Delta-Robot

Release 0.1

Yaman Alsaady

Oct 09, 2023

CONTENTS:

CHAPTER

ONE

MODBUS PACKAGE

1.1 igus_modbus Module

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`

`controll_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 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_porgrams()`

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
True if the mode was successfully set, False if an invalid mode is provided.

Return type
bool

`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).

TWO

GRIPPER PACKAGE

2.1 Gripper Module

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 = 9600*, *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

`controll`(*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`

`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

CHAPTER THREE

GUI PACKAGE

class GUI.gui.**App**(_)

Bases: Frame

add()

clear_list()

control_widgets()

enable_robot()

error_widgets()

fontsize = 20

gripper_mov()

gripper_widgets(*tab_name*, *row*, *column*)

load_pragram()

load_widgets()

logo_widgets(*img='/home/yaman/Git/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()

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_tabs(_)`

`update_theme(_)`

`GUI.gui.main()`

CHAPTER

FOUR

EXAMPLE PACKAGE

4.1 Example.example module

`Example.example.`**`main`**`()`

4.2 Example.example2 module

`Example.example2.`**`main`**`()`

4.3 Example.example3 module

`Example.example3.`**`main`**`()`

4.4 Example.example_gripper module

CHAPTER FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX