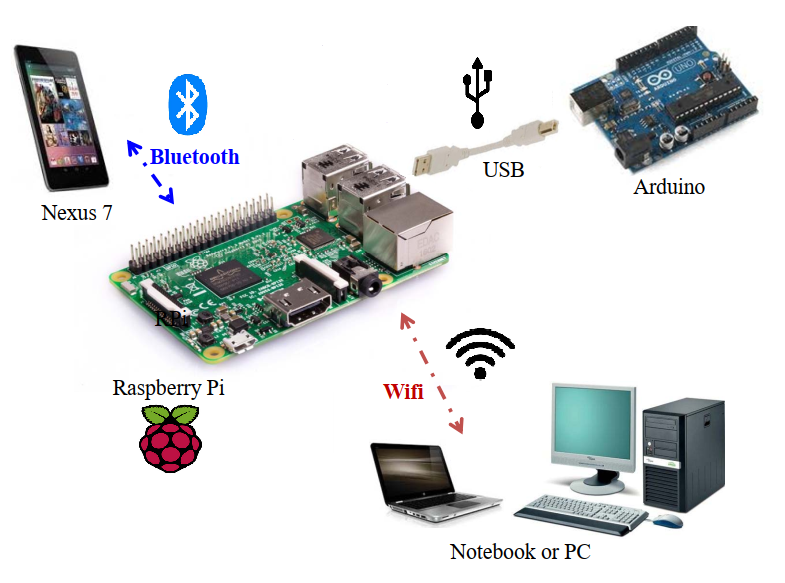
Table of Contents

1. Communication in the Raspberry Pi
   1. Arduino
   2. Android Tablet
   3. PC (Algorithm)
   4. Multithreading
2. Image Recognition
3. **Communication in the Raspberry Pi**



The Raspberry Pi (RPI) is a low-cost credit-card sized computer that runs on Linux operating system. In this project, RPI is acted as a middleman and used for information exchange server between various device via 3 communication protocols: Bluetooth, Serial Connection and TCP/IP Socket connection over WIFI.

* 1. **Arduino**

Communication between Arduino and RPI is Serial Connection.

**Send Message to Arduino**

RPI will convert the string message into ascii format before sending over to Arduino. RPI sends commands from the PC (algorithm) in STRING format to Arduino. Some commands are given below:

|  |  |
| --- | --- |
| **Command** | **Description** |
| S | start calibration at start zone |
| C | calibrate the robot to be in 3x3 box |
| U | angle calibration of the robot to straighten ahead |
| V | pull sensor data |
| W | move forward |
| L | turn left |
| R | turn right |

**Receive Message from Arduino**

RPI will convert the message into utf-8 format after receiving from Arduino. Arduino sends “movement done” to indicate a success in execution of the movement and sensor readings to RPI and forwarded to PC (algorithm) in STRING format. Sensor readings contain 6 integer number for each of the 6 sensors. Each number indicate the number of blocks away from the obstacle. Below is an example of sensor readings:

|  |  |
| --- | --- |
| **Message** | **Description** |
| [3,2,1,3,2,1] | 6 sensor readings |

* 1. **Android Tablet**

Communication between RPI and Android is Bluetooth connection.

**Send Message to Android**

RPI sends information about the arena from PC (algorithm) and image recognition string that is running on RPI in STRING format to android. RPI will format the raw data from PC to the expected string structure for android as shown below. This is to ensure the map is always updated with any obstacle or image discovered on the android tablet. Below are the examples of message:

|  |  |
| --- | --- |
| **Message** | **Description** |
| #im:(id,x,y)(id,x,y) | Image String |
| #grid: string,x,y,degree | MDF String, Robot Position & Robot Orientation |
| exe | Exploration done |

**Receive Message from Android**

RPI will convert the message into utf-8 format after receiving JSON string from Android. Android sends a message to RPI and extracts the content in JSON and format the string if required which later forwarded to PC (algorithm) in STRING format. Below are the examples of message:

**JSON Message Structure**

{

"destination": "pc",

"content": "exs",

"topic": "exploration"

}

|  |  |
| --- | --- |
| **Message** | **Description** |
| waypoint [x,y] | waypoint of the arena |
| beginFastest | start fastest path |
| exs | Start exploration |

* 1. **PC (Algorithm)**

Communication between RPI and PC is TCP/IP Socket connection over WIFI. Using WIFI allows us to enable devices to connect to a message broker, Paho MQTT (Message Queuing Telemetry Transport) to publish messages, and to subscribe to topics and receive published messages. This helps to RPI to manipulate and retrieval of data easily as subscriber will only receive the message from a topic that has been published.

**Paho MQTT**

|  |  |
| --- | --- |
| **Code** | **Description** |
| self.client.subscribe("pc") | subscribe to pc topic |
| publish.single("pc", message, hostname="192.168.30.1") | publish a message in pc topic |
| self.client.message\_callback\_add("pc", self.on\_message\_pc) | when message is received from pc topic, on\_message\_pc will be called |
| def on\_message\_pc() {// To-do code} | Raw message will be processed in this method and send to respective device |

**Send Message to PC**

RPI sends sensor readings from Arduino to PC, commands and waypoint from Android to PC in STRING format. Below are the examples of message:

|  |  |
| --- | --- |
| **Message** | **Description** |
| [3,2,1,3,2,1] | 6 sensor readings |
| exs | Start exploration |
| waypoint [x,y] | waypoint of the arena |
| beginFastest | start fastest path |

**Receive Message from PC**

RPI will convert the message into utf-8 format after receiving JSON string from PC. PC sends a message to RPI and extracts the content in JSON and format the string if required which later forwarded to Arduino or Android in STRING format. Below are the examples of message:

**JSON Message Structure**

{

"destination": "android",

"content": {"position": "(x,y)", "map": "MDF STRING", "orientation": "degree"},

"topic": " update after movement"

}

|  |  |
| --- | --- |
| **Message** | **Description** |
| MDF string,x,y,degree | MDF String, Robot Position & Robot Orientation |
| exe | Exploration done |
| V | pull sensor data |
| S | start calibration at start zone |
| C | calibrate the robot to be in 3x3 box |
| U | angle calibration of the robot to straighten ahead |
| W | move forward |
| L | turn left |
| R | turn right |
| YYYYYJYYYYGGGYYYY | Sequence of movement for fastest path |

* 1. **Multithreading**

Multithreading provides simultaneous execution of two or more devices connected to RPI to maximum utilize the CPU time and ensures less delay as possible. One thread will be allocated to each device. Image recognition is done in RPI and executed in multithreading to ensure the robot to explore the arena and recognise image concurrently.

1. **Image Recognition**
   1. **TensorFlow**

It is an open source platform for machine learning to train an object detection neural network through Google’s TensorFlow object detection API on windows 10. For this MDP, it is necessary to create a new object detector that can locate and identify the set of images on the arena.

* 1. **Requirements**

|  |  |
| --- | --- |
| * TensorFlow 1.5 * TensorFlow-gpu 1.5 * CUDA 9.0 * cuDNN 7.0.5 * Anaconda with Python 3.6 * Protobuf 3.13.0.1 * Pillow 8.0.1 | * Lxml 4.6.1 * Cython 0.29.21 * Contextlib2 0.6.0 * Matplotlib 3.3.2 * Pandas 1.1.3 * Opencv-python 1.0 |

* 1. **Setup Object detection directory**

This directory will be used for image recognition for MDP project. The TensorFlow repository can be downloaded from [1] and [2]. Copy the file from [2] to object\_detection folder.

* 1. **Setup Anaconda virtual environment**

The virtual environment is an isolated workplace for doing various project as different project requires different installation setting. By having different environment, we can work with specific versions of libraries or Python itself without affecting other Python projects. Virtual environments make it easy to cleanly separate different projects and avoid problems with different dependencies and version requirements across components. The detailed steps can be followed as shown in [2] section 2d.

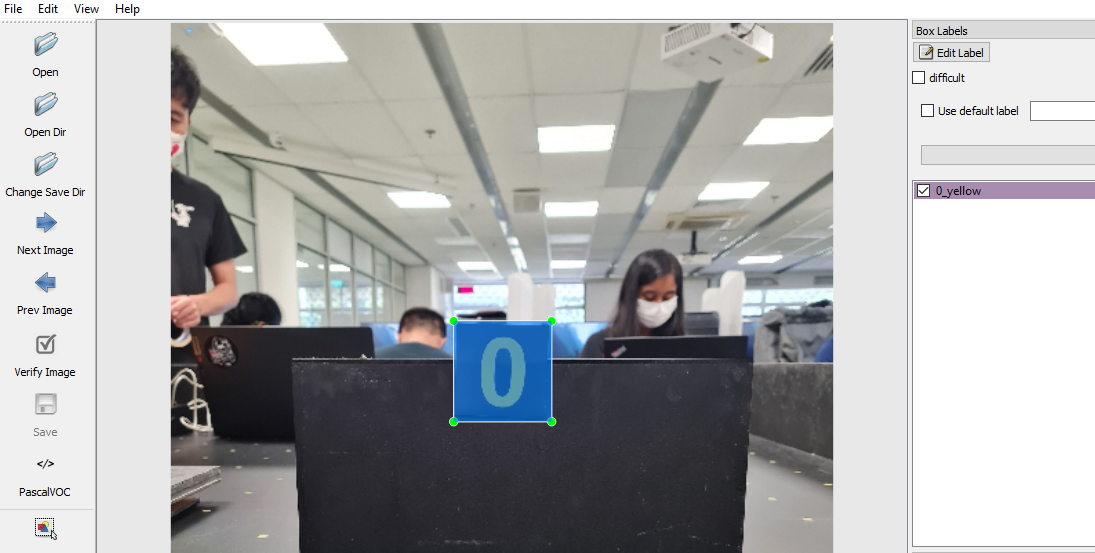
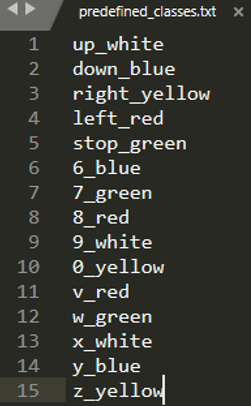
* 1. **Gather and label images**
     1. **Gather images**

TensorFlow requires hundreds of images of an object to train a good detection classifier. To have robust classifier, the training images should have random objects in the image; a variety of different background and lighting conditions; different angle of the image. Total of 659 images has been taken for our training data. Each image should be size of 300 kilobytes or less; should not have resolution higher than 1280 by 720 pixels. The larger the images are, the longer it will take to train the classifier. The 659 images will be split into 80:20 ratio where 80% of the image will be for training data set and 20% will be for test data set. Create test and train folders in object\_detection/images folder and place the images to the folders, respectively. Below are some examples of the training images:

* + 1. **Label images**

Labelling is required to identify the boundary of the training images to be recognised during training. Since we have total of 15 images to be identified in the arena, there will be 15 classes for our classifier as shown below. The software for labelling can be downloaded from this repository [3].



Upon saved, xml file will be auto generated contain coordinates of the bounding rectangles for each object in the image.



* 1. **Choosing a model**

TensorFlow has provided several object detection models which are pre-trained classifiers that have a specific neural network architecture. Some models provide high speed in detection but lower accuracy rate (ssd mobilenet) and low speed in detection but higher accuracy (faster rcnn).

For this MDP, ssd\_mobilenet\_v2\_quantized\_300x300\_coco model will be used to run the object detection on Raspberry Pi such that Raspberry Pi has low computational power than a laptop and ssd mobilenet can achieve satisfactory accuracy rate in detecting the image in our arena. Download the ssd\_mobilenet\_v2\_quantized\_300x300\_coco zip file [4].

* 1. **Generate training data**

Test and Train record file need to be generated that will serve as input data to the TensorFlow training model. The detailed steps can be followed as shown in [2] section 4.

* 1. **Create label map and configure training**
     1. **Create label map**

Label map tells the trainer what each object is by defining a mapping of class names to class ID numbers. The detailed steps can be followed as shown in [2] section 5a.

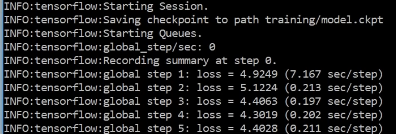
* + 1. **Configure training**

Configure the object detection training pipeline that defines which models and what parameters will be used for training, points to the training images and data. The detailed steps can be followed as shown in [2] section 5b

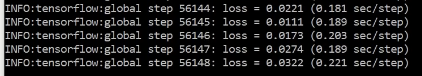
* 1. **Train object detector classifier**

Training starts here. The detailed steps can be followed as shown in [2] section 6.

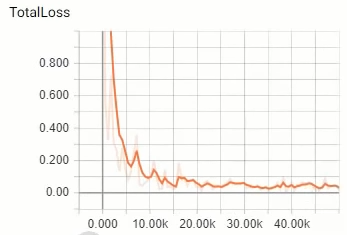
When training begins, it will step through training batches and reporting the loss at each step. At the start, the loss rate is high, and it gets lower as the object detection classifier trains.



After few hours of training, the loss should be around 0.05 to be accurate enough. Terminal the program by press CTRL+C.



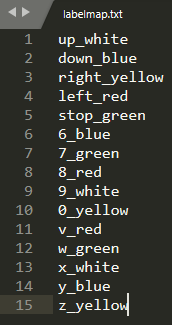
In the Tensorboard, select TotalLoss tab to see the overall loss of the classifier over time. X axis is the loss rate and Y axis is the number of steps.



* 1. **Export inference graph**

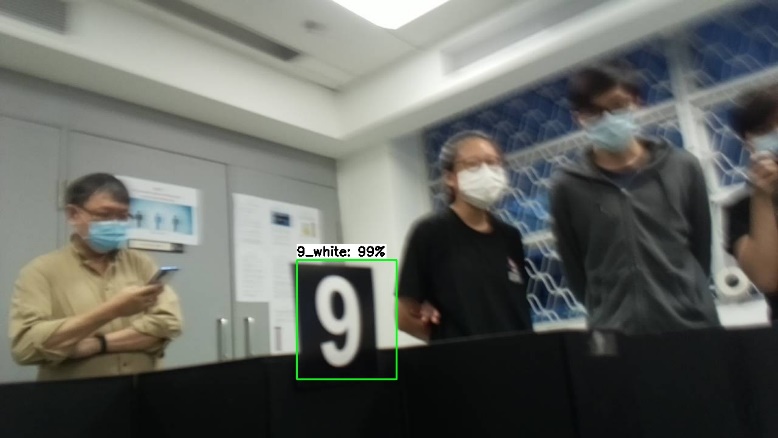
TensorFlow saves training checkpoints every 5 minutes and stores in the training folder, the checkpoint of the highest step count will be used for the object classifier. The detailed steps can be followed as shown in [2] section 7.

Now we must create a folder “tflite\_model” to be copy over to Raspberry Pi. Create a new file labelmap.txt with the content below. Copy detect.tfile and labelmap.txt to the “tflite\_model” folder and put it in a thumb drive.



* 1. **Setup in Raspberry Pi**

For the MDP, I will be using webcam in video mode to capture and recognise the image in real time. For every frame in the video will be recognized. I had set a threshold of 93% accuracy rate to be saved to a folder for collage. The detailed steps can be followed as shown in [6].



* 1. **Reference**

[1] GitHub TensorFlow Repository 🡪 <https://github.com/tensorflow/models>

[2] GitHub EdjeElectronics Repository 🡪 <https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10>

[3] GitHub Labellmg Repository 🡪 <https://github.com/tzutalin/labelImg>

[4] GitHub COCO-trained models Repository 🡪 <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md>

[5] GitHub TensorFlow Lite Setup on Raspberry Pi Repository 🡪 <https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi>

[6] Raspberry Pi TensorFlow setup 🡪 <https://youtu.be/aimSGOAUI8Y>