Poppy project Documentation

Release 1.1.0

INRIA

October 16, 2015

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Getting Started

1.1.1 Program your robot using Python, Snap!, etc...

CHAPTER

TWO

ADVANCED SECTIONS

2.1 Assembly Guides

2.1.1 Working with Dynamixel servomotor

2.1.2 Assembly guide for a Poppy Humanoid or Poppy Torso robot

2.2 Development Guides

2.2.1 Poppy-humanoid library

2.2.2 Poppy-torso library

2.2.3 Poppy-ergo-jr library

2.2.4 Poppy-creature library

2.2.5 Pypot library

Introduction

Quickstarts

Pypot in details

Other useful features

2.2.6 Setup the internal board

2.2.7 APIs

poppy_humanoid package

Subpackages

poppy_humanoid.primitives package

Submodules

poppy_humanoid.primitives.dance module

poppy_humanoid.primitives.idle module

poppy_humanoid.primitives.interaction module

poppy_humanoid.primitives.posture module

poppy_humanoid.primitives.safe module

Module contents

Submodules

poppy_humanoid.poppy_humanoid module

Module contents

poppy_torso package

Subpackages

poppy_torso.primitives package

Submodules

poppy_torso.primitives.dance module

poppy_torso.primitives.idle module

poppy_torso.primitives.interaction module

poppy_torso.primitives.posture module

poppy_torso.primitives.safe module

Module contents

poppy_torso.utils package

Submodules

poppy_torso.utils.min_jerk module

Module contents

Submodules

poppy_torso.poppy_torso module

Module contents

poppy_ergo_jr package

Submodules

poppy_ergo_jr.dance module

poppy_ergo_jr.headfollow module

poppy_ergo_jr.jump module

poppy_ergo_jr.poppy_ergo_jr module

poppy_ergo_jr.postures module

Module contents

poppy-creature package

Subpackages

poppy.creatures package

Submodules

poppy.creatures.abstractcreature module

poppy.creatures.poppy_sim module

poppy.creatures.services_launcher module

poppy.creatures.snap_launcher module

Module contents

Module contents

pypot package

Subpackages

pypot.dynamixel package

Subpackages

pypot.dynamixel.io package

Submodules

pypot.dynamixel.io.abstract_io module

class pypot.dynamixel.io.abstract_io.**AbstractDxlIO**(*port*, *baudrate=1000000*, *timeout=0.05*, *use_sync_read=False*, *error_handler_cls=None*, *convert=True*)

Bases: pypot.robot.io.AbstractIO

Low-level class to handle the serial communication with the robotis motors.

At instanciation, it opens the serial port and sets the communication parameters.

Parameters

- **port** (*string*) – the serial port to use (e.g. Unix (/dev/tty...), Windows (COM...)).
- **baudrate** (*int*) – default for new motors: 57600, for PyPot motors: 1000000
- **timeout** (*float*) – read timeout in seconds
- **use_sync_read** (*bool*) – whether or not to use the SYNC_READ instruction
- **error_handler** (*DxlErrorHandler*) – set a handler that will receive the different errors
- **convert** (*bool*) – whether or not convert values to units expressed in the standard system

Raises DxlError if the port is already used.

classmethod get_used_ports()

open(*port*, *baudrate=1000000*, *timeout=0.05*)

Opens a new serial communication (closes the previous communication if needed).

Raises DxlError if the port is already used.

close(*_force_lock=False*)

Closes the serial communication if opened.

flush(*_force_lock=False*)

Flushes the serial communication (both input and output).

port

Port used by the DxlIO. If set, will re-open a new connection.

baudrate
Baudrate used by the `DxlIO`. If set, will re-open a new connection.

timeout
Timeout used by the `DxlIO`. If set, will re-open a new connection.

closed
Checks if the connection is closed.

ping(*id*)
Pings the motor with the specified id.

Note: The motor id should always be included in [0, 253]. 254 is used for broadcast.

---

scan(*ids=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253]*)
Pings all ids within the specified list, by default it finds all the motors connected to the bus.

get_model(*ids*)
Gets the model for the specified motors.

change_id(*new_id_for_id*)
Changes the id of the specified motors (each id must be unique on the bus).

change_baudrate(*baudrate_for_ids*)
Changes the baudrate of the specified motors.

get_status_return_level(*ids*, ***kwargs*)
Gets the status level for the specified motors.

set_status_return_level(*srl_for_id*, ***kwargs*)
Sets status return level to the specified motors.

switch_led_on(*ids*)
Switches on the LED of the motors with the specified ids.

switch_led_off(*ids*)
Switches off the LED of the motors with the specified ids.

enable_torque(*ids*)
Enables torque of the motors with the specified ids.

disable_torque(*ids*)
Disables torque of the motors with the specified ids.

get_pid_gain(*ids*, ***kwargs*)
Gets the pid gain for the specified motors.

set_pid_gain(*pid_for_id*, ***kwargs*)
Sets the pid gain to the specified motors.

get_control_table(*ids*, ***kwargs*)
Gets the full control table for the specified motors.

..note:: This function requires the model for each motor to be known. Querring this additional information might add some extra delay.

exception pypot.dynamixel.io.abstract_io.**DxlError**
Bases: exceptions.Exception

Base class for all errors encountered using DxlIO.

exception pypot.dynamixel.io.abstract_io.**DxlCommunicationError**(*dxl_io*, *message*, *instruction_packet*)
Bases: *pypot.dynamixel.io.abstract_io.DxlError*

Base error for communication error encountered when using DxlIO.

exception pypot.dynamixel.io.abstract_io.**DxlTimeoutError**(*dxl_io*, *instruction_packet*, *ids*)
Bases: *pypot.dynamixel.io.abstract_io.DxlCommunicationError*

Timeout error encountered when using DxlIO.

pypot.dynamixel.io.abstract_io.**with_True**(**args*, ***kwds*)

pypot.dynamixel.io.io module

class pypot.dynamixel.io.io.**DxlIO**(*port*, *baudrate=1000000*, *timeout=0.05*, *use_sync_read=False*, *error_handler_cls=None*, *convert=True*)
Bases: *pypot.dynamixel.io.abstract_io.AbstractDxlIO*

At instanciation, it opens the serial port and sets the communication parameters.

Parameters

- **port** (*string*) – the serial port to use (e.g. Unix (/dev/tty...), Windows (COM...)).
- **baudrate** (*int*) – default for new motors: 57600, for PyPot motors: 1000000
- **timeout** (*float*) – read timeout in seconds
- **use_sync_read** (*bool*) – whether or not to use the SYNC_READ instruction
- **error_handler** (*DxlErrorHandler*) – set a handler that will receive the different errors
- **convert** (*bool*) – whether or not convert values to units expressed in the standard system

Raises DxlError if the port is already used.

factory_reset()
Reset all motors on the bus to their factory default settings.

get_control_mode(*ids*)
Gets the mode ('joint' or 'wheel') for the specified motors.

set_wheel_mode(*ids*)
Sets the specified motors to wheel mode.

set_joint_mode(*ids*)
Sets the specified motors to joint mode.

set_control_mode(*mode_for_id*)

set_angle_limit(*limit_for_id*, ***kwargs*)
Sets the angle limit to the specified motors.

get_alarm_LED(*ids*, ***kwargs*)
Gets alarm LED from the specified motors.

get_alarm_shutdown(*ids*, ***kwargs*)
Gets alarm shutdown from the specified motors.

get_angle_limit(*ids*, ***kwargs*)
Gets angle limit from the specified motors.

get_compliance_margin(*ids*, ***kwargs*)
Gets compliance margin from the specified motors.

get_compliance_slope(*ids*, ***kwargs*)
Gets compliance slope from the specified motors.

get_drive_mode(*ids*, ***kwargs*)
Gets drive mode from the specified motors.

get_firmware(*ids*, ***kwargs*)
Gets firmware from the specified motors.

get_goal_position(*ids*, ***kwargs*)
Gets goal position from the specified motors.

get_goal_position_speed_load(*ids*, ***kwargs*)
Gets goal position speed load from the specified motors.

get_highest_temperature_limit(*ids*, ***kwargs*)
Gets highest temperature limit from the specified motors.

get_max_torque(*ids*, ***kwargs*)
Gets max torque from the specified motors.

get_moving_speed(*ids*, ***kwargs*)
Gets moving speed from the specified motors.

get_present_load(*ids*, ***kwargs*)
Gets present load from the specified motors.

get_present_position(*ids*, ***kwargs*)
Gets present position from the specified motors.

get_present_position_speed_load(*ids*, ***kwargs*)
Gets present position speed load from the specified motors.

get_present_speed(*ids*, ***kwargs*)
Gets present speed from the specified motors.

get_present_temperature(*ids*, ***kwargs*)
Gets present temperature from the specified motors.

get_present_voltage(*ids*, ***kwargs*)
Gets present voltage from the specified motors.

get_return_delay_time(*ids*, ***kwargs*)
Gets return delay time from the specified motors.

get_torque_limit(*ids*, ***kwargs*)
Gets torque limit from the specified motors.

get_voltage_limit(*ids*, ***kwargs*)
Gets voltage limit from the specified motors.

is_led_on(*ids*, ***kwargs*)
Gets LED from the specified motors.

is_moving(*ids*, ***kwargs*)
Gets moving from the specified motors.

is_torque_enabled(*ids*, ***kwargs*)
Gets torque_enable from the specified motors.

set_alarm_LED(*value_for_id*, ***kwargs*)
Sets alarm LED to the specified motors.

set_alarm_shutdown(*value_for_id*, ***kwargs*)
Sets alarm shutdown to the specified motors.

set_compliance_margin(*value_for_id*, ***kwargs*)
Sets compliance margin to the specified motors.

set_compliance_slope(*value_for_id*, ***kwargs*)
Sets compliance slope to the specified motors.

set_drive_mode(*value_for_id*, ***kwargs*)
Sets drive mode to the specified motors.

set_goal_position(*value_for_id*, ***kwargs*)
Sets goal position to the specified motors.

set_goal_position_speed_load(*value_for_id*, ***kwargs*)
Sets goal position speed load to the specified motors.

set_highest_temperature_limit(*value_for_id*, ***kwargs*)
Sets highest temperature limit to the specified motors.

set_max_torque(*value_for_id*, ***kwargs*)
Sets max torque to the specified motors.

set_moving_speed(*value_for_id*, ***kwargs*)
Sets moving speed to the specified motors.

set_return_delay_time(*value_for_id*, ***kwargs*)
Sets return delay time to the specified motors.

set_torque_limit(*value_for_id*, ***kwargs*)
Sets torque limit to the specified motors.

set_voltage_limit(*value_for_id*, ***kwargs*)
Sets voltage limit to the specified motors.

pypot.dynamixel.io.io_320 module

class pypot.dynamixel.io.io_320.**Dxl320IO**(*port*, *baudrate=1000000*, *timeout=0.05*, *use_sync_read=False*, *error_handler_cls=None*, *convert=True*)

Bases: *pypot.dynamixel.io.abstract_io.AbstractDxlIO*

At instanciation, it opens the serial port and sets the communication parameters.

Parameters

- **port** (*string*) – the serial port to use (e.g. Unix (/dev/tty...), Windows (COM...)).
- **baudrate** (*int*) – default for new motors: 57600, for PyPot motors: 1000000
- **timeout** (*float*) – read timeout in seconds

- **use_sync_read** (*bool*) – whether or not to use the SYNC_READ instruction
- **error_handler** (*DxlErrorHandler*) – set a handler that will receive the different errors
- **convert** (*bool*) – whether or not convert values to units expressed in the standard system

Raises `DxlError` if the port is already used.

set_wheel_mode(*ids*)

set_joint_mode(*ids*)

get_goal_position_speed_load(*ids*)

set_goal_position_speed_load(*value_for_ids*)

factory_reset(*ids*, *except_ids=False*, *except_baudrate_and_ids=False*)
Reset all motors on the bus to their factory default settings.

get_LED_color(*ids*, ***kwargs*)
Gets LED color from the specified motors.

get_alarm_shutdown(*ids*, ***kwargs*)
Gets alarm shutdown from the specified motors.

get_angle_limit(*ids*, ***kwargs*)
Gets angle limit from the specified motors.

get_control_mode(*ids*, ***kwargs*)
Gets control mode from the specified motors.

get_firmware(*ids*, ***kwargs*)
Gets firmware from the specified motors.

get_goal_position(*ids*, ***kwargs*)
Gets goal position from the specified motors.

get_highest_temperature_limit(*ids*, ***kwargs*)
Gets highest temperature limit from the specified motors.

get_max_torque(*ids*, ***kwargs*)
Gets max torque from the specified motors.

get_moving_speed(*ids*, ***kwargs*)
Gets moving speed from the specified motors.

get_present_load(*ids*, ***kwargs*)
Gets present load from the specified motors.

get_present_position(*ids*, ***kwargs*)
Gets present position from the specified motors.

get_present_position_speed_load(*ids*, ***kwargs*)
Gets present position speed load from the specified motors.

get_present_speed(*ids*, ***kwargs*)
Gets present speed from the specified motors.

get_present_temperature(*ids*, ***kwargs*)
Gets present temperature from the specified motors.

get_present_voltage(*ids*, ***kwargs*)
Gets present voltage from the specified motors.

get_return_delay_time(*ids*, ***kwargs*)
Gets return delay time from the specified motors.

get_torque_limit(*ids*, ***kwargs*)
Gets torque limit from the specified motors.

get_voltage_limit(*ids*, ***kwargs*)
Gets voltage limit from the specified motors.

is_led_on(*ids*, ***kwargs*)
Gets LED from the specified motors.

is_moving(*ids*, ***kwargs*)
Gets moving from the specified motors.

is_torque_enabled(*ids*, ***kwargs*)
Gets torque_enable from the specified motors.

set_LED_color(*value_for_id*, ***kwargs*)
Sets LED color to the specified motors.

set_alarm_shutdown(*value_for_id*, ***kwargs*)
Sets alarm shutdown to the specified motors.

set_angle_limit(*value_for_id*, ***kwargs*)
Sets angle limit to the specified motors.

set_control_mode(*value_for_id*, ***kwargs*)
Sets control mode to the specified motors.

set_goal_position(*value_for_id*, ***kwargs*)
Sets goal position to the specified motors.

set_highest_temperature_limit(*value_for_id*, ***kwargs*)
Sets highest temperature limit to the specified motors.

set_max_torque(*value_for_id*, ***kwargs*)
Sets max torque to the specified motors.

set_moving_speed(*value_for_id*, ***kwargs*)
Sets moving speed to the specified motors.

set_return_delay_time(*value_for_id*, ***kwargs*)
Sets return delay time to the specified motors.

set_torque_limit(*value_for_id*, ***kwargs*)
Sets torque limit to the specified motors.

set_voltage_limit(*value_for_id*, ***kwargs*)
Sets voltage limit to the specified motors.

Module contents

pypot.dynamixel.protocol package

Submodules

pypot.dynamixel.protocol.v1 module

class pypot.dynamixel.protocol.v1.**DxlInstruction**

Bases: object

PING = 1

READ_DATA = 2

WRITE_DATA = 3

RESET = 6

SYNC_WRITE = 131

SYNC_READ = 132

class pypot.dynamixel.protocol.v1.**DxlPacketHeader**

Bases: pypot.dynamixel.protocol.v1.DxlPacketHeader

This class represents the header of a Dxl Packet.

They are constructed as follows [0xFF, 0xFF, ID, LENGTH] where:

- ID represents the ID of the motor who received (resp. sent) the intruction (resp. status) packet.
- LENGTH represents the length of the rest of the packet

length = 4

marker = bytearray(b'\xff\xff')

classmethod from_string(*data*)

class pypot.dynamixel.protocol.v1.**DxlInstructionPacket**

Bases: pypot.dynamixel.protocol.v1.DxlInstructionPacket

This class is used to represent a dynamixel instruction packet.

An instruction packet is constructed as follows: [0xFF, 0xFF, ID, LENGTH, INSTRUCTION, PARAM 1, PARAM 2, ..., PARAM N, CHECKSUM]

(for more details see http://support.robotis.com/en/product/dxl_main.htm)

to_array()

to_string()

length

checksum

class pypot.dynamixel.protocol.v1.**DxlPingPacket**

Bases: pypot.dynamixel.protocol.v1.DxlInstructionPacket

This class is used to represent ping packet.

class pypot.dynamixel.protocol.v1.**DxlResetPacket**

Bases: pypot.dynamixel.protocol.v1.DxlInstructionPacket

This class is used to represent reset packet.

class pypot.dynamixel.protocol.v1.**DxlReadDataPacket**

Bases: pypot.dynamixel.protocol.v1.DxlInstructionPacket

This class is used to represent read data packet (to read value).

class pypot.dynamixel.protocol.v1.**DxlSyncReadPacket**

Bases: pypot.dynamixel.protocol.v1.DxlInstructionPacket

This class is used to represent sync read packet (to synchronously read values).

class pypot.dynamixel.protocol.v1.**DxlWriteDataPacket**
Bases: pypot.dynamixel.protocol.v1.DxlInstructionPacket

This class is used to reprensent write data packet (to write value).

class pypot.dynamixel.protocol.v1.**DxlSyncWritePacket**
Bases: pypot.dynamixel.protocol.v1.DxlInstructionPacket

This class is used to represent sync write packet (to synchronously write values).

class pypot.dynamixel.protocol.v1.**DxlStatusPacket**
Bases: pypot.dynamixel.protocol.v1.DxlStatusPacket

This class is used to represent a dynamixel status packet.

A status packet is constructed as follows: [0xFF, 0xFF, ID, LENGTH, ERROR, PARAM 1, PARAM 2, ..., PARAM N, CHECKSUM]

(for more details see http://support.robotis.com/en/product/dxl_main.htm)

classmethod from_string(*data*)

pypot.dynamixel.protocol.v2 module

class pypot.dynamixel.protocol.v2.**DxlInstruction**
Bases: object

PING = 1

READ_DATA = 2

WRITE_DATA = 3

RESET = 6

SYNC_READ = 130

SYNC_WRITE = 131

class pypot.dynamixel.protocol.v2.**DxlPacketHeader**
Bases: pypot.dynamixel.protocol.v2.DxlPacketHeader

This class represents the header of a Dxl Packet.

They are constructed as follows [0xFF, 0xFF, 0xFD, 0x00, ID, LEN_L, LEN_H] where:

- ID represents the ID of the motor who received (resp. sent) the intruction (resp. status) packet.
- LEN_L, LEN_H represents the length of the rest of the packet

length = 7

marker = bytearray(b'\xff\xff\xfd\x00')

classmethod from_string(*data*)

class pypot.dynamixel.protocol.v2.**DxlInstructionPacket**
Bases: pypot.dynamixel.protocol.v2.DxlInstructionPacket

This class is used to represent a dynamixel instruction packet.

An instruction packet is constructed as follows: [0xFF, 0xFF, 0xFD, 0x00, ID, LEN_L, LEN_H, INST, PARAM 1, PARAM 2, ..., PARAM N, CRC_L, CRC_H]

(for more details see http://support.robotis.com/en/product/dxl_main.htm)

to_array()

to_string()

length

checksum

class pypot.dynamixel.protocol.v2.**DxlPingPacket**
Bases: pypot.dynamixel.protocol.v2.DxlInstructionPacket

This class is used to represent ping packet.

class pypot.dynamixel.protocol.v2.**DxlResetPacket**
Bases: pypot.dynamixel.protocol.v2.DxlInstructionPacket

This class is used to represent factory reset packet.

class pypot.dynamixel.protocol.v2.**DxlReadDataPacket**
Bases: pypot.dynamixel.protocol.v2.DxlInstructionPacket

This class is used to represent read data packet (to read value).

class pypot.dynamixel.protocol.v2.**DxlSyncReadPacket**
Bases: pypot.dynamixel.protocol.v2.DxlInstructionPacket

This class is used to represent sync read packet (to synchronously read values).

class pypot.dynamixel.protocol.v2.**DxlWriteDataPacket**
Bases: pypot.dynamixel.protocol.v2.DxlInstructionPacket

This class is used to reprensent write data packet (to write value).

class pypot.dynamixel.protocol.v2.**DxlSyncWritePacket**
Bases: pypot.dynamixel.protocol.v2.DxlInstructionPacket

This class is used to represent sync write packet (to synchronously write values).

class pypot.dynamixel.protocol.v2.**DxlStatusPacket**
Bases: pypot.dynamixel.protocol.v2.DxlStatusPacket

This class is used to represent a dynamixel status packet.

A status packet is constructed as follows: [0xFF, 0xFF, 0xFD, 0x00, ID, LEN_L, LEN_H, 0x55, ERROR, PARAM 1, PARAM 2, ..., PARAM N, CRC_L, CRC_H]

(for more details see http://support.robotis.com/en/product/dxl_main.htm)

classmethod from_string(*data*)

pypot.dynamixel.protocol.v2.**crc16**(*data_blk*, *data_blk_size*, *crc_accum=0*)

Module contents

Submodules

pypot.dynamixel.controller module

class pypot.dynamixel.controller.**DxlController**(*io*, *motors*, *sync_freq*, *synchronous*, *mode*, *regname*, *varname=None*)
Bases: pypot.robot.controller.MotorsController

working_motors

synced_motors

setup()

update()

get_register(*motors*)
Gets the value from the specified register and sets it to the *DxlMotor*.

set_register(*motors*)
Gets the value from *DxlMotor* and sets it to the specified register.

class pypot.dynamixel.controller.**AngleLimitRegisterController**(*io*, *motors*, *sync_freq*, *synchronous*)
Bases: *pypot.dynamixel.controller.DxlController*

synced_motors

get_register(*motors*)

class pypot.dynamixel.controller.**PosSpeedLoadDxlController**(*io*, *motors*, *sync_freq*)
Bases: *pypot.dynamixel.controller.DxlController*

setup()

update()

get_present_position_speed_load(*motors*)

set_goal_position_speed_load(*motors*)

pypot.dynamixel.conversion module This module describes all the conversion method used to transform value from the representation used by the dynamixel motor to a more standard form (e.g. degrees, volt...).

For compatibility issue all comparison method should be written in the following form (even if the model is not actually used):

- def my_conversion_from_dxl_to_si(value, model): ...
- def my_conversion_from_si_to_dxl(value, model): ...

Note: If the control is readonly you only need to write the dxl_to_si conversion.

pypot.dynamixel.conversion.**dxl_to_degree**(*value*, *model*)

pypot.dynamixel.conversion.**degree_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_speed**(*value*, *model*)

pypot.dynamixel.conversion.**speed_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_torque**(*value*, *model*)

pypot.dynamixel.conversion.**torque_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_load**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_pid**(*value*, *model*)

pypot.dynamixel.conversion.**pid_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_model**(*value*, *dummy=None*)

pypot.dynamixel.conversion.**check_bit**(*value*, *offset*)

pypot.dynamixel.conversion.**dxl_to_drive_mode**(*value*, *model*)

pypot.dynamixel.conversion.**drive_mode_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_baudrate**(*value*, *model*)

pypot.dynamixel.conversion.**baudrate_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_rdt**(*value*, *model*)

pypot.dynamixel.conversion.**rdt_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_temperature**(*value*, *model*)

pypot.dynamixel.conversion.**temperature_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_voltage**(*value*, *model*)

pypot.dynamixel.conversion.**voltage_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_status**(*value*, *model*)

pypot.dynamixel.conversion.**status_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_alarm**(*value*, *model*)

pypot.dynamixel.conversion.**decode_error**(*error_code*)

pypot.dynamixel.conversion.**alarm_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**XL320LEDColors**
alias of Colors

pypot.dynamixel.conversion.**dxl_to_led_color**(*value*, *model*)

pypot.dynamixel.conversion.**led_color_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_to_control_mode**(*value*, _)

pypot.dynamixel.conversion.**control_mode_to_dxl**(*mode*, _)

pypot.dynamixel.conversion.**dxl_to_bool**(*value*, *model*)

pypot.dynamixel.conversion.**bool_to_dxl**(*value*, *model*)

pypot.dynamixel.conversion.**dxl_decode**(*data*)

pypot.dynamixel.conversion.**dxl_decode_all**(*data*, *nb_elem*)

pypot.dynamixel.conversion.**dxl_code**(*value*, *length*)

pypot.dynamixel.conversion.**dxl_code_all**(*value*, *length*, *nb_elem*)

pypot.dynamixel.error module

class pypot.dynamixel.error.**DxlErrorHandler**

Bases: object

This class is used to represent all the error that you can/should handle.

The errors can be of two types:

- communication error (timeout, communication)
- motor error (voltage, limit, overload...)

This class was designed as an abstract class and so you should write your own handler by subclassing this class and defining the apropriate behavior for your program.

> **Warning:** The motor error should be overload carrefuly as they can indicate important mechanical issue.

handle_timeout(*timeout_error*)

handle_communication_error(*communication_error*)

handle_input_voltage_error(*instruction_packet*)

handle_angle_limit_error(*instruction_packet*)

handle_overheating_error(*instruction_packet*)

handle_range_error(*instruction_packet*)

handle_checksum_error(*instruction_packet*)

handle_overload_error(*instruction_packet*)

handle_instruction_error(*instruction_packet*)

handle_none_error(*instruction_packet*)

class pypot.dynamixel.error.**BaseErrorHandler**

Bases: *pypot.dynamixel.error.DxlErrorHandler*

This class is a basic handler that just skip the communication errors.

handle_timeout(*timeout_error*)

handle_communication_error(*com_error*)

handle_none_error(*instruction_packet*)

pypot.dynamixel.motor module

class pypot.dynamixel.motor.**DxlRegister**(*rw=False*)

Bases: object

class pypot.dynamixel.motor.**DxlOrientedRegister**(*rw=False*)

Bases: *pypot.dynamixel.motor.DxlRegister*

class pypot.dynamixel.motor.**DxlPositionRegister**(*rw=False*)

Bases: *pypot.dynamixel.motor.DxlOrientedRegister*

class pypot.dynamixel.motor.**RegisterOwner**

Bases: type

class pypot.dynamixel.motor.**DxlMotor**(*id*, *name=None*, *model=''*, *direct=True*, *offset=0.0*, *broken=False*)

Bases: pypot.robot.motor.Motor

High-level class used to represent and control a generic dynamixel motor.

This class provides all level access to (see *registers* for an exhaustive list):

- motor id
- motor name
- motor model
- present position/speed/load
- goal position/speed/load
- compliant
- motor orientation and offset
- angle limit
- temperature
- voltage

This class represents a generic robotis motor and you define your own subclass for specific motors (see *DxlMXMotor* or *DxlAXRXMotor*).

Those properties are synchronized with the real motors values thanks to a *DxlController*.

registers = ['registers', 'goal_speed', 'compliant', 'safe_compliant', 'angle_limit', 'present_load', 'id', 'present_tempe

goal_speed
Goal speed (in degrees per second) of the motor.

This property can be used to control your motor in speed. Setting a goal speed will automatically change the moving speed and sets the goal position as the angle limit.

Note: The motor will turn until reaching the angle limit. But this is not a wheel mode, so the motor will stop at its limits.

---

compliant_behavior

compliant

angle_limit

goto_behavior

goto_position(*position*, *duration*, *control=None*, *wait=False*)
Automatically sets the goal position and the moving speed to reach the desired position within the duration.

class pypot.dynamixel.motor.**DxlAXRXMotor**(*id*, *name=None*, *model=''*, *direct=True*, *offset=0.0*, *broken=False*)
Bases: *pypot.dynamixel.motor.DxlMotor*

This class represents the AX robotis motor.

This class adds access to:

- compliance margin/slope (see the robotis website for details)

registers = ['registers', 'goal_speed', 'compliant', 'safe_compliant', 'angle_limit', 'present_load', 'id', 'present_tempe

class pypot.dynamixel.motor.**DxlMXMotor**(*id*, *name=None*, *model=''*, *direct=True*, *offset=0.0*, *broken=False*)
Bases: *pypot.dynamixel.motor.DxlMotor*

This class represents the RX and MX robotis motor.

This class adds access to:

- PID gains (see the robotis website for details)

This class represents the RX and MX robotis motor.

This class adds access to:

- PID gains (see the robotis website for details)

registers = ['registers', 'goal_speed', 'compliant', 'safe_compliant', 'angle_limit', 'present_load', 'id', 'present_tempe

class pypot.dynamixel.motor.**DxlXL320Motor**(*id*, *name=None*, *model='XL-320'*, *direct=True*, *offset=0.0*, *broken=False*)
Bases: *pypot.dynamixel.motor.DxlMXMotor*

registers = ['registers', 'goal_speed', 'compliant', 'safe_compliant', 'angle_limit', 'present_load', 'id', 'present_tempe

class pypot.dynamixel.motor.**SafeCompliance**(*motor*, *frequency=50*)
Bases: pypot.utils.stoppablethread.StoppableLoopThread

This class creates a controller to active compliance only if the current motor position is included in the angle limit, else the compliance is turned off.

update()

teardown()

pypot.dynamixel.syncloop module

class pypot.dynamixel.syncloop.**MetaDxlController**(*io*, *motors*, *controllers*)

Bases: pypot.robot.controller.MotorsController

Synchronizes the reading/writing of *DxlMotor* with the real motors.

This class handles synchronization loops that automatically read/write values from the "software" *DxlMotor* with their "hardware" equivalent. Those loops shared a same DxlIO connection to avoid collision in the bus. Each loop run within its own thread as its own frequency.

> **Warning:** As all the loop attached to a controller shared the same bus, you should make sure that they can run without slowing down the other ones.

setup()

Starts all the synchronization loops.

update()

teardown()

Stops the synchronization loops.

class pypot.dynamixel.syncloop.**BaseDxlController**(*io*, *motors*)

Bases: *pypot.dynamixel.syncloop.MetaDxlController*

Implements a basic controller that synchronized the most frequently used values.

More precisely, this controller:

- reads the present position, speed, load at 50Hz
- writes the goal position, moving speed and torque limit at 50Hz
- writes the pid gains (or compliance margin and slope) at 10Hz
- reads the present voltage and temperature at 1Hz

class pypot.dynamixel.syncloop.**LightDxlController**(*io*, *motors*)

Bases: *pypot.dynamixel.syncloop.MetaDxlController*

Module contents

pypot.dynamixel.**get_available_ports**(*only_free=False*)

pypot.dynamixel.**get_port_vendor_info**(*port=None*)

Return vendor informations of a usb2serial device. It may depends on the Operating System. :param string port: port of the usb2serial device

Example

Result with a USB2Dynamixel on Linux: In [1]: import pypot.dynamixel In [2]: pypot.dynamixel.get_port_vendor_info('/dev/ttyUSB0') Out[2]: 'USB VID:PID=0403:6001 SNR=A7005LKE'

pypot.dynamixel.**find_port**(*ids*, *strict=True*)

Find the port with the specified attached motor ids.

Parameters

- **ids** (*list*) – list of motor ids to find
- **strict** (*bool*) – specify if all ids should be find (when set to False, only half motor must be found)

Warning: If two (or more) ports are attached to the same list of motor ids the first match will be returned.

pypot.dynamixel.**autodetect_robot**()
Creates a Robot by detecting dynamixel motors on all available ports.

pypot.primitive package

Submodules

pypot.primitive.manager module

class pypot.primitive.manager.**PrimitiveManager**(*motors*, *freq=50*, *filter=<functools.partial object>*)
Bases: pypot.utils.stoppablethread.StoppableLoopThread

Combines all Primitive orders and affect them to the real motors.

At a predefined frequency, the manager gathers all the orders sent by the primitive to the "fake" motors, combined them thanks to the filter function and affect them to the "real" motors.

Note: The primitives are automatically added (resp. removed) to the manager when they are started (resp. stopped).

Parameters

- **motors** (list of *DxlMotor*) – list of real motors used by the attached primitives
- **freq** (*int*) – update frequency
- **filter** (*func*) – function used to combine the different request (default mean)

add(*p*)
Add a primitive to the manager. The primitive automatically attached itself when started.

remove(*p*)
Remove a primitive from the manager. The primitive automatically remove itself when stopped.

primitives
List of all attached Primitive.

update()
Combined at a predefined frequency the request orders and affect them to the real motors.

stop()
Stop the primitive manager.

pypot.primitive.move module

class pypot.primitive.move.**Move**(*freq*)
Bases: object

Simple class used to represent a movement.

This class simply wraps a sequence of positions of specified motors. The sequence must be recorded at a predefined frequency. This move can be recorded through the `MoveRecorder` class and played thanks to a `MovePlayer`.

framerate

add_position(*pos*, *time*)
Add a new position to the movement sequence.

Each position is typically stored as a dict of (time, (motor_name,motor_position)).

iterpositions()
Returns an iterator on the stored positions.

positions()
Returns a copy of the stored positions.

save(*file*)
Saves the `Move` to a json file.

Note: The format used to store the `Move` is extremely verbose and should be obviously optimized for long moves.

---

classmethod load(*file*)
Loads a `Move` from a json file.

class pypot.primitive.move.**MoveRecorder**(*robot*, *freq*, *tracked_motors*)
Bases: pypot.primitive.primitive.LoopPrimitive

Primitive used to record a `Move`.

The recording can be `start()` and `stop()` by using the `LoopPrimitive` methods.

Note: Re-starting the recording will create a new `Move` losing all the previously stored data.

---

setup()

update()

move
Returns the currently recorded `Move`.

add_tracked_motors(*tracked_motors*)
Add new motors to the recording

class pypot.primitive.move.**MovePlayer**(*robot*, *move=None*, *play_speed=1.0*, *move_filename=None*, *start_max_speed=50*, ***kwargs*)
Bases: pypot.primitive.primitive.LoopPrimitive

Primitive used to play a `Move`.

The playing can be `start()` and `stop()` by using the `LoopPrimitive` methods.

Warning: the primitive is run automatically the same framerate than the move record. The play_speed attribute change only time lockup/interpolation

setup()

update()

duration()

pypot.primitive.primitive module

class pypot.primitive.primitive.**Primitive**(*robot*)

Bases: pypot.utils.stoppablethread.StoppableThread

A Primitive is an elementary behavior that can easily be combined to create more complex behaviors.

A primitive is basically a thread with access to a "fake" robot to ensure a sort of sandboxing. More precisely, it means that the primitives will be able to:

- request values from the real robot (motor values, sensors or attached primitives)
- request modification of motor values (those calls will automatically be combined among all primitives by the PrimitiveManager).

The syntax of those requests directly match the equivalent code that you could write from the Robot. For instance you can write:

```
class MyPrimitive(Primitive):
    def run(self):
        while True:
            for m in self.robot.motors:
                m.goal_position = m.present_position + 10

            time.sleep(1)
```

Warning: In the example above, while it seems that you are setting a new goal_position, you are only requesting it. In particular, another primitive could request another goal_position and the result will be the combination of both request. For example, if you have two primitives: one setting the goal_position to 10 and the other setting the goal_position to -20, the real goal_position will be set to -5 (by default the mean of all request is used, see the PrimitiveManager class for details).

Primitives were developed to allow for the creation of complex behaviors such as walking. You could imagine - and this is what is actually done on the Poppy robot - having one primitive for the walking gait, another for the balance and another for handling falls.

Note: This class should always be extended to define your particular behavior in the run() method.

At instanciation, it automatically transforms the Robot into a MockupRobot.

Warning: You should not directly pass motors as argument to the primitive. If you need to, use the method get_mockup_motor() to transform them into "fake" motors. See the write_own_prim section for details.

methods = ['start', 'stop', 'pause', 'resume']

properties = []

setup()

Setup methods called before the run loop.

You can override this method to setup the environment needed by your primitive before the run loop. This method will be called every time the primitive is started/restarted.

run()

Run method of the primitive thread. You should always overwrite this method.

> **Warning:** You are responsible of handling the `should_stop()`, `should_pause()` and `wait_to_resume()` methods correctly so the code inside your run function matches the desired behavior. You can refer to the code of the `run()` method of the `LoopPrimitive` as an example.

After termination of the run function, the primitive will automatically be removed from the list of active primitives of the `PrimitiveManager`.

teardown()
Tear down methods called after the run loop.

You can override this method to clean up the environment needed by your primitive. This method will be called every time the primitive is stopped.

elapsed_time
Elapsed time (in seconds) since the primitive runs.

start()
Start or restart (the `stop()` method will automatically be called) the primitive.

stop(*wait=True*)
Requests the primitive to stop.

is_alive()
Determines whether the primitive is running or not.

The value will be true only when the `run()` function is executed.

get_mockup_motor(*motor*)
Gets the equivalent `MockupMotor`.

class pypot.primitive.primitive.**LoopPrimitive**(*robot*, *freq*)
Bases: pypot.primitive.primitive.Primitive

Simple primitive that call an update method at a predefined frequency.

You should write your own subclass where you only defined the `update()` method.

recent_update_frequencies
Returns the 10 most recent update frequencies.

The given frequencies are computed as short-term frequencies! The 0th element of the list corresponds to the most recent frequency.

run()
Calls the `update()` method at a predefined frequency (runs until stopped).

update()
Update methods that will be called at a predefined frequency.

class pypot.primitive.primitive.**MockupRobot**(*robot*)
Bases: object

Fake `Robot` used by the `Primitive` to ensure sandboxing.

goto_position(*position_for_motors*, *duration*, *control=None*, *wait=False*)

motors
List of all attached `MockupMotor`.

power_max()

class pypot.primitive.primitive.**MockupMotor**(*motor*)
Bases: object

Fake Motor used by the primitive to ensure sandboxing:

- the read instructions are directly delegate to the real motor
- the write instructions are stored as request waiting to be combined by the primitive manager.

goto_position(*position*, *duration*, *control=None*, *wait=False*)
Automatically sets the goal position and the moving speed to reach the desired position within the duration.

goal_speed
Goal speed (in degrees per second) of the motor.

This property can be used to control your motor in speed. Setting a goal speed will automatically change the moving speed and sets the goal position as the angle limit.

Note: The motor will turn until reaching the angle limit. But this is not a wheel mode, so the motor will stop at its limits.

---

pypot.primitive.utils module

class pypot.primitive.utils.**Sinus**(*robot*, *refresh_freq*, *motor_list*, *amp=1*, *freq=0.5*, *offset=0*, *phase=0*)
Bases: pypot.primitive.primitive.LoopPrimitive

Apply a sinus on the motor specified as argument. Parameters (amp, offset and phase) should be specified in degree.

properties = ['frequency', 'amplitude', 'offset', 'phase']

update()
Compute the sin(t) where t is the elapsed time since the primitive has been started.

frequency

amplitude

offset

phase

class pypot.primitive.utils.**Cosinus**(*robot*, *refresh_freq*, *motor_list*, *amp=1*, *freq=0.5*, *offset=0*, *phase=0*)
Bases: pypot.primitive.utils.Sinus

Apply a cosinus on the motor specified as argument. Parameters (amp, offset and phase) should be specified in degree.

class pypot.primitive.utils.**Square**(*robot*, *refresh_freq*, *motor_list*, *amp=1*, *freq=1.0*, *offset=0*, *phase=0*, *duty=0.5*)
Bases: pypot.primitive.utils.Sinus

Apply a square signal. Param (amp, freq, offset, phase, duty cycle).

update()

duty

class pypot.primitive.utils.**PositionWatcher**(*robot*, *refresh_freq*, *watched_motors*)
Bases: pypot.primitive.primitive.LoopPrimitive

record_positions

setup()

update()

plot(*ax*)

class pypot.primitive.utils.**SimplePosture**(*robot*, *duration*)
Bases: pypot.primitive.primitive.Primitive

setup()

run()

teardown()

Module contents

pypot.robot package

Submodules

pypot.robot.config module The config module allows the definition of the structure of your robot.

Configuration are written as Python dictionary so you can define/modify them programmatically. You can also import them form file such as JSON formatted file. In the configuration you have to define:

- controllers: For each defined controller, you can specify the port name, the attached motors and the synchronization mode.
- motors: You specify all motors belonging to your robot. You have to define their id, type, orientation, offset and angle_limit.
- motorgroups: It allows to define alias of group of motors. They can be nested.

pypot.robot.config.**from_config**(*config*, *strict=True*, *sync=True*, *use_dummy_io=False*)
Returns a Robot instance created from a configuration dictionnary.

Parameters

- **config** (*dict*) – robot configuration dictionary
- **strict** (*bool*) – make sure that all ports, motors are availaible.
- **sync** (*bool*) – choose if automatically starts the synchronization loops

For details on how to write such a configuration dictionnary, you should refer to the section config_file.

pypot.robot.config.**motor_from_configgnode**(*config*, *motor_name*)

pypot.robot.config.**sensor_from_configgnode**(*config*, *s_name*, *robot*)

pypot.robot.config.**dxl_io_from_configgnode**(*config*, *c_params*, *ids*, *strict*)

pypot.robot.config.**check_motor_limits**(*config*, *dxl_io*, *motor_names*)

pypot.robot.config.**instatiate_motors**(*config*)

pypot.robot.config.**make_alias**(*config*, *robot*)

pypot.robot.config.**from_json**(*json_file*, *sync=True*, *strict=True*, *use_dummy_io=False*)
Returns a Robot instance created from a JSON configuration file.

For details on how to write such a configuration file, you should refer to the section config_file.

pypot.robot.config.**use_dummy_robot**(*json_file*)

pypot.robot.controller module

class pypot.robot.controller.**AbstractController**(*io*, *sync_freq*)
Bases: pypot.utils.stoppablethread.StoppableLoopThread

Abstract class for motor/sensor controller.

The controller role is to synchronize the reading/writing of a set of instances with their "hardware" equivalent through an AbstractIO object. It is defined as a StoppableLoopThread where each loop update synchronizes values from the "software" objects with their "hardware" equivalent.

To define your Controller, you need to define the update() method. This method will be called at the predefined frequency. An exemple of how to do it can be found in BaseDxlController.

Parameters

- **io** (AbstractIO) – IO used to communicate with the hardware motors
- **sync_freq** (*float*) – synchronization frequency

start()

close()
Cleans and closes the controller.

class pypot.robot.controller.**MotorsController**(*io*, *motors*, *sync_freq=50*)
Bases: pypot.robot.controller.AbstractController

Abstract class for motors controller.

The controller synchronizes the reading/writing of a set of motor instances with their "hardware". Each update loop synchronizes values from the "software" *DxlMotor* with their "hardware" equivalent.

Parameters

- **io** (AbstractIO) – IO used to communicate with the hardware motors
- **motors** (*list*) – list of motors attached to the controller
- **sync_freq** (*float*) – synchronization frequency

class pypot.robot.controller.**DummyController**(*motors*)
Bases: pypot.robot.controller.MotorsController

update()

class pypot.robot.controller.**SensorsController**(*io*, *sensors*, *sync_freq=50.0*)
Bases: pypot.robot.controller.AbstractController

Abstract class for sensors controller.

The controller frequently pulls new data from a "real" sensor and updates its corresponding software instance.

Parameters

- **io** (AbstractIO) – IO used to communicate with the hardware motors
- **sensors** (*list*) – list of sensors attached to the controller
- **sync_freq** (*float*) – synchronization frequency

pypot.robot.io module

class pypot.robot.io.**AbstractIO**
Bases: object

AbstractIO class which handles communication with "hardware" motors.

close()
Clean and close the IO connection.

pypot.robot.motor module

class pypot.robot.motor.**Motor**(*name*)
Bases: object

Purely abstract class representing any motor object.

registers = []

name

pypot.robot.remote module

class pypot.robot.remote.**RemoteRobotClient**(*host*, *port*)
Bases: object

Remote Access to a Robot through the REST API.

This RemoteRobot gives you access to motors and alias. For each motor you can read/write all of their registers.

You also have access to primitives. More specifically you can start/stop them.

pypot.robot.remote.**from_remote**(*host*, *port*)
Remote access to a Robot through the REST API.

pypot.robot.robot module

class pypot.robot.robot.**Robot**(*motor_controllers=[]*, *sensor_controllers=[]*, *sync=True*)
Bases: object

This class is used to regroup all motors and sensors of your robots.

Most of the time, you do not want to directly instantiate this class, but you rather want to use a factory which creates a robot instance - e.g. from a python dictionnary (see config_file).

This class encapsulates the different controllers (such as dynamixel ones) that automatically synchronize the virtual sensors/effectors instances held by the robot class with the real devices. By doing so, each sensor/effector can be synchronized at a different frequency.

This class also provides a generic motors accessor in order to (more or less) easily extends this class to other types of motor.

Parameters

- **motor_controllers** (*list*) – motors controllers to attach to the robot
- **sensor_controllers** (*list*) – sensors controllers to attach to the robot
- **sync** (*bool*) – choose if automatically starts the synchronization loops

close()
Cleans the robot by stopping synchronization and all controllers.

start_sync()
Starts all the synchonization loop (sensor/effector controllers).

stop_sync()
Stops all the synchonization loop (sensor/effector controllers).

attach_primitive(*primitive*, *name*)

motors
Returns all the motors attached to the robot.

sensors
Returns all the sensors attached to the robot.

active_primitives
Returns all the primitives currently running on the robot.

primitives
Returns all the primitives name attached to the robot.

compliant
Returns a list of all the compliant motors.

goto_position(*position_for_motors*, *duration*, *control=None*, *wait=False*)
Moves a subset of the motors to a position within a specific duration.

Parameters

- **position_for_motors** (*dict*) – which motors you want to move {motor_name: pos, motor_name: pos,...}
- **duration** (*float*) – duration of the move
- **control** (*str*) – control type ('dummy', 'minjerk')
- **wait** (*bool*) – whether or not to wait for the end of the move

power_up()
Changes all settings to guarantee the motors will be used at their maximum power.

to_config()
Generates the config for the current robot.

Note: The generated config should be used as a basis and must probably be modified.

pypot.robot.sensor module

class pypot.robot.sensor.**Sensor**(*name*)
Bases: object

Purely abstract class representing any sensor object.

registers = []

name

class pypot.robot.sensor.**ObjectTracker**(*name*)
Bases: pypot.robot.sensor.Sensor

registers = ['position', 'orientation']

position

orientation

Module contents

pypot.sensor package

Subpackages

pypot.sensor.camera package

Submodules

pypot.sensor.camera.abstractcam module

class pypot.sensor.camera.abstractcam.**AbstractCamera**(*name*, *resolution*, *fps*)

Bases: pypot.robot.sensor.Sensor

registers = ['frame', 'resolution', 'fps']

frame

post_processing(*image*)

grab()

resolution

fps

pypot.sensor.camera.opencvcam module

pypot.sensor.camera.rpicam module

Module contents

class pypot.sensor.camera.**CameraController**(*camera*)

Bases: pypot.robot.controller.SensorsController

pypot.sensor.imagefeature package

Submodules

pypot.sensor.imagefeature.blob module

pypot.sensor.imagefeature.face module

pypot.sensor.imagefeature.marker module

Module contents

pypot.sensor.kinect package

Submodules

pypot.sensor.kinect.sensor module This code has been developed by Baptiste Busch: https://github.com/buschbapti

This module allows you to retrieve Skeleton information from a Kinect device. It is only the client side of a zmq client/server application.

The server part can be found at: https://bitbucket.org/buschbapti/kinectserver/src It used the Microsoft Kinect SDK and thus only work on Windows.

Of course, the client side can be used on any platform.

class pypot.sensor.kinect.sensor.**Skeleton**
Bases: pypot.sensor.kinect.sensor.Skeleton

joints = ('hip_center', 'spine', 'shoulder_center', 'head', 'shoulder_left', 'elbow_left', 'wrist_left', 'hand_left', 'shoulde

class pypot.sensor.kinect.sensor.**Joint**(*position*, *orientation*, *pixel_coordinate*)
Bases: tuple

orientation
Alias for field number 1

pixel_coordinate
Alias for field number 2

position
Alias for field number 0

class pypot.sensor.kinect.sensor.**KinectSensor**(*addr*, *port*)
Bases: object

remove_user(*user_index*)

remove_all_users()

tracked_skeleton

get_skeleton()

run()

Module contents

Submodules

pypot.sensor.optibridge module

class pypot.sensor.optibridge.**OptiBridgeServer**(*bridge_host*, *bridge_port*, *opti_addr*, *opti_port*, *obj_name*)
Bases: threading.Thread

run()

class pypot.sensor.optibridge.**OptiTrackClient**(*bridge_host*, *bridge_port*, *obj_name*)
Bases: threading.Thread

run()

tracked_objects

recent_tracked_objects

pypot.sensor.optitrack module

class pypot.sensor.optitrack.**TrackedObject**(*position*, *quaternion*, *orientation*, *timestamp*)

Bases: tuple

orientation
Alias for field number 2

position
Alias for field number 0

quaternion
Alias for field number 1

timestamp
Alias for field number 3

pypot.sensor.optitrack.**quat2euler**(*q*)

Module contents

pypot.server package

Submodules

pypot.server.httpserver module

class pypot.server.httpserver.**MyJSONEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Bases: json.encoder.JSONEncoder

JSONEncoder which tries to call a json property before using the enconding default function.

Constructor for JSONEncoder, with sensible defaults.

If skipkeys is false, then it is a TypeError to attempt encoding of keys that are not str, int, long, float or None. If skipkeys is True, such items are simply skipped.

If *ensure_ascii* is true (the default), all non-ASCII characters in the output are escaped with uXXXX sequences, and the results are str instances consisting of ASCII characters only. If ensure_ascii is False, a result may be a unicode instance. This usually happens if the input contains unicode strings or the *encoding* parameter is used.

If check_circular is true, then lists, dicts, and custom encoded objects will be checked for circular references during encoding to prevent an infinite recursion (which would cause an OverflowError). Otherwise, no such check takes place.

If allow_nan is true, then NaN, Infinity, and -Infinity will be encoded as such. This behavior is not JSON specification compliant, but is consistent with most JavaScript based encoders and decoders. Otherwise, it will be a ValueError to encode such floats.

If sort_keys is true, then the output of dictionaries will be sorted by key; this is useful for regression tests to ensure that JSON serializations can be compared on a day-to-day basis.

If indent is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0 will only insert newlines. None is the most compact representation. Since the default item separator is ‘, ‘, the output might include trailing whitespace when indent is specified. You can use separators=(‘,’, ‘: ‘) to avoid this.

If specified, separators should be a (item_separator, key_separator) tuple. The default is (', ', ': '). To get the most compact JSON representation you should specify (',', ':') to eliminate whitespace.

If specified, default is a function that gets called for objects that can't otherwise be serialized. It should return a JSON encodable version of the object or raise a `TypeError`.

If encoding is not None, then all input strings will be transformed into unicode using that encoding prior to JSON-encoding. The default is UTF-8.

default(*obj*)

class pypot.server.httpserver.**EnableCors**(*origin='*'*)

Bases: object

Enable CORS (Cross-Origin Resource Sharing) headers

name = 'enable_cors'

api = 2

apply(*fn*, *context*)

class pypot.server.httpserver.**HTTPRobotServer**(*robot*, *host*, *port*, *cross_domain_origin='*'*, *quiet=True*)

Bases: *pypot.server.server.AbstractServer*

Bottle based HTTPServer used to remote access a robot.

Please refer to the REST API for an exhaustive list of the possible routes.

run(*quiet=None*, *server='tornado'*)

Start the bottle server, run forever.

pypot.server.rest module

class pypot.server.rest.**RESTRobot**(*robot*)

Bases: object

REST API for a Robot.

Through the REST API you can currently access:

- the motors list (and the aliases)
- the registers list for a specific motor
- read/write a value from/to a register of a specific motor
- the sensors list
- the registers list for a specific motor
- read/write a value from/to a register of a specific motor
- the primitives list (and the active)
- start/stop primitives

get_motors_list(*alias='motors'*)

get_motor_registers_list(*motor*)

get_registers_list(*motor*)

get_motor_register_value(*motor*, *register*)

get_register_value(*motor*, *register*)

set_motor_register_value(*motor*, *register*, *value*)

set_register_value(*motor*, *register*, *value*)

get_motors_alias()

set_goto_position_for_motor(*motor*, *position*, *duration*)

get_sensors_list()

get_sensors_registers_list(*sensor*)

get_sensor_register_value(*sensor*, *register*)

set_sensor_register_value(*sensor*, *register*, *value*)

get_primitives_list()

get_running_primitives_list()

start_primitive(*primitive*)

stop_primitive(*primitive*)

pause_primitive(*primitive*)

resume_primitive(*primitive*)

get_primitive_properties_list(*primitive*)

get_primitive_property(*primitive*, *property*)

set_primitive_property(*primitive*, *property*, *value*)

get_primitive_methods_list(*primitive*)

call_primitive_method(*primitive*, *method*, *kwargs*)

start_move_recorder(*move_name*, *motors_name=None*)

attach_move_recorder(*move_name*, *motors_name*)

get_move_recorder_motors(*move_name*)

stop_move_recorder(*move_name*)
Allow more easily than stop_primitive() to save in a filename the recorcded move

start_move_player(*move_name*, *speed=1.0*, *backwards=False*)
Move player need to have a move file <move_name.record> in the working directory to play it

get_available_record_list()
Get list of json recorded movement files

remove_move_record(*move_name*)
Remove the json recorded movement file

pypot.server.server module

class pypot.server.server.**AbstractServer**(*robot*, *host*, *port*)
Bases: object

run()

class pypot.server.server.**RemoteRobotServer**(*robot*, *host*, *port*)
Bases: *pypot.server.server.AbstractServer*

run()

pypot.server.snap module

pypot.server.snap.**get_snap_user_projects_directory**()
pypot.server.snap.**find_local_ip**()

pypot.server.snap.**set_snap_server_variables**(*host*, *port*, *snap_extension='.xml'*, *path=None*)

Allow to change dynamically port and host variable in xml Snap! project file

class pypot.server.snap.**SnapRobotServer**(*robot*, *host*, *port*, *quiet=True*)

Bases: *pypot.server.server.AbstractServer*

run()

pypot.server.zmqserver module

class pypot.server.zmqserver.**ZMQRobotServer**(*robot*, *host*, *port*)

Bases: *pypot.server.server.AbstractServer*

A ZMQServer allowing remote access of a robot instance.

The server used the REQ/REP zmq pattern. You should always first send a request and then read the answer.

run()

Run an infinite REQ/REP loop.

handle_request(*request*)

Module contents

pypot.tools package

Subpackages

pypot.tools.herborist package

Submodules

pypot.tools.herborist.herborist module

pypot.tools.herborist.herborist.**get_dxl_connection**(*port*, *baudrate*, *protocol='MX'*)
pypot.tools.herborist.herborist.**release_dxl_connection**()

class pypot.tools.herborist.herborist.**HerboristApp**(*argv*)

Bases: PyQt4.QtGui.QApplication

class UpdatePortThread

Bases: PyQt4.QtCore.QThread

port_updated

run()

HerboristApp.**update_port**(*new_ports*)

HerboristApp.**update_motor_tree**(*baud_for_ids*)

HerboristApp.**start_scanning**()

HerboristApp.**abort_scanning**()

HerboristApp.**done_scanning**()

class HerboristApp.**ScanThread**(*port*, *baudrates*, *protocol*, *id_range*, *motor_tree*, *scan_progress*)
Bases: PyQt4.QtCore.QThread

done

part_done

run()

abort()

HerboristApp.**update_motor_view**()

HerboristApp.**update_motor_position**(*pos*)

HerboristApp.**motor_position_updated**(*pos*)

HerboristApp.**switch_torque**(*torque_enable*)

HerboristApp.**enable_motor_view**(*enabled*)

HerboristApp.**update_eeprom**()

class HerboristApp.**UpdateMotorThread**(*port*, *baudrate*, *protocol*, *mid*)
Bases: PyQt4.QtCore.QThread

position_updated

stop()

run()

HerboristApp.**port**

HerboristApp.**protocol**

HerboristApp.**usb_device**

HerboristApp.**baudrate**

HerboristApp.**id**

HerboristApp.**selected_motors**

HerboristApp.**ids**

pypot.tools.herborist.herborist.**main**()

Module contents

Submodules

pypot.tools.dxl_reset module Reset a dynamixel motor to "poppy" configuration.

This utility should only be used with a single motor connected to the bus. For the moment it's only working with robotis protocol v1 (AX, RX, MX motors).

To run it: $ poppy-reset-motor 42

The motor will now have the id 42, use a 1000000 baud rates, a 0µs return delay time. The angle limit are also set (by default to (-180, 180)). Its position is also set to its base position (default: 0).

For more complex use cases, see: $ poppy-reset-motor –help

pypot.tools.dxl_reset.**leave**(*msg*)

pypot.tools.dxl_reset.**almost_equal**(*a*, *b*)

pypot.tools.dxl_reset.**main**()

Module contents

pypot.utils package

Submodules

pypot.utils.appdirs module Utilities for determining application-specific dirs.

See <http://github.com/ActiveState/appdirs> for details and usage.

pypot.utils.appdirs.**user_data_dir**(*appname=None*, *appauthor=None*, *version=None*, *roaming=False*)

Return full path to the user-specific data dir for this application.

"appname" is the name of application. If None, just the system directory is returned.

"appauthor" (only used on Windows) is the name of the appauthor or distributing body for this application. Typically it is the owning company name. This falls back to appname. You may pass False to disable it.

"version" is an optional version path element to append to the path. You might want to use this if you want multiple versions of your app to be able to run independently. If used, this would typically be "<major>.<minor>". Only applied when appname is present.

"roaming" (boolean, default False) can be set True to use the Windows roaming appdata directory. That means that for users on a Windows network setup for roaming profiles, this user data will be sync'd on login. See <http://technet.microsoft.com/en-us/library/cc766489(WS.10).aspx> for a discussion of issues.

Typical user data directories are: Mac OS X: ~/Library/Application Support/<AppName> Unix: ~/.local/share/<AppName> # or in $XDG_DATA_HOME, if defined Win XP (not roaming): C:Documents and Settings<username>Application Data<AppAuthor><AppName> Win XP (roaming): C:Documents and Settings<username>Local SettingsApplication Data<AppAuthor><AppName> Win 7 (not roaming): C:Users<username>AppDataLocal<AppAuthor><AppName> Win 7 (roaming): C:Users<username>AppDataRoaming<AppAuthor><AppName>

For Unix, we follow the XDG spec and support $XDG_DATA_HOME. That means, by default "~/.local/share/<AppName>".

pypot.utils.appdirs.**site_data_dir**(*appname=None*, *appauthor=None*, *version=None*, *multipath=False*)

Return full path to the user-shared data dir for this application.

"appname" is the name of application. If None, just the system directory is returned.

"appauthor" (only used on Windows) is the name of the appauthor or distributing body for this application. Typically it is the owning company name. This falls back to appname. You may pass False to disable it.

"version" is an optional version path element to append to the path. You might want to use this if you want multiple versions of your app to be able to run independently. If used, this would typically be "<major>.<minor>". Only applied when appname is present.

"multipath" is an optional parameter only applicable to *nix which indicates that the entire list of data dirs should be returned. By default, the first item from XDG_DATA_DIRS is returned, or '/usr/local/share/<AppName>', if XDG_DATA_DIRS is not set

Typical user data directories are: Mac OS X: /Library/Application Support/<AppName> Unix: /usr/local/share/<AppName> or /usr/share/<AppName> Win XP: C:Documents and SettingsAll UsersApplication Data<AppAuthor><AppName> Vista: (Fail! "C:ProgramData" is a hidden *system* directory on Vista.) Win 7: C:ProgramData<AppAuthor><AppName> # Hidden, but writeable on Win 7.

For Unix, this is using the $XDG_DATA_DIRS[0] default.

WARNING: Do not use this on Windows. See the Vista-Fail note above for why.

pypot.utils.appdirs.**user_config_dir**(*appname=None*, *appauthor=None*, *version=None*, *roaming=False*)

Return full path to the user-specific config dir for this application.

"appname" is the name of application. If None, just the system directory is returned.

"appauthor" (only used on Windows) is the name of the appauthor or distributing body for this application. Typically it is the owning company name. This falls back to appname. You may pass False to disable it.

"version" is an optional version path element to append to the path. You might want to use this if you want multiple versions of your app to be able to run independently. If used, this would typically be "<major>.<minor>". Only applied when appname is present.

"roaming" (boolean, default False) can be set True to use the Windows roaming appdata directory. That means that for users on a Windows network setup for roaming profiles, this user data will be sync'd on login. See <http://technet.microsoft.com/en-us/library/cc766489(WS.10).aspx> for a discussion of issues.

Typical user data directories are: Mac OS X: same as user_data_dir Unix: ~/.config/<AppName> # or in $XDG_CONFIG_HOME, if defined Win *: same as user_data_dir

For Unix, we follow the XDG spec and support $XDG_CONFIG_HOME. That means, by deafult "~/.config/<AppName>".

pypot.utils.appdirs.**site_config_dir**(*appname=None*, *appauthor=None*, *version=None*, *multipath=False*)

Return full path to the user-shared data dir for this application.

"appname" is the name of application. If None, just the system directory is returned.

"appauthor" (only used on Windows) is the name of the appauthor or distributing body for this application. Typically it is the owning company name. This falls back to appname. You may pass False to disable it.

"version" is an optional version path element to append to the path. You might want to use this if you want multiple versions of your app to be able to run independently. If used, this would typically be "<major>.<minor>". Only applied when appname is present.

"multipath" is an optional parameter only applicable to *nix which indicates that the entire list of config dirs should be returned. By default, the first item from XDG_CONFIG_DIRS is returned, or '/etc/xdg/<AppName>', if XDG_CONFIG_DIRS is not set

Typical user data directories are: Mac OS X: same as site_data_dir Unix: /etc/xdg/<AppName> or $XDG_CONFIG_DIRS[i]/<AppName> for each value in

$XDG_CONFIG_DIRS

Win *: same as site_data_dir Vista: (Fail! "C:ProgramData" is a hidden *system* directory on Vista.)

For Unix, this is using the $XDG_CONFIG_DIRS[0] default, if multipath=False

WARNING: Do not use this on Windows. See the Vista-Fail note above for why.

pypot.utils.appdirs.**user_cache_dir**(*appname=None*, *appauthor=None*, *version=None*, *opinion=True*)

Return full path to the user-specific cache dir for this application.

"appname" is the name of application. If None, just the system directory is returned.

"appauthor" (only used on Windows) is the name of the appauthor or distributing body for this application. Typically it is the owning company name. This falls back to appname. You may pass False to disable it.

"version" is an optional version path element to append to the path. You might want to use this if you want multiple versions of your app to be able to run independently. If used, this would typically be "<major>.<minor>". Only applied when appname is present.

"opinion" (boolean) can be False to disable the appending of "Cache" to the base app data dir for Windows. See discussion below.

Typical user cache directories are: Mac OS X: ~/Library/Caches/<AppName> Unix: ~/.cache/<AppName> (XDG default) Win XP: C:Documents and Settings<username>Local SettingsApplication Data<AppAuthor><AppName>Cache Vista: C:Users<username>AppDataLocal<AppAuthor><AppName>Cache

On Windows the only suggestion in the MSDN docs is that local settings go in the *CSIDL_LOCAL_APPDATA* directory. This is identical to the non-roaming app data dir (the default returned by *user_data_dir* above). Apps typically put cache data somewhere *under* the given dir here. Some examples:

...MozillaFirefoxProfiles<ProfileName>Cache ...AcmeSuperAppCache1.0

OPINION: This function appends "Cache" to the *CSIDL_LOCAL_APPDATA* value. This can be disabled with the *opinion=False* option.

pypot.utils.appdirs.**user_log_dir**(*appname=None*, *appauthor=None*, *version=None*, *opinion=True*)

Return full path to the user-specific log dir for this application.

"appname" is the name of application. If None, just the system directory is returned.

"appauthor" (only used on Windows) is the name of the appauthor or distributing body for this application. Typically it is the owning company name. This falls back to appname. You may pass False to disable it.

"version" is an optional version path element to append to the path. You might want to use this if you want multiple versions of your app to be able to run independently. If used, this would typically be "<major>.<minor>". Only applied when appname is present.

"opinion" (boolean) can be False to disable the appending of "Logs" to the base app data dir for Windows, and "log" to the base cache dir for Unix. See discussion below.

Typical user cache directories are: Mac OS X: ~/Library/Logs/<AppName> Unix: ~/.cache/<AppName>/log # or under $XDG_CACHE_HOME if defined Win XP: C:Documents and Settings<username>Local SettingsApplication Data<AppAuthor><AppName>Logs Vista: C:Users<username>AppDataLocal<AppAuthor><AppName>Logs

On Windows the only suggestion in the MSDN docs is that local settings go in the *CSIDL_LOCAL_APPDATA* directory. (Note: I'm interested in examples of what some windows apps use for a logs dir.)

OPINION: This function appends "Logs" to the *CSIDL_LOCAL_APPDATA* value for Windows and appends "log" to the user cache dir for Unix. This can be disabled with the *opinion=False* option.

class pypot.utils.appdirs.**AppDirs**(*appname*, *appauthor=None*, *version=None*, *roaming=False*, *multipath=False*)

Bases: object

Convenience wrapper for getting application dirs.

user_data_dir

site_data_dir

user_config_dir

site_config_dir

user_cache_dir

user_log_dir

pypot.utils.interpolation module

class pypot.utils.interpolation.**KDTreeDict**(*gen_tree_on_add=False*, *distance_upper_bound=0.2*, *k_neighbors=2*)

Bases: dict

update(**args*, ***kwargs*)

generate_tree()

nearest_keys(*key*)
Find the nearest_keys (l2 distance) thanks to a cKDTree query

interpolate_motor_positions(*input_key*, *nearest_keys*)
Process linear interpolation to estimate actual speed and position of motors Method specific to the :meth:~pypot.primitive.move.Move.position() structure it is a KDTreeDict[timestamp] = {dict[motor]=(position,speed)}

pypot.utils.pypot_time module

pypot.utils.pypot_time.**time**()
pypot.utils.pypot_time.**sleep**(*t*)

pypot.utils.stoppablethread module

class pypot.utils.stoppablethread.**StoppableThread**(*setup=None*, *target=None*, *teardown=None*)

Bases: object

Stoppable version of python Thread.

This class provides the following mechanism on top of "classical" python Thread:

- you can stop the thread (if you defined your run method accordingly).
- you can restart a thread (stop it and re-run it)
- you can pause/resume a thread

> **Warning:** It is up to the subclass to correctly respond to the stop, pause/resume signals (see `run()` for details).

Parameters

- **setup** (*func*) – specific setup function to use (otherwise self.setup)
- **target** (*func*) – specific target function to use (otherwise self.run)
- **teardown** (*func*) – specific teardown function to use (otherwise self.teardown)

start()
Start the run method as a new thread.

It will first stop the thread if it is already running.

stop(*wait=True*)
Stop the thread.

More precisely, sends the stopping signal to the thread. It is then up top the run method to correctly responds.

join()
Wait for the thread termination.

running
Whether the thread is running.

started
Whether the thread has been started.

wait_to_start()
Wait for the thread to actually starts.

should_stop()
Signals if the thread should be stopped or not.

wait_to_stop()
Wait for the thread to terminate.

setup()
Setup method call just before the run.

run()
Run method of the thread.

Note: In order to be stoppable (resp. pausable), this method has to check the running property - as often as possible to improve responsivness - and terminate when `should_stop()` (resp. `should_pause()`) becomes True. For instance:

```
while self.should_stop():
    do_atom_work()
    ...
```

---

teardown()
Teardown method call just after the run.

should_pause()
Signals if the thread should be paused or not.

paused

pause()
Requests the thread to pause.

resume()
Requests the thread to resume.

wait_to_resume()
Waits until the thread is resumed.

pypot.utils.stoppablethread.**make_update_loop**(*thread*, *update_func*)
Makes a run loop which calls an update function at a predefined frequency.

class pypot.utils.stoppablethread.**StoppableLoopThread**(*frequency*, *update=None*)
Bases: pypot.utils.stoppablethread.StoppableThread

LoopThread calling an update method at a pre-defined frequency.

Note: This class does not mean to be accurate. The given frequency will be approximately followed - depending for instance on CPU load - and only reached if the update method takes less time than the chosen loop period.

Params float frequency called frequency of the `update()` method

run()
Called the update method at the pre-defined frequency.

update()
Update method called at the pre-defined frequency.

pypot.utils.trajectory module

class pypot.utils.trajectory.**MinimumJerkTrajectory**(*initial*, *final*, *duration*, *init_vel=0.0*, *init_acc=0.0*, *final_vel=0.0*, *final_acc=0.0*)
Bases: object

compute()

get_value(*t*)

domain(*x*)

test_domain(*x*)

fix_input(*x*)

get_generator()

class pypot.utils.trajectory.**GotoMinJerk**(*motor*, *position*, *duration*, *frequency=50*)
Bases: pypot.utils.stoppablethread.StoppableLoopThread

setup()

update()

elapsed_time

Module contents

class pypot.utils.**Point2D**(*x*, *y*)
Bases: tuple

x
Alias for field number 0

y
Alias for field number 1

class pypot.utils.**Point3D**(*x*, *y*, *z*)
Bases: tuple

x
Alias for field number 0

y
Alias for field number 1

z
Alias for field number 2

pypot.utils.**Point**
alias of Point3D

class pypot.utils.**Vector3D**(*x*, *y*, *z*)
Bases: tuple

x
Alias for field number 0

y
Alias for field number 1

z
Alias for field number 2

pypot.utils.**Vector**
alias of Vector3D

class pypot.utils.**Quaternion**(*x*, *y*, *z*, *w*)
Bases: tuple

w
Alias for field number 3

x
Alias for field number 0

y
Alias for field number 1

z
Alias for field number 2

pypot.utils.**attrsetter**(*item*)

class pypot.utils.**SyncEvent**(*period=0.1*)
Bases: object

request()

done()

is_recent

needed

pypot.vrep package

Subpackages

pypot.vrep.remoteApiBindings package

Submodules

pypot.vrep.remoteApiBindings.vrep module

pypot.vrep.remoteApiBindings.vrep.**tbs**(*str*)

pypot.vrep.remoteApiBindings.vrep.**py3compatible**(*f*)

pypot.vrep.remoteApiBindings.vrep.**simxGetJointPosition**(*clientID*, *jointHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetJointPosition**(*clientID*, *jointHandle*, *position*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetJointMatrix**(*clientID*, *jointHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetSphericalJointMatrix**(*clientID*, *jointHandle*, *matrix*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetJointTargetVelocity**(*clientID*, *jointHandle*, *targetVelocity*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetJointTargetPosition**(*clientID*, *jointHandle*, *targetPosition*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxJointGetForce**(*clientID*, *jointHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetJointForce**(*clientID*, *jointHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetJointForce**(*clientID*, *jointHandle*, *force*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxReadForceSensor**(*clientID*, *forceSensorHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxBreakForceSensor**(*clientID*, *forceSensorHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxReadVisionSensor**(*clientID*, *sensorHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectHandle**(*clientID*, *objectName*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetVisionSensorImage**(*clientID*, *sensorHandle*, *options*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetVisionSensorImage**(*clientID*, *sensorHandle*, *image*, *options*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetVisionSensorDepthBuffer**(*clientID*, *sensorHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectChild**(*clientID*, *parentObjectHandle*, *childIndex*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectParent**(*clientID*, *childObjectHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxReadProximitySensor**(*clientID*, *sensorHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxLoadModel**(*clientID*, *modelPathAndName*, *options*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxLoadUI**(*clientID*, *uiPathAndName*, *options*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxLoadScene**(*clientID*, *scenePathAndName*, *options*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxStartSimulation**(*clientID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxPauseSimulation**(*clientID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxStopSimulation**(*clientID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetUIHandle**(*clientID*, *uiName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetUISlider**(*clientID*, *uiHandle*, *uiButtonID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetUISlider**(*clientID*, *uiHandle*, *uiButtonID*, *position*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetUIEventButton**(*clientID*, *uiHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetUIButtonProperty**(*clientID*, *uiHandle*, *uiButtonID*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetUIButtonProperty**(*clientID*, *uiHandle*, *uiButtonID*, *prop*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxAddStatusbarMessage**(*clientID*, *message*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxAuxiliaryConsoleOpen**(*clientID*, *title*, *maxLines*, *mode*, *position*, *size*, *textColor*, *backgroundColor*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxAuxiliaryConsoleClose**(*clientID*, *consoleHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxAuxiliaryConsolePrint**(*clientID*, *consoleHandle*, *txt*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxAuxiliaryConsoleShow**(*clientID*, *consoleHandle*, *showState*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectOrientation**(*clientID*, *objectHandle*, *relativeToObjectHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectPosition**(*clientID*, *objectHandle*, *relativeToObjectHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetObjectOrientation**(*clientID*, *objectHandle*, *relativeToObjectHandle*, *eulerAngles*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetObjectPosition**(*clientID*, *objectHandle*, *relativeToObjectHandle*, *position*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetObjectParent**(*clientID*, *objectHandle*, *parentObject*, *keepInPlace*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetUIButtonLabel**(*clientID*, *uiHandle*, *uiButtonID*, *upStateLabel*, *downStateLabel*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetLastErrors**(*clientID*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetArrayParameter**(*clientID*, *paramIdentifier*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetArrayParameter**(*clientID*, *paramIdentifier*, *paramValues*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetBooleanParameter**(*clientID*, *paramIdentifier*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetBooleanParameter**(*clientID*, *paramIdentifier*, *paramValue*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetIntegerParameter**(*clientID*, *paramIdentifier*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetIntegerParameter**(*clientID*, *paramIdentifier*, *paramValue*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetFloatingParameter**(*clientID*, *paramIdentifier*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetFloatingParameter**(*clientID*, *paramIdentifier*, *paramValue*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetStringParameter**(*clientID*, *paramIdentifier*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetCollisionHandle**(*clientID*, *collisionObjectName*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetDistanceHandle**(*clientID*, *distanceObjectName*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxReadCollision**(*clientID*, *collisionObjectHandle*, *operationMode*)

Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxReadDistance**(*clientID*, *distanceObjectHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxRemoveObject**(*clientID*, *objectHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxRemoveModel**(*clientID*, *objectHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxRemoveUI**(*clientID*, *uiHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxCloseScene**(*clientID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjects**(*clientID*, *objectType*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxDisplayDialog**(*clientID*, *titleText*, *mainText*, *dialogType*, *initialText*, *titleColors*, *dialogColors*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxEndDialog**(*clientID*, *dialogHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetDialogInput**(*clientID*, *dialogHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetDialogResult**(*clientID*, *dialogHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxCopyPasteObjects**(*clientID*, *objectHandles*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectSelection**(*clientID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetObjectSelection**(*clientID*, *objectHandles*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxClearFloatSignal**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxClearIntegerSignal**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxClearStringSignal**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetFloatSignal**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetIntegerSignal**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetStringSignal**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetAndClearStringSignal**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxReadStringStream**(*clientID*, *signalName*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetFloatSignal**(*clientID*, *signalName*, *signalValue*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetIntegerSignal**(*clientID*, *signalName*, *signalValue*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetStringSignal**(*clientID*, *signalName*, *signalValue*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxAppendStringSignal**(*clientID*, *signalName*, *signalValue*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxWriteStringStream**(*clientID*, *signalName*, *signalValue*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectFloatParameter**(*clientID*, *objectHandle*, *parameterID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetObjectFloatParameter**(*clientID*, *objectHandle*, *parameterID*, *parameterValue*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectIntParameter**(*clientID*, *objectHandle*, *parameterID*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetObjectIntParameter**(*clientID*, *objectHandle*, *parameterID*, *parameterValue*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetModelProperty**(*clientID*, *objectHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSetModelProperty**(*clientID*, *objectHandle*, *prop*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxStart**(*connectionAddress*, *connectionPort*, *waitUntilConnected*, *doNotReconnectOnceDisconnected*, *timeOutInMs*, *commThreadCycleInMs*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxFinish**(*clientID*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetPingTime**(*clientID*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetLastCmdTime**(*clientID*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSynchronousTrigger**(*clientID*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxSynchronous**(*clientID*, *enable*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxPauseCommunication**(*clientID*, *enable*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetInMessageInfo**(*clientID*, *infoType*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetOutMessageInfo**(*clientID*, *infoType*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetConnectionId**(*clientID*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxCreateBuffer**(*bufferSize*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxReleaseBuffer**(*buffer*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxTransferFile**(*clientID*, *filePathAndName*, *fileName_serverSide*, *timeOut*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxEraseFile**(*clientID*, *fileName_serverSide*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxCreateDummy**(*clientID*, *size*, *color*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxQuery**(*clientID*, *signalName*, *signalValue*, *retSignalName*, *timeOutInMs*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectGroupData**(*clientID*, *objectType*, *dataType*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxGetObjectVelocity**(*clientID*, *objectHandle*, *operationMode*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxPackInts**(*intList*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxUnpackInts**(*intsPackedInString*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxPackFloats**(*floatList*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrep.**simxUnpackFloats**(*floatsPackedInString*)
Please have a look at the function description/documentation in the V-REP user manual

pypot.vrep.remoteApiBindings.vrepConst module

Module contents

Submodules

pypot.vrep.controller module

class pypot.vrep.controller.**VrepController**(*vrep_io*, *scene*, *motors*, *sync_freq=50.0*)
Bases: pypot.robot.controller.MotorsController

V-REP motors controller.

Parameters

- **vrep_io** (VrepIO) – vrep io instance
- **scene** (*str*) – path to the V-REP scene file to start
- **motors** (*list*) – list of motors attached to the controller
- **sync_freq** (*float*) – synchronization frequency

setup()
Setups the controller by reading/setting position for all motors.

update()
Synchronization update loop.

At each update all motor position are read from vrep and set to the motors. The motors target position are also send to v-rep.

class pypot.vrep.controller.**VrepObjectTracker**(*io*, *sensors*, *sync_freq=50.0*)
Bases: pypot.robot.controller.SensorsController

Tracks the 3D position and orientation of a V-REP object.

Parameters

- **io** (AbstractIO) – IO used to communicate with the hardware motors
- **sensors** (*list*) – list of sensors attached to the controller
- **sync_freq** (*float*) – synchronization frequency

setup()
Forces a first update to trigger V-REP streaming.

update()
Updates the position and orientation of the tracked objects.

class pypot.vrep.controller.**VrepCollisionDetector**(*name*)
Bases: pypot.robot.sensor.Sensor

colliding

class pypot.vrep.controller.**VrepCollisionTracker**(*io*, *sensors*, *sync_freq=50.0*)
Bases: pypot.robot.controller.SensorsController

Tracks collision state.

Parameters

- **io** (AbstractIO) – IO used to communicate with the hardware motors
- **sensors** (*list*) – list of sensors attached to the controller
- **sync_freq** (*float*) – synchronization frequency

setup()
Forces a first update to trigger V-REP streaming.

update()
Update the state of the collision detectors.

pypot.vrep.io module

class pypot.vrep.io.**VrepIO**(*vrep_host='127.0.0.1'*, *vrep_port=19997*, *scene=None*, *start=False*)
Bases: pypot.robot.io.AbstractIO

This class is used to get/set values from/to a V-REP scene.

It is based on V-REP remote API (http://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm).

Starts the connection with the V-REP remote API server.

Parameters

- **vrep_host** (*str*) – V-REP remote API server host
- **vrep_port** (*int*) – V-REP remote API server port
- **scene** (*str*) – path to a V-REP scene file
- **start** (*bool*) – whether to start the scene after loading it

Warning: Only one connection can be established with the V-REP remote server API. So before trying to connect make sure that all previously started connections have been closed (see `close_all_connections()`)

MAX_ITER = 5

TIMEOUT = 0.4

open_io()

close()
Closes the current connection.

load_scene(*scene_path*, *start=False*)
Loads a scene on the V-REP server.

Parameters

- **scene_path** (*str*) – path to a V-REP scene file
- **start** (*bool*) – whether to directly start the simulation after loading the scene

Note: It is assumed that the scene file is always available on the server side.

start_simulation()
Starts the simulation.

Note: Do nothing if the simulation is already started.

Warning: if you start the simulation just after stopping it, the simulation will likely not be started. Use `restart_simulation()` instead.

restart_simulation()
Re-starts the simulation.

stop_simulation()
Stops the simulation.

pause_simulation()
Pauses the simulation.

resume_simulation()
Resumes the simulation.

get_motor_position(*motor_name*)
Gets the motor current position.

set_motor_position(*motor_name*, *position*)
Sets the motor target position.

get_motor_force(*motor_name*)

set_motor_force(*motor_name*, *force*)

get_object_position(*object_name*, *relative_to_object=None*)
Gets the object position.

set_object_position(*object_name, position=[0, 0, 0]*)
Sets the object position.

get_object_orientation(*object_name*, *relative_to_object=None*)
Gets the object orientation.

get_object_handle(*obj*)
Gets the vrep object handle.

get_collision_state(*collision_name*)
Gets the collision state.

get_collision_handle(*collision*)
Gets a vrep collisions handle.

get_simulation_current_time(*timer='CurrentTime'*)
Gets the simulation current time.

add_cube(*name*, *position*, *sizes*, *mass*)
Add Cube

add_sphere(*name, position, sizes, mass, precision=[10, 10]*)
Add Sphere

add_cylinder(*name, position, sizes, mass, precision=[10, 10]*)
Add Cylinder

add_cone(*name, position, sizes, mass, precision=[10, 10]*)
Add Cone

change_object_name(*old_name*, *new_name*)
Change object name

call_remote_api(*func_name*, **args*, ***kwargs*)
Calls any remote API func in a thread_safe way.

Parameters

- **func_name** (*str*) – name of the remote API func to call
- **args** – args to pass to the remote API call
- **kwargs** – args to pass to the remote API call

Note: You can add an extra keyword to specify if you want to use the streaming or sending mode. The oneshot_wait mode is used by default (see here for details about possible modes).

Warning: You should not pass the clientId and the operationMode as arguments. They will be automatically added.

As an example you can retrieve all joints name using the following call:

```
vrep_io.remote_api_call('simxGetObjectGroupData',
                        vrep_io.remote_api.sim_object_joint_type,
                        0,
                        streaming=True)
```

pypot.vrep.io.**close_all_connections**()
Closes all opened connection to V-REP remote API server.

exception pypot.vrep.io.**VrepIOError**(*error_code*, *message*)
Bases: exceptions.Exception

Base class for V-REP IO Errors.

exception pypot.vrep.io.**VrepIOErrors**
Bases: exceptions.Exception

exception pypot.vrep.io.**VrepConnectionError**
Bases: exceptions.Exception

Base class for V-REP connection Errors.

Module contents
class pypot.vrep.**vrep_time**(*vrep_io*)

get_time()

sleep(*t*)

pypot.vrep.**from_vrep**(*config*, *vrep_host='127.0.0.1'*, *vrep_port=19997*, *scene=None*, *tracked_objects=[]*, *tracked_collisions=[]*)
Create a robot from a V-REP instance.

Parameters

- **config** (*str or dict*) – robot configuration (either the path to the json or directly the dictionary)
- **vrep_host** (*str*) – host of the V-REP server
- **vrep_port** (*int*) – port of the V-REP server
- **scene** (*str*) – path to the V-REP scene to load and start
- **tracked_objects** (*list*) – list of V-REP dummy object to track
- **tracked_collisions** (*list*) – list of V-REP collision to track

This function tries to connect to a V-REP instance and expects to find motors with names corresponding as the ones found in the config.

Note: The Robot returned will also provide a convenience reset_simulation method which resets the simulation and the robot position to its intial stance.

Note: Using the same configuration, you should be able to switch from a real to a simulated robot just by switching from from_config() to from_vrep(). For instance:

```
import json

with open('my_config.json') as f:
    config = json.load(f)

from pypot.robot import from_config
from pypot.vrep import from_vrep

real_robot = from_config(config)
simulated_robot = from_vrep(config, '127.0.0.1', 19997, 'poppy.ttt')
```

Submodules

pypot.kinematics module

class pypot.kinematics.**Link**
Bases: *pypot.kinematics.Link*

Link object as defined by the standard DH representation.

This representation is based on the following information: :param float theta: angle about previous z from old x to new x :param float d: offset along previous z to the common normal :param float a: offset along previous to the common normal :param float alpha: angle about common normal, from old z axis to new z axis

---

Note: We are only considering revolute joint.

Please refer to http://en.wikipedia.org/wiki/Denavit-Hartenberg_parameters for more details.

get_transformation_matrix(*theta*)
Computes the homogeneous transformation matrix for this link.

class pypot.kinematics.**Chain**
Bases: *pypot.kinematics.Chain*

Chain of Link that can be used to perform forward and inverse kinematics.

Parameters

- **links** (*list*) – list of Link that compose the chain
- **base** – the base homogeneous transformation matrix
- **tool** – the end tool homogeneous transformation matrix

forward_kinematics(*q*)
Computes the homogeneous transformation matrix of the end effector of the chain.

Parameters **q** (*vector*) – vector of the joint angles (theta 1, theta 2, ..., theta n)

inverse_kinematics(*end_effector_transformation*, *q=None*, *max_iter=1000*, *tolerance=0.05*, *mask=array([1., 1., 1., 1., 1., 1.])*, *use_pinv=False*)
Computes the joint angles corresponding to the end effector transformation.

Parameters

- **end_effector_transformation** – the end effector homogeneous transformation matrix
- **q** (*vector*) – initial estimate of the joint angles
- **max_iter** (*int*) – maximum number of iteration
- **tolerance** (*float*) – tolerance before convergence
- **mask** – specify the cartesian DOF that will be ignore (in the case of a chain with less than 6 joints).

Return type vector of the joint angles (theta 1, theta 2, ..., theta n)

pypot.kinematics.**transform_difference**(*t1*, *t2*)

pypot.kinematics.**rotation_from_transf**(*tm*)

pypot.kinematics.**translation_from_transf**(*tm*)

pypot.kinematics.**components_from_transf**(*tm*)

pypot.kinematics.**transf_from_components**(*R*, *T*)

pypot.kinematics.**transl**(*x*, *y*, *z*)

pypot.kinematics.**trotx**(*theta*)

pypot.kinematics.**troty**(*theta*)

pypot.kinematics.**trotz**(*theta*)

Module contents

p

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

T

U

V

W

X

Y

Z