University of Sydney

Major Project

Baton: TurtleBot Dynamic System Control via Gesture Recognition

(Group 17)

MTRX5700 - Experimental Robotics

14-05-21

470148329
470416309
480436153
470018365
500575765

Contents

List of Figures

List of Tables

List of Acronyms

ROS	Robotic Operating System
RPI	Raspberry Pi

1 Abstract

This report encompasses the development of a mechatronic system demonstrating the dynamic control of a TurtleBot utilizing hand gesture recognition. This specific system has been named 'Baton'. The system can recognize hand signals and gestures through a RGB camera or webcam and read such information being transformed into comprehensible data for a TurtleBot. This data is processed and sent in real-time to the TurtleBot, allowing for control over its movement. Such control includes the manipulation of the TurtleBot's movement in terms of velocity. The system employs the recognition of approximately seven gestures controlling forward, backward, acceleration, deceleration, left, and right movement of the TurtleBot.

The system has been named 'Baton' originating from the idea of an orchestral conductor's tool known as a 'Baton', utilized in controlling and guiding of an orchestra. Hence, comparable and summarising the key aspects of this system, employing hand signals and gestures to control the velocity and movement of a TurtleBot.

Keywords—TurtleBot, Mechatronic, Baton, RGB

2 Introduction

The simplest and most common method to control a robot is via a handheld controller either, connected physically to the robot using wires or a cable, else communicated over Bluetooth or infrared transmitted data. Commonly, toggle switches are included within controller design to allow the user to manipulate the robot's control. In the simplest case, such a controller would use a direct connection to command the motors and battery. Such robots usually have no intelligence.

Within industry it is a common and desirable goal to minimize labor encircling the user and production cost, in turn reducing market prices, engaging with a broader clientele and increasing sales. Eliminating physical connections and controllers within robot control and design will reduce the physical labor required by users and reduce the material and components required. Furthermore, allowing for an extension to the complexity of the robot, since commands are not limited to the toggle switches or physical dimensions of the controller. the The elimination of physical controllers is an example of being able to achieve such goals in the field of experimental robotics, without compromising upon the features of the controller. The major challenge that comes with removal of a controller in robot control is to ensure the replacement can perform with similar or greater accuracy and precision.

The replacement of a physical controller discussed within this project will be through the utilization of hand gesture detection. This allows for a physical controller requirement to be completely eradicated and for there to be an extendable amount of commands and features that can be directed upon the robot via the user. Furthermore, these features can be implemented onto the robot's system at any time, even after completion of the robot. Therefore, meaning that after and during the testing, monitoring and control stage of the project, respectively, software updates can be an efficient manner to build upon the robot's system, ensuring its accuracy and performance is kept to a high standard. Alike, any bugs or future issues can be addressed in this manner without the hassle or requirement of hardware connectivity that a physical controller would introduce.

For these reasons it is significant to continue to investigate methods to simplify and minimize physical robot dynamic control. Currently, the robotics industry has displayed hand gesture control in various areas and concentrations of engineering. Some examples include:

- Monitor/Screen Control - Vehicle Infotainment Systems
- Remote Surveillance Robot - Military Applications
- Patrolling and Surveying - SWAT and Boarder Patrol Applications
- Entertainment/Design - Architectural Software, Gaming and Virtual Reality (VR)

Correspondingly, the future of the robotic industry would benefit from employing such a design within the applications of experimental projects. Some examples include:

- Wheelchairs - Physically Challenged/Medical Field
- Robotic Arms - Industrial Grade Applications

This report will cover the application of hand gesture detection to manipulate the velocity and movement of a TurtleBot. The various sections include; outline of the **system requirements** and **specific goals** of the system, the breakdown of each **component** of the system design (inclusive of **hardware** and **software** components), discussion of all **algorithms** and **methods** employed, **experimental setup** or **design integration** of mentioned components, description of **testing**, **testing environment** and obtained **results**, a **discussion** of obtained results and **conclusion** on achievement of set system requirements and a mention of **future developments**.

3 Background

3.1 Focus

3.1.1 Primary Focus - Hand Gesture Robot Control

For concentration of the scope of robots that exist within the realm of gesture control, this project will encompass addressing eliminating a physical controller in application of a surveillance robot. Furthermore, undertaking the mission of having speed and directional control (velocity and acceleration) via hand gestures that are directed towards a physical webcam attached to a computer. This computer will be running the primary program, allowing for the processing of the data captured by the webcam and communicating to the TurtleBot. Hence, the TurtleBot will carry out the movements corresponding to the defined hand gestures that are detected via the webcam, displayed by the user.

3.1.2 Secondary Focus - Mapping and Surveillance

An extension upon the project's outcomes include that of implementing a mapping functionality, specifically illustrated visually for the user of the system, denoting the surrounding area of the robot's physical environment.
Alike, implementation of a live camera feed from a mounted camera upon the TurtleBot extends upon the objective of surveillance, additionally improves upon the comprehensibility of the robot's movement (eliminating the necessity to physically maintain visual contact with the robot).

3.2 Design Outcomes

3.2.1 Primary Focus - Hand Gesture Robot Control

The primary design outcomes of this system include:

- Detection and Recognition of Hand Gestures
- Classing of Hand Gestures into Specific Commands
- Saving of Hand Gestures Recognised by the System
- Display of Recognised Hand Gesture in the form of a Label
- Linking of Gestures to Publishing Functions
- Sending or Publishing of Data to the TurtleBot
- Movement of TurtleBot through a Physical Environment
  - Corresponding with Gesture Commands

3.2.2 Secondary Focus - Mapping and Surveillance

The extended design outcomes of this system include:

- Illustration of surrounding area using GMapping - Simultaneous Localization and Mapping (SLAM)
- Display of Camera Feed from TurtleBot

3.3 Published Literature

There are various resources, papers and journals that had assisted in achievement of this project's success and coincides with the focus or scope of this project, existent in literature. Some of these publications that were consulted include those revolving around **teleoperated** robotics - those that enable human control from a distance, aligning with surveillance applications.[1] Within the realm of gesture control defines various ideas aligning with our aspects of our project, specifically in the area of recognizing signals and formations created by a user's hand but, still requires a transmission device.[2] Furthermore, below are highlighted some exemplar projects that coincide with this system's area of concentration.

3.3.1 Robotic Car Control System - Exemplar 1

This project uses gesture recognition to control the movement of the robotic car. The system captures gestures through a webcam and recognizes gestures by image processing. The whole system can be divided into four stages, acquiring images, processing images, gesture recognition and transmitting instructions, respectively. At the beginning the user places the hand directly in front of the webcam to show gestures, and the camera will continuously capture gesture images. A series of processing needs to be performed on the image after the image is captured. First, un-distort the image and adjust the exposure of the image. Then increase the saturation of the image to enhance the RGB value. The next step is to convert the RGB image into HSV since the skin area of the HSV image is easier to be recognized. Afterwards the HSV image will be converted into binary image. Therefore, the area of the hand in the image can be determined. After determining the gesture through machine learning, the command will be sent to robotic car with a ZigBee transmitter. The robotic car will start to move according to the received instruction. [3]

3.3.2 Contact-less Device Control - Exemplar 2

This project uses gesture recognition technology to control the robotic arm to deliver surgical equipment and control the lighting system in operation room. Compared with voice control, gesture control has higher accuracy since the voice control will be affected by other sounds in the environment. The gesture recognition system can be divided into three parts, detection, tracking and recognition. Since while showing gestures, the hand is closer to the webcam than other parts of the body. the system determines the the hand by judging the part that closest to the webcam. Extract the depth map from the webcam. The pixels that have the smaller depth value means that there are closer to the webcam. First exclude the pixels with a depth value of 0, since a depth value of 0 means that it is occluded, so it cannot be in the area of the hand. Then set a critical value, if the depth value of the pixel is less than the critical value, it is judged as in the area of the hand. Mark all these pixels as white and the remaining pixels are marked in black. Afterwards, using a square filter to segment the area of the hand for following recognition. Resize the image so that it can pass through the CNN network. The network contains two convolutional layers and pooling layer. In the convolutional layer, all images will pass the filter and the features of the image will be extracted. In the pooling layer, all the extracted features will be summarized and used for deep learning. Finally, the neurons in the output layer will recognize gestures in the image.[4][5]

3.4 Software Environment

This system software environment includes the programming language of **Python** alongside operating system **Ubuntu 18.04** and **20.04** with **ROS Noetic** open source operating system. Furthermore, system modelling and aspects of the testing stage of the project has been completed utilizing software **Gazebo**, **RViz** and **Turtle-Sim**, **GMapping (SLAM)** environment.

3.5 Hardware Environment

This system hardware environment includes a **computer** that will run the program, a **webcam** that detects hand gestures, a **TurtleBot** that will be the main robot which will demonstrate movement via provided hand gestures, a mounted **LIDAR** upon the TurtleBot along with a **Pi Cam** as the camera mounted as well.

3.6 Gesture Selection

The American Sign Language is chosen as the gesture data set of our system. This language widely used by people with disabilities such as hearing issues. This sign language is very vivid and easy to understand. For example, the stop gesture used by our system is the same as the stop gesture used by the police while directing traffic. In addition, the difference between gestures is obvious, which is conducive for machine learning to capture features, thereby improving recognition accuracy. Our system can recognize 7 kinds of gesture representing 7 different command: Move forward, Move backward, Turn left, Turn right, Stop moving, Move backward, speed up and slow down. Shown below.

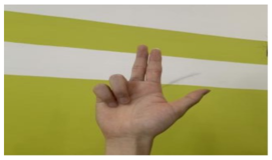
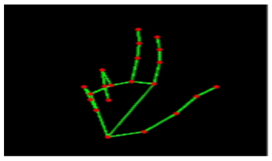
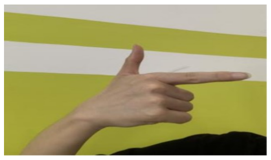
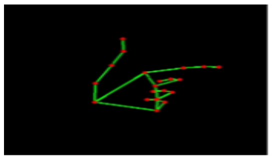
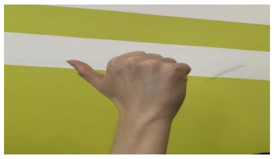
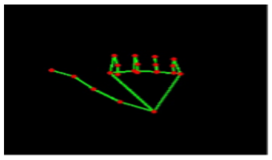
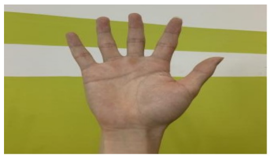
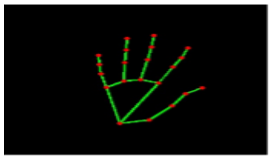
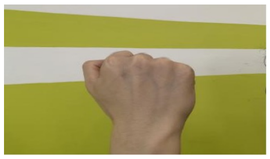
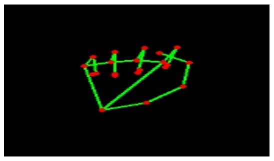
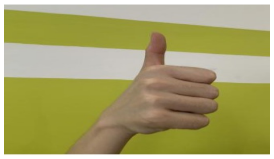
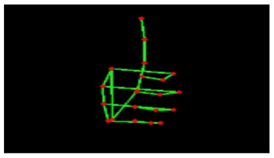
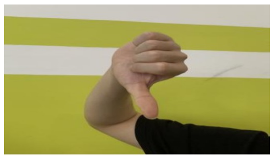
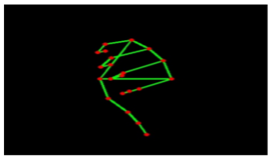
Gesture Meaning	Gesture in Real World	Gesture with Landmark
Forward		
Left		
Right		
Stop		
Backward		
Fast		
Slow		

Table 1: Gesture table

4 Experimental Setup

4.1 Systems Design

4.1.1 Gesture Recognition

To program the gesture recognition aspect of this system, we have elaborated upon and utilized the **MediaPipe** hand gesture recognition library. Instead of taking large number data sets, MediaPipe allows us to isolate the hand skeleton and use that as an input to the neural network deep learning model. As all background and other complexities are eliminated, the model gives a high accuracy result from very limited resource data set. This allows the model to be rapidly implemented and integrated to different project and usage while still maintaining the flexibility for user to assign and hand gestures wanted to be used.

Furthermore, this report introduces a gesture recognition scheme from MediaPipe with modifications made and named as MediaPipe Hands. We have utilized this scheme and applied machine learning technology to locate 21 3D landmarks for the shown gestures in Figure 1. The whole MediaPipe Hands system contains two models, a hand detection model and a hand landmark model. The hand detection model can extract the area of the hand from the entire image and isolate it. Moreover, the hand landmark model takes the previously extracted hand area as input and returns 21 3D landmarks.

Different hand sizes and occlusions make it very difficult to detect fingers. Therefore, the hand detection model chooses to detect the palm because the palm has a specific shape and it is easier to detect than the finger. The application of non-maximum suppression algorithm can effectively solve the situation of making a fist or hands occlude each other. Then using a square bounding box to extract the area of the hand after detecting the palm position. Afterwards, the hand landmark model marks 21 3D landmarks on the extracted hands. Each landmark has coordinates and can be accurately located. The gesture can be judged by analyzing the coordinates.

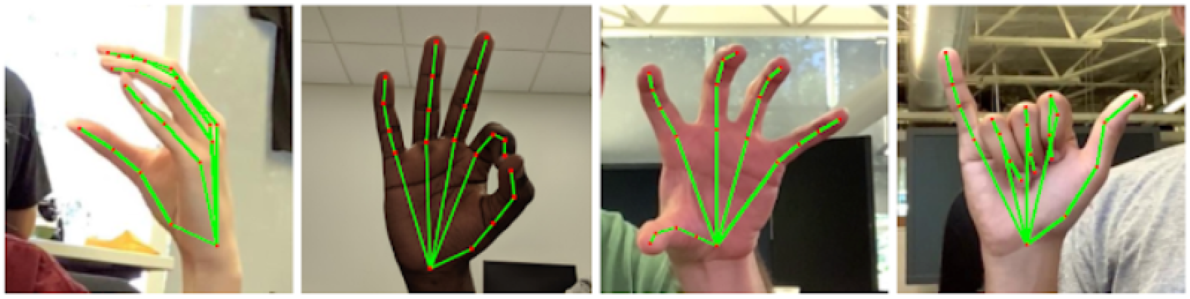


Figure 1: Hand Landmarks (Google.GitHub)

Data Collection

Python code to efficiently collect the image that will later be used as the training and testing data-set is establish, named CollectingIMG.py. The code utilized the OpenCV and MediaPipe library to save the image of hand skeleton in an isolated black background in a sequential series and number range prior set. The image is then saved to an allocated folder. For the purpose and scope of the project demonstration, 7 gestures is considered with ten sample image taken for each gesture. This include (1) forward, (2) backward, (3) accelerate, (4) decelerate, (5) left turn, (6) right turn, and (7) stop, as shown in Figure 2.

Data Preparation

The LabelImg software available at *https://github.com/tzutalin/labelImg* is then used. Using the software, the collected imaged are annotated into XML file in PASCAL VOC format that is used to train the NN.

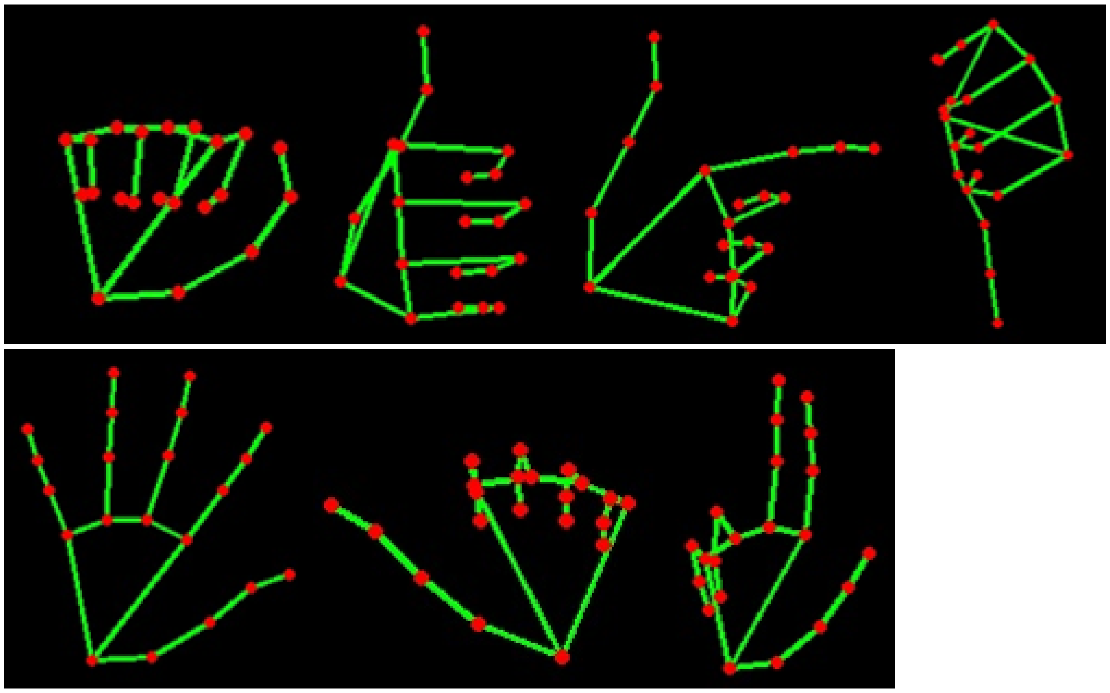
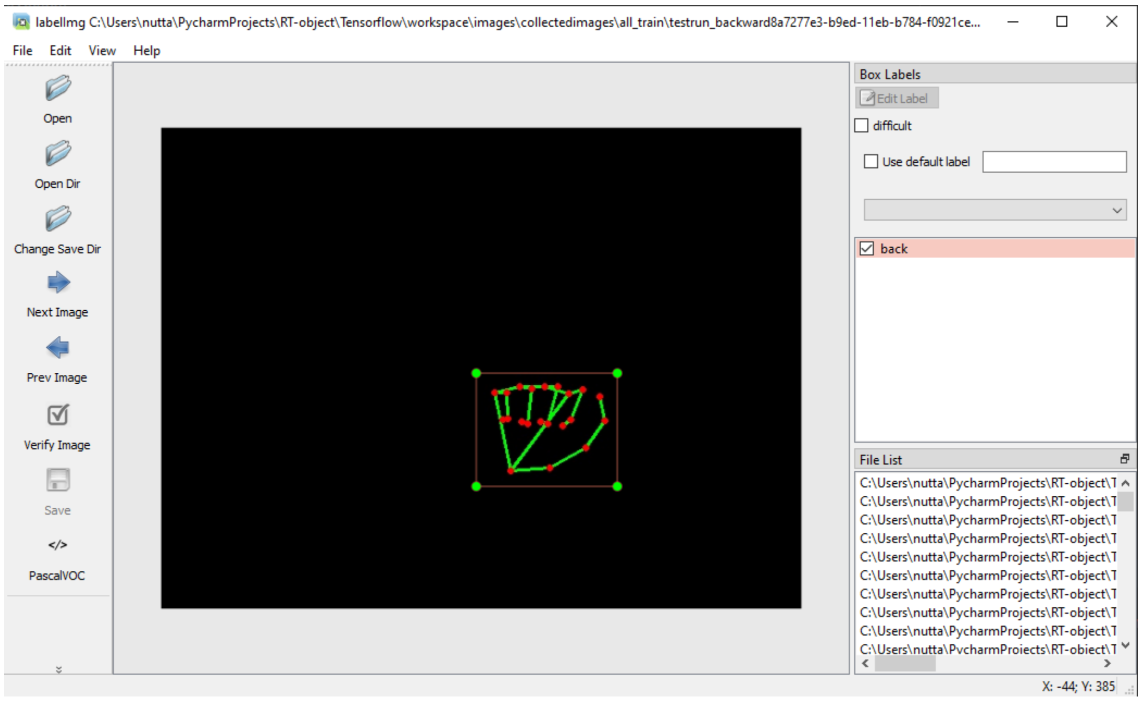


Figure 2: Different Hand Skeleton Poses used as Input Data-set to the Deep Learning NNC

Using the two file available (Figure 3), Tensorflow records for both the train data-set and record data set are created with the used of the standard generate_tfrecord.py available here With the records following the TFRecord format, the files can be directly implemented to train and test the standard Tensorflow Model Garden, from the respiratory *https://github.com/tensorflow/models*.



labelImg C:\Users\nutta\PycharmProjects\RT-object\Tensorflow\workspace\images\collectedimages\all_train\testrun_backward8a7277e3-b9ed-11eb-b784-f0921ce...

☑ back

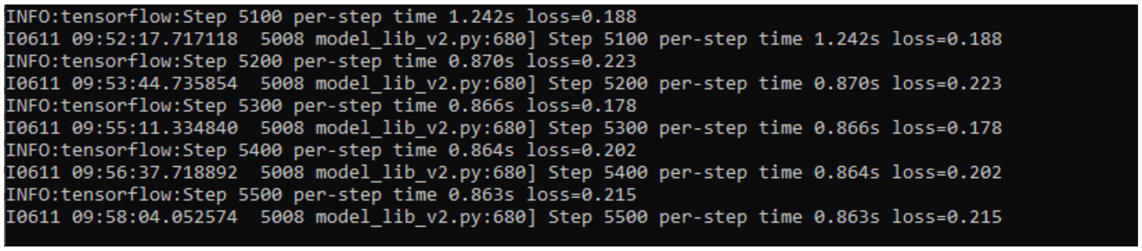
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T
C:\Users\nutta\PycharmProjects\RT-object\T

X: -44; Y: 385

Figure 3: Labeling Software - Image is Annotated into XML file in PASCAL VOC format that is used to train the NNC

Training and Using the Model

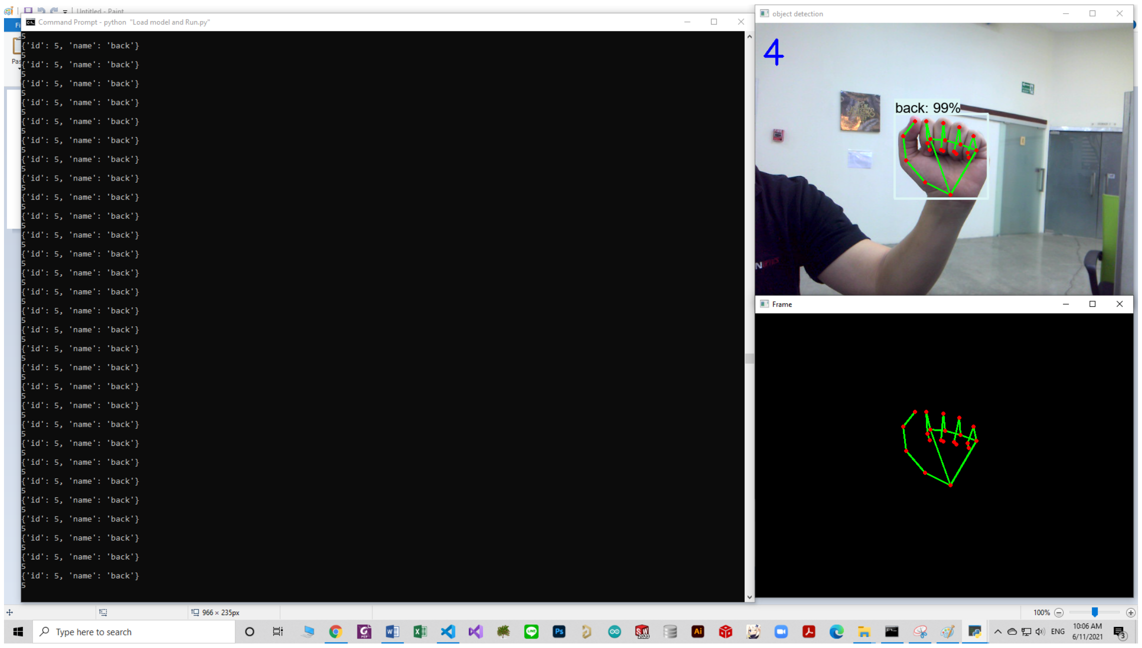
Using the standard object detection API available in the Tensorflow Model Garden respiratory as mean, the NN to detect, localized, and identify the object, which in this context a hand pose, can be simply construct, train, and deploy. The train and test records previously created were implemented to the object detection model available, and a training loop using batch size of 4 images with 5500 iteration was computed for the purpose of this project. Multiple checkpoint are being save into the workspace directories *Tensorflow\workspace\models\mt_ssd_mobnet*. At the end of the training session, the computed loss fluctuate from around 0.20, as shown in Figure 4.



```
INFO:tensorflow:Step 5100 per-step time 1.242s loss=0.188
I0611 09:52:17.717118  5008 model_lib_v2.py:680] Step 5100 per-step time 1.242s loss=0.188
INFO:tensorflow:Step 5200 per-step time 0.870s loss=0.223
I0611 09:53:44.735854  5008 model_lib_v2.py:680] Step 5200 per-step time 0.870s loss=0.223
INFO:tensorflow:Step 5300 per-step time 0.866s loss=0.178
I0611 09:55:11.334840  5008 model_lib_v2.py:680] Step 5300 per-step time 0.866s loss=0.178
INFO:tensorflow:Step 5400 per-step time 0.864s loss=0.202
I0611 09:56:37.718892  5008 model_lib_v2.py:680] Step 5400 per-step time 0.864s loss=0.202
INFO:tensorflow:Step 5500 per-step time 0.863s loss=0.215
I0611 09:58:04.052574  5008 model_lib_v2.py:680] Step 5500 per-step time 0.863s loss=0.215
```

Figure 4: CCN Training Result

Once completed, the trained model is then loaded and used to predict the hand gesture pose on real-time through the implementation of OpenCV and MediaPipe library. The hand skeleton resulted from MediaPipe library shown in Figure 5's bottom right corner window is then pass into the train NN model.



```
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
{'id': 5, 'name': 'back'}
5
```

Figure 5: Real-Time Hand Pose Final Prediction Result

The prediction is then reported back into the terminal as presented in Figure 5's left window. Due to the project in which high precision and accuracy is a priority, the MediaPipe library is configured to capture only a single hand skeleton, and the model accuracy minimum threshold was set to 0.90, or 90 percent.

4.1.2 TurtleBot Configuration

To configure the TurtleBot movement and satisfy the system required movement, velocity publishers have been employed to control speed and direction, to coincide with decided commands which are implemented or recognised by the gesture recognition aspect of the system. As part of the design decisions of the system, the assumption made in operation of the gesture control is to have a **forward** and **backward** hand gestures continuously perform the movement even when the hand gesture is no longer presented by the participant. In contrast, **turning** movements and **acceleration/deceleration** requires the gesture to be continuously denoted to the webcam/camera of the system. Therefore, allowing the turning and acceleration/deceleration control to be fully manipulable by the user of the system.

To publish directional control to the TurtleBot, the system design employs altering of the angular velocity message publishing existent within a switch or the index from the gesture recognition component. Furthermore, the forward and backward velocity control was designed through publishing linear velocity messages. The acceleration and deceleration publishing is accomplished using constant incremental velocity publishing embedded within both the situations that velocity changes is detect by the gesture recognition.

4.1.3 Integration

During the integration process, we must achieve following requirements to ensure system work perfect. Firstly, we need to understand two parts of the codes, gesture recognized model and TurtleBot movement.More specifically, Deep learning part and ROS part. Secondly, the first problem encountered is where to add the motion control logical conditions of the ROS part, Our primary design is to integral two part of code to minimize the workload and reduce computing space and consider what is the suitable motion control logical condition for different motions. First, we add an index variable, which will represent the id (Shown in table below) of the result of current recognition gesture.

Gesture	**ID number**
no Gesture(means following current state)	0
Forward	1
Left	2
Right	3
Stop	4
Back	5
Fast	6
Slow	7

Then the index is used for the condition judgment of motion control. Therefore, the overall logic is that every time an index (representing a new gesture) is obtained, it will enter the situation determination, and there will be different motion control logic corresponding to different situations. Then publish through ROS to control TurtleBot.

4.1.4 Real-Time Camera Feed

Real time camera feed from the **RPi** cam is integrated to the gesture control package. This is achieved by subscribing to the RPi cam topic *raspicam_node/image/compressed* that is published from TurtleBot's *rpicam* launch file following the command *roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch.* This allows the TurtleBot's view to be streamed on remote PC in real time, which extends the system's functionality. The feed in compressed image data is processed through OpenCV before streaming.

4.2 Experiment Setup

The whole experiment is mainly divided into three stages. The first is gesture recognition and accuracy improvement. This stage is mainly to find a suitable training model, collect required training data set and image processing to improve the accuracy of gesture detection. The second part is the model environment test of the entire system, combined with ROS and deep learning functions, test the feasibility of the system in simulated environments, such as Turtle-Sim and simulation TurtleBot. The last step is Physical Testing, testing through the actual TurtleBot to ensure that the system and the machine can work in the practical environment.

4.2.1 Simulated Testing

Gesture bot simulation is achieved with the help of existing ROS dependencies - Turtle-Sim (Figure 6) and Gazebo. During the development phase of the motion control part, the code was tested by publishing velocity commands to topic */turtleX/cmd_vel*, which is subscribed by the Turtle-Sim package, and the gesture input was simulated using user keyboard inputs from number 0 to 6. This is used to test how the simulated TurtleBot responses to velocity inputs, and certain adjustments were made to make sure the robot is able to performing turning, accelerating, decelerating, moving in straight lines, and moving in arcs. After testing the motion control subsection, the gesture recognition section was integrated to

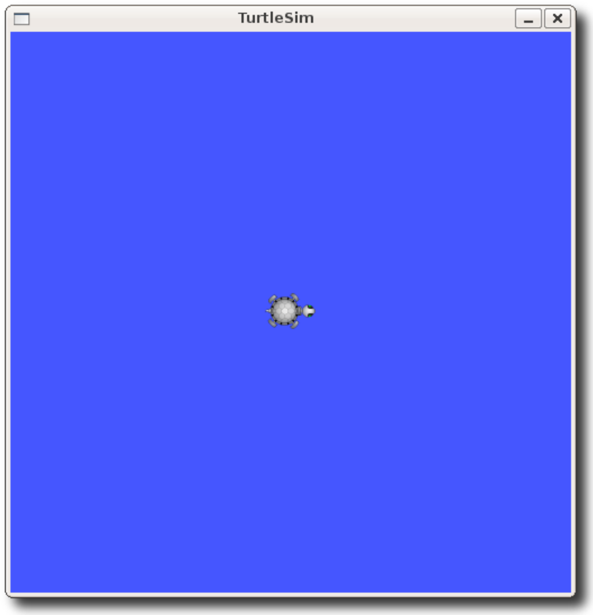


Figure 6: TurtleSim simulation

the code to form a completed gesture control model. This model was then tested using the **Gazebo** platform inside the **TurtleBot3 House** map as shown in Figure 7. To publish velocity commands to the simulated TurtleBot in Gazebo, the published topic is */cmd_vel*. With the integrated model, the motion of the robot was controlled by gesture inputs from the team members. This simulation tests how the robot responds to gesture inputs instead of simple keyboard input, and if any changing is required between the integration of gesture recognition and motion control.

4.2.2 Physical Testing

Unlike the simulated testings, in real world scenarios the actual TurtleBot responds to velocity commands differently. For example, 0.5 linear velocity would look slow when navigating in the simulations, however when the same velocity value is applied to the physical robot the speed outcome is in fact faster than desired. Physical testing is achieved by running **roscore** on the remote PC, and from the PC directly ssh into the physical TurtleBot's Raspberry Pi computer. Once roscore is running on the robot, the command *roslaunch turtlebot3_bringup turtlebot3_robot.launch* is executed, which enables the robot to establish

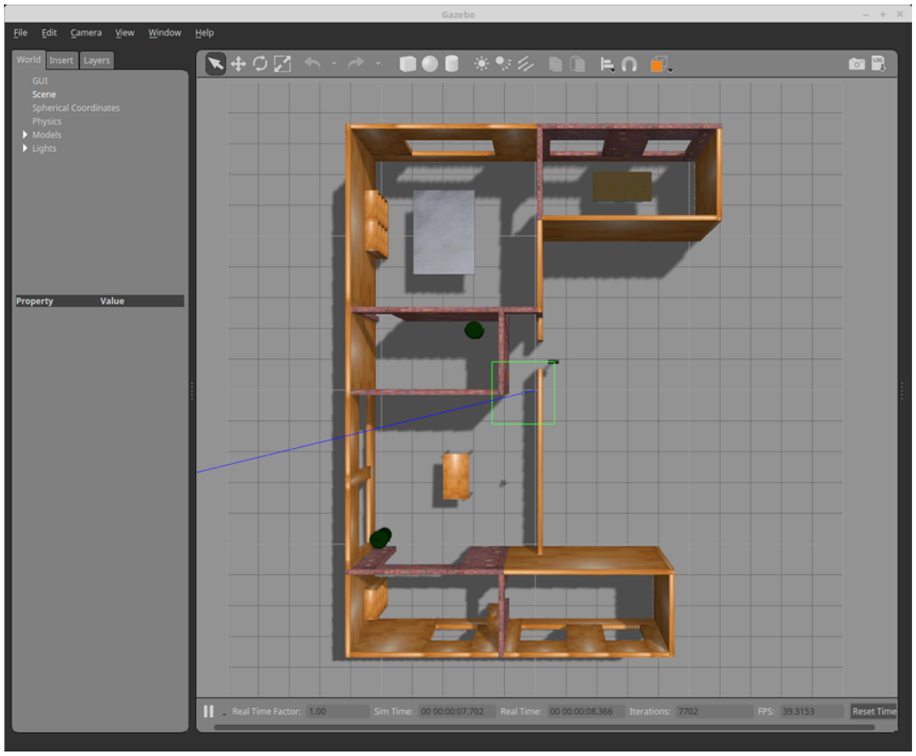


Figure 7: Turtlebot3 House map in Gazebo

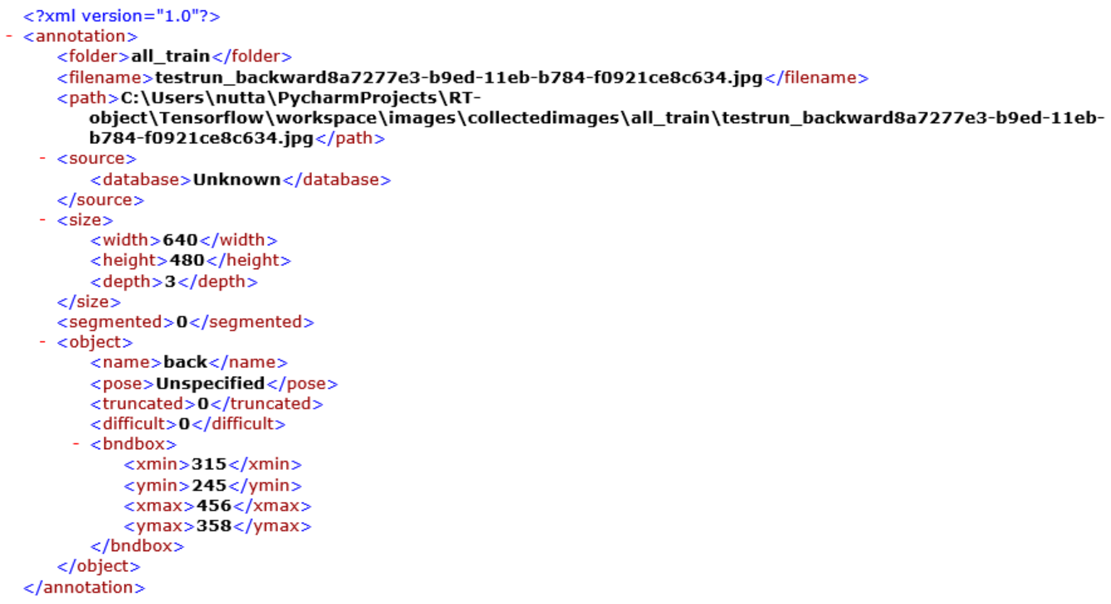
subscriptions and publications to different ROS topics including */cmd_vel*. The following command, *rosnode info*, is executed from the remote PC to ensure the TurtleBot node is successfully initialized.

After launching the TurtleBot node, the gesture control model is launched from the remote PC. The forward/backward gestures were first tested to tune the linear velocity of the robot. Compared to a simulated environment, the TurtleBot's response to velocity commands in real life tends to be too fast, and hence the default linear velocity was decreased to reach a desired speed. Next the left/right turn gestures were tested to tune the angular velocity. The acceleration gestures were tested at the end to ensure a smooth acceleration can be performed by the TurtleBot.

5 Result

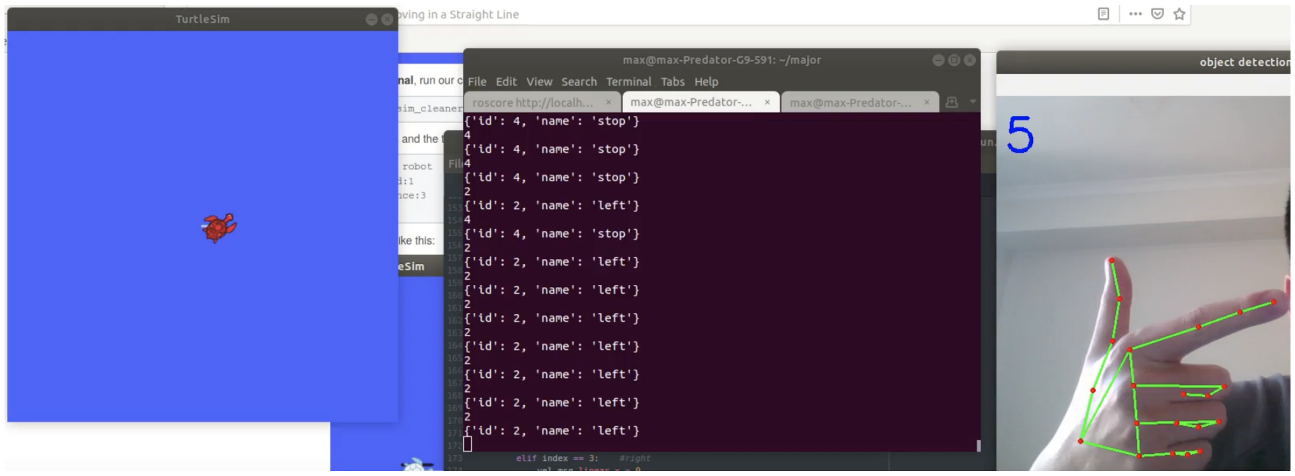
5.1 Gesture Recognition

From the tested result, an accurate prediction of hand poses was achieved for all 7 different hand gestures, considering the very limited amount of data-set being used to train and test the model. This is mostly due to the fact that MediaPipe allow the processed image to be minimized; the isolation of the hand skeleton from the background and other contour present that usually add complexity, and thus affect the model accuracy. With the result, it is clear that the used of skeleton recognition NN universally available such as MediaPipe can be implemented as alternative or in complementary image filtration mean to quickly and accuracy created a NN model that can be further implemented into different project.



```
    <?xml version="1.0"?>
  - <annotation>
        <folder>all_train</folder>
        <filename>testrun_backward8a7277e3-b9ed-11eb-b784-f0921ce8c634.jpg</filename>
        <path>C:\Users\nutta\PycharmProjects\RT-
           object\Tensorflow\workspace\images\collectedimages\all_train\testrun_backward8a7277e3-b9ed-11eb-
           b784-f0921ce8c634.jpg</path>
      - <source>
           <database>Unknown</database>
        </source>
      - <size>
           <width>640</width>
           <height>480</height>
           <depth>3</depth>
        </size>
        <segmented>0</segmented>
      - <object>
           <name>back</name>
           <pose>Unspecified</pose>
           <truncated>0</truncated>
           <difficult>0</difficult>
         - <bndbox>
              <xmin>315</xmin>
              <ymin>245</ymin>
              <xmax>456</xmax>
              <ymax>358</ymax>
           </bndbox>
        </object>
    </annotation>
```

Figure 8: Labeling Software Interface & XML Annotated Script Output



```
{'id': 4, 'name': 'stop'}
4
{'id': 4, 'name': 'stop'}
4
{'id': 4, 'name': 'stop'}
2
{'id': 2, 'name': 'left'}
4
{'id': 4, 'name': 'stop'}
2
{'id': 2, 'name': 'left'}
2
{'id': 2, 'name': 'left'}
2
{'id': 2, 'name': 'left'}
2
{'id': 2, 'name': 'left'}
2
{'id': 2, 'name': 'left'}
2
{'id': 2, 'name': 'left'}
2
{'id': 2, 'name': 'left'}
```

Figure 9: Integration result

5.2 Integration

The integration result is based on the experiment design which need to achieve two aims.

- Get the id of each gesture accurately
- In a simulated environment, use Turtle-Sm to test simple gesture commands, such as forward and backward. To ensure the normal operation of the entire system

The final integrated gesture control system setup is shown in Figure 9, the integration part is working perfect, the Turtle-Sim could do basic movement according to the gesture control and in the terminals, it shows current gesture meaning and corresponding ID.

5.3 TurtleBot Configuration

The results of the TurtleBot velocity and direction control was successful with physical tests in relation to the defined requirements of the system. The configuration of TurtleBot therefore, operated as expected with forward and backward remaining at a defined 0.1 and -0.1 respectively. Furthermore, it has behaviour as expected with the forward and backward movement being sustained when gesture is no longer displayed to the webcam (only for forward and backward gestures and movement situation). For the left and right turning being defined as 0.3 and -0.3 respectively. Furthermore, the turning configuration operated according to set goals of the system being a slow rotation that allows accurate manipulation for the user to control the TurtleBot's direction with upmost flexibility and desirability (depending on the user). With regard to the acceleration and deceleration of the TurtleBot, the system is limited to a maximum and minimum velocity and increments of the velocity values therefore, the acceleration and deceleration of the TurtleBot displays a elapsed low time to reach maximum or minimum velocity. The defined increments of velocity for each of these functions are 0.05 and -0.05 for acceleration and deceleration respectively.

6 Discussion

With the Mediapipe library, each of the 21 landmarks on the hand is present, and can be located in term of x and y position within the frame reference. Using the known position, only the hands section that is focused can be cropped and put into the deep learning neural network. Doing so will allow the processing image to be smaller. This thus, should reduce the processing time necessary. From experimental development, however, the time saved from doing so does not provide significant increase in the overall frame rate and that the accuracy of the overall prediction of the network model had reduced. This thus was not implemented into the final iteration of the system

Compared with the gesture recognition system reviewed in the background section, our system has higher recognition accuracy with a relatively smaller data-set. The recognition of all gestures maintain an accuracy of above 90 percent. Our system can also recognize gestures of different sizes and allow a slight distortion of the gestures. At the same time, our system has a very fast response time. Once a gesture is detected, the TurtleBot will respond in a very short time. In addition, the system can also filter background noise. However, the system still has some limitations. Our system cannot recognize two gestures at the same time. For example, when two gestures appear in front of the webcam, the system will only recognize the gesture that is closer to the center of the webcam. In addition, the system cannot make accurate judgments when the gesture is partially occluded.

A substantial drawback of this method still remain. From the result, after the stream image pas through the MediaPipe NN, the frame rate substantially drop from around 60 FPS to 30 FPS, and again after the gesture prediction NN to around 4 FPS base on CPU performance. Resizing and cropping of skeleton image before sending it to the prediction model show no improvement on the frame rate.This show that each NN took considerate amount of time to computed, and thus the overall method may not be suitable in situation where the CPU was limited or where GPU was not available, such as for example raspberry pi, etc. The GPU can be used to further accelerate the computational processing time, however, was not investigated within this project scope.

With regard to configuration of the TurtleBot movement, the resulting velocity limits that are met upon acceleration and deceleration were not expected to be reached with such a low elapsed time. Furthermore, the limits themselves are of a lower value then expected upon proposing the acceleration and deceleration functions. During experimentation, another limitation was found when switching gesture inputs between Stop and Deceleration. During the gesture transformation from any other gestures to the Decelerate gesture, the recognition model occasionally interprets a Stop signal. It is possible that during gesture transformation, the hand gesture resembles the shape similar to a stop sign at some point. This shall be improved by adding input filters to either the train model or motion control model - input shall be validated only if detected no less than a certain number of frames.

7 Conclusions and Future Work

To further improve the performance of the system, what can be thought of is have the system be able to detect two gestures at one time. Set priorities between different gestures. For example, the stop gesture should have the highest priority. When two gestures show up at the same time, the TurtleBot should respond to the gesture with higher priority. In addition, the robot can move under a doubled acceleration or deceleration if there are two acceleration or deceleration gestures show up. And if the system can still recognize gestures and maintain high accuracy even when the gestures are partially occluded. In order to prevent other gestures that accidentally appear in the webcam from affecting the control of TurtleBot, the system should automatically exclude gestures that are too far away from the webcam. The functionality of building a map of the surrounding area of the TurtleBot can also be considered. And, if the TurtleBot can automatically stop moving or deflect an angle to avoid collision while detecting an obstacle ahead. Furthermore, there is an intelligent function can be considered, which is to add a facial recognition functionality so that people with hand disability can also driving the TurtleBot. Moreover, to relate greater to the scope of what was accomplished within this project, combination with SLAM GMapping and Real Time camera feed display would allow for a more accurate and functional surveillance robot, absent to the restrictions of having a physical controller.

Base on the hand recognition, multiple hand skeleton can be detected simultaneously using the MediaPipe library. This open multiple opportunity and possibility to the hand pose which can be used as mean of control. The skeleton images can be compact into series creating a possibility to be able to predict the actual hand gesture motion rather than hand posed for example waving, opening and closing hand, rotating of hand, etc. Using the landmarks available from the MediaPipe library as complementary testing point, a more accurate hand pose recognition, and even hand pose recognition can be achieved.

With reference to the TurtleBot movement and velocity configuration it would be recommended that in future development, the velocity limits be raised and the acceleration and deceleration function increments upon velocity be in some manner decreased. This would increase the time to reach maximum acceleration and deceleration as well as increasing the maximum and minimum values themselves.

References

[1] Mithileysh Sathiyanarayanan, Syed Azharuddin, Santhosh Kumar, and Gibran Khan. Gesture controlled robot for military purpose. *International Journal For Technological Research In Engineering (IJTRE)*, 1:2347–4718, 07 2014.

[2] Sunil Kumar Jena, Arjuna Rao B, and Yellagi B. Design and fabrication of hand gesture controlled wireless robot. *International Journal & Magazine of Engineering, Technology, Management and Research*, 2(7):216–218, 07 2015.

[3] Arkaprabha Lodh, Debopama Ghosh, and Debosama Ghosh. Accelerometer and arduino based gesture controlled robocar. *International Journal For Technological Research In Engineering (IJTRE)*, 7(8):9056–9063, 08 2018.

[4] Armin Dietz, Stephan Schröder, Andreas Pösch, Klaus Frank, and Eduard Reithmeier. Contactless surgery light control based on 3d gesture recognition. volume 41, pages 138–146, 09 2016.

[5] Vijay Javanjal, Ajeet Kumar, Durriya Korasawala, Gayatri Bari, and Aishwarya Narkhede. Design and development of gesture controlled robot. *International Research Journal of Engineering and Technology(IRJET)*, 6(5):5423–5425, 05 2019.

A Source Code

The source code for this project can be accessed from the following GitHub link: https://github.com/latlongheight/MTRX5700-Baton-Bot