robomasterpy

Release 0.1.0

LI Zhennan

Jul 03, 2020

CONTENTS

1 User Guide **3**
- 1.1 Installation of robomasterpy . . . 3
- 1.2 Quick Start . . . 3
- 1.3 API . . . 3

2 Health and Safety Notice **11**

3 Paperwork **13**

4 Acknowledgement **15**

5 License **17**

Index **19**

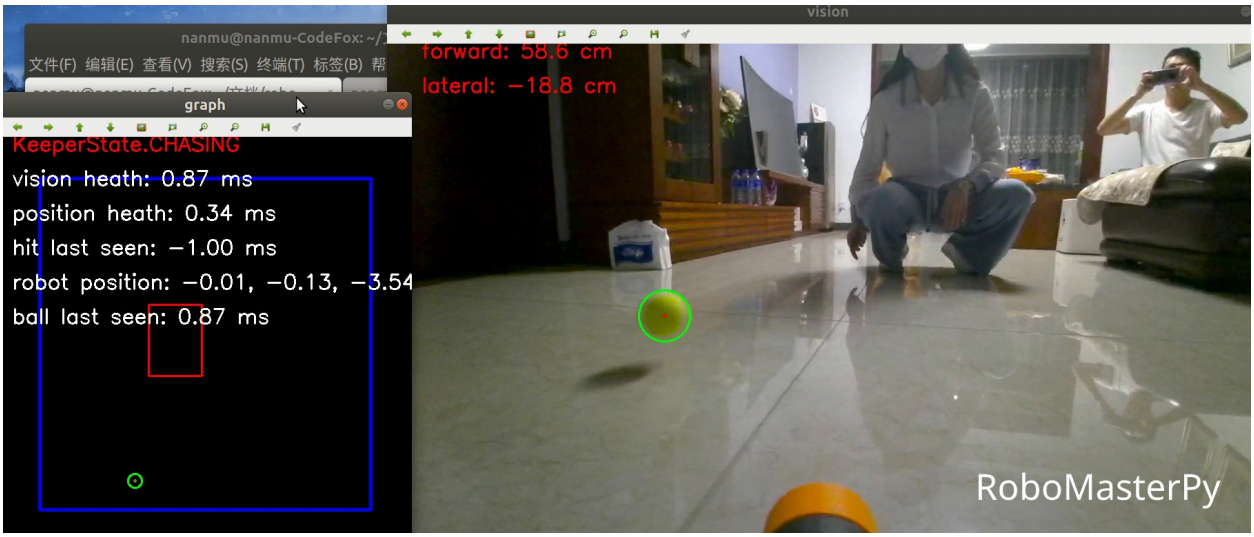
Release v0.1.0. (*Installation*)

RoboMasterPy is a Python library and framework for RoboMaster EP.

The library(client) is straightforward to use:

```
>>> import robomasterpy as rm

# IP of RoboMaster is detected under router mode
>>> cmd = rm.Commander()

# check RoboMaster's API version
>>> cmd.version()
'version 00.00.00.60'

>>> cmd.get_robot_mode()
'chassis_lead'

# ensure your Robomaster has enough room to move
>>> cmd.chassis_move(x=-1, z=30)
'ok'

# activate video streaming,
# which can be handled by the framework.
>>> cmd.stream(rm.SWITCH_ON)
'ok'

# activate gimbal attitude push at 5Hz,
# which can be handled by the framework.
>>> cmd.gimbal_push_on(attitude_freq=5)
'ok'

# Watch out!
>>> cmd.blaster_fire()
'ok'
```

The framework deals with video streaming, push and event, provides a high-level interface for controlling and communication. You can build your controlling logic basing on it, for example:



- Drive your robomaster using keyboard;
- Make your robomaster a goalkeeper.

CHAPTER

ONE

USER GUIDE

1.1 Installation of robomasterpy

RoboMasterPy requires Python 3.6 and above.

```
pip install robomasterpy
```

If you are using Python 3.6.x, you need to install `dataclasses`:

```
pip install dataclasses
```

1.2 Quick Start

TODO

1.3 API

1.3.1 Receive IP Broadcast

Robomaster broadcast its IP address under router mode.

`robomasterpy.`**`get_broadcast_ip`**(*timeout: float = None*) → str

IP

Parameters **timeout** –

Returns IP

1.3.2 Commander

Commander is a client for Robomaster TCP API.

class `robomasterpy.`**`Commander`**(*ip: str = ''*, *timeout: float = 30*)

armor_event(*attr: str*, *switch: bool*) → str

Parameters

- **attr** – ARMOR_ENUMS

- **switch** –

Returns okraise

armor_sensitivity(*value: int*) → str

Parameters

value – 5.

Returns okraise

audio(*switch: bool*) → str

Parameters

switch – /

Returns okraise

blaster_fire() → str

Returns

okraise

chassis_move(*x: float = 0*, *y: float = 0*, *z: float = 0*, *speed_xy: float = None*, *speed_z: float = None*) → str

Parameters

- **x** – x m
- **y** – y m
- **z** – z °
- **speed_xy** – xy m/s
- **speed_z** – z °/s

Return ok okraise

chassis_push_off(*position: bool = False*, *attitude: bool = False*, *status: bool = False*, *all: bool = False*) → str

Parameters

- **position** –
- **attitude** –
- **status** –
- **all** –

Returns okraise

chassis_push_on(*position_freq: int = None*, *attitude_freq: int = None*, *status_freq: int = None*, *all_freq: int = None*) → str

1, 5, 10, 20, 30, 50

Parameters

- **position_freq** – None
- **attitude_freq** – None
- **status_freq** – None
- **all_freq** –

Returns okraise

chassis_speed(*x: float = 0*, *y: float = 0*, *z: float = 0*) → str

Parameters

- **x** – x m/s
- **y** – y m/s
- **z** – z °/s

Returns okraise

chassis_wheel(*w1: int = 0, w2: int = 0, w3: int = 0, w4: int = 0*) → str

Parameters

- **w1** – rpm
- **w2** – rpm
- **w3** – rpm
- **w4** – rpm

Return ok okraise

close()

do(**args*) → str

Parameters **args** – list

Returns

get_armor_sensitivity() → int

:return:

get_chassis_attitude() → robomasterpy.client.ChassisAttitude

Returns pitch (°)roll (°)yaw (°)

get_chassis_position() → robomasterpy.client.ChassisPosition

()

Returns x (m)y (m)(°)

get_chassis_speed() → robomasterpy.client.ChassisSpeed

Returns x (m/s)y (m/s)z (°/s)w1 (rpm)w2 (rpm)w3 (rpm)w4 (rpm)

get_chassis_status() → robomasterpy.client.ChassisStatus

Returns ChassisStatus

get_gimbal_attitude() → robomasterpy.client.GimbalAttitude

:return: pitch (°)yaw (°)

get_ip() → str

IP

Returns IP

get_ir_sensor_distance(*id: int*) → int

ID

Parameters **id** – ID

Returns ID mm

get_robot_mode() → str

Returns enum MODE_*

gimbal_move(*pitch: float = 0, yaw: float = 0, pitch_speed: float = None, yaw_speed: float = None*) → str

Parameters

- **pitch** – pitch °
- **yaw** – yaw °
- **pitch_speed** – pitch °/s
- **yaw_speed** – yaw °/s

Returns okraise

gimbal_moveto(*pitch: float = 0, yaw: float = 0, pitch_speed: float = None, yaw_speed: float = None*) → str

Parameters

- **pitch** – pitch °
- **yaw** – yaw °
- **pitch_speed** – pitch °/s
- **yaw_speed** – yaw °/s

Returns okraise

gimbal_push_off(*attitude: bool = True*) → str

Parameters **attitude** –

Returns okraise

gimbal_push_on(*attitude_freq: int = 5*) → str

1, 5, 10, 20, 30, 50

Parameters **attitude_freq** –

Returns okraise

gimbal_recenter()

:return: okraise

gimbal_resume()

:return: okraise

gimbal_speed(*pitch: float, yaw: float*) → str

Parameters

- **pitch** – pitch °/s
- **yaw** – yaw °/s

Returns okraise

gimbal_suspend()

:return: okraise

ir_sensor_measure(*switch: bool = True*)

/

Parameters **switch** – /

Returns okraise

led_control(*comp: str, effect: str, r: int, g: int, b: int*) → str

LED LED

Parameters

- **comp** – LED LED_ENUMS
- **effect** – LED_EFFECT_ENUMS
- **r** – RGB
- **g** – RGB
- **b** – RGB

Returns okraise

robot_mode(*mode: str*) → str

Parameters

mode – enum MODE_*

Returns okraise

sound_event(*attr: str*, *switch: bool*) → str

Parameters

- **attr** – SOUND_ENUMS
- **switch** –

Returns okraise

stream(*switch: bool*) → str

Parameters

switch – /

Returns okraise

version() → str

query robomaster SDK version

Returns version string

1.3.3 Framework

The framework deals with video streaming, push and event, you can build your controlling logic basing on it, for example(TODO: add links):

- Drive your robomaster using keyboard;
- Make your robomaster a goalkeeper.

class robomasterpy.framework.**Hub**

TERMINATION_TIMEOUT = 10

close()

run()

Start workers and block the main process until all workers terminate.

Mind tries to shutdown itself gracefully when receiving SIGTERM or SIGINT.

worker(*worker_class*, *name: str*, *args: Tuple = ()*, *kwargs=None*)

Register worker to process sensor data fetching, calculation, inference,controlling, communication and more. All workers run in their own operating system process.

Parameters

- **worker_class** –
- **name** –
- **args** – args to func
- **kwargs** – kwargs to func

class robomasterpy.framework.**Worker**(*name: str*, *out: Optional[multiprocessing.context.BaseContext.Queue]*, *protocol: Optional[str]*, *address: Tuple[str, int]*, *timeout: Optional[float]*, *loop: bool = True*)

QUEUE_TIMEOUT: float = 0.05

close()

property closed

get_address() → Tuple[str, int]

property logger

property name

work() → None

class robomasterpy.framework.**Vision**(*name: str, out: Optional[multiprocessing.context.BaseContext.Queue], ip: str, processing: Callable[[...], None], none_is_valid=False*)

QUEUE_TIMEOUT = 0.05

TIMEOUT: float = 5.0

close()

property closed

get_address() → Tuple[str, int]

property logger

property name

work() → None

class robomasterpy.framework.**PushListener**(*name: str, out: multiprocessing.context.BaseContext.Queue*)

PUSH_TYPES: Tuple[str] = ('chassis', 'gimbal')

PUSH_TYPE_CHASSIS: str = 'chassis'

PUSH_TYPE_GIMBAL: str = 'gimbal'

QUEUE_TIMEOUT = 0.05

close()

property closed

get_address() → Tuple[str, int]

property logger

property name

work() → None

class robomasterpy.framework.**EventListener**(*name: str, out: multiprocessing.context.BaseContext.Queue, ip: str*)

EVENT_TYPES: Tuple[str] = ('armor', 'sound')

EVENT_TYPE_ARMOR: str = 'armor'

EVENT_TYPE_SOUND: str = 'sound'

QUEUE_TIMEOUT = 0.05

close()

property closed

get_address() → Tuple[str, int]

property logger

property name

work() → None

class robomasterpy.framework.**Mind**(*name: str, queues: Tuple[multiprocessing.context.BaseContext.Queue, ...], ip: str, processing: Callable[[...], None], timeout: float = 30, loop: bool = True*)

QUEUE_TIMEOUT: float = 0.05

close()

property closed

get_address() → Tuple[str, int]

property logger

property name

work() → None

1.3.4 Helpers for Measure

Some helpers for measure and analyze distance from video stream.

robomasterpy.measure.**pinhole_distance**(*actual_size: float, pixel_size: float, focal_length: float = 710*) → float

Estimate distance between camera and object using pinhole camera model.

Parameters

- **actual_size** – Actual object size in meters.
- **pixel_size** – Object size in pixels.
- **focal_length** – 1280*720Perceived focal length at specified image resolution, default to 1280*720.

Returns Distance between camera and object, in meters.

robomasterpy.measure.**distance_decomposition**(*pixel_x: float, distance: float, horizontal_pixels: float = 1280, horizontal_degrees: float = 96*) → Tuple[float, float, float]

Decomposition distance into forward vector(forward as positive) and lateral vector(right as positive). This function requires that both ends of the line segment are at the same altitude.

Parameters

- **pixel_x** – x the x coordinate of the object on the image, in pixels.
- **distance** – Distance in meter.
- **horizontal_pixels** – 1280. The number of pixels in the horizontal direction of the image, the default is 1280.
- **horizontal_degrees** – 96. The horizontal viewing angle of the image, the default is 96.

Returns forward vector and lateral vector in meters; horizontal angle in degrees.

CHAPTER

TWO

HEALTH AND SAFETY NOTICE

- Your Robomaster may hurt people or pet, break stuffs or itself;
- Make sure your RoboMaster has enough room to move; make sure the ground is clear;
- Start slowly, avoid using high speed for debugging;
- Use cushion;
- Stay safe and have fun!

CHAPTER

THREE

PAPERWORK

RoboMasterPy is a fan work, and it has no concern with DJI.

DJI, RoboMaster are trademarks of SZ DJI Technology Co., Ltd.

CHAPTER

FOUR

ACKNOWLEDGEMENT

RoboMasterPy was developed during a RoboMaster EP developing contest. The author would like to thank DJI for hardware and technical support.

CHAPTER

FIVE

LICENSE

RoboMasterPy is released under MIT license.

INDEX

A

armor_event() (*robomasterpy.Commander method*), 3
armor_sensitivity() (*robomasterpy.Commander method*), 4
audio() (*robomasterpy.Commander method*), 4

B

blaster_fire() (*robomasterpy.Commander method*), 4

C

chassis_move() (*robomasterpy.Commander method*), 4
chassis_push_off() (*robomasterpy.Commander method*), 4
chassis_push_on() (*robomasterpy.Commander method*), 4
chassis_speed() (*robomasterpy.Commander method*), 4
chassis_wheel() (*robomasterpy.Commander method*), 5
close() (*robomasterpy.Commander method*), 5
close() (*robomasterpy.framework.EventListener method*), 8
close() (*robomasterpy.framework.Hub method*), 7
close() (*robomasterpy.framework.Mind method*), 9
close() (*robomasterpy.framework.PushListener method*), 8
close() (*robomasterpy.framework.Vision method*), 8
close() (*robomasterpy.framework.Worker method*), 7
closed() (*robomasterpy.framework.EventListener property*), 8
closed() (*robomasterpy.framework.Mind property*), 9
closed() (*robomasterpy.framework.PushListener property*), 8
closed() (*robomasterpy.framework.Vision property*), 8
closed() (*robomasterpy.framework.Worker property*), 8
Commander (*class in robomasterpy*), 3

D

distance_decomposition() (*in module robomasterpy.measure*), 9
do() (*robomasterpy.Commander method*), 5

E

EVENT_TYPE_ARMOR (*robomasterpy.framework.EventListener attribute*), 8
EVENT_TYPE_SOUND (*robomasterpy.framework.EventListener attribute*), 8
EVENT_TYPES (*robomasterpy.framework.EventListener attribute*), 8
EventListener (*class in robomasterpy.framework*), 8

G

get_address() (*robomasterpy.framework.EventListener method*), 9
get_address() (*robomasterpy.framework.Mind method*), 9
get_address() (*robomasterpy.framework.PushListener method*), 8
get_address() (*robomasterpy.framework.Vision method*), 8
get_address() (*robomasterpy.framework.Worker method*), 8
get_armor_sensitivity() (*robomasterpy.Commander method*), 5
get_broadcast_ip() (*in module robomasterpy*), 3
get_chassis_attitude() (*robomasterpy.Commander method*), 5
get_chassis_position() (*robomasterpy.Commander method*), 5
get_chassis_speed() (*robomasterpy.Commander method*), 5
get_chassis_status() (*robomasterpy.Commander method*), 5
get_gimbal_attitude() (*robomasterpy.Commander method*), 5

get_ip() (*robomasterpy.Commander method*), 5
get_ir_sensor_distance() (*robomasterpy.Commander method*), 5
get_robot_mode() (*robomasterpy.Commander method*), 5
gimbal_move() (*robomasterpy.Commander method*), 5
gimbal_moveto() (*robomasterpy.Commander method*), 6
gimbal_push_off() (*robomasterpy.Commander method*), 6
gimbal_push_on() (*robomasterpy.Commander method*), 6
gimbal_recenter() (*robomasterpy.Commander method*), 6
gimbal_resume() (*robomasterpy.Commander method*), 6
gimbal_speed() (*robomasterpy.Commander method*), 6
gimbal_suspend() (*robomasterpy.Commander method*), 6

H

Hub (*class in robomasterpy.framework*), 7

I

ir_sensor_measure() (*robomasterpy.Commander method*), 6

L

led_control() (*robomasterpy.Commander method*), 6
logger() (*robomasterpy.framework.EventListener property*), 9
logger() (*robomasterpy.framework.Mind property*), 9
logger() (*robomasterpy.framework.PushListener property*), 8
logger() (*robomasterpy.framework.Vision property*), 8
logger() (*robomasterpy.framework.Worker property*), 8

M

Mind (*class in robomasterpy.framework*), 9

N

name() (*robomasterpy.framework.EventListener property*), 9
name() (*robomasterpy.framework.Mind property*), 9
name() (*robomasterpy.framework.PushListener property*), 8
name() (*robomasterpy.framework.Vision property*), 8
name() (*robomasterpy.framework.Worker property*), 8

P

pinhole_distance() (*in module robomasterpy.measure*), 9
PUSH_TYPE_CHASSIS (*robomasterpy.framework.PushListener attribute*), 8
PUSH_TYPE_GIMBAL (*robomasterpy.framework.PushListener attribute*), 8
PUSH_TYPES (*robomasterpy.framework.PushListener attribute*), 8
PushListener (*class in robomasterpy.framework*), 8

Q

QUEUE_TIMEOUT (*robomasterpy.framework.EventListener attribute*), 8
QUEUE_TIMEOUT (*robomasterpy.framework.Mind attribute*), 9
QUEUE_TIMEOUT (*robomasterpy.framework.PushListener attribute*), 8
QUEUE_TIMEOUT (*robomasterpy.framework.Vision attribute*), 8
QUEUE_TIMEOUT (*robomasterpy.framework.Worker attribute*), 7

R

robot_mode() (*robomasterpy.Commander method*), 6
run() (*robomasterpy.framework.Hub method*), 7

S

sound_event() (*robomasterpy.Commander method*), 7
stream() (*robomasterpy.Commander method*), 7

T

TERMINATION_TIMEOUT (*robomasterpy.framework.Hub attribute*), 7
TIMEOUT (*robomasterpy.framework.Vision attribute*), 8

V

version() (*robomasterpy.Commander method*), 7
Vision (*class in robomasterpy.framework*), 8

W

work() (*robomasterpy.framework.EventListener method*), 9
work() (*robomasterpy.framework.Mind method*), 9
work() (*robomasterpy.framework.PushListener method*), 8
work() (*robomasterpy.framework.Vision method*), 8
work() (*robomasterpy.framework.Worker method*), 8

`Worker` (*class in robomasterpy.framework*), 7
`worker()` (*robomasterpy.framework.Hub method*), 7