# Spockbots

***Release 1.0***

## Spockbots

**Nov 06, 2019**

# CONTENTS

CHAPTER

ONE

# SPOCKBOTS API

## 1.1 Menu

We named it *0_menu.py* so it shows up on the top in the brick program:

```
     Crane
>>>  Swing
     Calibrate
     ....
```

Displays a menu in which we muve with the UP DOWN keys up and down. We leave with the left key and select a program with the right key.

## 1.2 City Runs

### 1.2.1 run.calibrate

run.calibrate.**run_calibrate**()
Run the calibration

**Returns** a file called calibrate.txt that containes the minimum black and the maximum white value for the sensors

### 1.2.2 run.check

run.check.**run_check**()
Checks the robot by driving the large and medium motors and flashing the color sensors

Order:

- Large Motor left
- Large Motor left
- Medium Motor left
- Medium Motor left
- Color Sensor left
- Color Sensor right
- Color Sensor back

### 1.2.3 run.crane

run.crane.**run_crane**()

### 1.2.4 run.led

run.led.**run_led**()

### 1.2.5 run.swing

run.swing.**run_swing**()

### 1.2.6 run.turn_to_black module

## 1.3 Spockbots API

### 1.3.1 spockbots.check

spockbots.check.**check**(*speed=100*, *angle=360*)
do a robot check by

a) turning on the large motors one at a time

b) turning on the medium motors one at a time

c) turning on the light sensors one at a time

**Parameters**

- **speed** –
- **angle** –

**Returns**

### 1.3.2 spockbots.colorsensor

**class** spockbots.colorsensor.**SpockbotsColorSensor**(*port=3*)
Bases: object

**clear**()

**flash**()
flashes the color sensor by switching between color and reflective mode

**info**()
prints the black and white value read form the sensor

**light**()

**read**()
reads the color sensor data form the file :return:

**reflection**()

**set_black**()
sets the current value to black

**set_white**()
sets the current value to white :return:

**value**()
reads the current value mapped between 0 and 100 :return:

**write**()
append the black and white value to a file

**class** spockbots.colorsensor.**SpockbotsColorSensors**(*ports=[2, 3, 4], speed=5*)
Bases: object

**clear**()

**flash**(*ports=[2, 3, 4]*)

**info**(*ports=[2, 3, 4]*)

**read**()

**sensor**(*port*)

**test**(*ports=[2, 3, 4]*)

**value**(*i*)

**write**(*ports=[2, 3, 4]*)

### 1.3.3 spockbots.gyro

### 1.3.4 spockbots.motor

spockbots.motor.**PRINT**(**args*)

**class** spockbots.motor.**SpockbotsMotor**(*direction=None*)
Bases: object

**angle_to_distance**(*angle*)

**beep**()
The robot will make a beep

**calibrate**(*speed, distance=15, ports=[2, 3, 4], direction='front'*)

**distance_to_angle**(*distance*)
calculation to convert the distance from cm into rotations.

**Parameters** **distance** – The distance in cm

**Returns** The rotations to be traveled for the given distance

**distance_to_rotation**(*distance*)
calculation to convert the distance from cm into rotations.

**Parameters** **distance** – The distance in cm

**Returns** The rotations to be traveled for the given distance

**followline**(*speed=25*, *distance=None*, *t=None*, *port=3*, *right=True*, *black=0*, *white=100*, *delta=-35*, *factor=0.7*)

**forward**(*speed*, *distance*, *brake=None*)

**gotoblack**(*speed*, *port*, *black=10*)
The robot moves to the black line while using the sensor on the given port

**Parameters**

- **speed** – The speed
- **port** – The port 1,2,3,4
- **black** – The value to stop

**gotowhite**(*speed*, *port*, *white=90*)
The robot moves to the white line while using the sensor on the given port

**Parameters**

- **speed** – The speed
- **port** – The port 1,2,3,4
- **white** – The value to stop

**light**(*port*)

**on**(*speed*, *steering=0*)

**reset**()

**setup**(*direction=None*)

**still**()

**stop**(*brake=None*)
stops all motors on all different drive modes

**Parameters** **brake** – None, brake, coast, hold

**Returns**

**tunrtoblack**(*speed*, *direction='left'*, *port=3*, *black=10*)
turns the robot to the balck line. :param speed: :param port: :param black: :return:

**turn**(*speed*, *angle*)
takes the radius of the robot and dives on it for a distance based on the ancle :param speed: :param angle: :return:

### 1.3.5 spockbots.output

spockbots.output.**PRINT**(**args*, *x=None*, *y=None*)

spockbots.output.**beep**()
The robot will make a beep

spockbots.output.**clear**()

spockbots.output.**flash**(*colors=['RED', 'BLACK', 'RED', 'BLACK', 'GREEN'], delay=0.1*)
The robot will flash the LEDs and beep twice

spockbots.output.**led**(*color*, *brightness=255*)

spockbots.output.**readfile**(*name*)

spockbots.output.**sound**(*pitch=1500*, *duration=300*)

spockbots.output.**voltage**()

spockbots.output.**writefile**(*name*, *msg*)

### 1.3.6 spockbots.robot

### 1.3.7 spockbots.systemgyro

**class** spockbots.systemgyro.**Gyro**

Bases: object

**angle**()

**connect**()

**get**()

**info**()

**mode**(*kind*)

GYRO-G&A GYRO-ANG GYRO-RATE GYRO-CAL

Not supported: GYRO-FAS TILT-RATE TILT-ANG

**Parameters** **kind** –

**Returns**

**rate**()

**reset**()

**still**(*count=10*, *still=5*)

**test**(*n*)

## 1.4 Examples

### 1.4.1 door

### 1.4.2 gyro

### 1.4.3 interpreter

### 1.4.4 m

### 1.4.5 test

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## r



## s

# INDEX

## S



## T



## V



## W