

THESIS REPORT
ON
HAND GESTURE CONTROL OF A ROBOT USING INTELLIGENT
TECHNIQUES

Submitted in partial fulfillment of the requirement of
BITS F421T THESIS

BY

SAPTADEEP DEBNATH
2014AATS0061U

Under the supervision of

Dr. Alexander Gepperth
Full Professor (HDR), Computer Science Department
University of Applied Sciences Fulda, Germany

&

Dr. V. Kalaichelvi
Asst. Professor, Electrical & Electronics Department
BITS Pilani Dubai Campus, UAE



ACADEMIC RESEARCH DIVISION

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DUBAI CAMPUS, DUBAI, U.A.E.

January 2018 – May 2018

ACKNOWLEDGEMENTS

I express my sincere thanks and gratitude to our Director, BITS Pilani, Dubai Campus, Prof. Dr. R.N. Saha for the motivation, encouragement and support to pursue my Project. I would also like to thank Dr. Neeru Sood, Associate Dean of the Academic Research Division, BITS Pilani, Dubai Campus for her prompt support and encouragement in my thesis work.

I would like to express my deepest sense of gratitude, first and foremost, to my Supervisor Dr. Alexander Gepperth, Professor, Computer Science Department, Hochschule Fulda, Germany, and my co-supervisor Dr. V. Kalaichelvi, Asst. Professor, Electrical and Electronics Department, BITS Pilani, Dubai Campus, UAE, for their valuable guidance, support and encouragement during the course of this thesis. I am extremely grateful to them for their able guidance, valuable technical inputs and useful suggestions.

Last, but not the least, I am grateful to the Electrical and Electronics Department Faculty Members for their valuable suggestions. Above all, I thank the Lord for giving me the strength to carry out this work to the best of my abilities.

Saptadeep Debnath

2014AATS0061U

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DUBAI CAMPUS, DUBAI, U.A.E.**

CERTIFICATE

This is to certify that the Thesis entitled, '**Hand Gesture Control of a Robot Using Intelligent Techniques**' and submitted by **Saptadeep Debnath**, ID No. **2014AATS0061U** in partial fulfillment of the requirement of BITS F421T Thesis embodies the work done by him/her under my supervision.

Date:



Signature of Supervisor

Name- Dr. Alexander Gepperth

Designation- Full Professor (HDR)

Signature of the Co-Supervisor

Name- Dr. V. Kalaichelvi

Designation- Asst. Professor

LIST OF SYMBOLS & ABBREVIATIONS USED

1. ROS - Robot Operating System
2. LSTM - Long-Short Term Memory
3. RNN - Recurrent Neural Network
4. LIDAR - Light Detection and Ranging
5. SLAM - Simultaneous Localization and Mapping
6. RVIZ - ROS Visualization
7. BPTT - Back Propagation Through Time
8. MCL - Monte Carlo Localization
9. PCD - Point Cloud Data

BITS, Pilani – Dubai Campus
Dubai International Academic City (DIAC)
Dubai, UAE

Course Name: First Degree Thesis

Course No: BITS F421T

Duration: 4 months

Date of Start: 01.02.2018

Date of Submission:

Title of Report: Hand Gesture Control of a Robot Using Intelligent Techniques

Name: Saptadeep Debnath

ID Number: 2014AATS0061U

Discipline: Electronics and Communication Engineering

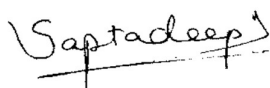
Name of Thesis Supervisor: Dr. Alexander Gepperth

Name of Thesis Co-Supervisor: Dr. V. Kalaichelvi

Keywords: Gesture Recognition, TurtleBot, Robot Operating System, RNN, LSTM

Research Area: Robotics and Intelligent Systems

Abstract: This research presents a unique application of hand gestures in the robotics field, to control the movement of a mobile robot using hand gestures. It is carried out in two folds – a study on machine learning algorithms to detect four classes of hand gestures and an elaborate investigation on the implementation of Robot Operating System on the research platform, TurtleBot. The initial part of this thesis revolves around the study of both the areas and preliminary tests. In recent times, a lot of emphasis has been put on the use of Recurrent Neural Networks for applications which have a temporal dependency. A major advantage of using RNN is its ability to adapt its network with the influx of new incoming data.



Signature of the Student

Date:

Signature of Faculty

Date:

ACKNOWLEDGEMENT

CERTIFICATE

LIST OF SYMBOLS & ABBREVIATIONS USED

ABSTRACT

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1 OBJECTIVE AND SCOPE.....1
1.2 MOTIVATION.....1
1.3 LITERATURE SURVEY.....1
1.4 ROBOT NAVIGATION – A GENERAL INTRODUCTION.....2

Chapter 2: ROBOT PLATFORM

2.1 HARDWARE COMPONENTS.....3
2.2 ROS: OVERVIEW.....3
2.3 OFFBOARD PROCESSING4
2.4 AUTONOMOUS NAVIGATION5

Chapter 3: GESTURE RECOGNITION USING LSTM NETWORKS

3.1 TENSORFLOW.....7
3.2 RNN: OVERVIEW7
3.3 DATASET8
3.4 LSTM ARCHITECTURE8

Chapter 4: EXPERIMENTAL REUSULTS ON LSTM NETWORK..... 10

Chapter 5: ROS ARCHITECTURE

5.1 MONTE CARLO LOCALISATION 17
5.2 DEPTH CAMERA 18
5.3 SYSTEM DEVELOPMENT. 19

Chapter 6: CONCLUSION AND FUTURE PLAN OF WORK 21

REFERENCES

LIST OF FIGURES

FIGURE 1: TurtleBot 2 (Research Platform)	3
FIGURE 2: TurtleBot – Host PC Network Configuration	5
FIGURE 3: Depth Map generated by the TurtleBot	6
FIGURE 4: A Recurrent Network	7
FIGURE 5: Input dataset representation	8
FIGURE 6: LSTM Network over time-steps	9
FIGURE 7: Depiction of input dataset and the certain ‘output frames’ in the Frames-axis	11
FIGURE 8: Variation of performance accuracy at different ‘output frame’	11
FIGURE 9: Variation of performance accuracy according to the split of training-testing data	12
FIGURE 10: Depiction of input dataset and the addition of ‘x’ frames in the Frames-axis.....	12
FIGURE 11: Variation of performance accuracy according to the number of frames prepended with (a) first frame and (b) zero frame	13
FIGURE 12: Depiction of input dataset and the addition of random sample in the Frames-axis.....	13
FIGURE 13: Variation of performance accuracy at two different instances’	14
FIGURE 14: Depiction of input dataset by embedding the video sample at n th index	14
FIGURE 15: Variation of performance accuracy according to the number of frames prepended with (a) zero frame and (b) random frame	15
FIGURE 16: Depiction of input dataset by embedding the video sample at a random index between 0 to n index	15
FIGURE 17: Variation of performance accuracy according to the number of frames prepended with (a) random frame and (b) zero frame	16
FIGURE 18: Depth Camera User Interface	18
FIGURE 19: Rostopics and Rosnodes published	19
FIGURE 20: Data flow as a whole system	19

CHAPTER 1: INTRODUCTION

1.1 Objective and Scope

The objective of this research is to send messages and control a robotic platform in an indoor environment, using dynamic freehand gestures, on the Robot Operating System (ROS) platform. It involves a backend machine learning approach to be implemented to train the system for recognizing the freehand gestures.

1.2 Motivation

The recent times have seen a huge technological leap in the field of extended reality, whether it be virtual reality or augmented reality. It involves human-machine interactions via computer generated signals and wearables. A major development in the extended reality area is the inclusion of hand gesture recognition as a sensory input [1, 2], which can easily be implemented with the help of a depth sensor (for example Kinect sensor). The applications of hand gesture in day-to-day life are vast. A few instances where it can be implemented are, home automation system: switching the light off with a wave of a hand, increasing or decreasing the volume of the music with a twist of the index and the thumb, and many more. Extensive research is being carried out all over the world to increase the performance of recognition of these free hand gestures. However, we explore new applications in which, these rather easy to collect data and implement, free hand gesture can be applied. The main focus is on controlling a robot through hand gestures and navigating it through a known environment autonomously.

1.3 Literature Survey

Control of a robot through hand gestures is a fairly new development which is gaining popularity in the robotics area. Recent development in hand gesture recognition has seen the use of time-of-flight sensors [3], which are a plane based sensor in contrast to the point based LIDAR sensors which are generally used. In this research deep neural network was used to train the system, to recognize the hand gestures. In regards to the hardware aspect of the project, a model proposed by researchers from Basra University, Iraq, integrated the detection and recognition of hand gestures, to the movement of a mobile robot. By using 15 template vectors, they were able to achieve a 'recognition rate' of 98%. A similar research conducted was done in Gheorghe Asachi Technical University, Romania [4], in which they implemented a novel idea of incorporating the hand position in the 3D space

with the hand gesture. The aim of the thesis is to use image processing and deep neural network techniques to adequately detect and recognize the hand gestures. Pass on the message as a signal to control the robotic platform in an indoor environment, which will be implemented on the Robot Operating System.

1.4 Robot Navigation – a General Introduction

Robot navigation is the ability of the robot to move from a starting point to a finishing point according to its frame of reference. It includes three major components – awareness of the surrounding, path planning and the ability to move. A robot generally has a sensory input device (for example camera), the data from which can be processed and utilized to localize itself in an unknown environment. Path planning is a vital part of robot navigation, in which the robot produces a continuous motion from the starting point (S) till the goal point (G), using the localization map generated by it. Last but not the least, the actuation provided by the motors and wheels for the motion of the robot itself.

CHAPTER 2: ROBOT PLATFORM

2.1 Hardware Components

A TurtleBot as depicted in figure 1, is used as the robot platform for this research. The main advantage of using the TurtleBot is it's up to date open source firmware available on ROS wiki. It has three main off the shelf components, a mobile base – 'Kobuki' base, 3D sensor and an onboard computer. A sensor suite of Orbecc Astra camera is used as the 3D sensor in addition to the cliff sensor, wheel-drop sensor, and the bumper sensors. The 3D sensor is used to create the depth map which can later be utilized for autonomous navigation. The system is also equipped with a 16 GB RAM Intel NUC (Next Unit of Computing) running on Intel 4th Generation Core i5 processor. The onboard computer runs on the Ubuntu 16.04 with ROS Kinect distribution. It has a 4S1P 2200 mAh battery which provides 3 hours of operational time.



Figure 1: TurtleBot 2 (research platform)

2.2 ROS: Overview

ROS or Robot Operating System is an open source meta-operating system for a robot platform [7]. It provides services that are expected from an operating system like, package management, hardware abstraction, device control and relay of messages between

processes. ROS has the capability to run a common thread through multiple systems and provides a reliable communication between multiple processes that may or may not take place on a single computer. It has proved to be a stable platform, due to the interoperability provided by the message passing interface, enabling it to interface to both latest hardware and cutting-edge algorithms. This paves the way for a higher learning curve for developers. Well planned ROS architecture carefully separates the low-level control of the hardware and the high-level implementation of the algorithms. This helps in using a hardware simulator program, such as Gazebo, which can temporarily replace the hardware dependency and hence run the high-level algorithms in a simulated environment. The advantage of using ROS over the other meta-operating systems meant for robotics is its widespread support across the robotics community.

2.3 Off-board Processing

ROS Master is an integral part of the ROS architecture [8]. As the name suggests, it acts as the naming and registration service for the rest of the nodes in the ROS system. The two main functions carried out by the ROS system, publishing data and subscribing to data, is carried over by the instructions provided by the ROS Master.

Though the onboard computer, Intel NUC is considered to be one of the high-level micro-computers, it is still not enough to compute the amount of data processing required by this research. A more efficient way to handle the cluster of processing required in this research is by off-board processing. Off-board processing is a term used for a situation when, some raw data collected by the onboard sensor suite is parsed onto an off-board computing station, which is then processed upon, and required triggering outputs are sent to the onboard processor for further actions. At this earlier stages of research, the stereo images, which are essentially the frames of the video input from the stereo camera, are processed further to produce a depth map. This level of processing needs a high amount of computational power, which is not fulfilled by Intel NUC, without adding onto some low latency issues.

Therefore, the main part of this research is concentrated on utilizing the maximum out of the ROS architecture. A system variable, `ROS_MASTER_URI` allows to run the ROS master on more than one system, which in our case is the TurtleBot and the Host PC. By

changing this ROS environment variable in the ‘source’ file on both the TurtleBot and the Host PC, it instructs the Host PC to run the same ROS Master as the TurtleBot, which is done over the Wi-Fi. This enables the Host PC to launch several application present on the TurtleBot. The following commands are implemented:

On TurtleBot-

```
>> echo export ROS_MASTER_URI=http://localhost:11311 >> ~/.bashrc  
>> echo export ROS_HOSTNAME=IP_OF_TURTLEBOT >> ~/.bashrc
```

On Host PC-

```
>> echo export ROS_MASTER_URI=http://IP_OF_TURTLEBOT:11311 >> ~/.bashrc  
>> echo export ROS_HOSTNAME=IP_OF_PC >> ~/.bashrc
```

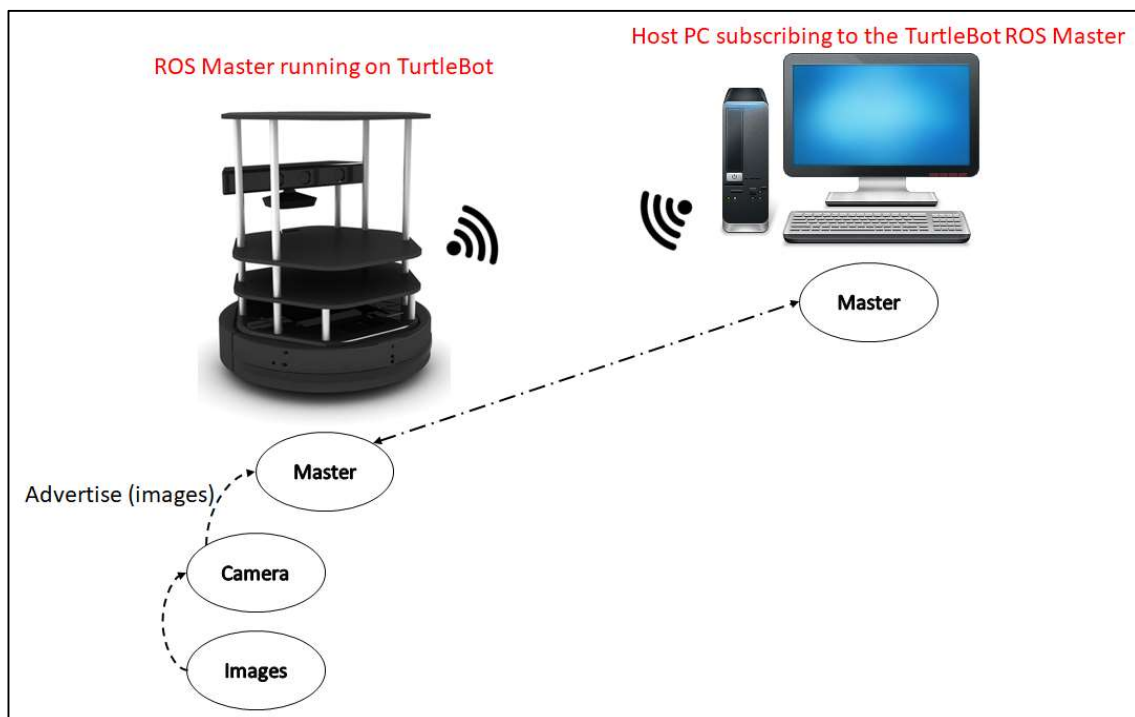


Figure 2: TurtleBot – Host PC Network Configuration

2.4 Autonomous Navigation

The network configuration as explained in the previous section allows the system to process the input from the stereo camera on the Host PC with a higher computation capability. The

stereo footage acquired by the Orbecc Astra camera is processed further to generate a depth map on the Host PC.

First, a minimal software available in the TurtleBot package is launched which initializes the basic hardware devices of the TurtleBot, the Kobuki base, Orbecc Astra camera and the additional sensors. After that, SLAM (Simultaneous Localization and Mapping) technique is used to map the environment using the input from the stereo cameras and visualized on the Host PC in an RVIZ environment. All this data is processed as a depth map as shown in figure 3, which is further used as a reference for the robot platform for autonomous navigation. For initial testing phase, the robot uses the Monte-Carlo Localization algorithm for localizing itself and for moving from point A to B in a known map.

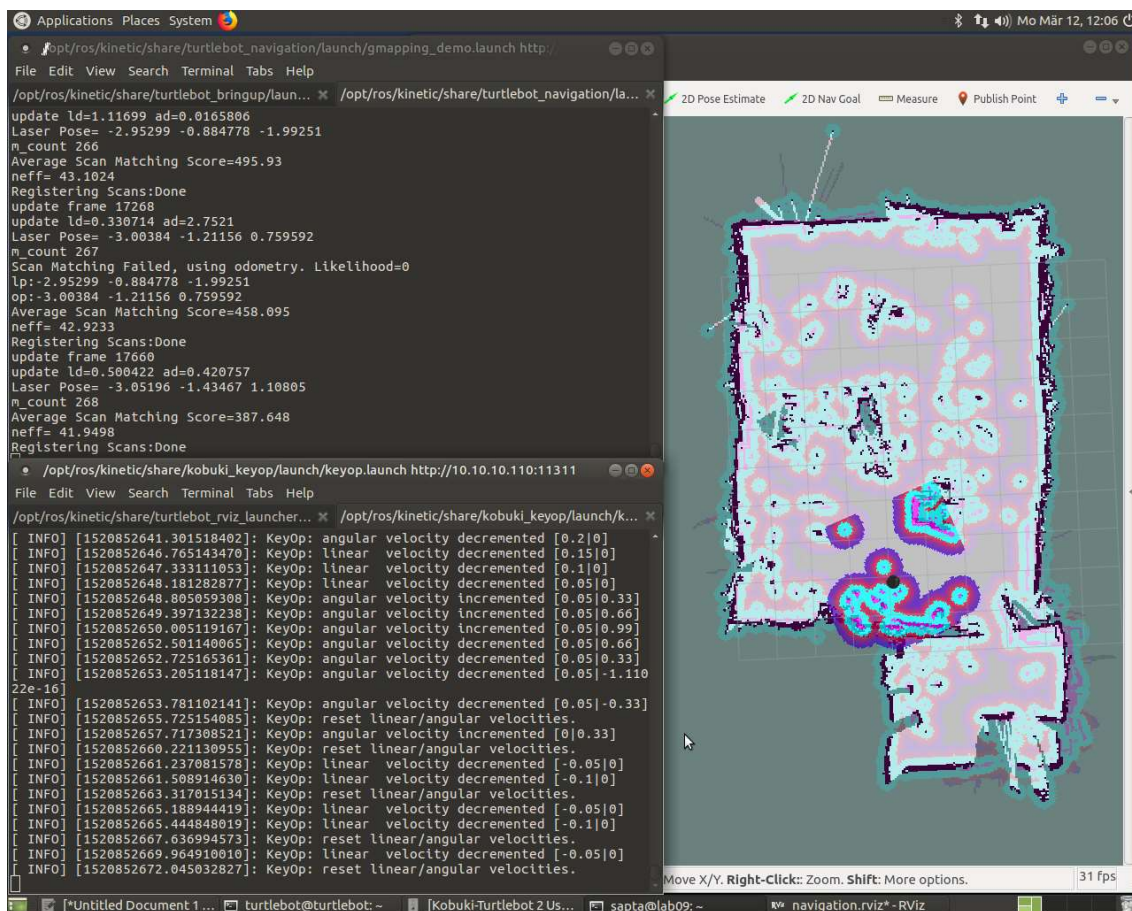


Figure 3: Depth Map generated by the TurtleBot

CHAPTER 3: GESTURE RECOGNITION USING LSTM NETWORKS

3.1 TensorFlow

TensorFlow is an open-source symbolic math library released by Google, used for practice and application of Machine Learning Algorithms. It is continually updated, developed and maintained by Google, which ensures its high effectiveness. Since being open source, the contributions by various researchers are also openly available for further development. TensorFlow uses a single dataflow graph to represent all computation and state in a machine learning algorithm models [9]. The TensorFlow network is based on multi-dimensional arrays or also known as tensors, which are capable of holding complex data.

3.2 RNN: Overview

Most of the real-life examples in the domain of image and video processing are in form of sequential data. The phenomenon when the next time step data depends on the previous time step data, or in other words temporal dependency, is categorized as sequential data. To overcome such types of research problems RNNs or Recurrent Neural Networks are used [10]. In recurrent networks, previous time steps are represented by neurons with a recurrent connection, as shown in figure 4. The history length in a recurrent network is unlimited, but it is possible to compress that in a low dimensional space. It is also possible to form short-term memory, so as to deal better with position invariance.

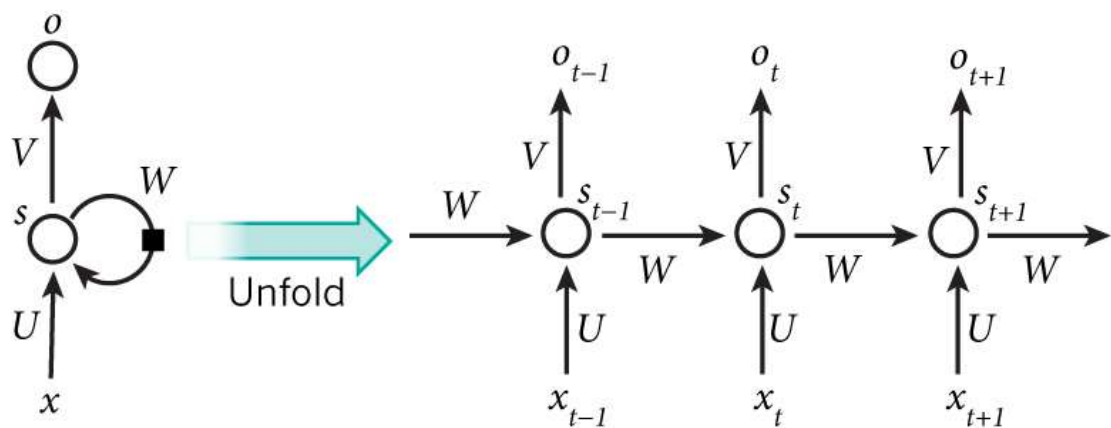


Figure 4: A Recurrent Network

3.3 Dataset

The dataset of the hand gestures comprises of four actions – close hand, open hand, pinching-in, and pinching-out. Each of the hand gestures are treated as classes, therefore it makes up for four classes in this dataset. A total of 390 video samples collected by 10 people are computed, with each video sample consisting 40 frames and each frame with 625 descriptors sampled in a histogram. The TensorFlow program takes in the array of $390 \times 40 \times 625$ as the input dataset as shown in figure 5. The gestures described in the input data can be categorized to have temporal dependence, as the consequent frame depends on the previous frame, it is because of this reason why RNN with an LSTM network is suitable for such type of a problem.

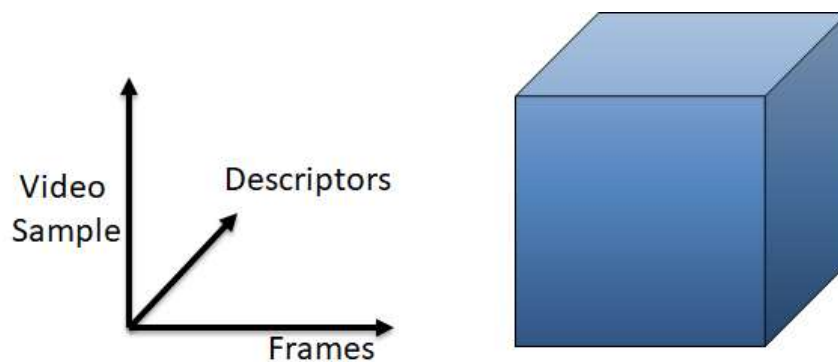


Figure 5: Input dataset representation

3.4 LSTM Architecture

RNNs are used extensively with Back Propagation Through Time or BPTT. The gradient calculation of a layer in RNN, proceeds from back to front, thus the term used back propagation. RNNs with BPTT thus suffer from a phenomenon called vanishing gradient problem. It can be explained as following, let an RNN have four hidden layers such that the gradient of the hidden layers be 0.3, 0.4 and 0.5. According to this, the gradient of the first hidden layer will be 0.006. It is thus seen that as the network becomes deeper the gradient towards the initial layers vanishes, and thus the initial layers train slower than the final layers. To overcome this effect of diminishing gradient, Hochreiter proposed LSTM Networks or Long-Short Term Memory Networks [12].

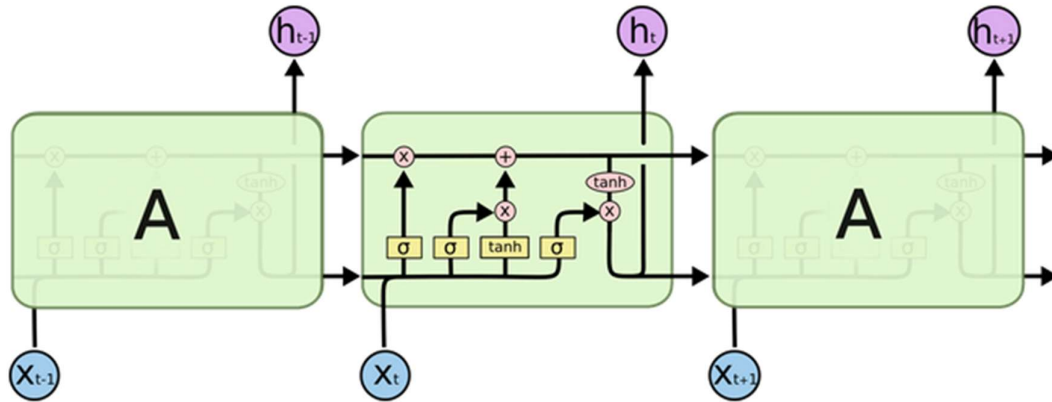


Figure 6: LSTM Network over time-steps

A single-layered LSTM network is depicted in figure 6. Here x_t refers to as the input at time t . The input of a time-step is given by the output of the previous time-step. The cell state or the C_t , referred to as the memory of the cell is unique to a particular neuron. It is particular to note that, as in a general Neural Network there are hidden layers, in an RNN with LSTM there are memory blocks. Similarly, as in a general Neural Network, neurons are activated, in an RNN with LSTM memory cells are activated. In this research, we focus on deep RNN with multiple LSTM approach.

CHAPTER 4: EXPERIMENTAL REUSULTS ON LSTM NETWORK

A Recurrent Neural Network with LSTM is implemented on the dataset as mentioned in Section 3.3. Six specific experiments are carried out on the given dataset:

- Testing the effect of ‘output frame’ variable, which is the frame for which classifier output is computed.
- Evaluating the effect of training the network person-by-person, i.e. training the network with the data from person 1-8 and testing the network with the data from person 9-10. The other case being, a randomize data for both training and testing.
- Determining the dependence of classification on onset delay with variable frames.
- Testing if the classification depends on what has been classified before.
- Repeating the test, for the determining the dependence of classification on onset delay but with a definite number of frames.
- Evaluating how well the network handles randomness.

The default training parameters are set as following:

- Batch Size (default = 10); the subset size of the training sample used in order to train the network in its learning process,
- Layers (default = 3); the number of LSTM layers in the memory block,
- Cells (default = 20); the number of cells in each LSTM layer,
- Iterations (default = 10000); the number of training iterations,
- Output Frame (default = 39); the frame for which the classifier output is computed,
- FracTest (default = 0.2); the proportion of test samples.

Experiment No. 1 – The first experiment is performed to determine the change in performance accuracy with respect to the ‘output frame’. As explained earlier, ‘output frame’ is the frame for which the classification is evaluated. The ‘output frame’ parameter is varied from 10 to 40 in the steps of 10, as shown in figure 7. The trend between the Output Frame and the performance accuracy is depicted in figure 8. It is clearly seen as the performance accuracy increases with the increase in the output frame, which is intuitively correct as a higher ‘output frame’ provides more information for testing than a lower ‘output frame’.

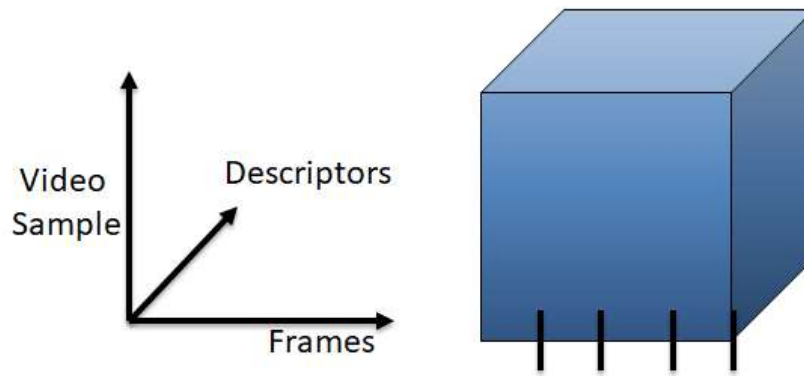


Figure 7: Depiction of input dataset and the certain ‘output frames’ in the Frames-axis

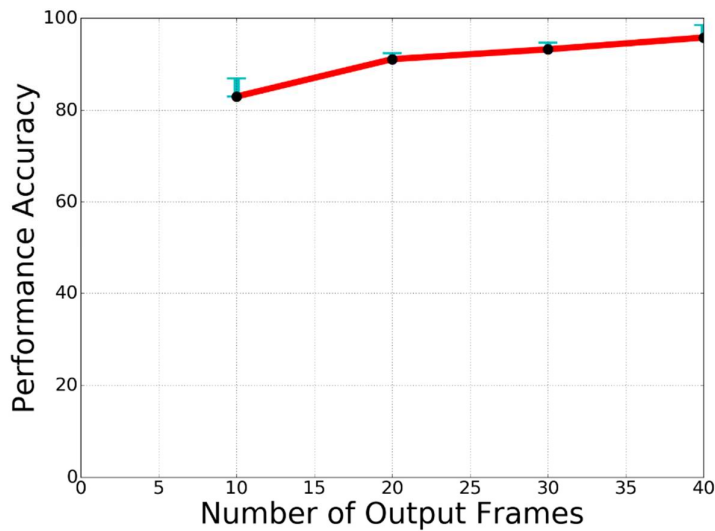


Figure 8: Variation of performance accuracy at different ‘output frame’

Experiment No. 2 – In this experiment, we observe the changes in performance accuracy with respect to the split in input training data, essentially when the training and testing data is split according to people in contrast to randomizing the data available. As explained before the dataset contains the gesture information from 10 people. So, in case 1 we take the data from persons 1-8 for training and 9-10 for testing. In the other case, the training and testing samples are randomized, having no dependency on the persons. It is observed, as shown in figure 9, that the performance accuracy for the test performed on randomized data is higher than that of the test performed on data being split person-by-person.

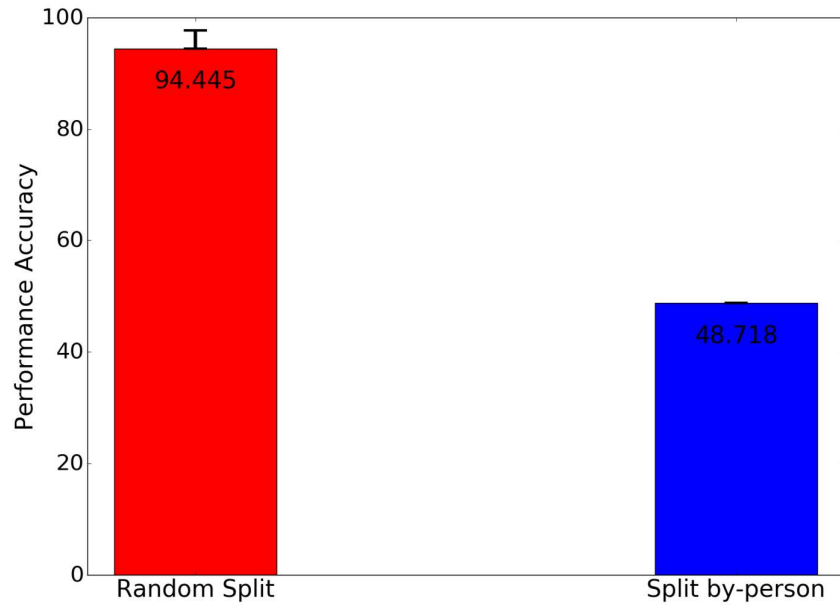


Figure 9: Variation of performance accuracy according to the split of training-testing data

Experiment No. 3 – Further on, we test the network for onset delay. In this experiment, ‘x’ number of additional frames are prepended (added before) to the video sample. Therefore the number of frames changes from 40 to $x+40$, where x ranges from 0-20, as shown in figure 10. The tests are performed by prepending two sets of frames, the first frame, and the zero frame. The classification is evaluated at the end of each sample giving us the trend as shown in figure 11. From the tests results, it is clearly seen that the onset delay does not have much effect on the performance accuracy.

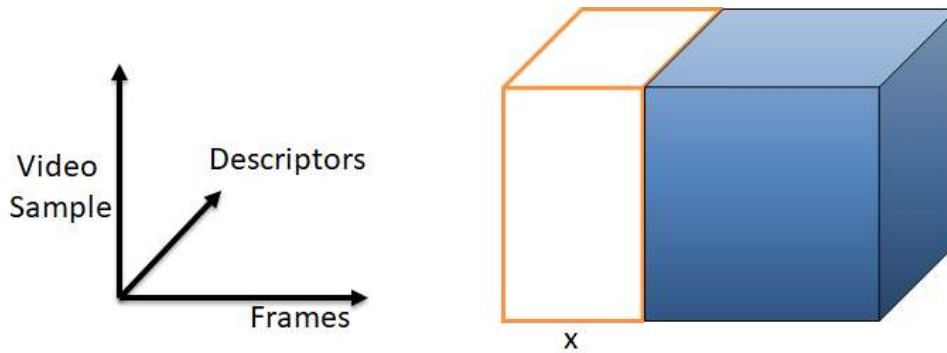


Figure 10: Depiction of input dataset and the addition of ‘x’ frames in the Frames-axis

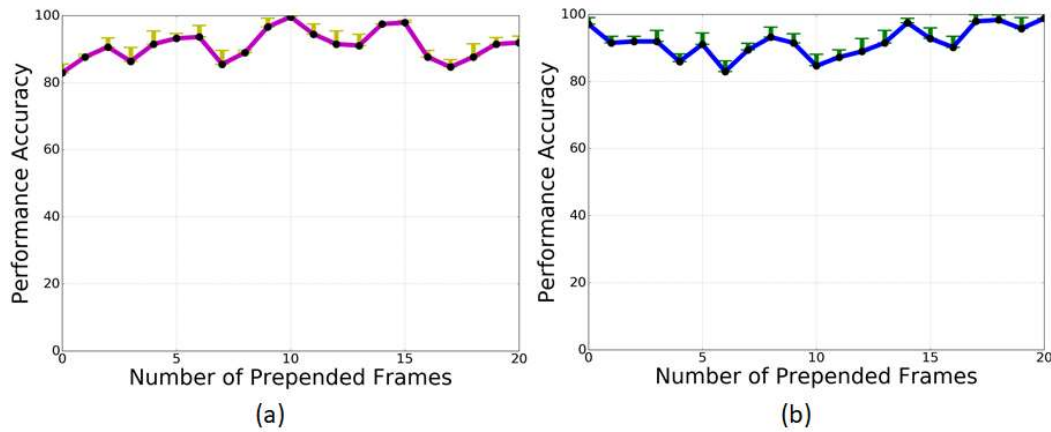


Figure 11: Variation of performance accuracy according to the number of frames prepended with (a) first frame and (b) zero frame

Experiment No. 4 – In the real-time implementation of the LSTM network, the video frames are not going to be constant (i.e. 40 as in the training and testing phase) but will be a continuous input from the camera. In the next experiment, we test if the classification depends on what has been classified before. A random video sample (40 frames) is prepended before the test sample, making it a total of 80 frames. The classification is evaluated at two instances, first at the 40th frame and second at the 80th frame. Figure 13 shows that the performance accuracy of the LSTM network clearly depends on the previous classification. To tackle this problem during the real-time implementation two measures have been taken. First, ‘k’ ($k > 1$) instances of LSTM network will be implemented simultaneously, with a delay after each network so as to fit in each of the frames in the classification. Secondly, after evaluating the classifier after each instance the network parameters will be reinitialized so as to remove its memory of the previous classification.

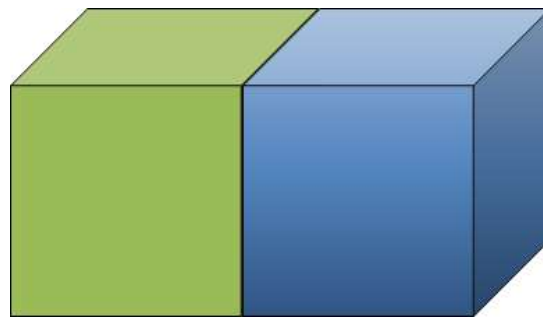


Figure 12: Depiction of input dataset and the addition of random sample in the Frames-axis

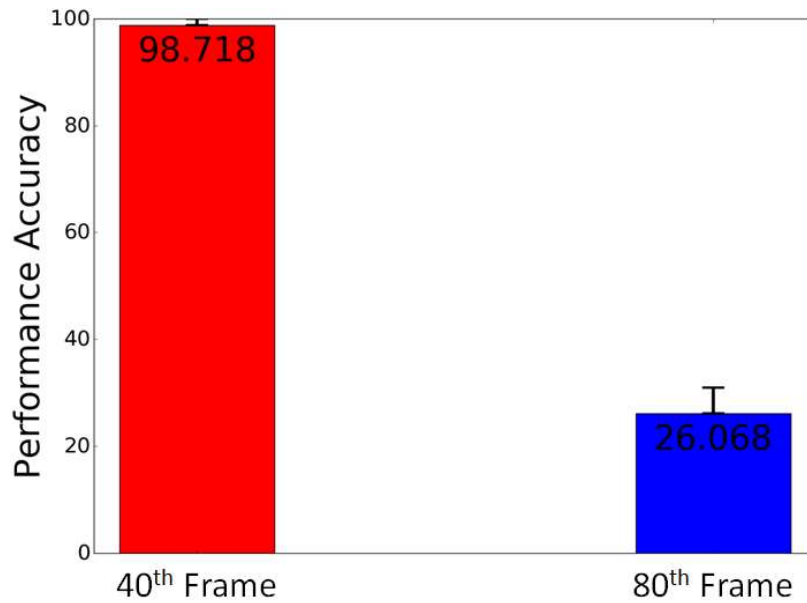


Figure 13: Variation of performance accuracy at two different instances

Experiment No. 5 – This experiment is carried out as a continuation of experiment no. 3, but by keeping the number of frames constant i.e. 60 frames. A sample set of 60 frames are initialized with (a) zero frames and (b) random frames. As shown in figure 14, the test sample is embedded at the n^{th} index, where n ranges from 0-20. Figure 15 (a) and (b) shows the trends between the performance accuracy and the n^{th} index at which the test sample is embedded. It is seen that the performance accuracy is not much affected by the position of the test sample in the 60 frame array.

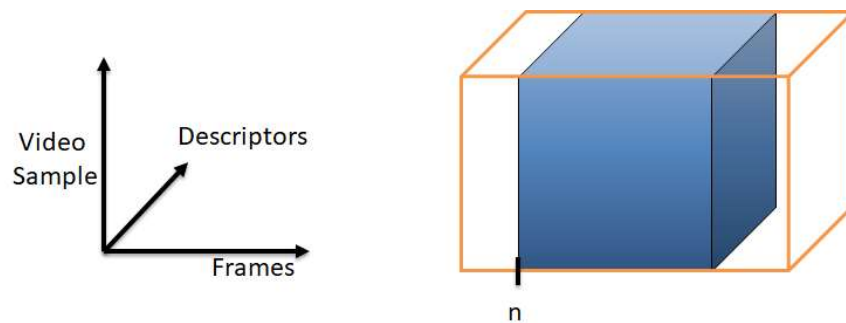


Figure 14: Depiction of input dataset by embedding the video sample at n^{th} index

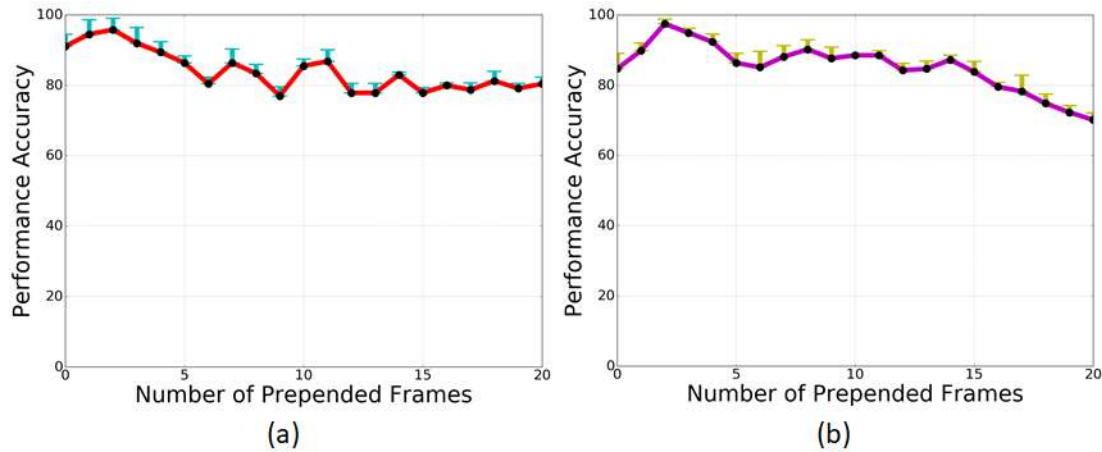


Figure 15: Variation of performance accuracy according to the number of frames prepended with (a) zero frame and (b) random frame

Experiment No. 6 – In this experiment, we test how well the network handles randomness. The sample set is initialized in the same way as experiment no. 5, with (a) random frames and (b) zero frames. Each test sample is embedded at a random index from 0-n, where n ranges from 0-20. This experiment is performed by varying the value of n from 0-20. Figure 17 depicts the trends of performance accuracy with the change in the value of n. The LSTM network handles the randomness with ease, as can be seen from the results.

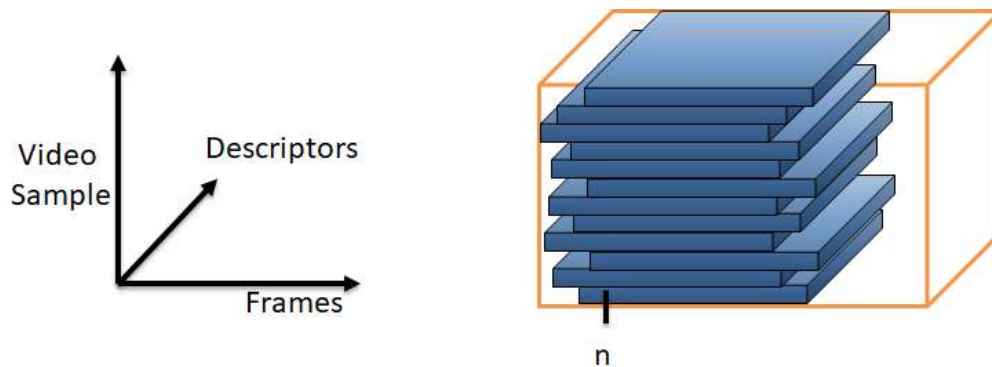


Figure 16: Depiction of input dataset by embedding the video sample at a random index between 0 to n index

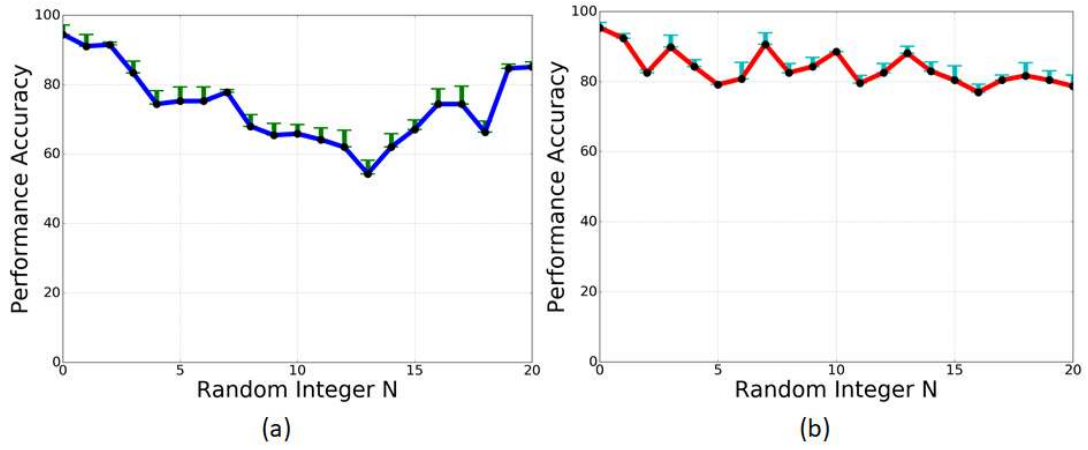


Figure 17: Variation of performance accuracy according to the number of frames prepended with (a) random frame and (b) zero frame

CHAPTER 5: ROS ARCHITECTURE

5.1 Monte Carlo Localization

In continuation to Section 1.4, the major component of autonomous robot navigation is the awareness of its surrounding, termed as mobile robot localization. It is the estimation of robot's location and orientation relative to its environment [13]. The two most challenging problems in the field of robot localization are *global localization problem* and *kidnapped robot problem*. In *global localization problem*, the robot is unaware of its initial pose (location and orientation) and has to determine it from scratch. The assumption of error in robot's estimate to be small fails in this instance, because of the large room for error. The *kidnapped robot problem* is much more challenging, in which a well-localized robot is teleported to an unknown environment.

Monte Carlo Localization or MCL is a probabilistic localization algorithm. The robot's estimate of its current state is called the *belief* which is a probability density function distributed over the state space. It is able to accommodate the non-linearity in robot's perception and consecutive motion, and is thus able to tackle the complex *global localization problem* and *kidnapped robot problem*.

After mapping the unknown environment by implementing Simultaneous Localization and Mapping (SLAM) on the Orbecc Astra Depth Camera, the resultant map is stored for further use as explained in Section 2.4. As can be seen in the [video](#), the robot is first initialized in the map on the ROS Visualization platform known as rviz. Next, for the ease of access, the robot's initial estimated pose is marked in the rviz user interface. Finally, a goal point and orientation is provided for the robot. The robot calculates an optimal path and follows it to the goal point.

During this research, an important aspect of the autonomous navigation "Dynamic Obstacle Avoidance" was also tested. The [video](#) shows that the robot is first initialized in the map, with known obstacles defined in it. In this experiment, we introduce an unknown 'still' obstacle (a box in this case). As can be seen at the beginning of the experiment, the obstacle is not known in the default loaded map. But the robot determines the obstacle right in front

of it, when following the path to its goal point. It then simultaneously determines the obstacle, localizes itself around it and calculates a new path to the goal point.

5.2 Depth Camera

For recognizing hand gestures, a Creative Senz3D camera is setup on the host PC. This Depth and Gesture Recognition Camera distributed by the company Creative is developed in collaboration with Intel. This lightweight camera has RGB video resolution of 720p (1280x720). With 30 frames per second and a 74 degree field of view, it is a perfect fit for the required application. The camera has windows 8 and 10 compatibility, but since the research is done on an Ubuntu platform, an external package (*/softkinetic_camera*) is utilized. Figure 18 shows the UI of rviz when the .launch file from the package is launched. The UI has 4 panes, monochrome image, colored image, depth image and depiction of 3D points in a 3D plane. This .launch file publishes various rostopics under the rosnode */softkinetic_camera*, as in figure 19.

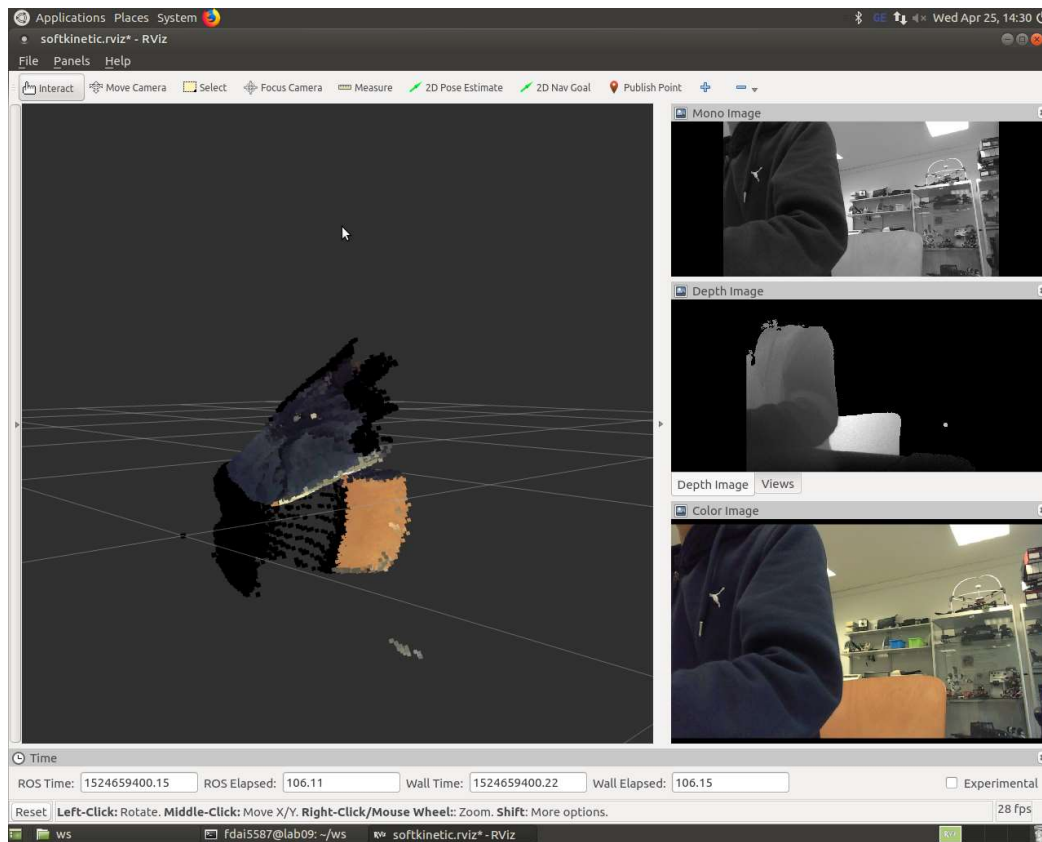


Figure 18: Depth Camera User Interface

```

Applications Places System
fdai5587@lab09: ~
File Edit View Search Terminal Tabs Help
/home/fdai5587/ws/src/fullpkg/launch/campf.launch http://localhost:11311
fdai5587@lab09:~$ rostopic list
/output
/rosout
/rosout_agg
/softkinetic_camera/depth/camera_info
/softkinetic_camera/depth/image_raw
/softkinetic_camera/depth/image_raw/compressed
/softkinetic_camera/depth/image_raw/compressed/parameter_descriptions
/softkinetic_camera/depth/image_raw/compressed/parameter_updates
/softkinetic_camera/depth/points
/softkinetic_camera/parameter_descriptions
/softkinetic_camera/parameter_updates
/softkinetic_camera/rgb/camera_info
/softkinetic_camera/rgb/image_color
/softkinetic_camera/rgb/image_color/compressed
/softkinetic_camera/rgb/image_color/compressed/parameter_descriptions
/softkinetic_camera/rgb/image_color/compressed/parameter_updates
/softkinetic_camera/rgb/image_mono
/softkinetic_camera/rgb/image_mono/compressed
/softkinetic_camera/rgb/image_mono/compressed/parameter_descriptions
/softkinetic_camera/rgb/image_mono/compressed/parameter_updates
/tf
fdai5587@lab09:~$ rosnodet list
/rosout
/softkinect_tf
/softkinetic_camera
/tc
fdai5587@lab09:~$

```

Figure 19: Rostopics and Rosnodes published

5.3 System Development

The figure 20 shows that the whole system can be branched out in 4 different sub-parts, Camera Node, PCD to Histogram, LSTM Network and Teleoperation Node, which finally results in the motion of the robot in the desired manner.



Figure 20: Data flow as a whole system

1. Camera Node

As described in Section 5.2, various rostopics are published under the rosnode `‘/softkinetic_camera’` when the `.launch` file is launched from the said package. The rostopic `‘/softkinetic_camera/depth/points’` publishes the Point Cloud Data (PCD) of a given frame, which is then passed on to the next phase.

2. PCD to Histogram

The PCD from the previous stage is then converted to the said 625 binned histogram in this phase. The PCD is subscribed by a code written in C++. This code is responsible for computing the Point Feature Histogram of the given PCD. This information is then published as a rostopic `‘/hist’`.

3. LSTM Network

The 625 binned histogram of a single frame is then subscribed by the LSTM Network written in Python. The network gives the class of the input and the prediction. If the output prediction is above the described threshold, it is published as the control gesture as a rostopic `‘/cgest’`. After each classification, the internal states and the activities of the LSTM network are reset.

4. Teleoperation Node

Finally, the accepted gesture class is subscribed by the teleoperation node resulting in the corresponding motion.

A single `.launch` file is written to implement all the functions in a single go. The following nodes are launched at the same instance, TurtleBot minimal software and the four nodes as described above. The robot is first initialized in a known map. A gesture class is recognized by the LSTM and passed on to the teleoperation node leading to the consecutive motion.

CHAPTER 6: CONCLUSION AND FUTURE PLAN OF WORK

The core basics of the main research problem are dealt with in the initial phases. After discussing about the objectives and the scope of this research and the motivation, we explore the related work being done in this field by fellow researchers. This report has been bifurcated into two main parts, one being the robotic platform and ROS, the other being the hand gesture recognition and machine learning.

We discuss the robotic platform, the TurtleBot 2, and the reason for choosing this as the research platform. We dwell more in the Robot Operating System or ROS, the advantages, usage and the basic setup of it for this research. Further on, we generate a depth map of an unknown environment and use that map to enable the robot to navigate autonomously when given the starting position and the goal position.

In the next section, the implementation of TensorFlow in the field of RNNs and the basics of RNNs is studied. Finally, the dataset acquired is been tested on an LSTM network. Our main finding is that LSTM can be tremendously robust if this robustness is somehow reflected in the training data. In case the variations are difficult or impossible to model, like the variations in performing gestures between different persons, this technique is no longer applicable, and the learning algorithms must cope with this variability which requires a huge amount of data.

The final section deals with the development of the real time system. The real time system developed on the ROS platform, translates a given hand gesture into the desired motion by the TurtleBot.

Next steps on the machine learning aspect will be to include benchmarking the real-time system and constructing a larger database, with more gesture types, for test purposes, and also to unambiguously determine performance.

REFERENCES

- [1] Xu, D., 2006, August. A neural network approach for hand gesture recognition in virtual reality driving training system of SPG. In Pattern Recognition, 2006. ICPR 2006. 18th International Conference on (Vol. 3, pp. 519-522). IEEE.
- [2] Buchmann, V., Violich, S., Billingham, M. and Cockburn, A., 2004, June. FingARtips: gesture based direct manipulation in Augmented Reality. In Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia (pp. 212-221). ACM.
- [3] Sarkar, A., Gepperth, A., Handmann, U., & Kopinski, T. "Dynamic Hand Gesture Recognition for Mobile Systems Using Deep LSTM". In 2017 International Conference on Intelligent Human Computer Interaction (pp. 19-31). Springer, Cham.
- [4] Pasarica, A., Miron, C., Arotaritei, D., Andrusac, G., Costin, H., & Rotariu, C. "Remote control of a robotic platform based on hand gesture recognition". In E-Health and Bioengineering Conference (EHB), 2017 (pp. 643-646). IEEE.
- [5] Gao, X., Shi, L., & Wang, Q. "The design of robotic wheelchair control system based on hand gesture control for the disabled". In Robotics and Automation Sciences (ICRAS), 2017 International Conference on (pp. 30-34). IEEE.
- [6] Yu, Y., Wang, X., Zhong, Z., & Zhang, Y. "ROS-based UAV control using hand gesture recognition". In Control and Decision Conference (CCDC), 2017 29th Chinese (pp. 6795-6799). IEEE.
- [7] O'Kane, J.M., 2014. A gentle introduction to ROS.
- [8] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A.Y., 2009, May. ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).
- [9] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., 2016, November. TensorFlow: A System for Large-Scale Machine Learning. In OSDI (Vol. 16, pp. 265-283).
- [10] Mikolov, T., Karafiát, M., Burget, L., Černocký, J. and Khudanpur, S., 2010. Recurrent neural network based language model. In Eleventh Annual Conference of the International Speech Communication Association.
- [11] Zaremba, W., Sutskever, I. and Vinyals, O., 2014. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329.
- [12] Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.

- [13] Thrun, S., Fox, D., Burgard, W. and Dellaert, F., 2001. Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, 128(1-2), pp.99-141.