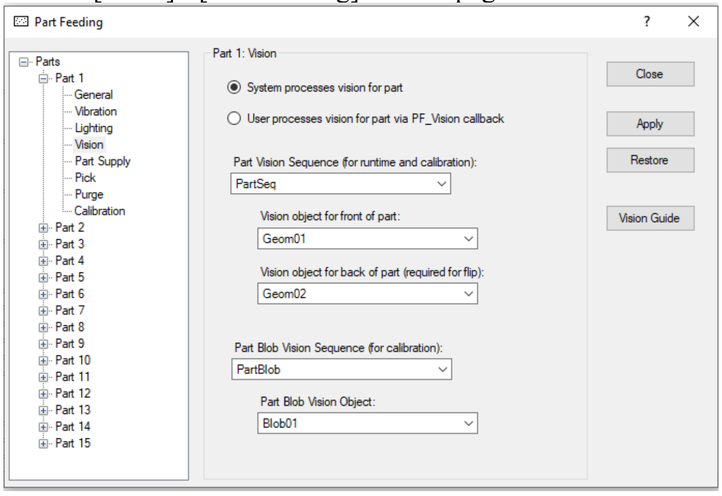
Statistics
Total: 1,749,993 Min: 0 Max: 155 Avg: 1.5 Median: 0 StdDev: 9.8
ThresholdMethod: GlobalUser

Drag the red ThresholdHigh bar on the histogram until the part is properly binarized. Click the Update as necessary and click the OK button on the histogram window when finished.

Now when the Part Blob sequence is run, the image of the feeder will be completely removed. The part will appear black on a completely white background. The feeder has effectively been masked out of the Part Blob sequence.



12. Select "PartBlob" as the Part Blob Vision Sequence for the desired part in the EPSON RC+ 7.0 - Menu - [Tools] - [Part Feeding] Vision page.



Part Feeding

- Parts
  - Part 1
    - General
    - Vibration
    - Lighting
    - Vision
    - Part Supply
    - Pick
    - Purge
    - Calibration
  - Part 2
  - Part 3
  - Part 4
  - Part 5
  - Part 6
  - Part 7
  - Part 8
  - Part 9
  - Part 10
  - Part 11
  - Part 12
  - Part 13
  - Part 14
  - Part 15

Part 1: Vision

[✓ System processes vision for part] User processes vision for part via PF_Vision callback

Part Vision Sequence (for runtime and calibration): PartSeq

Vision object for front of part: Geom01

Vision object for back of part (required for flip): Geom02

Part Blob Vision Sequence (for calibration): PartBlob

Part Blob Vision Object: Blob01

Close
Apply
Restore
Vision Guide

No special part feeding code is required.

Sample Code

Main.prg

```
Function main
   If Not Motor = On Then
      Motor On
   EndIf
   Power Low
   Jump park
   PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
   Do While PF_QueLen(PartID) > 0
      P0 = PF_QueGet(PartID)
      Jump P0
      On Gripper; Wait 0.2;
      Jump Place
      Off Gripper; Wait 0.2;
      PF_QueRemove PartID
      If PF_IsStopRequested(PartID) = True Then
         Exit Do
      EndIf
   Loop
   PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

9.8.2 Program Example 8.2

Example Type:
Using Part Blob SearchWinType RotatedRectangle

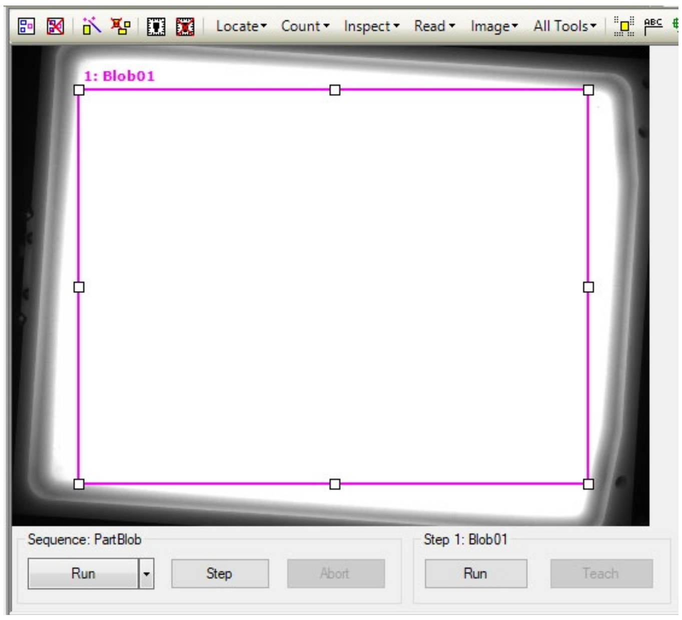
Configuration

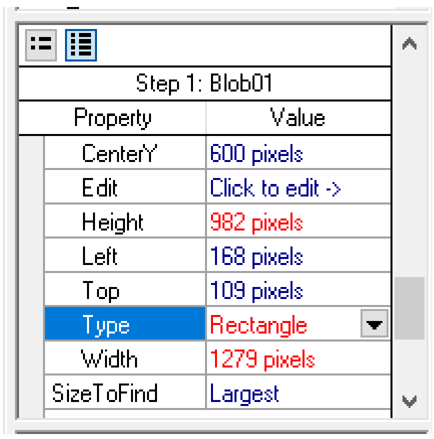
Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1

Description

When the camera's field of view and feeder tray are not parallel to each other, it can be difficult to properly size the Part Blob's search window. If the search window is too large, then the Part Blob may detect the tray as parts. If the search window is too small, then the system can not make proper judgements on the quantity and dispersion of the parts on the tray. Starting with EPSONRC+ 7.5.2, the Part Blob object can use a "RotatedSearchWin" for the SearchWinType property.

Here is an example of when the SearchWinType is set to "Rectangle".

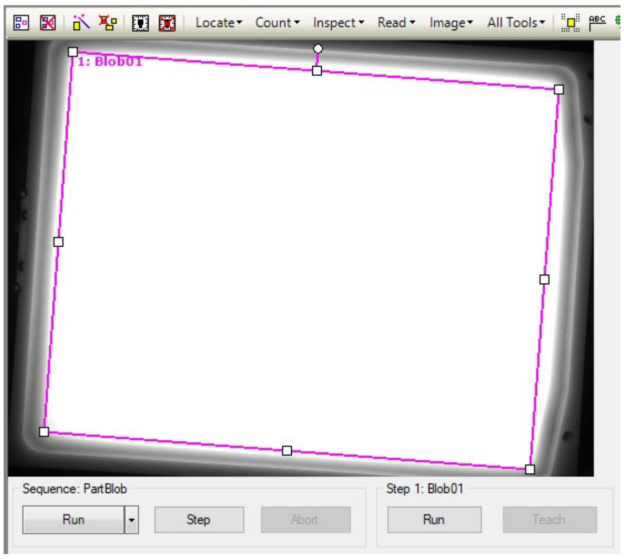


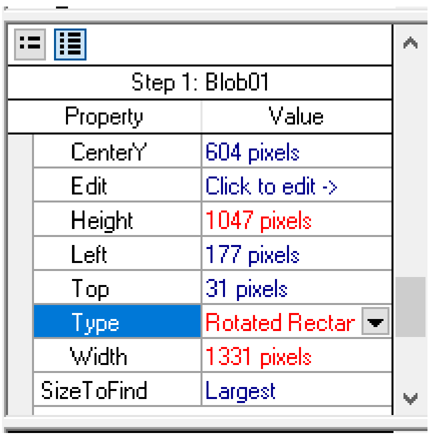


Step 1: Blob01

Property	Value
CenterY	600 pixels
Edit	Click to edit ->
Height	982 pixels
Left	168 pixels
Top	109 pixels
Type	Rectangle
Width	1279 pixels
SizeToFind	Largest

When the Part Blob's SearchWinType is set to "RotatedRectangle", the camera's field of view and the feeder's tray can be aligned.





Step 1: Blob01

Property	Value
CenterY	604 pixels
Edit	Click to edit ->
Height	1047 pixels
Left	177 pixels
Top	31 pixels
Type	Rotated Rectar
Width	1331 pixels
SizeToFind	Largest

NOTE

The SearchWin Angle property is restricted to +/-45 degrees for the Part Blob.

No special part feeding code is required.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
    Do While PF_QueLen(PartID) > 0
        P0 = PF_QueGet(PartID)
        Jump P0
        On Gripper; Wait 0.2;
        Jump Place
        Off Gripper; Wait 0.2;
        PF_QueRemove PartID
        If PF_IsStopRequested(PartID) = True Then
            Exit Do
        EndIf
    Loop
    PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

9.8.3 Program Example 8.3

Example Type:
Using Don’t Care Pixels for the Part Blob Search Window

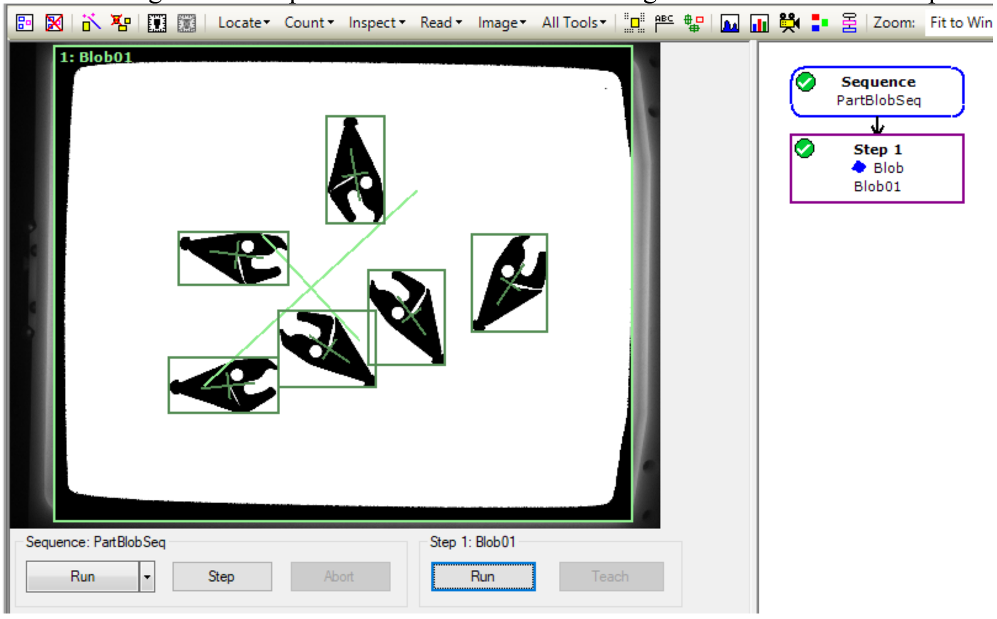
Configuration

Number of Robots: 1
Number of Feeders: 1
Number of Parts Types on the Feeder: 1
Number of Placement Positions: 1
Camera Orientation: Fixed Downward Camera over Feeder #1

Description

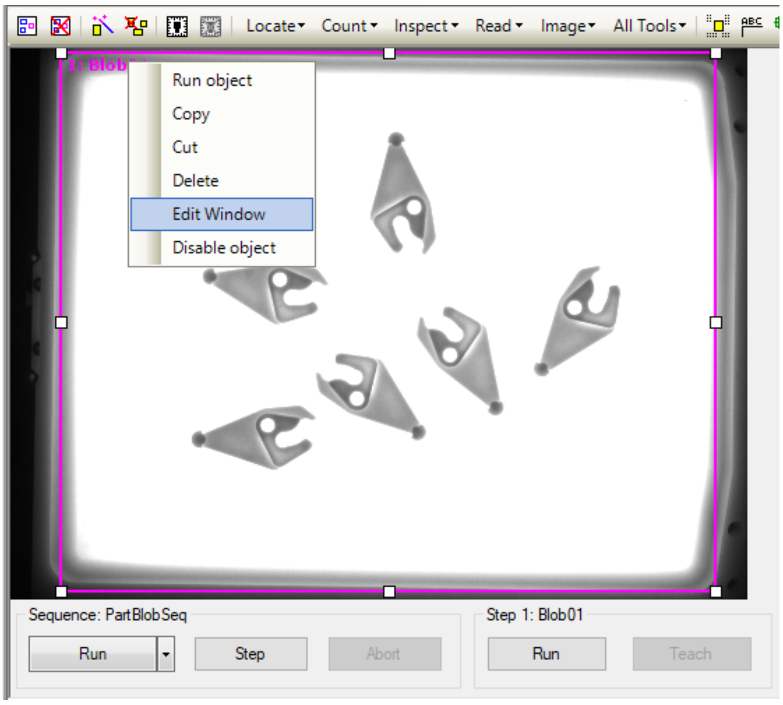
Even with the backlight turned on, the corners of the tray can have shadows. If the Part Blob search window includes the corners and if the Blob’s thresholds are not properly adjusted, then the shadows can be mistakenly identified as parts. The Part Blob Sequence is used to detect individual parts or an accumulation of parts. If the Part Blob sees the shadows as parts, then the system will make a bad decision on how to vibrate.

The following is an example of how the corner shadowing can be misidentified as parts.

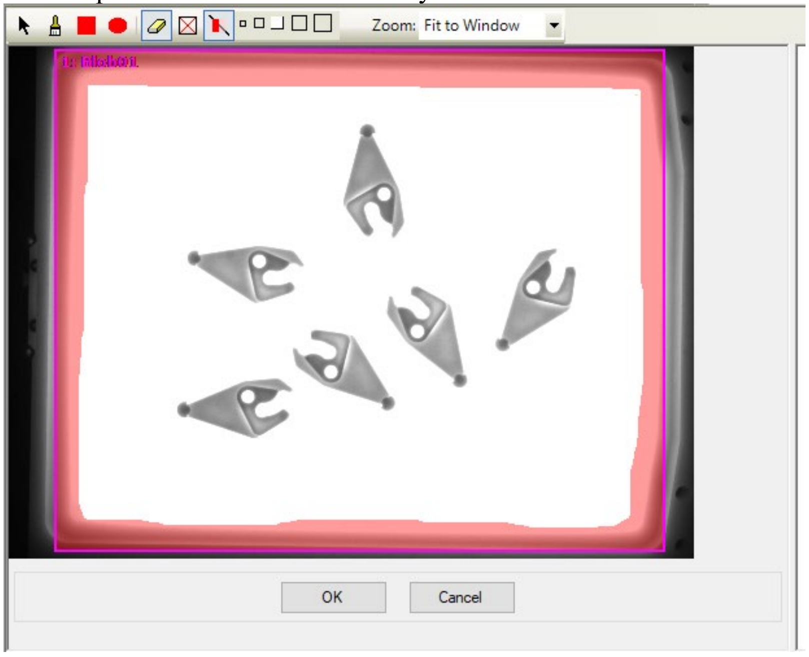


The Blob’s Search Window can be masked by painting “Don’t Care Pixels” in the regions where you do not want to search (i.e., the corners of the tray).

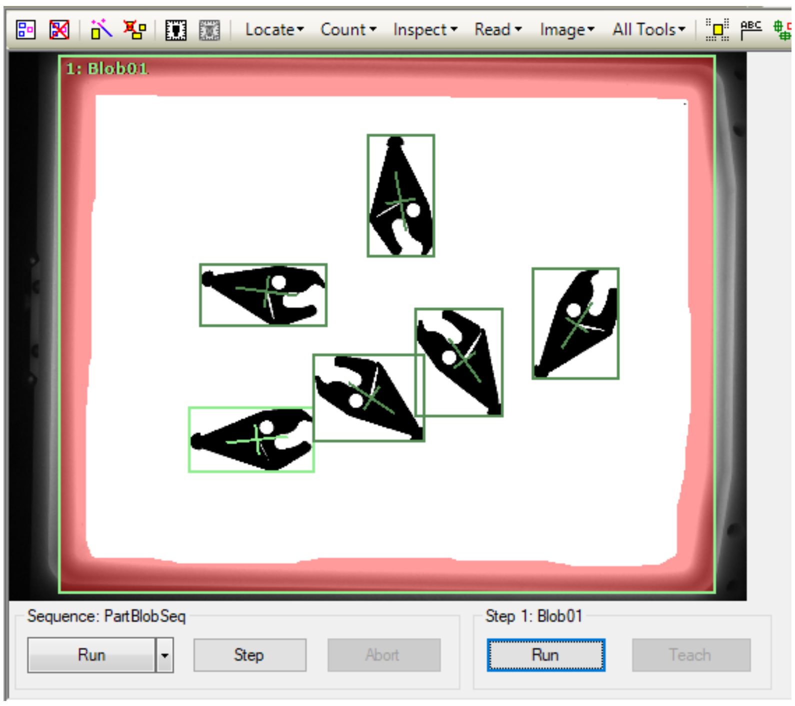
Right click on the Blob and select “Edit Window” from the fly out menu or select the SearchWin “EditWindow” property for the Blob.



Use the paint brush to remove the tray from the Search Window.



Now when you Run the Blob, the tray will not be included in the Search Window (as shown below).



No special part feeding code is required.

Sample Code

Main.prg

```
Function main
    If Not Motor = On Then
        Motor On
    EndIf
    Power Low
    Jump park
    PF_Start 1
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
    Do While PF_QueLen(PartID) > 0
        P0 = PF_QueGet(PartID)
        Jump P0
        On Gripper; Wait 0.2;
        Jump Place
        Off Gripper; Wait 0.2;
        PF_QueRemove PartID
        If PF_IsStopRequested(PartID) = True Then
            Exit Do
        EndIf
    Loop
    PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

Advanced

1. Multiple Parts & Multiple Robots

This chapter explains how to use multiple parts on the same feeder at the same time. It also outlines how multiple robots can safely access the same feeder when running a group of parts.

1.1 Specifications & Requirements for Multiple Parts & Multiple Robots

- A maximum of 4 Parts Feeders can be used with 1 robot controller.
- Part Feeders cannot be shared between robot controllers. For example, two T/VT series robot cannot share the same feeder. If you want multiple robots to share a feeder then you must use an Controller which supports multiple robots on a single robot controller.
- A system can have any combination of feeder models. For example, a system could have one IF-80, one IF-240, one IF-380 and one IF-530 or any other combination.
- A maximum of 4 parts can run on the same feeder at the same time. When multiple parts are running on the same feeder at the same time, the parts should have similar physical characteristics. In other words, the parts should weigh about the same amount, have similar dimensions, be made of similar materials, have approximately the same surface area etc…. Every part will use its own unique vibration parameters. Consequently, a part's vibration parameters should not cause other parts to fly out of the feeder.
- A maximum of 2 robots can use the same feeder at the same time. If you attempt to PF_Start a grouping of parts (i.e., PF_Start 1, 2, 5) and the parts are assigned to more than 2 robots, an error will occur.
- When PF_Start is executed, a single part or a grouping of parts will run on the feeder. If you attempt to execute PF_Start multiple times for parts that are assigned to the same feeder #, a message (issued from the PF_Status callback) will appear indicating that the "feeder is already in use" and the second PF_Start will not be executed.
- When executing PF_Start with a grouping of parts (i.e., PF_Start 1, 2, 5), all the parts must be assigned to the same feeder #.
- PF_Start runs a single part or a grouping of parts on a feeder. Each feeder that is running is assigned a specific controller task# (see the table below). The user's application code should not use a task that has been reserved for the feeder. If a Feeder # is not being used, then its task is available for the user's code. If you use the PF_InitLog statement for logging Part Feeding data, specific timer #'s is reserved for the system and should not be used in the user's code. If you use the feeder control commands (PF_Center, PF_ConterByShift, PF_Flip, PF_Shift), specific SyncLock #'s are reserved for the system and should not be used in the user's code. The Task #, Timer # and SyncLock # are specific to the Feeder # (as shown below).

Feeder Number	Task	Timer	SyncLock
1	32	63	63
2	31	62	62
3	30	61	61
4	29	60	60

- The first part that is listed in the PF_Start statement will be the first “Active Part” (i.e., the first desired part). Selection of the next desired part is done with the PF_ActivePart statement. PF_ActivePart will be discussed in more detail throughout this chapter.
- The feeding action (vibrations, pick area and supply method) is performed for the current Active Part. Every part that is listed in the PF_Start statement will use its own settings when it has been selected as the Active Part. For example, each part in the PF_Start grouping could have a different pick area if that’s what is needed for the application.
- The PartID that is passed into each of the callback functions will be the Part # of the current Active Part (i.e., the current desired part).
- Images are acquired for every part that is listed in the PF_Start statement and each part’s queue will be loaded with the vision results. Every part uses its own vision and lighting settings. For example, one part could use “system processes vision” and another part in the PF_Start list could use “user processes vision” - all parts use their own settings. Similarly, every part can use its own specific lighting criteria - front lights, no backlight, different brightness etc…
- The Part Feeding Log File (started with the PF_InitLog statement) will contain the data for all the parts that are running on a feeder.
- When multiple robots are accessing the same feeder, the user’s application code must use the PF_AccessFeeder & PF_ReleaseFeeder statements to ensure that the robots cannot collide. This will be covered in more detail throughout the chapter.
- Every part must be calibrated individually. Once again, it is assumed that the parts that are running on a feeder at the same time will have similar characteristics (size, weight, material etc…).

1.2 Key Concepts for Multiple Parts and Multiple Robots

1.2.1 PF_ActivePart

The PF_ActivePart statement notifies the Part Feeding process of the user's intent at runtime. The system needs to know what part is desired. The system will feed (uses the correct vibration settings, supplies parts from the hopper etc…) to ensure that the desired part is available.

As a simple illustration of why PF_ActivePart is needed at runtime, let's consider a "kitting" application where all the parts are being fed by the same Part Feeder. An empty box is delivered to the robot on a conveyor. A barcode on the box indicates what part type and the quantity of parts that need to be placed into the box. The desired part type and the pick quantity are only known when the box is presented to the barcode reader. The Part Feeder needs to vibrate and find the parts that are needed to fill the order.

As another example of how PF_ActivePart is used, let's consider a two-part assembly operation. Both parts are on the same Part Feeder. For this application, the robot needs to pick one of Part #1 and place it in a fixture. Then the robot picks two of Part #2 and inserts them into Part #1. PF_ActivePart tells the system which part it needs to feed so that the assembly process can be completed. To make this application more realistic, let's assume that each part is inspected by a vision system prior to assembly. If Part #1 fails its inspection, PF_ActivePart must remain as Part #1 so that the robot can go back to the feeder and get a good Part #1. If Part#1's inspection passes then PF_ActivePart needs to switch to Part #2.

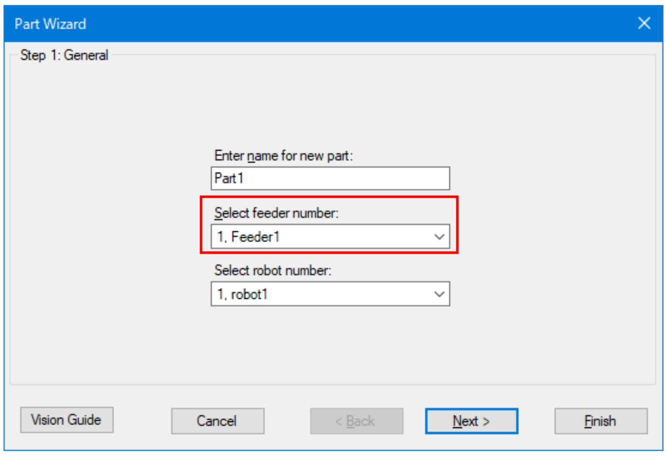
The first Part ID in PF_Start statement is the initial Active Part. PF_ActivePart is normally set prior to exiting the PF_Robot callback so that the feeding action will be specific to the desired part.

PF_ActivePart will be demonstrated by an example in a later section of this chapter.

1.2.2 PF_Start

As previously mentioned, up to four Part types can run on the same feeder at the same time. This is accomplished by specifying each of the Part ID's when executing the PF_Start statement.

For example, Part 1, 2, 3 and 4 all use Feeder #1. The feeder # was specified during the Part Wizard when each of the parts was originally created.


Part Wizard
Step 1: General
Enter name for new part: Part1
Select feeder number: 1, Feeder1
Select robot number: 1, robot1

Only Parts that are assigned to the same feeder can be run together with PF_Start. To run all 4 parts on Feeder #1 at the same time, the code would look like the following –

```
PF_Start 1, 2, 3, 4
```

Part #1 will be the initial Active Part for this example. The feeder will initially use Part #1's vibration parameters. For this example, a value of "1" will be sent as the PartID parameter in each of the callback functions until the PF_ActivePart is changed to a different Part ID.

NOTE

Unless PF_ActivePart is executed with a different PartID, the Active Part will continue to be the first Part in the PF_Start list.

If the user's code attempts to start another part on the same feeder, a "Feeder In Use Error" will occur. The error is handled in the PF_Status callback and the status value will be constant PF__D_STATUS_FEEDERINUSE_ERROR.

Here is an example of how the "Feeder In Use" error can occur->

Parts 1, 2, 3, 4 are all using Feeder #1

```
PF_Start 1, 3, 4 'starts a grouping of parts on Feeder #1
```

'A "Feeder In Use" error will occur if the next line of code is executed

```
PF_Start 2
```

If you want to run all 4 parts on Feeder #1 at the same time, you must execute the following statement instead –

```
PF_Start 1, 2, 3, 4
```

A maximum of 2 robots can use the same feeder at the same time. If PF_Start attempts to run parts that use more than 2 robots on the same feeder at the same time, a "Max Robots Per Feeder" error will occur.

For example ->

Part 1: Uses Robot #1 and Feeder #1

Part 3: Uses Robot #2 and Feeder #1

Part 5: Uses Robot #3 and Feeder #1

`PF_Start 1,3,5` <- Error 7731: The maximum number of simultaneous feeders for the controller type has been exceeded.

1.2.3 Vision and Queue Loading

By default, vision images are acquired for all parts running on the same feeder. There is also a mechanism to acquire the image for only the Active Part (this will be discussed in the next section). Each part uses its specific lighting requirements (front | backlight, mobile camera, user processes vision using the PF_Vision callback etc….).

After all the Part Vision Sequences have run, all of the queues are loaded with the vision results.

TIP

For added efficiency the CameraBrightness and CameraContrast can be set the same for all part vision sequences. The vision sequence for the first part in the PF_Start list would have the RuntimeAcquire property set to Stationary. The remaining part vision sequences using the same feeder would have RuntimeAcquire set to None., This eliminates the time for additional grabs. This method assumes that all parts can be found using the same lighting conditions.

1.2.4 PF_Robot Return Values

All callback functions require you to return a value. Normally the return value will indicate that the operation has completed successfully (constant PF_CALLBACK_SUCCESS). The return value can be used to change how the system operates. The PF_Robot return values redirects the main process flow and provides different behavior depending upon the "use case scenario". Having different return values gives the developer the ability to control the Part Feeding process flow for different situations.

Let's begin by describing each of the return values for the PF_Robot callback function –

PF_CALLBACK_SUCCESS:

When the PF_Robot callback function finishes and "PF_Robot = PF_CALLBACK_SUCCESS", the system will check to see if the next Active Part (desired part) is available in its queue. If the Active Part is available, then the system will call the PF_Robot function again. The PartID parameter that is passed into PF_Robot will be the part # for the Active Part. In the case where the Active Part is available, there is no need to acquire a new image or vibrate the feeder since the desired part is already in the queue. If the Active Part is not available, then the system will acquire new images for every part that was executed in the PF_Start statement. The vision results will be loaded into every part queue and the system will make a judgement of whether to vibrate the feeder.

PF_CALLBACK_RESTART:

When the PF_Robot callback function finishes and "PF_Robot = PF_CALLBACK_RESTART", the system will acquire new images for every part that was executed in the PF_Start statement regardless of whether there are parts available in the Active Part's queue. The vision results will be loaded into all the part queues and the system will make a judgement of whether to vibrate the feeder. The main purpose of the PF_CALLBACK_RESTART return value is to force new images to be acquired, queues reloaded, re-judgement and re-feeding from the beginning of the main Part Feeding process loop. This return value is convenient when you want to acquire new images for every pick and place cycle. For example, if surrounding parts are accidentally disrupted during the pick and place, it may be helpful to acquire new images and refresh the queues.

PF_CALLBACK_RESTART_ACTIVEPART:

When the PF_Robot callback function finishes and "PF_Robot = PF_CALLBACK_RESTART_ACTIVEPART", the system will acquire a new image for only the Active Part (not all the parts listed in the PF_Start statement). This return value is particularly useful when multiple robots are sharing the same feeder. PF_CALLBACK_RESTART_ACTIVEPART can be used to prevent duplication of parts in multiple queues. The return value forces a new image to be acquired for only the Active Part and only the Active Part's queue will be loaded. The queues for all the other parts listed in the PF_Start statement are cleared. This is best illustrated by an example.

Suppose that the feeder has one physical part type on its platform and two robots are picking up parts from the feeder. For this application, two parts are added in the Part Feeding dialog. Each part will have a different Robot #. Additionally, the vision sequences for each part will have a different vision calibration (since the camera is calibrated to a specific robot). Even though there is only one physical part type on the platform, two logical parts must be created (one for each robot). The PF_Start statement will include both part numbers. Vision is acquired for both logical parts and both queues are loaded. Because both vision sequences are locating the same physical parts on the platform, there will be duplication of coordinates in the queues. If robot #1 picks a part and removes it from its queue, the part will remain in the other robot's queue. As a result, robot #2 will attempt to pick up a part that was already removed by robot #1. PF_CALLBACK_RESTART_ACTIVEPART is used to ensure that a part is only loaded into one queue. As a result, no special sorting and no special queue distribution is required. Please refer to the following section for details on how to program this use case scenario.

Software 9.3 Two Robots - One Part

1.2.5 PF_AccessFeeder / PF_ReleaseFeeder

PF_AccessFeeder / PF_ReleaseFeeder locks and unlocks access to a feeder to prevent potential collisions on a multi-robot / one feeder system. These commands are required when two robots are sharing the same feeder at the same time. PF_AccessFeeder is a mutual exclusion lock. If a lock has already been obtained, PF_AccessFeeder will pause the task (i.e., wait its turn) until the lock is released or until the specified timeout (optional) is reached. When a robot finishes using a feeder, it must release the lock using the PF_ReleaseFeeder statement in order to relinquish the feeder to the other robot. PF_ReleaseFeeder can be used inside the "!...!" parallel processing statement of a motion command to allow one robot to approach the feeder as the other robot is departing from the feeder.

The code to prevent robots from colliding while accessing a feeder looks something like the following:

```
PF_AccessFeeder 1
Pick = PF_QueGet(1)
PF_QueRemove (1)
Jump Pick
On gripper
Wait .25
Jump Place ! D80; PF_ReleaseFeeder 1 !
```

1.2.6 PF_Stop

The PF_Stop statement has PartID as a parameter. For a single part system, the PartID is the same as the part that is running. For a multi-part system, the PartID can be the part # of any of the parts that are running on the feeder. In other words, you can specify any part in the PF_Start list. However, please be aware the PartID that is passed to the PF_CycleStop callback function will be the Part ID for the current Active Part.

1.2.7 PF_InitLog

PF_InitLog initiates the Part Feeding log file. All the parts listed in the PF_Start statement will have data logged to the file. Each entry in the log files will include the Part ID for the current Active Part. For example, the number of parts processed inside the PF_Robot callback will be recorded for the current Active Part. Please refer to the following chapter for further details.

Software "5. Part Feeding Log File"

1.2.8 PF_QtyAdjHopperTime

The PF_QtyAdjHopperTime function calculates how much time a hopper should be turned on to supply the "optimal number of parts". The time is calculated from the number of parts that are supplied within a specific amount of time, the approximate number of parts that are currently on the feeder platform and the "Optimal Number of Parts" value that was determined during the "Part Area" calibration for the part. The PF_QtyAdjHopperTime function can be executed anywhere in the user's code. For example, it could be executed before the PF_Start statement in order to supply parts prior to running. In that case, the Part Feeding system has no idea what group of parts will be used on the feeder. Perhaps one part will be running on the feeder or maybe four different parts will be running on the feeder at the same time. If PF_QtyAdjHopperTime is executed prior to PF_Start, the calculation can only guess that the part specified by the PartID parameter will be the only part on the platform. In that case, only the Part Blob vision sequence (for that PartID) is used. If PF_QtyAdjHopperTime is executed after PF_Start has been executed, then the Part Feeding system knows which parts are running on the feeder (since they were specified in the PF_Start statement). In that case, the Part Blob vision sequence (for the supplied PartID) is run as well as each individual Part Sequence. When multiple parts are running at the same time, the system assumes that an equal quantity of each part type is optimal. In some rare cases, however, that may not be true. For example, it may be desirable to have twice as many of Part #1's as Part #2's. In that case, the user's code should scale (multiply or divide) the calculated time that is returned from the PF_QtyAdjHopperTime function in order to supply the desired amount of parts from the hopper.

1.3 Tutorials

The purpose of this section is to demonstrate how to implement both a multi-part application as well as a multi-part / multi-robot application. In many cases we will simply explain the steps to follow but will not explain the details behind what was done. It is assumed that you have already read Introduction chapter 7 *"Let's Use the Part Feeding Option"* and you understand how to create a new part, perform a part calibration and write a program for a one robot / one-part application.

1.3.1 Tutorial #1: 1 Robot, 1 Feeder, 2 Parts

For this tutorial, there are two parts running on the same feeder at the same time. The parts are physically different. Part #1 and Part #2 are being used. There is a Fixed Downward camera over the feeder. The robot will pick and place two of Part #1 and then pick and place one of Part #2. This operation will be performed continuously in a loop. Each part will be picked up with a different gripper (i.e., output bit). The quantity of parts and the pick order are important for this assembly application. The process of alternating between Part #1 and Part #2 will be accomplished using "PF_ActivePart". At first, we will write the code so that the robot motion is performed inside of the PF_Robot callback function. Then we will improve the robot throughput by performing the motion in a separate multitask and by parallel processing the feeder vibration.

(1) Create a new EPSON RC+ project.

(2) Calibrate the Fixed Downward camera.

(3) From Vision Guide, create the Part Blob Sequence. The “Part Blob Sequence” is used for feedback during the Part Calibration and at runtime to make judgements about how to best feed the parts. The Part Blob Sequence detects individual parts or clumps of parts at runtime. The Part Blob Sequence generally contains a single Blob vision object. The Part Blob Sequence will need to have the camera calibration assigned to the Calibration property. The Blob object’s NumberToFind property should be set to “All”. The ThresholdAuto property of the Blob should be set to False (default value). Place parts on the platform, open the Histogram window and adjust the Threshold High and Low until only the parts are detected. Set the Blob’s MinArea to roughly 0.9 times that of the part’s area. If the part has multiple sides, then set the MinArea so that the part can be found in any orientation. You can have separate Part Blob Sequences for each of the two parts (running on the feeder) but in general you can use the same Part Blob Sequence for all the parts. Make sure that the Part Blob Sequence can detect each Part that will be running on the feeder. The Blob object’s Search Window should fill as much of the platform area as possible. That said, it is critical that the Blob object only finds the Parts and not the feeder tray itself. If the Part Blob sequence finds any portion of the tray then the system will not function properly.

(4) From Vision Guide, create the Part Sequence for each of the two Parts that will be running on the feeder. Make sure that the Calibration property for each sequence is set. Typically, a Geometric object is used to locate the Front (and a second Geometric object is typically used to find the Back of the part if Flip is required). The vision object’s NumberToFind property is normally set to “All”.

(5) Go to the [Tools]-[Part Feeding] dialog and add a new part. The “Part Wizard” will walk you through the process of adding a part. For more details about the Part Wizard, please refer to the following chapter.

Introduction 8 “Let’s Use the Part Feeding Option”.

(6) Go to the [Calibration] page for the new Part and click on the <Calibrate> button to start the Calibration Wizard. Please refer to the following section for details on how to use the Calibration Wizard.

Introduction “8.12 Calibration & Test”

(7) Go to the [Pick] page for Part #1 and click the <Teach> button. Jog the robot to the part pick height and teach the “Pick Z”.

(8) Click the <Add> button in the [Part Feeding] dialog to add Part #2. Configure the part using the “Part Wizard”.

(9) Go to the [Calibration] page and click the <Calibrate> button to start the Calibration Wizard.

(10) Go to the [Pick] page for Part #2 and click the <Teach> button. Jog the robot to the part pick height and teach the “Pick Z”.

(11) Close the Part Feeding dialog. The Part Feeding template code (i.e., the Part Feeding callback functions) is automatically created.

(12) Teach robot points for “park” and “place” and label them respectively.

(13) Modify the template code as follows –

Main.prg

```
Function Main
      Robot 1
      Motor On
      Power High
      Speed 50
      Accel 50, 50
      Jump park

      PF_Start 1, 2
Fend
```

PartFeeding.prg

```
Global Integer numPicked ' number of parts that have been picked

Function PF_Robot(PartID As Integer) As Integer

      Integer numRequired, gripperOutput

      Select PartID
            Case 1
                  numRequired = 2   ' number of parts required
                  gripperOutput = 1
            Case 2
                  numRequired = 1
                  gripperOutput = 2
      Send
      Do
            If PF_QueLen(PartID) > 0 Then
                  P0 = PF_QueGet(PartID)
                  PF_QueRemove(PartID)
                  Jump P0
                  On gripperOutput
                  Wait .1
                  Jump place
                  Off gripperOutput
                  Wait .1
                  numPicked = numPicked + 1
            Else
                  ' Not enough parts were picked
                  PF_ActivePart PartID   ' No change in Active Part
                  PF_Robot = PF_CALLBACK_SUCCESS
                  Exit Function
            EndIf
      Loop Until numPicked = numRequired

      numPicked = 0
      ' select the next Active Part
      If PartID = 1 then
            PF_ActivePart 2
      Else
            PF_ActivePart 1
      EndIf
      PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

For the sample code shown above, the feeder will vibrate (if necessary) only after the robot has completed its motion to “place” and the PF_Robot function has finished.

The code can be restructured such that the feeder vibration will occur in parallel with the robot motion. The PF_Robot callback will be used to notify a motion task (function Main in this example) that parts are available to be picked up.

Memory IO (labeled “PartsToPick1” and “PartsToPick2”) are used to signal when parts are available.

When the last available part is being placed (80% of the way through the motion for this example), the motion task signals the PF_Robot function to finish and return a value. The return values let’s the system know that it is ok to acquire new images, vibrate, supply parts from a hopper etc…We will now modify our tutorial code so that the feeder action can occur in parallel with the robot motion.

Main.prg

```
Function Main
       Integer numToPick1, numToPick2, numPicked

       Motor On
       Power High
       Speed 50
       Accel 50, 50
       Jump park

       MemOff PartsToPick1
       MemOff PartsToPick2
       numToPick1 = 2
       numToPick2 = 1

       PF_Start 1,2

       Do
             numPicked = 0

             Do
                   Wait MemSw(PartsToPick1) = On
                   pick = PF_QueGet(1)
                   PF_QueRemove (1)
                   Jump pick
                   On gripper1
                   Wait .1
                   numPicked = numPicked + 1
                   If numPicked < numToPick1 And PF_QueLen(1) > 0 Then
                         Jump place
                   Else
                         ' Last part or no more parts available to pick
                         If numPicked = numToPick1 Then
                              ' Select the next part
                              PF_ActivePart 2
                         EndIf
                         Jump place ! D80; MemOff PartsToPick1 !
                   EndIf
                   Off gripper1
                   Wait .1
             Loop Until numPicked = numToPick1
```

```
            numPicked = 0
            Do
                  Wait MemSw(PartsToPick2) = On
                  pick = PF_QueGet(2)
                  PF_QueRemove (2)
                  Jump pick
                  On gripper2
                  Wait .1
                  numPicked = numPicked + 1
                  If numPicked < numToPick2 And PF_QueLen(2) > 0 Then
                        Jump place
                  Else
                        ' Last part or no more parts available to pick
                        If numPicked = numToPick2 Then
                             ' Select the next part
                             PF_ActivePart 1
                        EndIf
                        Jump place ! D80; MemOff PartsToPick2 !
                  EndIf
                  Off gripper2
                  Wait .1
            Loop Until numPicked = numToPick2
      Loop
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
      Select PartID
            Case 1
                  MemOn PartsToPick1
                  Wait MemSw(PartsToPick1) = Off
            Case 2
                  MemOn PartsToPick2
                  Wait MemSw(PartsToPick2) = Off
      Send

      PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

1.3.2 Tutorial #2: 2 Robots, 1 Feeder, 2 Parts

For this tutorial, there are two parts running on the same feeder at the same time. The parts are physically different. Part #1 and Part #2 are being used. There is a Fixed Downward camera over the feeder. Two robots will be sharing the same feeder. The robots will take turns picking from the feeder. Each robot will pick and place its own part. Robot #1 will pick up one of Part #1 and then Robot #2 will pick up one of Part #2. The pick order matters for this application. The alternating pick order is accomplished with "PF_ActivePart". At first, we will write the code so that the robot motion is performed inside of the PF_Robot callback function. In this case, the robot motion will be sequential. In other words, only one robot will move at a time. Then we will improve the robot throughput by performing motion in a separate multitasks. The revised tutorial will also include PF_AccessFeeder / PF_ReleaseFeeder. PF_AccessFeeder / PF_ReleaseFeeder are used to ensure that the robots will not collide when picking from the feeder.

(1) Create a new EPSON RC+ project.

(2) Calibrate the Fixed Downward camera for Robot #1.

(3) Calibrate the Fixed Downward camera for Robot #2.

(4) From Vision Guide, create the Part Blob Sequence for Part #1. The Part Blob Sequence generally contains a single Blob vision object. The sequence will need to have the camera calibration that was performed for Robot #1 assigned to the Calibration property. The Blob object's NumberToFind property should be set to "All". The ThresholdAuto property of the Blob should be set to False (default value). Place parts on the platform, open the Histogram window and adjust the Threshold High and Low until only the parts are detected. Set the Blob's MinArea to roughly 0.9 times that of the part's area. If the part has multiple sides, then set the MinArea so that the part can be found in any orientation. Make sure that the Part Blob Sequence can detect Part #1. The Blob object's Search Window should fill as much of the platform area as possible. That said, it is critical that the Blob object only finds the Part and not the feeder tray itself.

(5) From Vision Guide, create the Part Blob Sequence for Part #2. The Part Blob Sequence generally contains a single Blob vision object. The sequence will need to have the camera calibration that was performed for Robot #2 assigned to the Calibration property. The Blob object's NumberToFind property should be set to "All". The ThresholdAuto property of the Blob should be set to False (default value). Place parts on the platform, open the Histogram window and adjust the Threshold High and Low until only the parts are detected. Set the Blob's MinArea to roughly 0.9 times that of the part's area. If the part has multiple sides, then set the MinArea so that the part can be found in any orientation. Make sure that the Part Blob Sequence can detect Part #2. The Blob object's Search Window should fill as much of the platform area as possible. That said, it is critical that the Blob object only finds the Part and not the feeder tray itself.

(6) From Vision Guide, create the Part Sequence which will be used to find Part#1. Make sure to set the Calibration property to the calibration that was performed for Robot #1. Typically, a Geometric object is used to locate the Front (and a second Geometric object is typically used to find the Back of the part if Flip is required). The vision object's NumberToFind property is normally set to "All".

(7) From Vision Guide, create the Part Sequence which will be used to find Part#2. Make sure to set the Calibration property to the calibration that was performed for Robot #2. Typically, a Geometric object is used to locate the Front (and a second Geometric object is typically used to find the Back of the part if Flip is required). The vision object's NumberToFind property is normally set to "All".

(8) Go to the [Tools]-[Part Feeding] dialog and Add a new part for Part #1. The "Part Wizard" will walk you through the process of adding a part. Make sure that you select Robot #1 on the first page of the Part Wizard. For more details about the Part Wizard, please refer to the following chapter.

Introduction chapter 7 "Let's Use the Part Feeding Option"

(9) Go to the [Calibration] page for the new Part #1 and click on the <Calibrate> button to start the Calibration Wizard. Please refer to the following section for details on how to use the Calibration Wizard.

Introduction "8.12 Calibration & Test"

(10) Go to the [Pick] page for Part #1 and click the <Teach> button. Jog Robot #1 to the part pick height and teach the "Pick Z".

(11) Click the <Add> button in the [Part Feeding] dialog to add Part #2. Configure the part using the "Part Wizard". Make sure that you select Robot #2 on the first page of the Part Wizard.

(12) Go to the [Calibration] page and click the <Calibrate> button to start the Calibration Wizard.

(13) Go to the [Pick] page for Part #2 and click the <Teach> button. Jog Robot #2 to the part pick height and teach the "Pick Z".

(14) Close the [Part Feeding] dialog.
The Part Feeding template code (i.e., the Part Feeding callback functions) is automatically created.

(15) Modify the template code as follows –

Main.prg

```
Function Main
       Robot 1
       Motor On
       Power High
       Speed 50
       Accel 50, 50
       Jump park
       Robot 2
       Motor On
       Power High
       Speed 50
       Accel 50, 50
       Jump park

       PF_Start 1, 2
Fend
```

PartFeeding.prg

```
Function PF_Robot(PartID As Integer) As Integer
       If PF_QueLen(PartID) > 0 Then
             Select PartID
                   Case 1
                         Robot 1
                         P0 = PF_QueGet(1)
                         PF_QueRemove (1)
                         Jump P0 /R
                         On rbt1Gripper
                         Wait .25
                         Jump place
                         Off rbt1Gripper
                         Wait .25
                         PF_ActivePart 2
                   Case 2
                         Robot 2
                         P0 = PF_QueGet(2)
                         PF_QueRemove (2)
                         Jump P0 /L
                         On rbt2Gripper
                         Wait .25
                         Jump place
                         Off rbt2Gripper
                         Wait .25
                         PF_ActivePart 1
             Send

       EndIf

       PF_Robot = PF_CALLBACK_SUCCESS

Fend
```

We will now modify the example code so that the robot motion will be performed in separate multitasks. When one robot is leaving the feeder, the other robot can begin moving toward the feeder. When robots share a feeder with parallel motion, it is critical that the PF_AccessFeeder and PF_ReleaseFeeder commands are used to prevent robot collisions.

In addition, the revised code will parallel process the feeder vibration and robot motion. In this tutorial, when each robot is 80% of the way to its place position, the robot has cleared the camera's field of view and an image can be acquired.

Furthermore, when each robot is 80% of the way to its place position, it is safe for the other robot to begin moving toward the feeder. Of course, this is just an example. The actual percentage of motion depends on the speed and relative positioning of the robots in your specific situation. Each robot has a point labeled "park" and a point labeled "place". For this tutorial, Robot #1 picks from the feeder in a Righty arm orientation and Robot #2 picks from the feeder in a Lefty arm orientation.

Here is the revised template code -

Main.prg

```
Function Main
       Robot 1
       Motor On
       Power High
       Speed 50
       Accel 50, 50
       Jump park
       Robot 2
       Motor On
       Power High
       Speed 50
       Accel 50, 50
       Jump park
       MemOff PartsToPick1
       MemOff PartsToPick2

       PF_Start 1, 2
       Xqt Robot1PickPlace
       Xqt Robot2PickPlace
Fend

Function Robot1PickPlace
       Robot 1
       Do
            Wait MemSw(PartsToPick1) = On
            PF_AccessFeeder 1
            P0 = PF_QueGet(1)
            PF_QueRemove (1)
            Jump P0 /R
            On rbt1Gripper
            Wait .25
            Jump place ! D80; MemOff PartsToPick1; PF_ReleaseFeeder 1 !
            Off rbt1Gripper
            Wait .25
       Loop
Fend
```

```
Function Robot2PickPlace
       Robot 2
       Do
             Wait MemSw(PartsToPick2) = On
             PF_AccessFeeder 1
             P0 = PF_QueGet(2)
             PF_QueRemove (2)
             Jump P0 /L
             On rbt2Gripper
             Wait .25
             Jump place ! D80; MemOff PartsToPick2; PF_ReleaseFeeder 1 !
             Off rbt2Gripper
             Wait .25
       Loop
Fend
```

<u>PartFeeding.prg</u>

```
Function PF_Robot(PartID As Integer) As Integer
       Select PartID
             Case 1
                   If PF_QueLen(1) > 0 Then
                         MemOn PartsToPick1
                         Wait MemSw(PartsToPick1) = Off
                         PF_ActivePart 2
                   Else
                         PF_ActivePart 1
                   EndIf
             Case 2
                   If PF_QueLen(2) > 0 Then
                         MemOn PartsToPick2
                         Wait MemSw(PartsToPick2) = Off
                         PF_ActivePart 1
                   Else
                         PF_ActivePart 2
                   EndIf
       Send
       PF_Robot = PF_CALLBACK_SUCCESS
Fend
```

1.4 Multi-Part / Multi-Robot Summary

Here is a brief summary of the key concepts that are required for a multiple part / multiple robot Part Feeding application –

A) **PF_Start** allows you to specify up to 4 parts that you want to run on the same feeder at the same time

B) **PF_ActivePart** allows you to select which part is currently desired. The system will vibrate to ensure that the desired part is available.

C) Up to 2 robots can access the same feeder at the same time. **PF_AccessFeeder** & **PF_ReleaseFeeder** ensure that both robots can safely pick from the feeder without the possibility of collision.

D) The **PF_Robot Return Value** controls the main process flow.

The following return values provide different functionality -

PF_Robot = PF_CALLBACK_SUCCESS

If the PF_ActivePart (i.e., the desired part) is available, the system will call the PF_Robot function again without re-acquiring vision images or reloading the part queues. If the PF_ActivePart is not available, new images are acquired each of the parts and the part queues are reload.

PF_Robot = PF_CALLBACK_RESTART

The system will acquire new images for every part that was executed in the PF_Start statement and reload all the part queues.

PF_Robot = PF_CALLBACK_RESTART_ACTIVEPART

The system will acquire a new image for the PF_ActivePart only. The PF_ActivePart's queue will be loaded with the vision results and all other part queues will be cleared.

Please refer to the following section for additional information on multiple parts & multiple robots.

Application programming examples in *Introduction 8.3 Tutorials*

2. Platform Types

The following section decides how and when to use each of the platform types.

2.1 Standard Platform Types

There are 3 standard platform types – Flat, Anit-stick and Anti-roll. All standard platforms are available in either white or black.

2.1.1 Platform color

For most applications the best imaging is achieved using the feeder's built-in LED backlight. When using the built-in backlight, a white translucent tray is used. Backlighting provides a silhouette of the part's outline. Surface features of the part will not be visible to the vision system. In some cases, the maximum image contrast can be achieved with a black platform and using custom front lighting. In general, however, the system is meant to be used with the backlight option. Consider whether the part can be found (right side up | upside down, angular orientation etc …) when it is backlit. Sometimes transparent and semi-transparent objects have poor contrast when backlit. Different colored backlights can be beneficial for transparent parts.

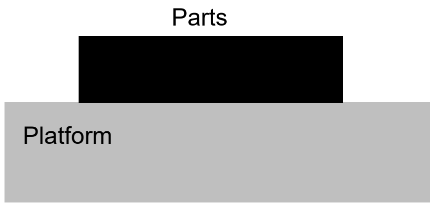
2.1.2 Platform Material

Depending upon the feeder model, different material may be available for Antistatic ESD or Medical Grade applications (refer to the specific feeder hardware manual for details).

2.1.3 Standard Platforms Usage

- **Flat:** Parts that have a stable orientation when seated on a tabletop can use a Flat Platform. The parts should have a stable equilibrium and fast stabilization time after vibration. For high-mix low-volume production, most applications use a Flat Platform.

Appearance



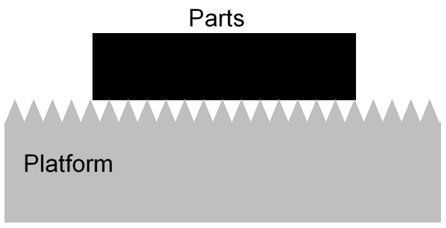
Cross-sectional view

The platforms supplied by Epson meet a flatness and parallelism specification to ensure picking precision as summarized in the table below.

	IF-80	IF-240	IF-380 / IF-530
Flatness of the surface [mm]	0.1	0.2	0.6
Parallelism between surface and reference [mm]	0.1	0.5	0.6

- **Anti-Stick:** Anti-stick platforms have narrow grooves to reduce surface contact for flat and light components. This reduces friction forces and improves the component movement on the platform surface. Parts that do not spread well because of kinetic friction (sliding friction or dynamic friction) are a good candidate for Anti-Stick platforms.

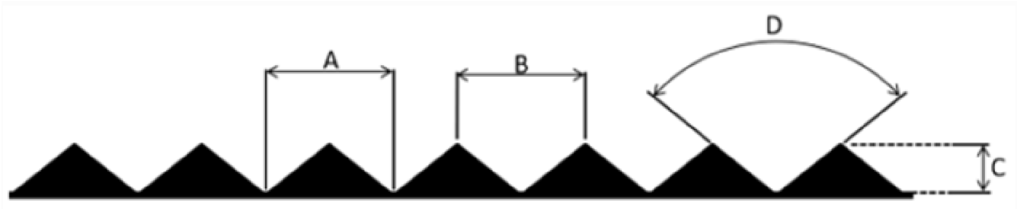
Surface



Cross-sectional view

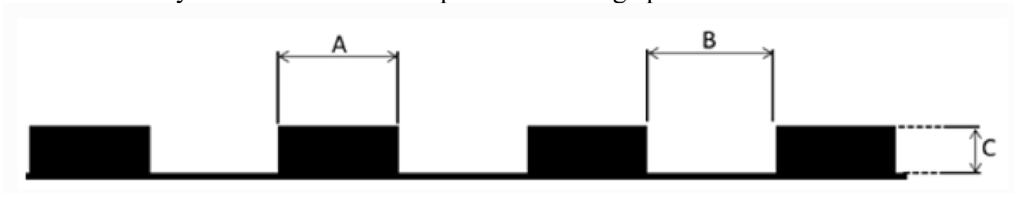
	A	B	C	D
IF-80	0.4	0.4	0.2	90

Geometry of standard anti-stick platform for small parts



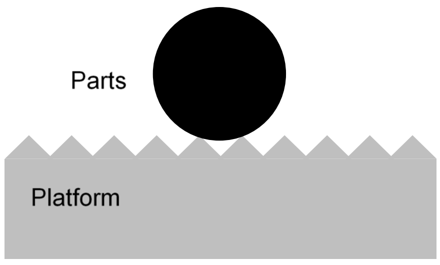
	A	B	C
IF-240	0.7	1.3	0.5

Geometry of standard anti-stick platform for large parts



- **Anti-Roll:** Anti-Roll platforms have a machined, structured surface that can stabilize parts that tend to roll on the platform. The Anti-Roll platform is particularly useful when cylindrical components are being fed. The Anti-Roll platform reduces the stabilization time by preventing the parts from rolling.

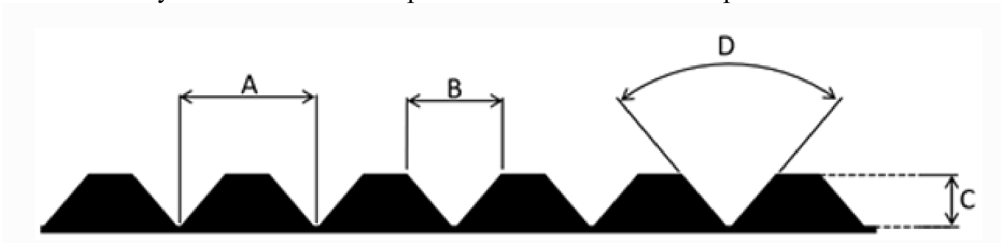
Surface



Cross-sectional view

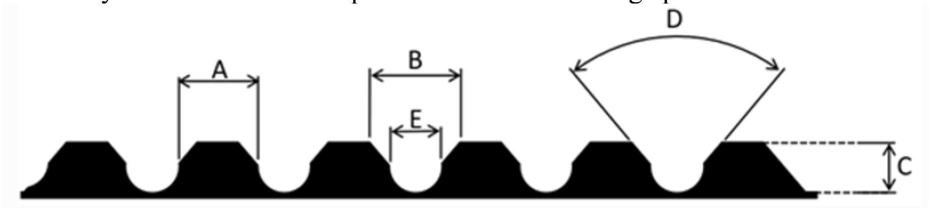
	A	B	C	D	Suitable for Parts
IF-80	1.25	1	0.5	90	ø 0.7mm – ø 1.5mm
IF-80	2.75	2.5	1.25	90	ø 1.5mm – ø 3.5mm
IF-240	1.25	1	0.5	90	ø 1.7mm – ø 3.5mm
IF-240	3	2.5	1.25	90	ø 3.5mm – ø 7mm
IF-240	5.5	5	2.5	90	ø7mm – ø 14mm
IF-380	3	2.5	0.722	120	ø 3mm – ø 5mm
IF-380	5.5	5	1.443	120	ø 5mm – ø 10mm
IF-530	6.5	6	1.732	120	ø 6mm – ø 12mm

Geometry of standard anti-roll platform structure for small parts



	A	B	C	D	E	Suitable for Parts
IF-380	10.5	12	5.31	120	2	ø 10mm – ø 24mm
IF-530	12.5	14	4.9	120	4	ø 12mm – ø 28mm

Geometry of standard anti-roll platform structure for large parts



2.2 Custom Platforms

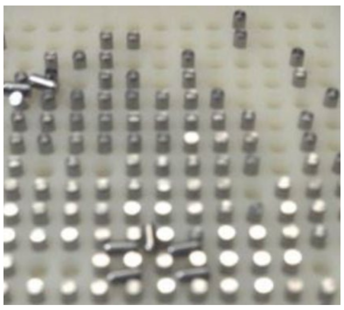
NOTE Custom platforms need to be designed and manufactured by the customer.

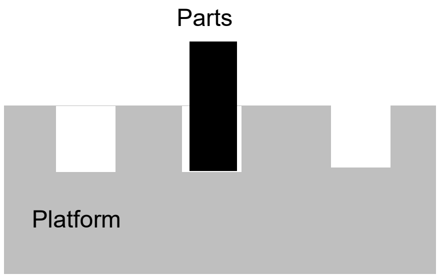
2.2.1 Basic designs for custom platforms

There are 3 basic designs for custom platforms – Holes, Slots, and Pockets. Custom platforms must be designed and manufactured by the user. In the case of a custom platform, the goal is to sufficiently pre-orientate parts in Holes, Slots and Pockets such that the desired cycle time is obtained. A custom platform is necessary to orient the part correctly when the natural resting position of the parts does not match the picking orientation.

The following summarizes what parts are typically used with each platform types:

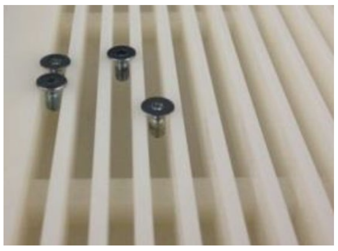
- **Holes:**

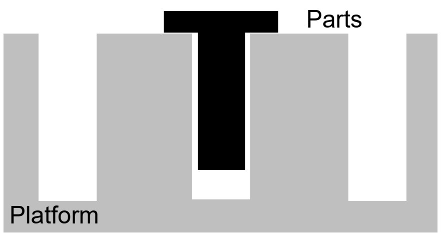

Surface



Cross-sectional view

- **Slots:**

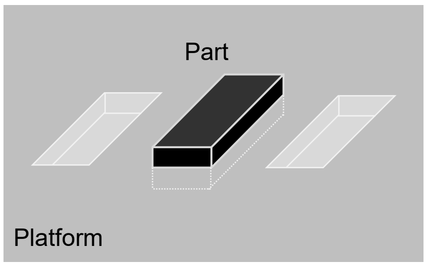

Surface



Cross-sectional view

- **Pockets:**

Surface



Oblique view

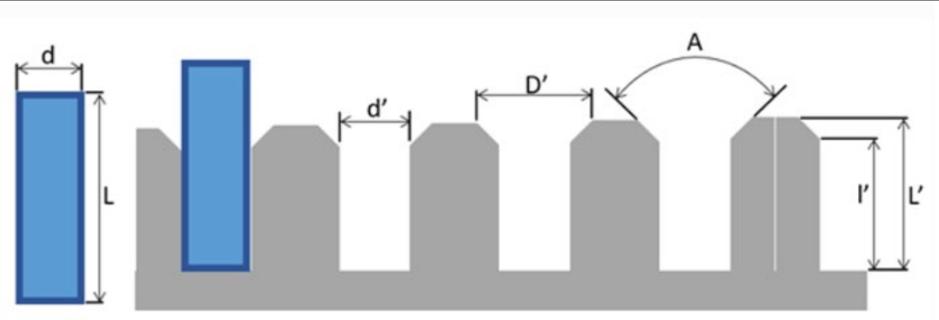
2.2.2 Guidelines for Custom Platform Design

Holes:

Holes are useful when cylindrical components are to be fed and presented upright.
The basic design for a platform with holes should consider the following items:

- In general, the simpler design is better.
- The diameter of the hole (d’) is the most important dimension of the plate. It is mandatory to have a large enough diameter to allow the part to straighten out once inside the hole. For parts up to 3.5mm of diameter, 0.05 mm is added to the maximum part diameter (d). However, for larger parts, with diameter larger than 3.5mm, as well as non-cylindrical parts (e.g. conical), a larger diameter is required.
- The hole should be deep enough (l’) to allow the part to be guided along the walls if necessary. If the workpiece rests on the bottom, there is no need for a long guide.
- It is preferable to guide the part at least on 1/3 of the height of the part (L), ideally ½, so that the piece stays as straight as possible.
- Attention should also be paid to the residual height of the part above the plate (L’).
- The chamfer (A) is essential to allow the part to fall into the hole. In most cases, the chamfer angle is 60°.
- The weight of the custom platform should be the same as the weight of the standard flat platform. If the weight changes significantly, the resonant frequency may change and affect the operation of the feeder. In this case, it may be possible to adjust the frequency of each feeder operation.

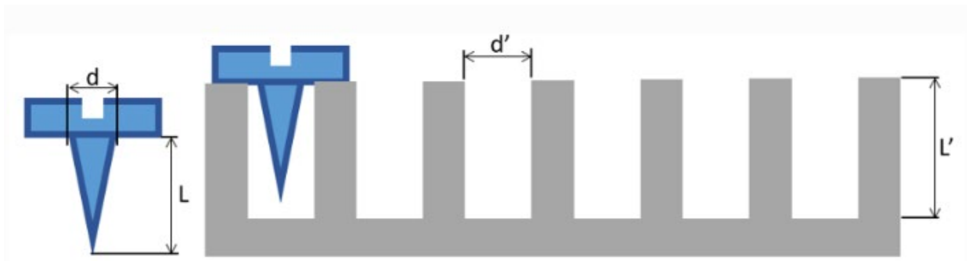
	D’	d’	l’	A
IF-80	>0.5*L	d+0.05mm	0.5*L	60
IF-240	>0.5*L	d+0.1mm	0.5*L	60
IF-380	>0.5*L	d+0.5mm	0.5*L	60
IF-530	>0.5*L	d+1.0mm	0.5*L	60



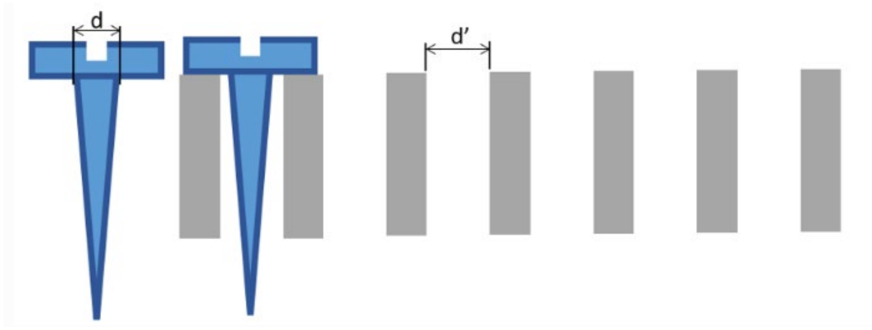
Slots:

A structured platform with slots is used to supply screw-type components to be fitted vertically.
The basic design for a platform with slots should consider the following items:

- In general, the simpler design is better.
- A platform with non-traversing grooves is used to supply components with a maximum length of 60 mm. For longer parts, grooves are manufactured through the whole thickness of the plate.
- The width of the slot (d') is determined by the diameter of the component (d). In general, 0.05mm to 0.1mm is added to the maximum diameter size (allow for tolerance). The tolerance on the width of the slot, which will depend on the machining and is not identical on all sizes of feeder, must also be taken into consideration. A 0/+ tolerance is preferred.
- The depth of the slot (L') is usually not very important because the workpiece does not rest on the bottom. It must therefore be large enough to allow the part to tip and not touch the bottom. For the tolerance, a 0/+ tolerance is usually used.



Non-traversing slots can be manufactured for parts up to 60mm long structure for large parts.



For parts longer than 60mm, traversing slots must be manufactured through the entire thickness of the plate.

	d'	L'
IF-80	d+0.05mm	L'>L
IF-240	d+0.1mm	L'>L
IF-380	d+0.5mm	-
IF-530	d+1.0mm	-

- If the slot goes through the bottom of the platform, an "internal diffuser" is required to prevent the operator from seeing the LED backlight directly and to prevent the backlight light from entering the camera directly. The "internal diffuser" is placed above the backlight.
- The weight of the custom platform should be the same as the weight of the standard flat platform. If the weight changes significantly, the resonant frequency may change and affect the operation of the feeder. In this case, it may be possible to adjust the frequency of each feeder operation.

Pockets:
The basic design for a platform with pockets should consider the following items:

- The pockets should be designed such that the parts can be easily grasp by the robot. Ideally the parts will be pre-oriented such that a flat, parallel surface on the part can be picked up by a vacuum cup gripper.
- The part does not need to be perfectly aligned in the pocket. The purpose of the vision system is to provide the position and rotation of the part with in a 2D plane. Once again, the simpler the design, the cheaper the platform. Moreover, a simple platform design typically functions better.

These are only general guidelines. The specific design should be adapted on a case-by-case basis.

2.2.3 Permissible Platform Weight

	Maximum weight of the platform **1	Maximum weight of component **2
IF-80	150 g	50 g
IF-240	800 g	400 g
IF-380	4 kg	1.5 kg
IF-530	5 kg	2 kg

**1

For the IF-80/IF-240, it represents the maximum platform weight (without components).

For the IF-380/IF-530, it represents the maximum weight of the frame + platform assembly (without components).

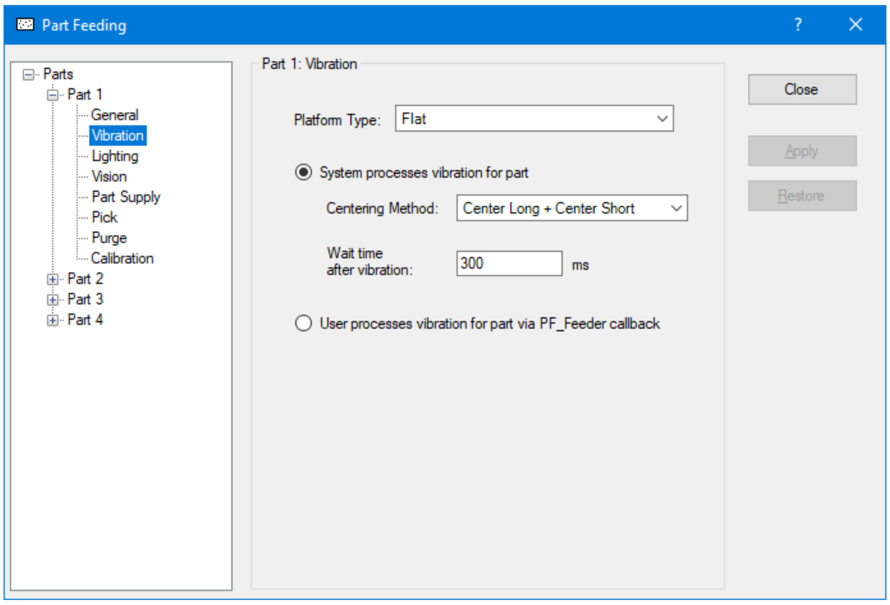
**2

For the IF-80/IF-240, it represents the maximum weight of components (without platform).

For the IF-380/IF-530, it represents the maximum weight of components (without frame + platform assembly).

2.3 Platform Selection

Platform Type selection is made from the Part Feeding dialog EPSON RC+ 7.0-Menu-[Tools]-[Part Feeding]-[Vibration Page]. (See also Software "2.3.2 Vibration")



Platform Type: Flat

(●) System processes vibration for part

Centering Method: Center Long + Center Short

Wait time after vibration: 300 ms

( ) User processes vibration for part via PF_Feeder callback

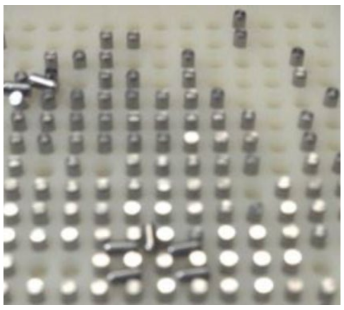
When a standard platform type (Flat, Anit-stick and Anti-roll) is selected, the user has the option of letting the System process the feeder vibration or the User can handle the vibration via the PF_Feeder callback. Typically, it is best for the System to process the feeder vibration.

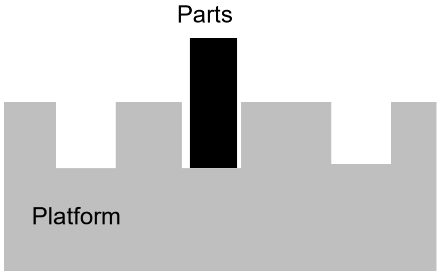
When "User processes vibration for part via PF_Feeder callback" is selected on the Vibration page, the user decides how to feed the parts. The System will make a recommendation of how to vibrate the feeder. This recommendation is provided to the PF_Feeder callback as the "state" parameter. This will be explained in more detail in the next section.

When a custom plate type (Slots, Holes and Pockets) is selected, the user must handle the vibration for the part via the PF_Feeder callback. In the case of Custom platforms, the system has no knowledge of how the plate is machined and consequently, the system can not properly determine how to best feed the parts. (Because the optimum vibration type changes depending on the processing of the platform.)

2.3.1 Example for Custom Platform

For this example, a custom platform with holes has been designed to vertically pre-orient pins.


Surface



Cross-sectional view

When the user wants to run the vision and load the part queue themselves, select Menu-[Tool]-[PartFeeding]-[Parts]-[Vision]- "User Processes Vision via PF_Vision callback". For this example, however, the System will process the Vision.
The robot pick & place will be performed inside the PF_Robot callback.
The Platform Type has been selected as "Holes". Because this is a custom platform, the "User processes vibration for part via the PF_Feeder callback" is the only allowable selection.
The top of the pins is found by the Part Sequence and the coordinates are automatically loaded into the part queue. For this example, no vision object is being used to find the back of the part.
After vision acquires an image and Front parts are loaded into the part queue, the PF_Feeder callback is called. The user's code judges how to vibrate the feeder inside the PF_Feeder callback. The quantity of "Front" parts is provided to the PF_Feeder callback as the parameter called "NumFrontParts". For this example, if the "NumFrontParts" is greater than 0 then no vibration is required since parts are available to be picked up by the robot. In this case, the PF_Feeder return value will need to be set to "PF_CALLBACK_SUCCESS". This return value tells the system to go ahead and call the PF_Robot callback.
If the "NumFrontParts" equals 0 then the sample code VRun's the Part Blob sequence to determine if there is a clump of parts or whether there are no parts at all. If the Part Blob sequence does not find any parts, then the hopper is turned on to supply parts. If the Part Blob sequence finds something then the feeder Flips, Shifts Forward and then Shifts Backward so that the pins can fall into the holes. Whenever parts have been vibrated on the feeder, the system will need to re-acquire new vision images. This is accomplished by setting the return value to "PF_CALLBACK_RESTART".
"PF_CALLBACK_RESTART" will restart the Part Feeding process from the beginning, re-aquire new images, reload the part queue and then call PF_Feeder once again to determine if any further action is required.

TIP

A Flip, a long duration Shift Forward and a short duration Shift Backward is the typical feeding strategy for Custom Platforms. See section *9.5.2 Program Example 5.2* for more details.

NOTE

The constant PF_FEEDER_UNKNOWN is passed to PF_Feeder when the Platform Type is Holes, Slots or Pockets. In the case of Custom Platforms, the system has no knowledge of how the surface is machined and consequently, the system can not properly determine how to best feed the parts. See section *9.5.2 Program Example 5.2* for more details.

```
Function PF_Feeder(PartID As Integer, NumFrontParts As Integer,
NumBackParts As Integer, state As Integer) As Integer

' Example for Structured Platform with holes state = PF_FEEDER_UNKNOWN

    Integer PFControlReturnVal
    Integer numFound

Select True

    ' OK to Pick
    Case NumFrontParts > 0
    ' Call PF_Robot because there are parts ready to pick
        PF_Feeder = PF_CALLBACK_SUCCESS

    ' No Front parts were found but there are Back parts
    Case NumFrontParts = 0 And NumBackParts <> 0

        ' Flip, long Shift Forward and short Shift Backward
        PF_Flip PartID, 500
        PF_Shift PartID, PF_SHIFT_FORWARD, 1000
        PF_Shift PartID, PF_SHIFT_BACKWARD, 300

        PF_Feeder = PF_CALLBACK_RESTART ' Restart and re-acquire images

    ' There are no Front or Back parts found
    ' Either there is a clump of parts or there are no parts on the tray
    ' Acquire an image from the Part Blob sequence to make a determination
    Case NumFrontParts = 0 And NumBackParts = 0

        PF_Backlight 1, On      ' Backlight On
        VRun PartBlob           ' Acquire images
        PF_Backlight 1, Off     ' Backlight Off
        VGet PartBlob.Blob01.NumberFound, numFound    ' Were any Blobs found?

        If numFound > 0 Then    ' Clump of parts found

            ' Flip, long Shift Forward and short Shift Backward
            PF_Flip PartID, 500
            PF_Shift PartID, PF_SHIFT_FORWARD, 1000
            PF_Shift PartID, PF_SHIFT_BACKWARD, 300

        Else    ' No parts found

            ' Call the Control callback to supply more parts
            PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)

        EndIf

            PF_Feeder = PF_CALLBACK_RESTART ' Restart and re-acquire images
    Send

Fend
```

2.3.2 Example for Standard Flat Platform Using the PF_Feeder Callback

For this example, a standard flat platform is being used.
Normally the system would determine how to best handle the vibration for a Flat plate. For this example, however, the user has determined that the vibration requires some special operation which will requires handling the vibration themselves. When the Platform Type is Flat, Anti-Stick or Anti-Roll, the system can make the judgement of how to best vibrate the parts. The system's judgement is provided to the PF_Feeder callback using a parameter called "state".
The different states are defined with constants in the "PartFeeding.inc" file. For example, the constant "PF_FEEDER_PICKOK" means that parts are available to be picked up by the robot. As another example, the constant "PF_FEEDER_FLIP" is passed to the PF_Feeder callback when the system has determined that the best action is to Flip the parts.Conceptually, the user could recreate the system processing by using the PF_Feeder "state"and the corresponding vibration statements.
The following sample demonstrates the basic concept of how to use the "state" parameter to perform the recommended action. This example is not meant to be all inclusive. Refer to section *9.5.1 Program Example 5.1* for a more complete example.

```
Function PF_Feeder(PartID As Integer, NumFrontParts As Integer,
NumBackParts As Integer, state As Integer) As Integer

    Integer PFControlReturnVal, PFStatusReturnVal
    Boolean PFPurgeStatus

    Select state

        Case PF_FEEDER_PICKOK
            PF_Feeder = PF_CALLBACK_SUCCESS ' Call PF_Robot because there are parts ready to
                                            ' be be picked

        Case PF_FEEDER_SUPPLY
            ' Hopper Supply
            PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)
            PF_CenterByShift PartID ' Center parts after hopper supply
            PF_Feeder = PF_CALLBACK_RESTART ' Restart and re-acquire images

        Case PF_FEEDER_FLIP
            PF_Flip PartID
            PF_Feeder = PF_CALLBACK_RESTART ' Restart and re-acquire images

        Case PF_FEEDER_CENTER_FLIP
            PF_Center PartID, PF_CENTER_LONG_AXIS, 900
            PF_Center PartID, PF_CENTER_SHORT_AXIS
            PF_Flip PartID
            PF_Feeder = PF_CALLBACK_RESTART ' Restart and re-acquire images

        Case PF_FEEDER_HOPPER_EMPTY
            ' Notify user that the hopper is empty
            PFStatusReturnVal = PF_Status(PartID, PF_STATUS_NOPART)
            ' Supply parts from hopper
            PFControlReturnVal = PF_Control(PartID, PF_CONTROL_SUPPLY_FIRST)
            PF_Center PartID, PF_CENTER_LONG_AXIS
            ' Center, Flip and Separate parts
            PF_Center PartID, PF_CENTER_SHORT_AXIS
            PF_Flip PartID
            PF_Feeder = PF_CALLBACK_RESTART ' Restart and re-acquire images

        Case PF_FEEDER_WRONGPART
            If PF_Info(PartID, PF_INFO_ID_OBJECT_PURGE_ENABLED) Then
            ' If Purge is enabled then Purge using vision feedback
            ' Each purge attempt will last for 1500 msec
            ' 0 parts can remain on the platform
            ' 5 retries will be attempted
                PFPurgeStatus = PF_Purge(1, 2, 1500, 3, 5)
                If PFPurgeStatus = False Then
                    Print "Purge was not successful"
                    Quit All
                EndIf
            Else ' Notify user that the wrong part may be on the feeder
                Print "Wrong part may be on the feeder"
                Quit All
            EndIf

    Send
Fend
```

Troubleshooting

Troubleshooting

Don’t know the IP address of the feeder

(1) Set the IP address of the feeder to the default value according to *EPSON RC+7.0 Option Part Feeding 7.0 IF-240 4.3.2 Recover IP address using default IP address*.

(2) *Use* the default IP address in EPSON RC+ 7.0 and connect to the feeder according to section *2.1. System Configuration* in the Hardware manual. Then, change the IP address.

Feeder does not vibrate or vibration is weak

- Is the feeder LED lit up or flashing? If not lit up or flashing, it is possible that power is not being supplied.
- If a message displayed, perform the recovery procedures indicated for the corresponding error message.
- It is possible that feeder calibration has failed. Refer to *2.4 Calibration&Test* of the Software manual to perform feeder calibration again.
  If that doesn't work, load the default values and run the calibration again.

If none of the above is applicable, it is possible that the feeder has a malfunction.
Please inquire with us.

Parts in the feeder do not move smoothly or are separated unevenly

- It is possible that the rigidity of the frame to which the feeder is mounted is low, thereby prevent vibrational energy to be properly transmitted to the platform.
  Replace with a highly rigid frame.
- It is possible that feeder calibration has failed. Refer to *2.4 Calibration&Test* of the Software manual to perform feeder calibration again.
  If that doesn't work, load the default values and run the calibration again.
- It is possible that the platform has become worn due to use over a long period.
  The platform is a consumable part. Replace the platform.

If none of the above is applicable, it is possible that the feeder has a malfunction. Please inquire with us.

Hopper does not vibrate

- If the hopper was purchased from Epson, check that the hopper and feeder are connected properly.
  Refer to 6. *Toubleshooting* of the Hardware (Hopper) manual.
- Check that settings for communication with the hopper are properly specified.
- Refer to *4. Part Feeding Callback Function - PF_Control* of the Software manual and check that hopper operation programming has been properly performed.

Parts completely fill up the platform

- It is possible that too many parts are loaded in the feeder tray for a single hopper operation. Properly adjust the quantity loaded from the hopper.

Parts on the platform run out

- Check that there are parts in the hopper.
- It is possible that parts from the hopper are not being properly loaded. Adjust the quantity loaded from the hopper.
- If the hopper was purchased from Epson, check that the hopper and feeder are connected properly.

Trouble questionnaire

[Basics] month/date/year:

Your company	Department
Name	TEL : E-MAIL :
Manipulator model/Serial number /	Controller model name/Serial number /
Date of trouble occurred	Occasion Tooling/during production others ( )

[Troubleshooting report]

No.	Check item	Countermeasure (summary)	Result	Notes
1	Vision error is occurred?	Cancel vision error then try reactivating the feeder system.	Trouble solved Yes/No Not applicable	
2	Feeder error occurred during EPSON RC+ operation.	Refer to *8. Errors that Occur While Using EPSON RC+* and try resolve the obstacle. Check the countermeasure is working by testing feeder communication from EPSON RC+.	Trouble solved Yes/No Not applicable	
3	Feeder error 2582 occurred.	Perform measures of troubleshooting in the manual and try resolve the obstacles. Check the countermeasure is working by testing feeder communication from EPSON RC+.	Trouble solved Yes/No Not applicable	If error occurs even after taking all the measures described in 1 to 3, there is a problem with the feeder body. Try 4 and after.
4	EPSON RC+LED (Power and S-Power) of the feeder does not light up.	Check for the feeder power supply.	Trouble solved Yes/No Not applicable	If there is no problem with the feeder power supply, there is a problem with the feeder body. Try 5 and after.
5	Backlight will not light up.	Exchange the backlight then try testing backlight with EPSON RC+.	Trouble solved Yes/No Not applicable	If not light up even after backlight test, there is a problem with the feeder body. Try 6 and after.
6	In other case	Test calibration from EPSON RC+ and check the motion of the feeder.	Trouble solved Yes/No Not applicable	If not vibrate as configured, there is a problem with the feeder body.

[Other notices, questions for Epson]