

Since the beginning of these worksheets, you were probably thinking “when will I be able to directly control my robot?” Well, now is the time! I am going to show you how to use a Python library that will recognise a Bluetooth connected controller. I used an Xbox One controller, the type in Figure 1. You can use any Bluetooth controller as long as it is supported by the library we will be using.



Figure 1 Xbox One Bluetooth controller

Initial Setup

Before beginning to code the necessary functionality to allow us to control our robot with a controller, it is important to configure our Pi to accept the controller. If your Raspberry Pi does not have built in Bluetooth, you can connect a Bluetooth dongle to it via a USB port to allow connections. To be able to use the sticks and buttons on a controller, we are going to use the Approximate Engineering – Input library: [Welcome to Approximate Engineering's Python Game Controller Documentation! — Approximate Engineering - Input 2.6.3 documentation \(approxeng.github.io\)](https://approxeng.github.io) (Oinn, 2021)

As there are many different types of controllers, setup will depend on which controller you have access to. Therefore, I recommend reading the documentation for the library that relates to the controller you plan to use.

Getting the code

You can install the code using pip, you'll have to add a few native libraries first though:

```
$ sudo apt-get install python-dev python-pip gcc  
$ pip3 install approxeng.input
```

Figure 2 Install Approximate Engineering - Input library (Oinn, 2021)

Figure 2 displays the commands you will have to use to download the libraries needed to be able to control your robot with a controller. You are probably wondering where these commands must go. This is where the terminal comes in. To open a terminal, click the terminal icon in the top left corner of your Raspberry Pi's toolbar (Figure 3).



Figure 3 Opening the terminal

After clicking the icon, you will see a window open that looks largely empty with some text inside of it (Figure 4). Most people interact with computers using a graphical user interface (GUI) however us programmers sometimes need more direct access to the system we are using. That is where the terminal comes in! The terminal allows us to type and send commands directly to the computer. Be careful when using it as you could make changes to your system that you do not intend if you use the wrong commands.

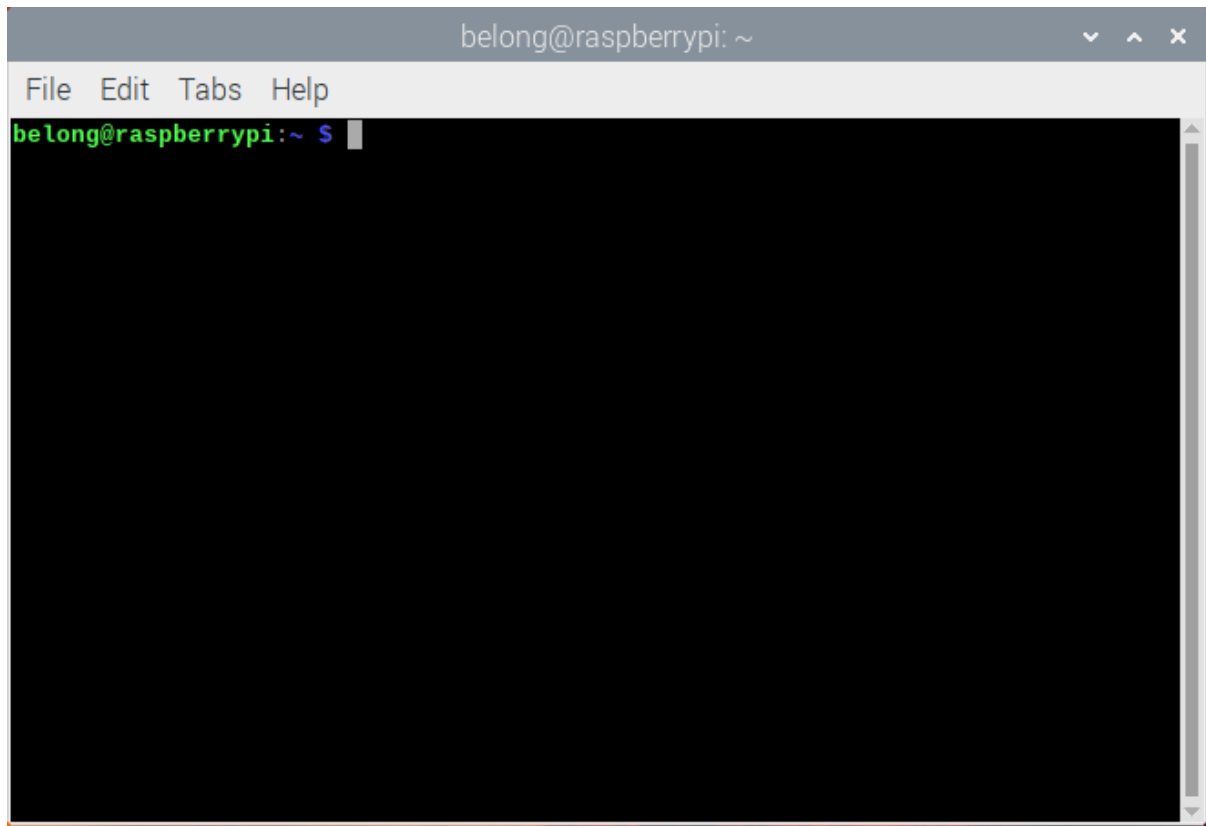
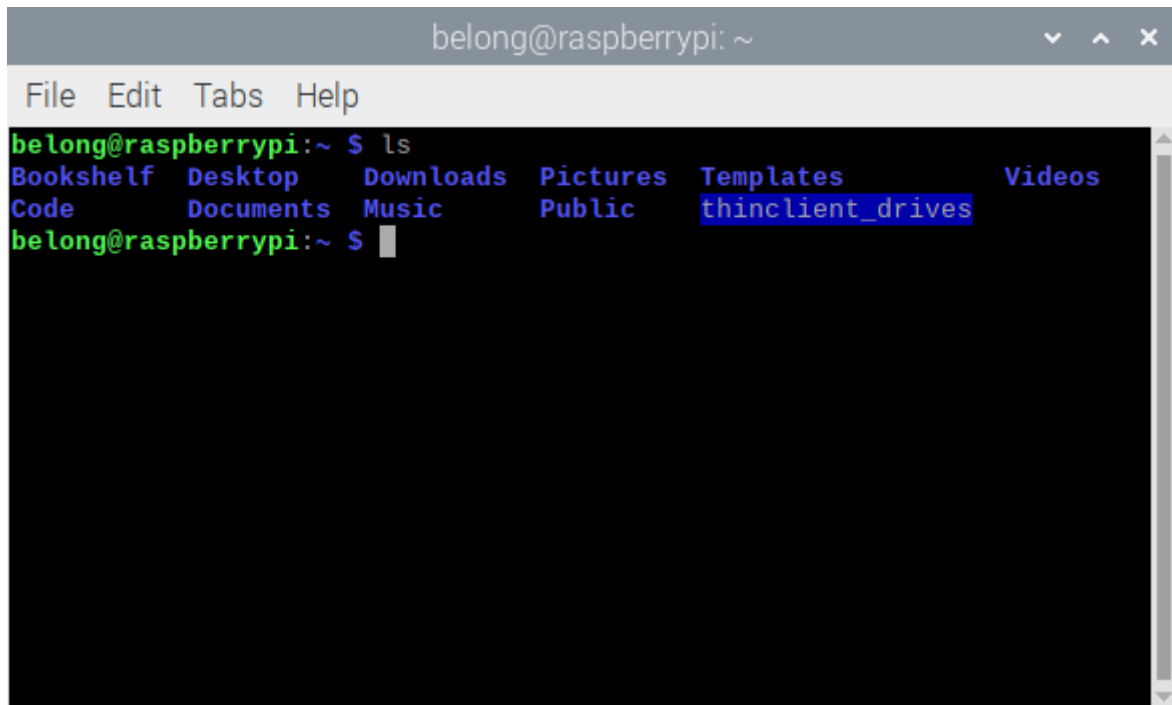


Figure 4 Terminal window

One of the functions of the terminal is to act as a way to navigate your computers file system. When you open it up, you will be placed in your home directory. Think of a directory like a folder on your PC. If you type “ls” and hit enter, the terminal will display the contents of the directory you are currently inside (Figure 5). You can see in Figure 6 that the contents of the home directory, match with what is shown in the home folder.

A terminal window titled 'belong@raspberrypi: ~' with a menu bar containing 'File', 'Edit', 'Tabs', and 'Help'. The terminal shows the command 'ls' being executed, resulting in a color-coded listing of the home directory's contents. The listing is as follows:
belong@raspberrypi:~ \$ ls
Bookshelf Desktop Downloads Pictures Templates Videos
Code Documents Music Public thinclient_drives
The prompt 'belong@raspberrypi:~ \$' is followed by a cursor.

```
belong@raspberrypi:~ $ ls
Bookshelf Desktop Downloads Pictures Templates Videos
Code Documents Music Public thinclient_drives
belong@raspberrypi:~ $
```

Figure 5 Contents of Home Directory

It may seem difficult to understand why we sometimes want to interact with a computer in this way however the terminal is a very powerful tool. It can make certain tasks faster to complete but it is mainly useful because it allows you to do things the GUI of an OS normally does not let you. To learn more about the basic commands needed to use the terminal, please see the following link: [Basics Linux/Unix Commands with Examples & Syntax \(List\) \(guru99.com\)](#) (Brent, 2022).

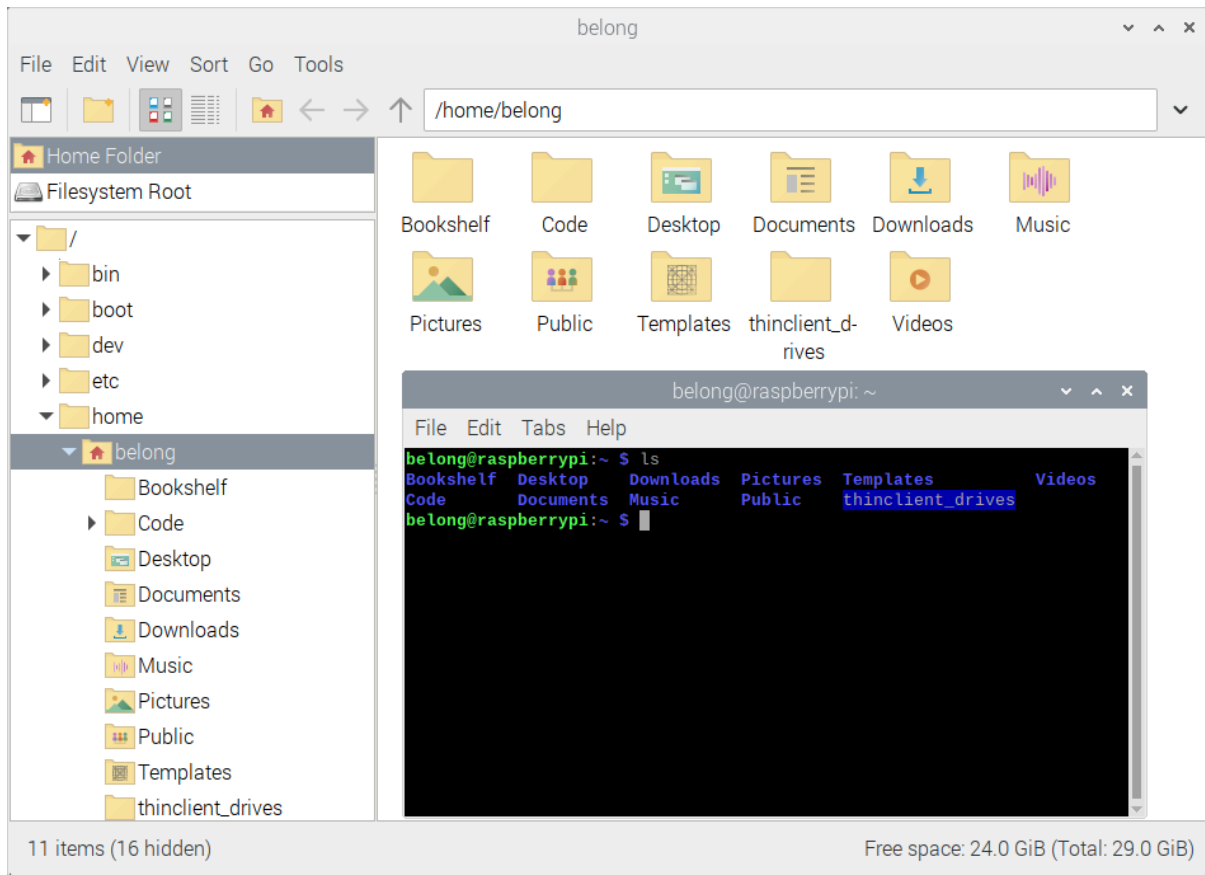


Figure 6 Comparison of terminal and file manager

Connecting your controller

After choosing a controller which is supported by the library, we can connect it to the robot using Bluetooth. The Approximate Engineering documentation walks you through how to connect to a control using the terminal. Follow this link to see how that is done: [Pairing Controllers over Bluetooth — Approximate Engineering - Input 2.6.3 documentation \(approxeng.github.io\)](https://approxeng.github.io/2.6.3_documentation/) (Oinn, 2021). Try to connect your controller this way before attempting to use the GUI like I am going to show you below as your Pi may not have built in Bluetooth.



Figure 7 Select to manage Bluetooth devices

If you were unable to connect your controller using the terminal, you can select to manage your Pi's Bluetooth devices by clicking the Bluetooth icon in the top right-hand corner of your Pi's toolbar (Figure 7). This will result in a small options box appearing, as shown in Figure 8.

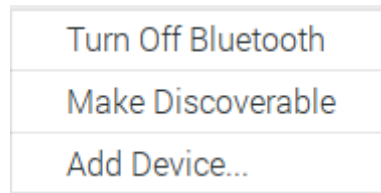


Figure 8 Bluetooth options

If you select to “Add Device...”, a new box will appear. Put your controller into Bluetooth pairing mode and wait for it to appear in the device search box. It may take a few seconds for your device to appear. When it does, select it and click the pair button. If the pair fails, repeat the process until you successfully connect. If you still cannot pair your controller, you may be missing some drivers you need to download to your Pi to get it to connect.

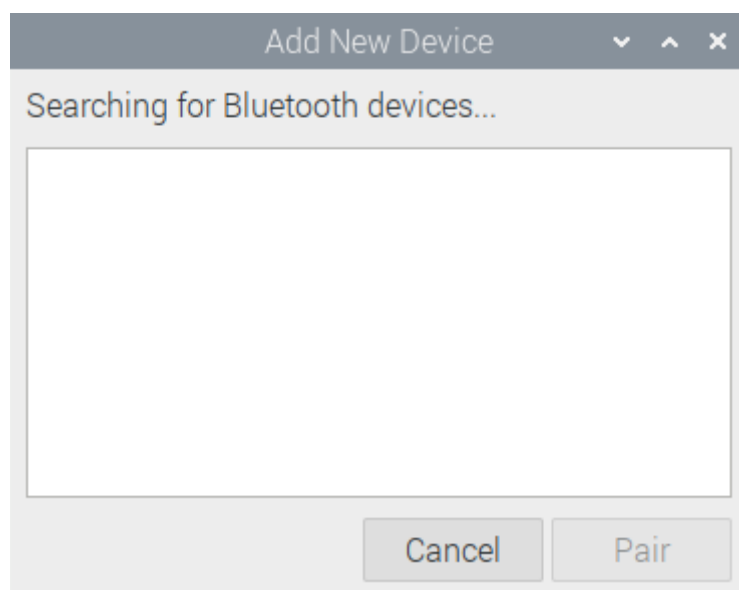


Figure 9 Add New Device

Once you manage to pair a controller, we can begin to create a program that can be used to control the motors.

Controlling your robot

Open Thonny IDE and import the libraries we are going to be using, as shown in Figure 10.

“ControllerResource” is necessary to determine if a controller is connected to the Raspberry Pi and to take its inputs. As how far the control stick is moved on the controller will determine the speed of our robot, we do not need to set any motor constants. Instead, we are going to create a constant called “DEADZONE”. A deadzone is an area in which no inputs are registered. When playing games, every controller takes advantage of having a deadzone to avoid small, accidental inputs throwing a player off their game. It also helps to combat stick drift, a problem where the controller thinks the stick is centred but it is actually not. When you are testing this program, you can experiment with changing the deadzone value to see what feels right.

```

1 # Import necessary Python libraries
2 from approxeng.input.selectbinder import ControllerResource
3 from gpiozero import CamJamKitRobot
4
5 # Controller stick deadzone
6 # Any values between the range -DEADZONE < 0 < DEADZONE will not register
7 # Adjust value according to your controller
8 # Value must be between 0 and 1
9 DEADZONE = 0.3
10
11 # Setting object so its associated functions can be called
12 robot = CamJamKitRobot()

```

Figure 10 Setting up libraries, constants and objects

Next, we are going to create the functions used to power the motors according to input from the left stick of our controller. You can see in Figure 12 how these functions take x-axis and y-axis values as their parameters. This is because the Approximate Engineering library matches the controller sticks movement onto graphical axes with 4 quadrants where x is between -1 and 1 and y is between -1 and 1 (Figure 11).

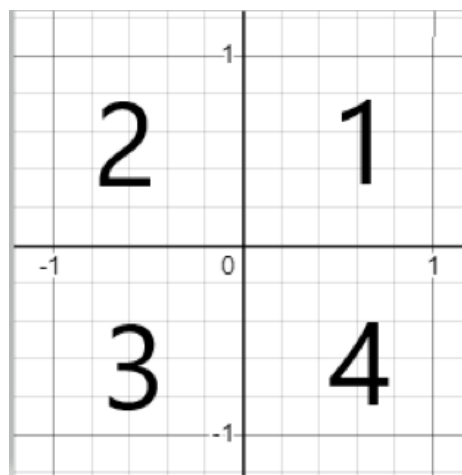


Figure 11 Controller axes

```

12 # Functions to control the motors of the robot
13 # Based on the x-axis and y-axis values from left stick of controller
14 def forward(y):
15     robot.forward(speed = y)
16
17 def backward(y):
18     robot.backward(speed = -y)
19
20 def forwardWithRight(x, y):
21     robot.forward(speed = y, curve_right = x)
22
23 def forwardWithLeft(x, y):
24     robot.forward(speed = y, curve_left = -x)
25
26 def backwardWithRight(x, y):
27     robot.backward(speed = -y, curve_right = x)
28
29 def backwardWithLeft(x, y):
30     robot.backward(speed = -y, curve_left = -x)
31
32 def left(x):
33     robot.left()
34
35 def right(x):
36     robot.right()
37
38 def stop():
39     robot.stop()

```

Figure 12 Motor functions

The functions in Figure 12 map to the graph in Figure 11. The “forward” and “backward” functions are mapped along the y-axis, allowing the robot to move forward or backwards without turning. The “forwardWithRight” function maps to quadrant 1 and the “forwardWithLeft” function maps to quadrant 2. Quadrant 3 maps to “backwardWithLeft” and quadrant 4 maps to “backwardWithRight”. The “left” and “right” functions are mapped along the x-axis itself, allowing the robot to turn in place. This may sound a bit complicated but really you just have to imagine how you move the control stick from the centre will place the sticks position in one of the four quadrants or along the x/y axis of the graph in Figure 11. For example, if you want your robot to move up and to the right, you will push the stick in that direction, placing it in quadrant 1. This will trigger the “forwardWithRight” function, moving the robot.

Now you understand how the control sticks movements match to the created functions, it’s time to start our main event loop (Figure 13). We query the “controller” object to get the x and y values of the left stick so that they can be used to determine where the stick is moved to, as shown in Figure 14. These values can be negative if the stick is being pulled down so that is why some of the functions in Figure 12 use negative versions of the values to flip them back to a positive. This is because speed can only be between 0 and 1 when used with the motor functions.


```

41 with ControllerResource() as controller:
42     while controller.connected:
43         # Using "try" with "except" helps handle any errors that occur
44         try:
45             # Get the x-axis and y-axis values from the left stick of connected controller
46             leftStickX = controller["lx"]
47             leftStickY = controller["ly"]

```

Figure 13 Starting main event loop

The final “else” statement in Figure 14 ensures that the robot will stay still if none of the statements above it are triggered.

```

49 # Determine where stick is moved to
50
51 if leftStickY > DEADZONE and (0 <= leftStickX < DEADZONE or -DEADZONE < leftStickX <= 0):
52     forward(leftStickY)
53
54 elif leftStickY < -DEADZONE and (0 <= leftStickX < DEADZONE or -DEADZONE < leftStickX <= 0):
55     backward(leftStickY)
56
57 elif leftStickX >= 0 and leftStickY > DEADZONE:
58     forwardWithRight(leftStickX, leftStickY)
59
60 elif leftStickX <= 0 and leftStickY > DEADZONE:
61     forwardWithLeft(leftStickX, leftStickY)
62
63 elif leftStickX >= 0 and leftStickY < -DEADZONE:
64     backwardWithRight(leftStickX, leftStickY)
65
66 elif leftStickX <= 0 and leftStickY < -DEADZONE:
67     backwardWithLeft(leftStickX, leftStickY)
68
69 elif leftStickX < -DEADZONE and (0 <= leftStickY < DEADZONE or -DEADZONE < leftStickY <= 0):
70     left(leftStickX)
71
72 elif leftStickX > DEADZONE and (0 <= leftStickY < DEADZONE or -DEADZONE < leftStickY <= 0):
73     right(leftStickX)
74
75 # If no stick movement is detected, stop the motors
76 else:
77     stop()

```

Figure 14 Determine where stick is moved

Finally, we end our program by adding an “except” block so we can quit the program along with some “print” statements to send the x and y values from the controllers left stick to the Shell (Figure 15). As “except” blocks only trigger when their exception is raised, the “print” statements will output text to the Shell until an exception is raised. This happens even though they come after the “except” block in the code. It is important to recognise that your code won’t always run from beginning to end like it reads. It is useful to output the left stick values in case you need to debug your program.

```

79 # Press ctrl + c to stop the motors and terminate the program
80 except KeyboardInterrupt:
81     stop()
82     quit()
83
84 # Output the x-axis and y-axis values
85 # str() converts the values into a string so that print() can output them.
86 print("y: " + str(leftStickY))
87 print("x: " + str(leftStickX))
88

```

Figure 15 End program

Challenge

- Using the Approximate Engineering documentation as a guide, perhaps you could add button controls to your code? You could potentially insert your other programs into your controller code and trigger them by pressing different buttons.

END OF WORKSHEET 5

Reference List

Brent, M., 2022. *Basics Linux/Unix Commands with Examples & Syntax (List)*. [Online]
Available at: <https://www.guru99.com/must-know-linux-commands.html>
[Accessed 21 04 2022].

Oinn, T., 2021. *Welcome to Approximate Engineering's Python Game Controller Documentation!*. [Online]
Available at: <https://approxeng.github.io/approxeng.input/>
[Accessed 18 04 2022].