Deep Neural Network for Recognition of State and Action Trajectories from RGB-D data

by

Nihal Solomon Soans

(Under the direction of Prashant Doshi)

Abstract

In the world of robotics there are multitude of algorithms that enable a robot to operate; however, there is need for an optimal solution for enabling said robot to detect an object in its surroundings. We approached this problem using a network designed using CNN's and Conv-LSTM. This helps the network refer to previous images as a sequence to classify what motion the observed object is performing in a 3-dimensional space in conjunction to the objects state in that space. The robot is able to run the network using a small amount of computational power without sacrificing accuracy and speed. This network has the ability to adapt to new and different environments.

Index words: deep-neural-networks, cnn, convlstm, image-processing, robotics, computer-vision, state-action, iRL

Deep Neural Network for Recognition of State and Action Trajectories from RGB-D data

by

Nihal Solomon Soans

B.E, St Jospeh's Engineering College, 2013

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2018

Deep Neural Network for Recognition of State and Action Trajectories from RGB-D data

by

Nihal Solomon Soans

Major Professor: Prashant Doshi

Committee: Yi Hong
Ramviyas Nattanmai Parasuraman

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
December 2018

Acknowledgments

I Sincerely express my wholehearted gratitude to my mentor Dr. Prashant Doshi, Director, Thinc Lab, University of Georgia for his sustained interest, cooperation and guidance; and also for his inspiration, good will and unstinted support through all hardships.

I owe my deepest gratitude to Dr. Yi Hong who made this thesis many folds easier with her continuous guidance and support in the field of Deep Learning.

I would like to thank Dr. Ramviyas Nattanmai Parasuraman who has made available his support in a number of ways and for his time and effort to evaluate my work.

I express my gratitude and love to my parents and sister for their constant encouragement and support to all my academic adventures.

I am indebted to many of my friends and lab-mates for their comments, debates and deliberations which helped me in many ways on this work.

Table of Contents

Page

Acknowledgments iv

List of Figures vii

List of Tables x

Chapter

1 Introduction 1

1.1 Observations in Robots 1

1.2 Object Detection in Images 2

1.3 Motivation 2

1.4 Contribution 3

1.5 Problem Domains that are Used in this Work 4

2 Related Work 9

2.1 Methods using Geometry of Objects 9

2.2 Approaches using 3D-CNN 10

2.3 Methods Based on Capturing Pose, Intent or some Feature of the Object 11

2.4 Methods based on Multimodal Data 12

2.5 Methods Based on dividing the video into clips 13

2.6 Methods Based on Image Segmentation or Object Recognition 15

2.7 Summary 16

3 Deep Neural Network for Recognition of State and Action Trajectories . . . 17
3.1 Input to the Network . . . 17
3.2 Structure of training data . . . 18
3.3 Architecture of SA-Net . . . 22
3.4 Training . . . 30
3.5 Running the Experiment . . . 31
4 Results . . . 33
4.1 SA-Net Results . . . 33
4.2 Physical Experiment . . . 34
4.3 Robustness tests . . . 34
4.4 Ablation Study . . . 37
4.5 Resource Usage . . . 40
5 Conclusion and Future Work . . . 42
5.1 Conclusion . . . 42
5.2 Future work . . . 42
Bibliography . . . 44

List of Figures

1.1 Example of Object Detection working on our experiment 2

1.2 X_t and Y_t coordinates mapped on to the architecture map of the building . . 4

1.3 The turtle-bots mapped to the architectural map. Blue circle is the attacker. Pink and Red circles are the Patroller 6

1.4 Attacker Patroller Experiment, (a)is how the experiment took place, (b) is the turtlebot Pink, (c) is the turtlebot Red, (d) is the Attacker 7

1.5 State space of X_p and Y_p dimension of PhantomX arm (a) represents the X_p,Y_p Co-ordinates on the phantomX in the experiment and (b) Blueprint of the State space for the manipulator arm, shown here is the X_p and Y_p 7

1.6 State space of X_p and Z_p dimension of PhantomX arm (a) represents the the Z_p Co-ordinate on the phantomX in the experiment (b) Blueprint of the State space for the manipulator arm, shown here is the X_p and Z_p 8

1.7 (a)PhanthomX Manipulator Arm (b) Shows how the Manipulator Arm experiment was conducted 8

2.1 Structured Segmented Network re-drawn from[1] 15

3.1 Data-set Image Sample. (a) is the frame received by the network from the RGB frame.(b) is the frame from the depth sensor converted to a head map red or hotter pixel represents far away objects and blue or cool represents closer pixels this frame is shown using this method for easier human viewing 17

3.2 Forward Motion of the observed object in this case the patroller, We use direction the Kinect is facing and if it moves in this direction we call it forward motion 19

3.3 Left Motion of the turtle bot with respect to the observed object. Left motion is defined as the anti-clockwise rotation of the turtlebot 20
3.4 Right Motion of the turtle bot with respect to the observed object. Right motion is defined as the clockwise rotation of the turtlebot 20
3.5 Forward Motion of the Phantom Manipulator arm. This motion is defined as the movement of the end effector away from the base 21
3.6 Left Motion of the Phantom Manipulator arm. This motion is defined as the rotation of the arm around its z axis on the base in a anti-clockwise motion. 21
3.7 Right Motion of the Phantom Manipulator arm. This motion is defined as the rotation of the arm around its z axis on the base in a anti-clockwise motion . 21
3.8 Backward Motion of the Phantom Manipulator arm. This motion is defined as the movement of the end effector towards the base 21
3.9 Upward Motion of the Phantom Manipulator arm.This motion is defined as the movement of the end effector away from the ground upwards 22
3.10 Downward Motion of the Phantom Manipulator arm.This motion is defined as the movement of the end effector towards the ground downwards 22
3.11 High level Network design showing the two phases and the data that is transmitted to and from those modules. 23
3.12 Cropping Factor Graph. Axis X represents X from Equation 3.1 and Axis Y represents the crop rate or C_r 24
3.13 SA-Net Network 25
3.14 Masking Process, (a) shows the image captured by the sensor for the turtlebot experiment, (b) is the image stored in memory when there is no object found in this case a turtlebot, (c) and (d) are masked images produced by this mask 26
3.15 Detailed SA-Net Bottom Network layers 27
3.16 Detailed SA-Net Top Network layers 29

3.17 Inputs into the Network layers. The image on the left is the input to the top layer which recognizes the Coordinates of the object this frame is of size 640 by 480. The set of three frames to the right are the cropped frames of size 100 by 150. These frames will be the input to the bottom layer 30
3.18 ROS Nodes Architecture when Deployed on Robot 32
4.1 The figures below are images of the experiment being conducted for the noise test. We have broken the noise test into three categories. The random placement of similar boxes around the experiment area (a) shows two of the five test cases for this type of test.Same color external objects, (b) shows two of the five images of the experiment being conducted here we use a person with the same color jacket as the box on top of the turtlebot, this person would be walking up and down the experiment area.Low light test (c) shows two of the five experiments being conducted here we dimmed the light by 50% 36
4.2 The two images are a sample of the occlusion test being performed for the turtlebots. We cover one half of the turtlebot with a piece of cloth 37
4.3 Two frames of *theta* in the turtlebot experiment.(a) is when the turtlebot is facing north and (b) is when turtlebot is facing south 39

List of Tables

2.1 Our dataset tested on centroid detection. Accuracy measured in percentage . 10

2.2 Our dataset tested on Combining Motion and Appearance. Accuracy measured in percentage 11

2.3 Our dataset tested on Multi-Task Learning. Accuracy measured in percentage 14

3.1 Bottom Stream Layer Parameters. Each Row is labeled from layers shown in Figure 3.15 28

3.2 Top Stream Layer Parameter. Each Row is labeled from layers shown in Figure 3.16 29

4.1 Number of samples in the dataset after preprocessing 33

4.2 SA-Net results per fold for turtle-bot dataset on testing set. These are the accuracy results of state and action as percentage from non physical experiments 33

4.3 SA-Net results per fold for PhantomX arm dataset on testing set. These are the accuracy results of state and action as percentage from non physical experiments. 34

4.4 SA-Net accuracy in physical experiments in percentage for turtlebot and manipulator arm under normal conditions. Since turtlebots don't have a motion with respect to Z the accuracy is left as blank 34

4.5 SA-Net noise test results for turtle-bot experiment in percentage. Average of 15 experiments 5 for colored shirt, 5 for box and 5 with dimmed light 35

4.6 SA-Net occlusion test results for turtle-bot experiment in percentage. The results are an average of 10 experiments.The first row shows how SA-Net works on normal conditions. Second row is how SA-net performed on the occlusion tests and third row shows how the centroid method compares to the same test. We can see that SA-Net doesn't show complete failure in occlusion, where as centroid method does the not do very well in State detection. . . . 35
4.7 Ablation Study on Relative X and Y on turtlebot expriment 37
4.8 Involvement of Time Series data on θ 38
4.9 Involvement of depth as multimodal 39
4.10 Ablation study for Object Detection 40
4.11 Time to train all values in HH:MM 40
4.12 SA-Net memory usage 41
4.13 Performance with Respect to Prediction Time for two objects 41

Chapter 1

Introduction

Human Beings have been interested in Artificial Intelligence for a long time. Artificial Intelligence is the state of machine being able to do certain tasks that a human being can do intellectually. We have come a long way in the field of artificial intelligence still a long way, for that reason we take AI(Artificial Intelligence) research one step at a time.

We see AI almost every where in our lives for example in internet searches or recommendations systems when we indulge ourselves in online shopping so forth, but when its combined with robotics the picture gets more blurred as there is huge amount of data received in real time this can also include noise. Robots are machines which have a sophisticated set of sensors, motors etc which try to emulate or mimic human beings. Robotics is the field of study of these robots. When robots perform tasks by observing its surroundings and then acting according to the observation. Multitude of goals can be given to the robot making the life for us humans easier. In the world of robotics there are multitude of algorithms that enable a robot to operate but there is a need for an optimal solution for enabling this robot to detect an object around its surroundings, where the robot knows where that objects location is currently on a map also if that object is in motion then the robot should also be able to get the objects action.

1.1 Observations in Robots

Observations are a key feature in any robotics application using SONAR, RGB Camera, LASERs or even sound to perceive this observations. We define observation as the information perceived by the robot via some sensor. In our experiment we are using Microsoft Kinect

v1 which includes a RGB sensor and a depth sensor which uses infrared LASER matrix projector to help ascertain depth. These observation are processed by an algorithm to help the robot understand its surrounding and act on those stimulus provided by the surroundings.

1.2 Object Detection in Images

Object Detection is a image processing technique used in computer vision, this method involves detecting multiple objects in a single image. In contrast object classification is the method of detecting a single object in a image. There are many algorithms out there, we will be concentrating on deep learning approaches such as YOLO[2] or Faster-RCNN[3]. We plan on using object detection to detect the object which will be the input to our network.

Figure 1.1: Example of Object Detection working on our experiment

1.3 Motivation

Convolutional Neural Networks or CNN have been there in the world of Supervised image recognition for some time and it has helped classify images accurately; namely VGG[4], googlnet[5], resnet[6] to name a few. Since many iRL or inverse reinforcement algorithms such as ones described in [7] rely mainly on data from observations, which consists mainly of trajectories based on state and action. Having a noisy observation will cause the algorithm to either fail or have erroneous results. There is a need to have a system or a model which is capable of recognizing this state and action with good accuracy. This model should be able to run in any environment with our taking up huge amount of resources.

1.4 Contribution

Our goal is to detect and classify both state and action of an object that the viewer is observing. We describe action detection as classifying an action such as forward, backward movements in real time. Since the robots will be running on systems having limited resources the model should be fast to get the right classification together with low processor and memory utilization. Since action is attributed to movement or motion of an object our data is temporal its almost impossible to detect action from a single image or frame, our idea is to merge the concepts of image segmentation together with Convolutional Neural Networks and LSTMs to get the model to recognize both state and action of objects. We plan on using object recognition algorithms to aid in the detection of action and state of that object. We also plan on using a multi-modal data set which combines both RGB Sensor information along with depth sensor information this data will be introduced in detail later.
The main idea for the action detection is to use a cropped frame which includes the object from time t and backtrack the movement inside that bounding box for time $t-1$ and $t-2$. This is the same way human being also perceive motion when observing the action of an object. To do this we plan on using LSTM or Long Short-Term Memory [8] to help and back track the features, this helps the model to refer to previous images in a sequence to detect the action of the observed object being performed in a 3-dimensional space.
Key contributions of the thesis summarized to

- We created a two stream deep learning model which can detect both state and action of observed object.Action can be detected by backtracking on features and state by position of object on a map.

- The network runs within the time constraints and within resource allocation such as processor and memory

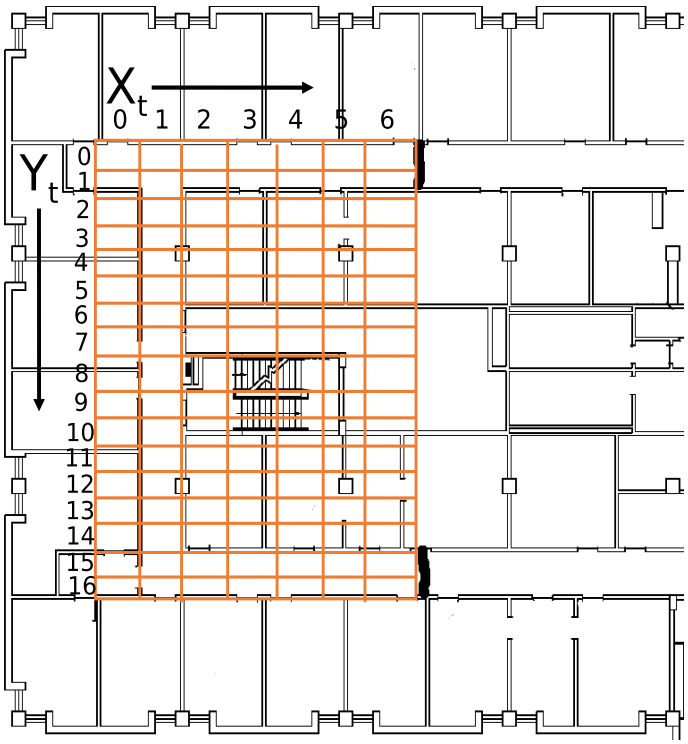
- The proposed network can be implemented in completely different environments for example turtle-bot experiment and pick and place manipulator arm experiment which have different set of features.

1.5 Problem Domains that are Used in this Work

Our proposed deep learning model is tested on two completely different environments. One is the attacker and patroller experiment proposed in [7] and second is pick and place experiment using phantomX pincher arm.

1.5.1 Patroller-Attacker Experiment

Figure 1.2: X_t and Y_t coordinates mapped on to the architecture map of the building



In this experiment there are three turtle-bots in which one is an attacker bot and two are patrol-bots. The partoller robot's job is to keep patrolling around the corridor and the attacker bot is to infiltrate the area guarded by the partoller without being detected. The

State and Action of each of the turtle-bot is to be predicted in real time. The State space for this experiment is defined as X_t, Y_t, θ_t, where X_t and Y_t is the x and y co-ordinate on the map shown in figure 1.2. θ is the direction or heading of the robot $\theta_t = North, West, South, East$ X_t is defined as

$$0 \leq X_t \leq 7 \tag{1.1}$$

Y_t is defined as

$$0 \leq Y_t \leq 17 \tag{1.2}$$

θ_t is set of all directions the gripper faces $\theta_t = North, South, East, West$ and can be mathematically represented as

$$0 \leq \theta_t \leq 3 \tag{1.3}$$

The Action Space A_t is the set of all actions that can be performed by the turtle-bot which contains $A_t = Forward, Stop, Right, Left$ and can be mathematically represented as

$$0 \leq A_t \leq 3 \tag{1.4}$$

1.5.2 Pick and Place Experiment

In this experiment we use a single PhantomX manipulator arm performing a pick and place of objects while a camera capable of RGB sensor and Depth sensor data is place 45°with respect to the ground to observe the arm. The State and Action of the arm is to be predicted in real-time. Figure 1.7(b) we can see two objects one a chocolate and one rubber cap. The Kinect v1 is mounted on a tripod 5 feet of the ground facing downwards.

The State space for pick and place experiment is defined as X_p, Y_p, Z_p, θ_p
where X_p is defined as

$$0 \leq X_p \leq 4 \tag{1.5}$$

Figure 1.3: The turtle-bots mapped to the architectural map. Blue circle is the attacker. Pink and Red circles are the Patroller

Y_p is defined as

$$0 \leq Y_p \leq 8 \tag{1.6}$$

Z_p is defined as

$$0 \leq Z_p \leq 6 \tag{1.7}$$

θ_p is set of all directions the gripper faces $\theta_p = North, South, East, West, Up, Down$ and can represented mathematically as

$$0 \leq \theta_p \leq 5 \tag{1.8}$$

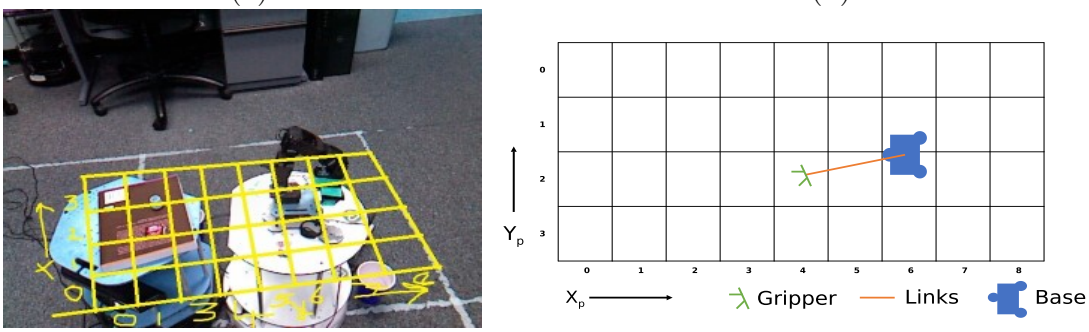
The Action Space A_p is the set of all actions that can be performed by the phantomX manipulator arm which contains $A_p = Forward, Stop, Up, Down, Right, Back$ and can be

Figure 1.4: Attacker Patroller Experiment, (a)is how the experiment took place, (b) is the turtlebot Pink, (c) is the turtlebot Red, (d) is the Attacker

(a) (b) (c) (d)

Figure 1.5: State space of X_p and Y_p dimension of PhantomX arm (a) represents the X_p,Y_p Co-ordinates on the phantomX in the experiment and (b) Blueprint of the State space for the manipulator arm, shown here is the X_p and Y_p

(a) (b)



mathematically represented as

$$0 \leq A_t \leq 5 \tag{1.9}$$

Figure 1.6: State space of X_p and Z_p dimension of PhantomX arm (a) represents the the Z_p Co-ordinate on the phantomX in the experiment (b) Blueprint of the State space for the manipulator arm, shown here is the X_p and Z_p

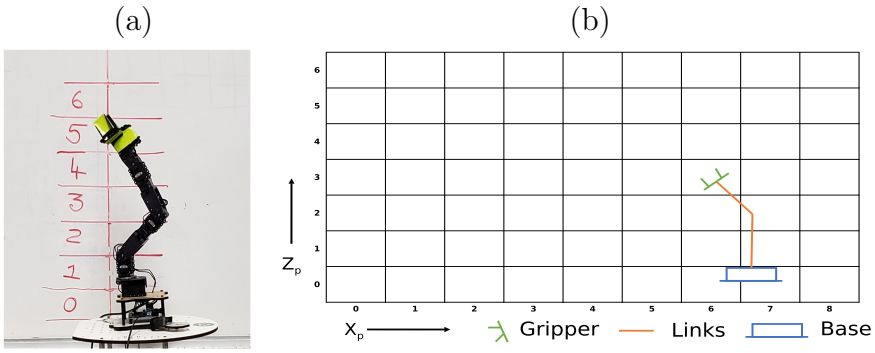
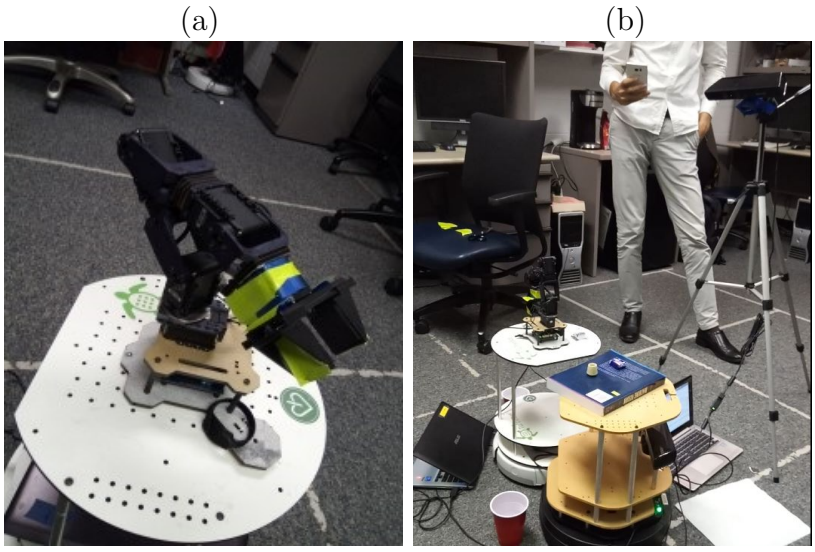


Figure 1.7: (a)PhanthomX Manipulator Arm (b) Shows how the Manipulator Arm experiment was conducted



The State space for the manipulator bot is in three dimensions and Figure 1.5(a) and Figure 1.6(a) can be quite confusing. Figure 1.5(b) and Figure 1.6(b) shows the blueprint or schema of show the state space is when divided into two 2D grids. Figure 1.5 shows the X_p and Y_p and Figure 1.6 shows Z_p with respect to X_p

Chapter 2

Related Work

This chapter presents a review on different state and action recognition algorithms using video or sequence of images and is organized as follows: Subsection 2.1 discusses about methods that use mathematical geometry to help to determine the state and action of the object, 2.2 discusses 3D CNN based approaches, 2.3 talks about recognizing the action of the object using some features or intent, 2.4 discusses about different approaches using multimodal data, 2.5 discusses motion detection by diving one video into multiple clips.

2.1 Methods using Geometry of Objects

The section discusses about methods that use geometrical changes in a 2D space to help to determine the state and action of the object.

2.1.1 Multi-Robot Inverse Reinforcement Learning under Occlusion with Interactions

In this paper[7] the authors use a geometry based approach to get the state and action of the TurtleBot. The author uses a colored box on top of the TurtleBot and uses this box to calculate the centroid of this box by detecting the color as a blob, the state is detected as the distance between the observer bot and observed bot using the depth data from the Kinect sensor, θ or direction is detected by using three frames for which centroid has been calculated and then a trajectory or a line is drawn based, Depending on that line θ is recognized this is the same for action, as in the case of action the area of the square is also used. This method works very well in our experiment, but introduction of noise such as another object with the

same color or if the object is occluded by covering half the object then the algorithm fails to work.

Method	State	Action
Centroid Detection±SD	94.41±0.00185	78.45±0.2486
SA-Net±SD	97.20±0.3864	96.16±0.6782

Table 2.1: Our dataset tested on centroid detection. Accuracy measured in percentage

2.2 Approaches using 3D-CNN

The section discusses the use of a 3D CNN to detect action of an object but having a three dimensional kernel.

2.2.1 3D Convolutional Neural Networks for Human Action Recognition

This paper[9] the author proposes to use three dimensional convolutional neural network or 3D-CNN to solve the human action classification problem using temporal data. The authors also perform sub-sampling from adjacent frames which is obtained by merging data from the channels also they used regularization using auxiliary outputs boosting the overall performance of the network.

2.2.2 Temporal Activity Detection in Untrimmed Videos using Recurrent Neural Networks

The paper[10] creates a simple model using three dimensional convolutional neural network for action detection in untrimmed videos , the author uses a series of 3D-CNN and LSTM layers to detect actions for 16 frames at a time then using equation 2.1 [10] the best possible action is then calculated over the whole video clip. Here x is the probability of the action x for the sample i, k is the number of samples.

$$p_i(x) = \frac{1}{2k} \sum_{j=i-k}^{i+k} p_i(x) \tag{2.1}$$

Both the above 3D-CNN methods although are good for using where features are temporal in nature, an addition of an extra dimension from the 3D-CNN parameters will cost computational, memory and can exceed our processing times.

2.3 Methods Based on Capturing Pose, Intent or some Feature of the Object

This section discusses different approaches or algorithms that use either the intent, pose or a feature of object. For example like if the pose of a person is bending down then he could be picking up something.

2.3.1 Action Recognition and Detection by Combining Motion and Appearance Features

This paper[11] proposed a technique using fused features of improved Dense Trajectory or iDT which is known for being able to extract motion information and CNN features to detect the object and its pose. This method does not produce the required accuracy results on our experiments. More over our experiment especially the TurtleBot experiment does not have a pose, it does have orientation.

Method	State	Action
Combining Motion and Appearance±SD	NA	62.45±0.1548
SA-Net±SD	97.20±0.3864	96.16±0.6782

Table 2.2: Our dataset tested on Combining Motion and Appearance. Accuracy measured in percentage

2.3.2 Action Detection by Implicit Intentional Motion Clustering

This paper[12] proposes a method of detecting human actions by relating actions to intent of the human, for example the human would be moving his hand towards a table with a glass of water as he starts to move, we can see that his intent would be to grab that cup of

water or the table. By being able to detect this "intent" the author creates a model which first creates a space-time trajectory graph to capture the intent, then partition it to different clusters which is later used as proposals. This paper models the object in this case humans to have intent, but our experiment works on objects which may or may not have an intent. Modeling this scenario sometimes way not yield the required results. Also, the features of our objects are quite much similar.

2.4 Methods based on Multimodal Data

We discuss the algorithms or models that use depth data as the extra information for recognizing the action of the object.

2.4.1 Multimodal Deep Learning for Robust RGB-D Object Recognition

This paper[13] presents a different approach to detect action by using multimodal data in two streams one for RGB and one for depth, the authors are aiming to get object detection by using two streams in network the RGB stream layers are loaded with pre-trained weights from image-net data-set and the depth stream is then trained on the depth data from the depth data-set. Finally both are trained simultaneously using the Washington RGB-D Object dataset, [14]. During preprocessing of the images the authors converted the the depth images into a heat map based on the distance and then converted it to HMM format used in [15] this helped the network overcome the noisy sensor information especially in the depth sensor data.The results for this type of model for the dataset was good, since the model was trained on data from multiple side two out of three sides, the problem that we have described may not have data from multiple sides and that is where this model fails , we need a model which is robust enough to eliminate sensor noise while not going through too much of preprocessing.

2.5 Methods Based on dividing the video into clips

The section talks about the algorithms that uses divide and conquer strategy that we all know by dividing a video or a section of the video into many clips of n frames to help and detect the action or state of the object.

2.5.1 Deep Networks for Video Classification

The paper[16] discusses a deep learning model using LSTM and Feature pooling to be able to classify videos, the author first gets the optical flow of the frames from the videos and passes it to the model along with the raw images which is passed through a deep LSTM[8] layer. If there are n frames, then there would be n-time steps in the LSTM layer. The CNN layer is a pre-trained weight from google-net[17] and will be replicated for each frame hence there would be no need to train the CNN with n time steps. This allows the model to run in a single shot format. Therefore, aiding the network classify the video. This type of approach can be used to detect actions, but it fails the action detection property where we must detect it in real time as we have a live stream and detection must be done in real time.

2.5.2 Human activity prediction: Early recognition of ongoing activities from streaming videos

This paper[18] uses time as the third dimension to concatenate the frame along the time axis to get XYT axis, where X and Y are width and height of the video frame and T is the time frame. This paper then formulates an integral bag of actions which is then used to detect an incomplete action. This integral bag-of words is an approach that creates an integral histogram to represent the actions by computing the likelihood of $P(O|A_p, d)$ where O is the observation in this case the data with XYT axis with t is the length of axis T. A_p is the action being performed and d being the possible progress level of the activity. This model keeps appending the frames to this can consume a lot of memory which in our case is

limited. There is also the possibility of having many changing actions in a very short period. This model does not help when the videos are in this format.

2.5.3 Efficient Action Detection in Untrimmed Videos via Multi-Task Learning

This paper[10] proposes a light weight deep neural network model that uses 3D CNN to be able to recognize temporal features. Which is then passes to a Fully connected layer. The model also requires fewer iterations for training the network and works well in multiple data-sets. Introduction of this model along with state fails to work as it suddenly becomes heavier and results in over-fitting the forward action in our TurtleBot experiment. When using multimodal data, the network gets confused with the depth data a assumes that the depth data is temporal information.

Method	State	Action
Multi-Task Learning±SD	NA	55.73±0.5126
SA-Net±SD	97.20±0.3864	96.16±0.6782

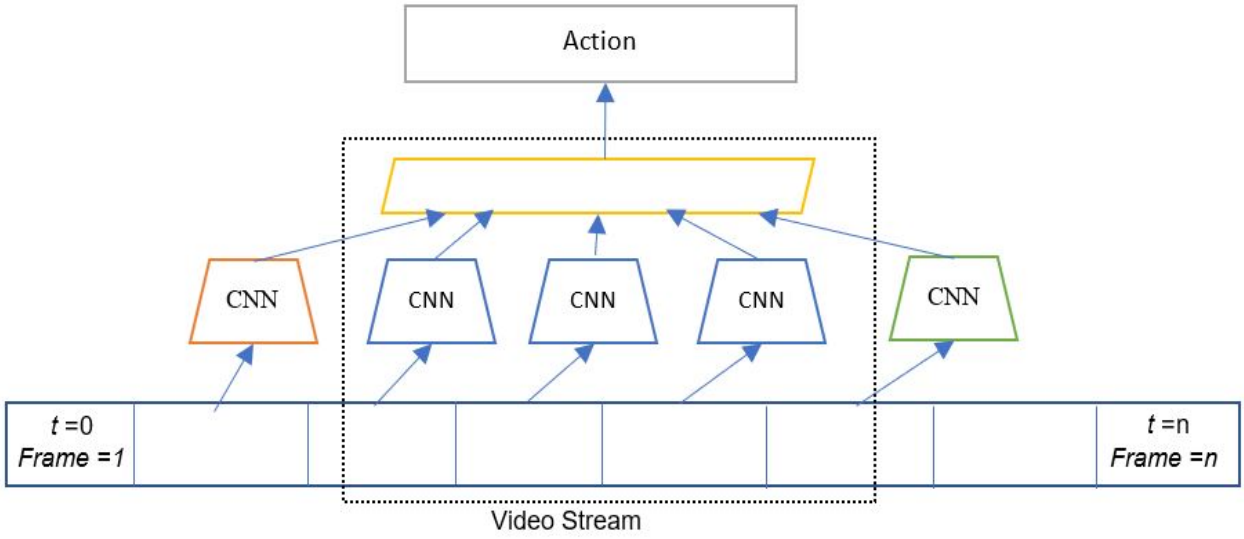
Table 2.3: Our dataset tested on Multi-Task Learning. Accuracy measured in percentage

2.5.4 Temporal Action Detection with Structured Segment Networks

This paper[1] proposed a structured segmented network for detection of motion in videos, the author proposed this model by dividing the video into multiple frames pass it through a Convolutional Neural network by segmenting the video.

This network takes 4 frames as one clip, Figure 2.1 shows a redrawn and simplified version of the network proposed in [1]. This set of clips shares its last from to the next clip of frames shown in orange CNN and the first of the next clip of frames. This is shown in the green CNN in Figure 2.1. Then through multiple detection the network can then classify what action the object or person is performing. This method is widely used for network classification and has good results overall, our problem cannot supply the frame for the next clip for features

Figure 2.1: Structured Segmented Network re-drawn from[1]



which is represented in green CNN in figure 2.1. More over the action in our experiments keeps changing based on what the robot is doing it can also show changing action for every clip. This method will not be able to classify that as it can only detection one motion or action in each video clip.

2.6 Methods Based on Image Segmentation or Object Recognition

This section discusses about algorithms that use of combining both the image segmentation or object recognition to recognize detection.

2.6.1 Tube Convolutional Neural Network for Action Detection in Videos

In this paper[19] the authors have created a model having a tube like structure for video classification hence the name Tube Convolutional Neural Network. For this the authors have proposed a method similar to Faster - RCNN [20] along with time as the added dimension

has they try to detect action by dividing the video into multiple clips of 8 frames each this is then passed through a 3D-Convolutional Network or 3D-CNN network to get the spatio-temporal features the last layer will have a tensor of depth = 1 that is one frame, Region of Interest Pooling proposed in [3] using anchor boxes to segment action from the clips, these feature are then passed on to the next clip. This model works fine on systems with huge amount of computational resources although the results from this model are very good, the network does fails when the action is based on motion with respect to the object from the eyes of a observer also it is unable to detect state of the model.

2.6.2 Learning Rich Features from RGB-D Images for Object Detection and Segmentation

This paper[3] proposes a method for detection of objects using multimodal data in this case RGB-D to be able to perform image segmentation of objects. The author creates 2.5D bounding boxes and pixel masks for the dataset. This method can be altered to detect the state of the object using the bounding boxes and masks instead of detecting the object. This approach although good can take so much more time as we are adding another dimension, we also must consider the problem of computational resources.

2.7 Summary

There are many methods for action recognition either by latching on to features of the object or by diving the whole video into smaller clips, But there is only one method which can localize observed objected and do action recognition at the same time. The method which is currently present for detection of action and state has a locked environment and wont work in some other environment.

Chapter 3

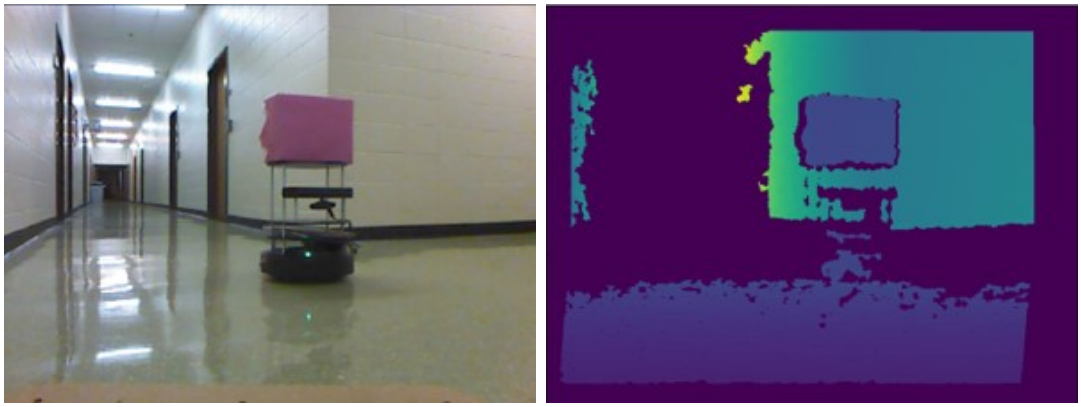
Deep Neural Network for Recognition of State and Action Trajectories

This section contains the architecture of the deep neural model, how its trained and executed on the robots also how the input to the network is stored and processed.

3.1 Input to the Network

Two main inputs for the network are taken from the depth and RGB stream of the Kinect v1 which is attached to the Turtle-bot. In the case of the manipulator arm experiment the Kinect v1 is mounted on a tripod 5 feet from the ground at an angle of -45°with respect to ground level.

Figure 3.1: Data-set Image Sample. (a) is the frame received by the network from the RGB frame.(b) is the frame from the depth sensor converted to a head map red or hotter pixel represents far away objects and blue or cool represents closer pixels this frame is shown using this method for easier human viewing



(a) (b)

3.1.1 RGB

The RGB stream from the Kinect v1 produces frames having pixel width of 640 and height 480 pixels with 3 channels, each channel representing Red, Green and Blue, these frames look like its taken from a normal camera as shown in figure 3.1(a) Each frame looks just like a image that we as humans can perceive. The total frame size of each is [640 , 480 ,3] and each pixel has values from 0 to 255 of type 8-bit integer.

3.1.2 Depth

The Depth stream are frames produced by Kinect's infra-red laser projector, The data from this sensor are a single channel frame of pixel width 640 and height 480 pixels. Each pixel corresponds to the depth at that pixel. For example if a value of a pixel at (200,300) is 700 it means at (200,300) on the frame there is an object at a distance of 700millimeters. These pixels have a range from 400 to 7500 and is of data type 16-bit integer. Figure 3.1(b) gives an example of how this depth frame looks like in human understandable format.

One sample contains three frames of four channel images of size (100, 150, 4) each which we also called as cropped frame and one frame of size (640, 480, 4) which we also call as full frame. The three frames are labeled as *t, t-1, t-2* and the four channels are Red, Blue, Green and Depth which is used for classifying action and θ. The full frame is used to classify State other than θ.

3.2 Structure of training data

In this experiment we are introducing a new data-set for action and state recognition. This dataset consists of two experiments one is the patroller attacker and the other is the manipulator arm experiment. Both these experiments have same frame size as discussed in Section 3.1 as we are using the same Kinect v1. The data is stored in a tree type folder structure with each folder representing the labels. The parent folder contains the Action and then the State. Based on this folder structure we create labels one-hot encoded in the form of numpy array.

Storing it as an numpy array helps train the network faster as it prevents bottle-necking from reading data directly from hard disk this can cause GPU starvation, increasing the time to train.

One of the main difficulties of this data set is action with respect to the observed object. Figure 3.2 shows three motions which are all classified as forward. Figure 3.3 show how the

Figure 3.2: Forward Motion of the observed object in this case the patroller, We use direction the Kinect is facing and if it moves in this direction we call it forward motion

robot is labeled as left, Left motion is defined as the motion of rotation along its z axis.

Figure 3.3: Left Motion of the turtle bot with respect to the observed object. Left motion is defined as the anti-clockwise rotation of the turtlebot

Figure 3.4: Right Motion of the turtle bot with respect to the observed object. Right motion is defined as the clockwise rotation of the turtlebot

The PhantomX Manipulator arm has many more motions when compared to the turtlebot experiment. Figures 3.5, 3.6, 3.7, 3.8, 3.9, 3.10 show frame by frame examples for the motions of the manipulator arm.

Figure 3.5: Forward Motion of the Phantom Manipulator arm. This motion is defined as the movement of the end effector away from the base

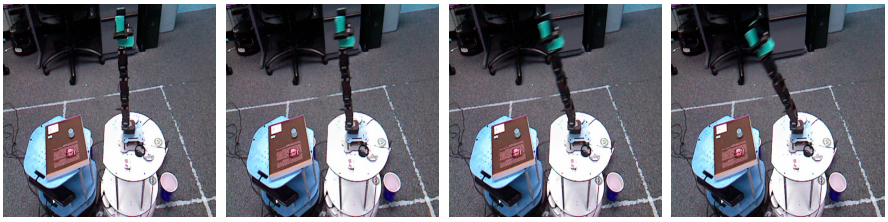


Figure 3.6: Left Motion of the Phantom Manipulator arm. This motion is defined as the rotation of the arm around its z axis on the base in a anti-clockwise motion.

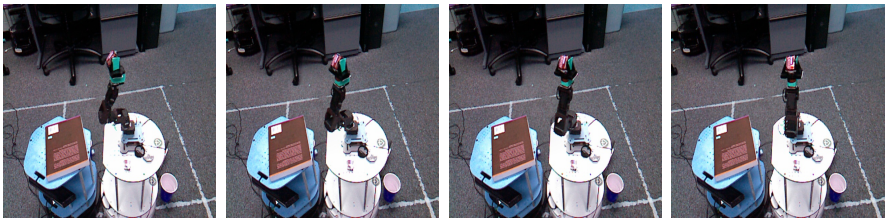


Figure 3.7: Right Motion of the Phantom Manipulator arm. This motion is defined as the rotation of the arm around its z axis on the base in a anti-clockwise motion

Figure 3.8: Backward Motion of the Phantom Manipulator arm. This motion is defined as the movement of the end effector towards the base

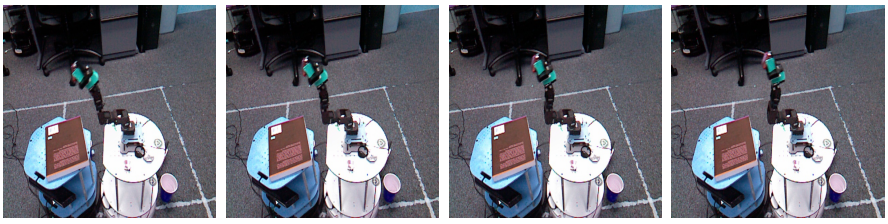


Figure 3.9: Upward Motion of the Phantom Manipulator arm.This motion is defined as the movement of the end effector away from the ground upwards

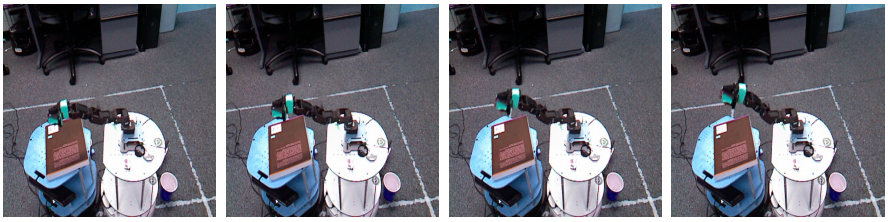
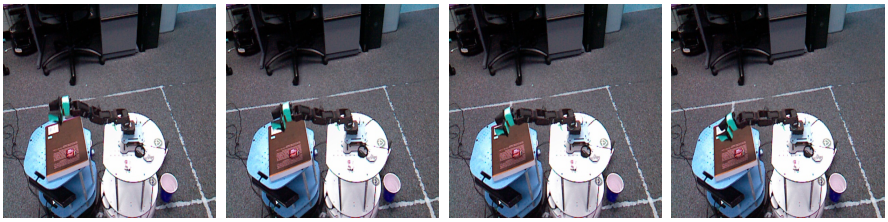


Figure 3.10: Downward Motion of the Phantom Manipulator arm.This motion is defined as the movement of the end effector towards the ground downwards



3.3 Architecture of SA-Net

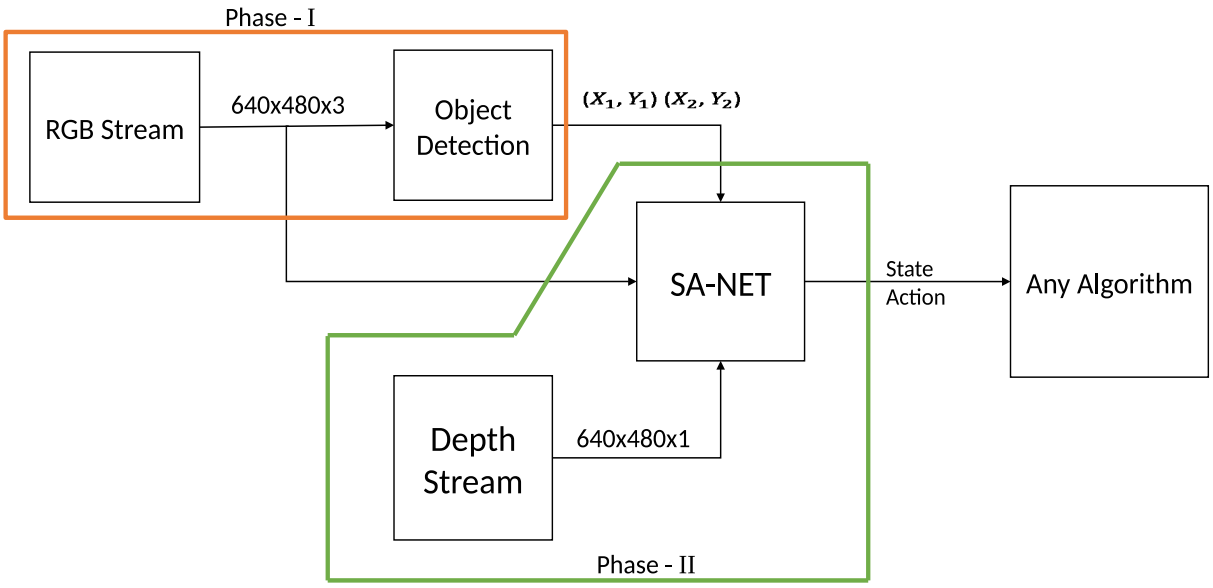
The whole network consist of a two phases:

1. Phase - I task is to detect and recognize the object

2. Phase - II task to recognize the state and action of this object

The reason the network has two phases is because one phase crops the object and the other recognizes the state and action. Our motivation for this is mainly how we human observe an object. When we look at an object and perceive its behavior, we tend to focus only on that object and for a moment ignore other objects. To simulate this we are using cropping and masking. Figure 3.17 shows the inputs to the Network for both top and bottom network stream.

Figure 3.11: High level Network design showing the two phases and the data that is transmitted to and from those modules.



3.3.1 Phase 1

This phase consists of the object detection phase using the RGB data the model which is trained to get the objects location multiple methods can be employed here such as Faster-RCN[21], YOLO[2] or any object detection tool. This phase shown in Figure 3.11 is the boxes labeled RGB stream and Object Detection. This is a very crucial part of our network. The Phase of the network deal with the object detection at time t using only the RGB data from the RGB sensor. The main task of this phase is to get the X and Y co-ordinates in the image which contains the object, for example the turtle-bot. This phase does not require the depth data as it might interfere with the object detection tools. We have tested three types of models for this phase. Namely Faster-RCNN, YOLO and Object Tracking algorithms.

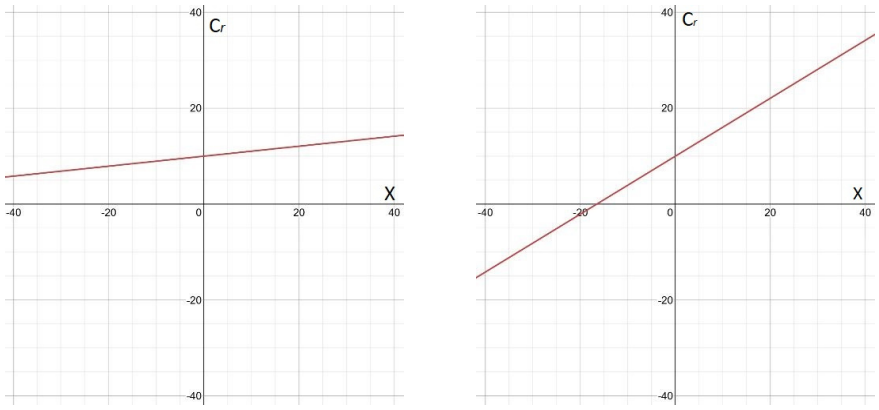
When the [x_1, y_1, x_2, y_2] is received from the object detection we need to crop the image from frame *t, t-1, t-2* which will be passed into the top network stream of Phase - II explained in Section 3.3.2. Since there might be some amount error in object detection need to have

a little bit of buffer when cropping. This buffered cropping will not be uniform as when the object is far we will need to have lower amount of crop compared to when the object is close to the turtlebot. To this we have employed a crop linear equation 3.1

$$C_r = (C_f * X) + C_{min} \tag{3.1}$$

Where C_r is the crop rate this will tell us how much of cropping buffer the network will need. C_{min} is the minimum amount of cropping and is a value defined by us we use 10 pixels. C_f is the cropping factor this is how aggressively you want to crop to the maximum crop. x is the the difference in length from x_1 to x_2, i.e $X = x_1 - x_2$. Figure 3.12 shows how C_f affects the cropping. Figure 3.12(a) is when C_f is 0.1 and Figure 3.12(b) is when C_f is 0.6.

Figure 3.12: Cropping Factor Graph. Axis X represents X from Equation 3.1 and Axis Y represents the crop rate or C_r

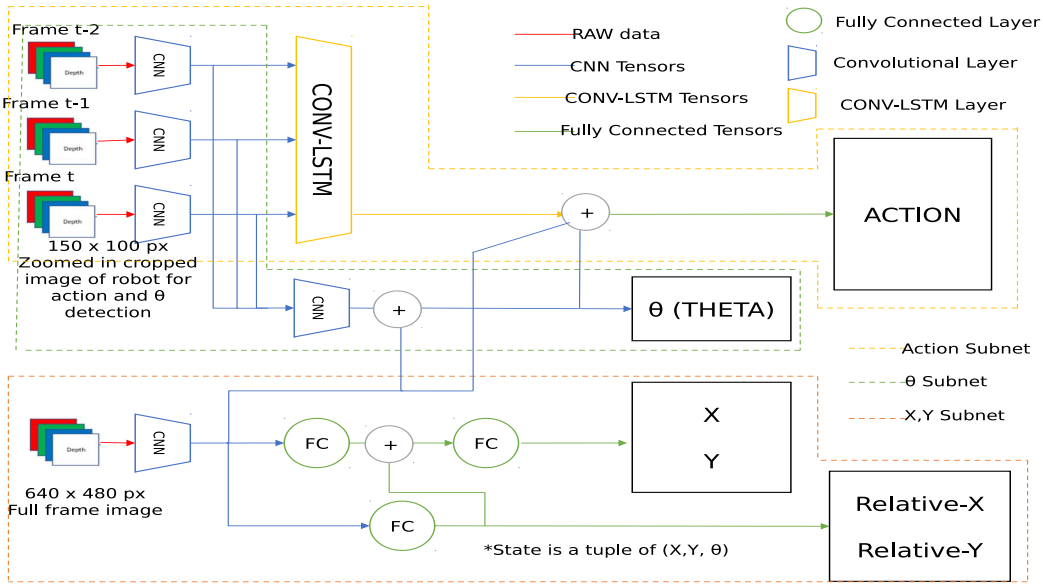


(a) (b)

3.3.2 Phase 2

Phase 2 or second part of the model contains the section of the network assigned to do all the heavy lifting, the task of this phase is to get the X, Y and θ co-ordinates and depth data from the depth sensor and detect the action and state from these features

Figure 3.13: SA-Net Network



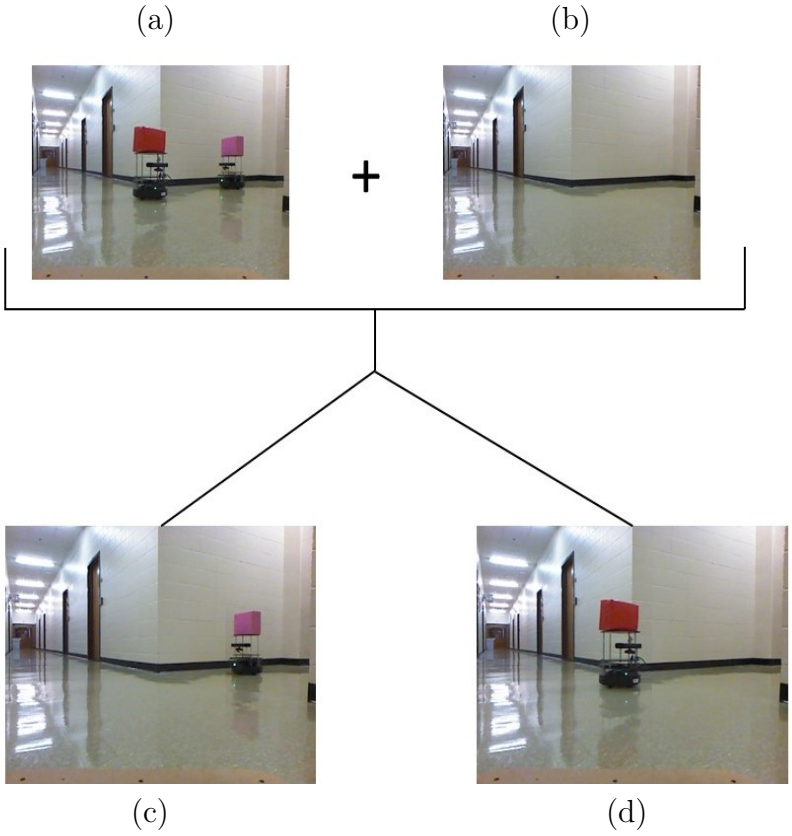
SA-Net consists of two networks which we define as a duplex network, SA-Net is called so because of the two streams of networks which run side by side. We have divided it as top network Stream and bottom network stream.

Bottom Network Stream

The bottom Network stream is tasked with learning the states X and Y of the object. The input for this layer is the full frame image of size (640, 480, 4) for current time step t. This frame is the same image that passed to the object Detection in Section 3.3.1. Since we can have more then one object the network would get confused with which objects for which we have to detect the State. We used the method of masking. Masking of the image consist of having a frame held in memory when phase - I does not detect an object. For example in Figure 3.14 we see there are two turtlebots both red and pink. The solution is by masking all but that object as shown in Figure 3.14(a) and Figure 3.14(b). This is done by cropping the object's [x_1, y_1, x_2, y_2] on the sensor frame shown in Figure 3.14(a) and placing that image on the mask image shown in Figure 3.14(b). This produces n_{obj} number of images,

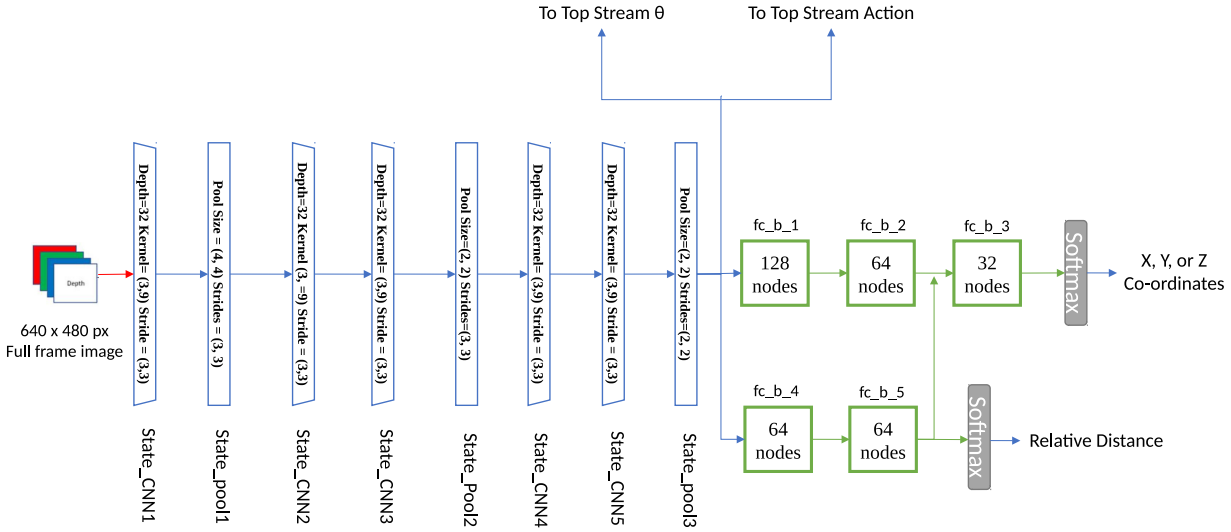
where n_{obj} is the number of objects.This masked image is fed to the bottom stream network for state recognition.

Figure 3.14: Masking Process, (a) shows the image captured by the sensor for the turtlebot experiment, (b) is the image stored in memory when there is no object found in this case a turtlebot, (c) and (d) are masked images produced by this mask



This layer consists of five CNN layers and three average pooling layers.

Figure 3.15: Detailed SA-Net Bottom Network layers



The five CNN layers labeled as *state_CNN1, state_CNN2, state_CNN3, state_CNN4, state_CNN5* and pooling layers as *state_pool1, state_pool2, state_pool3*. Table 3.1 will give you the layers and its parameters for bottom stream shown in Figure 3.15

Layers *fc_b_4* and *fc_b_5* are part of what we define as an intercept branch. This intercept learns the relative distance from the observer and the object for each of the coordinates and can be negative integer. This branch aids in learning the relative distance from the observer to the observed object.

Layer	Parameters
State_CNN1	Depth=32 Kernel= (3,9) Stride = (3,3)
State_pool1	Pool Size = (4, 4) Strides = (3, 3)
State_CNN2	Depth=32 Kernel (3, 9) Stride = (3,3)
State_CNN3	Depth=32 Kernel= (3,9) Stride = (3,3)
State_pool2	Pool Size=(2, 2) Strides=(3, 3)
State_CNN4	Depth=32 Kernel= (3,9) Stride = (3,3)
State_CNN5	Depth=32 Kernel= (3,9) Stride = (3,3)
State_pool3	Pool Size=(2, 2) Strides=(2, 2)
fc_b_1	Nodes=128
fc_b_2	Nodes=64
fc_b_3	Nodes=32
fc_b_4	Nodes=64
fc_b_5	Nodes=32

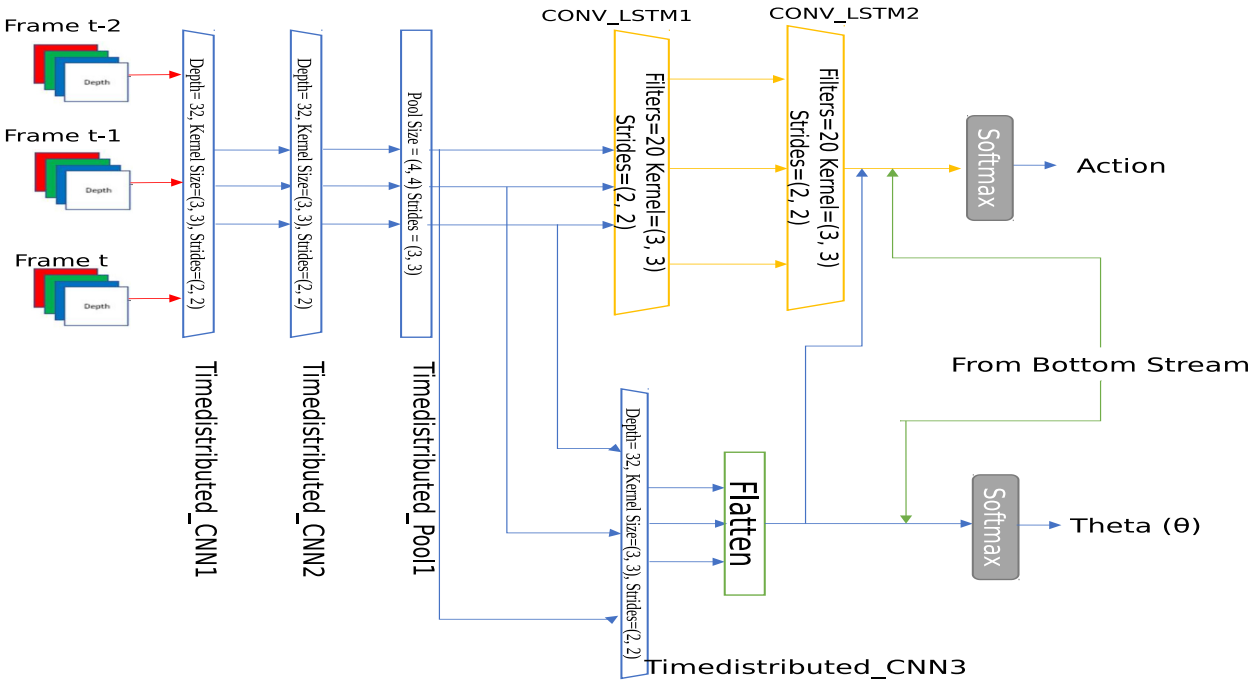
Table 3.1: Bottom Stream Layer Parameters. Each Row is labeled from layers shown in Figure 3.15

Top Network Stream

The top network stream is tasked with detecting Action and θ. This network takes a temporal data as input which is passed to a Time Distributed CNN layer. This temporal input of size three is labeled as *t-2, t-1 and t* where t is the current time frame for which action has to be detected. The reason we are using three frames is because two frame is too less for the network to learn anything and anything more than three becomes too heavy for a simple computer system to handle. The features from this set of layers is then pass to a CONV-LSTM layer [22] the output from this Conv-LSTM layer is flattened and directly passed to softmax to classify the action. The Conv-LSTM helps the network remember what happened

in the past frames. For θ the features from the time distributed layer is passed to another time distributed layer. The features of this layer is also passed to directly to softmax layer.

Figure 3.16: Detailed SA-Net Top Network layers



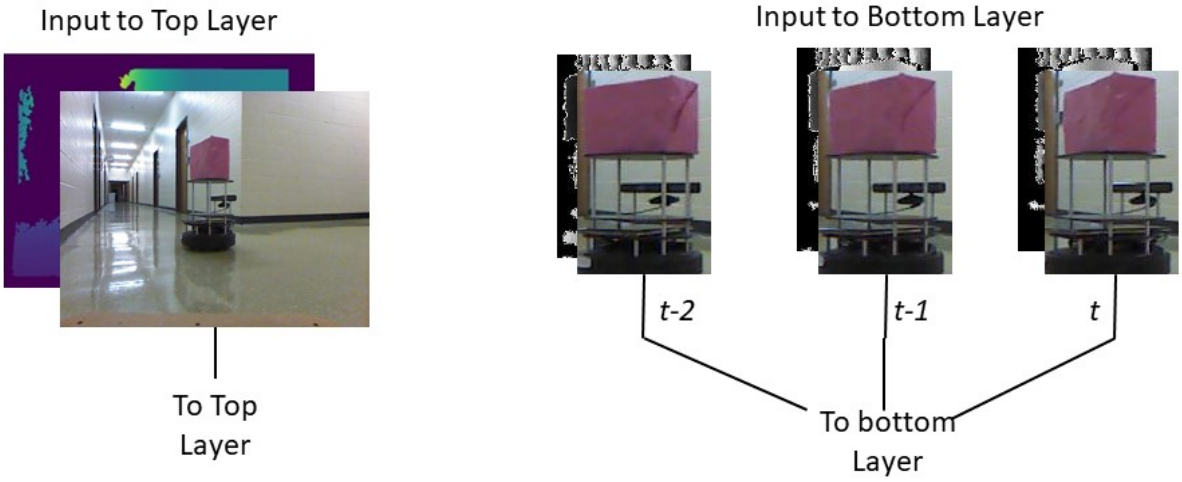
This part of the network has three Time distributed layers names *Timedistributed_CNN1, Timedistributed_CNN2, Timedistributed_CNN3*, two Conv-LSTM layers labeled *CONV_LSTM1, CONV_LSTM2*. Table 3.2 shows all the layers and its parameters.

Layer	Parameters
Timedistributed_CNN1	Depth= 32, Kernel=(3, 3), Strides=(2, 2)
Timedistributed_CNN2	Depth= 32, Kernel=(3, 3), Strides=(2, 2)
Timedistributed_pool1	Pool Size = (4, 4) Strides = (3, 3)
Timedistributed_CNN3	Depth=32 Kernel= (3,9) Stride = (3,3)
CONV_LSTM2	Filters=20 Kernel=(3, 3) Strides=(2, 2)
CONV_LSTM2	Filters=20 Kernel=(3, 3) Strides=(2, 2)

Table 3.2: Top Stream Layer Parameter. Each Row is labeled from layers shown in Figure 3.16

In Figure 3.16 and Figure 3.15 the features from both the top stream and bottom stream share features, because we want the network to learn the Action based State. This includes X,Y and θ.

Figure 3.17: Inputs into the Network layers. The image on the left is the input to the top layer which recognizes the Coordinates of the object this frame is of size 640 by 480. The set of three frames to the right are the cropped frames of size 100 by 150. These frames will be the input to the bottom layer



3.4 Training

The dataset is first divided into five stratified cross-validation folds. The stratified cross validation is used because when the sets are created the initial samples are a single action for example it could be $a_t = stop$. To prevent the network from receiving an unbalanced data set we plan to use the stratified approach when shuffling. The whole experiment is divided into two phases and the networks have to be trained in parallel.

In Phase-I we first trained a Faster-RCNN[21] on the images that have been annotated. We have 500 annotated images. This trained faster-RCNN model is then run on the whole dataset to get the (x_1,y_1,x_2,y_2) of the turtlebot. This is then fed to the YOLO [2] as input.

In this phase we make faster-RCNN train YOLO network till it gives us a decent IOU or intersection over union of about 96.

After Phase-I is sufficiently trained the time series images from time *t , t-1, t-2* which have 4 channels of RGB-D and one full frame image is then fed to the network in batches of 128.This as discussed earlier have to be done on the fly as loading the whole data is impractical. These Images have a time gap of 5 to 10 for the turtlebot experiment and 2 to 6 on the manipulator arm.We define time gap as the number of frames dropped per time step. This is done because the camera sensors have a probability of dropping frames or the turtlebot running slower. For the turtlebot experiment a range of 5 to 10 is taken as time gaps 1 to 4 is too less to see any changes in action. In the manipulator arm experiment the range of 2 to 6 is taken as anything more than 6 the action was already completed and new one has begun this confuses the network.

We trained the network on Intel Xenon E7 processor together with a Nvidia GTX 1080 having 8 GB of RAM. We use a generator to generate the data, because once the data is ready it exceeds 700 GB and having that much RAM on a single system is impractical. The network is almost end to end as there is only an interruption of cropping the image. We used a batch size of 128 with learning rate set to 0.00001 or 10^{-5}. To make sure all the layers are trained we use early stopping to stop the training when the validation loss is same for 5 epochs.

The whole network is written using keras[23] with tensorflow[24] as back-end and python as the main programming language.

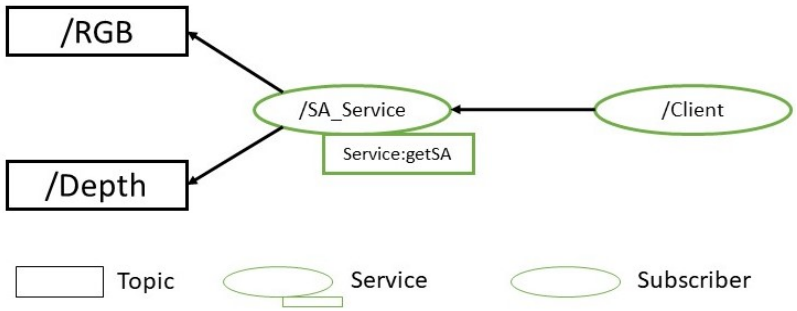
3.5 Running the Experiment

Once the model is trained the weights are then transferred to the robot. The robot which is running on ROS[25]. The network is always on standby as a ROS service. When any algorithm awaiting a state and action pair sends a requests then the network grabs the data from the sensor and returns the state and action as ROS SRV message. The ROS nodes is

designed as Figure 3.18

/RGB and */Depth* are two sensor nodes or topics which are publishers. The */RGB* node uses

Figure 3.18: ROS Nodes Architecture when Deployed on Robot



the RGB sensor of the Kinect v1 to publish RGB images. These images are in format called bayerRGB. These images are little different from our normal images as they have reduced number of green and red pixels. The */Depth* sensor publishes images of two dimension where each pixel representing the distance. The sensors can measure a distance of 400 millimeters to 4000 millimeters for every pixel. Each pixel is stored as a unsigned 16 bit integer. The */SA_Service* is where the network is stored this service subscribes and monitors */RGB* and */Depth* topics when a request is sent the to the service */SA_Service* the SA-net is executed and the results are sent via a customized SRV response which have X, Y, Z, θ and Action as integers as per equations 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9. The node */client* can be any client algorithm.

Chapter 4

Results

This section shows how the network performed on different data-sets, tests, resource usage and ablation studies.

4.1 SA-Net Results

Experiment	Training Set	Testing Set
Turtlebot	48840 samples	12215 samples
Manupilator arm	7714 samples	1932 samples

Table 4.1: Number of samples in the dataset after preprocessing

We trained the network by using stratified k folds. To train our model we used $k = 5$. The cross-validation is to ensure the network is not over-fitting and can work with any random data. We used the built-in package from scikit-learn[26]. Figure 4.2 and Figure 4.3 show the testing set results for each fold. These results are based on 80:20 split of data on training and testing set. The exact number of samples in the training and testing set is given in Table 4.1. The results based on pure physical experiments are explained from Section 4.2.

SA-Net	X	Y	θ	Action
Run 1	98.853	99.96	99.99	99.97
Run 2	98.853	99.96	99.99	99.95
Run 3	98.853	99.95	100	99.95
Run 4	98.885	99.97	99.97	99.94
Run 5	98.81	99.99	100	99.97
Mean ± SD	98.8508±0.0238	99.966±0.01356	99.99±0.01095	99.7374±0.012

Table 4.2: SA-Net results per fold for turtle-bot dataset on testing set. These are the accuracy results of state and action as percentage from non physical experiments

SA-Net	X	Y	Z	θ	Action
Run 1	97.63	95.23	96.54	98.19	99.12
Run 2	97.65	95.19	96.56	98.12	99.14
Run 3	97.62	95.2	96.58	98.23	99.1
Run 4	97.63	95.22	95.59	98.17	99.14
Run 5	97.66	95.21	95.55	98.15	99.16
Mean ± SD	97.638±0.0146	95.21±0.0141	96.164±0.4853	98.172±0.0370	99.132±0.0203

Table 4.3: SA-Net results per fold for PhantomX arm dataset on testing set. These are the accuracy results of state and action as percentage from non physical experiments.

4.2 Physical Experiment

The first main test to the network was done in ideal test conditions. Table 4.4 shows the accuracy results for the network implemented on the turtlebot and PhantomX arm using ROS[25] and frames recorded. This was then check with labels manually. The accuracy is calculated by dividing the number of correct predictions by the ground truth and multiplying it by 100.

Environment	X	Y	Z	θ	Action
Turtlebot± SD	97.23±0.2940	98.12±0.4845	-	96.25±0.6663	96.16±0.6782
Arm ± SD	87.56±0.01458	89.25±0.0158	91.12±0.0333	88.32±0.0089	91.18±0.0142

Table 4.4: SA-Net accuracy in physical experiments in percentage for turtlebot and manipulator arm under normal conditions. Since turtlebots don't have a motion with respect to Z the accuracy is left as blank

4.3 Robustness tests

We evaluated our model with three different types of tests as shown in Table 4.2 to show robustness in the model. The model is first trained on the Training data. This trained network is then subjected to two different tests one is with noise and other with occlusion. Since the only method that can detect state and action is the centroid method we compare SA-Net with the centroid method.

1. **Noise Test:** In this experiment we test the robustness of the network to be able to handle external noise. The noise here is defined as objects which look like or have similar characteristics of the object that we are observing. For this we use a human being with a similar coloured shirt or a box of the same colour just lying on the floor. The models is then tested on this set. Based on third row of Table 4.5 we can clearly see that the centroid method is failing to recognition the object and perform action and state recognition.

Test	X	Y	θ	Action
SA-net w/Noise ± SD	92.65±0.8670	96.65±0.7184	95.23±0.3964	95.12±0.7582
Centroid method w/Noise ± SD	34.2±6.6151	44.43±1.4190	23.23±1.3137	42.45±1.8749

Table 4.5: SA-Net noise test results for turtle-bot experiment in percentage. Average of 15 experiments 5 for colored shirt, 5 for box and 5 with dimmed light

2. **Occlusion Test:** In this experiment we test to see how well the models work on occlusion. To do this the object is covered partially to approximate 50% occlusion; by covering the object by a cardboard box or a white towel. The robots are then made to run the experiments again and the results are tested against ground truth which are manually labeled. From third row Table 4.6 we can clearly see that our model outperforms the other model.

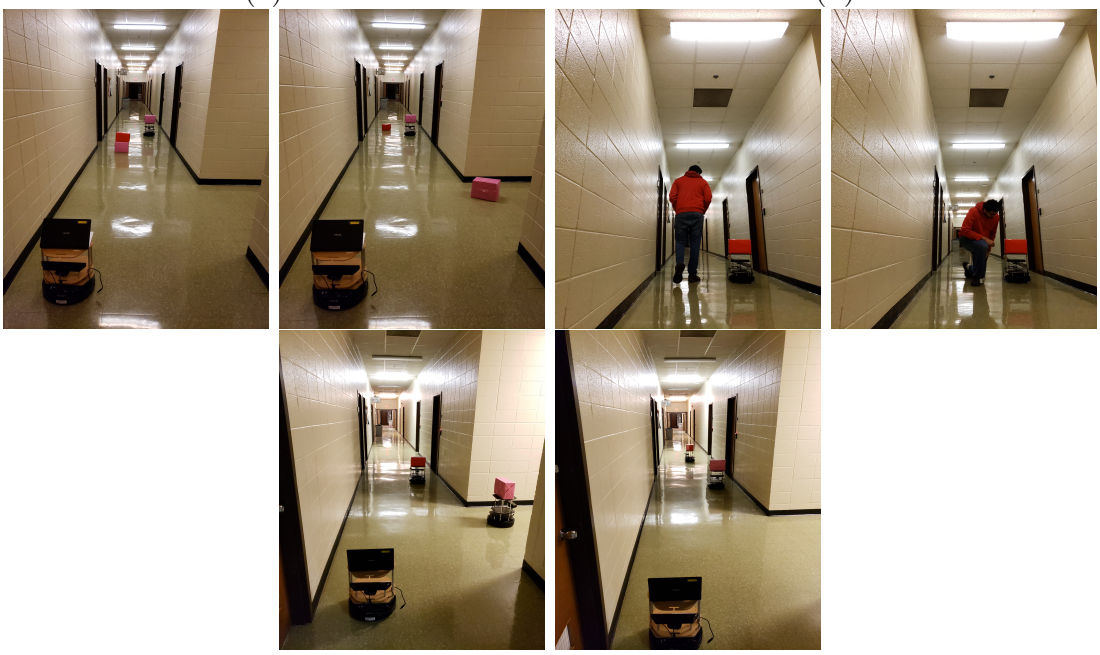
Experiment	X	Y	θ	Action
SA-Net w/Occlusion±SD	45.15±0.8682	54.60±0.7561	64.12±0.9874	46.36±1.007
Centroid method w/Occlusion±SD	18.232.1305±	17.34±1.5698	14.42±0.1458	43.12±0.8046

Table 4.6: SA-Net occlusion test results for turtle-bot experiment in percentage. The results are an average of 10 experiments.The first row shows how SA-Net works on normal conditions. Second row is how SA-net performed on the occlusion tests and third row shows how the centroid method compares to the same test. We can see that SA-Net doesn't show complete failure in occlusion, where as centroid method does the not do very well in State detection.

To show that our network works for different environments we also tested our model on a manipulator arm by detecting the state and action of the gripper. Gripper of the manipulator arm is the part of the arm that emulates our hand and has the capacity to grip or hold on to

Figure 4.1: The figures below are images of the experiment being conducted for the noise test. We have broken the noise test into three categories. The random placement of similar boxes around the experiment area (a) shows two of the five test cases for this type of test.Same color external objects, (b) shows two of the five images of the experiment being conducted here we use a person with the same color jacket as the box on top of the turtlebot, this person would be walking up and down the experiment area.Low light test (c) shows two of the five experiments being conducted here we dimmed the light by 50%

(a) (b)



(c)

a object. Since the manipulator arm works on a 3-Dimensional space the only change that was done on the network was changing the last softmax layers to Incorporate the extra Z coordinate and the up and down actions explained in Section 1.5.2. Table 4.4 shows the results performed by our network on the manipulator arm. The reduction of accuracy compared to turtlebot experiment is due to the increase in complexity because of the introduction of a new dimension and increase in action space.

Figure 4.2: The two images are a sample of the occlusion test being performed for the turtlebots. We cover one half of the turtlebot with a piece of cloth

4.4 Ablation Study

In this section we perform ablation study on parts of the network to check and see if that part of the network indeed contributes to the results of the model. Ablation study in machine learning is the method of removing a part or parts of a machine learning model and then conducting experiments on that model which has been re-modeled with part of the model missing.

4.4.1 Relative X and Y

In this study we eliminate the part of the network which contributes to relative X and Y this will show how the network will work without training the sub-network of relative X and Y. We expect the network to memorize the location by relying more on RGB data and also unable to detect when there is change in position.

Method	X	Y	θ	Action
SA-Net Normal± SD	97.23±0.2940	98.12±0.4845	96.25±0.6663	96.16±0.6782
SA-Net w/o Rel X & Y± SD	81.378±1.4965	91.436±1.4559	91.237±1.2587	83.633±1.8564

Table 4.7: Ablation Study on Relative X and Y on turtlebot expriment

The main reason this part of the network exists is to learn the distance in the floor map when compared to what its seeing, removing this part of the network the network will fail to learn the distance or gap between the observer and observed object. There over fitting on training set with almost 100% results but when tested on training set we get the results on table 4.7

4.4.2 Temporal data for theta

In this study we eliminate the part of the network responsible for temporal information θ and check if θ really does need the temporal information. Since we as human beings can recognize the direction of an object from a single image. The network should be able to recognize the direction (θ) without the use of temporal data. Since theta(θ) is required to understand the action the accuracy of action will reduce.

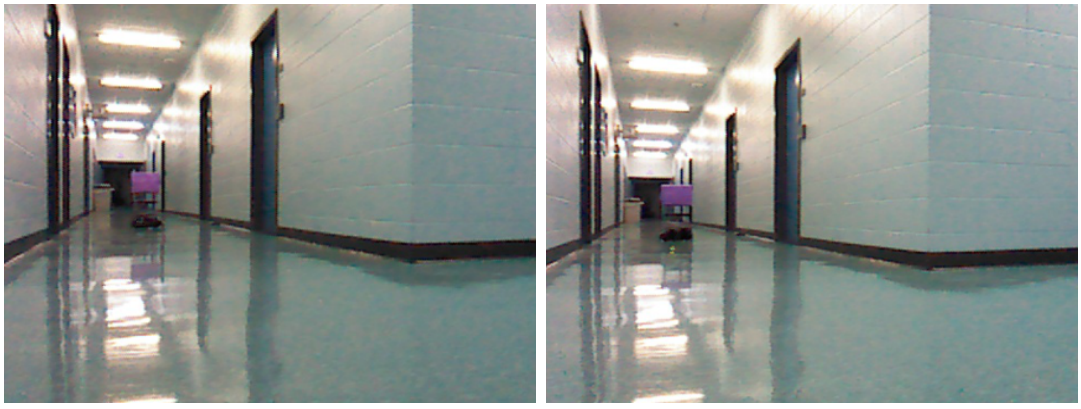
Method	X	Y	θ	Action
SA-Net Normal± SD	97.23±0.2940	98.12±0.4845	96.25±0.6663	96.16±0.6782
SA-Net only data from time t± SD	96.56±0.0063	98.32±0.0048	79.43±1.3344	78.86±3.5681

Table 4.8: Involvement of Time Series data on θ

This feature of our model is for handling symmetric objects with features which are very hard to distinguish which direction the object is facing. For example a man wearing a mask on his back. But if he starts to walk we know which side is his true front. During this ablation study we found that it is required by the network to get the features of the three time steps to identify where the bot is going. By removing this from the network there are many position in out turtle-bot experiment that its very hard to identify where the turtle bot is heading.

Figure 4.3 shows the complication of theta and why we need the temporal features for θ for cases like this shows the importance of why we need the temporal information to help identify θ

Figure 4.3: Two frames of *theta* in the turtlebot experiment.(a) is when the turtlebot is facing north and (b) is when turtlebot is facing south



(a) (b)

4.4.3 Multimodal Data

We study why depth data is needed for our network and how it will behave when its removed. Based on only RGB data we can make the network learn the state and action. But there are many states which are the same and will require the need of multimodal data. Without this section of the network the network can memorize certain features and overfit on those characteristics.

Method	X	Y	θ	Action
SA-Net Normal± SD	97.23±0.2940	98.12±0.4845	96.25±0.6663	96.16±0.6782
SA-Net RGB ± SD	87.23±1.4968	95.12±1.4861	83.56±1.5189	81.12±1.00001

Table 4.9: Involvement of depth as multimodal

4.4.4 Object Detection

In this study we removed the object recognition part of the network and fed the sequence of frames. Object Detection is the core part of phase-I explained in Section 3.3.1

Method	X	Y	θ	Action
SA-Net Normal± SD	97.23±0.2940	98.12±0.4845	96.25±0.6663	96.16±0.6782
SA-Net without Obj Detect± SD	68.74±2.2565	69.95±1.2556	21.65±2.3909	33.89±0.9604

Table 4.10: Ablation study for Object Detection

Object Detection is a main part of our model without it action or theta will fail to work. We tried passing the network with the non cropped images for which the results came out quite bad as seen on table 4.10.

4.5 Resource Usage

SA-Net takes a long time to train on our computer mainly because of the sheer size of the data-set along with the time required to prepossess the data before passing it to the network. Since we decided to work with 5 fold cross validation.

SA-Net	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Time in hours	26:15	26:43	26.54	26.18	26.78

Table 4.11: Time to train all values in HH:MM

Table 4.11 show the time required to train the model end to end. This is mainly attributed to the huge data-set size and memory constraints. An ideal system would need 700Gb of RAM to work at max capacity which is just absurd. So we had to resort to data generation on the fly using 10 processing threads which keeps generating data and provides this batch to the network. This can be seen similar to how a queue of people dropping of mail in a post-office. Once the data is given to the model it then starts to generate next batch taking into account what the other 9 processes are processing.

Table 4.12 is the amount of total ram withheld by the ROS service in order to work. This is the max amount taken by the process. Table 4.13 shows max time in seconds it requires to

Method	Memory usage
SA-Net	742MB± 3MB

Table 4.12: SA-Net memory usage

Method	Prediction Time
SA-Net + Retina-net	6s± 0.4s
SA-Net + Yolov2	1.1s± 0.3s

Table 4.13: Performance with Respect to Prediction Time for two objects

execute the model for the turtlebot experiment which has two objects. The network should be able to detect both the action and the state of the object.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This work provides a method for recognizing state and action for inverse reinforcement learning algorithms. Table 4.2 and Table 4.3 for the turtlebot experiment and the manipulator arm shows that the network recognizes the state and action of the objects accurately on a testing set. Table 4.4 shows that the network recognizes the state and action of the objects accurately on physical experiments for both turtlebot and manipulator arm. These results also prove that the network works on different environments by running the same model in two completely different environments. Table 4.12 and Table 4.13 show the resource utilization in real time on a simple system with limited resources.

SA-Net can also hold its accuracy even in noisy environments and occlusion. The network can work on completely different environments (example: phantomX manipulator arm) which have increased state or action space and additional dimensions. SA-Net is able to recognize the direction of an object which has similar features by using features from temporal data. We also conducted ablation studies discussed in Section 4.4 on the network to show that the complicated network was necessary.

5.2 Future work

Since the model is not completely end to end we will be implementing the network similar to how an RCNN [3] works; it should be able to detect state and action on the fly using region proposals.

In our state space θ was defined as discrete integers representing direction, in the future we want to implement a network capable of learning theta (θ) as a continuous space rather than just a directions.

As the observer is turned at an angle, the accuracy of the network slowly diminishes even if it was in the same state. The next iteration of the network we want to add a way to suppress this attenuation due to change in observer angle.

Currently the network trained on red and pink turtlebots. An addition of a green turtle bot should not be a problem. Since we have only defined two classes for the turtlebot experiment the model might recognize it as either red or green, because red and green are the only two classes trained. In future we want the model to be able to learn new objects on the fly, this will add the ability of being able to recognize more than what it was trained for.

Bibliography

[1] Yue Zhao, Yuanjun Xiong, Limin Wang, Zhirong Wu, Dahua Lin, and Xiaoou Tang. Temporal action detection with structured segment networks. *CoRR*, abs/1704.06228, 2017.

[2] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[3] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] Kenneth Bogert and Prashant Doshi. Multi-robot inverse reinforc ement learning under occlusion with interactions. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 173–180, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory, 1997.

[9] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013.

[10] Alberto Montes, Amaia Salvador, and Xavier Giró i Nieto. Temporal activity detection in untrimmed videos with recurrent neural networks. *CoRR*, abs/1608.08128, 2016.

[11] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition and detection by combining motion and appearance features. 2014.

[12] Wei Chen and Jason Corso. Action detection by implicit intentional motion clustering. pages 3298–3306, 12 2015.

[13] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin A. Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust RGB-D object recognition. *CoRR*, abs/1507.06821, 2015.

[14] Georgios Georgakis, Md. Alimoor Reza, Arsalan Mousavian, Phi-Hung Le, and Jana Kosecka. Multiview RGB-D dataset for object instance detection. *CoRR*, abs/1609.07826, 2016.

[15] Saurabh Gupta, Ross B. Girshick, Pablo Arbelaez, and Jitendra Malik. Learning rich features from RGB-D images for object detection and segmentation. *CoRR*, abs/1407.5736, 2014.

[16] Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *CoRR*, abs/1503.08909, 2015.

[17] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[18] M. S. Ryoo, Thomas J. Fuchs, Lu Xia, J. K. Aggarwal, and Larry H. Matthies. Early recognition of human activities from first-person videos using onset representations. *CoRR*, abs/1406.5309, 2014.

[19] Rui Hou, Chen Chen, and Mubarak Shah. Tube convolutional neural network (T-CNN) for action detection in videos. *CoRR*, abs/1703.10664, 2017.

[20] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[21] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017.

[22] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[23] François Chollet et al. Keras, 2015.

[24] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[25] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.