

Modular Robotics Kit

1. Introduction

1.1 Getting started with Raspberry Pi

1.2 Installing Raspberry Pi operating system (Raspbian)

1.3 Set up remote connection

1.4 Command Line and Python

1.5 Raspberry Pi Camera Tutorial

1.6 Others

1.6.1 Installing Raspberry Pi remotely

2. Assembly & Demo Codes

2.1 Adept DarkPaw Spiderbot Tutorial

2.2 Robot Assembly

2.3 Robot Demo code

2.3.1 Using Robot API

2.3.2 Understanding Robot Gait (optional)

3. Exercises

3.1 Obstacle detection & Avoidance

3.1.1 Demo code

3.1.2 Exercise 1: roam avoiding obstacles

3.2 Hand Gesture recognition

3.2.1 Demo code

3.2.2 Exercise 2: Training your own model

3.2.3 Exercise 3: Emergency reaction to hand gestures

3.3 Image Recognition

3.3.1 Demo code

3.3.2 Exercise 4: Locating target objects

4. Further project ideas

4.1.1 Plants finding bot

4.1.2 Rough terrain bot

4.1.3 Room mapping bot

Appendix

I Common Terminal Commands

II Tutorials and references

1 Introduction

1.1 Getting started with Raspberry Pi

Raspberry Pi is a small credit-card sized computer that can do anything a you can expect a desktop computer to do. We are using it to power the quadruped robot.

Required components for your Raspberry Pi:

- USB-C charging cable
- Micro SD card (at least 32GB is recommended)

If you are new to Raspberry Pi, it is recommended that when setting up you get:

- Keyboard
- Mouse
- monitor (Need HDMI to mini HDMI cable since Raspberry Pi 4 has mini HDMI ports)
- USB MicroSD card reader if your computer doesn't have a microSD slot

After you've finished setting up the connection you don't need have them anymore.

1.2 Installing Raspberry Pi operating system (Raspbian)

You will need to install an operating system on the Raspberry Pi first. We recommend installing the Raspberry Pi OS which is the official supported operating system. All the further instructions in this documentation will also assume this OS.

1. Download the Raspberry Pi Imager from: <https://www.raspberrypi.org/software/>

Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. [Watch our 45-second video](#) to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Windows](#)


[Download for macOS](#)

[Download for Ubuntu for x86](#)

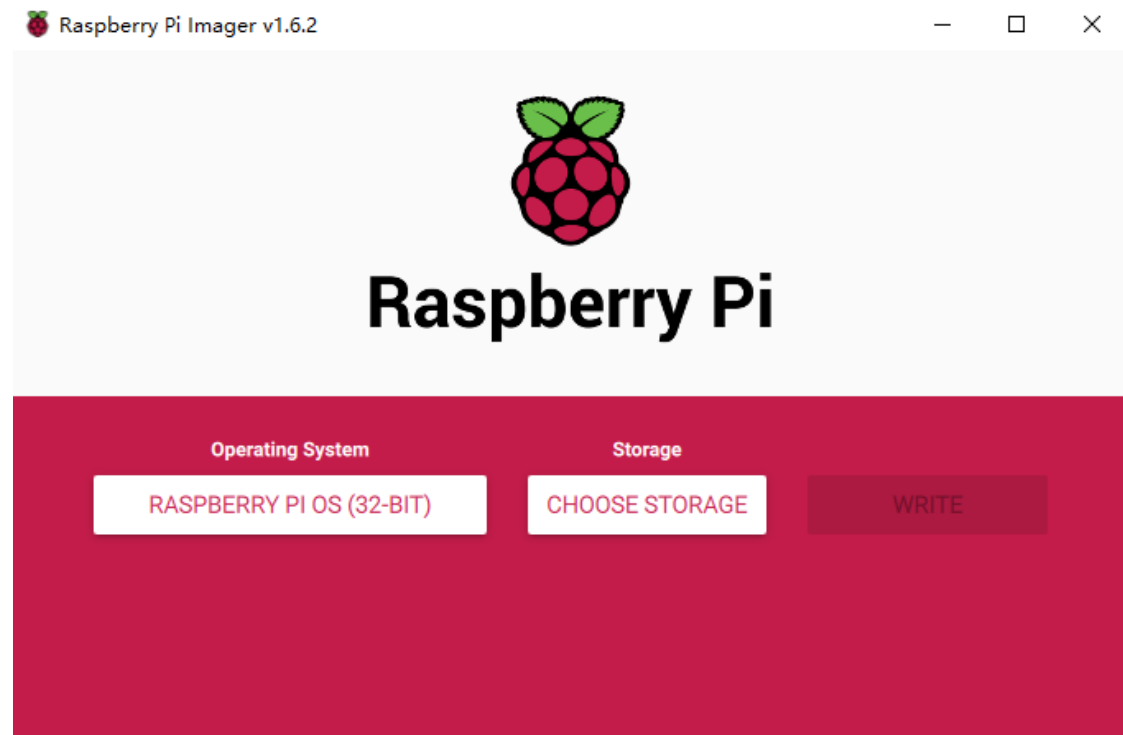
To install on **Raspberry Pi OS**, type

```
sudo apt install rpi-imager
```

in a Terminal window.



2. Run the Raspberry Pi Imager, choose Raspberry Pi OS and choose the microSD card storage. Warning: the imager will overwrite any data on the storage, please make sure you choose the right storage location and that it is empty.



3. Connect the mouse, screen and keyboard to the Raspberry Pi. Plug in the power supply. The Pi will automatically power on when connected to power supply. Warning: cutting out the power may damage the Pi or corrupt data on the SD card, so make sure you are ready to carry on when you plug in the power supply
4. After you plug in the power supply. The operating system will be installed automatically. Follow the instructions and you will be all set.

1.3 Setup remote connection

Even though you can run the your Pi as an individual computer, it is a much better option to run it headless via remote connection. We will talk about two ways to connect to your Raspberry Pi remotely.

SSH Connection:

First of all ensure your Pi is connected to your local network. If you're not connected yet, click on the button on the top right to find your network. Activate the terminal by Ctrl+Alt+T.

To find your IP address, type:

```
$ ifconfig
```

Note: do not include the \$ sign, it indicates typing a command line command. For the rest of the tutorial, all the command line for Pi assumes Raspberry Pi OS and all the command line for pc assumes Windows. An example of the address is shown below:

```
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.137.134 netmask 255.255.255.0 broadcast 192.168.137.255
    inet6 fe80::da71:fee0:82f6:1c96 prefixlen 64 scopeid 0x20<link>
    ether 00:9a:90:00:4c:5d txqueuelen 1000 (Ethernet)
    RX packets 15244 bytes 16719863 (15.9 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 8935 bytes 8507574 (8.1 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

pi@raspberrypi:~$
```

Next up, type

```
$ sudo raspi-config
```

and then choose interface options. Enable SSH connection.

On Windows, you will need an SSH client. PuTTY is recommended: <https://www.putty.org/>

For file transfer, WinSCP is recommended: <https://winscp.net/eng/index.php>

On Mac or Linux, type:

```
$ ssh pi@192.168.XX.XXX
```

where you replace Xs with your IP address.

Remote Desktop Application:

Open the terminal on your Pi and type

```
$ sudo apt-get install xrdp
```

Then open Remote Desktop on your computer. Connect by IP address and enter your username and password. (By default they are **pi** and **raspberrypi**)

1.4 Command Line and Python

Command Line:

To use your Pi, you can either use a graphic interface or via the terminal. It would always be good to learn some of the most commonly used commands. The following section would walk you through the basics of command line in a Linux system.

1. To open a new terminal in graphical interface, click the tab on the top left or use the hotkey **Ctrl+Alt+T**.

2. To show current directory:

```
$ pwd
```

It should be showing `"/home/pi"`.

3. To list all files and folders in current directory:

```
$ ls
```

4. To make a new directory called Test, and check that it exists:

```
$ mkdir
```

```
$ ls
```

5. Change directory into the "Test", and check that you are in the "Test" directory:

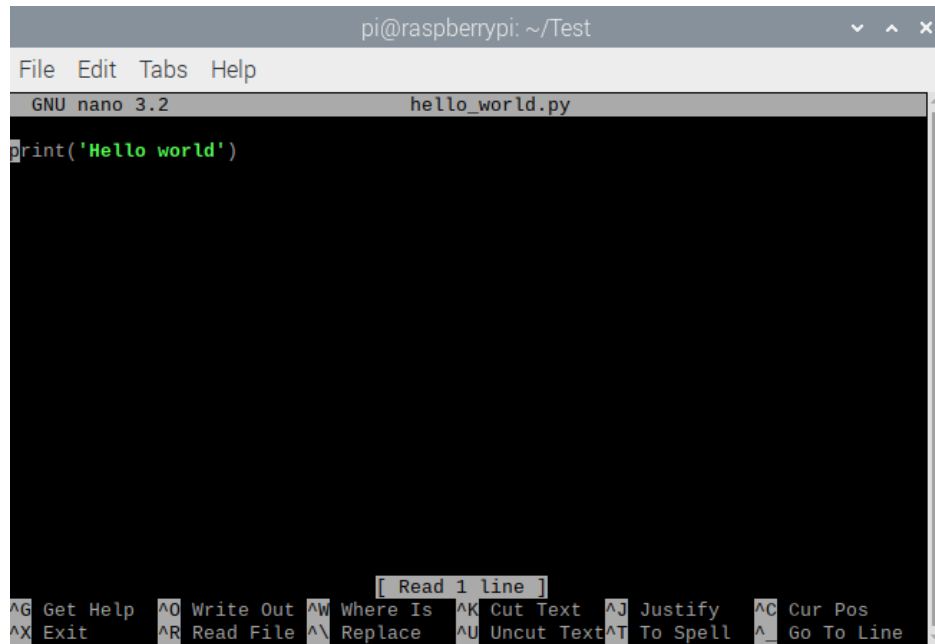
```
$ cd Test
```

```
$ pwd
```

6. Create a python file using the linux command line text editor, Nano:

```
$ nano hello_world.py
```

You should get a graphical interface as shown below. Type in **print ('Hello world')** and press **Ctrl + O** followed by Enter to save to the file (all options are listed at the bottom). Press **Ctrl + X** to exit.



7. To run the code:

```
$ python3 hello_world.py
```

8. To remove the directory along with the python file:

```
$ cd ~
```

```
$ rm -r Test
```

Python Libraries and Virtualenv:

You would need to use different python libraries for your project. Sometimes, you would need to use different versions of Python and libraries to make them compatible, virtual environment would help keep them tidy.

1. In a new terminal, type:

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

This will ensure your pi is up to date.

2. Install Virtualenv with pip, Python's package manager:

```
$ sudo pip3 install virtualenv
```

3. Once finished, create a new virtual environment:

```
$ virtualenv npctest
```

4. Then activate the virtual environment

```
$ source npctest/bin/activate
```

If successful, you will see (npctest) in front of pi@raspberrypi in your terminal. That means you are in the virtual environment.

5. In this new virtual environment, without installing more libraries, you can only use

standard libraries:

```
$ python3
```

```
>>>import numpy as np
```

This should throw an error as numpy is not yet installed in the virtual environment.

6. Exit python shell, and install a specific version of numpy:

```
>>>exit()
```

```
$ pip3 install numpy==1.14.5
```

7. Try it again and check the version:

```
$ python3
```

```
>>>import numpy as np
```

```
>>>np.__version__
```

8. Exit the shell and deactivate the virtual environment:

```
$ deactivate
```

If you check the numpy version again, it should show a different version.

9. To delete the virtual environment:

```
$ rm -r np-test
```

You may still be wondering why we bother going through all those processes, when Numpy is in fact, already installed on the Pi. The real benefit comes when your projects require different versions of the same library to be compatible with each other, or even different versions of the Python Interpreter, for some cases. Especially for one-time use libraries, using a virtual environment would be much neater than installing it on your system path.

1.5 Raspberry Pi Camera Tutorial

You've learnt how to navigate with command line and install python packages in the previous section. Now you will learn how to use some of the libraries.

We'll be working with Python through out the tutorial. Your Pi will come installed with Python and an IDE. If you are new to Python, here's a link to a Python crash course to get you started:

<https://pythonprogramming.net/python-fundamental-tutorials/>

The Raspberry Pi Camera module is a PCB mounting connected by a ribbon cable. Please shutdown your Pi before plugging in the camera module.

To get the demo code, open a new terminal and type

```
$ git clone https://github.com/JieruiShi/Modular-Robotics-Kit.git
```

Once that is done, you should be able to see a new directory named Tutorials.

Before running the test code, make sure camera port is enabled:

```
$ sudo raspi-config
```

The two script, pictureCapture.py and videoCapture.py, will use the camera to take a picture and a ten second video. You can run them in the Thonny IDE.

Note that the start_preview() function in the script only works if your Pi is connected to a monitor. For more on picamera, refer to: <https://picamera.readthedocs.io/>

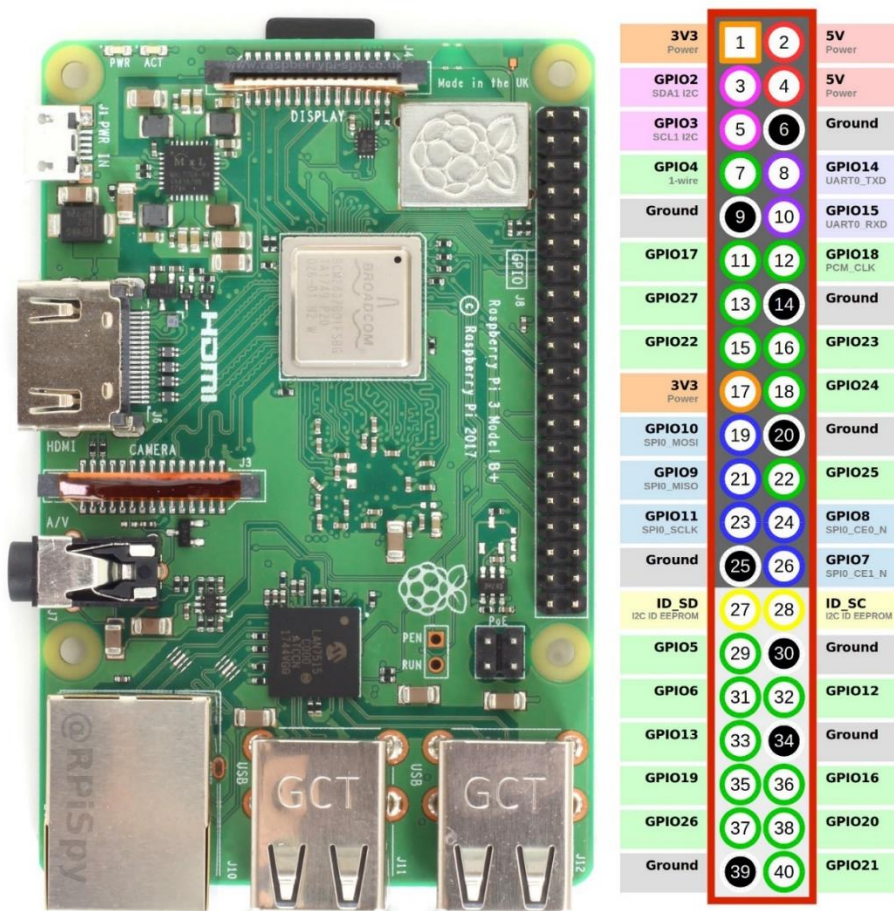
1.6 Others

1.6.1 Raspberry Pi GPIO pins

It is not required to understand how GPIO works for you to program the robot, but it will be helpful if you were to do some Raspberry Pi projects yourself. You would need to prepare yourself:

- some jumper wires (female headers on at least one side)
- Resistors
- Breadboard
- Electric components you would like to test (LEDs, motors, buzzer, ultrasonic sensors, IR linesensors)

GPIO pins stands for general purpose input output pins. They can be used to take input signals or give output signals. You can even use the Raspberry Pi GPIO pins to supply power. Below is a configuration diagram of all the pins. Note that Raspberry Pi does not have analog input pins, which you may expect if you have been using Arduino. But you can always use a ADC converter. Refer to the Gpiozero libraries for instructions on how to control the pins with Python: <https://gpiozero.readthedocs.io/>



2 Assembly & Demo Codes

2.1 Adeept Darkpaw Spiderbot Tutorial

In this kit, we use the quadraped spiderbot kit from Adeept. Here's the link to their tutorial for the kit: <https://www.adeept.com/learn/detail-38.html>

It is strongly recommended that you go through their tutorial before assembling the robot, so you are confident that you know how each individual component works. The robot is designed to be quite compact, so once you finish assembling, it wouldn't be easy to remove the individual components.

Here are some of the main parts of the tutorial:

Installing dependent libraries:

In a new terminal on Pi, type

```
$ sudo git clone https://github.com/adeept/adeept_darkpaw.git
```

```
$ sudo python3 adeept_darkpaw/setup.py
```

This will download the code for using the robot and install dependent libraries. You will notice that there is a **startup.sh** script under **/home/pi** which will run **webserver.py** upon every startup of the pi.

Using Servos:

This part of the tutorial correspond to: <https://www.adeept.com/learn/tutorial-68.html>

Servo is a rotary actuator which allows for precise control of position output.

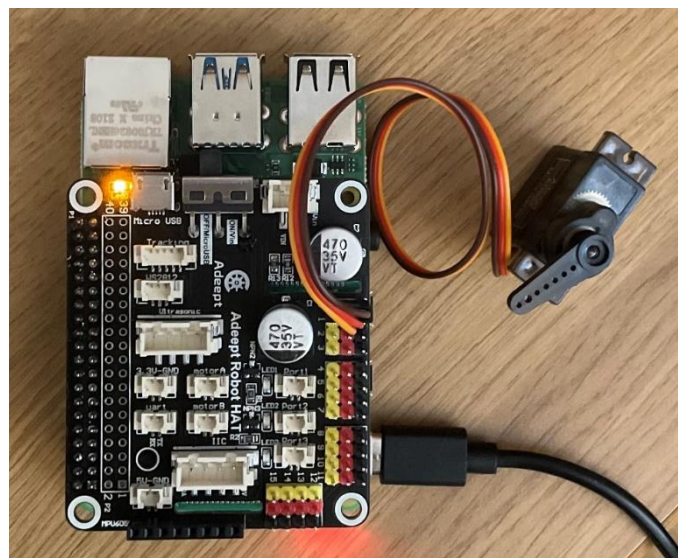
In a new terminal, type:

```
$ python3 "Pi/2 Assembly & Demo Codes/2.1 Servos/servo1.py"
```

```
$ python3 "Pi/2 Assembly & Demo Codes/2.1 Servos/servo2.py"
```

To run the code in two different ways.

Setup:



The demo code changes the pwm signal in port 0.

Set_pwm() method will be used most in the following tutorials. Here's an example of it's use:

```
import Adafruit_PCA9685
pwm = Adafruit_PCA9685.PCA9685()
pwm.set_pwm_freq(50)
pwm.set_pwm(0, 0, 300)
```

The three parameters for set_pwm() method are port number, offset and PWM duty cycle, with the third parameter being a value between 100 and 560, and corresponding to 0 to 180 degrees. (Note: The servo may not respond to values close to 100 and 560, as a general rule of thumb, try to keep the value between 150 and 500).

Also, do note that it takes time for the servo to reach the expected angle with set_pwm(), while the next line of code will be executed immediately after the signal is sent, not after the servo reaches the expected angle. So you would often need a time.sleep() function after set_pwm() to ensure the servo has reached its expected output position.

More about Servo Driver:

If you wondered why there are sixteen pwm pins available on the shield, check out the Adafruit PCA9685: <https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all>

You may also want to go through the exercises in the next section just with the camera before you assemble.

2.2 Robot Assembly

Before Assembling:

Before you assemble, make sure you go through the precautions:

<https://www.adeept.com/learn/tutorial-76.html>

If you've followed the tutorial, you would have installed the code on your Pi for controlling the robot. WebServer.py is a script which will run on powering on your Pi. It will move your Servos connected to the pins to the middle position, it is equivalent to running: pwm.set_all_freq (0,300).

Whenever you are mounting a servo horn onto the servo, keep the servo attached to the hat so you are confident the servo is at the right angle.

While Assembling:

Assembly tutorial: <https://www.adeept.com/learn/tutorial-77.html>

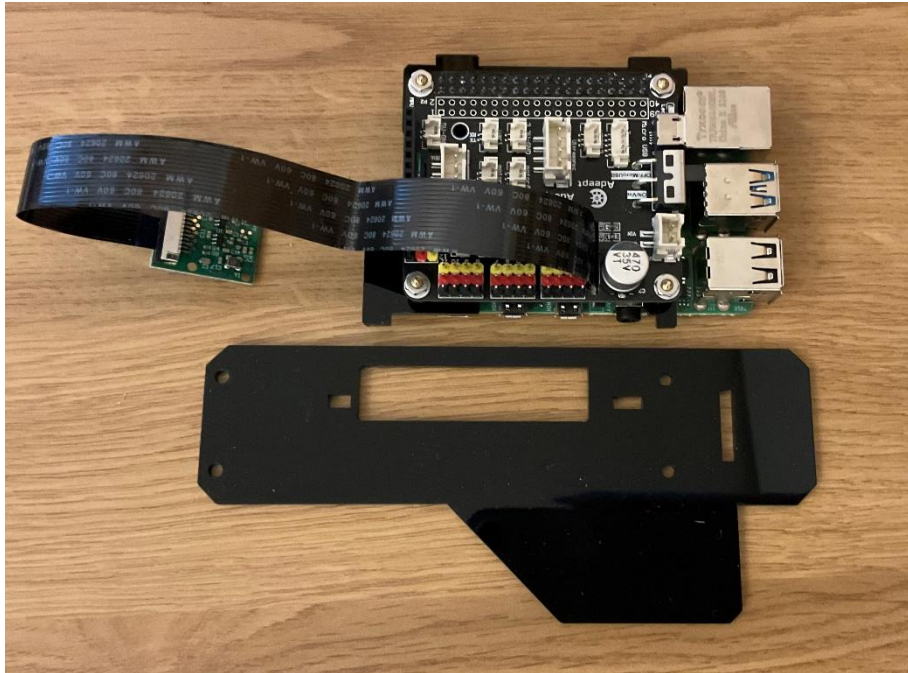
Some tips on assembling:

Before assembling the HAT and support piece to your Pi, make sure the camera is connected

to the camera port properly (as shown below). It would be very hard to adjust the connection once it is fixed.

It would be much more helpful to take off the piece shown below from the top of the Pi when connecting the servos and other components to the HAT.

When connecting joints on the legs and other moving component, don't screw them too tight, as it will affect the performance. When screwing servos to the legs, Try to fix them as tightly as possible, as the screws would carry the load when the spiderbot is walking.



Testing:

Web controller tutorial: <https://www.aadept.com/learn/tutorial-82.html>

It would be good to do some testing after you finished assembling. You don't have to mount the cover on immediately after you've finished. It will make it easier when you have to make changes. You can either turn on the Pi by powering it using the USB as normal or using the battery supply. However, do not use both at the same time. The lithium ion batteries that comes with the robot cannot last too long, so it is recommended that when debugging, use the power supply from the cable. The lithium ion batteries need to be charged with the lithium ion batteries that comes with it.

There is a startup script that will run webServer.py. If it doesn't run, type the following in terminal:

```
$ sudo killall python3
```

```
$ sudo python3 aadept_darkpaw/server/webServer.py
```

When it is running, go to your computer which should be in the same local network as your Pi. Type your Pi's IP address + ':5000' in the browser.

(Example: <http://192.168.XXX.XXX:5000/>)

Fine tune the Servo angles by adjusting in the PWM INIT SET. Always test moving the robot afterwards when you are confident that the angles are correct. If the angles are not optimal,

some of the actions will cause the robot to topple over. Pressing the SETPWM will update the values to SpiderG.py file.

If the servos are out of the correcting range, you might have to disassemble the legs to correct the servo horn position on the servo. Make sure this time to do it while it is connected to the HAT.

2.3 Robot Demo Code

2.3.1 Using Robot API

Controlling Robot via APIs: <https://www.aadept.com/learn/tutorial-80.html>

You should've fine tuned the servo angles in the previous section. The webServer.py will update the SpiderG.py file with the correct initial servo angles. For any code that you write to control robot movement, just copy the SpiderG.py (in "Adept_DarkPaw/server/") file to the same directory as your code and `import SpiderG`.

Here's the demo code that enables the robot to walk in a straight line, then turn around, and enter the rest mode:

```
$ python3 "Pi/2 Assembly & Demo Codes/2.3 Gaits/APIdemo.py"
```

2.3.2 Understanding Robot Gait (optional)

Controlling Robot via APIs: <https://www.aadept.com/learn/tutorial-79.html>

This is the most complicated part of the program, and it is probably not intuitive how the servos coordinate to make the Servos work. But at the end of the day, it is just using the simple `pwm.set_pwm()` function, with the values are generated real-time through some calculation.

If you want to give a go at how to generate your own gait, this crappy gait code may give you some ideas:

```
$ python3 "Pi/2 Assembly & Demo Codes/2.3 Gaits/crappyGait.py"
```

3 Exercises

Set up virtualenv:

Before you run any of the demo codes for the exercises, it is recommended that you set up a virtual environment on your pc and Pi. (Strongly recommended that you do it for pc as there may be conflicts between different versions of Tensorflow and it's dependent libraries). It is okay to not do it for Pi as you are not likely to use it for anything else when it's mounted on the robot.

The following exercises assumes that you run the command line when in virtual environment, from the directory "Modular-Robotics-Kit". Also, assuming a Windows OS for pc and Raspberry Pi OS for Pi.

PC:

Get the demo code from github on pc:

```
$ git clone https://github.com/JieruiShi/Modular-Robotics-Kit.git
```

Navigate into the directory "Modular-Robotics-Kit". Install virtualenv if you haven't already:

```
$ pip install virtualenv
```

Then create a new virtualenv and activate it:

```
$ virtualenv newvenv
```

```
$ newvenv\Scripts\activate
```

On MacOS or Linux system, the above line would be:

```
$ source newvenv/bin/activate
```

Once you are in the virtual environment, (newvenv) will be shown in the terminal, install all dependent libraries for this tutorial:

```
$ pip install opencv-python mediapipe tensorflow tensorflow-gpu  
tensorflow-datasets sklearn
```

Pi:

On Pi, navigate into the same directory, Modular-Robotics-Kit. Install the virtualenv library if you haven't already, then create the new virtualenv and activate it:

```
$ virtualenv newvenv
```

```
$ source newvenv/bin/activate
```

Once you are in the virtual environment, install the dependent libraries:

```
$ sudo pip3 install opencv-python mediapipe-rpi4
```

```
$ echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable  
main" | sudo tee /etc/apt/sources.list.d/coral-edgetpu.list
```



```
$ curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo  
apt-key add -  
$ sudo apt-get update  
$ sudo apt-get install python3-tflite-runtime
```

3.1 Obstacle Detection & Avoidance

In this section, you will learn:

- Using pre-trained model to do object detection
- Using APIs for tflite on Raspberry Pi to make predictions
- Using predictions from model to control robot movement

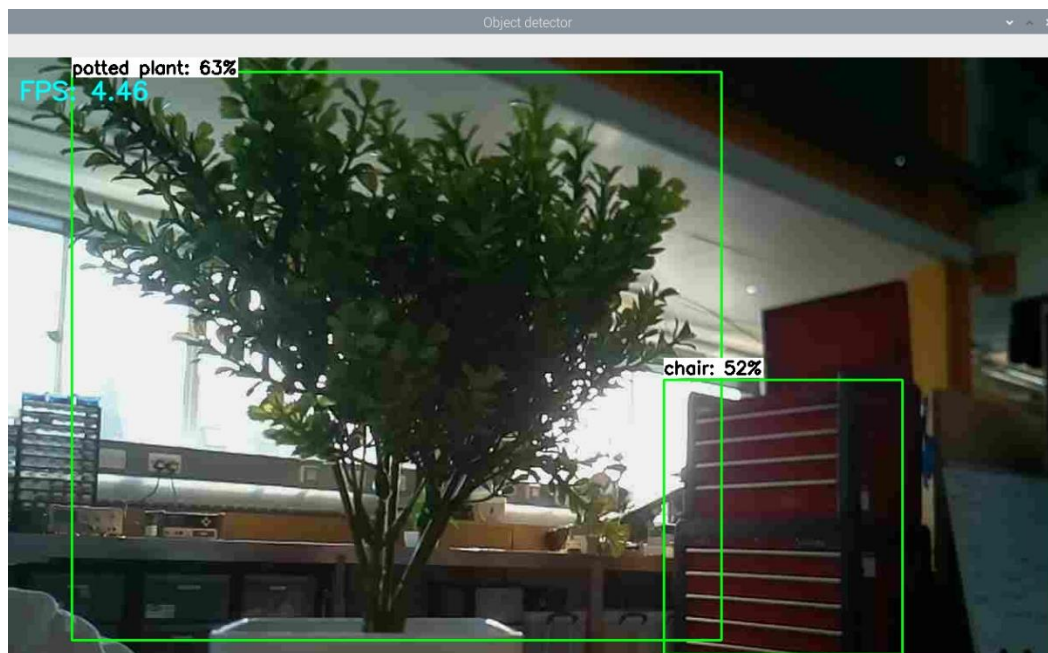
3.1.1 Demo Code

Open a new terminal on Pi, enter the virtual environment, and run the demo code:

```
$ source newvenv/bin/activate
```

```
$ sudo python3 Pi/3\ Exercises/3.1\ Object Detection/objectDetection.py
```

The demo code does object detection and draws bounding boxes over all the object it recognizes.



To understand how the code work, we are going to go into a bit more details into the two main libraries that the code use, OpenCV and Tensorflow. (Documentation page: <https://opencv.org/>, <https://www.tensorflow.org/>)

OpenCV is an open sourced machine learning and computer vision library, it provides functions for real-time computer vision applications. In the demo app, it provides all the function to better visualize the view from camera (unlike `picamera.start_preview()`, using OpenCV for display, it can be seen even using a remote desktop to display). It also provides functions for converting image to a tensor (higher dimensions of arrays) containing numeric values resizing frames to fit model, drawing bounding boxes and putting down text labels.

Tensorflow is the library that can build and train deep neural networks and provide solution to our various machine learning tasks. The Raspberry Pi is not capable of training models on it, but we can deploy trained models on it. We would use Tensorflow Lite as it is the recommended version to run our models on mobile or embedded devices.

TFlite API:

In this section, we will walk through the demo code to teach you how to use Tensorflow Lite to run inference. (Tflite API: https://www.tensorflow.org/api_docs/python/tf/lite/Interpreter)

1. Import the Interpreter class first, then load a tflite model, call `allocate_tensors()` method before starting Inference.

```
from tf.lite_runtime.interpreter import Interpreter

interpreter=Interpreter(model_path='Sample_TFLite_model/detect.tflite')

interpreter.allocate_tensors()
```

2. Get the input and output details of the model, you will be using this information later on to decide the shape of the input and what the output is.

```
input_details = interpreter.get_input_details()

output_details = interpreter.get_output_details()
```

Try running `modelDetails.py` to print out the details of the model. It may not make sense at first glance, but here are some of the key informations:

```
input: [{..., 'index': 175, 'shape': array([ 1, 300, 300, 3])},...]

output: [{..., 'index': 167, 'shape': array([ 1, 10, 4])},...],

{..., 'index': 168, 'shape': array([ 1, 10])},...},

{..., 'index': 169, 'shape': array([ 1, 10])},...},...]
```

There is only one element for input, it represents each frame of the video from the camera. The shape `[1,300,300,3]` means `300 * 300`, pixels, each represented by 3 RGB values. There are multiple elements for output. The first element, which has a shape of `[1,10,4]`, represents 10 bounding boxes, 4 values for the xy coordinates of the box. The second and third element both have the shape `[1,10]` and they represent the class index and confident score of the objects it has detected. The class index will be used to map to the labelmap to get the text label of the object it detected.

3. Send input data into the model using the `set_tensor` method. Then use the `invoke` method to run inference. Note that the `input_data` is obtained by using OpenCV to capture the frame from the camera, convert to numeric values and resizing to the shape `[1,300,300,3]`.

```
interpreter.set_tensor(input_details[0]['index'],input_data)

interpreter.invoke()
```

4. Get the results from inference by using `get_tensor` method. The `output_details` will come in useful for you to retrieve the right information by index.

```
boxes = interpreter.get_tensor(output_details[0]['index'])[0]
```

```
classes = interpreter.get_tensor(output_details[1]['index'])[0]
```

```
scores = interpreter.get_tensor(output_details[2]['index'])[0]
```

Note that step 3 and 4 has to be in the main loop, as you want to be calling them per frame, repeatedly.

3.1.2 Exercise 1: roam avoiding human

If you've gone through the tutorial in section 2, you should be able to use the API to control the spiderbot's movement. You should have also fine tuned the servos, so you should replace the `SpiderG.py` file in directory `"/home/pi/Modular-Robotics-Kit/Pi/3 Exercises/3.1 Object Detection"` with your own version `SpiderG.py`.

Run the `avoidHuman.py` script. It acts as a starting point for this exercise. In the demo code, the robot can only move forwards, and stops when stopping a human for 5 frames in a row. It will start moving again only when it sees no human for 5 frames in a row.

Tasks:

- Try using the robot API to make the robot roam around, instead of just walking in a straight line
- Try using the other actions from the robot control to make it react differently to a human, for example: turn around and walk the other way.

Extension Task (optional):

- Make the robot able to spot obstacles in general, there are two ways to do this, either use an object detection model that recognizes the obstacles in the environment, or use a depth estimation model (https://tfhub.dev/intel/midas/v2_1_small/1)

For more available tflite models, go to: <https://www.tensorflow.org/hub>

3.2 Hand Gesture Recognition

In this section, you will learn:

- To build, train and run your own RNN (Recurrent Neural Network) model
- Using MediaPipe solutions to preprocess video stream
- Converting models to run on Raspberry Pi

What you need:

- A webcam if your computer doesn't already have one
- A pi with camera module (or USB webcam)

This section is hugely influenced by this tutorial:

<https://www.youtube.com/watch?v=doDUihpi6ro>

If you want to learn more about what each part of the code is doing, do check it out.

3.2.1 Demo Code

MediaPipe:

MediaPipe offers cross-platform machine learning solutions for live and streaming media,

<https://google.github.io/mediapipe/>

Whenever you restart the terminal, you have to re-enter the virtual environment. On your pc, enter the virtualenv that you created earlier:

```
$ newvenv\Scripts\activate
```

Run the demo app that make uses of MediaPipe Holistic (tracks human pose, face landmarks and hand):

```
$ python "pc\3.2 Hand Gesture Recognition\mediapipe-demo.py"
```

This will enable you to use mediapipe to track your hand positions.

Hand Gesture Demo:

Run the demo code for Hand Gesture Recognition:

```
$ python "pc\3.2 Hand Gesture Recognition\loadModel.py"
```

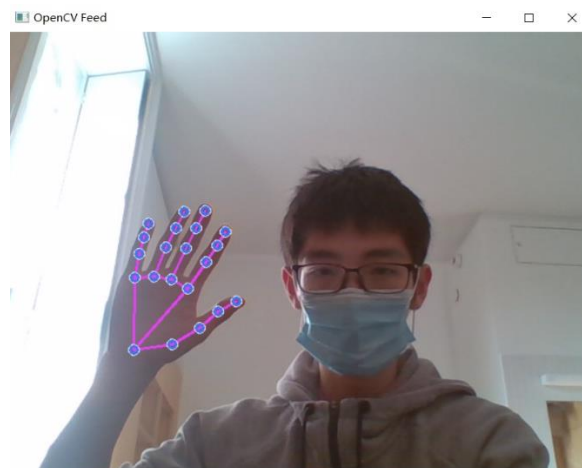
This script will use the pre-trained model "actionDetection" to do real-time prediction on the actions shown to the webcam.

3.2.2 Exercise 2: Training your own model

Raspberry Pi is not powerful enough to train a model, so you should do this on your pc. Let's talk about the model we are going to build first.

In short, a neural network takes in some input and each of the different layers of neurons will go through some computing to extract some features from the previous layers. In the end, it compares the features it extracted to the label that was given and try to fit different features to different labels.

In this model that we are going to train, we are actually not passing in the whole video to the model, but rather the key points extracted from the video. As shown in the previous mediapipe demo, it is really helpful at extracting the key nodes on your hands. The picture below shows the mediapipe demo, tracing 21 nodes on my right hand. Each node would have x,y,z coordinate values associated with them.



To get more accurate results, we are also passing a sequence of frames into the model instead of a single frame of value . By default, each training data has thirty frames, each frame containing $21 \times 3 \times 2$ elements (21 nodes from each hand, 3 coordinates for each node). The final shape of each training data would be (30,126), meaning an array containing 30 arrays which each contains 126 numeric elements, or a 30 by 126 matrix.

Our neural network layers consists of 3 LSTM layers and 3 Dense layers. LSTM stands for Long short-term memory, this layer is useful for tasks when the problem includes a sequence of event. As we are passing a 30 frame video frame worth of data, how the nodes on your hand changes are what matters. Dense layers basically narrows down the number of features from the previous layers, and it is one of the most commonly used layers.

When you run inference (using the model to predict), the output would be a array containing elements between 0 and 1, which add up to 1. For example, [0.1, 0.1, 0.8] as output means the model predicts it is very likely to be action 3.

Instructions:

Collect the data to train for the 3 hand gestures (up,down,stop) to train, by default it would be 30 videos for each action:

```
$ python "pc\3.2 Hand Gesture Recognition\collectData.py"
```

When you've finished collecting the data, you should see a file in the same directory as the collectData.py called "MP_Data", it contains all the data as numpy arrays. Then train the model, it will generate two model files, a directory named actionDetection and a tflite file named actionDetection.tflite:

```
$ python "pc\3.2 Hand Gesture Recognition\trainModel.py"
```

Shown below is the part of the code that builds our model (line 30~36):

```
model = Sequential()
model.add(LSTM(64,return_sequences=True,activation='relu',
input_shape=(sequence_length,126)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
```

Copy the tflite model from pc to Raspberry Pi, into the directory "Pi/3 Exercises/3.2 Hand Gesture Recognition", then run the demo:

```
$ python "Pi\3.2 Hand Gesture Recognition\actionDetection.py"
```

If you want to change the actions that you want to predict, or parameters affecting the training, you can change it in the modelParams.py file before collecting data. Do remember to change the copy of modelParams.py on the Raspberry Pi before running as well.

Extension Task (optional):

- The MediaPipe Holistic model also returns face landmarks, make use of those landmarks to train a face expression prediction model.

3.2.3 Exercise 3: Emergency react to hand gestures

You have seen how the model "reads" your hand gestures and display them on a window produced by opencv in Exercise 2. Instead of displaying them, you can use them to affect how the robot moves. Try running the reactToAction.py script. The logic is fairly straight-forward: it uses the previous prediction values to decide whether the robot should "StandUp" or "StayLow":

```
if predictions[-1] == 0:
    SpiderG.walk("StandUp")
```

```
elif predictions[-1] == 1:  
    SpiderG.walk("StayLow")
```

Tasks:

- Try using your own model that you've trained in exercise 2.
- Try making it so the robot stops when seeing a "stop" action

Extension Task (optional):

- The model requires the past 30 frames of data to make a prediction, which may make the program lag severely. Try optimizing the program. (Hint: when you change the sequence length you use to train the model, don't forget that the video recorded by computer has a much higher framerate than that of the video recorded by Pi)

3.3 Image recognition

In this section, you will learn:

- To run inference on jpg files with a image classification model
- Using transfer learning to build your own image classification model

3.3.1 Demo Code

In the terminal in Pi, change directory into “Modular-Robotics-Kit\3 Exercises\3.3 Image Classifier”, and run image classification with:

```
$ sudo python3 food_classifier.py --imagePath = "Sample/1.jpg"
```

The imagePath is an argument we pass into the script when running it. It contains information about the file path of image you want to do classification of. Other than the sample images provided, try passing your own images into the model as well.

We've seen in previous examples that the OpenCV library provide functions to convert frames from video streams into numpy array data. In this example, however we want to convert static image into numpy arrays to be passed into the model. We will be using the python library for processing images, Pillow.

```
from PIL import Image
img = Image.open(args.imagePath)
imgArray = np.array(img.resize((192,192)))
```

What the code does is simply open the image passed as an argument and resize it into 192 * 192 pixel image (shape [192,192,3]) because the model takes input of shape [1,192,192,3]

3.3.2 Exercise4: Training your own model

You may realize that in the previous example, the model does not always return your expected results. This may be because that what you have in mind does not exists in the labelmap anyway. Depending on the data the model is trained on, it might be biased and not suit to your specific task.

Therefore we will be training a new model using transfer learning. The difference between this exercise and exercise is that instead of building the whole model ourselves (3 LSTM layers, 3 Dense layers), we are going to use a base model, mobilenet_v2. It has complicated layers configured carefully by a group of data scientists and will extract the features from the picture more efficiently. What we need to do is to add the last layer of the network to decide which labels the features extracted from the previous layers should correspond to.

Let's talk a bit more about the model that we built for this case. In the example, we use a dataset which has labeled images of 5 different type of flowers. The base model is

mobilenet_v2, using its weight from 'imagenet', it has various layers that will help us extract features from the images that we can later use to classify our images. We set it to take images with a shape of [160,160,3], (default is [224,224,3]), and output a feature map with shape [5,5,1280]. Next we add a layer GlobalAveragePooling2D first to get an average over the 5x5 spatial location, yielding a featuremap with shape [1280]. Finally we use a Dense(5) layers to get the outpt with shape [5], which is basically an array that gives the confidence level the model predicts for each type of flower.

Instructions:

We are going to train a flower classification model using the public dataset from Kaggle.

Download the dataset from: <https://www.kaggle.com/alxmamaev/flowers-recognition>

Unzip it in the directory "Modular-Robotics-Kit\pc\3.3 Transfer Learning for Image Classification", the root directory should have the name "flowers", it contains 5 subdirectories each containing hundreds of photos of 5 different kind of flowers.

Run the split.py file to split the dataset into train,test and valid:

```
$ python "pc\3.3 Transfer Learning\split.py"
```

You should see a new directory named "flowers-split", it contains the same data, but split into the train, test and valid category. The purpose of this step is that in the next step to load the files into the Tensorflow **tf.data.Dataset** object, it requires the directory structures to be in the following form:

```
path/to/image_dir/
  split_name/  # Ex: 'train'
    label1/   # Ex: 'airplane' or '0015'
      xxx.png
      xxy.png
      xxz.png
    label2/
      xxx.png
      xxy.png
      xxz.png
  split_name/  # Ex: 'test'
  ...
```

Then run trainModel.py:

```
$ python "pc\3.3 Transfer Learning\trainModel.py"
```

Note that for the 3 layers that we used for this model:

```
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')

base_model.trainable = False
```

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(5, activation = 'softmax')
```

We are only changing the weights for the **global_average_layer** and **prediction_layer** when training. This is because we are only using the **base_model** for feature extraction, and we don't want to make any changes to it. If without the line **base_model.trainable = False**, Any small random steps in changing the weights in the model may cause the model to “forget” all it learned.

Finally, copy the tflite model the script returned to “Pi/3 Exercises/3.3 Flower Classification” and run the `flowerClassification.py` to predict for flowers.

Extension Task (optional):

- The object detection model in 3.1 can recognize and track objects in real-time video, but the accuracy may not be that well. In a situation where the robot uses the object detection to find a certain object, try constructing a pipeline that will pass the picture into the image classification model to confirm that it got the right object.

4 Exercises

This section of the documentation contains open-ended exercises that will give you the opportunity to make use of what you've learnt in the previous section in a real-life scenario.

4.1 Plants finding bot

Task Description:

In a fully robotic piloted flower shop, you want to deploy a robot that is capable of locating all the potted plants and by doing image classification, inform your customers of what plants they are.

Hints:

- Locate the rough location of potted plants by Object Detection
- Pass photos of the potted plants when close enough to the image classification pipeline

4.2 Rough terrain bot

Task Description:

You want to deploy an autonomous robot that follows a human on an uneven terrain, and be able to get back whenever it topples. It should be able to produce a map showing the path that it took in the end.

Hints:

- Try calling the different actions of the robot when it topples over, use the Gyroscope to determine if the robot is upright or not.
- Use the gyroscope to produce the map of the route

4.3 Room mapping bot (advanced)

Task Description:

You want to deploy an autonomous robot that roams around in a confined environment, potentially dangerous, and produce a 3D mesh output of the interior of the room.

Hints:

- Try finding a depth estimation model that will return values to form a point cloud of the room.

Appendix

I. Common Terminal Commands

`$ ls`

Lists all files and folders under current work directory.

`$ pwd`

Display path of current work directory.

`$ cd <directory>`

Change directory to <directory>.

`$ cd ..`

Change directory to parent directory.

`$ cd ~`

Change directory to home directory.

`$ sudo shutdown -h now`

Perform system shutdown properly.

`$ sudo shutdown -r now`

Perform system restart.

`$ sudo raspi-config`

Opens the Pi configuration tool.

`$ sudo apt-get update`

Get updates on available packages.

`$ sudo apt-get upgrade`

Install available updates, need to run update command first.

`$ sudo apt-get install <package>`

Install <package>.

`$ sudo killall python3`

Terminate all running python processes.

`$ sudo pip3 install <package>`

Install python packages.

II. Tutorials and References

Useful Tutorials:

Python Absolute beginners: <https://www.python.org/about/gettingstarted/>

Raspberry Pi tutorial: <https://pythonprogramming.net/introduction-raspberry-pi-tutorials/>

Tensorflow Crash course Tutorial: <https://www.youtube.com/watch?v=tPYj3fFJGjk&t=16843s>

Google ML Crash course: <https://developers.google.com/machine-learning/crash-course>

Hand Gesture Recognition Tutorial: <https://www.youtube.com/watch?v=doDUihpj6ro>

Tensorflow Transfer Learning Tutorial:

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/transfer_learning.ipynb

Libraries:

Gpiozero Documentation: <https://gpiozero.readthedocs.io/en/stable/>

Mediapipe: <https://google.github.io/mediapipe/>

Tensorflow Documentation: https://www.tensorflow.org/api_docs/python/tf

OpenCV Documentation: <https://docs.opencv.org/4.5.3/>

Resources:

TensorFlow Hub: <https://www.tensorflow.org/hub>

Kaggle Datasets: <https://www.kaggle.com/datasets>

III. CuR minibot

Cambridge University Robotics Society (CuR) hosts an annual freshers event where participants build a wheeled robot, minibot that is controlled by Arduino.

A simple upgrade can be made on the design by changing the Arduino to a Pi and sensors to a camera. This is suitable for users who want to make their own wheeled robots from scratch (if you really don't like legged robots!)

CAD Design (2020 version):

<https://drive.google.com/file/d/1aW9FOVjDzbReDr6lvF7u-5qywm3ykyJ-/view?usp=sharing>

BOM:

- DC Motors and wheel * 2
- Servo motors * 2
- Breadboard
- AA battery holder
- AA batteries * 4
- 9V battery
- 9V battery connector
- Arduino Uno (Raspberry Pi)
- USB type B (not needed for Pi)
- HC-SR04 sensors (optional for Pi) * 3
- MPU6050 sensors
- TB6612FNG motor driver
- KY-033 Line follower (optional for Pi) * 2
- 3D printed parts (from CAD)
- Laser cutted parts (from CAD)
- Cable ties
- Jumper wires
- Nuts and Screws