Chapter 4

Suggestion

4.1 Introduction

This section defines and explains the suggested prototype concerning the required hardware and software components. It firstly introduces the goals of the suggestion before describing the components of the prototype. Finally, the software components are introduced.

4.2 Design goal

This section will give an overview of the whole system. The system will be explained in its context to show how the simulation system using reinforcement learning and deep reinforcement learning in order to find the path to the destination for EV3 robot. It will also describe the requirement specification of prototype as building a software by providing its functional and nonfunctional requirements.

4.2.1 Design Principles

The following principles need to be considered in the suggestion and subsequently in the development of the prototype:

- The simulation of Q-learning which can show the ability in path finding problem.

- A framework can help to train a deep neural network
- The simulation of Deep Q-Network which can use the same hyperparameters with Q-learning in order to compare those two.
- A user interface will provide the visualized result for easier understanding how reinforcement learning agent learning by itself.
- A server can help the robot and computer interact with each other.
- A robot which can follow the path and detect obstacles.

4.2.2 Prototype requirement specification

4.2.2.1 Functional requirement

This system will consist of two parts: one simulation software and EV3 robot. The simulation application will be used to simulate path-finding environments which help the EV3 robot to navigate to the target.
The simulation software will need a flexible input value in size of environment such as $m * n$ matrix A where m is the row number and n is the column number. A matrix will be used as the input of Q-learning and Deep Q-Network algorithm. Then, the users will be able to change the state of environment. Users can select the position starting point, target point and obstacles point. The maximum of starting point and target point is unique and no limitation with obstacles. After that, users can select a button to start training the model. The result will be viewed in real-time which can show how the agent learns by itself in order to solve path-finding problem. Hence, each type of points should appear in different colors so the user can easily distinguish them. The simulation software also provided a server which can send and receive messages between the computer and robot via wireless connection. The mock-up design of simulation software was shown in Figure 4.1.
The prototype robot will be able to send and receive messages from the server. It receives messages which contain the command from the computer as the brain of the robot and then send back messages to the computer information about the current position of the robot. Furthermore, the robot has to detect the environment by sensors in order to follow the path to the target and process the steering

command from computer such as up,down,left,right. The robot also has the ability to detect the obstacles along the way to the target and send back avoidance detection messages to the computer. User interface is not necessary for the robot so everything should be controlled by the computer.

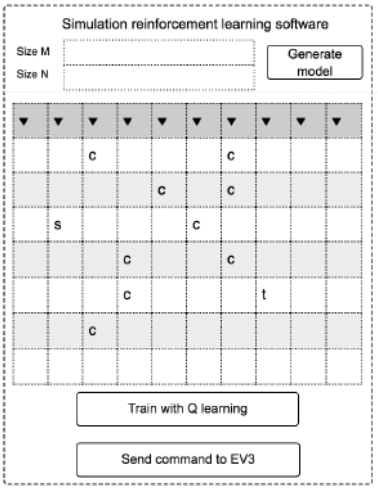


FIGURE 4.1: Mockup design for simulation software

4.2.2.2 Nonfunctional requirement

A functional requirement describes what a software system should do, while nonfunctional requirements place constrains on how the system will do. Following is nonfunctional requirement for the whole system:

- The system needs to use the same design for both Q-learning and Deep Q-network in order to re-use the code from both of them and can even apply more algorithms later
- Frequently used functions should be tested for usability, as should complex and critical functions be without any error or crash.
- Simulation software's user interface should be user friendly and easy to use
- Simulation software can show the speed of learning algorithms in real time

4.3 Prototype Components

This section introduces the suggested components of the prototype. The selection of an adequate software and hardware platform is a key element, it can be useful for a proper validation mechanism.
There are various research projects using different types of open-source software combined with low-cost hardware in prototypes as shown in Section 2. Following this principle, it is suggested to use the hardware components Lego Mindstorms EV3 with included sensors in its robotic kit combining with ev3dev running Python while using Macbook as a brain of robot with help from Tensorflow numerical computation library and Mosquitto server.

4.3.1 Lego Mindstorms EV3

As mentioned in Section 2.4, Lego Mindstorms kit is very common in research area due to its reliability and flexibility. So the Lego Mindstorms EV3 will be chosen as robotics kit to build an autonomous robot.
According to Lego user guide [95], the control center and power station of Lego Mindstorms is EV3 brick. It included four input ports to connect motors and another four ports to connect sensors. There is one Mini-USB PC port used to connect the EV3 Brick to a computer. A USB Host Port can be used to add a USB Wi-fi dongle for connecting to a wireless network and another SD Card Port to increase the available memory for EV3 Brick. The EV3 color sensor is able to recognize seven different colors and measures light intensity. The EV3 Infrared Sensor can detect objects, track and find the remote Infrared Beacon. In this master thesis, two motor ports will be used for left and right wheels, two ports for color sensors, one port for infrared sensor, SD Card Port for installing ev3dev operating system and USB Port for usb dongle. Figure 4.2 shows two types of EV3 sensors which are used in the prototype.

4.3.2 ev3dev

To interface to the physical hardware of Lego Mindstorms EV3, ev3dev operating system has been installed [96]. With the original EV3 kit, the user has to use the programming blocks in the ev3 software, directly on EV3 brick but it is not

(A) Color Sensor (B) Infrared Sensor

FIGURE 4.2: EV3 Sensors

flexible enough to develop a complex block. Another option is using developer kits of Lego Group. However, it does not get any support from Lego and they recommend not using this option due to its complexity [97]. ev3dev came to resolve these problems. ev3dev is an open source project based on Debian Linux operating system that runs on different hardware devices such as Lego Mindstorms EV3, BrickPi and even Raspberry Pi. Users can boot EV3 from a microSD card containing ev3dev image and then the user can interact with the EV3 via SSH using the command line in a command prompt window (Terminal on MacOSX). The EV3 brick can run the vast majority of popular programming such as JavaScript, Java, Go, C/C++ and the especially Python which will be used in this prototype. ev3dev provided a python interface for devices, which let the user control motors, sensors, hardware buttons, LCD displays of robot. The reason to choose Python in ev3dev is the same as on simulation software as well as message server and that is a good enough complexity-performance trade-off with a full suite of tools for implementing machine learning algorithms.

4.3.3 Macbook laptop

As the brain of robot, the Macbook laptop will be used. The simulation software will run on the Macbook. It includes Python3 developing environment for implementing learning algorithms as well as message server. It will simulate the environment first and then train the agent how to find the path to the target and send command control to the robot. It also provides the user interface which helps user easily control and visualize how learning algorithm works.

4.3.4 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs [98]. TensorFlow is used across Google to apply deep learning to a lot of different areas such as Google Photos, Gmail or Google Search [99]. TensorFlow implemented in Python allows users to express computation model as a graph of data flows. Each node in this graph represents a mathematical operation, whereas edges connection represents exchange data from one node to another. Data in TensorFlow are represented as tensor - a set of primitive values shaped into an array of any number of dimensions. Implementing deep neural network in this way allows users to take the speed of GPU in matrix multiplies and provides much faster computation performance. Tensorflow provides multiples APIs, the lowest is TensorFlow core - which provides user with completed programming control but this is complex to build. There are several higher level APIs built on top of TensorFlow Core, these APIs are typically easier to learn and use than Core API. In this master thesis, Keras will be considered as high-level neural network API for TensorFlow [100]. Keras was built with a focus on enabling fast experimentation and it runs on the top of either Theano and TensorFlow. For deep reinforcement learning algorithm, the author will create a Keras model and then train it with input from environment; therefore, it will provide a prediction for what the agent should do next. The detail of Keras implementation will be discussed in Section 5.

4.3.5 Mosquitto

MQTT is an ISO standard for publish or subscribe lightweight message for using on top of the TCP/IP protocol [101]. The main purpose of MQTT is to minimize network bandwidth and device resource requirements whereas it ensures reliability and assurance of delivery. Due to these reasons, MQTT is regularly used in mobile applications and Internet of Things devices.
Mosquitto is an open source message broker that implements the MQTT protocol. Mosquitto provides a lightweight server suitable for all cases from full power machines to embedded and low power machines. Figure 4.3 illustrates how MQTT broker works **??**. At first, MQTT client sends a Subscribe message included topic name to the MQTT broker in order to receive relevant message. Each subscription message will be confirmed by the broker via a Suback message. Afterward, when

MQTT clients send a publish message with a topic, MQTT Brober will receive this message and forward it to the corresponding MQTT client which has already subscribed this topic. In this thesis, a mosquitto server will be implemented in Macbook to control publish and subscribe messages between Lego Mindstorm EV3 and Macbook.

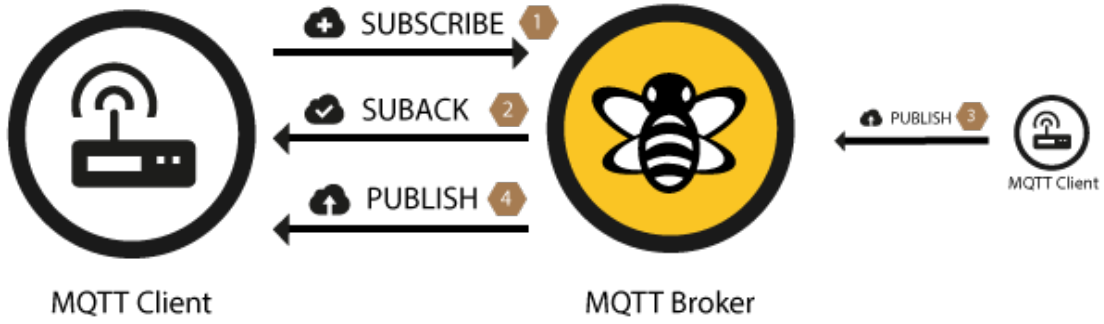


FIGURE 4.3: How MQTT broker works

4.3.6 Environment

In order to test path-finding algorithms, grid map is a highly popular method. Path-finding in grid environments is commonly found in application areas such as mobile robots [102], multi agents [103] and video games [104]. Grid environments are easy to use and offer fast memory access [105]. Because of their simplicity, this thesis will use grid as the main environment to train the reinforcement learning and deep reinforcement learning algorithm. Figure 4.4 shows an example of grid environment which will be used in this master thesis. The map format begins with the number of rows and columns, the intersections between row and column are nodes. There are different types of node such as start, target, obstacle or nothing, represented by green, blue, red color or empty, respectively. In order to find the path from start node to the target node,the robot has to go from one node to another and avoid obstacles.

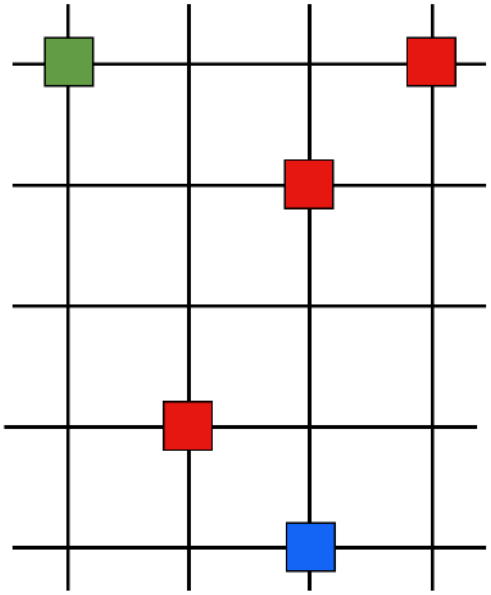


FIGURE 4.4: Grid map environment

4.3.7 Prototype's architecture

So as to clarify the prototype suggestion, the prototype's architecture in this thesis is shown in figure 4.5. It involves all of software and hardware components required for building the prototype. Both Macbook laptop and Lego Mindstorms EV3 robot have to be connected to the same wi-fi connection in order to send and receive messages from the Mosquitto server.

4.4 Conclusions

Taking into account all the results discussed in the previous sections, two main functions of prototype are summarize as following:

- The simulation software which has the ability to show and compare how reinforcement learning and deep reinforcement learning work in grid environment.
- A mobile robot which has the ability to connect with simulation softwares in order to find out the path from one node to another and avoid obstacles on grid map.

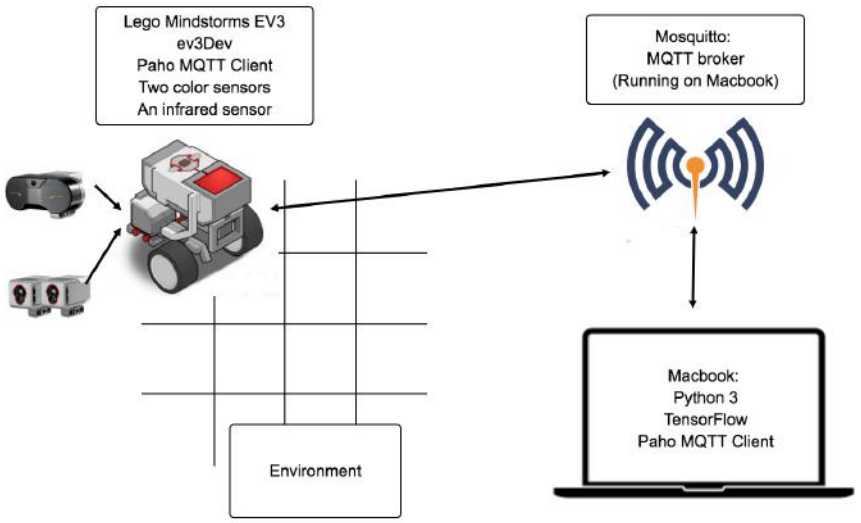


FIGURE 4.5: Prototype's architecture

The details of the problem of how to implement the suggestion will be discussed in section 5.

Chapter 5

Development

5.1 Introduction

This section introduces and explains the development environment of simulation software as well as the robot prototype. the forth sub-research question : *"How to implement reinforcement learning algorithms for Lego Mindstorms EV3 to recognize ways, avoid obstacles, and find the target in partial known grid map environment?"* will be answered in the conclusion section.

5.2 Development Environment

5.2.1 Lego Mindstorms EV3 setup

5.2.1.1 ev3dev

As discussed in the Suggestion section, this thesis will use ev3dev as the operating system for EV3 brick. Due to memory limitation of EV3 brick, a micro SD card will be prepared to flash an ev3Dev image. In this case, this thesis will use SanDisk Ultra 16 GB microSDHC Memory Card (Figure 5.2a) - all micro SD or micro SDHC cards with 2GB or larger capacity will also work. Following is step by step guide on how to install ev3dev on Lego Mindstorms EV3:

Step 1: Download the latest ev3dev image file from : `https://github.com/ev3dev/ev3dev/releases/`

Step 2: Download and install Etcher from `https://etcher.io/`. Then use Etcher to flash ev3dev image downloaded from previous step to micro SD card as shown in Figure 5.1.

Step 3: Put the SD Card in EV3 brick's SD slot and power on.

Step 4: Wait for a few minutes until ev3dev is successfully booted.

FIGURE 5.1: Etcher homescreen

The original EV3 brick does not support wifi connection so a wifi USB dongle has to be prepared. The prototype used Edimax EW-7811UN as USB dongle (Figure 5.2b) in order to support the EV3 brick to connect to the wireless network. Now EV3 have wi-fi network connection. It is able to connect EV3 from SSH. SSH connection allows terminal commands to run securely on EV3 from laptop. As a result, it is possible to run programs, change settings and install new programs directly from laptop without any physical connection. Following steps show how to connect to EV3 from Macbook laptop:

Step 1: Boot ev3dev operating system and connect to the same Wi-Fi network as Macbook laptop.

Step 2: Open Terminal

(A) SanDisk Memory Card (B) Edimax EW-7811UN

FIGURE 5.2: Additional hardware components

Step 3: Type: `ssh robot@ev3dev.local`

Step 4: Enter EV3's password when prompted. The default password is `maker`

5.2.1.2 Motors and Sensors

In Section 4.3.1 it was demonstrated that the EV3 brick has four motor ports and another four sensor ports. To implement the simulation, the EV3 robot need two large motors, two color sensors and an infrared sensor.

The Large Motor has a built-in rotation sensor with 1-degree resolution for precise control. In order to drive the robot, Lego given Move Steering or Move Tank programming block in the EV3 software to coordinate the action simultaneously in two large motors. However, ev3dev does not support Move block so it have to control the large motor separately. Default port for two large motors is Port B and Port C so the left large motor will assign to Port B and the right large motor will assign to Port C.

The color sensor can detect the color or intensity of light that enters a small window on the face of the sensors. There are two color sensors used to prototype with Reflected Light Intensity Mode which helps it measure the intensity of light reflected back from a red light-emitting lamp. With this mode, color sensor will return value in range from 0 (very dark) to 100 (very light), this can help the robot follow the black line with a white surface (Section 5.4). Both of color sensors will be mounted in the front of prototype. The left color sensor will assign to Port 1 and the right color sensor will assign to Port 2.

The infrared sensor is designed for proximity detection for the EV3 robot. It is

often used in detecting obstacles or for remote control of the robots from EV3 infrared beacon. An infrared sensor will be mounted in the front of the robot and on the top of the color sensors. The infrared sensor will assign to Port 4. Figure 5.3 shows the EV3 brick with required motors, sensors and its corresponding ports for prototype.

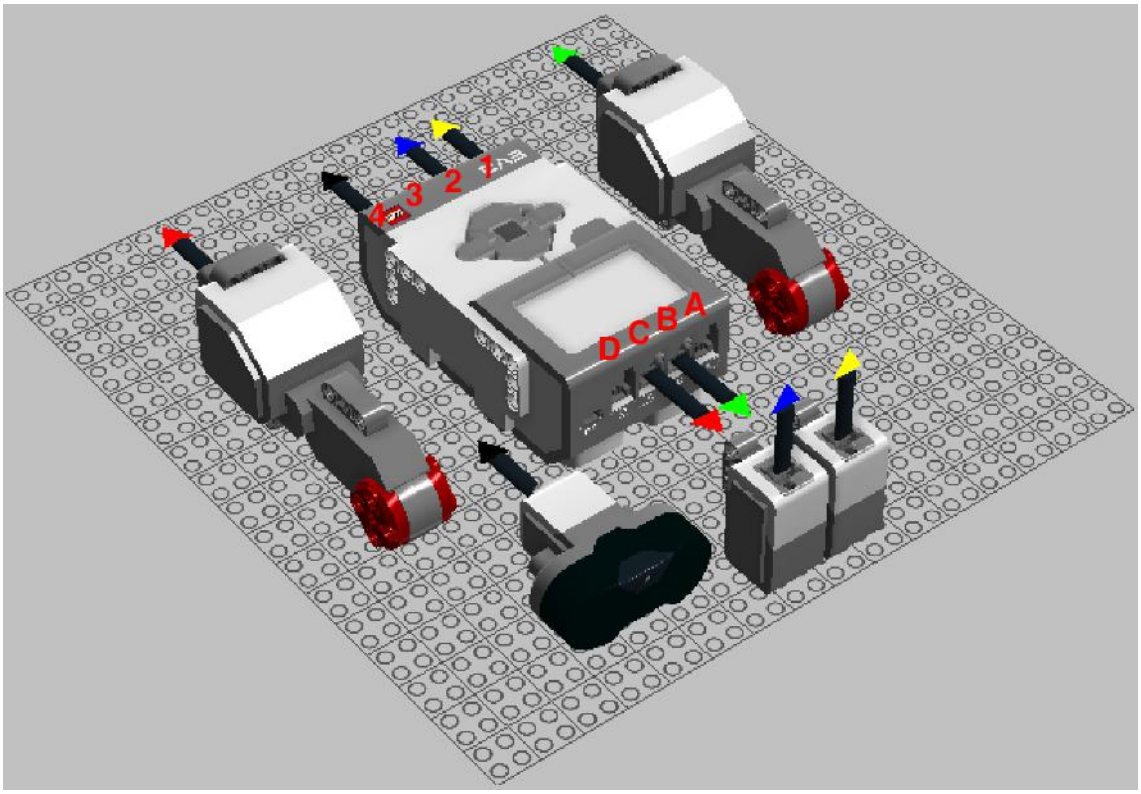


FIGURE 5.3: EV3's design

5.2.2 Macbook setup

5.2.2.1 Conda and Tensorflow

Conda is a package manager application that quickly installs, runs, and updates packages and their dependencies [106]. With Conda, developers can easily install different versions of software packages and switch easily between them. With just a few commands, Conda can set up a totally separate environment to run a different version of Python or other packages without any effect on the normal environment. The following steps will show how to install Anaconda, TensorFlow and Keras for Macbook:

Step 1: Download the graphical installer from : `https://repo.continuum.io/archive/Anaconda3-4.4.0-MacOSX-x86_64.pkg`

Step 2: Double-click the downloaded .pkg file and follow the instructions

Step 3: Open Terminal

Step 4: Create a conda environment named tensorflow by type command : `conda create -n py35 python=3.5 tensorflow`

Step 5: Activate the TensorFlow conda environment by : `source activate tensorflow`

Step 6: Install TensorFlow inside conda environment by: `pip3 install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.2.1-py3-none-any.whl`

Step 7: Install Keras - TensorFlow Highlevel API by: `pip3 install keras`

5.2.2.2 MQTT

As discussed in Section 4 there are two main components required by MQTT, they are MQTT broker and MQTT client. Mosquitto has been chosen as MQTT broker and Paho MQTT as MQTT client for Macbook laptop. To install and run Mosquitto server , do the following steps:

Step 1: Install Homebrew package management system by follow instruction in `https://brew.sh/`

Step 2: Install Mosquitto by : `brew install mosquitto`

Step 3: Install Paho MQTT client by : `pip install paho-mqtt`

Step 4: Start Mosquitto server on localhost : `brew services start mosquitto`

There are some commands also necessary in order to control Mosquitto server such as :

- Stop Mosquitto server : `brew services stop mosquitto`
- Restart Mosquitto server : `brew services restart mosquitto`

- Subscribe to a topic : `mosquitto_sub -h 127.0.0.1 -t TOPIC_NAME`
- Subscribe to all topics : `mosquitto_sub -h 127.0.0.1 -t '#'`
- Publish to a topic : `mosquitto_pub -h 127.0.0.1 -t TOPIC_NAME -m MESSAGE`

5.2.2.3 Python IDE

Pycharm Community is free and lightweight IDE for Python and Scientific development [107]. PyCharm offers an intelligence coding assistance which provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactorings and rich navigation capabilities. There are several built-in developer tools such as built-in terminal, integration with major VCS and database tools, an integrated ssh terminal, and integration with Anaconda package manager. One more importance, Pycharm has been supported by ev3Dev as a python development environment for EV3 brick. Tutorial to set up Pycharm for ev3dev can be found at `http://www.ev3dev.org/docs/tutorials/setting-up-python-pycharm/`. Figure 5.4 shows the structure of Pycharm project with its two sub projects.
`Ev3Project` is a python project which runs on EV3 robot. There are two important files: `run.py`, which allow MQTT clients to subscribe commands from laptop and then process these commands to control the robot whereas `Variable.py` is sharing constant variables of the project.
The second sub project - `Path_finding_simulation` is a python project running on Macbook. `qlearning_path_finding.py` and `deep_qlearning_path_finding.py` are used to simulate the Q learning and Deep Q learning environment, where they share some constant variables from `variable.py`. After successfully training the deep learning model, it is able to save the keras model on `save` folder to reuse that model later.

5.2.3 User interface

According to the mockup suggestion discussed in the previous section, the user interface has been implemented as shown in Figure 5.5. The guideline following will show how it works:

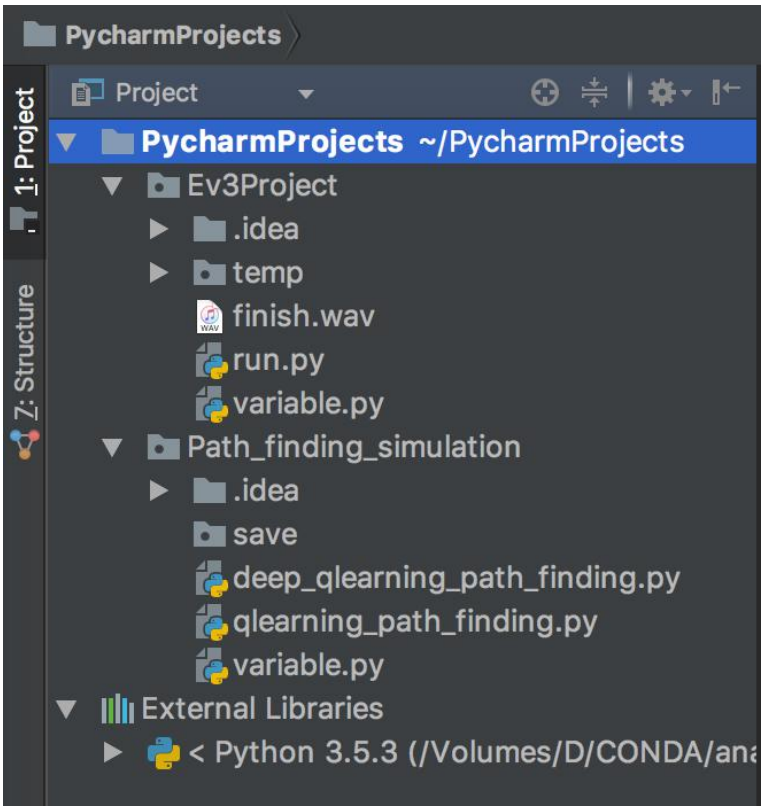


FIGURE 5.4: Pycharm project

Step 1: User inserts the number of row and for the grid map which will generate later for training agent to find the path.

Step 2: Click to `Generate size` button to create the grid map

Step 3: User clicks to the grid to choose start point(green), target point(blue) or obstacles(red)

Step 4: Click button `Train with deep reinforcement learning` to train the agent in finding paths to the target.

Step 5: (Optional) If the user wants to control EV3 robot, make sure the location of EV3 robot corresponds with grid map and the default direction of robot is from top to bottom of grid map. Because EV3 does not understand the current direction of the robot so the `Reset EV3 direction` button aims to reset directions of EV3 to default direction and the user has to put the robot on the start point with direction from top to bottom.

Step 6: Button `Start run EV3` is responsible for sending control commands from simulation application to EV3 in order to move robots to the target point.

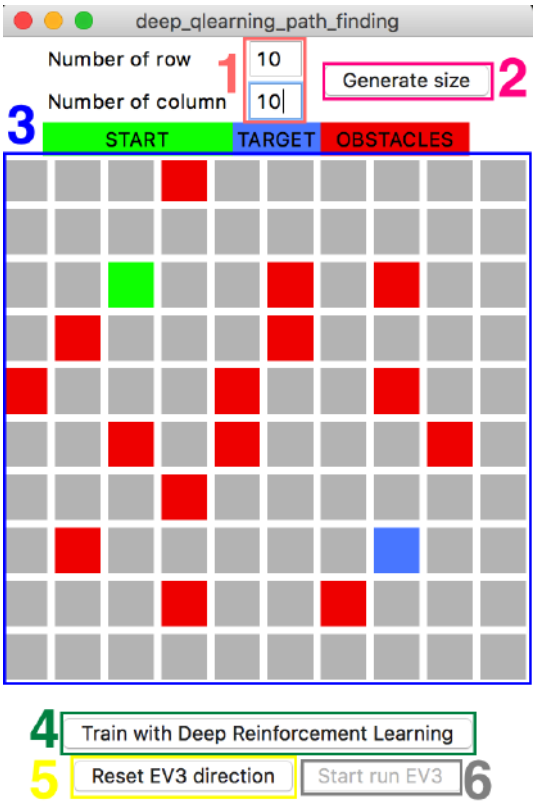


FIGURE 5.5: The user interface screenshot

5.2.4 Hardware and Software overview

The hardware and software used in this thesis are enumerated in Table 5.1.

5.3 Simulation software

This section addresses the environment description which will be used by both Q-learning and Deep Q-Network and then this section will discuss what will be different when implementing these two algorithms.

Name	Function	Specification
Hardware		
Lego Mindstorms EV3	Robot	TI Sitara AM1808,64 MB RAM, 16 MB Flash
Macbook Pro Retina	Robot's brain, server, simulation environment	Intel Core i5, 8GB RAM, Intel Iris 6100 VGA
SanDisk Ultra microSD	Boot ev3dev image	16 GB
Edimax EW-7811UN	USB Dongle	150Mbps Wireless
EV3 Color Sensor	Path following	n/a
EV3 Infrared Sensor	Detect obstacles	n/a
Software		
TensorFlow	Deep learning API	v1.2.1
Keras	TensorFlow high level API	v2.0.6
Python3	Macbook developing environment	v3.5
Python3	EV3 developing environment	v3.5
ev3dev	Lego's robot operating system	v0.8.1
MacOS Sierra	Macbook's operating system	v10.12.5
Pycharm	Python IDE	v2017.1.4
Anaconda	Python distribution	v4.3.22
Paho MQTT	MQTT python client	v1.2.2
Mosquitto	MQTT broker	v3.1.1

Table 5.1: Hardware and Software specifications

5.3.1 Environment Description

The simulation software consists of the environment and the agents. As defined by Sutton et al. [50], the agent is a learner or decision-maker whereas the environment is what the agent interacts with and comprises everything outside the agent. In the simulation software, the agent can be Q-learning agent or Deep Q-Network agent. In order to compare reinforcement learning algorithms, the author will setup both of them in the same environment.

In the grid map, there is a matrix S of $n * m$ whereas n is number of rows and m is number of columns. In order to convert graphical map to a matrix, each state

of the matrix will be denoted as a float number. The followings show the different states of the matrix:

- Obstacle: Set an obstacle here so the agent has to avoid this cell. (0.0)
- Visited: Where the agent has already been in one episode. (0.25)
- Occupied: The current cell which is being used by the learning agent. (0.5)
- Target: Agent have to find the path from starting cell to target cell. (0.75)
- Empty: It is valid for the agent to move to this cell. (1.0)

In figure 5.5, it is easy to see that Occupied has been shown in green color (the starting point will be set as Occupied point for the first action), Target is blue in color, Obstacle is red and Empty is grey. The Visited cell is used only to calculate the reward function so it will not be shown on the user interface. The agent is allowed to move only on the empty cells and it has to avoid moving out the grid map, step in to collision and the purpose is to get to the target. There are 4 actions that are possible for the agent corresponding with integers 0-3:

- 0 - Go left: Move to the cell on the left hand
- 1 - Go up: Move to the cell on the top
- 2 - Go right: Move to the cell on the right hand
- 3 - Go down: Move to the cell on the bottom

Figure 5.6 illustrates the possible moves for the agent. This master thesis limit the action of environment to only four above actions and do not allow the agent to move in a diagonal direction which will also be used for the robot. This thesis encoded the direction to four integers because it will be predicted later by learning agent which will be discussed later on.
In reinforcement learning, the purpose of the agent is to formalize in terms of a special reward signal passing from the environment to the agent [50]. For each time step, the reward is a number and the agent has to maximize the total amount of rewards it receives. It does not only maximize immediate rewards but also cumulative rewards. When teaching a child, parents try to ignore child's bad behavior and reward their good behavior. The author used the same idea for

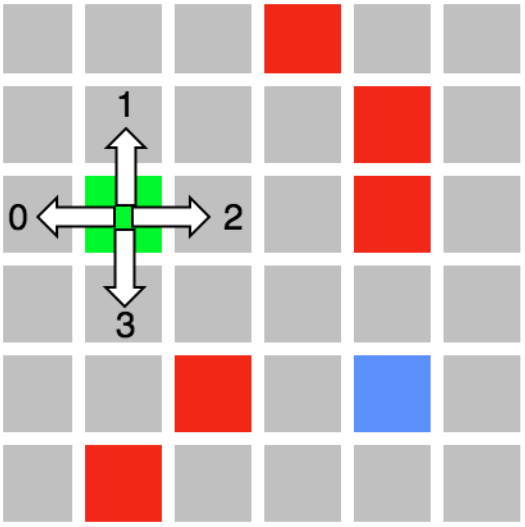


FIGURE 5.6: Actions and rewards example

learning agent in this case. Environment will give a positive reward when the agent does a good action and give it punishment when it implements a bad action. In order to build a reward function for reinforcement learning, following rules will be defined:

- The range of reward will be from -1.0 to 1.0
- The reward can be a positive for reward or negative for punishment
- For every move from one cell to another cell, it will cost -0.05 points. This will keep the agent from wandering around and find the shortest path to the target.
- In order to avoid the obstacles, a punishment with -0.75 points will be given to the agent if they enter a Obstacle cell. This is a strict penalty, so hopefully the agent will learn itself to avoid this punishment completely.
- Above mentioned rules will be applied in case the agent moves outside the grid map boundaries with -0.75 points.
- The agent will get penalized by -0.25 points for any action to the cell already visited. The term Visited cell mentioned before will be used here to find out which cell has been already visited or not. It means that the visited cell should not be considered.

- To avoid infinite loops , a minimum total reward will be set up as (-0.5 * environment size). When the total reward is below the minimum value, the episodes will stop and skip to the next episode.

Environment also define the end state for each episode by `Done` variable. `Done` variable will return `True` in case most of recent actions lead to end state or return False in remaining cases. There are 3 cases leading to the End state: agent reaches the target, total reward gets below threshold or agent cannot perform any valid action (obstacles blocking the way). Figure 5.7 shows an example of action and reward function. It is easy to see that the reward for every legal move is -0.05 points such as from State 1 to State 2, from State 2 to State 3 or from State 4 to State 5. When the agent moves to a Visited cell, it will get a punishment worth -0.25 point. For the final move, the agent reaches the target so the reward is 1.

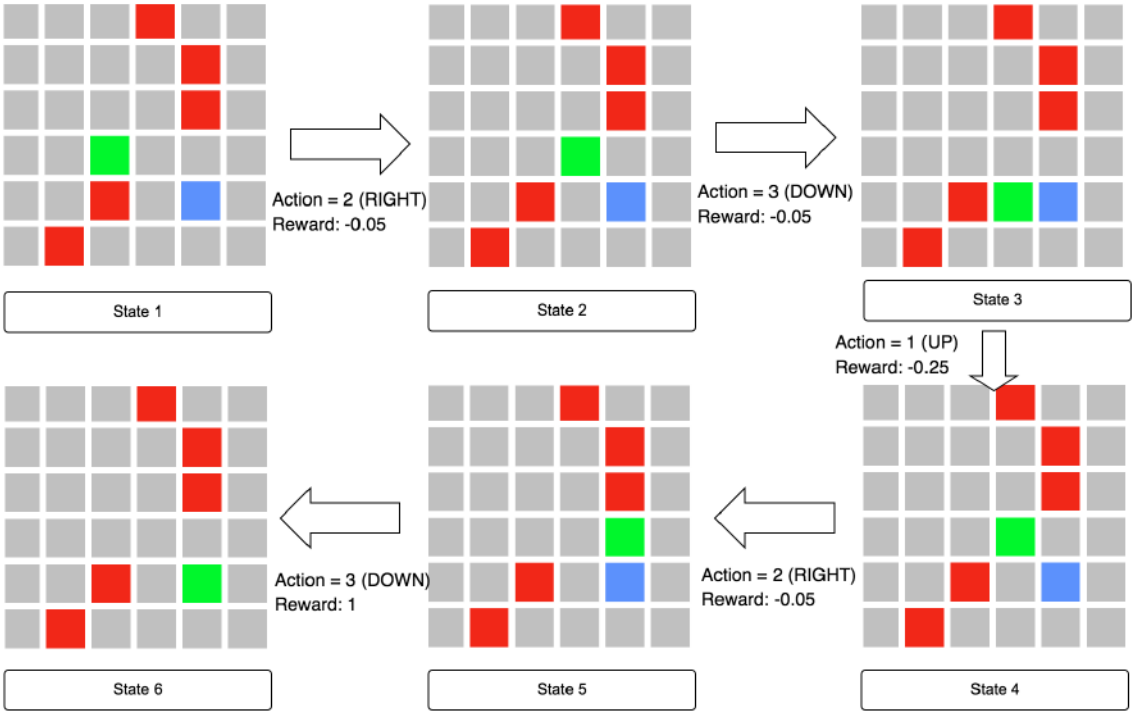


FIGURE 5.7: An example of actions and reward function

5.3.2 Q-learning

For implementing the Q-learning algorithm, developers need to follow the Q-learning pseudo-code 3 mentioned in the section 2.3.2. The most important component of Q-learning is Q table. In the traditional Q-learning algorithm, It is

required to create a table Q with the size $m * n$ where m is state size and n is action size. The action size is fixed with 4 actions: left, up, right or down. However the state size will increase exponentially when the size of matrix is change. For example, there is a matrix with size $a * a$ and each cell have different five states: Occupied, Target, Obstacle, Visited or Empty so the state size is a^{2^5} in total. Therefore, it is not effective if creating a Q table with all possible states. To solve this problem, this thesis will use DataFrame from Pandas - a python data [108]. DataFrame is two-dimension size-mutable, heterogeneous tabular data structure with labeled axes. More precisely, Pandas data frame consists of three main components: the data, the index and the columns. It is able to create a table including four columns corresponding to four actions with empty data. When users add data with current state information to the table, if the data exists the corresponding data will update; otherwise, it will add a new record so the Q table will contain only the occupied state.

$$\boldsymbol{Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s^{'}, a^{'}) - Q(s,a)]} \tag{5.1}$$

Look at the most simple form of Q-learning 5.1, the Q table will store a value for each state-action pair in a cell. In every state, the algorithm will update the Q value of current state based on Q value for next state and its reward. As mentioned before, the same model for Q-learning and Deep Q-Network will be used. After some experiments, following hyperparameters will be used for both reinforcement learning algorithms:

Name	Value	Description
α	0.01	Learning rate
γ	0.9	Discount factor
ϵ	0.1	ϵ-greedy exploration

TABLE 5.2: Table to test captions and labels

5.3.3 Deep Q-Network

In the section 2.3.3 it was demonstrated that Deep Q-Network is a combination between Q-learning and Deep Neural Network. In Q-learning, a Q-learning table will be use which represents the memory of what the agent has learned through

experience. However, Deep Q-Network uses a neural network to save its memory. Look at the Figure 5.8, where an agent interacts with an environment [109]. After every step, the agent has its state and it is asked to choose an action. After taking the best action, the environment will transit to the next state and the agent will get a reward. Then the neural network uses its state and reward in order to make it learn about the environment. It is important to note that the agent has no prior knowledge of which state the environment would transit to or what the reward may be. By performing the action and during training, the agent can observe these quantities and improve itself. The output of neural network is a probability distribution over actions with range from 0 to 1. The higher probability of action may provide better cumulative reward for the learning agent. Figure 5.9 shows

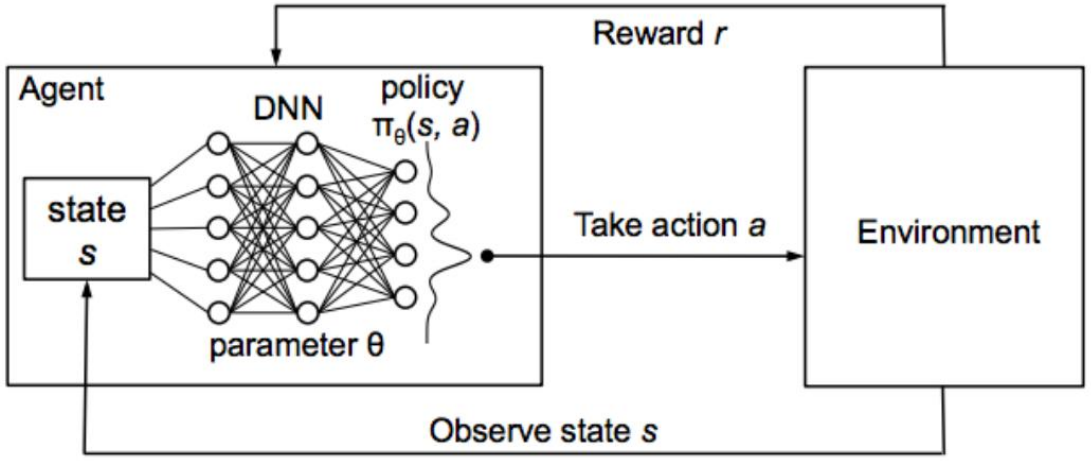


FIGURE 5.8: Deep Q-Network

our Deep Q-Network agent. On the left-hand side, the grid map has been inserted by user including start point, target point, and collisions. It is too complicated for the learning agent to understand a visual map so the author have to convert them to the matrix with the rule defined in Section 5.3.1. Like the traditional neural network, the neural network of Deep Q-Network also requires an input layer which contains a number of nodes. In fact, the matrix is two dimensional array so it have to reshape to the form of a one dimensional array. Then the hidden layer extracts features of the input space and the output layer translates them into the probability distribution. With support from TensorFlow library, it is able to build a numerical computation graph to perform these tasks. Listing 1 shows how to build TensorFlow neural network with its high level API - Keras. The `Sequential` model is a linear stack of layers. Then a list of layer instances can pass

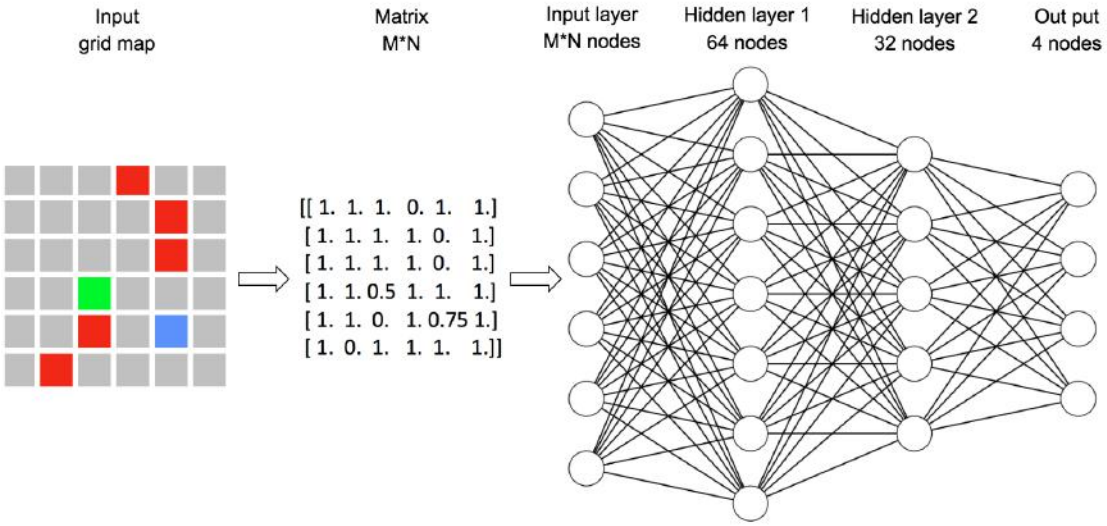


FIGURE 5.9: Deep Q-Network architecture

to the constructor by `Dense`. The activation function can either be used through an `Activation` layer, or through the `activation` argument as implemented code. The loss function is required to compile a model. Keras supports several common loss functions such as `mse` (mean squared error), `mae` (mean absolute error), `mape` (mean absolute percentage error), `msle` (mean squared logarithmic error), `kld` (kullback leibler divergence) or so on. In order to minimize the loss function, an optimizer is also required by Keras model. After few experiments, the author choose `tanh` as activation function, `mse` for loss function and `adam` for optimization algorithm.

```
def _build_model(self):
# Neural Net for Deep Q-Network Model
        model = Sequential()
        model.add(Dense(64, input_shape=(self.state_size,)
                , activation='tanh'))
        model.add(Dense(32, activation='tanh'))
        model.add(Dense(self.action_size))
        model.compile(loss='mse', optimizer='adam')
        return model
```

LISTING 1: Build neural network with Keras

5.4 Robot path following

This section will focus on how to make EV3 robot detect its surroundings and make decisions to follow the path with fuzzy logic. In the traditional logic, the truth values of variables may only be the integer values 0 or 1. However, It can not handle the concept of partial truth where the truth value may range between true or false. The term fuzzy logic was introduced by Lotfi Zadeh as fuzzy set theory [110]. Fuzzy logic is an approach to compute based on degrees of truth rather than the usual true or false. The fuzzy logic also includes 0 and 1 as extreme cases of truth but also includes the values in between and it is closer to the way our brains work. Fuzzy logic has been applied to many fields from control theory to artificial intelligence. One of recent success of fuzzy logic is fighter pilot. The University of Cincinnati designed an AI based on fuzzy logic, this system beat retied United States Air Force in multiple Simulated Air Combat Missons with Unmanned Combat Aerial Vehicle [111].
One of the possible solutions to make EV3 follow the line is using a color sensor. There are different three modes for EV3 color sensor: detecting the color of and object, measuring reflected light, or measuring ambient light. In this prototype, EV3 color sensor will be used to check the reflected light level - the intensity from the surface that the LED light reflects. If EV3 brick placed on a table and mounted the light sensor close to the floor, it would be able to detect the intensity of the light. The location of the color sensor will decide how well the robot will respond to the value when following the path so Figure 5.10 shows how to setup EV3 color sensor.

The intensity levels will range from 0 to 100 when it is properly calibrated.

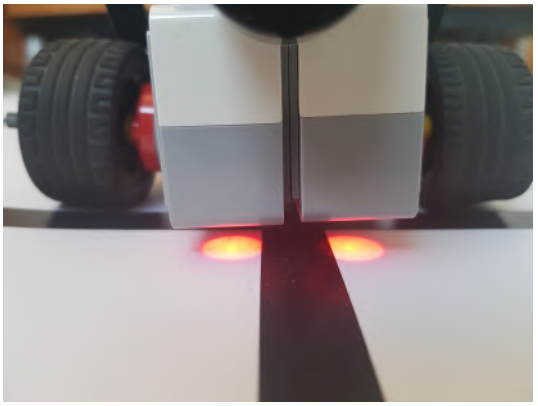


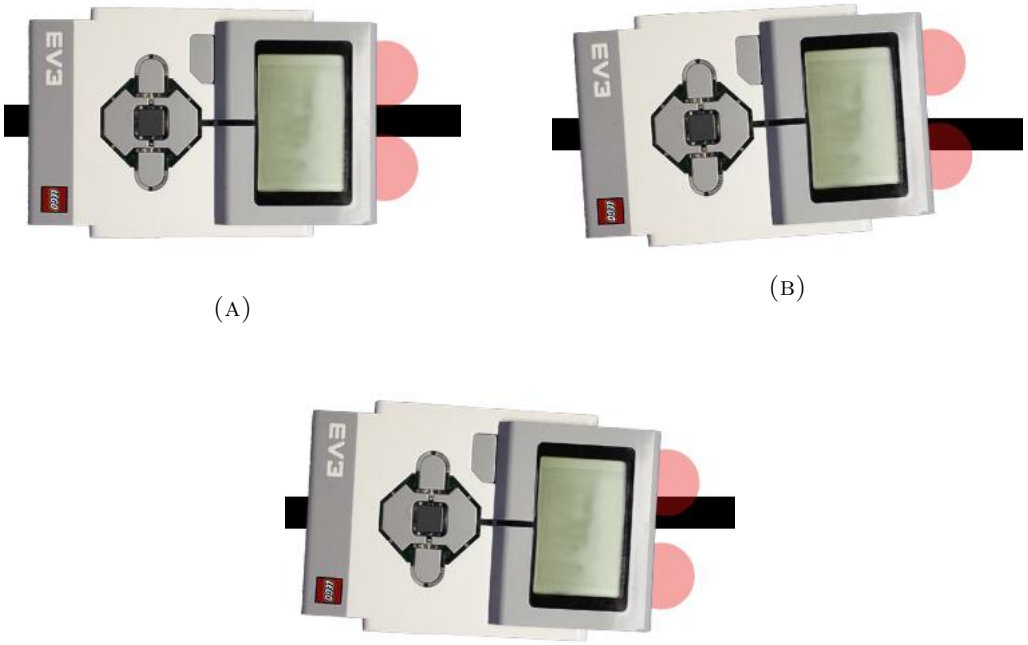
FIGURE 5.10: Setup EV3 color sensor

However, the intensity level often shows a different range of color because the room has light conditions that may change. The calibration means teaching the sensor what is black and what is white color corresponding with value 0 and 100. In case two color sensors exist, the same calibration will apply to both sensors. STEMRobotics has created a course for teaching student how to calibrate EV3 light sensor [112]. However, because of some sensors always return different values from each other, users have to write a custom calibration by code or EV3 software [113]. The most simple line following programs only uses one color sensor. Sensor will either detect black or white color and adjust accordingly. More precisely, the robot does not only try to follow the line but also try to find the edge of it. Robot must choose to follow the left or the right edge of the line. Then the robot will check the light value that is dark or bright. Based on this value, the robot can adjust the motor to go to the right or the left. But what will happen if the grid map contains perpendicular lines? The single color sensor cannot be used in this case, that is the reason why two color sensors mode steps in. Please note that in the original Lego Mindstorms EV3 Retail or Educational kit, there is only one EV3 color sensor, it is possible to buy a second sensor separately. If two color sensors are available, the space between them should be set up just a bit wider than the line of the grid map.

The basic ideal of using two color sensors is shown in Figure 5.11. There are 3 cases for both sensors:

- Figure 5.11a: If both sensors do not detect the line: continue forward.
- Figure 5.11b: If the right sensor detects the line and the left sensor does not: turn right slightly until the robot is back on the right edge.
- Figure 5.11c: If the left sensor detects the line and the right sensor does not: turn left slightly until the robot is back on the normal state.

In order to make the robot follow the line, a fuzzy logic system which can control the power of motors will be provided. In general, to steer the model robot to the left, the power of right motor power should be higher than the left motor or vice versa. Assume D is the difference between the value of left color sensor V_{left} and right color sensor V_{right} so $D = V_{left} - V_{right}$. Depending on the value of D, the power of left and right motors will be adjusted. The fuzzy rules in Table 5.3 describe the interaction between sensors input and robot direction. Trapezoidal



(A) (B)

(C)

FIGURE 5.11: EV3 with two color sensors mode

membership functions are used both for the left and right motors output as illustrated in Figure 5.12. When D increases, the Left Motor Power tends to get a higher value, the Right Motor Power tends to get a lower value and vice versa. There is one balance point where D is zero and the power of left and right motors are the same. This will lead the robot to go straight as the description in Fuzzy Rule 3.

Name	Rule
Rule 1	IF D IS very low THEN turn sharply to the left
Rule 2	IF D IS low THEN turn slightly to the left
Rule 3	IF D IS 0 THEN stay straight
Rule 4	IF D IS high THEN turn slightly to the right
Rule 5	IF D IS very high THEN turn sharply to the right

TABLE 5.3: Fuzzy rules for sensors value and robot direction

FIGURE 5.12: Fuzzy Logic Membership Function for the Left and Right Motor

5.5 Path-finding and Obstacle Avoidance

Reinforcement learning is a real-time, online learning method. In order to interact with a grid map environment, Lego Mindstorms EV3 robot has the ability to detect the obstacles along the way and avoid them. An example of grid environment has been shown in Figure 5.13. In this example, the grid map environment contains five rows and four columns with edge 20 cm for each cell. EV3 robot has been placed at the top left of the grid map at $(0, 0)$ and it has to find the path to the target at $(4, 2)$ and make sure the robot will not hit two obstacles at $(1, 2)$ and $(3, 1)$. Along the way to the target, it may have some dynamic obstacles so the agent must possibly avoid the unknown obstacles too.

In order to detect the unknown obstacles, an EV3 infrared sensor has been mounted on the front of the robot. With the Proximity Mode, Infrared Sensor uses the light waves reflected back from the obstacle to estimate the distance between the sensor and that object. The range of values is from 0 - close to 100 - far. However, it does not provide a specific number of centimeters but it may detect an object up to 70 cm far way. Figure 5.14 shows that the distance D is the proximity value between the EV3 robot and the obstacle. In this prototype,

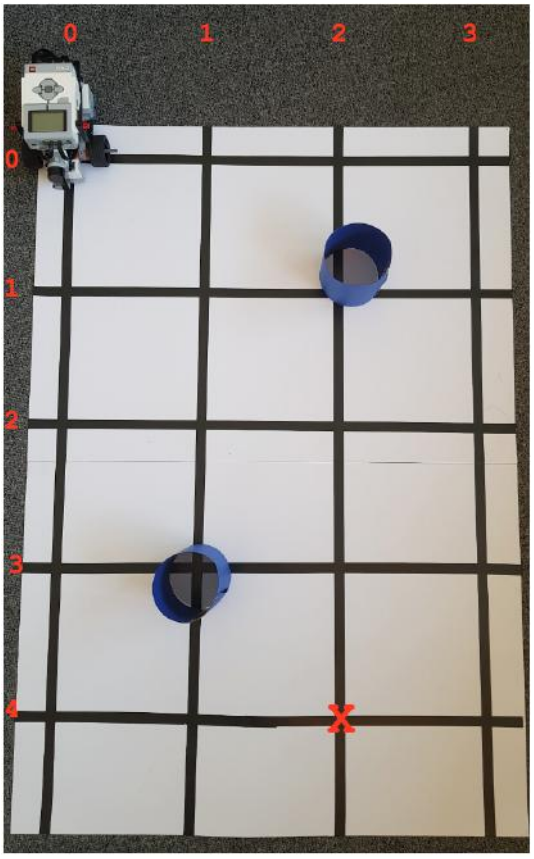


FIGURE 5.13: EV3 on grid environment

the robot meets the obstacle when EV3 Infrared sensor returns value D less than 20.

The path finding agent goes as pseudo code shown in 6. The key ideal of this

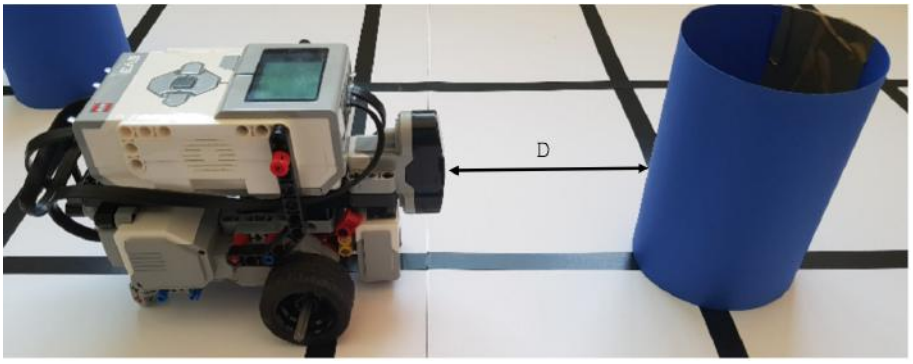


FIGURE 5.14: Setup EV3 infrared sensor

setup is that the robot tries to find the optimal path from start point to target and if it meets an unknown obstacle along the optimal path then it will use the

sub-optimal path which does not include the obstacle, otherwise the robot will retrain the model with an updated environment.

Algorithm 6 Path Finding

Required: A partial known grid environment with *startPoint*,*targetPoint* and *listObstacles*
Optional: Some unknown obstacles
while reach to the target **do**
    findingPath(*startPoint*,*targetPoint*,*listObstacles*)
    Save the successful path to memory M
    Follow the optimal path in M
    if Meet the *obstacle* **then**
        Append *obstacle* to *listObstacles*
        Follow the sub-optimal paths which does not include *listObstacles*
    if Reach to the target **then**
        break
    else
        Update *startPoint*
        continue

5.6 Conclusions

This section addresses the forth sub-research question : *"How to implement reinforcement learning algorithms for Lego Mindstorms EV3 to recognize ways, avoid obstacles, and find the target in partial known grid map environment?"*. Firstly, a simulation software has been built in order to visualize how the reinforcement learning works. Then a Lego Mindstorm EV3 will be extended by a frame to mount two color sensors and an infrared sensor in front of the robot in order to follow the path and avoid obstacles, respectively. EV3 robot has been tested in the grid map environment which contains known and unknown obstacles. In the next section, the simulation and prototype results will be discussed.