



# Implementation of an application programming interface and graphical user interface for the 3-Axis Delta Robot

## Documentation

Presented by  
Yaman Alsaady

Emden, February 6, 2024

Supervised by  
M. Eng. Jeffrey Wermann

## Contents:

|                            |                            |           |
|----------------------------|----------------------------|-----------|
| <b>1</b>                   | <b>Main package</b>        | <b>1</b>  |
| 1.1                        | Run GUI . . . . .          | 1         |
| 1.2                        | Run Examples . . . . .     | 1         |
| 1.3                        | Gripper Golbal . . . . .   | 1         |
| <b>2</b>                   | <b>Modbus package</b>      | <b>3</b>  |
| 2.1                        | igus modbus . . . . .      | 3         |
| <b>3</b>                   | <b>Gripper package</b>     | <b>17</b> |
| 3.1                        | Gripper . . . . .          | 17        |
| <b>4</b>                   | <b>GUI package</b>         | <b>19</b> |
| 4.1                        | GUI . . . . .              | 19        |
| <b>5</b>                   | <b>Example package</b>     | <b>21</b> |
| 5.1                        | Move . . . . .             | 21        |
| 5.2                        | Control programs . . . . . | 21        |
| 5.3                        | Gripper . . . . .          | 22        |
| <b>Python Module Index</b> |                            | <b>23</b> |
| <b>Index</b>               |                            | <b>25</b> |



## Main package

### 1.1 Run GUI

**Author:**

Yaman Alsaady

**Description:**

Main script to launch the GUI.

This script imports the main function from the GUI module and calls it to launch the graphical user interface (GUI).

### 1.2 Run Examples

**Description:**

Main function to select and run different example modules.

This script prompts the user to choose from three example modules: Move, Gripper, and Control\_programs. After the user selects a module by entering a number, the corresponding main function of that module is executed. If the user enters an invalid number, a message indicating that the command is not recognized is printed.

### 1.3 Gripper Golbal

**Description:**

Main script for continuous control of the gripper using Modbus.

This script initializes a Gripper object and continuously calls its modbus() method as long as the gripper is connected. If the gripper gets disconnected, a new Gripper object is created to reestablish the connection.



## Modbus package

### 2.1 igus modbus

#### 2.1.1 igus\_modbus Module

**Author:**

Yaman Alsaady

**Description:**

This module provides a Python interface for controlling a Delta Robot (from igus) using Modbus TCP communication.

**Classes:**

- Robot: Represents the Robot and provides methods for controlling it.

**Usage:**

To use this module, create an instance of the 'Robot' class with the IP address and the port of the Robot as a parameter.

**Example**

```
from igus_modbus import Robot
# Create a Delta Robot instance with the IP address '192.168.1.11'
delta_robot = Robot('192.168.3.11')

# Perform actions with the Delta Robot
delta_robot.enable()
delta_robot.reference()

delta_robot.set_position_endeffector(0, 0, 250)
delta_robot.set_velocity(120)
delta_robot.move_endeffector_absolute()

class src.igus_modbus.Robot(address: str, port: int = 502)
    Bases: object
```

**break\_time = 5****control\_gripper(opening: int, orientation: int, signal: int = 6)**

Control the gripper using specified values and a Modbus signal.

**Parameters**

- **opening (int)** - The value for the gripper opening.
- **orientation (int)** - The value for the gripper orientation.
- **signal (int)** - The Modbus signal number to enable/disable gripper control. Default is 6.

**Returns**

True if the gripper control was successful, False otherwise.

**Return type**

bool

**control\_programs(action: str)**

Control robot programs.

This method allows you to control robot programs by sending specific commands. Supported actions are: 'start', 'continue', 'pause', and 'stop'.

**Parameters**

- **action (str)** - The action to perform ('start', 'continue', 'pause', or 'stop').

**disable()**

Disable the motors of the Robot.

This method disables the motors of the robot by writing 0 to the coil 53.

**Returns**

None

**enable()**

Enable the motors of the Robot.

This method enables the motors of the robot by writing 1 to the coil 53.

**Returns**

None

**get\_digital\_input(number: int)**

Get the state of a digital input.

This method allows you to get the state of a digital input by specifying its number.

**Parameters**

- **number (int)** - The number of the digital input (1 to 64).

**Returns**

The state of the digital input (True for ON, False for OFF).

**Return type**

bool

**get\_digital\_output(number)**

Get the state of a digital output.

This method allows you to get the state of a digital output by specifying its number.

**Parameters**

**number** (*int*) – The number of the digital output (1 to 64).

**Returns**

The state of the digital output (True for ON, False for OFF).

**Return type**

bool

**get\_globale\_signal(*number*: int)**

Get the state of a global signal.

This method allows you to get the state of a global signal by specifying its number.

**Parameters**

**number** (*int*) – The number of the global signal (1 to 100).

**Returns**

The state of the global signal (True for ON, False for OFF).

**Return type**

bool

**get\_info\_message()**

Get the information or error message from the Delta Robot.

This method reads the information or error message from the Delta Robot's control unit. The message is typically a short text, similar to what is displayed on a manual control unit.

**Returns**

The information or error message as a string.

**Return type**

str

**get\_kinematics\_error()**

Get the kinematics error description.

This method reads the kinematics error code from the robot controller and returns a human-readable description of the error.

**Returns**

A string describing the kinematics error.

**Return type**

str

**get\_list\_of\_programs()**

Get a list of available robot programs.

This method retrieves a list of robot programs from the robot controller. It communicates with the robot controller to gather program names.

**Returns**

A list of program names.

**Return type**

list

**get\_number\_of\_current\_program()**

Get the number of currently active programs on the Delta Robot.

This method reads the number of currently active programs on the Delta Robot and returns the count. Note: The main program is typically represented as program number 0.

**Returns**

The number of currently active programs.

**Return type**

int

**get\_number\_of\_loaded\_programs()**

Get the number of loaded programs on the Delta Robot.

This method reads the number of loaded programs on the Delta Robot and returns the count.

**Returns**

The number of loaded programs.

**Return type**

int

**get\_operation\_mode()**

Get the operation mode description.

This method reads the operation mode code from the robot controller and returns a human-readable description of the mode.

**Returns**

A string describing the operation mode.

**Return type**

str

**get\_orientation\_endeffector()**

Get the orientation of the Delta Robot's end effector.

This method reads the orientation values from input registers and returns them.

**Returns**

A list (a, b, c) representing the orientation values.

**Return type**

list[float]

**get\_position\_axes()**

Get the positions of the Delta Robot's axes.

This method reads the positions of the robot's axes (A1, A2, and A3) from input registers and returns them as a tuple.

**Returns**

A list (a1\_pos, a2\_pos, a3\_pos) representing the positions of the robot's axes.

**Return type**

list[float]

**get\_position\_endeffector()**

Get the Cartesian position of the Delta Robot's end effector. This method reads the X, Y, and Z positions of the end effector from input registers and returns them in millimeters as a tuple.

**Returns**

A list (x\_pos, y\_pos, z\_pos) representing the Cartesian position of the end effector in millimeters.

**Return type**

list[float]

**get\_program\_name()**

Get the name of the robot program.

This method reads the name of the robot program.

**Returns**

The name of the robot program.

**Return type**

str

**get\_program\_replay\_mode()**

Get the current replay mode of the robot program.

This method reads the replay mode of the robot program and returns a descriptive string. The possible replay modes are: - "Run program once": The robot program will run once and stop. - "Repeat program": The robot program will continuously repeat. - "Execute instructions step by step": The robot program will execute instructions one at a time. - "Fast" (Not used): A mode that is not currently used.

**Returns**

A descriptive string representing the current replay mode.

**Return type**

str

**get\_program\_runstate()**

Get the current run state of the robot program.

This method reads the run state of the robot program and returns a descriptive string. The possible run states are: - "Program is not running": The robot program is not currently executing. - "Program is running": The robot program is actively running. - "Program is paused": The robot program is paused but can be resumed.

**Returns**

A descriptive string representing the current run state.

**Return type**

str

**get\_readable\_number\_variable(number: int)**

Get the value of a readable Modbus variable.

This method allows you to retrieve the value of a Modbus variable for reading. Please ensure that the variable name in your program follows the naming convention: mb\_num\_r1 - mb\_num\_r16.

**Parameters**

**number (int)** - The number of the Modbus variable (1 to 16).

**Returns**

The value of the Modbus variable, or False if the number is out of range.

**Return type**

int or bool

**get\_readable\_position\_variable(*number: int*)**

Get the value of a readable position Modbus variable.

This method allows you to retrieve the value of a readable position Modbus variable. Ensure that the variable name in your program follows the naming convention, e.g., mb\_pos\_r1.

**Parameters**

**number** (*int*) – The number of the Modbus variable (1 to 16).

**Returns**

A list containing axis, cartesian, orientation values, and conversion type, or False if the number is out of range.

**Return type**

list

**get\_robot\_errors()**

Get a list of error descriptions indicating the robot's current error states.

This method reads the status of various error-related coils on the robot controller and returns a list of error descriptions if any errors are detected.

**Returns**

A list of error descriptions or "No error" if there are no errors.

**Return type**

list[str]

**get\_stop\_reason\_description()**

Get a description of the reason for the robot's current stop condition.

This method reads the stop reason code from the robot controller and returns a human-readable description of the reason for the stop.

**Returns**

A string describing the reason for the stop.

**Return type**

str

**get\_writable\_number\_variable(*number: int*)**

Get the value of a writable Modbus variable.

This method allows you to retrieve the value of a Modbus variable for writing. Ensure that the variable name in your program adheres to the naming convention: mb\_num\_w1 - mb\_num\_w16.

**Parameters**

**number** (*int*) – The number of the Modbus variable (1 to 16).

**Returns**

The value of the Modbus variable, or False if the number is out of range.

**Return type**

int or bool

**get\_writable\_position\_variable(number: int)**

Get the value of a writable position Modbus variable.

This method allows you to retrieve the value of a writable position Modbus variable. Ensure that the variable name in your program follows the naming convention, e.g., mb\_pos\_w1.

**Parameters**

**number (int)** – The number of the Modbus variable (1 to 16).

**Returns**

A list containing axis, cartesian, orientation values, and conversion type, or False if the number is out of range.

**Return type**

list

**is\_connected = False****is\_enabled()**

Check if the Robot is enabled.

This method checks the state of the Robot's motors by reading coil 53.

**Returns**

True if the motors are enabled, False otherwise.

**Return type**

bool

**is\_general\_error()**

Check if the robot has general errors.

This method checks the state of the robot's error status coil and returns True if the robot has general errors, or False if there are no errors.

**Returns**

True if the robot has general errors, False otherwise.

**Return type**

bool

**is\_kinematics\_error()**

Check if the robot has kinematics-related errors.

This method checks the state of the robot's kinematics error status coil and returns True if the robot has kinematics-related errors, or False if there are no kinematics errors.

**Returns**

True if the robot has kinematics-related errors, False otherwise.

**Return type**

bool

**is\_moving()**

Check if the Robot is moving.

This method checks the state of the Robot's motion by reading coil 112.

**Returns**

True if the Robot is currently moving, False otherwise.

**Return type**

bool

**is\_program\_loaded()**

Check if a program is loaded.

This method checks if a program is loaded on the robot controller.

**Returns**

True if a program is loaded, False otherwise.

**Return type**

bool

**is\_referenced()**

Check if the Robot is referenced.

This method checks if the Robot has been referenced by reading coil 60.

**Returns**

True if the Robot is referenced, False otherwise.

**Return type**

bool

**is\_zero\_torque()**

Check if the robot is in a zero torque state.

This method reads a Modbus coil to determine if the robot is currently in a state where it applies zero torque. It returns True if zero torque is detected, False otherwise.

**Returns**

True if the robot is in a zero torque state, False otherwise.

**Return type**

bool

**move\_axes(*wait: bool = True, relative: str = False*)**

Move the end effector to the target position.

This method moves the end effector to the specified axes position by controlling the appropriate coil. The movement can be relative or absolute 'relative' parameter. To specify the position, use the method `set_position_axes`.

**Parameters**

- **wait (bool)** – If True (default), wait until the movement is complete before returning.
- **relative (bool)** – If False (default), the movement will be absolute, otherwise will be relative to the current position

**move\_circular(*radius: float, step: float = 0.5, start\_angle: int = 0, stop\_angle: int = 360*)**

Move the robot's end effector in a circular path.

This method moves the robot's end effector in a circular path in the X-Y plane. The circular path is defined by a radius, and you can specify the step size, start angle, and stop angle.

**Parameters**

- **radius** (*float*) – The radius of the circular path in millimeters.
- **step** (*float*) – The step size in degrees for moving along the circular path (default is 0.5 degrees).
- **start\_angle** (*float*) – The starting angle of the circular path in degrees (default is 0 degrees).
- **stop\_angle** (*float*) – The stopping angle of the circular path in degrees (default is 360 degrees).

### **move\_endeffector**(*wait: bool = True, relative: str = None*)

Move the end effector to the target position.

This method moves the end effector to the specified Cartesian position by controlling the appropriate coil. The movement can be relative to different reference frames (base, tool) based on the ‘relative’ parameter. To specify the position, use the method set\_position\_endeffector(x, y, z).

#### **Parameters**

- **wait** (*bool*) – If True (default), wait until the movement is complete before returning.
- **relative** (*str or None*) – Specifies the reference frame for the movement (None for absolute, ‘base’, or ‘tool’).

### **print\_list\_of\_programs()**

Print a list of available robot programs.

This method retrieves a list of robot programs and prints them to the console. If the robot is not connected, it will return without performing any action.

#### **Returns**

None

### **read\_string**(*read*)

Read a string from a sequence of registers.

This method reads a string from a sequence of registers and returns the decoded string.

#### **Parameters**

- read** (*list*) – The sequence of registers containing the string data.

#### **Returns**

The decoded string.

#### **Return type**

str

### **reference**(*force: bool = False*)

Reference the Robot.

This method references the robot by writing a rising edge to the coil 60. If ‘only\_once’ is set to True (default), it will only reference the robot if it’s not already referenced.

#### **Parameters**

- force** (*bool*) – If False, the robot will only be referenced if not already referenced, otherwise, it will be referenced each time this method is called (default is False).

**Returns**

None

**reset()**

Reset the Delta Robot.

This method resets the robot by writing a rising edge to the coil.

**Returns**

None

**set\_and\_move(val\_1: float, val\_2: float, val\_3: float, movement: int = 'cartesian', relative: str = None, wait: bool = True, velocity: float = None)**

Set the target position and move the end effector.

This method sets the target position of the end effector using the 'set\_position\_endeffector' method, adjusts the velocity if specified, and then moves the end effector to the target position using the 'move\_endeffector' method. The movement can be relative to different reference frames (base, tool) based on the 'relative' parameter. You can choose to wait until the movement is complete before returning.

**Parameters**

- **val\_1 (float)** - The target position value (X, A1, or other axis, depending on the 'movement' parameter).
- **val\_2 (float)** - The target position value (Y, A2, or other axis, depending on the 'movement' parameter).
- **val\_3 (float)** - The target position value (Z, A3, or other axis, depending on the 'movement' parameter).
- **movement (str)** - Specifies the type of movement ('cartesian' or 'axes').
- **relative (str or None)** - Specifies the reference frame for the movement (None for absolute, 'base', or 'tool').
- **wait (bool)** - If True (default), wait until the movement is complete before returning.
- **velocity (float or None)** - Optional velocity setting in millimeters per second.

**set\_digital\_output(number: int, state: bool)**

Set the state of a digital output.

This method allows you to set the state of a digital output by specifying its number and state.

**Parameters**

- **number (int)** - The number of the digital output (1 to 64).
- **state (bool)** - The state to set (True for ON, False for OFF).

**set\_globale\_signal(number: int, state: bool)**

Set the state of a global signal.

This method allows you to set the state of a global signal by specifying its number and state.

**Parameters**

- **number** (*int*) - The number of the global signal (1 to 100).
- **state** (*bool*) - The state to set (True for ON, False for OFF).

### **set\_number\_variables**(*number: int = 1, value: int = 0*)

Set the value of a writable Modbus variable.

This method allows you to set the value of a Modbus variable for program use. Please note that the variable name in your program should follow the naming convention: mb\_num\_w1 - mb\_num\_w16.

#### **Parameters**

- **number** (*int*) - The number of the Modbus variable (1 to 16).
- **value** (*int*) - The value to set for the Modbus variable.

#### **Returns**

True if the operation was successful, False if the number is out of range.

#### **Return type**

*bool*

### **set\_orientation\_endeffector**(*a\_val: float, b\_val: float, c\_val: float*)

Set the orientation of the end effector.

This method allows you to set the orientation of the robot's end effector by specifying the angles '*a\_val*', '*b\_val*', and '*c\_val*' for orientation around the X, Y, and Z axes, respectively.

#### **Parameters**

- **a\_val** (*float*) - The orientation angle around the X-axis in degrees.
- **b\_val** (*float*) - The orientation angle around the Y-axis in degrees.
- **c\_val** (*float*) - The orientation angle around the Z-axis in degrees.

#### **Returns**

None

### **set\_override\_velocity**(*velocity: float = 20*)

Set the override velocity for robot movements.

This method allows you to adjust the velocity override for robot movements. The *velocity* parameter specifies the desired velocity as a percentage (0-100), with 100 being the maximum velocity. The default is 20%.

#### **Parameters**

**velocity** (*float*) - The desired velocity override as a percentage (0-100).

#### **Returns**

None

### **set\_position\_axes**(*a1\_val: float, a2\_val: float, a3\_val: float*)

Set the target position of the endeffector

This method allows you to set the target positions of the robot's axes. The input values '*a1\_val*', '*a2\_val*', and '*a3\_val*' represent the target positions for each axis. The positions are converted to the appropriate format and written to the respective registers.

The position can be absolute or relative. To make the robot move, use the method `move_axes()`.

**Parameters**

- **a1\_val** (*float*) - The target position for axis A1.
- **a2\_val** (*float*) - The target position for axis A2.
- **a3\_val** (*float*) - The target position for axis A3.

**Returns**

None

**set\_position\_endeffector**(*x\_val: float, y\_val: float, z\_val: float*)

Set the target position of the end effector in millimeters.

This method sets the target position of the end effector in millimeters. The position can be absolute or relative to the base or to itself. To make the robot move to the specified position, use the ‘move\_endeffector’ method.

**Parameters**

- **x\_val** (*float*) - The target X position in millimeters.
- **y\_val** (*float*) - The target Y position in millimeters.
- **z\_val** (*float*) - The target Z position in millimeters.

**Returns**

None

**set\_position\_variable**(*number=1, movement: str = 'cartesian', a1: int = None, a2: int = None, a3: int = None, x: int = None, y: int = None, z: int = None, a: int = 0, b: int = 0, c: int = 180, conversion: int = 0*)

Set the target position for robot movement in a robot program.

This method allows you to set the target position for robot movement in a program. You can specify the target position either in Cartesian or axis values. Ensure the variable name in your program follows the naming convention, e.g., mb\_pos\_w1.

**Parameters**

- **number** (*int*) - The number of the Modbus variable (1 to 16).
- **movement** (*str*) - The type of movement (either “cartesian” or “axes”).
- **a1** (*int*) - The value of axis A1 (if movement is “axes”).
- **a2** (*int*) - The value of axis A2 (if movement is “axes”).
- **a3** (*int*) - The value of axis A3 (if movement is “axes”).
- **x** (*int*) - The X-coordinate value (if movement is “cartesian”).
- **y** (*int*) - The Y-coordinate value (if movement is “cartesian”).
- **z** (*int*) - The Z-coordinate value (if movement is “cartesian”).
- **a** (*int*) - The orientation A value (if movement is “cartesian”).
- **b** (*int*) - The orientation B value (if movement is “cartesian”).
- **c** (*int*) - The orientation C value (if movement is “cartesian”).
- **conversion** (*int*) - The conversion type (useful for converting between joint and cartesian positions).

**Returns**

True if the operation was successful, False if the number is out of range or invalid parameters.

**Return type**

bool

**set\_program\_name(name)**

Set the name of the robot program.

This method allows you to set the name of the robot program.

**Parameters**

**name** (str) - The name of the robot program.

**set\_program\_replay\_mode(mode: str = 'once')**

Set the program replay mode for the robot.

This method allows you to configure the program replay mode for the robot. The *mode* parameter specifies the desired mode and can take one of the following values: - "once" (Default): Play the program once. - "repeat": Repeat the program continuously. - "step": Step through the program one instruction at a time. - "fast": Not used (for future expansion).

**Parameters**

**mode** (str) - The desired program replay mode.

**Returns**

0 if

**Return type**

int

**set\_velocity(velocity: bool)**

Set the velocity of the Robot.

This method sets the velocity of the robot in millimeters per second. For cartesian motions the value is set as a multiple of 1mm/s, for joint motions it is a multiple of 1% (relative to the maximum velocity) The actual motion speed also depends on the global override value (holding register 187).

**Parameters**

**velocity** (float) - The desired velocity in millimeters per second (or in percent).

**Returns**

None

**set\_zero\_torque(enable: bool = True)**

Set the zero torque state for manual movement.

This method allows you to enable or disable the zero torque state, which allows manual movement of the robot by hand.

**Parameters**

**enable** (bool) - True to enable zero torque (for manual movement), False to disable.

**Returns**

None

**shutdown()**

Reset the Delta Robot.

This method shut the robot down by writing a rising edge to the coil.

**Returns**

None

**write\_string(string, ad, number=32)**

Write a string to a sequence of registers.

This method allows you to write a string to a sequence of registers, starting from a specified address.

**Parameters**

- **string (str)** - The string to write.
- **ad (int)** - The starting address to write the string.
- **number (int)** - The maximum number of characters to write (default is 32).

## Gripper package

### 3.1 Gripper

#### 3.1.1 Gripper Module

**Author:**

Yaman Alsaady

**Description:**

This module provides a Python interface for controlling a gripper device through serial communication.

**Classes:**

- **Gripper:** Represents the gripper and provides methods for controlling its opening and orientation.

**Usage:**

To use this module, create an instance of the 'Gripper' class with the appropriate serial port and settings.

```
class src.gripper.Gripper(port: str = '/dev/ttyUSB0', baudrate: int = 115200, timeout: int = 1)
```

Bases: object

**close()** → bool

Close the gripper fully.

This method sets the gripper opening to 100 percent.

**Returns**

True if the operation was successful, False otherwise.

**Return type**

bool

```
control(opening: int, orientation: int = None) → bool
```

Control the gripper's opening and orientation.

This method allows you to control the gripper's opening (0 to 100 percent) and, optionally, its orientation in degrees.

**Parameters**

- **opening (int)** – The desired gripper opening in percent (0 to 100).
- **orientation (int or None)** – The desired gripper orientation in degrees (optional).

**Returns**

True if the operation was successful, False otherwise.

**Return type**

bool

**is\_connected: bool = False**

**modbus(signal: int = 6, var1: int = 15, var2: int = 16)**

Control the gripper using Modbus signals and variables.

**Parameters**

- **signal (int)** – The Modbus signal number to enable/disable gripper control. Default is 6.
- **var1 (int)** – The Modbus variable number for reading the gripper opening. Default is 15.
- **var2 (int)** – The Modbus variable number for reading the gripper orientation. Default is 16.

**Returns**

True if the gripper control was successful, False otherwise.

**Return type**

bool

**open() → bool**

Open the gripper fully.

This method sets the gripper opening to 0 percent.

**Returns**

True if the operation was successful, False otherwise.

**Return type**

bool

**opening = 0**

**orientation = 0**

**rotate(orientation: int) → bool**

Rotate the gripper to a specific orientation.

This method sets the gripper orientation to the specified degree value.

**Parameters**

- **orientation (int)** – The desired gripper orientation in degrees.

**Returns**

True if the operation was successful, False otherwise.

**Return type**

bool

# 4

## GUI package

### 4.1 GUI

#### 4.1.1 Delta Robot Control Application

**Author:**

Yaman Alsaady

**Description:**

- This application provides a graphical user interface (GUI) to control a Delta robot and its gripper.
- The GUI allows the user to control the robot's movements, adjust speed settings, operate the gripper, and manage robot programs.

**Classes:**

- App: Represents the main application window and contains methods for initializing the GUI and running the application.

```
class GUI.gui.App(_)
    Bases: Frame
    add()
    clear_list()
    connect()
    control_widgets()
    enable_robot()
    error_widgets()
    fontsize = 20
    gripper_mov()
    gripper_widgets(tab_name, row, column)
```

```
load_program()
load_widgets()
locale_load_program()
locale_programs_widgets()
locale_update_list()
logo_widgets(img='/home/yaman/src/IGUS_Delta_Robot/Modbus/GUI/img/hsel_logo_dark.png')
move_widgets(tab_name, row, column)
program_names(list)
programs_widgets()
remove_list_element()
run_list()
save_list()
setting_widgets()
show_positions(list)
sort_list()
speed_widgets()
split_list(list)
tabs()
teach_widgets()
update()
Enter event loop until all pending events have been processed by Tcl.
update_error()
update_list()
update_tabs(_)
update_theme()

GUI.gui.main()
```

# 5

## Example package

### 5.1 Move

**Author:**

Yaman Alsaady

**Description:**

This example demonstrates how to move the Delta robot in a rectangular pattern. It begins by establishing a connection to the robot, and if successful, performs some basic movements.

**Example.Move.main()**

This function creates an instance of the Robot class, establishes a connection to the robot, and executes a predefined sequence of movements.

### 5.2 Control programs

**Description:**

This example demonstrates controlling the gripper with the Delta robot. The following code provide an overview of the basic functions for controlling the gripper.

**Example.Control\_programs.main()**

This function creates an instance of the Robot class, establishes a connection to the robot, and interacts with its control programs.

## **5.3 Gripper**

### **Description:**

This example shows how to control programs on the Delta robot. It demonstrates the selection of a program, starting, stopping, pausing, and resuming execution.

### **Example.Gripper.main()**

This function creates an instance of the Robot class, establishes a connection to the robot, and executes a predefined sequence of movements combined with gripper control commands.

## Python Module Index

### e

example, 1  
Example.Control\_programs, 21  
Example.Gripper, 22  
Example.Move, 21

### g

gripper\_global, 1  
GUI.gui, 19

### m

main, 1

### s

src.gripper, 17  
src.igus\_modbus, 3



# Index

## A

`add()` (*GUI.gui.App method*), 19  
`App` (*class in GUI.gui*), 19

## B

`break_time` (*src.igus\_modbus.Robot attribute*), 3

## C

`clear_list()` (*GUI.gui.App method*), 19  
`close()` (*src.gripper.Gripper method*), 17  
`connect()` (*GUI.gui.App method*), 19  
`control_gripper()` (*src.igus\_modbus.Robot method*), 4  
`control_widgets()` (*GUI.gui.App method*), 19  
`controll()` (*src.gripper.Gripper method*), 17  
`controll_programs()` (*src.igus\_modbus.Robot method*), 4

## D

`disable()` (*src.igus\_modbus.Robot method*), 4

## E

`enable()` (*src.igus\_modbus.Robot method*), 4  
`enable_robot()` (*GUI.gui.App method*), 19  
`error_widgets()` (*GUI.gui.App method*), 19  
`example`  
    `module`, 1  
`Example.Control_programs`  
    `module`, 21  
`Example.Gripper`  
    `module`, 22  
`Example.Move`  
    `module`, 21

## F

`fontsize` (*GUI.gui.App attribute*), 19

## G

`get_digital_input()` (*src.igus\_modbus.Robot method*), 4  
`get_digital_output()` (*src.igus\_modbus.Robot method*), 4  
`get_globale_signal()` (*src.igus\_modbus.Robot method*), 5

`get_info_message()` (*src.igus\_modbus.Robot method*),  
    5  
`get_kinematics_error()` (*src.igus\_modbus.Robot method*), 5  
`get_list_of_pograms()` (*src.igus\_modbus.Robot method*), 5  
`get_number_of_current_program()`  
    (*src.igus\_modbus.Robot method*), 5  
`get_number_of_loaded_programs()`  
    (*src.igus\_modbus.Robot method*), 6  
`get_operation_mode()` (*src.igus\_modbus.Robot method*), 6  
`get_orientation_endeffector()`  
    (*src.igus\_modbus.Robot method*), 6  
`get_position_axes()` (*src.igus\_modbus.Robot method*), 6  
`get_position_endeffector()` (*src.igus\_modbus.Robot method*), 6  
`get_program_name()` (*src.igus\_modbus.Robot method*),  
    7  
`get_program_replay_mode()` (*src.igus\_modbus.Robot method*), 7  
`get_program_runstate()` (*src.igus\_modbus.Robot method*), 7  
`get_readable_number_variable()`  
    (*src.igus\_modbus.Robot method*), 7  
`get_readable_position_variable()`  
    (*src.igus\_modbus.Robot method*), 8  
`get_robot_errors()` (*src.igus\_modbus.Robot method*),  
    8  
`get_stop_reason_description()`  
    (*src.igus\_modbus.Robot method*), 8  
`get_writable_number_variable()`  
    (*src.igus\_modbus.Robot method*), 8  
`get_writable_position_variable()`  
    (*src.igus\_modbus.Robot method*), 8  
`Gripper` (*class in src.gripper*), 17  
`gripper_global`  
    `module`, 1  
`gripper_mov()` (*GUI.gui.App method*), 19  
`gripper_widgets()` (*GUI.gui.App method*), 19  
`GUI.gui`  
    `module`, 19

|

`is_connected` (*src.gripper.Gripper attribute*), 18

is\_connected (*src.igus\_modbus.Robot attribute*), 9  
is\_enabled() (*src.igus\_modbus.Robot method*), 9  
is\_general\_error() (*src.igus\_modbus.Robot method*),  
    9  
is\_kinematics\_error() (*src.igus\_modbus.Robot  
method*), 9  
is\_moving() (*src.igus\_modbus.Robot method*), 9  
is\_program\_loaded() (*src.igus\_modbus.Robot  
method*), 10  
is\_referenced() (*src.igus\_modbus.Robot method*), 10  
is\_zero\_torque() (*src.igus\_modbus.Robot method*), 10

## L

load\_program() (*GUI.gui.App method*), 19  
load\_widgets() (*GUI.gui.App method*), 20  
locale\_load\_program() (*GUI.gui.App method*), 20  
locale\_programs\_widgets() (*GUI.gui.App method*), 20  
locale\_update\_list() (*GUI.gui.App method*), 20  
logo\_widgets() (*GUI.gui.App method*), 20

## M

main  
    module, 1  
main() (*in module Example.Control\_programs*), 21  
main() (*in module Example.Gripper*), 22  
main() (*in module Example.Move*), 21  
main() (*in module GUI.gui*), 20  
modbus() (*src.gripper.Gripper method*), 18  
module  
    example, 1  
    Example.Control\_programs, 21  
    Example.Gripper, 22  
    Example.Move, 21  
    gripper\_global, 1  
    GUI.gui, 19  
    main, 1  
    src.gripper, 17  
    src.igus\_modbus, 3  
move\_axes() (*src.igus\_modbus.Robot method*), 10  
move\_circular() (*src.igus\_modbus.Robot method*), 10  
move\_endeffector() (*src.igus\_modbus.Robot method*),  
    11  
move\_widgets() (*GUI.gui.App method*), 20

## O

open() (*src.gripper.Gripper method*), 18  
opening (*src.gripper.Gripper attribute*), 18  
orientation (*src.gripper.Gripper attribute*), 18

## P

print\_list\_of\_programs() (*src.igus\_modbus.Robot  
method*), 11  
program\_names() (*GUI.gui.App method*), 20  
programs\_widgets() (*GUI.gui.App method*), 20

## R

read\_string() (*src.igus\_modbus.Robot method*), 11  
reference() (*src.igus\_modbus.Robot method*), 11  
remove\_list\_element() (*GUI.gui.App method*), 20  
reset() (*src.igus\_modbus.Robot method*), 12  
Robot (*class in src.igus\_modbus*), 3  
rotate() (*src.gripper.Gripper method*), 18  
run\_list() (*GUI.gui.App method*), 20

## S

save\_list() (*GUI.gui.App method*), 20  
set\_and\_move() (*src.igus\_modbus.Robot method*), 12  
set\_digital\_output() (*src.igus\_modbus.Robot  
method*), 12  
set\_globale\_signal() (*src.igus\_modbus.Robot  
method*), 12  
set\_number\_variables() (*src.igus\_modbus.Robot  
method*), 13  
set\_orientation\_endeffector()  
    (*src.igus\_modbus.Robot method*), 13  
set\_override\_velocity() (*src.igus\_modbus.Robot  
method*), 13  
set\_position\_axes() (*src.igus\_modbus.Robot  
method*), 13  
set\_position\_endeffector() (*src.igus\_modbus.Robot  
method*), 14  
set\_position\_variable() (*src.igus\_modbus.Robot  
method*), 14  
set\_program\_name() (*src.igus\_modbus.Robot method*),  
    15  
set\_program\_replay\_mode() (*src.igus\_modbus.Robot  
method*), 15  
set\_velocity() (*src.igus\_modbus.Robot method*), 15  
set\_zero\_torque() (*src.igus\_modbus.Robot method*),  
    15  
setting\_widgets() (*GUI.gui.App method*), 20  
show\_positions() (*GUI.gui.App method*), 20  
shutdown() (*src.igus\_modbus.Robot method*), 15  
sort\_list() (*GUI.gui.App method*), 20  
speed\_widgets() (*GUI.gui.App method*), 20  
split\_list() (*GUI.gui.App method*), 20  
src.gripper  
    module, 17  
src.igus\_modbus  
    module, 3

## T

tabs() (*GUI.gui.App method*), 20  
teach\_widgets() (*GUI.gui.App method*), 20

## U

update() (*GUI.gui.App method*), 20  
update\_error() (*GUI.gui.App method*), 20  
update\_list() (*GUI.gui.App method*), 20  
update\_tabs() (*GUI.gui.App method*), 20  
update\_theme() (*GUI.gui.App method*), 20

## W

write\_string() (*src.igus\_modbus.Robot method*), 16