eYSIP 2017

Tutorial on Interfacing a Wireless Joystick Controller for Quadcopters

Xbox® One Controller[1]

Heethesh Vhavle
Sanam Shakya
Pushkar Raj

Duration of Internship: 22/05/2017 – 07/07/2017

2017, e-Yantra Publication

Wireless Joystick Controller for Quadcopters

1.1 Introduction

The project deals with the study of control algorithms and to develop a custom firmware for quadcopter (flight controller) on 32-bit micro-controllers. The flight controller is designed to control parameters such as the throttle, yaw, pitch and roll and consists of algorithms considering various motion and dynamics. One of the goals is to develop a wireless joystick controller for simple maneuvering of the quadcopter.

A very common method of piloting drones is by using a RC transmitter with 6-8 channels. In this project an USB-HID gamepad controller is interfaced with Python, which in turn transmits commands (MSP frames) to the quadcopter over Wi-Fi. Other than throttle and direction control for drones, there are certain other controls that must be provided to a pilot for a stable and safe flight. The list of controls implemented in the firmware is explained later.

In this project, the Xbox® One Controller[1] is used. this controller is a USB-HID Gamepad compliant device and is interfaced with Python with the help of *pygame* module. The inputs from the controller are read, parsed, converted and then packed into MSP frames and sent to the quadcopter. Any other USB game controller can also be interfaced in the same way.

For communication with the drone and debugging purposes, it will be easier to do so with the help of serial communication port. But the PC is connected to the drone over Wi-Fi. A virtual COM port can be created with the help of

softwares such as Eltima Serial to Ethernet Connector or HW Virtual Serial Port which redirects the data from TCP/IP sockets to the COM ports.

1.2 List of Controls

Although flight planner/controller software such as MultiWii Conf exist, it is very difficult to pilot a drone using them, especially in emergency situations. Following is a list of controls implemented on the controller.

Figure 1.1: Labeled diagram showing the mapping of the different controls on the Xbox® One Controller[1]

- **Throttle** — *Range: 1000 - 2000* — This sets the base throttle value for all the motors in manual control mode and is used to control the altitude set point in auto flight mode.

- **Roll** — *Range: -60 - +60* — This channel varies the set point for the roll angle to maneuver the drone in the left-right direction.

- **Pitch** — *Range: -60 - +60* — This channel varies the set point for the pitch angle to maneuver the drone in the left-right direction.

- **Yaw** — *Range: -180 - +180* — This channel varies the set point for the yaw angle rotate the drone either in the clockwise or anti-clockwise direction.

- **Arm** — *Range: 0 or 2000 on AUX1* — Initially, when the drone in switched on, the motors must be disabled for safety. When the drone is ready for flight, the pilot must enable the motors and this is referred to as Arming the drone.

- **Altitude Hold** — *Range: 0 or 2000 on AUX2* — This control toggles between manual and auto flight modes. In manual flight mode, the user is responsible for maintaining the altitude by varying the throttle. In auto mode, the drone will hold its current altitude.

- **Pitch Trim** — *Range: Increments of ±0.4 on AUX3* — Adds an offset angle to the pitch axis so that drone is perfectly balanced about this axis.

- **Roll Trim** — *Range: Increments of ±0.4 on AUX4* — Adds an offset angle to the roll axis so that drone is perfectly balanced about this axis.

- **Emergency Stop** — *Range: None* — Immediately disarms the drone and resets all the other controls. Note that the same button is used to terminate the communication and close the COM port.

1.3 Interfacing in Python

The *pygame.joystick* module manages the joystick devices on a computer. Joystick devices include trackballs and video-game-style gamepads, and the module allows the use of multiple *buttons* and *hats*. The complete documentation and usage of this module can be found here. First, we have to initialize the joystick module as follows.

```
1 pygame.init()
2 j = pygame.joystick.Joystick(0)
3 j.init()
```

The Joystick object allows you to get information about the types of controls on a joystick device. Once the device is initialized the pygame event queue will start receiving events about its input. Pygame handles all its event messaging through an event queue. *pygame.event.pump()* handles all this for us. In the program loop, we to make some sort of call to the event queue *(line 4)*.

Now, we can read all the data from the controller as shown below. The *get_numaxes()* and *get_numbuttons()* methods return the number of joystick axes and buttons on the controller respectively.

```
1  def getJoystick():
2      out = [0]*5 + [0]*10 + [0] # List to store output data
3      it = 0 # Iterator
4
5      # Handle event queue
6      pygame.event.pump()
7
8      # Read input from the analog sticks
9      for i in range(0, j.get_numaxes()):
10         out[it] = j.get_axis(i)
11         it += 1
12
13     # Read input from buttons
14     for i in range(0, j.get_numbuttons()):
15         out[it] = j.get_button(i)
16         it += 1
```

```
17
18      # Read input from the hat (D-pad)
19      out[it] = j.get_hat(0)
20
21      # Return the data
22      return out
```

The analog stick axes data are floating point values ranging from -1 to +1. The button outputs are digital values (0 or 1). The hat values are ($\pm$1) for both the axes.

Now that we have the data from the controller, we have to send it to the quadcopter in a format that it understands. We will use the MultiWii Serial Protocol. The *MultiWii.py* provides methods to send frames serially over a COM port. The tutorial of working of MSP is already covered. In Python, we can store the data payload in the form of a list and use the *struct.pack()* method to convert our payload into a byte array. The packing formats are explained here. Create a new MultiWii object and initialize the COM port as follows.

```
1  from MultiWii import *
2
3  drone = MultiWii()
4  drone.start('COM10') # Virtual COM port number
```

We have to now convert the controller data to scale it to ranges as given in *section 1.2*. Note that these ranges are not standard MSP ranges and have been modified as per our requirement. The user can send any custom data on the auxiliary channels. We will use the *MSP_SET_RAW_RC* frame to send our control data. The following is a simple example to do this.

```
1  # List to store converted MSP data to be sent
2  set_raw_rc = [0, 0, 0, 0, 0, 0, 0, 0]
3
4  # Get the input data from the controller
5  js = getJoystick()
6
7  # Increase/Decrease throttle
8  set_raw_rc[0] = int(constrain(set_raw_rc[0] + (js[2])/3.0), 1000, 2000))
```

```
9
10 # Roll angle
11 set_raw_rc[1] = int(js[1] * 60)
12
13 # Pitch angle
14 set_raw_rc[2] = int(js[0] * 60) # Multiply by -60 if axis is inverted
15
16 # Yaw angle
17 set_raw_rc[3] = int(js[4] * 180)
18
19 # AUX1 - Arm drone button
20 set_raw_rc[4] = js[5] * 2000
21
22 # AUX2 - Altitude hold button
23 set_raw_rc[5] = js[8] * 2000
24
25 # Pack eight INT8 elements and send the frame
26 drone.sendFrame(drone.SET_RAW_RC, set_raw_rc, '<8h')
27
28 # Delay for 100ms
29 time.sleep(0.1)
```

Run the above snippet in your program loop (*try-except* in a *while* loop recommended). Finally, before your program ends, make sure you close the COM port and uninitialize the joystick module as follows to prevent the program from hanging.

```
1 pygame.quit()
2 drone.close()
```

The above snippet is just a simple example. You can implement your own controls. The controls can also be made *sticky*, where the joystick axis is used only to increase or decrease values, instead of giving the absolute value of the joystick position. You can even make the joystick controller vibrate for user feedback, if your controller supports it. A CLI in Python was developed to visualize the joystick controls as shown in *Figure 1.2*. This is the *link* for the video demonstration of the joystick controller interfaced with it.

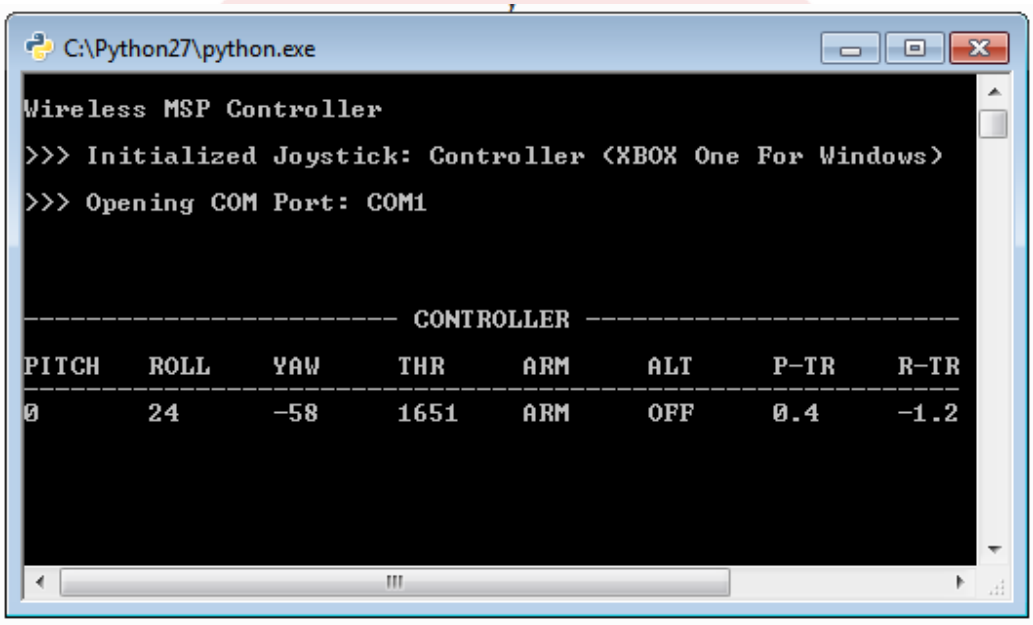

Figure 1.2: A CLI developed in Python for the wireless joystick controller

References

[1] Xbox® One is a trademark of Microsoft Corporation. The Xbox® Logos are trademarks of Microsoft® Corporation.