
PiCar

Release 0.0.1

Russell Shomberg

Dec 08, 2019

CONTENTS:

1	The picar API reference	1
2	The “server” module	3
3	The “client” module	5
	Python Module Index	7
	Index	9

THE PICAR API REFERENCE

```
class picar.Camera(resolution=640.48)
```

Camera class which can read or stream over a socket

```
close()
```

Close camera nicely

```
read()
```

Return frame same as cv2.read()

```
stream(sock)
```

Continuous capture and send over network

```
class picar.LineSensor(pin_middle=16, pin_left=19, pin_right=20, blackLine=True)
```

Line sensor consisting of 3 elements in a linear array

```
class picar.Motor(en, pin1, pin2)
```

Class for controlling the DC motor

```
coast()
```

Releases motor to coast

```
drive(speed)
```

checks direction and moves

```
forward(speed=100)
```

Drives forward

```
reverse(speed=100)
```

Drives backwards

```
stop()
```

Stops motor after pulling in the opposite direction to hard stop

```
class picar.PiCar
```

Provides an interface to control the car.

Upon construction, this class initializes all controls and sensors. For controls, the car has a motor, a turning servo, and 2 servos controlling the head. For sensors, the car has a line sensor, a sonar, and a camera. The constructor expects all the peripherals to be plugged into the pi in a specific manner which can only be changed by directly changing the code of the constructor.

```
all_ahed()
```

Bring all controls to forward

```
all_stop()
```

Stop the car and face forward. Exit any current mode

ebrake (*dir=1*)

Hard brake by reversing for a second

Change dir to -1 to use ebrake when reversed

follow_line (*darkLine=False, speed=1, gain=(10, 60, 0), nHist=100, maxAng=30*)

Look for a line and drive along following it

The follow line is a generator needs to be continuously called in order to continue following the line.

pulse (*runTime, coastTime*)

Continuously pulse the motor.

Generally, this method should be called as a thread to run in the background. Otherwise it will block the program from running.

run_cmd (*cmd, arg=""*)

Run a preset command

sonar_scan (*distance=1, scanSpeed=1, tiltAngle=0*)

Measure distances across full range of sonar

Car will first look all the way to the left. It will then slowly turn all the way to the right, while making a sonar measurement at each angle. distance determines how long the ping should wait for a response. scanSpeed determines how far apart the pings are angularly. A slower scanSpeed means more pings and denser set of results. scanSpeed must be an integer greater than 0.

sonar_scan() returns a list of distances and a second list of corresponding angles.

track_object ()

Keep an object in view and follow it

class `picar.Servo` (*pin, pwmMin, pwmMax, pwmCenter, angMin, angMax, angCenter*)

Class defining a servo

center ()

Go to center

goto (*angle*)

Rotate to angle

max ()

Go to max

min ()

Go to min

rotate (*angle*)

Rotate an increment

THE “SERVER” MODULE

```
class server.FootageStream(camera, addr, port=5555)
```

Continuously serves requests for the latest frame

```
    run()
```

Send images to the connected socket

```
class server.Reciever(addr="", port=5000)
```

```
    run()
```

Method representing the thread's activity.

You may override this method in a subclass. The standard `run()` method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the `args` and `kwargs` arguments, respectively.

```
server.main()
```

Start the picar and wait for commands

THE “CLIENT” MODULE

```
class client.Keyboard (transmitter)
    Keystroke commands that can be sent over the transmitter

    run ()
        Background thread listening to for keyboard events

    run_cmd (cmd)
        Send a command to picar reciever
```

```
class client.LiveFeed (addr='192.168.0.212', port=5555, threaded=False)
    Thread that receives frames from server and makes them available to other client threads.

    read ()
        Read a single frame from server

    run ()
        Continuously read frames and update in background
```

```
class client.ObjectTracker (feed, transmitter, tracker='kcf', mode='watch')
    Object tracker is a thread that has an opencv tracker

    The tracker is given access to a live feed where gets updated frames and to a transmitter which can send commands to a picar. Object tracking with the PiCar is poor, likely due to the instability of the head, particularly when moving.

    deselect ()
        Clear selected object and stop tracking

    run ()
        Track object in background using latest frame

        Updates the box location for the viewer. Also sends command using transmitter so the car will look towards the object thus centering it in the screen.

    select (frame)
        Select object to track from a frame

        Gives the user the ability to select an object from a paused frame. The selected contains the portion of the image which will be tracked.

    track (frame)
        Get a single update of the tracker.
```

```
class client.Transmitter (addr='192.168.0.212', port=5000)
    Class containing a socket and the ability to send commands to the server

    There should only be one instance unless multiple cars are being controlled. Every controller must be passed the transmitter in order to work.
```

send_cmd (*cmd*)

Send command to picar reciever

`client.voice_command()`

Get input from the user using voice recognition.

Voice commands are limited because implimenting a call for each can become complicated. The car has a tendency to get out of control, because the user cannot respond quickly enough.

PYTHON MODULE INDEX

c

client, 5

p

picar, 1

s

server, 3

A

`all_ahead()` (*picar.PiCar method*), 1
`all_stop()` (*picar.PiCar method*), 1

C

`Camera` (*class in picar*), 1
`center()` (*picar.Servo method*), 2
`client` (*module*), 5
`close()` (*picar.Camera method*), 1
`coast()` (*picar.Motor method*), 1

D

`deselect()` (*client.ObjectTracker method*), 5
`drive()` (*picar.Motor method*), 1

E

`ebrake()` (*picar.PiCar method*), 1

F

`follow_line()` (*picar.PiCar method*), 2
`FootageStream` (*class in server*), 3
`forward()` (*picar.Motor method*), 1

G

`goto()` (*picar.Servo method*), 2

K

`Keyboard` (*class in client*), 5

L

`LineSensor` (*class in picar*), 1
`LiveFeed` (*class in client*), 5

M

`main()` (*in module server*), 3
`max()` (*picar.Servo method*), 2
`min()` (*picar.Servo method*), 2
`Motor` (*class in picar*), 1

O

`ObjectTracker` (*class in client*), 5

P

`PiCar` (*class in picar*), 1
`picar` (*module*), 1
`pulse()` (*picar.PiCar method*), 2

R

`read()` (*client.LiveFeed method*), 5
`read()` (*picar.Camera method*), 1
`Reciever` (*class in server*), 3
`reverse()` (*picar.Motor method*), 1
`rotate()` (*picar.Servo method*), 2
`run()` (*client.Keyboard method*), 5
`run()` (*client.LiveFeed method*), 5
`run()` (*client.ObjectTracker method*), 5
`run()` (*server.FootageStream method*), 3
`run()` (*server.Reciever method*), 3
`run_cmd()` (*client.Keyboard method*), 5
`run_cmd()` (*picar.PiCar method*), 2

S

`select()` (*client.ObjectTracker method*), 5
`send_cmd()` (*client.Transmitter method*), 5
`server` (*module*), 3
`Servo` (*class in picar*), 2
`sonar_scan()` (*picar.PiCar method*), 2
`stop()` (*picar.Motor method*), 1
`stream()` (*picar.Camera method*), 1

T

`track()` (*client.ObjectTracker method*), 5
`track_object()` (*picar.PiCar method*), 2
`Transmitter` (*class in client*), 5

V

`voice_command()` (*in module client*), 6