TDDD63 Project:
API

Jonathan Doherty

November 1, 2012

Contents

1 Using the API

To use the functions contained in this document you have to import the API module containing all the functions. This is shown by the example below.

```
from API import *
```

The structure of the API is the following.

Function:
Displays the function.

Argument:
Shows you what the parameters are and what they stand for.

Return:
Tells you what value the function will return when calling it.

Description:
Gives you additional information about the function that may be good to know.

Example:
Example code showing you how to call the function.

2 Connection and Status

2.1 connect()

Function:

connect(brick_id_address = None)

Argument:

brick_id_address – ID address of your brick

Return:

None

Description:

Connects your computer with the brick enabling data transfer over Bluetooth or USB and initializes MotorControl.rxe. When no argument is supplied tries to search for Bluetooth and USB by itself and connect but if valid argument is given skips these steps and connects directly.

Example:

```
#Default connection (slow)
connect()

#Connect with a bricks ID address (fast)
connect( "00:23:43:2A:5A:3F" )
```

2.2 close()

Function:

close()

Argument:

–

Return:

None

Description:

Close connection to brick and stop MotorControl.rxe.

Example:

close()

```
#Close connection
close()
```

3 Sensors

3.1 play_file()

Function:

play_file(sound, times=1)

Argument:

sound – The sound file residing on the brick to play

times – How many times the sound file should be played

Return:

None

Description:

Plays any of the sound files residing on the brick given amount of times. If no value is given to "times" default value is one.

Example:

```
#Play Green sound file three times
play_file("Green", 3)

#Play Goodbye sound file once
play_file("Goodbye)
```

3.2 get_touch1_sample()

Function:

get_touch1_sample(port = PORT_1)

Argument:

port – The port to use the sensor with

Return:

Boolean

Description:

Returns True if the touch sensor at port 1 is pressed down otherwise False.

If no parameter is supplied it will use the compass at port 1. If you give it a parameter it will use that value as the port number to use.

Example:

```
#Get current Touch Sensor reading
get_touch1_sample()

#Use Touch Sensor at another port,
#in this case port 3.
activate_color_sensor(2)
```

3.3 get_touch2_sample()

Function:

get_touch2_sample(port = PORT_2)

Argument:

port – The port to use the sensor with

Return:

Boolean

Description:

Returns True if the touch sensor at port 2 is pressed down otherwise False.

If no parameter is supplied it will use the compass at port 2. If you give it a parameter it will use that value as the port number to use.

Example:

```
#Get current Touch Sensor reading
get_touch2_sample()

#Use Touch Sensor at another port,
#in this case port 4.
get_touch2_sample(3)
```

3.4 activate_color_sensor()

Function:

activate_color_sensor(port = PORT_3)

Argument:

port – The port to use the sensor with

Return:

None

Description:

If no parameter is supplied it will activate the color sensor at port 3 making it possible to change color mode and read values from it. If you give it a parameter it will use that value as the port number to activate it.

Example:

```
#Activate Color Sensor
activate_color_sensor()

#Activate Color Sensor at another port,
#in this case port 2.
activate_color_sensor(2)
```

3.5 get_color_sample()

Function:

get_color_sample()

Argument:

–

Return:

int

Description:

In Color mode returns an integer value between 1 and 6 representing the current color.
Black = 1
Blue = 2
Green = 3
Yellow = 4
Red = 5
White = 6

In Light mode returns an integer value representing the currently brightness or darkness.

Brightness = high values

Darkness = low values

Example:

```
#Get current Color Sensor reading
get_color_sample()
```

3.6 set_color()

Function:

set_color(color)

Argument:

color – string value of the color you wish to activate

Return:

None

Description:

Switches colors and modes by changing the color.

all = Color sensor mode
red = Light color sensor mode with red color
green = Light color sensor mode with green color
blue = Light color sensor mode with blue color
none = Turns off all colors

Example:

```
#Set Color Sensor color to red activating Light mode
set_color( "red" )

#Set Color Sensor color to all activating Color mode
set_color( "all" )

#Set Color Sensor color to none turning off all modes
# and colors
set_color( "none" )
```

3.7 get_color()

Function:

get_color_()

Argument:

–

Return:

string

Description:

Returns the current color activated as an string.

"all"
"red"
"green"
"blue"
"none"

Example:

```
#Get the currently activated Color Sensor color/mode
get_color_()
```

3.8 get_ultrasonic_sample()

Function:

get_ultrasonic_sample(port = PORT_4)

Argument:

port – The port to use the sensor with

Return:

int

Description:

Returns a value between 0 and 200 that shows how far/close an object is to the sensor.

If no parameter is supplied it will use the compass at port 4. If you give it a parameter it will use that value as the port number to use.

Example:

```
#Get current Ultrasonic Sensor reading
get_ultrasonic_sample()

#Use Ultrasonic Sensor at another port,
#in this case port 1.
get_ultrasonic_sample(0)
```

3.9 get_compass_sample()

Function:

get_compass_sample(port = PORT_2)

Argument:

port – The port to use the sensor with

Return:

int

Description:

Returns heading from North in degrees, 0 to 360 degrees.

If no parameter is supplied it will use the compass at port 2. If you give it a parameter it will use that value as the port number to use.

Important, the compass sensor should be places as far away from the motors as possible for more accurate readings.

Example:

```
#Get compass reading
get_compass_sample()

#Use Compass at another port,
#in this case port 1.
activate_color_sensor(0)
```

3.10 get_relative_compass

Function:

get_relative_compass(target = 0, port = PORT_2)

Argument:

port – The port to use the sensor with
target – The target you want to use

Return:

int

Description:

Returns how far away from target you are if you have 360 as target and you are on 340 or 20 you are 20/-20 away from target, uses the values -180 to 180 to determine how far away from target you are.

If no parameter is supplied it will use the compass at port 2. If you give it a parameter it will use that value as the port number to use. Same with target if no parameter is supplied it will use 0 as target

Important, the compass sensor should be places as far away from the motors as possible for more accurate readings.

Example:

```
#Get relative compass reading with 50 as target
get_relative_compass(50)

#Use relative compass at another port with target as 50,
#in this case port 1.
get_relative_compass(50, 0)
```

3.11 is_in_range_compass

Function:

is_in_range_compass(minValue, maxValue, port = PORT_2)

Argument:

minValue – First value to track
maxValue – Second value to track
port – The port to use the sensor with

Return:

int

Description:

Returns True if compass is between the values minValue and maxValue, otherwise returns False.

If no port parameter is supplied it will use the compass at port 2.

Important, the compass sensor should be places as far away from the motors as possible for more accurate readings.

Example:

```
#Get True if compass between 50 and 100
is_in_range_compass(50, 100)

#Use relative compass at another port
#in this case port 1.
is_in_range_compass(50, 100, 0)
```

3.12 get_x_acceleration()

Function:

get_x_acceleration(port = PORT_1)

Argument:

port – The port to use the sensor with

Return:

int

Description:

Returns the x axis.

If no parameter is supplied it will use the compass at port 1. If you give it a parameter it will use that value as the port number to use.

Example:

```
#Get current x axis reading
get_x_acceleration()

#Use acceleration sensor at another port,
#in this case port 3.
get_x_acceleration()(2)
```

3.13 get_y_acceleration()

Function:

get_y_acceleration(port = PORT_1)

Argument:

port – The port to use the sensor with

Return:

int

Description:

Returns the y axis.

If no parameter is supplied it will use the compass at port 1. If you give it a parameter it will use that value as the port number to use.

Example:

```
#Get current x axis reading
get_y_acceleration()

#Use acceleration sensor at another port,
#in this case port 3.
get_y_acceleration()(2)
```

3.14 get_z_acceleration()

Function:

get_z_acceleration(port = PORT_1)

Argument:

port – The port to use the sensor with

Return:

int

Description:

Returns the z axis.

If no parameter is supplied it will use the compass at port 1. If you give it a parameter it will use that value as the port number to use.

Example:

```
#Get current x axis reading
get_z_acceleration()

#Use acceleration sensor at another port,
#in this case port 3.
get_z_acceleration()(2)
```

4 Moving

4.1 run_motor_A()

Function:

run_motor_A(power, tacho_units)

Argument:

power – With what speed/power to run the motor, value between -100 and 100

tacho_units – Number of motor degrees to turn, value between 0-999999

Return:

None

Description:

Rotates the motor at port A given amount of tacho_units with a speed of power.

(power = 100 moves forward and -100 moves backward)

Example:

```
#Run motor forward
run_motor_A(100, 720)

#Run motor bakward
run_motor_A(-100, 720)
```

4.2 run_motor_B()

Function:

run_motor_B(power, tacho_units)

Argument:

power – With what speed/power to run the motor, value between -100 and 100

tacho_units – Number of motor degrees to turn, value between 0-999999

Return:

None

Description:

Rotates the motor at port B given amount of tacho_units with a speed of power.

(power = 100 moves forward and -100 moves backward)

Example:

```
#Run motor forward
run_motor_B(80, 90)

#Run motor backward
run_motor_B(-80, 360)
```

4.3 run_motor_C()

Function:

run_motor_C(power, tacho_units)

Argument:

power – With what speed/power to run the motor, value between -100 and 100

tacho_units – Number of motor degrees to turn, value between 0-999999

Return:

None

Description:

Rotates the motor at port C given amount of tacho_units with a speed of power.

(power = 100 moves forward and -100 moves backward)

Example:

```
#Run motor 180 tacho_units with a power of 100
run_motor_C(100, 180)

#Run motor reversed 180 tacho_units with a power of -100
run_motor_C(-100, 180)
```

4.4 move()

Function:

move(power, tacho_units)

Argument:

power – With what speed/power to run the motor, value between -100 and 100

tacho_units – Number of motor degrees to turn, value between 0-999999

Return:

None

Description:

Moves the robot given amount of tacho_units forward or backward by rotating motors at port B and port C in synchronization.

(power = 100 moves forward and -100 moves backward)

Example:

```
#Run motor B and C forward simultaneously
move(100, 420)

#Run motor B and C backward simultaneously
move(-100, 420)
```

4.5 move_forever()

Function:

move_forever(power)

Argument:

power – With what speed/power to run the motor, value between -100 and 100

Return:

None

Description:

Moves the robot forward or backward forever by rotating motors at port B and port C in synchronization. Unlike move() this function tells the motor to run forever until you tell it to stop by calling stop_move_forever(). But it will only work using either sleep() or placing it inside a while loop.

OBS!

It is very important you call stop_move_forever() before calling any other function like turn_left(), turn_right() and move() otherwise the program may halt without any error.

(power = 100 moves forward and -100 moves backward)

Example:

```
#This works!
while True:
    move_forever(100)
    if True:
        stop_move_forever()
        turn_left(100, 360)

#This does not work!
#stop_move_forever() missing
while True:
    move_forever(100)
    if True:
        turn_left(100, 360)

#This works!
move_forever(100)
sleep(2)
stop_move_forever()

#This does not work!
#Sleep() missing
move_forever(100)
stop_move_forever()
```

4.6 turn_left_forever()

Function:

turn_left_forever(power)

Argument:

–

Return:

None

Description:

This function sets the motor at port B to run forever with the given power.

(power = 100 moves forward and -100 moves backward)

Example:

```
#Motors at port C and B are running at power 100.
move_forever(100)

#We tell them to run with a power of 0
stop_move_forever()

#We then tell the motor at port B to run with a power of
#100 this will make the robot turn left forever
turn_left_forever(100)

#We can also modify the speed of a running motor
#This will increase the motor at port B to run with a
#power of 100 instead of 60. But the motor at port C
#will still run with power of 60
move_forever(60)
turn_left_forever(100)
```

4.7 turn_right_forever()

Function:

turn_right_forever(power)

Argument:

–

Return:

None

Description:

This function sets the motor at port C to run forever with the given power.

(power = 100 moves forward and -100 moves backward)

Example:

```
#Motors at port C and B are running at power 100.
move_forever(100)

#We tell them to run with a power of 0
stop_move_forever()

#We then tell the motor at port C to run with a power of
#100 this will make the robot turn right forever
turn_right_forever(100)

#We can also modify the speed of a running motor
#This will increase the motor at port C to run with a
#power of 100 instead of 60. But the motor at port B
#will still run with power of 60
move_forever(60)
turn_right_forever(100)
```

4.8 turn_left()

Function:

turn_left(power, tacho_units)

Argument:

power – With what speed/power to run the motor, value between 0 and 100

tacho_units – Number of motor degrees to turn, value between 0-999999

Return:

None

Description:

Turns the robot given amount of tacho_units by rotating motors at port C and port B opposite of each other to turn left.

Example:

```
#Turn left 90 motor degrees
turn_left(100, 90)
```

4.9 turn_right()

Function:

turn_right(power, tacho_units)

Argument:

power – With what speed/power to run the motor, value between 0 and 100

tacho_units – Number of motor degrees to turn, value between 0-999999

Return:

None

Description:

Turns the robot given amount of tacho_units by rotating motors at port C and port B opposite of each other to turn right.

Example:

```
#Turn right 180 motor degrees
turn_right(100, 180)
```