Manual

FlexiVision

Kassow Robots Plug-In

ars | Feeding Industrial Robotics

FlexiBowl® by ars

TABLE OF CONTENTS

1. Installing the Plug-In
2. Using the Plug-in
3. Plug-In Functions
    1. Communication Functions
    2. Recipe Management Functions
    3. Vision and Movement Functions
    4. Value Acquisition Functions
4. Implementation Example
5. FlexiVision® command List

This Plug-In was developed with the aim of facilitating communication between **Kassow Robots** and the FlexiVision® system.

With this integration, a fast and secure interface with the FlexiVision® vision system can be implemented simply. The Plug-In provides a **stable and immediate connection** and the ability to send all the commands required for the correct operation of an application with a FlexiVision® vision system.

FlexiVision® Plug-In

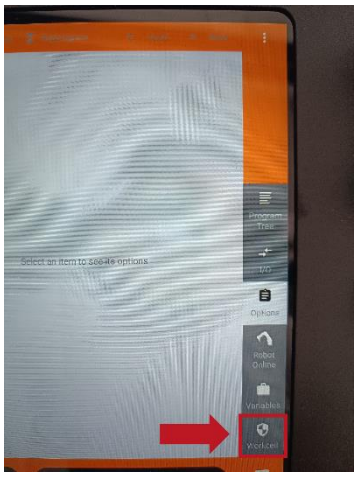
kassow robots

strong · fast · simple

Installing the Plug-In

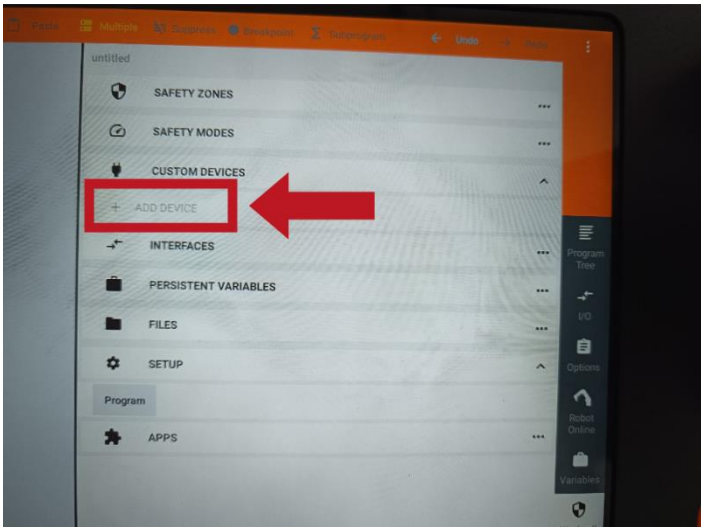
Before starting, open the side door of the robot controller and insert the USB stick with the Plug-In into the USB port on the right side of the controller.

Step 1.



Select the "**Workcell**" window from one of the two side menus of the Pendant.

Step 2.



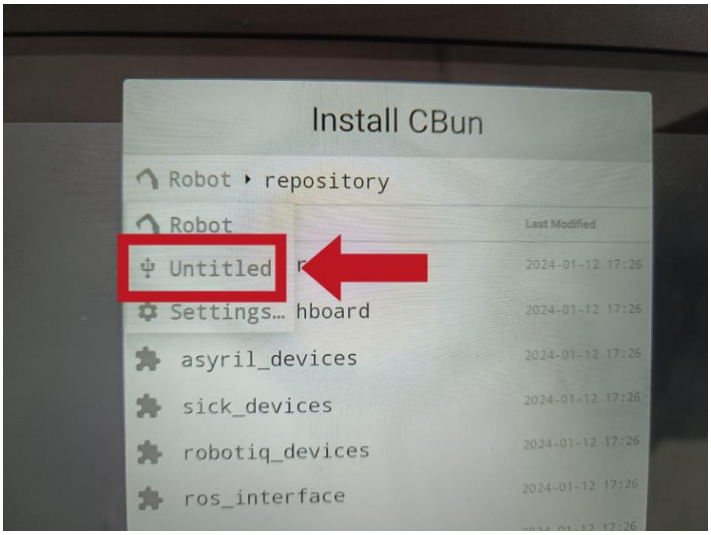
Go to the '**+ ADD DEVICE**' section .

Installing the Plug-In

Step 3.

Press **“+“**.

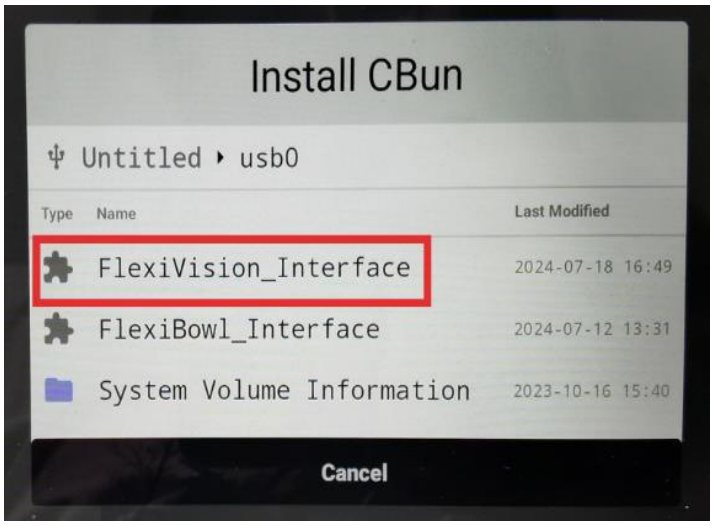
Step 4.



Go to the **'Untitled'** section .

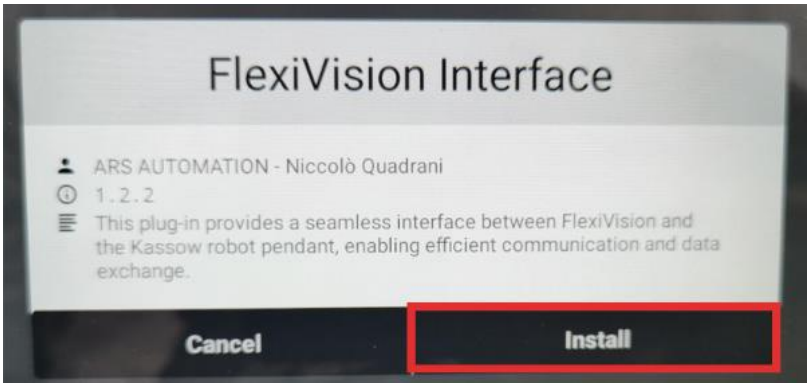
Installing the Plug-In

Step 5.



Select the '**FlexiVision_Interface**' file.

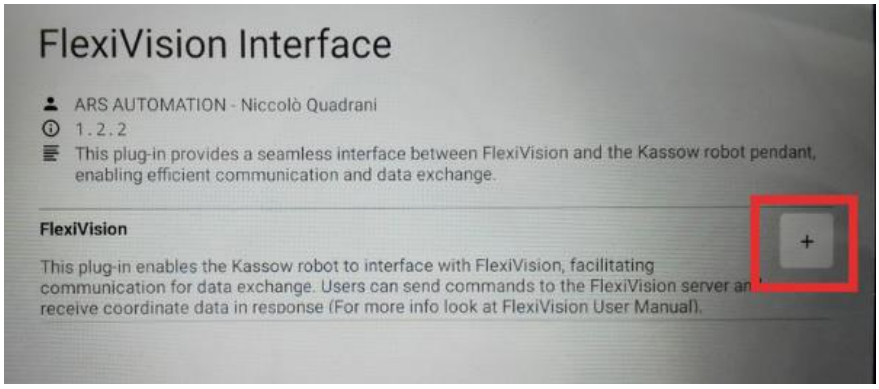
Step 6.



Select '**Install**' to successfully install the Plug-In within the controller.

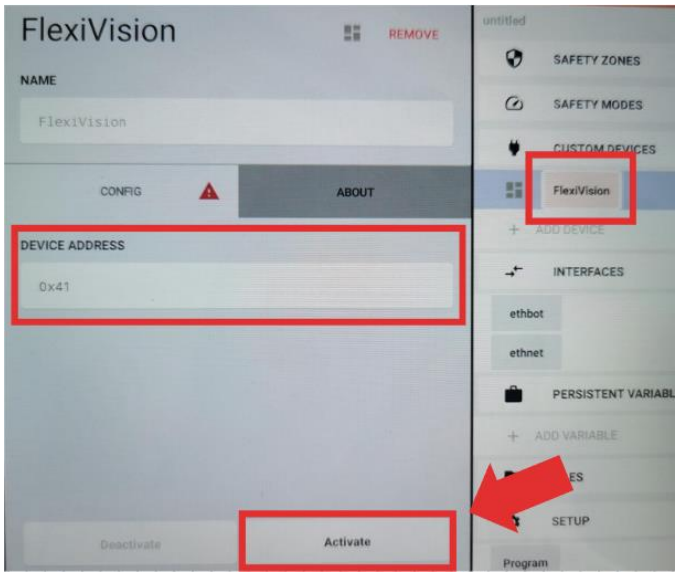
Installing the Plug-In

Step 7.



Press the **'+'** button to add the plug-in to the **'Program Tree'.**

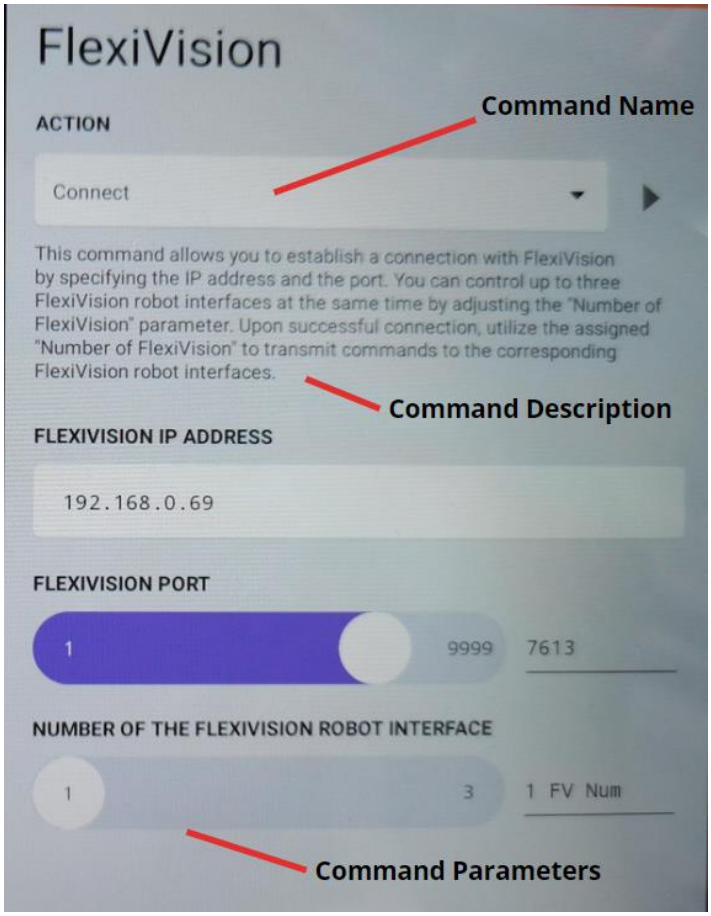
Step 8.



Press on the red **'FlexiVision'** rectangle under **'CUSTOM DEVICES',** then enter a name in the **'DEVICE NAME'** field or leave the Default name and press the **'Activate'** button.
If the operation is successful, the **'FlexiVision'** rectangle will turn light green and you will be able to see the **'FlexiVision'** block in the **'Program Tree'** section (bottom green section).
Here you will be able, by dragging the block, to use the FlexiVision® functions within your program.

Using the Plug-in

GUI explanation and use of functions.



The **'Command** Name ' section indicates the name of the command to be executed with the selected block. Clicking on the command name will open a dialogue box for choosing the desired command.

Under the section of the command name, you will find a brief **description** of the function offered by the command.

At the bottom of the screen, there is the section dedicated to the **parameters required** for the command to be executed correctly.

These parameters vary depending on the chosen command and must be configured according to the specific needs of the user.

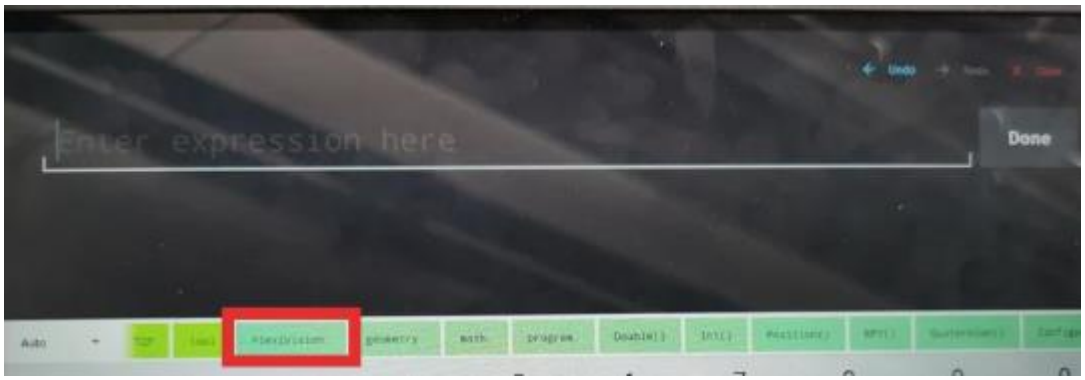
One parameter common to all commands is **'Number of FlexiVision Robot Interface',** which identifies the number of the FlexiVision® interface that the command should be sent to. This parameter is essential, as the Plug-in supports a maximum of three FlexiVision® interfaces per robot.

For a thorough understanding of how the various commands work and the appropriate parameters to send, refer to the next section of the documentation.

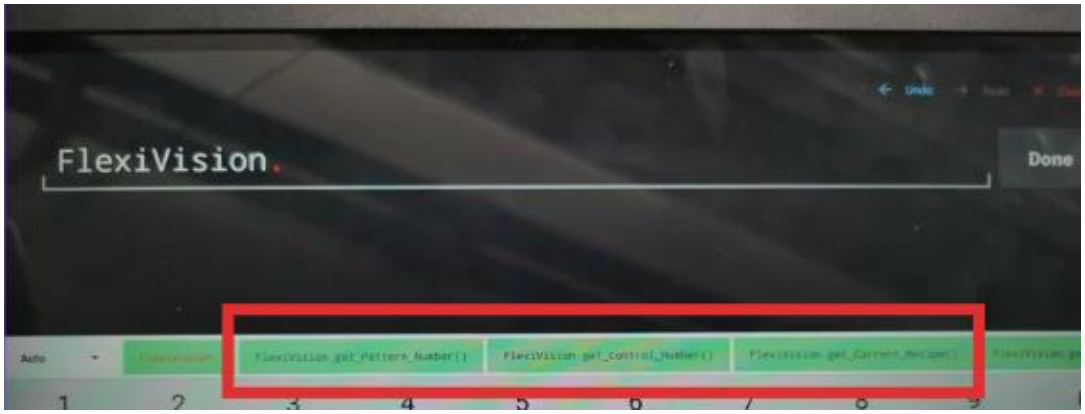
Using the Plug-in

GUI explanation and use of methods.

To use the plug-in's methods, it is necessary to use the **'SET'** command in the program tree, in the first section enter a variable consistent with the output of the method, and in the second section follow the steps below:



1. Select the **'FlexiVision'** block.



2. Select the method you want to apply (Example. **"FlexiVision.get_X()"**)

3. Enter the parameters required by the method using the pendant keyboard.

At the end of the method execution, the output value will be saved in the variable inserted in the first section of the **'SET'** command

FlexiBowl® by ars

Plug-In Functions

1. Communication Functions

1.1 FlexiVision – Connect

The Connect function allows a stable connection to be established with the specified FlexiVision® interface by making it possible to send commands, this command is compulsorily the first to be used in the application before the other commands.

Function

```
CBUN_PCALL  MyDevice::connection(string IP, int Port, int n_FlexiVision)
```

Parameters

`string IP` - IP of the PC that the FlexiVision software is running on.

`intPort` - FlexiVision port opened in the 'Robot' section of the application.

`int n_FlexiVision` - ID of the FlexiVision interface connected to, to be reused in the commands.

Return Value

`CBUN_PCALL_RET_OK` - Connection successfully established.

`CBUN_PCALL_RET_ERROR` - A connection could not be established.

1.2 FlexVision – Test Connection

The Test Connection function allows you to check the connection between the robot and a FlexiVision® interface at any time during the program.

Function

```
CBUN_PCALL  MyDevice::test_Connection(int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision whose connection is to be checked.

Return Value

`CBUN_PCALL_RET_OK` - Stable connection and communication possible.

`CBUN_PCALL_RET_ERROR` - Communication with the FlexiBowl was not possible.

1.3 FlexiVision – Send custom command

The Send custom command function allows a custom string to be sent to the selected FlexiVision® interface.

Function

```
CBUN_PCALL  MyDevice::send_Custom_Command(string custom, int n_FlexiVision)
```

Parameters

`custom` string - String to be sent to the FlexiBowl.

`int n_FlexiVision` - ID of FlexiVision that the command is to be sent to.

Return Value

`CBUN_PCALL_RET_OK` - *FlexiVision correctly received the command.*

`CBUN_PCALL_RET_ERROR` - *FlexiVision did not receive the command correctly.*

Plug-In Functions

2. Recipe Management Functions

2.1 FlexiVision – Get Recipe

The Get Recipe function allows the user to read the recipe currently loaded on FlexiVision® and save it in the variable 'currentRecipeName', **the name of the recipe read must be numeric.**

If the name of the recipe is not numeric, the function will only consider numeric characters, in the absence of which it will display -1.

To read the result use the 'get_Current_Recipe()' method

(Example. recipeName="Exmpl123" -> currentRecipeName = "123")

Function

```
CBUN_PCALL  MyDevice::get_Recipe(int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of the FlexiVision interface that you want to know the recipe for.

Return Value

`CBUN_PCALL_RET_OK` - *Recipe saved correctly.*

`CBUN_PCALL_RET_ERROR` - Communication error, unable to save the recipe.

2.2 FlexiVision – Set Recipe

The Set Recipe function allows the user to change the recipe in the program, this command accepts strings as recipe names.

Function

```
CBUN_PCALL  MyDevice::set_Recipe(string recipe_Name,int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of the FlexiVision interface that you want to upload the recipe to.

`String recipe_Name` - Name of the recipe to be loaded.

Return Value

`CBUN_PCALL_RET_OK` - *Recipe successfully uploaded.*

`CBUN_PCALL_RET_ERROR` - Communication error, unable to edit the recipe.

Plug-In Functions

3. Vision and Movement Functions

3.1 FlexiVision – Start Locator

The Start Locator function makes the FlexiBowl® move, using the saved sequence, until the vision finds an object that can be picked up by the robot.

Function

```
CBUN_PCALL  MyDevice::start_Locator(int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`CBUN_PCALL_RET_OK` - An object was located and the coordinates were saved.

`CBUN_PCALL_RET_ERROR` - Communication error, unable to send the command or FlexiVision error.

3.2 FlexiVision – Turn Locator

The Turn Locator function moves the FlexiBowl® using the saved sequence and then takes the picture and checks if there is an object that can be picked up by the robot.

Function

```
CBUN_PCALL  MyDevice::turn_Locator(int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`CBUN_PCALL_RET_OK` - Turn Locator sent successfully.

`CBUN_PCALL_RET_ERROR` - Communication error, unable to send the command.

3.3 FlexiVision – Stop Locator

The Stop Locator function instantly terminates the FlexiVision® object search process.

Function

```
CBUN_PCALL  MyDevice::stop_Locator(int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`CBUN_PCALL_RET_OK` - Stop Locator sent successfully.

`CBUN_PCALL_RET_ERROR` - Communication error, unable to send the command.

Plug-In Functions

3.4 FlexiVision – Start Empty

The Start Empty function starts the connected FlexiBowl® with the Quick Emptying procedure, which uses quick movements to remove objects from the FlexiBowl®.

Function

```
CBUN_PCALL  MyDevice::start_Empty(int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`CBUN_PCALL_RET_OK` - The procedure was successfully completed.

`CBUN_PCALL_RET_ERROR` - Communication error, unable to send the command or FlexiVision error.

3.5 FlexiVision – Start Control

The Start Control function starts the control room activation procedure, the response is customisable but to be interpreted by the plug-in it must be of the type: ControlN;x;y;rz.
(Example. "Control1;3;4;5")

Function

```
CBUN_PCALL  MyDevice::start_Control(int n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`CBUN_PCALL_RET_OK` - Reply received correctly.

`CBUN_PCALL_RET_ERROR` - Communication error, unable to send the command.

This plug-in does not allow the 'State_Locator' function of FlexiVision® to be used because the robot cannot work with string values and because the 'State_Locator' function is the only function of FlexiVision® that can only return String values.

Plug-In Functions

4. Value Acquisition Functions

4.1 FlexiVision.get_X

The FlexiVision.get_X function is a function to acquire the value of the X coordinate received with the FlexiVision® response to commands such as "Start Control', 'Start Locator' and 'Turn Locator'.

Function

```
kr2_program_api::Number MyDevice::get_X(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`double X` - *Value of the X co-ordinate of the object found by the vision.*

4.2 FlexiVision.get_Y

The FlexiVision.get_Y function is a function to acquire the value of the Y coordinate received with the FlexiVision® response to commands such as "Start Control', 'Start Locator' and 'Turn Locator'.

Function

```
kr2_program_api::Number MyDevice::get_Y(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`double Y` - *Value of the Y co-ordinate of the object found by the vision.*

4.3 FlexiVision.get_RZ

FlexiVision.get_RZ is a function to acquire the RZ co-ordinate value received with FlexiVision®'s response to commands such as 'Start Control', 'Start Locator' and 'Turn Locator'.

Function

```
kr2_program_api::Number MyDevice::get_RZ(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`double RZ` - *Value of the RZ co-ordinate of the object found by the vision.*

Plug-In Functions

4.4 FlexiVision.get_Move_Flag

FlexiVision.get_Move_Flag is a function to acquire the value of the flag for the movement, if this function displays 1 then the robot can proceed with picking the piece, then the flag is set to 0.

Function

```
kr2_program_api::Number MyDevice::get_Move_Flag(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`int` *Move_Flag*- Value of the move flag.

4.5 FlexiVision.get_Hopper_Flag

FlexiVision.get_Hopper_Flag is a function to acquire the value of the hopper activation flag, if this function displays 1 then the hopper IO output must be set to ON, after returning the flag value, the flag will be reset to 0.

Function

```
kr2_program_api::Number MyDevice::get_Hopper_Flag(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`int Hopper_Flag` - *Value of the Hopper flag.*

4.6 FlexiVision.get_Hopper_Sign

FlexiVision.get_Hopper_Sign is a function to acquire the hopper IO signal value that has been sent from FlexiVision®.

Function

```
kr2_program_api::Number MyDevice::get_Hopper_Sign(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`int HopperSign` - IO signal sent by FlexiVision.

4.7 FlexiVision.get_Hopper_Time

The FlexiVision.get_Hopper_Time function acquires the time, expressed in milliseconds, for which the hopper is to remain active, as specified by the FlexiVision® system.

Function

```
kr2_program_api::Number MyDevice::get_Hopper_Time(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`Double HopperTime` - Time (ms) sent by FlexiVision.

Plug-In Functions

4.8 FlexiVision.get_Pattern_Number

FlexiVision.get_Pattern_Number is a function to acquire the identification number of the Pattern found by FlexiVision®.

(Example. Pattern**1**;x;y;r – Pattern number = 1)

Function

```
kr2_program_api::Number MyDevice::get_Pattern_Number(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`int PatternNumber` - Pattern ID detected by the vision system.

4.9 FlexiVision.get_Control_Number

FlexiVision.get_Control_Number is a function to acquire the identification number detected by the 'Start Control' function.

(Example. Control**1**;x;y;r – Control number = 1)

Function

```
kr2_program_api::Number MyDevice::get_Control_Number(const kr2_program_api::Number& n_FlexiVision)
```

Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`int ControlNumber` - Control ID detected by the vision system.

4.10 FlexiVision.get_Current_Recipe

FlexiVision.get_Current_Recipe is a function to acquire the name of the currently loaded recipe within the selected FlexiVision interface, the name will be returned in numeric form (int type variable) so in case you want to use this function correctly, it is mandatory to have a numeric recipe name.

Function

```
kr2_program_api::Number MyDevice::get_Current_Recipe(const kr2_program_api::Number& n_FlexiVision)
```

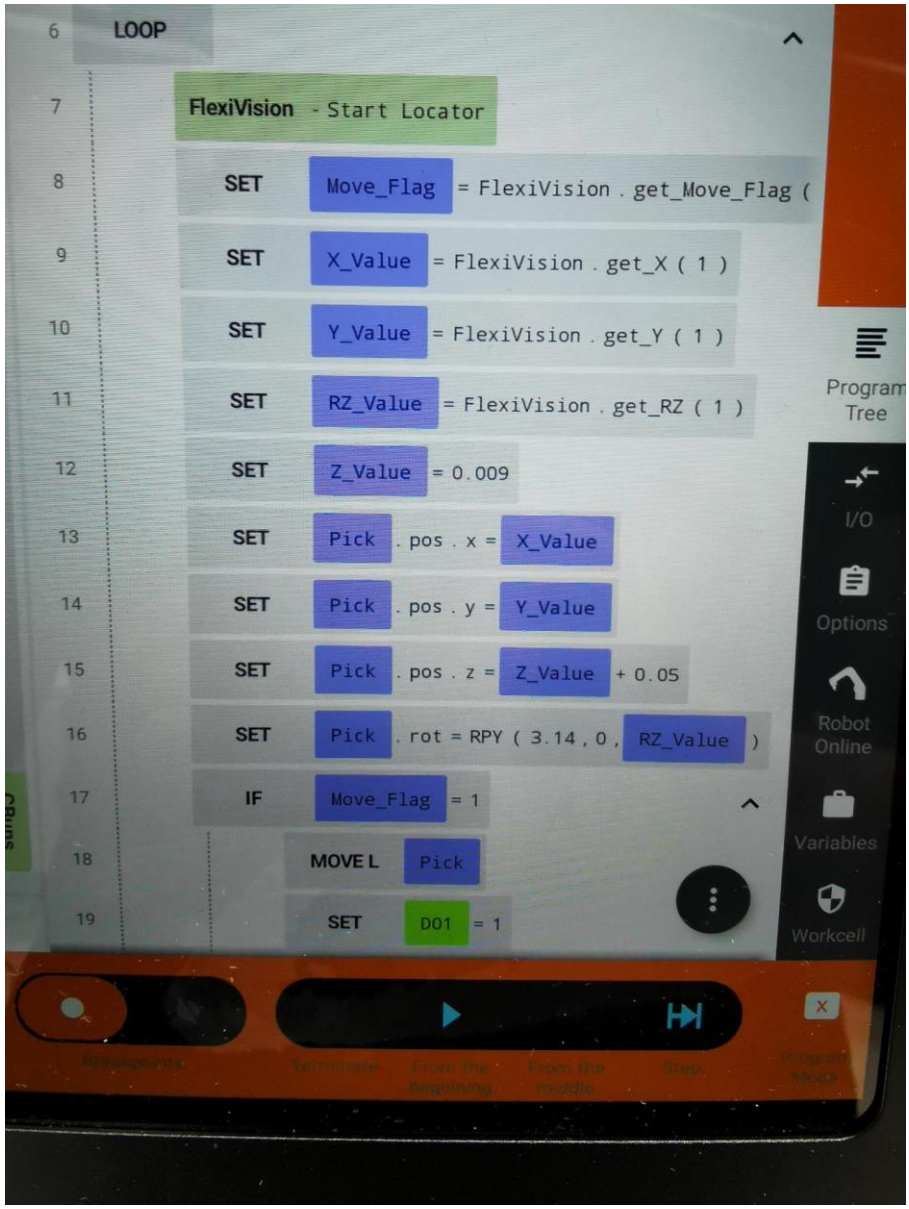
Parameters

`int n_FlexiVision` - ID of FlexiVision interface that the command is to be sent to.

Return Value

`int CurrentRecipeName`- *Name of the recipe uploaded to FlexiVision, the name shall be compulsorily numeric if you want to read.*

Implementation Example



To use the plug-in correctly, after establishing the connection correctly, use a code like the one above.

As you can see, first the FlexiVision block is used to send the **'Start Locator'** command**,** then enter the 'SET' commands, acquire and save the necessary data (x,y,rz and MoveFlag) in variables using the plug-in's methods while entering the Z according to the height of the object to be taken and according to the frame.

With another series of **'SETs'** enter the values of the variables within the pick point (To change the co-ordinates of a point use 'NamePoint.pos.coord()' where coord can be x,y or z and use 'NamePoint.rot.RPY()' to change the rotation, as in the figure)

FlexiVision Command List

To send the command to FlexiVision®, the value of the string "command" must be changed.

N_Mission	Command	Action
1	“start_Locator"	Starts the parts localisation process by recalling the FlexiBowl® handling routine in case there are no parts that can be picked up. ***Return:*** “Pattern1;x;y;r”.
2	“stop_Locator"	Stops the process of locating the object with the aid of FlexiBowl®.
3	“turn_Locator"	If no parts are picked up, by this command the operator can make the Flexibowl® rotate and the “start_Locator” routine start. ***Return:*** “Pattern1;x;y;r”.
4	“test_Locator"	Starts the process of locating the object without the aid of FlexiBowl®. ***Return:*** “Pattern1;x;y;r”.
5	“start_Control"	Starts the inspection cycle. ***Return:*** “Control1;x;y;r”.
6	“state_Locator"	Locator status diagnostics is shown: ***Return:*** “Locator is Running” “Locator is in Error” “Locator is not Running”.
7	“start_Empty"	Start the FlexiBowl® Quick-Emptying sequence. ***Return:*** “start_Empty ended”
8	“get_Recipe"	The name of the recipe currently loaded on FlexiVision® is shown. ***Return:*** “recipe name”.
9	“set_Recipe=recipe name"	The recipe corresponding to the sent "recipe name" is loaded.