CHAPTER 1
INTRODUCTION

For visually impaired people cane is a common tool used for navigation. This tool is primitive and includes persons experience and senses to work. After conducting the survey and noting down the needs for visually impaired people we came to conclusion that there is a need for much smarter system. The plan is to add an embedded system module based on raspberry pi to be placed on cane. The cane will have AI embedded on to it which will navigate person from source to destination avoiding obstacles like potholes, crowds etc. Basic structure of cane is inspired by how guide dogs work by pulling person through crowd similarly the cane will have a wheel below and pull the person and navigate through crowd. An AI based navigation software trained via reinforcement learning is embedded on the raspberry pi which uses object detection to get input for obstacles and make decisions accordingly.

1.1 Overview

Currently there are thousands of blind people all over the globe. These include people from low sightseeing to complete lost of visual. They find it very difficult while crossing the road or reaching to their respective destination with the help any other individual. The traditional stick cannot help to detect the obstacles in front or the potholes in the way. It is outdated. Hence there is a need to update it using today's technology. Although a lot of systems are proposed but none of them are adopted due to unconventional designs and lack of adaptiveness in real word environment.

Most of the cane-based designs on the market are embedded systems and don't work well on dynamic environments. The hollow market shows how impractical just embedded based solutions are. We have trained our AI model to detect objects on reinforcement learning so that it can be used on dynamic environments. Our patented hardware design is based on guiding-assistance rather than feedback-assistance ie. It works by guiding the person through pull-force by avoiding any obstacles. This is a completely different solution which is there on the current market and does not require any training.

1.2 Motivation

Most of the tools made for visually impaired people are primitive and every smart tools are rejected due to lack of their practicality. As there is a need for smart and practical tool for day to day life of visually impaired person we plan to create such tool, this smart cane is inspired by guide dogs which help in navigation and assistance to visually impaired. the smart cane will work in similar fashion by navigating person by guiding through voice and vibratory haptic.

1.3 Problem Definition and Objectives

To make a AI based robotic assistant which can help blind people in navigation and day to day struggles and as a result help in overall mobility of user. Making it available at much cheaper prices without compromising with the functionalities and minimizing its maintenance and operating cost.

Objectives

To build hardware and supporting software for a smart cane that will:

- Detect potholes
- Detect vehicles
- Detect stairs
- Detect fallen items
- Detect and avoid pedestrians
- Provides audio and vibratory feedback
- Help the user to navigate from current location to given destination.

1.4 Project Scope and Limitations

Scope: Mobility is one of the biggest challenge faced by a visually impaired person and current market solution include white cane which requires lot of training to use and guide dogs which are expensive and hard to manage. To solve the problem of mobility we combined both the advantages of a stick and guide dog into a features maintenance and ease of use through our novel idea which replicates the working dog for its assistance. Problem of mobility includes navigation and obstacle avoidance.

Limitations: To supplement lack of vision with other senses lot of R&D and Testing is required to build a truly useful product. The product has to be in line with current technologies and trends with visually impaired people. The product is expected to have a high learnability curve and a good interface interaction

1.5 Methodologies of Problem solving

Reinforcement Learning for obstacle avoidance and crowd maneuvering

Reinforcement learning is an area of Machine Learning. Reinforcement. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from the supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of training dataset, it is bound to learn from its experience.

To solve the problem of obstacle avoidance we have used deep Q-learning. Q-learning, like virtually all RL methods, is one type of algorithm used to calculate state-action values. It falls under the class of temporal difference (TD) algorithms, which suggests that time differences between actions taken and rewards received are involved.
We have set a 2-D virtual simulation environment on which we train our agent using this model and finally embedding it into our hardware

Tensorflow Api for object detection

Creating accurate machine learning models capable of localizing and identifying multiple objects in a single image remains a core challenge in computer vision. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. we have created a custom image database to train our model to identify objects like pedestrians, potholes, zebra crossing etc. and giving voice haptik to the user.

Android UI for point to point navigation using Google maps API

Current trends of Blind UI is leaning towards smartphones due to introduction of voice assistants and various sensors in phones, To make use of this we have made a custom UI to connect with our cane for better experience.

CHAPTER 2
LITERATURE SURVEY

In the paper "Human and Car Detection System for Blind People" by Ayat Nada , Samia Mashaly, Mahmoud A. Fakhr, Ahmed F. Seddik proposed a solution, represented in a smart stick with infrared sensor to detect stair-cases and pair of ultrasonic sensor to detect any other obstacles in front of the user, within a range of four meters. Moreover, another sensor is placed at the bottom of the stick for the sake of avoiding puddles. Speech warning messages and the vibration motor are activated when any obstacle is detected. This proposed system uses the microcontroller 18F46K80 embedded system, vibration motor and ISD1932 flash memory. The stick is capable of detecting all obstacles in the range 4 meter during 39 ms and gives a suitable respect message empowering blind to move twice his normal speed because she/he feels safe. The smart stick is of low cost, fast response, low power consumption, light weight and ability to fold.

In the paper "Adaptive State Space Quantisation for Reinforcement Learning of Collision-Free Navigation" by Ben J.A. Krose and Joris W.M. van Dam | 1992 IEEE

In this paper the author describes a self-learning control system for a mobile robot. Based on sensor in formation the control system has to provide a steering signal in such a way that collisions are avoided. Since in our case no 'examples' are available, the system learns on the basis of an external reinforcement signal which is negative in case of a collision and zero otherwise. Rules from Temporal Difference learning are used to find the correct mapping between the (discrete) sensor input space and the steering signal. We describe the algorithm for learning the correct mapping from the input (state) vector to the output (steering) signal, and the algorithm which is used for a discrete coding of the input state space.

Drishti: An Integrated Indoor/Outdoor Blind Navigation System and Service | Lisa Ran, Sumi Helal and Steve Moore | University of Florida, Gainesville, FL 32611, USA

The author proposes model Drishti a cane based robot which uses a precise position measurement system, a wireless connection, a wearable computer, and a vocal communication interface to guide blind users and help them travel in familiar and unfamiliar environments independently and safely. Outdoors, it uses DGPS as its location system to keep the user as close as possible to the central line of sidewalks of campus and downtown areas; it provides the user with an optimal route by means of its dynamic routing and rerouting ability. The user can switch the system from an outdoor to an indoor environment with a simple vocal command. An OEM ultrasound positioning system is used to provide precise indoor location measurements. Experiments show an in-door accuracy of 22 cm. The user can get vocal prompts to avoid possible obstacles and step-by-step walking guidance to move about in an indoor environment. This paper describes the Drishti system and focuses on the indoor navigation design and lessons learned in integrating the indoor with the outdoor system.

Reinforcement Learning: An Introduction | Richard.R.Sutton, Andrew G Barto | MIT Press

This paper describes Reinforcement learning which is like many topics with names ending in -ing, such as machine learning, planning, and mountaineering, in that it is simultaneously a problem, a class of solution methods that work well on the class of problems, and the field that studies these problems and their solution methods. Reinforcement learning problems involve learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. Moreover, the learner is not told which actions to take, as in many forms of machine learning, but instead must discover which actions yield the most reward by trying them out. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These three characteristics—being closed-loop in an essential way, not having direct instructions as to what actions to take, and where the consequences of actions, including reward signals, play out over extended time periods—are the three most important distinguishing features of reinforcement learning problems.

CHAPTER 3
SOFTWARE REQUIREMENTS SPECIFICATIONS

3.1 Assumptions and Dependencies

It is assumed the project is currently a proof of concept and the agent is trained only on two dimensional environment. This project is ade fully on open source softwares and libraries. the project follows Incremental SDLC model for development.
Dependencies for model are Pygame and Pymunk for building environment. keras for training RL model. TensorFlow API for object detection. Android Studio for App development and raspberry pi support for interfacing it on hardware.

3.2 Functional Requirements

3.2.1 Object detection module

The object detection module should detect the objects that it is trained to detect and should do it with great accuracy.

3.2.2 Object avoidance module

The object avoidance module should should plan a path for avoidance of said objects and it should do it with great efficiency.

3.3 External Interface Requirement

This section provides a detailed description of all inputs into and outputs from the system. It also gives a description of the hardware, software and communication interfaces and provides basic prototypes of the user interface.

3.3.1 User Interfaces

Verbal commands to the android application on user’s mobile device for navigation to destination.

3.3.2 Hardware Interfaces

The system will communicate with the user using sensory input from a vibratory motor.

3.3.3 Software Interfaces

1. The software on the SOC will communicate with the users mobile device for exchanging API, GPS and verbal instructions.
2. The google map identify the user location and find the route to the destination in real time.

3.3.4 Communication Interfaces

The communication is handled via on board bluetooth module and the API requests are handled by the users mobile device.

3.4 Nonfunctional Requirements

3.4.1 Performance Requirements

The application should be robust and reliable.

The application must be interoperable.

The application should be easy to modify or maintain.

The application should provide real time location information.

3.4.2 Safety Requirements

System used shall not cause any harm to human users.

System should capture the images in real time and also process these images in real to avoid any working fault.

3.4.3 Security Requirements

The system should identify the source and destination location correctly on google maps and update that information in real time.

3.4.4 Software Quality Attributes

System should be easy to use.

The system should work in real time environment.

The system should be secured.

The system should generate and follow the accurate resultant path.

3.5 System Requirements

3.5.1 Database Requirements

Non relational databases or file system with an XML file for locating data will be used.

3.5.2 Software Requirements

1. Trained model embedded on the raspberry pi
2. Python 3 with dependencies
3. Tensor flow, Keras and other deep learning libraries.

3.5.3 Hardware Requirements

1. Raspberry pi
2. Camera module
3. GPS module
4. Servo motor
5. DC geared motor
6. Vibrator motor
7. Microphone module

3.6 Analysis Models: SDLC Model to be applied

The Waterfall Model was first Process Model to be introduced. It is very simple to understand and use. In a Waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. Waterfall model is the earliest SDLC approach that was used for software development.

In "The Waterfall" approach, the whole process of software development is divided into separate phases. The outcome of one phase acts as the input for the next phase sequentially. This means that any phase in the development process begins only if the previous phase is complete. The waterfall model is a sequential design process in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Conception, Initiation, Analysis, Design, Construction, Testing, Production/Implementation and Maintenance.

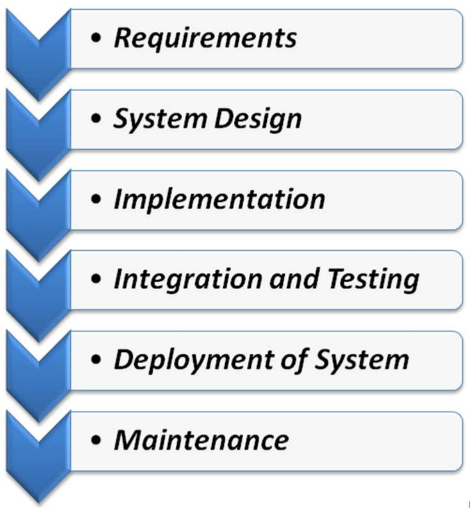


Fig:3.1 SDLC model

CHAPTER 4
SYSTEM DESIGN

4.1 System Architecture

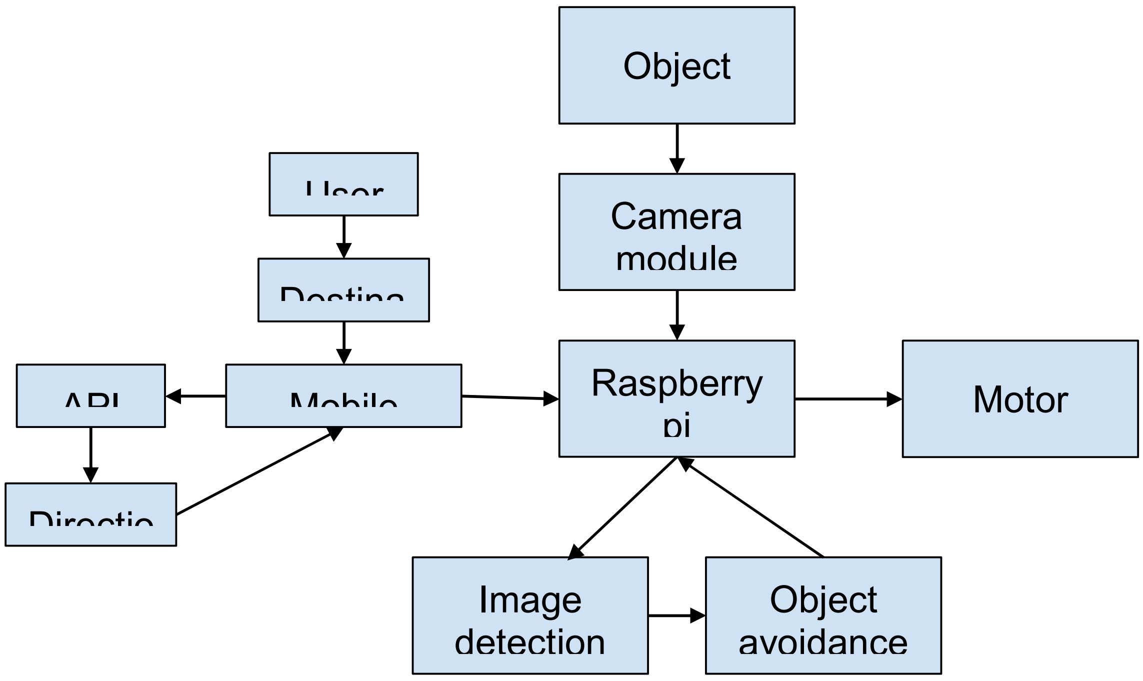


Fig:4.1 System architecture

4.2 Mathematical Model

Let S = {s,e,X,Y,Fme,DD,NDD,Mem}

s=Initial State

e=End State

X=Input given = {X1,X2}

X1 = Destination location given by user

X2 = Images captured by camera module

Y = Output obtained = {Y1}

Y1 = Motion commands

Fme = Object detection and avoidance

DD = It is deterministic data

NDD = It is Non-Deterministic data

Mem = Total Memory used by algorithm for image processing and decision making

4.2 Data Flow Diagrams

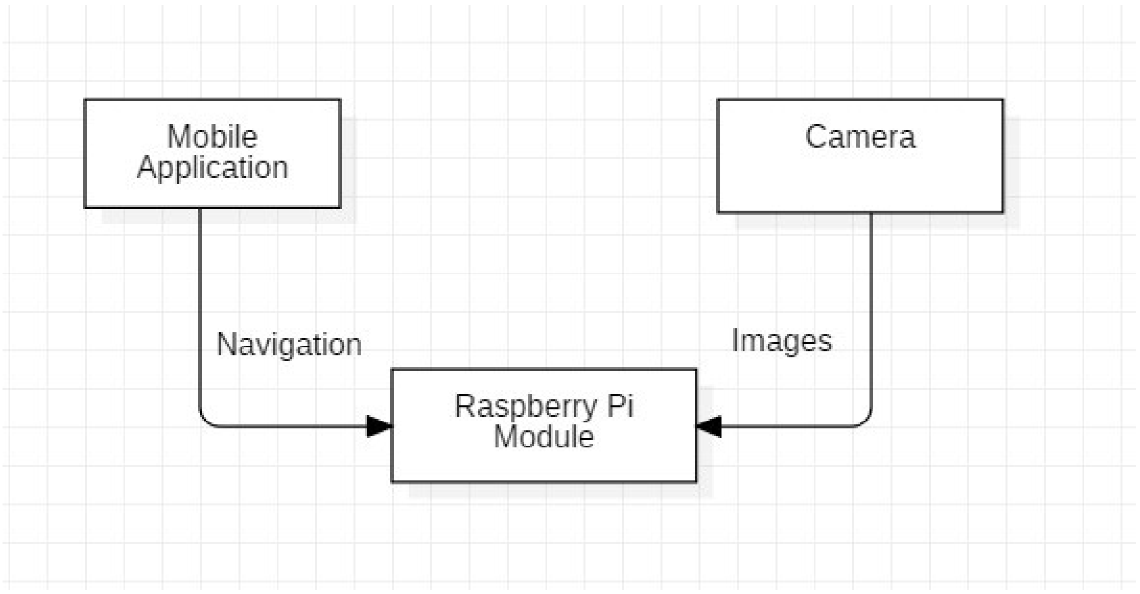


Fig. 4.2 DFD Level 0

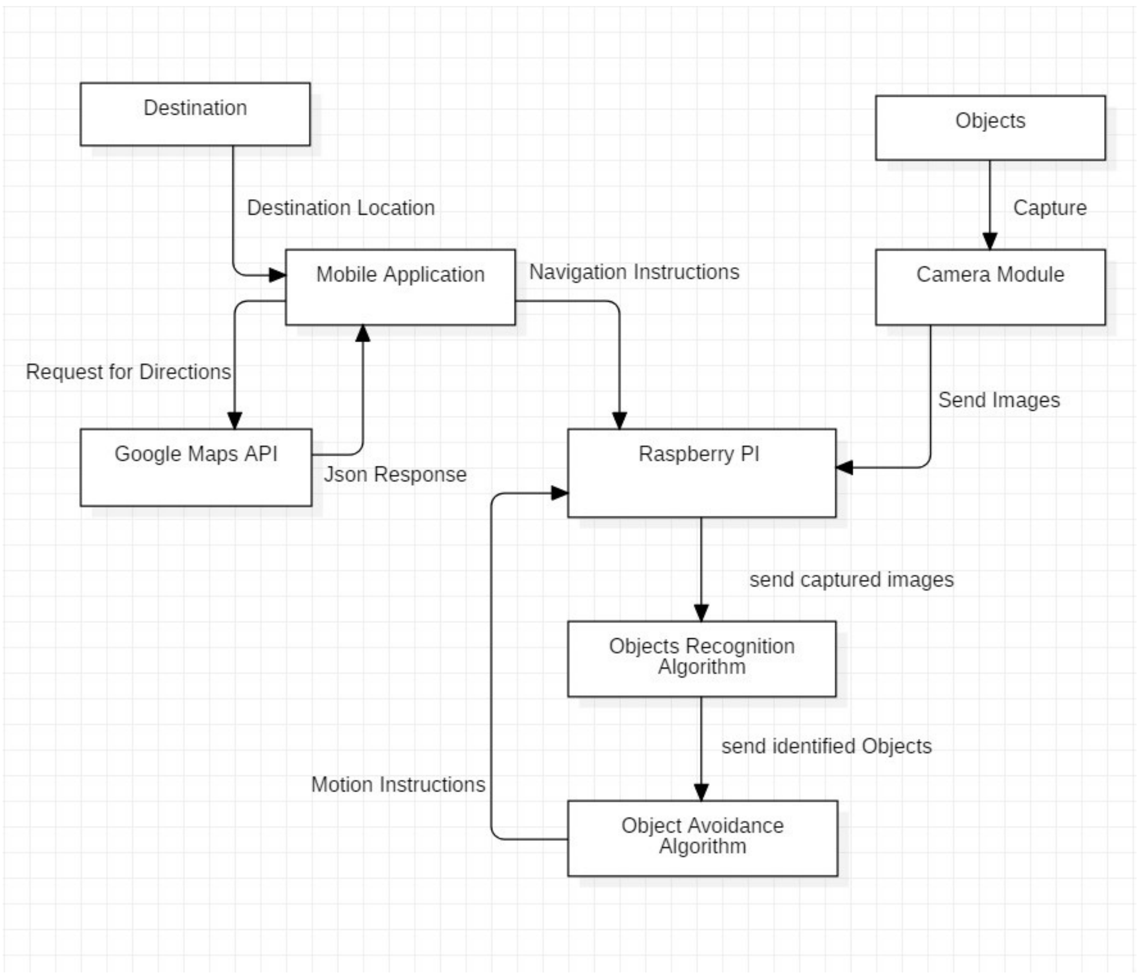


Fig:4.3 DFD Level 1

4.3 Sequence Diagram

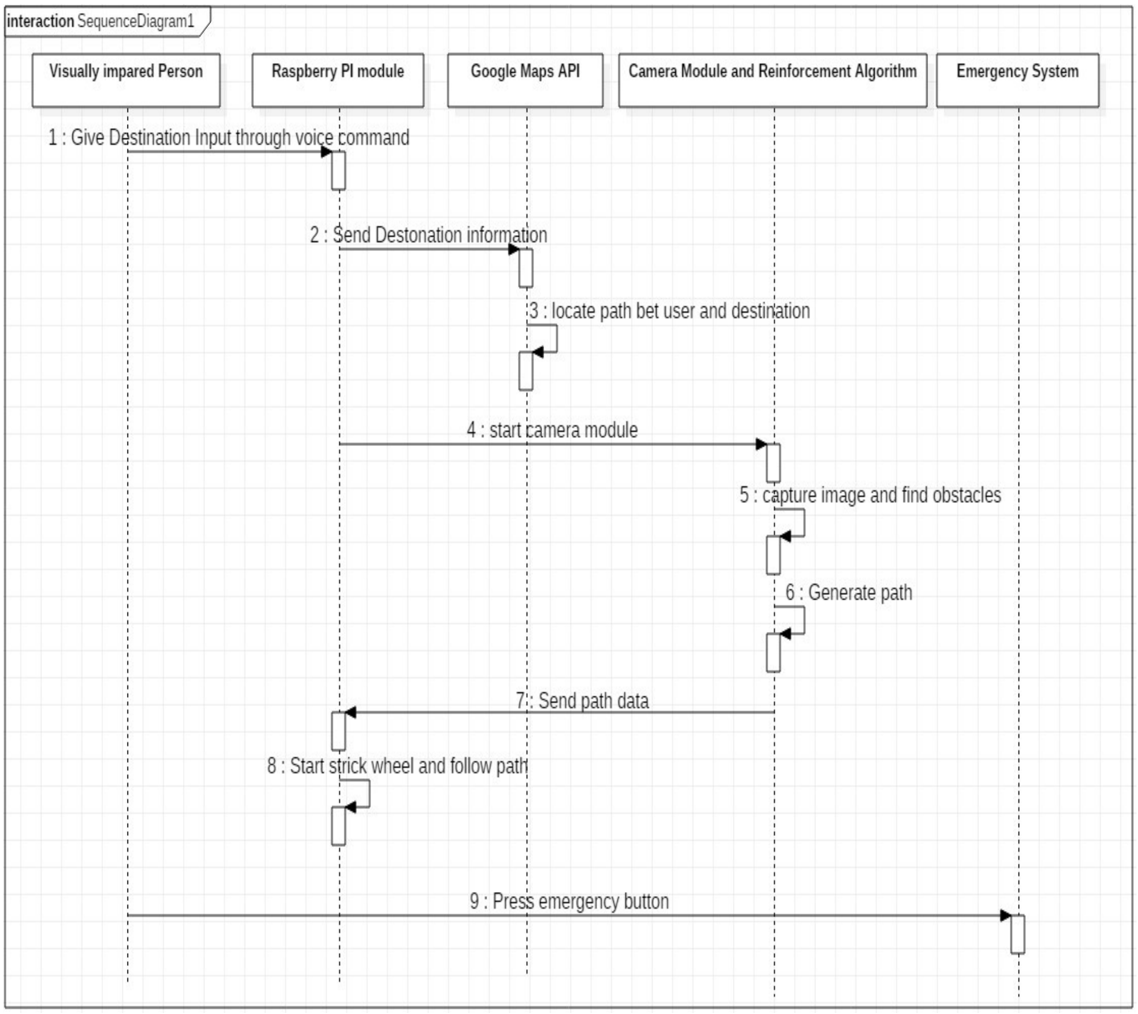

interaction SequenceDiagram1
Visually impared Person
Raspberry PI module
Google Maps API
Camera Module and Reinforcement Algorithm
Emergency System
1 : Give Destination Input through voice command
2 : Send Destonation information
3 : locate path bet user and destination
4 : start camera module
5 : capture image and find obstacles
6 : Generate path
7 : Send path data
8 : Start strick wheel and follow path
9 : Press emergency button

Fig: 4.4 Sequence Diagram

4.5 UML Diagram

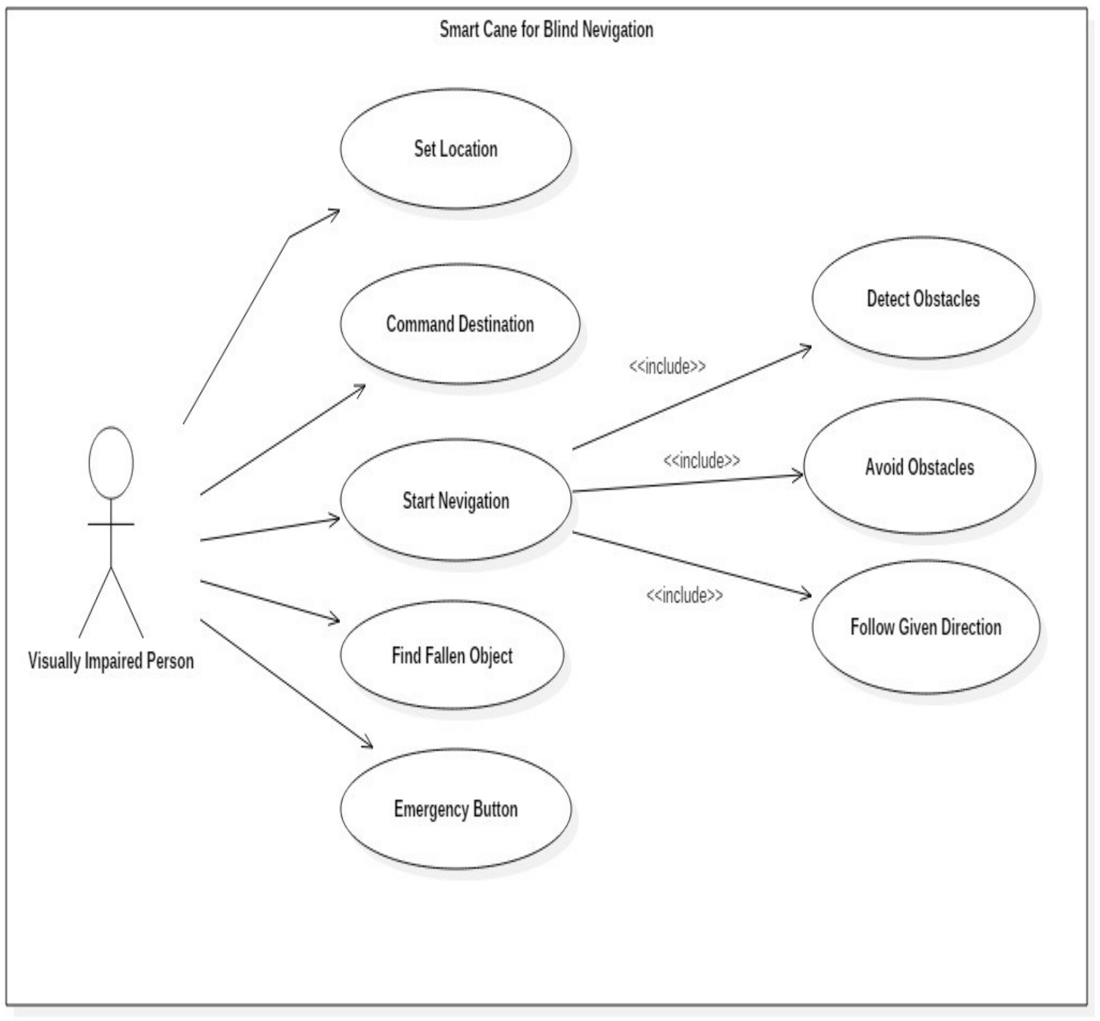


Fig. 4.5: UML Diagram

CHAPTER 5
PROJECT PLAN

5.1 Project Estimate

5.1.1 Reconciled Estimates

Sr. No	Software/Hardware	Quantity	Price
1	Raspberry Pi	1	Rs 3550
2	Lidar Sensor	1	Rs 3345
3	Servo motors	1	Rs 370
4	DC geared motor	1	Rs 150
5	3D printing	1	Rs 4000
6	Anaconda	1	Open source
7	Google cloud platform API	1	Rs 500
8	Android Studio	1	Open source

Table 5.1 Reconciled Estimates

5.1.2 Project Resources

Hardware resources required

- Intel Core i7 with 16GB RAM
- Nvidia GPU
- Android device
- Raspberry Pi

Software resources required

- Operating system: Windows 10
- Coding language: Python, Java, XML
- IDE: Android Studio

- Tools: Anaconda, Unity

5.2 Team Organization

5.2.1 Team Structure

- Android developer: Responsible for developing android application for receiving voice commands and interfacing with the Raspberry Pi.

- Database engineer: Responsible for building the image dataset for training the model.

- Machine Learning Engineer: Responsible for developing and building a object detection module using the dataset built. Design and train a object avoidance module using Unity and reinforcement learning.

- Robotics Engineer: Responsible for selecting components, connecting them and interfacing them with Raspberry Pi

5.2.2 Management Reporting and Communication

Activity	**Week**	**Deliverables**
Design Phase	4	Hardware design and 3d prints
Patent Filing	1	Patent Pending
First Iteration	5	Object Detection
Second Iteration	8	Obstacle Avoidance
Third Iteration	2	Android Development
Fourth Iteration	1	Hardware Interface
Testing and documentation	2	Project report

Table:5.2 Management and Communication

CHAPTER 6
PROJECT IMPLEMENTATION

6.1 Overview of Project Modules

There are following basic modules are present in the system.

- The Navigation application using Google Maps API
- Object and Pedestrian Detection module built using tensor flow.
- Collision avoidance module using Reinforcement Learning.
- Hardware module

6.1.1 Navigation Application:

This is the android application through which user can interact with the system. As name suggest this application is basically built for getting the destination location from the user. So user just need to give the destination location in form of voice command.

System can understand these voice commands and also user able to check whether system understand input correctly, if yes then user proceed to navigation.

During the navigation system can interact with user by providing voice output, like if there is left turn during the navigation then system gives output as ‘Turn Left’ in form of voice to the user.

6.1.2 Object Detection module

The object detection module uses the Tensorflow Object detection API. For this application we collected images of pedestrians, vehicles, zebra crossings and potholes. We drew a bounding box and labelled them. XML files were created that were converted to CSV format using a python script.
A pbtxt file was created that contained all the labels for the object detection model.
Anaconda environment was setup using CUDA and Tensorflow for training and testing the model.

6.1.2 Obstacle Avoidance Module

The object avoidance module is built using Deep Q-Learning algorithm. An environment was built in PyGame in python. Static and dynamic moving objects were added to it. An agent was designed with 3 virtual distance sensors. The neural network was designed and built for reinforcement learning and was run on the environment. The model was trained for 30 hours tweaking the learning rate, fitness scores and rewards. The model learned to avoid colliding with objects. The model was then evaluated and exported.

6.2 Tools and Technology Used

6.2.1 Android Studio:

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development.

Features Provided by Android Studio:

- Android-specific refactoring and quick fixes.
- Gradle based build support
- Lint tools to catch performance, usability, version compatibility and other problems
- ProGuard integration and app-signing capabilities
- Template-based wizards to create common Android designs and components
- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations.

We uses the Android Studio for the development of mobile application which is used by the visually impaired peoples. So user only need to give the destination location through the voice commands and application understand the user input using Google's Speech-To-Text API.

The architecture of android is as follows:

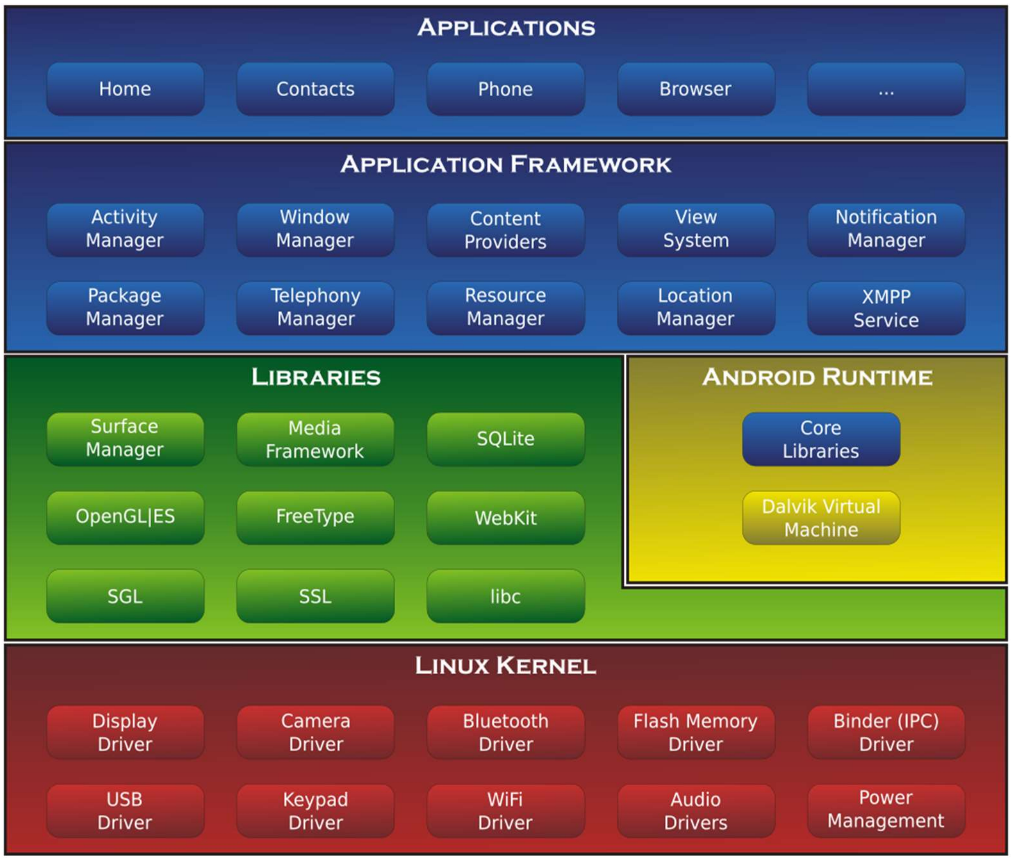


Fig.6.1 Android Architecture

Now-a-days Visually impaired peoples can also uses the smartphones. The major brands I Apple, Samsung, Google etc. each have built-in functions allowing us to use the screen by touch. Many smartphones new support gesture based operability. It requires a lot of practice to learn how to use it as it requires you learn specific "gestures" to accomplish each task. Gestures include combinations of tapping and swiping the screen (vertically as well as horizontally) with 1,2 or 3 fingers. I am "good" at it, experts can navigate their phones better than most sighted users.

Also many smartphones are available with preloaded Virtual Assistant example like 'Google Assistant'. Through this user can directly communicate with it and it provide the result.

6.2.2 Google Maps Platform

The Google Maps Platform also called as Google Maps API which is used for the embedding of Google Maps onto web pages of outside developers, using a simple JavaScript interface or a Flash interface. It is designed to work on both mobile devices as well as traditional desktop browser applications. The API includes language localization for over 50 languages, region localization and geocoding, and has mechanisms for enterprise developers who want to utilize the Google Maps API within an intranet. Google Maps API Premier customers can access the API HTTP services over a secure (HTTPS) connection.

Google Map API is not only a single API but it is a combination of different API which provides the different services offered by Google. Following are the some API's provided by Google Maps.

- Direction API
- Distance Matrix API
- Geolocation API
- Maps Static API
- Places API
- Roads API
- Street View Static API
- Time Zone API

From this list of API's we used Direction API, Maps Static API, Places API which is important for our System.

- Direction API: This API is used to provide the Navigation or Routes feature from source to destination.
- Maps Static API: Provide the Static map view of the World.

- Places API: Provide the details of different places on the map.

Although all these services provided by Google are not free of cost, but it was free for particular number of requests, means if you have account on Google Maps Platform then you can request for these services for specific number of times, after it you need to provide the amount for each request.

API services are accessed using key send with the HTTP request to the maps server. This API key is about 39 character long hash value used to check that requested user is authorized for getting particular services.

following is the example of request to Google Maps:

“https://maps.googleapis.com/maps/api/directions/json?
origin=<source_location>&destination=<destination_location>&key=<your_api_key
>”

6.2.3 Object Detection

6.2.3.1 CUDA

CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

6.2.3.2 Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS.

6.2.3.3 Tensorflow

TensorFlow is an open source library for fast numerical computing. It was created and is maintained by Google and released under the Apache 2.0 open source license. The API is nominally for the Python programming language, although there is access to the underlying C++ API. Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems. It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machines.

6.2.3.4 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc

LabelImg

LabelIng is a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. Annotations are saved as XML files in PASCAL VOC format, the format used by ImageNet. Besides, it also supports YOLO format.

6.2.4 Algorithm details

Algorithm used: Faster RCNN

Faster RCNN is the modified version of Fast RCNN. The major difference between them is that Fast RCNN uses selective search for generating Regions of Interest, while Faster RCNN uses “Region Proposal Network”, aka RPN. RPN takes image feature maps as an input and generates a set of object proposals, each with an objectness score as output.

The below steps are typically followed in a Faster RCNN approach:

1. We take an image as input and pass it to the ConvNet which returns the feature map for that image.
2. Region proposal network is applied on these feature maps. This returns the object proposals along with their objectness score.
3. A RoI pooling layer is applied on these proposals to bring down all the proposals to the same size.
4. Finally, the proposals are passed to a fully connected layer which has a softmax layer and a linear regression layer at its top, to classify and output the bounding boxes for objects.

To begin with, Faster RCNN takes the feature maps from CNN and passes them on to the Region Proposal Network. RPN uses a sliding window over these feature maps, and at each window, it generates k Anchor boxes of different shapes and sizes:

Anchor boxes are fixed sized boundary boxes that are placed throughout the image and have different shapes and sizes. For each anchor, RPN predicts two things:

- The first is the probability that an anchor is an object (it does not consider which class the object belongs to)
- Second is the bounding box regressor for adjusting the anchors to better fit the object

We now have bounding boxes of different shapes and sizes which are passed on to the RoI pooling layer. Now it might be possible that after the RPN step, there are proposals with no classes assigned to them. We can take each proposal and crop it so that each proposal contains an object. This is what the RoI pooling layer does. It extracts fixed sized feature maps for each anchor

Then these feature maps are passed to a fully connected layer which has a softmax and a linear regression layer. It finally classifies the object and predicts the bounding boxes for the identified objects.

6.3 Obstacle Avoidance module

This module is basically training and testing of our AI agent which is trained on 2-d simulation of a dynamic environment using Deep Q-learning.The agent automatically moves itself forward, faster and faster as the game progresses. If it runs into a wall or an obstacle, the session ends.There are three available actions at each frame: turn left, turn right, do nothing.At every frame, the game returns both a state and a reward.The state is a 1d array of sensor values, which can be 0, 1 or 2, as stated above.The reward is -500 if the car runs into something and 30 minus the sum of the sensor values if it doesn't. The concept here is that the lower the sum of the sensors, the further away it is from running into something, and so we reward that. The -500 is a big punishment.

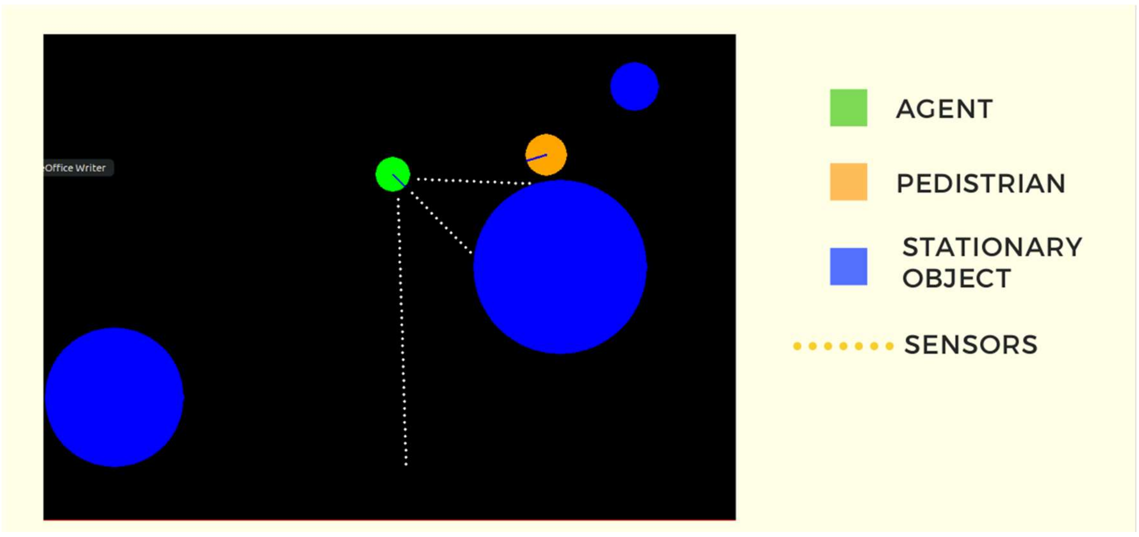


Fig. 6.2: 2-D Simulation Environment

The neural network

We use a relatively simple fully connected neural network as our model. We have three dense layers: input, a single hidden layer and output. We add a rectified linear units after the input and hidden layer to speed training, and also add a 0.2 dropout to prevent over-fitting. Given this isn't a very complex problem various models are trained and tested by changing density of hidden nodes, dropout value buffer size etc.

Using loss for hyper parameter tuning

After calculating loss value from our trained set we tune parameters by setting following test values.

- **Number of hidden neurons per layer**: 20x20, 164x150, 256x256, 512x512, 1000x1000
- **Batch size**: 32, 40, 100, 400
- **Buffer** (experience replay): 10,000, 50,000, 500,000
- **Gamma**: 0.9, 0.95

6.3.2 Algorithm Details

Deep-Q Learning Algorithm:

Q-learning, like virtually all RL methods, is one type of algorithm used to calculate state-action values. It falls under the class of *temporal difference* (TD) algorithms, which suggests that time differences between actions taken and rewards received are involved. Following Formula shows how Q- values are calculated:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max Q(S_{t+1}, a_{t+1}) - Q(S_t, A_t)]$$

We can use a neural network, instead of a lookup table, as our Q(s,a) function. Just like before, it will accept a state and an action and spit out the value of that state-action. Importantly, however, unlike a lookup table, a neural network also has a bunch of parameters associated with it. These are the weights. So our Q function actually looks like this:

Q(s,a,θ) where θ is a vector of parameters. And instead of iteratively updating values in a table, we will iteratively update the θ parameters of our neural network so that it learns to provide us with better estimates of state-action values.Of course we can use gradient descent (backpropagation) to train our Q neural network just like any other neural network.But what's our target y vector (expected output vector)? Since the net is not a table, we don't use the formula shown above, our target is simply: rt+1+γ□maxQ(s′,a′)rt+1+γ□maxQ(s′,a′) for the state-action that just happened. γ is a parameter 0→1 that is called the *discount factor*. Basically it determines how much each future reward is taken into consideration for updating our Q-value. If γ is close to 0, we heavily discount future rewards and thus mostly care about immediate rewards.s′ refers to the new state after having taken action a and a′ refers to the next actions possible in this new state. So maxQ(s′,a′) maxQ(s′,a′) means we calculate all the Q-values for each state-action pair in the new state, and take the maximum value to use in our new value update. t1 except when the state s′ is a terminal state. When we've reached a terminal state, the reward update is simply rt+1 A terminal state is the last state in an episode. In our case, there are 2 terminal states: the state where the player fell into the pit (and receives -10) and the state where the player has reached the goal (and receives +10). Any other state is non-terminal and the game is still in progress.

There are two keywords I need to mention as well: **on-policy** and **off-policy** methods. In on-policy methods we iteratively learn about state values at the same time that we improve our policy. In other words, the updates to our state values depend on the policy. In contrast, off-policy methods do not depend on the policy to update the value function. Q-learning is an off-policy method. It's advantageous because with off-policy methods, we can follow one policy while learning about another. For example, with Q-learning, we could always take completely random actions and yet we would still learn about another policy function of taking the best actions in every state. If there's ever a π referenced in the value update part of the algorithm then it's an on-policy method.

1. Start a new Session and move the car forward one frame without turning.
2. Get a reading of the sensors.

3. Based on those readings, predict Q values. These predictions show Agent's confidence that it should take each of the three actions listed above. The first time through, these will be worthless, but we have to start somewhere.
4. Generate a random number. If it's less than our epsilon (see below), choose a random action. If it's higher than our epsilon, choose the most confident action returned from our prediction.
5. Execute the action (left, right, nothing) and get another sensor reading and our reward.
6. We store these things—the original reading, the action we took, the reward we got and the new reading—in an array that we call a buffer.
7. Grab a random sample of "reading, action, reward, new reading" from our buffer and learn by building an X, y training set that we "fit" our model to.
8. We set the y value for the iteration to a prediction based on the original reading.
9. We make a new prediction based on our new reading (post-action state).
10. We take a look at the reward we were given by taking the action. If it's -500, we've run into something, and so we set the y for this iteration and this action to -500. If we didn't run into anything, we multiply our max predicted Q value by a gamma (to discount it) and set the iteration y value for the action we took.
11. Go back to step 2 until we run into something.
12. When we run into something, decrease our epsilon and go back to step

6.4 Hardware module

A 3D model was designed in Fusion 360 by taking specifications of the components. The model was then exported for printing using the STL format. The material used to print the 3D model is ABS, it has great strength and gives high quality prints.

Rest of the components were fixed on the stick for complete implementation

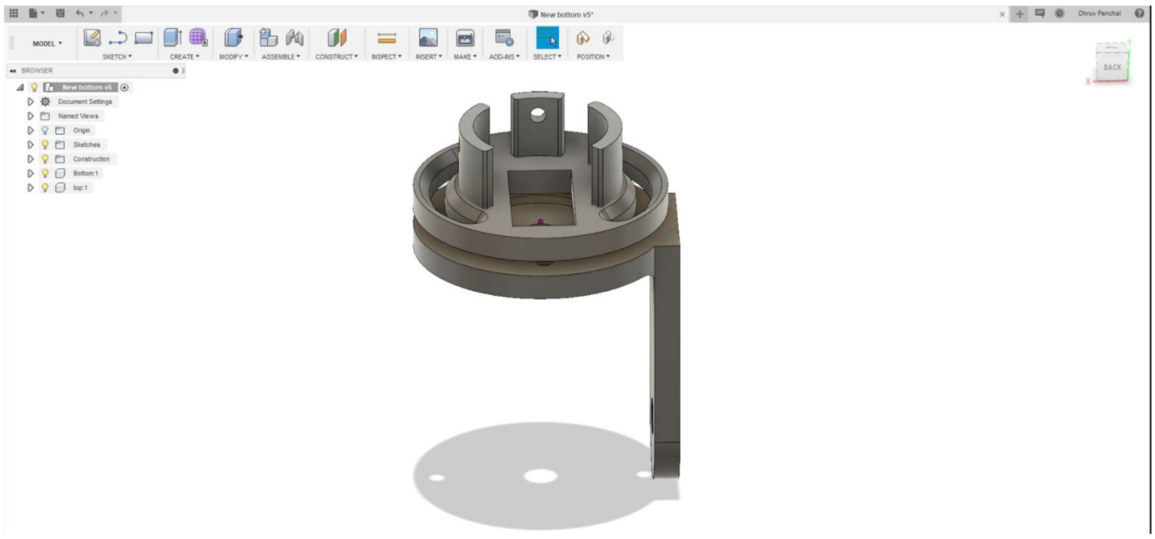


Fig. 6.3: 3D Printed Hardware

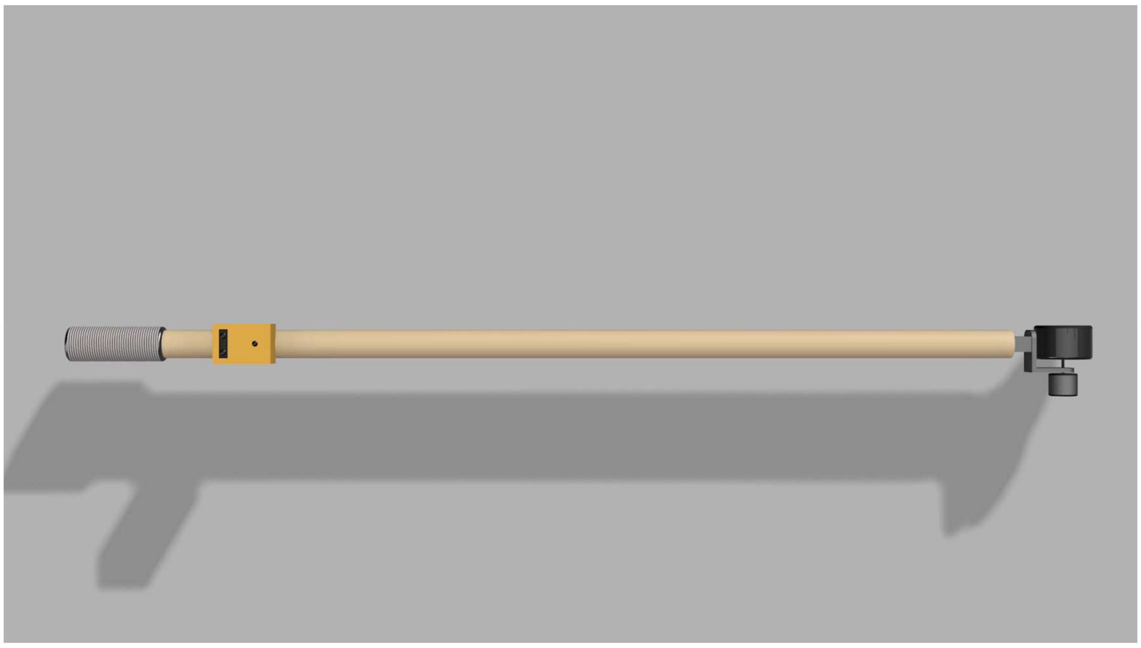


Fig. 6.4: side view

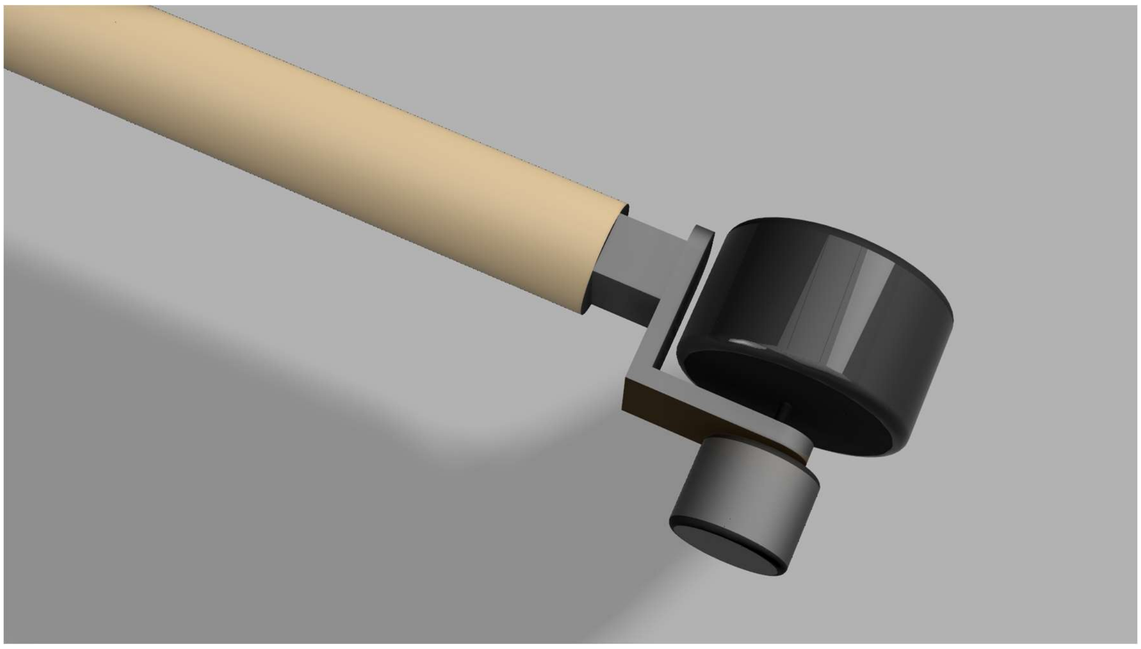


Fig. 6.5: wheel assembly

CHAPTER 7
SOFTWARE TESTING

7.1 Model testing and validation

Following parameters are used for testing and loss values are plotted :

- Number of hidden neurons per layer: 20x20, 164x150, 256x256, 512x512, 1000x1000
- Batch size: 32, 40, 100, 400
- Buffer (experience replay): 10,000, 50,000, 500,000
- Gamma: 0.9, 0.95

7.2 Type of testing

Manual Testing:

Step-1: this is the first step of verification o neural network . Hopefully you're seeing such a trending line. However, feeding data into system, replaying the training cycles and you're not seeing the "training error" line go down, then your neural network is struggling to find a pattern.

It is possible there isn't a relationship that can be mapped between your inputs and your outputs. Although as we talked about last time, you can review your neural network, the quality of your data and the size of the data pool. All this is painstaking, because alas neural networks are not a quick and easy solution.

Step-2*:* is to test it a bit more. Now for this you can't simply use data that you've already used for training—the neural network has learned to cope with that explicit case (seen in the training error graph). This means you need some additional data around to do this—as a rule of thumb, when I did neural networks for other people, it was about 25% of the data you'd used for training.

Typical results from this stage of verification can be seen above as the “test error” line, and notice how they always have a higher error rate than “training error” data. That shouldn’t be unexpected—the neural network trained on explicit data provided and re-verified in the “training error” data line

So far we’ve done most of this checking automatically. Looking at such a neural network or any other machine learning system, we can feel lost on what to test.

That feeling of being at a loss is fundamentally because as testers we often fall back on testing boundaries, especially for business rules. As we discussed last time a fundamental thing at issue with a neural network is there are no hard and fast boundaries, there’s a pattern. A pattern we can’t see—it’s inside the neural network, so we have to explore it.

There are a few patterns I learned from James Bach’s Rapid Software Testing course which I think apply here. What you need to do though is start by mapping out the data which has been used to date. Then why not try …

Initial exploration. You can start out by just taking a few data samples from your training and test data and running them through your neural network system to “get a feel”.

Scenario tests. Choose a few items of data which are far from where any decision boundaries should be, and see how it behaves. You might have an expert around who can provide you a simple example of “in this scenario, no questions asked, X should happen”.

7.2 Test cases and Test Results

Scenario #1: The biggest, slowest, most complex networks look good but will require a lot more training. (1000 x 1000 hidden neurons, 400 batch size with 10,000 replay experience had the lowest loss.)

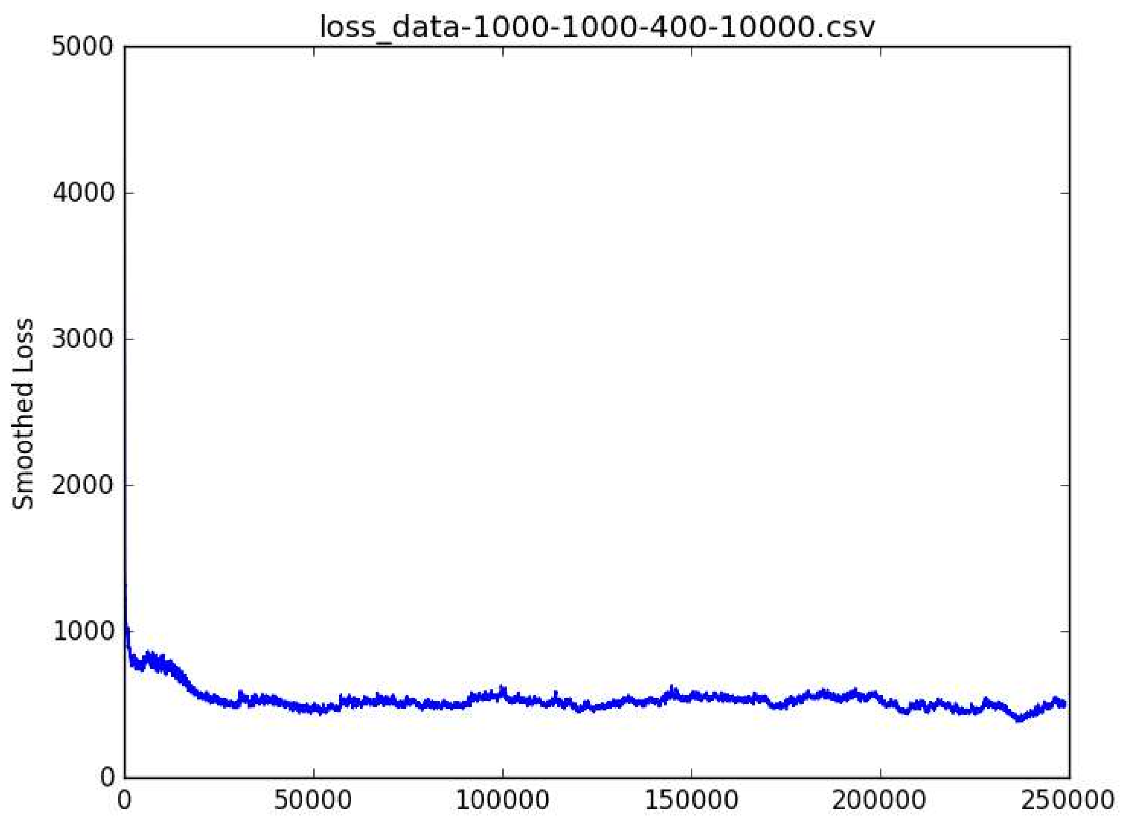


Fig. 7.1: Test Scenario 1

Conclusion: Very low loss, low variance, but also a flat learning rate

Scenario #2: 512 x 512 at 400 and 50,000 looks like a good tradeoff since it's a little faster and appears to be learning quickly and smoothly

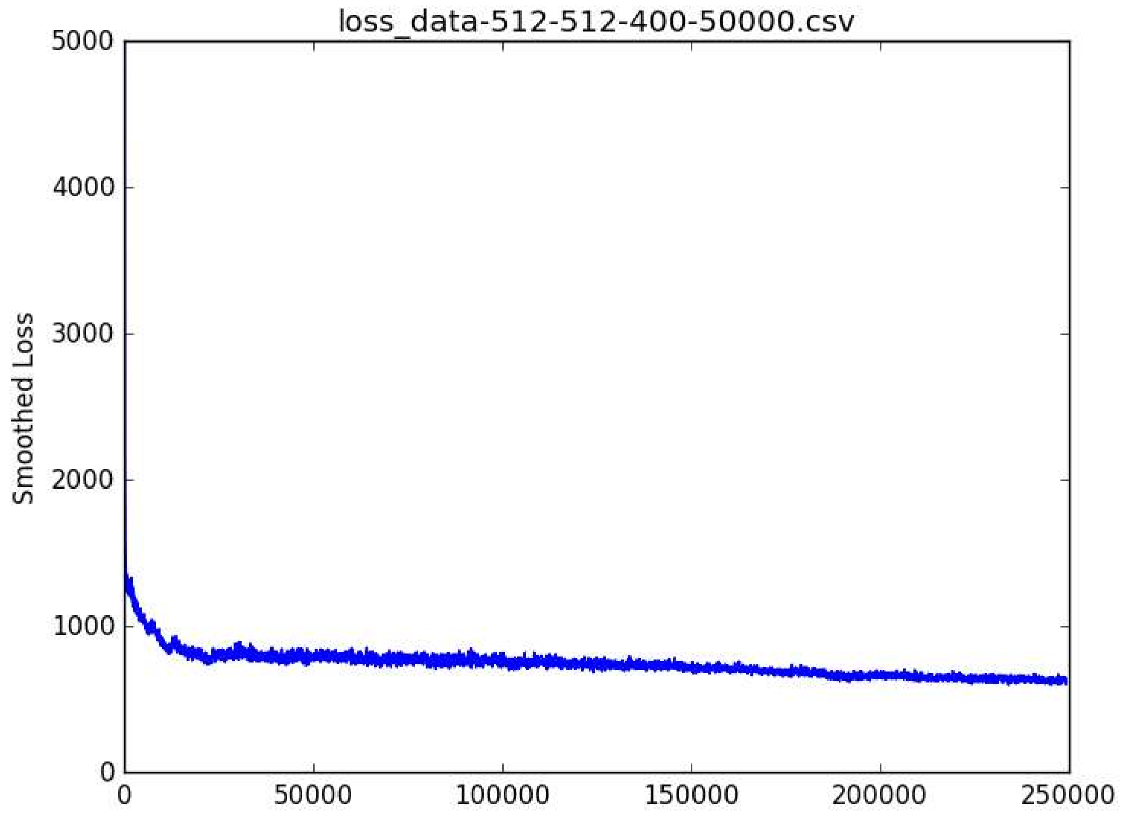


Fig. 7.2: Test Scenario 2

Conclusion: Low loss, low variance, continuous (slow) learning. Looks promising.

Scenario #3: Interesting curveball: Hidden layers of 164 x 150 neurons with 400 batch size and 50,000 buffer may be the best compromise of them all (see graphs below).

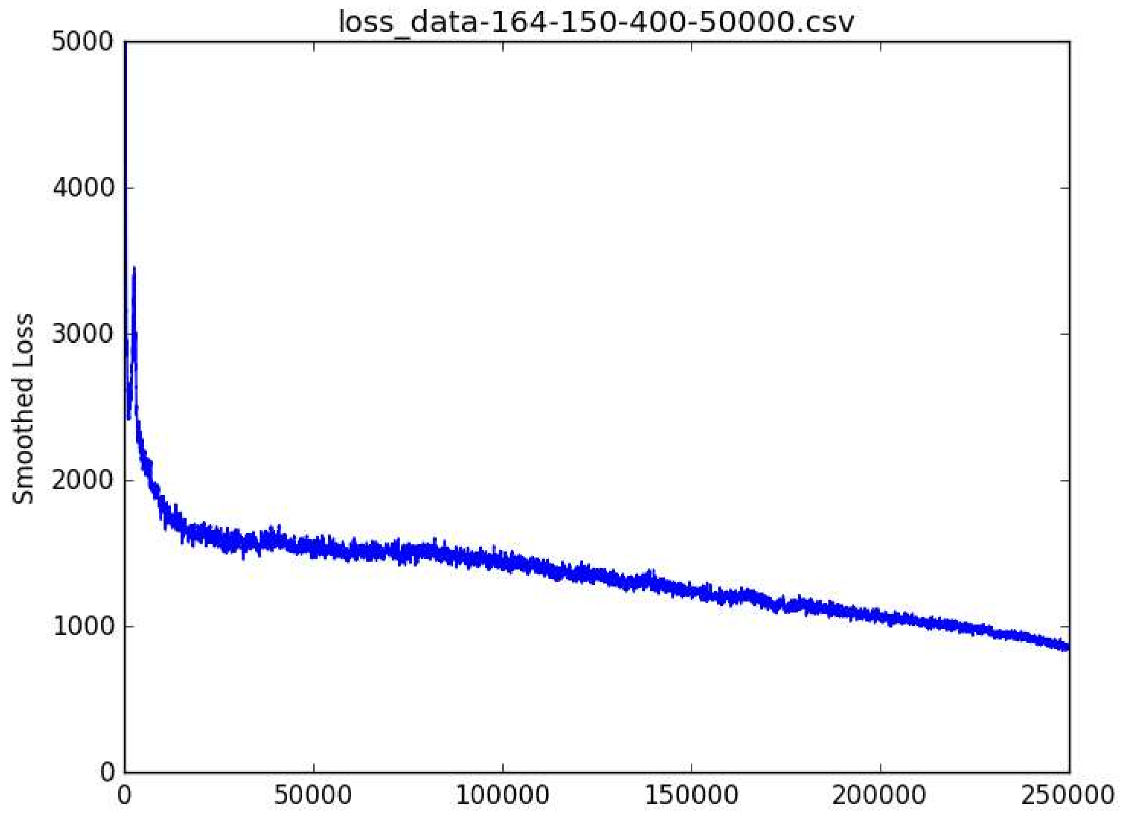


Fig. 7.3: Test Scenario 3

Conclusion: This is perhaps most promising due to high slope. It has a higher absolute loss than numbers 1 and 2 but a good learning rate that shows good potential and it’s much faster to learn than the larger networks.

CHAPTER 8
RESULTS

8.1 Outcomes

This project aimed to develop a solution for enabling the vision impaired to lessen the efforts and help needed for traveling. The process of planning, implementing, testing, validation and iteration led to completion of this project. Each stage of development came with challenges and strict deadlines. Following the planned workflow and process model resulted in being to deliver the product in planned time and capacity. Modules developed for implementing this project can be individually used as standalone software products or can be integrated into other products for enhancement of its capabilities.

8.2 Screenshot

UI

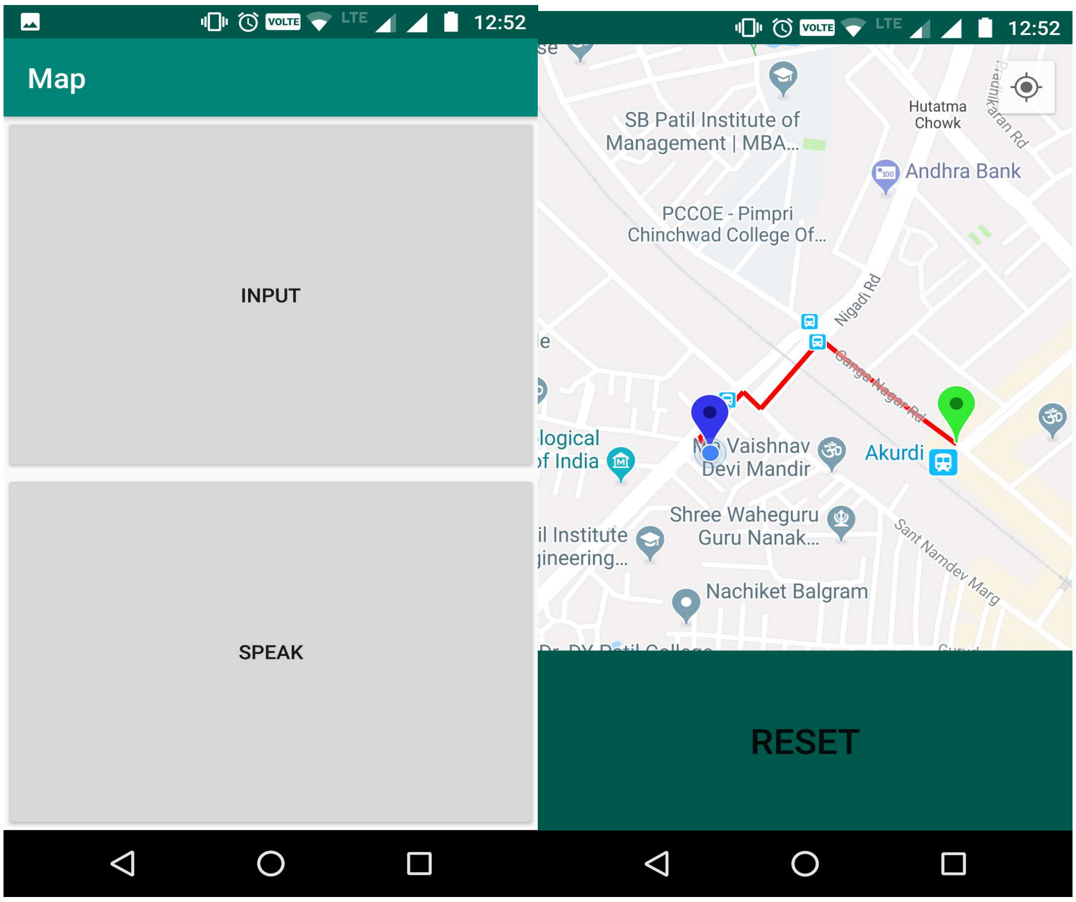


Fig. 8.1: UI

Object Detection

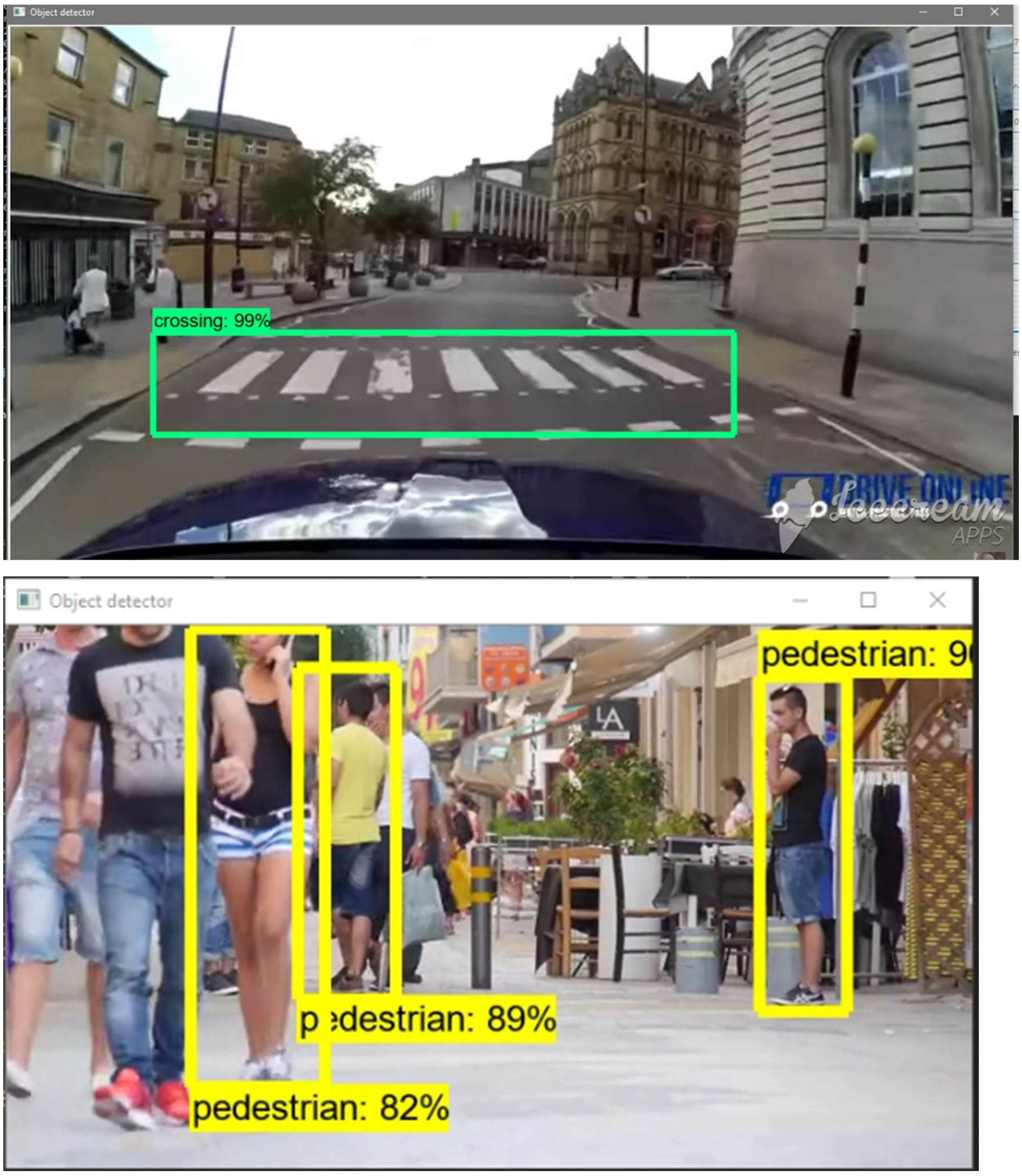


Fig. 8.2: Object Detection

Obstacle avoidance

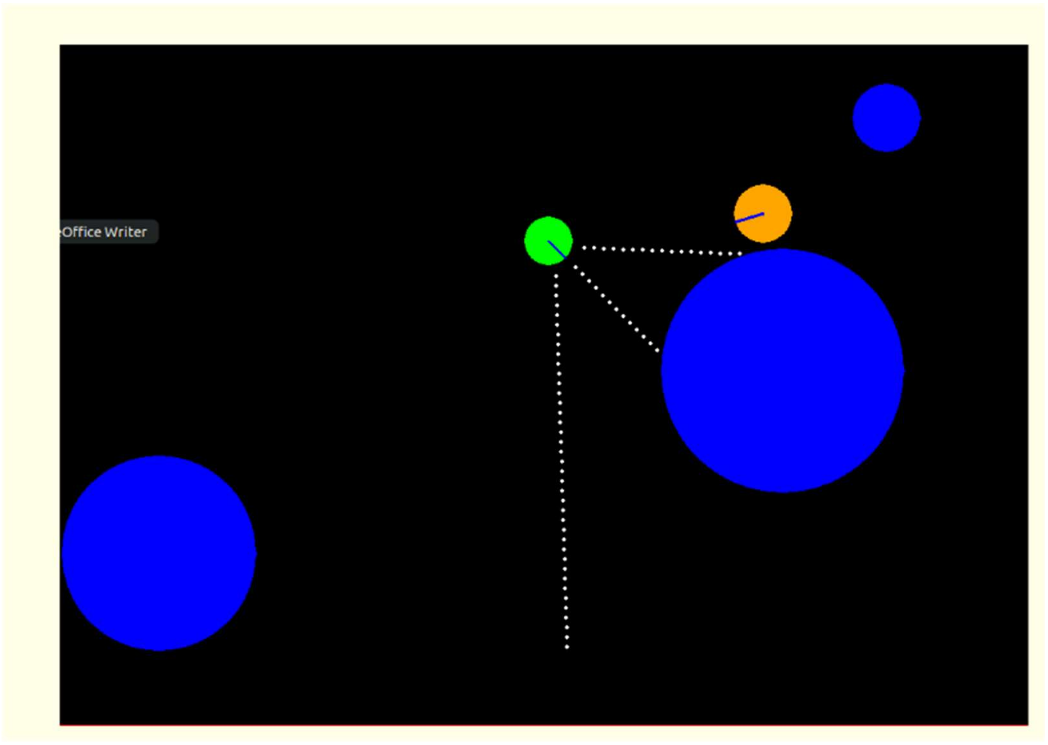


Fig. 8.3: Obstacle Avoidance

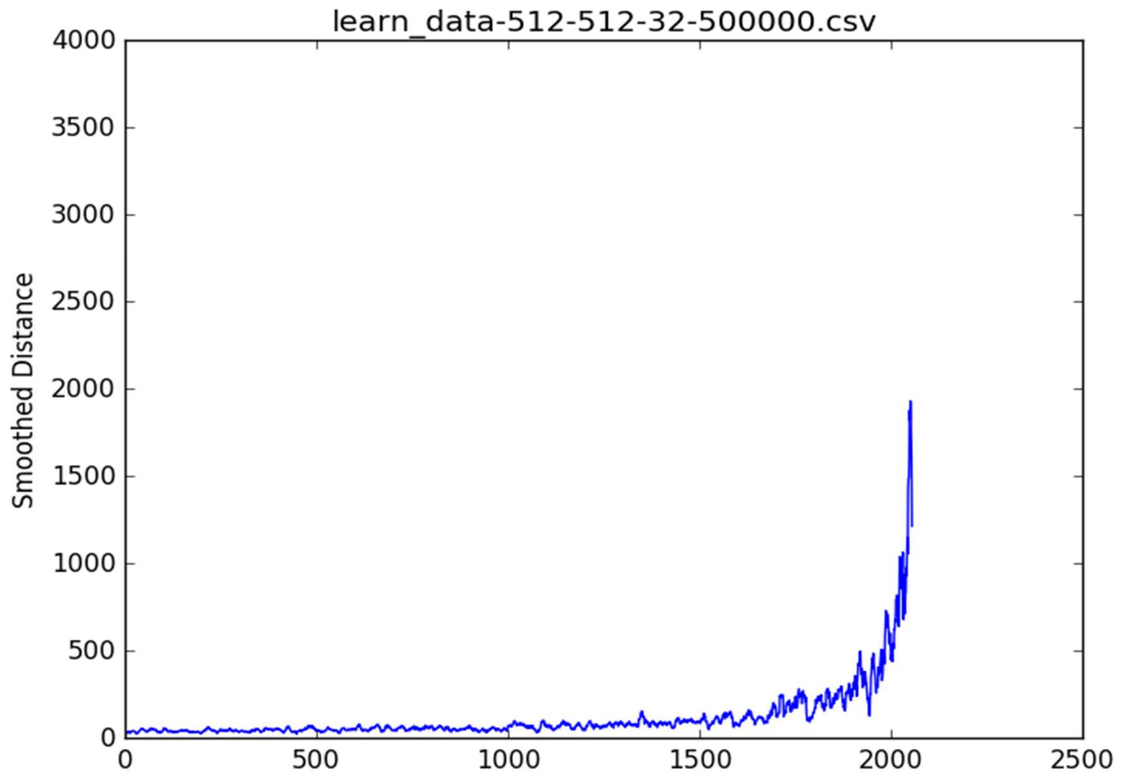


Fig. 8.4: Learning Graph

CHAPTER 9
CONCLUSIONS

9.1 Conclusion

Blindness may result from a disease, injury or other conditions that limit vision and because of which Blind people confront a number of challenges everyday. All too frequently, blindness affects a person's ability to self navigate outside well known environments and even simply walking down a crowded street. To provide a practical solution to this problem this Ai based robotic cane prototype is made which will ease the life of visually impaired.

9.2 Future Work

The smart AI based cane for blind navigation provide state of the art solution for the visually impaired people in terms of practicality and usability it works similar to guide dog with no maintenance and can be used in day to day life. The cane is made to be affordable and commercially viable and many future work such as finding of fallen or lost things on ground, emergency help special smart phone compatibility will be added .in the future. the prototype will also adapt accordingly in the testing phases and will improve with further updates.

9.3 Applications

1.Help blind people to navigate from one place to another place.

2. Provide safe navigation to the user

APPENDIX A

NP-Complete problem:

- An NP-complete decision problem is one belonging to both the NP and the NP-hard complexity classes.any given solution to an NP-complete problem can be verified quickly (in polynomial time), there is no known efficient way to locate a solution in the first place;
- In our project Image capturing and object Detection can be done in polynomial time so it is considered as NP-Complete problem.

NP-Hard problem:

- A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem. NP-hard therefore means "at least as hard as any NP-problem," although it might,in fact, be harder.
- Identification and choosing the correct path from source to user specified destination at real time considered as NP-Hard problem, which is used in our project.

APPENDIX B

[1] Ayat Nada , Samia Mashaly, Mahmoud A. Fakhr, Ahmed F. Seddik,”Human and Car Detection System for Blind People”,2018

[2] Michal Kochláň ; Michal Hodoň ; Lukáš Čechovič ; Ján Kapitulík ; Matúš Jurečka,”WSN for traffic monitoring using Raspberry Pi board”,2014

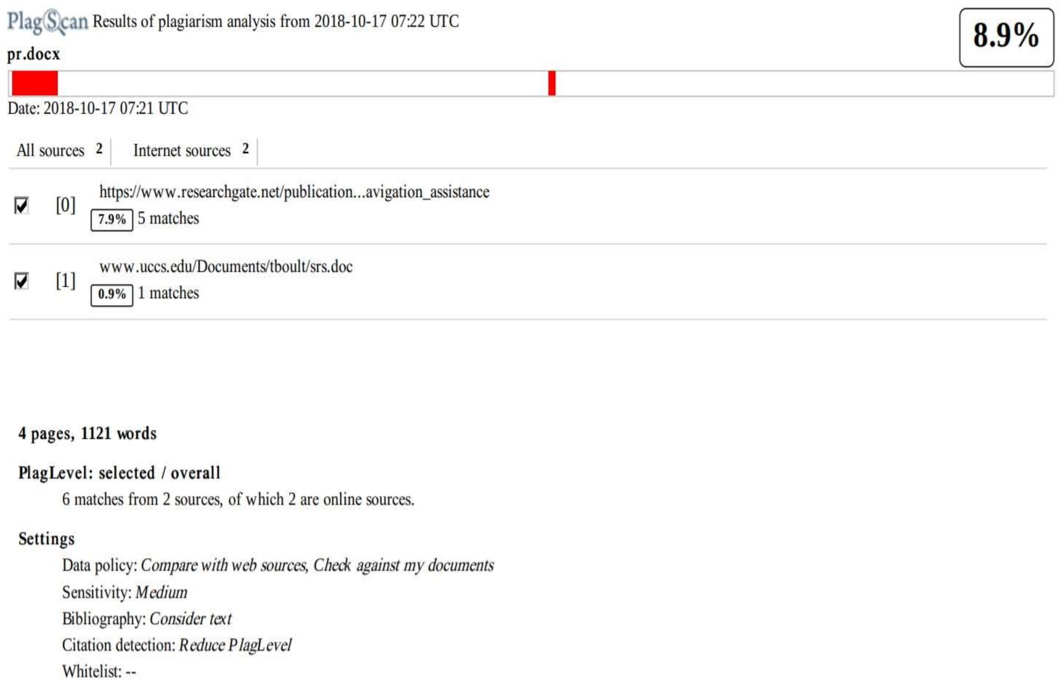
[3] Bhumika Gupta, Ashish Chaube, Ashish Negi, Umang Goel,”Study on Object Detection using Open-CV - Python”,2013

[4] Richard.R.Sutton, Andrew G Barto,”Reinforcement Learning: An Introduction”,1998

[5] Ben J.A. Krose and Joris W.M. van Dam,”Adaptive state space quantisation for reinforcement learning of collision-free navigation”,1992

APPENDIX C

Plagiarism Report: website:www.PlagScan.com



PlagScan Results of plagiarism analysis from 2018-10-17 07:22 UTC

pr.docx

8.9%

Date: 2018-10-17 07:21 UTC

All sources 2 | Internet sources 2

✓	[0]	https://www.researchgate.net/publication...avigation_assistance 7.9% 5 matches
✓	[1]	www.uccs.edu/Documents/tboult/srs.doc 0.9% 1 matches

4 pages, 1121 words

PlagLevel: selected / overall

6 matches from 2 sources, of which 2 are online sources.

Settings

Data policy: *Compare with web sources, Check against my documents*

Sensitivity: *Medium*

Bibliography: *Consider text*

Citation detection: *Reduce PlagLevel*

Whitelist: --

REFERENCES

[1] Ayat Nada , Samia Mashaly, Mahmoud A. Fakhr, Ahmed F. Seddik,”Human and Car Detection System for Blind People”,2018

[2] Michal Kochláň ; Michal Hodoň ; Lukáš Čechovič ; Ján Kapitulík ; Matúš Jurečka,”WSN for traffic monitoring using Raspberry Pi board”,2014

[3] Bhumika Gupta, Ashish Chaube, Ashish Negi, Umang Goel,”Study on Object Detection using Open-CV - Python”,2013

[4] Richard.R.Sutton, Andrew G Barto,”Reinforcement Learning: An Introduction”,1998

[5] Ben J.A. Krose and Joris W.M. van Dam,”Adaptive state space quantisation for reinforcement learning of collision-free navigation”,1992

[6] Rachid Sammouda ; Ahmad Alrjoub, “Mobile blind navigation system using RFID, Global Summit on Computer & Information Technology (GSCIT)”, 2015

[7] Sakmongkon Chumkamon ,Peranitti Tuvaphanthaphiphat ,”A Blind Navigation System Using RFID for Indoor Environments ”, 5th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008

[8] Curing retina Blindness foundation, “Tools of the blind and visually impaired”

[9] Carol Evans, “Psychological testing of blind, and visually impaired children”

[10] Mary K. Bauman, Carol A. Kropf, “Psychological tests used with blind and visually handicapped persons”, January 1979