Chapter 1

INTRODUCTION

Autonomous vision based robots are intelligent robots which take visual data, process it and provide appropriate output. A robot is a machine that can perform task automatically or with guidance. Here, a robot is designed on Raspberry Pi using OpenCV, which is used for human detection.

1.1 Background and basics

We are living in a modern era where technology has given utmost priority, this makes our life easier. Since safety is the major priority, the safety of house also becomes important this can be done by deploying a robot with human detection capability. Robotics is a combination of computational intelligence and physical machines (motors). Computational intelligence involves the programmed instructions.

Vision is one of the most important senses to a human being and in the past decade there has been an increased interest to use images in robotics. Machines may lack the vast knowledge of object recognition that a human brain can provide but the amount of computational power available in modern times make such machine vision a viable choice of input, and unlike a human eye, machine vision does not degrade over time providing consistent image capture. Images from a colored camera are used in this project as the source of information for processing.

Computer vision is becoming important and with that human detection for applications like video surveillance, autonomous driving vehicles, person recognition have also become important. Human Detection is challenging because everyone is different in appearances and there are wide range of poses.

1.2 Literature Survey

Table 1.1 Summary of literature survey

Title	Published Year	Basic Idea
Human Detection using HOG-SVM, Mixture of Gaussian and Background Contours Subtraction	2017	• Human detection using HOG-SVM algorithms
Human Detection and Tracking for Video Surveillance: A Cognitive Science Approach	2017	• Detect human beings in any frame. • Find the movement patterns of the humans in the frame.
Obstacle Detection and Avoidance Robot	2018	• Detect and avoid any obstacles in the path. • Focuses on the edge detection for higher accuracy.
Design and Implementation of an Autonomous Indoor Surveillance Robot based on Raspberry Pi	2019	• Identify the person and alert the owner. • Take necessary actions through the commands given by the owner.
Real-Time Human Motion Detection, Tracking and Activity Recognition with Skeletal Model	2020	• Creates human skeletal system based on deep neural networks.

1.3 Project Undertaken

1.3.1 Problem definition

Make a surveillance robot to identify the person, follow him and alert the user. Then, according to the instructions received by user the robot will:

- Send video to the user through telegram
- Stop following if the person is known.
- Return to its original position.

1.3.2 Scope statement

- This project is undertaken to ensure safety of the house or premise of the user.
- The project involves surveillance of a premises, in a limited area for a long period of time. It involves communication of robot with the user through telegram.
- To implement it, there must be successful identification of the person, it's tracking by the robot and transfer of commands from user to robot for further course of action.
- Though it is suitable for indoor and outdoor surveillance it is advised to use it in a closed outdoor premise. Also multiple entities are recommended if the area is bigger than the range of camera.
- While developing this project it was kept under assumption that the robot is placed at a place from where entry/exit of the premise can be easily overlooked.

1.4 Organization Of Project Report

The rest of the report is sorted in the following manner:
Chapter 2 describes about the Planning and management of Project. Chapter 3 presents and explains Design and analysis of the system. Chapter 4 elaborates implementation and coding part of the project. Chapter 5 elaborates testing of the project. Chapter 6 details result and discussions and conclusion and future work of the project is discussed in chapter 7 and 8 respectively. The list of references and appendices is also provided at the end of the report.

Chapter 2
PROJECT PLANNING AND MANAGEMENT

2.1 Introduction

SRS is considered as the base for the effort estimations and project scheduling

This document details the working plan for “Raspberry pi based Surveillance Robot for Real Time Intrusion Detection and Tracking”. It also specifies it’s requirements, functioning, technology used and an overall description of the project to the reader.

The aim of the project is to create an autonomous robot capable of human detection and surveillance.
The robot must have the ability to connect with the user over telegram app and send photo and video.
It will have ability to follow a person, receive and follow command of user.

2.2 System Requirement Specification (SRS)

2.2.1 System Overview

The product is a stand alone robot which is being developed to ensure security of a house. It is a raspberry pi based module with a camera ultrasonic sensor mounted on it. It is navigable, the wheels are moved by the motors installed which are powered by a battery.

The major functions of the system are:

1) Identification: Identifying whether the object seen in the camera is human or not. If it is human send it’s photo to user on telegram.
2) Following: Follow the person after sending photo.
3) Sending video: send a video as and when instructed by user.
4) Return to original position when instructed.

The system is easy to use implement and run. It requires to be executed only once, it runs till battery becomes low. Anyone with knowledge of using a mobile phone can use it.

2.2.2 Functional Requirements

At the core, Telegram Bots are special accounts that do not require an additional phone number to set up. Users can interact with bots in two ways:

- Send messages and commands to bots by opening a chat with them or by adding them to groups.
- Send requests directly from the input field by typing the bot's @username and a query.

This allows sending content from inline bots directly into any chat, group or channel.
Messages, commands and requests sent by users are passed to the software running on your servers. Our intermediary server handles all encryption and communication with the Telegram API for you. You communicate with this server via a simple HTTPS-interface that offers a simplified version of the Telegram API. We call that interface our Bot API.

2.2.3 Non-Functional Requirements

Performance:
The performance of the system depends highly on accuracy of the sentiment analysis and machine learning algorithms.
Also, the system depends on efficiency of the hardware components.
System is also dependent on the various user’s command for performing its further operations.

Security:
Only the authenticated user should be able to send commands to the system.
The images/videos captured from the system should be sent to the appropriate user only.
Anyone other than the concerned user should not be able to intercept the communication.

2.2.4 Deployment Environment

Hardware Requirements:

Raspberry Pi 3B
Ultrasonic Sensor Module HC-SR04
ROBOT Chassis
Wheels, DC Motors
Raspberry pi 5MP camera
Bread Board and Resistor (1k)
Motor Driver L298 2A
Connecting wires and Power supply or Power bank

Software Requirements:

Windows 10 and Python 3.7

Libraries:

1)Python cv2
2)Telegram
3)Socket
4)Imutils

2.2.5 External Interface Requirements

Telegram serves as the user interface.it is the intermediary between user and raspberry pi.
There is a UDP connection between the raspberry pi and the processing unit(in this case laptop), the processing of the data takes place on the laptop.

2.2.6 Other Requirements

Only the user who is authenticated can handle and use the system. The scaling or extension of system is only done by developers according to the need of users.

2.3 Project Process Modelling

A software process is a set of certain activities that leads to successful implementation of a project. These activities start right from research phase till the testing and deployment phase. It involves development of software from scratch and also modifying the project to inculcate necessary changes.

The project follows an incremental approach of implementation. Using this approach, it is possible to develop a basic working project, and then at each phase of development iteration different functionalities are added. The basic idea behind following this model is to implement the most important functionality first and then add new functions and features step by step in order of their priority. Incremental process model helps to break down the development into smaller, achievable portions, which also makes it easy to recognize and implementation errors before the whole system is developed. Developing the system in incremental steps, makes it easy to test, debug during a smaller iteration.

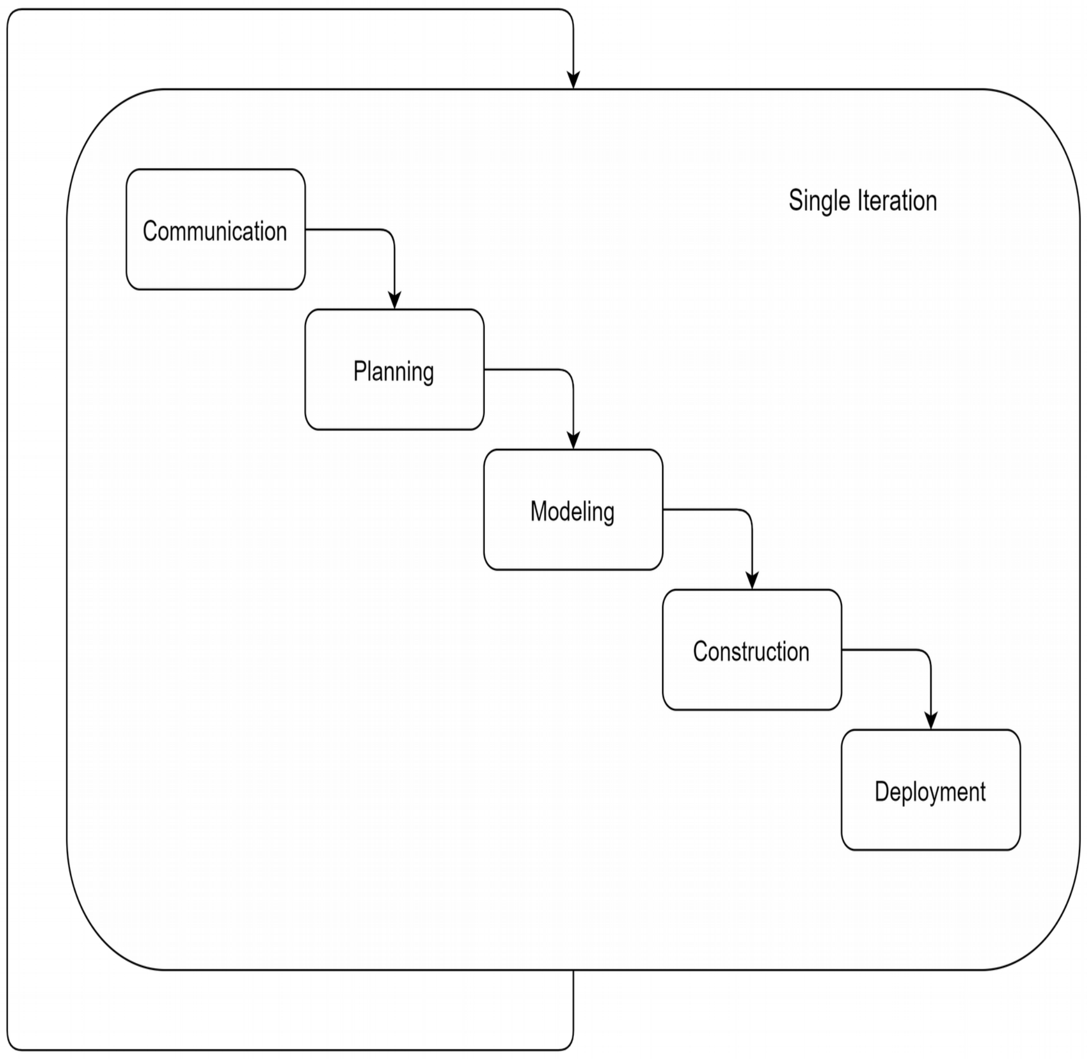

Fig. (2.1) Process Model

The whole process of software development is carried out iteratively, where each of the iteration occurs linearly as waterfall model. Incremental approach provides more flexibility which also reduces the cost of developing the project. The software development occurs in small increments allowing the system to handle changes easily.

2.4 Cost and Efforts Estimates

The constructive cost model is used for software cost estimation. The basic version is used to estimate cost, duration and number of people required. A semi-detached type is considered which specifies that the software project requires moderate level of complexity and team size and need a mix of rigid and no rigid requirements.

The constants required are ab, bb, cb, db where subscript b implies Basic Model.

According to COCOMO Model,

Effort Applied,

$EA = a_b(KLOC)^{bb}$

where,

$a_b = 3$

b = 1.12

KLOC = 6

Therefore,

EA = 3 ∗ (6)1.12

= 22.317

Hence, Effort Applied = 22.317

Development Time (D),

$D = c_b(EA)^{db}$

where, $c_b = 2.5$
db = 0.35
EA = 22.317

Therefore,

$D = 2.5 * (22.317)^{o.35} = 7.412$

Hence, Development Time = 7.412

Staff Size (SS),

SS = EA/D

where,

D = 7.412

EA = 22.317

Therefore,

SS = 22.317/7.412

= 3.0109

Hence, Staff Size = 3.0109

2.5 Project Schedule

2.5.1 Timeline Sem1

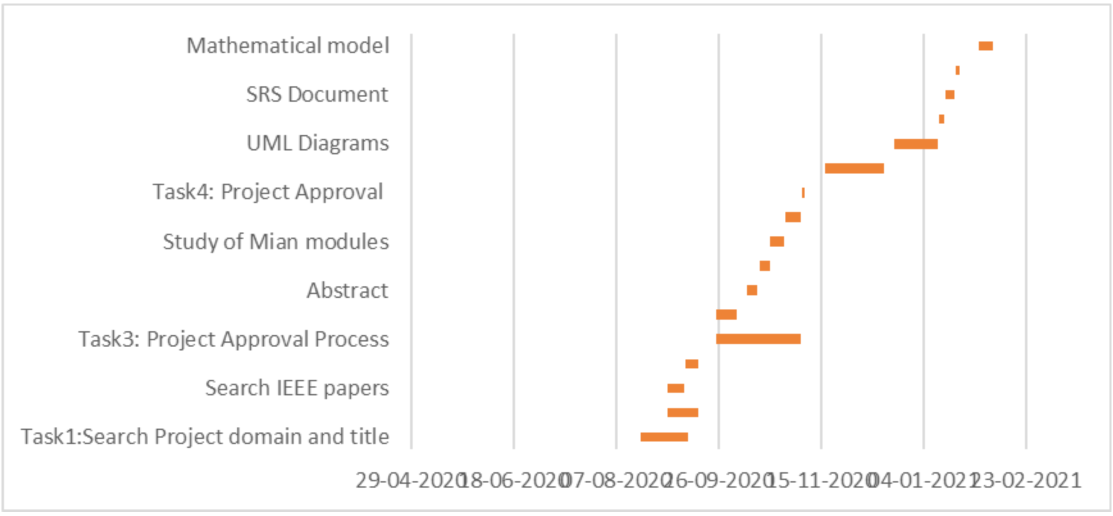

Fig. (2.2) Timeline Sem1

Chapter 3

ANALYSIS & DESIGN

3.1 Introduction

This chapter covers the analysis and design of the considered system. Design of the system is the frame on which the project rests and is then implemented. Analysis of system is as important as design, it gives the overview of the system.

3.2 IDEA Matrix

An IDEA matrix is a concept that evaluate various effects that the idea has. This tells us almost everything about the project.

Table 3.1 IDEA Matrix

IDEA	Deliverables	Parameters Affected
Increase	It increases the security of the house.	Performance
Improve	The communication between user and robot	Integration between human and robot.
Ignore	Stops following if person is known.	Identification
Deliver	Delivers efficient performance by following person.	Tracking and communication
Decrease	Decreases the concern of security of house.	Security
Drive	Facility to send videos, tracking.	Navigation, Identification
Educate	Educate people regarding security	Project Users
Eliminate	Eliminates the risk of intrusion.	Security
Accelerate	Accelerates the communication with user	Communication
Associate	Works with a processing	Processing

	unit to process data.	
Avoid	Avoids latency in communication by using UDP protocol and telegram for communication. Avoids risk of intrusion.	Comunication and Security

3.3 Mathematical Model

3.3.1 Algorithm used:

Single Shot Detection algorithm:

Single shot detection algorithm[1] is an object detection algorithm, here it is used to detect humans. It takes only a single shot to detect multiple object present in a image using multibox. In terms of performance and precision for object detection tasks,the single shot detection algorithm scores 74% mAP (mean Average Precision) at 59 frames per second on standard datasets such as PascalVOC and COCO. To better understand SSD, let’s startby explaining where the name of this architecture comes from:

Single Shot: this means that the tasks of object localization and classification are done in a single forward pass of the network .

MultiBox[2]: this is the name of a technique for bounding box regression developed by Szegedy et al. It is a a method for fast class-agnostic bounding box coordinate proposals.[2]

Detector: The network is an object detector that also classifies those detected objects.

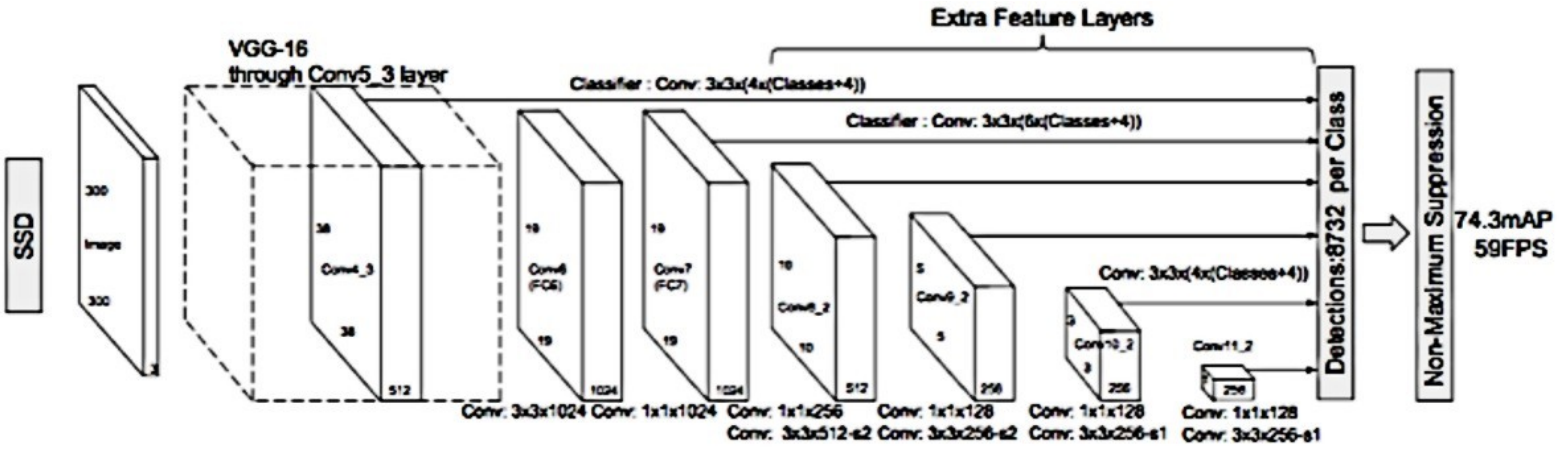

Fig. (3.1) Architecture of single shot detection algorithm

As you can see from the diagram above, SSD's architecture builds on the venerable VGG-16 architecture, but discards the fully connected layers. The reason VGG-16 was used as the base network is because of its strong performance in high quality image classification tasks and its popularity for problems where transfer learning helps in improving results. Instead of the original VGG fully connected layers, a set of auxiliary convolutional layers (from conv6 onwards) were added, thus enabling to extract features at multiple scales and progressively decrease the size of the input to each subsequent layer. SSD applies a classifier to multiple feature maps. If you perform an appropriate conv operation for each feature map, you get the following results.

(stride = 1, padding = 1) 3 x 3 x (#bounding box x (classes + offset)) / offset: (x, y, w, h)

- 3 x 3 x (4 x (classes + 4))
- 3 x 3 x (6 x (classes + 4))
- 3 x 3 x (6 x (classes + 4))
- 3 x 3 x (6 x (classes + 4))
- 3 x 3 x (4 x (classes + 4))

- 3 x 3 x (4 x (classes + 4)

So, you get a total of 8732 bounding boxes. Not all of these are considered but finally removed through non-maximal suppression. The point to note here is that the convolution layer is used as a classifier rather than considering a lot of bounding boxes. However, the SSD used convolutional layer to reduce the amount of computation and model parameters to improve speed and reduce model weight.[3]
Now, let us see the baisc tenets of multibox method:

The fundamental idea is to train a convolutional network that outputs the coordinates of the object bounding boxes directly. In order to achieve this, the MultiBox loss is the weighted sum of the following two losses:

- Confidence: a logistic loss on the estimates of a proposal corresponding to an object of interest.
- Location: a loss corresponding to some similarity measure between the objects and the closest matching object box predictions. L2 norm is used here(L2 adds “squared magnitude” of coefficient as penalty term to the loss function).
  The overall objective loss function is a weighted sum of the localization loss (*loc*) and the confidence loss (*Lconf*) where localization loss is the mismatch between the predicted boundary box and the ground truth box and confidence or classification loss is the loss in assigning class labels to predicted boxes. Therefore, the equation for total objective loss is defined in Equation 1.

$$L(x,c,l,g)=1/N(Lconf(x,c)+\alpha Lloc(x,l,g)) \quad \textbf{(1)}$$

Here, x is 1 if the prior is matched to the determined ground truth box, and 0 otherwise, N is the number of matched priors, l is predicted bounding box, g is ground-truth bounding box, c is class, *Lconf* is condence loss, *loc* is localization loss, and α is the weight for localization loss. Usually Smooth-L1 loss is used for localization on l and g and Softmax loss is used for optimizing confidence loss over multiple class confidences c.[4]

The alpha term helps us in balancing the contribution of the location loss. As usual in deep learning, the goal is to find the parameter values that most optimally reduce the loss function, thereby bringing our predictions closer to the ground truth.
In MultiBox, the researchers created what we call priors, which are pre-computed, fixed size bounding boxes that closely match the distribution of the original ground truth boxes. In fact those priors are selected in such a way that their Intersection over Union ratio is greater than 0.5. Therefore MultiBox starts with the priors as predictions and attempt to regress closer to the ground truth bounding boxes.

SSD Improvements: Back onto SSD, a number of tweaks were added to make this network even more capable of localizing and classifying objects.

- **Fixed Priors**: unlike MultiBox, every feature map cell is associated with a set of default bounding boxes of different dimensions and aspect ratios. These priors are manually (but carefully) chosen, whereas in MultiBox, they were chosen because their IoU with respect to the ground truth was over 0.5.This in theory should allow SSD to generalise for any type of input, without requiring a pre-training phase for prior generation.This in theory should allow SSD to generalise for any type of input, without requiring a pre-training phase for prior generation.
- **Location Loss**: SSD uses smooth L1-Norm (L1 adds “absolute magnitude” of coefficient as penalty term to the loss function) to calculate the location loss. While not as precise as L2-Norm(L2 adds “squared magnitude” of coefficient as penalty term to the loss function), it is still highly effective and gives SSD more room for manoeuvre.
- **Classification**: MultiBox does not perform object classification, whereas SSD does. Therefore, for each predicted bounding box, a set of *c* class predictions are computed, for every possible class in the dataset.

Additional observations on SSD:
The SSD paper makes the following additional observations:

- More default boxes results in more accurate detection, although there is an impact on speed

- Having MultiBox on multiple layers results in better detection as well, due to the detector running on features at multiple resolutions
- 80% of the time is spent on the base VGG-16 network: this means that with a faster and equally accurate network SSD's performance could be even better.
- SSD confuses objects with similar categories (e.g. animals). This is probably because locations are shared for multiple classes
- SSD produces worse performance on smaller objects, as they may not appear across all feature maps. Increasing the input image resolution alleviates this problem but does not completely address it.

Local Binary Pattern Algorithm:

In computer science, face recognition is basically the task of recognizing a person based on its facial image. It has become very popular in the last two decades, mainly because of the new methods developed and the high quality of the current videos/cameras. Note that face recognition is different of face detection:

Face Detection: it has the objective of finding the faces (location and size) in an image and probably extract them to be used by the face recognition algorithm.

Face Recognition: with the facial images already extracted, cropped, resized and usually converted to grayscale, the face recognition algorithm is responsible for finding characteristics which best describe the image.

The face recognition systems can operate basically in two modes:

Verification or authentication of a facial image: it basically compares the input facial image with the facial image related to the user which is requiring the authentication. It is basically a 1x1 comparison.

Identification or facial recognition: it basically compares the input facial image with all facial images from a dataset with the aim to find the user that matches that face. It is basically a 1xN comparison.

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.It can be seen as a unifying approach to the traditionally divergent statistical and structural models of texture analysis. Perhaps the most important property of the LBP operator in real-world applications is its robustness to monotonic gray-scale changes caused. Another important property is its computational simplicity, which makes it possible to analyze images in challenging real-time settings.[5]

The basic idea for developing the LBP operator was that two-dimensional surface textures can be described by two complementary measures: local spatial patterns and gray scale contrast. The original LBP operator forms labels for the image pixels by thresholding the 3 x 3 neighborhood of each pixel with the center value and considering the result as a binary number. The histogram of these $2^8 = 256$ different labels can then be used as a texture descriptor. This operator used jointly with a simple local contrast measure provided very good performance in unsupervised texture segmentation (Ojala and Pietikäinen 1999). After this, many related approaches have been developed for texture and color texture segmentation.

The LBP operator was extended to use neighborhoods of different sizes (Ojala et al. 2002). Using a circular neighborhood and bilinearly interpolating values at non-integer pixel coordinates allow any radius and number of pixels in the neighborhood. The gray scale variance of the local neighborhood can be used as the complementary contrast measure. In the following, the notation (P,R) will be used for pixel neighborhoods which means P sampling points on a circle of radius of R. See Fig. 3.2 for an example of LBP computation.

The value of the LBP code of a pixel (x_c, y_c) is given by:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \qquad s(x) = \begin{cases} 1, if\ x \geq 0; \\ 0, otherwise. \end{cases}$$

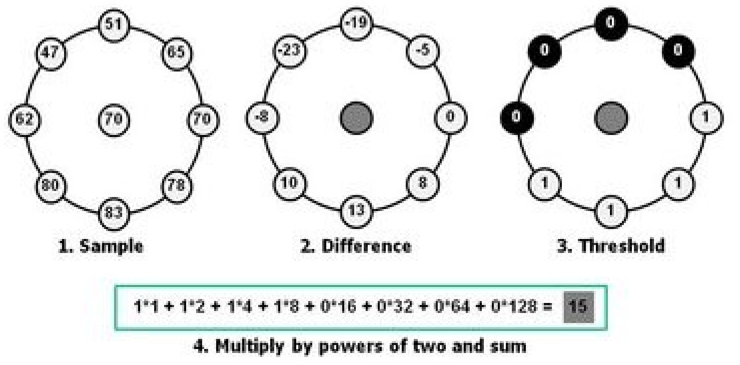

Fig. (3.2) An example of LBP computation

Face description using LBP:

In the LBP approach for texture classification, the occurrences of the LBP codes in an image are collected into a histogram. The classification is then performed by computing simple histogram similarities. However, considering a similar approach for facial image representation results in a loss of spatial information and therefore one should codify the texture information while retaining also their locations. One way to achieve this goal is to use the LBP texture descriptors to build several local descriptions of the face and combine them into a global description. Such local descriptions have been gaining interest lately which is understandable given the limitations of the holistic representations. These local feature based methods are more robust against variations in pose or illumination than holistic methods.

The basic methodology for LBP based face description proposed by Ahonen et al. (is as follows: The facial image is divided into local regions and LBP texture descriptors are extracted from each region independently. The descriptors are then concatenated to form a global description of the face, as shown in Fig. 3.3.

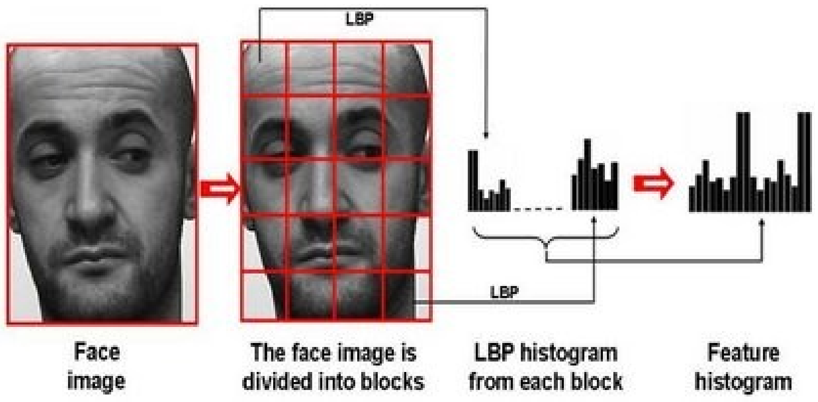
LBP
LBP
Face image
The face image is divided into blocks
LBP histogram from each block
Feature histogram

Fig. (3.3) Face description with local binary patterns

This histogram effectively has a description of the face on three different levels of locality: the LBP labels for the histogram contain information about the patterns on a pixel-level, the labels are summed over a small region to produce information on a regional level and the regional histograms are concatenated to build a global description of the face.

It should be noted that when using the histogram based methods the regions do not need to be rectangular. Neither do they need to be of the same size or shape, and they do not necessarily have to cover the whole image. It is also possible to have partially overlapping regions.

3.3.2 Methodology

Different methods for collection, processing and transmission of data were used during the execution of the project. The raspberry pi based robot is the unit for collection of data and it is also the unit for navigation.

Since the raspberry-pi has limited processing power, it was important to do processing in real time without compromising the speed of the robot. Human detection is an important aspect of the project. Communication with the user is also as important. We conduct these processes on a different medium(in this case a laptop). The flow of the execution of methods can described as:

Method of collection of data: The camera on raspberry pi based robot continuously captures images of the local environment and sends it to the processing unit. A UDP connection is established between the two for transfer of data.

Processing of data: The execution of process happens at the processing unit. It looks for any human presence in the image received from the raspberry-pi robot. The Single shot detection algorithm detects presence of human from the image that it gets. It is called single shot because it takes only a single shot to detect the objects present in the frame.

Communication: If there is a person in the frame then the processing unit send the user a message along with the photo of the person through the telegram app and starts following the person till it receives further command from the user.

User control: The user can control the raspberry pi based robot through telegram. The user can tell the robot to stop, send video recording of the person, or return to the original position. The user sends the instructions to the processing unit which in turn directs the robot to execute those instructions.

Also the model is evaluated on the basis of accuracy.

3.4 Feasibility Analysis

Majority of the literature report time complexity of deep learning models in terms of total time taken by the model for training and inference when run on specific hardware. We have used an approximation of the asymptotic complexity to get an idea of which segment of the model takes up the largest amount of time while training and inference. Forward pass of the base CNN is dominated by the time complexity of matrix multiplication. Number of multiplication to perform depends on size of input channel (or image resolution), number of convolutional layers, number of neurons in each layer, number of filters and size of each filter in each layer and the size of output feature map. Except for the input layer, size of input channel and output channel is determined by the number of neurons in each layer. at each layer there is an activation function which can be linear or quadratic based on the function. As ReLu activation function performs element-wise calculation, it runs in quadratic time on each neuron on each layer. All these are parameters or weights to be learned during training. Therefore, runtime of a convolutional model scales linearly with the total number of learnable weights. The total time-complexity of convolution layers for forward pass is as followed,

$$time_{forward} = time_{convulation} + time_{activation}$$

$$= O \sum_{l=1}^{L} n_{l-1} \cdot (f \cdot f) \cdot n_l \cdot (m_l \cdot m_l) + O(L \cdot n_c)$$

$$= O(weights)$$

Here, l is the index of a convolutional layer, L is the total number of layers, $nl-1$ is the number of input channels of the *lth* layer, f is filter height and width, nl is the number of filters in *lth* layer, *ml* is the size of output feature map and *nc* is number of neurons in a layer which in practice varies for each layer. For training, there is also back-propagation cost which also runs to calculate weights for all the parameters using chain-rule. Thus, *timebackprop* is also O(weights).

Since the problem can be solved in polynomial time, it can be termed as NP complete. It is feasible to implement the solution of the problem statement.

3.5 Architecture Diagram

3.5.1 Physical Architecture

The physical architecture diagram shows the physical connections in the raspberry-pi based robot. The main unit of raspberry pi is connected to battery and motor driver. The battery provides the energy and motor driver runs the motors controlling wheels. The ultrasonic sensor and camera is also connected to the raspberry pi and they are used for capturing data. Ultrasonic sensor keeps record of distance from nearby objects and camera captures images for processing.

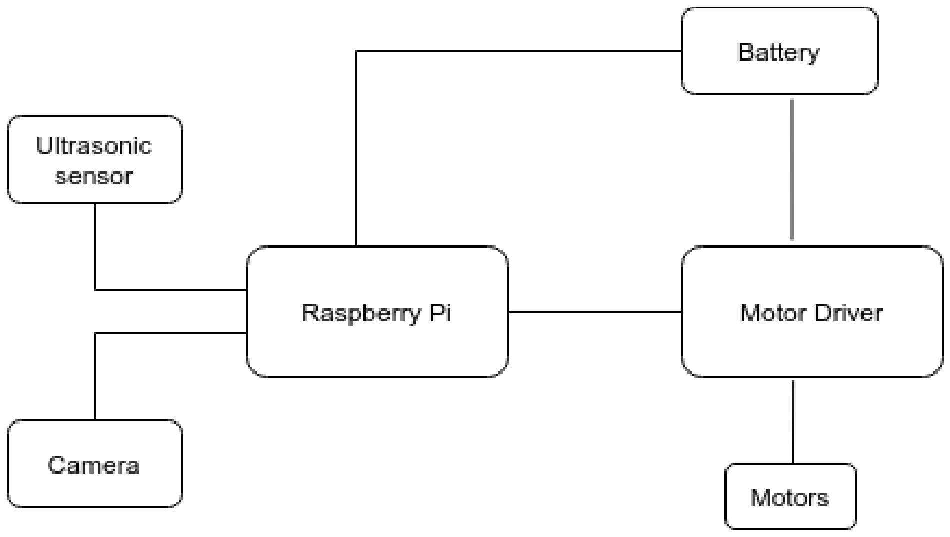

Fig. (3.4) Physical Architecture Diagram

3.5.2 Overview diagram

The overview diagram gives the detailed information of the connections in the system’s physical and logical components. The overview diagram shows the relationship between the raspberry pi and processing unit. The connections between the processing unit and telegram app and between user and telegram app. The overview diagram is given below:

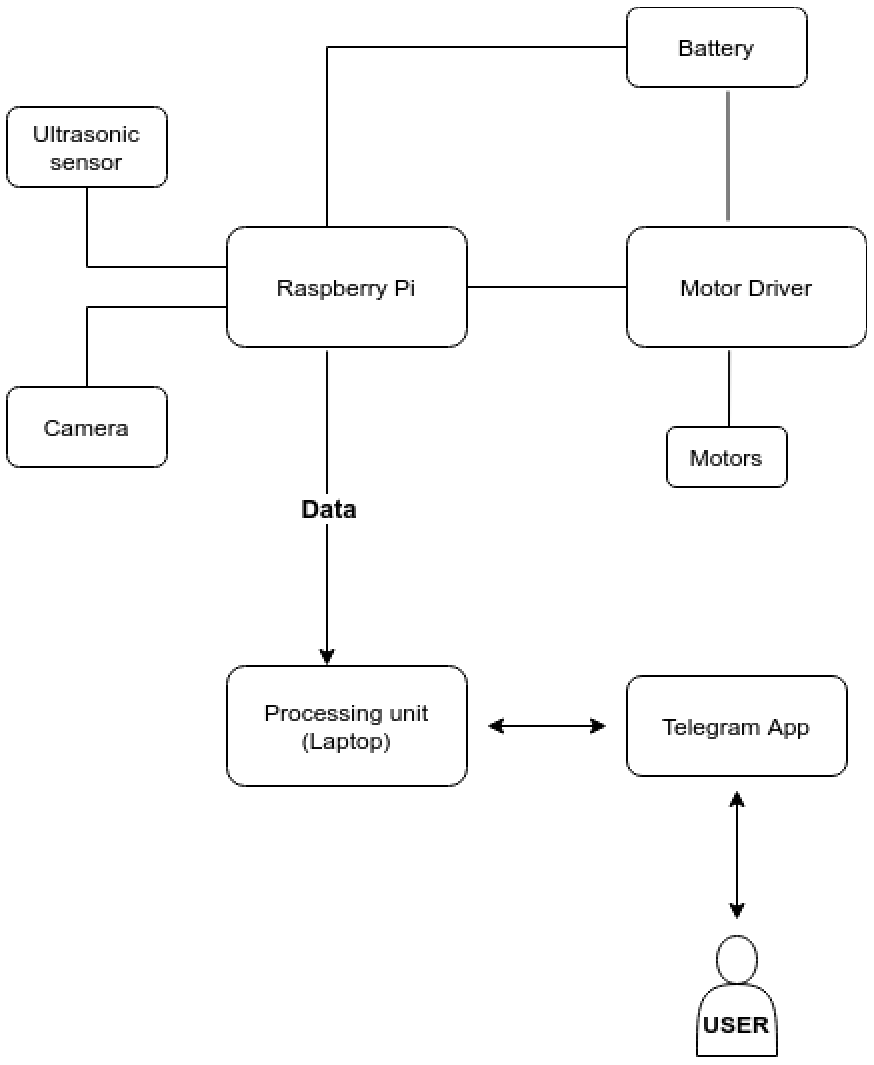

Fig. (3.5) Overview Diagram

3.6 UML Diagrams

3.6.1 Use case Diagram

The use case diagrams model the behavior of a system and help to capture the requirements of the system. The use case diagram for this project focuses on for main requirements of the user from the robot i.e. sending photo, sending video, coming at the reset position and stopping when instructed.

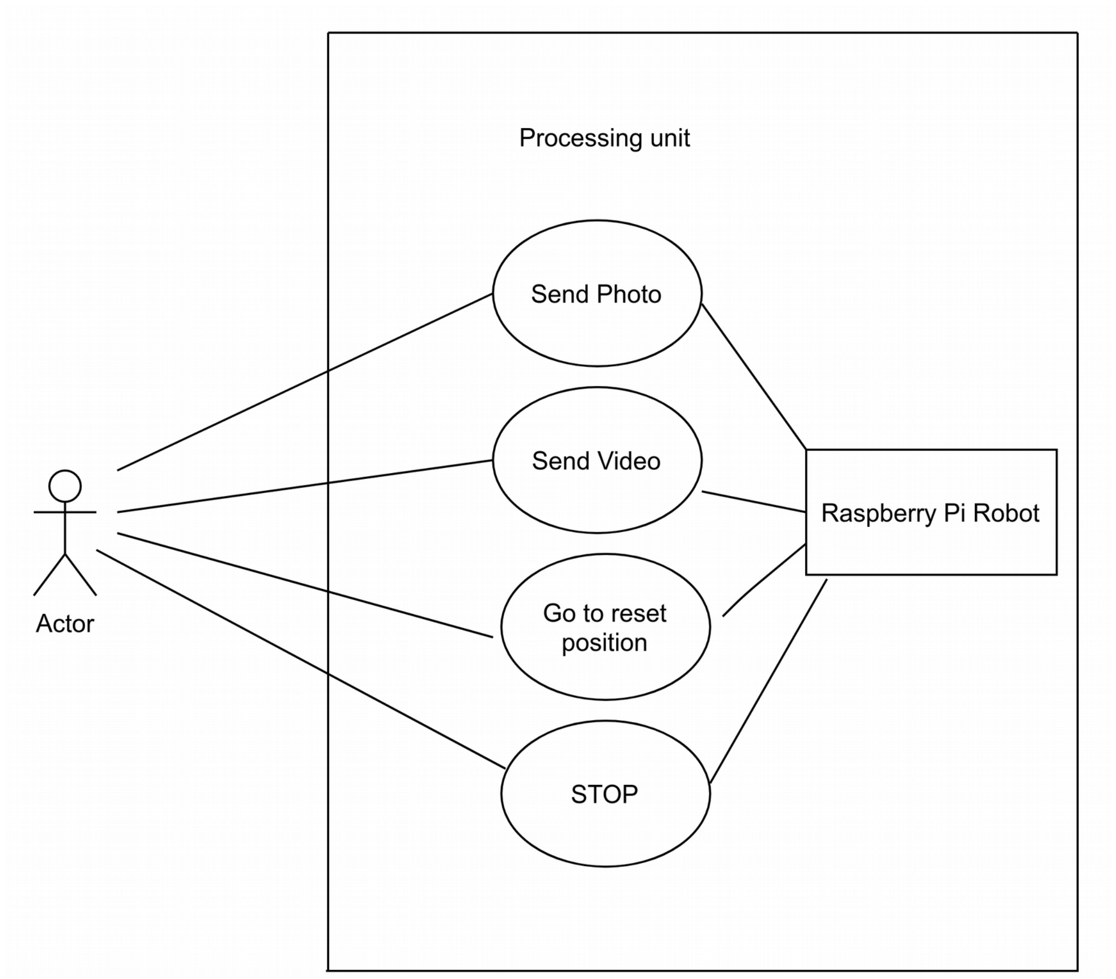

Fig. (3.6) Use Case Diagram

3.6.2 Activity Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.
This diagram represents various activities in the system along with the different actions that take place within each activity. It also depicts the flow of the actions within the activities and relation between different activities. The activity diagram is mentioned on next page.

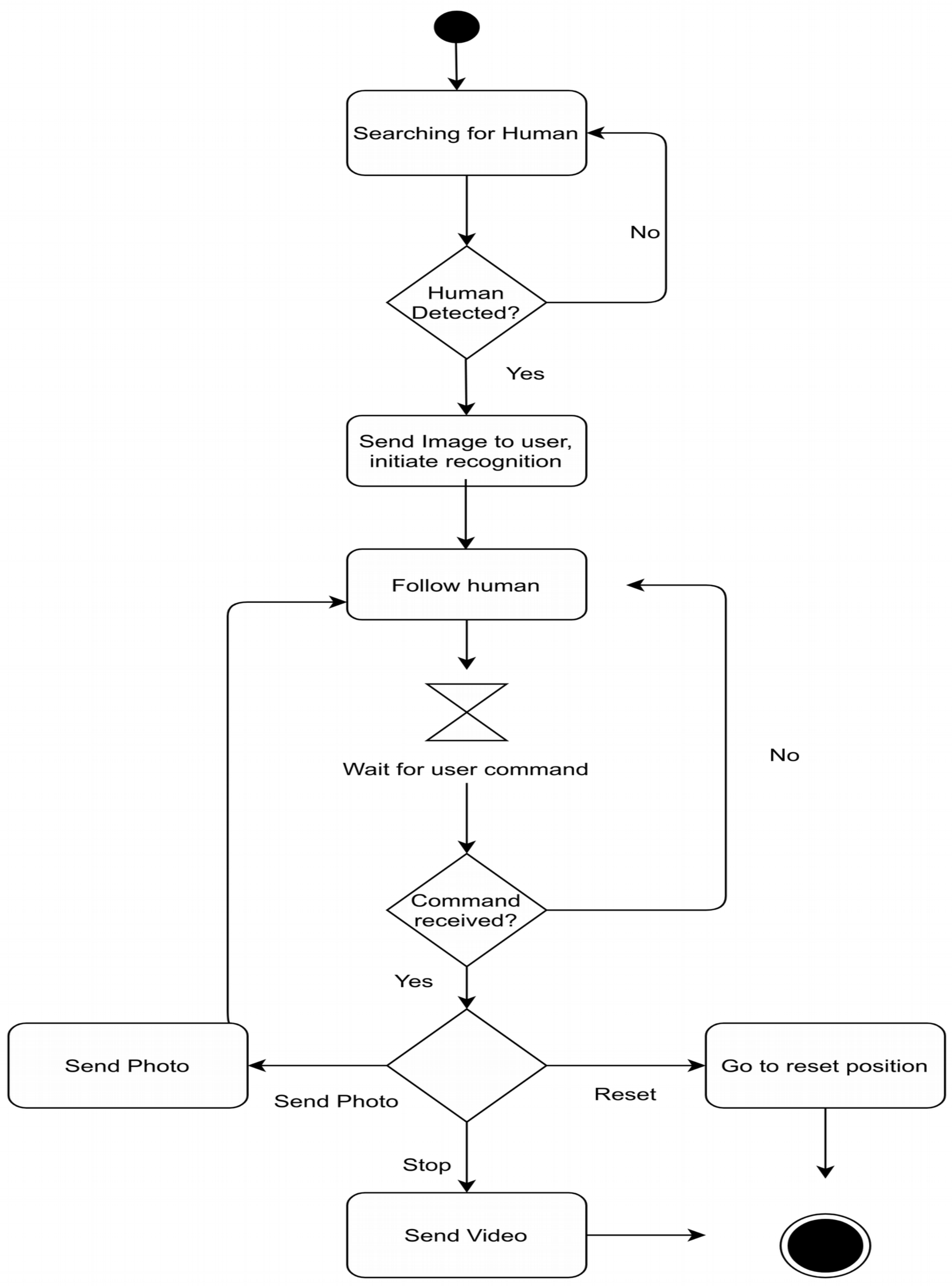

Fig (3.7): Activity Diagram

3.6.3 Sequence Diagram

The Sequence diagram represents the UML, which is used to visualize the sequence of calls in a system that is used to perform a specific functionality. The Sequence diagram shows the message flow from one object to another object. Sequence diagram is used to describe the behavior of several objects in a single use case.
Here, the sequence diagram shows the interaction between the use, the processing unit and the laptop. The sequence diagram is shown on next page.

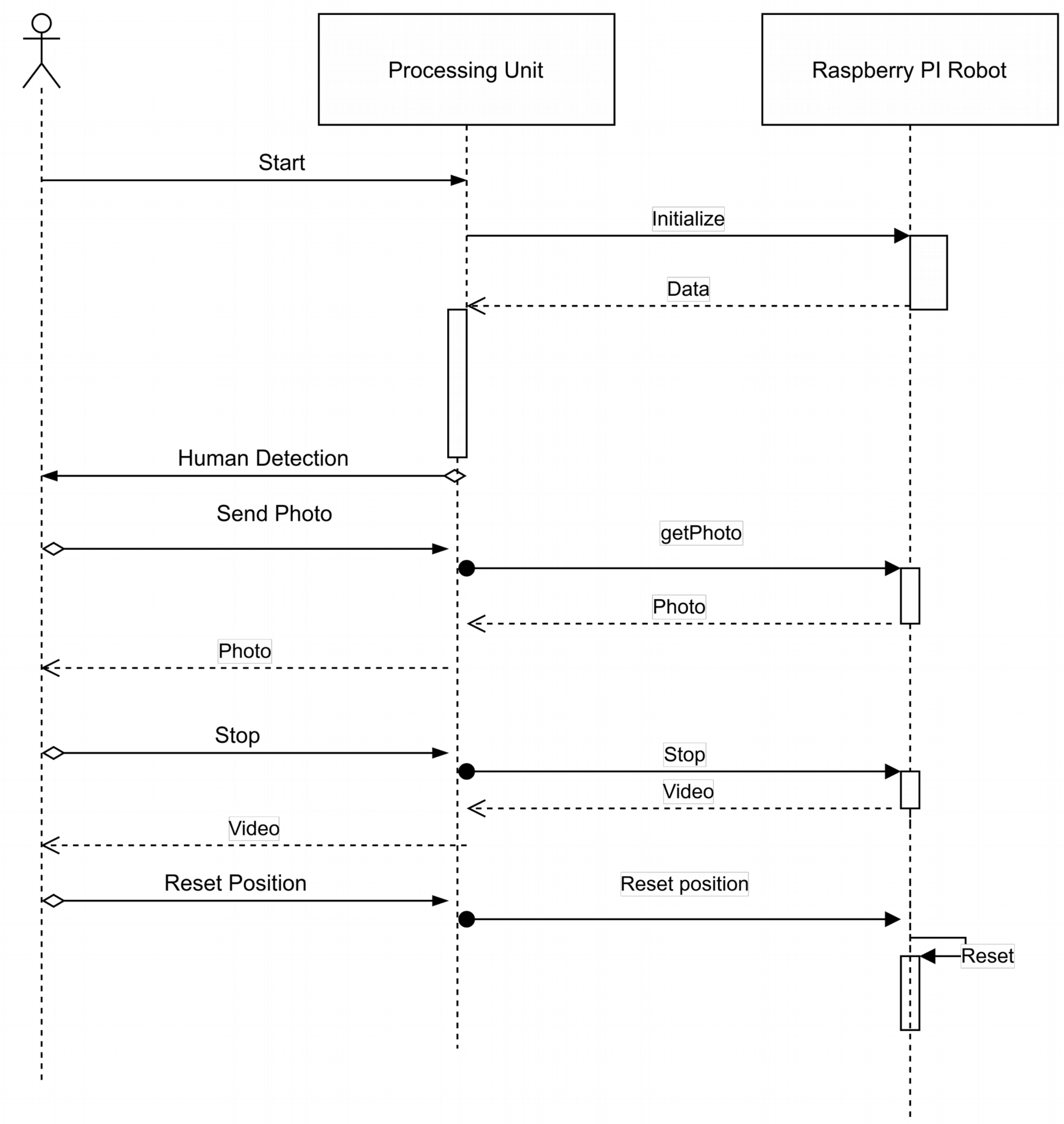

Fig. (3.8) Sequence Diagram

3.6.4 Component Diagram

In Unified Modeling Language, a component diagram depicts how components are wired together to form larger components or software systems. They are used to illustrate the structure of arbitrarily complex systems. This figure shows the interfaces present in the system and their relation.

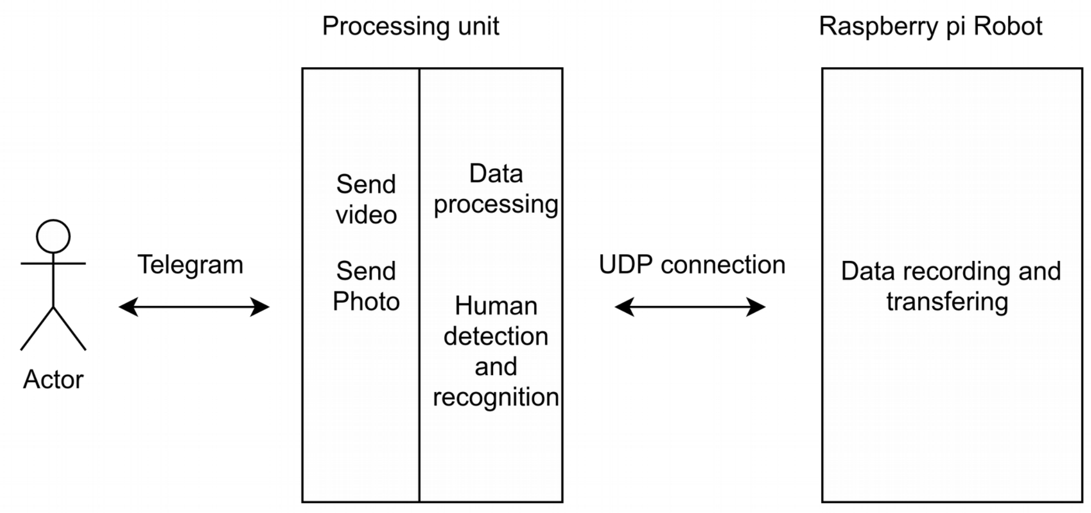

Fig. (3.9) Component Diagram

3.6.5 State Machine Diagram

State Machine diagrams are used to design interactive systems that respond to either internal or external event. State Machine diagram in UML visualizes the flow of execution from one state to another state of an object. The state machine diagram is shown in next page.

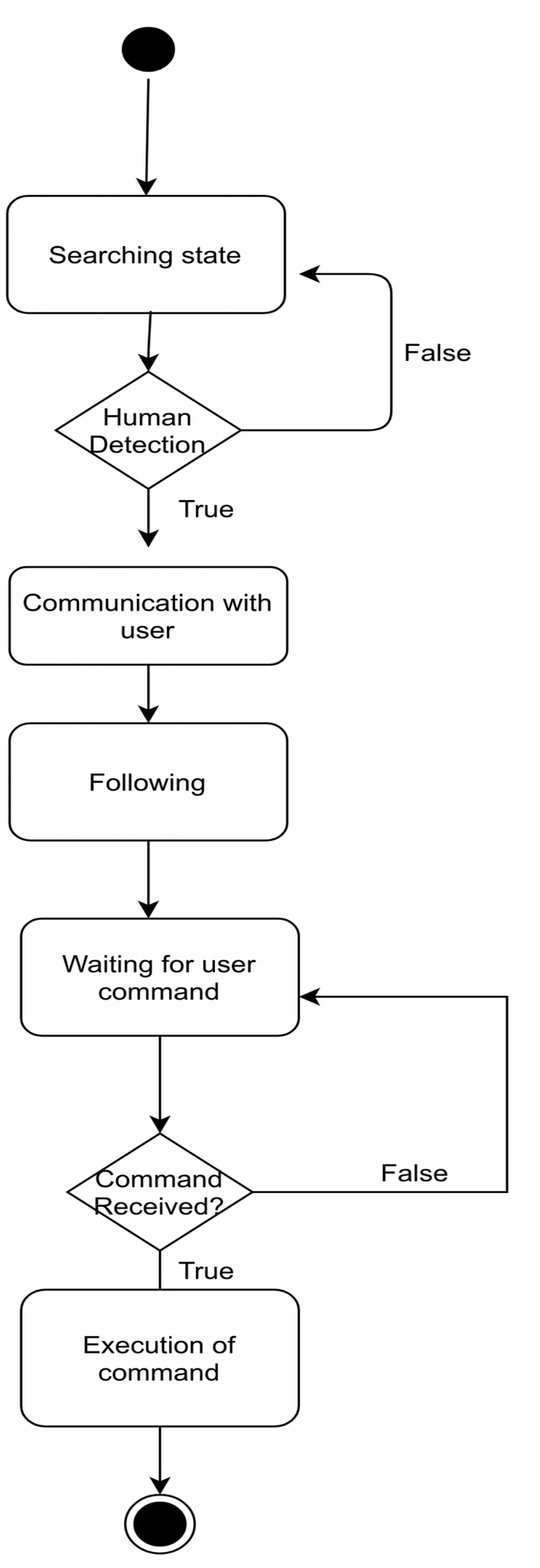

Fig. (3.10) State Machine Diagram

3.6.6 Deployment Diagram

A deployment diagram is a diagram that shows the configuration of run time processing nodes and the components that live on them. Deployment diagrams is a kind of structure diagram used in modeling the physical aspects of an object-oriented system. They are often be used to model the static deployment view of a system (topology of the hardware).

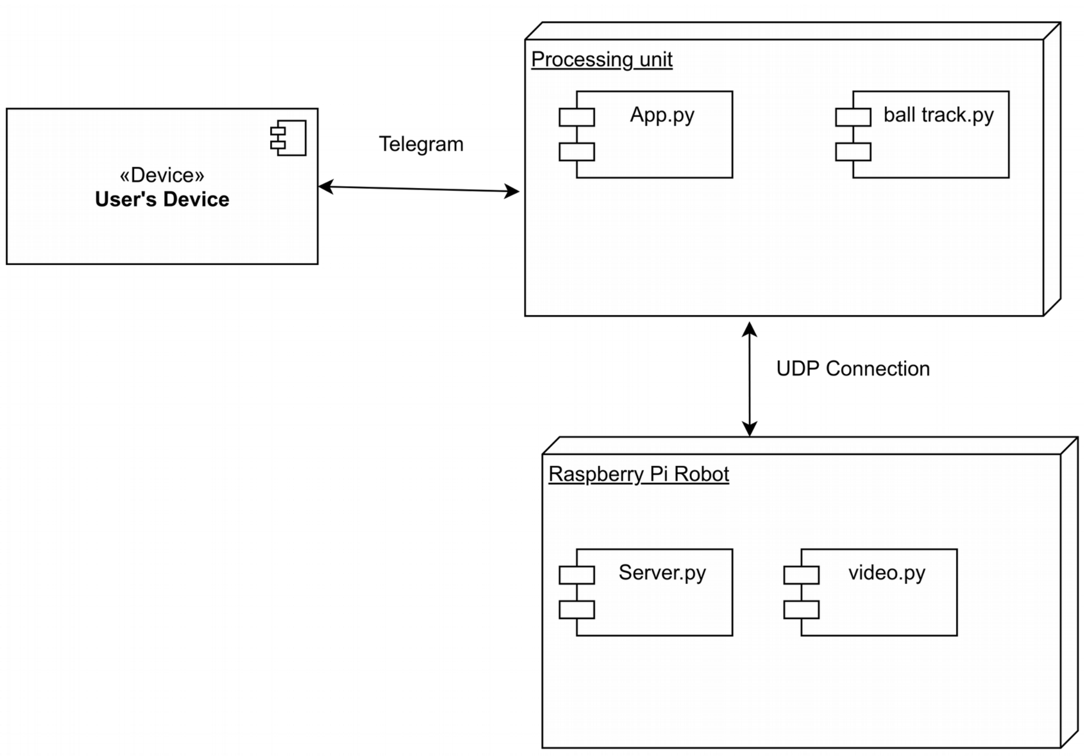

Fig. (3.11) Deployment Diagram

Chapter 4
IMPLEMENTATION & CODING

```
UDP_IP = "192.168.1.101"
UDP_PORT = 12345
sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
```

We are using UDP socket connection as an communication medium between laptop and Raspberrypi.

```
for i in range(number_files):
   globals()['image_{}'.format(i)] = face_recognition.load_image_file(list_of_files[i])
   globals()['image_encoding_{}'.format(i)] =
face_recognition.face_encodings(globals()['image_{}'.format(i)])[0]
   faces_encodings.append(globals()['image_encoding_{}'.format(i)])
# Create array of known names
   names[i] = names[i].replace(cur_direc, "")
   faces_names.append(names[i])
```

In this section we are training our robot with images of all the authorized people’s faces which may help it to detect a potential threat.

```
def check_msg():
   updates = bot.getUpdates(timeout=1000)
   for i in updates:
      last = i.message.text
      msgid = i.message.message_id
   return last,msgid
```

Here our robot checks for any new set of instructions or command received from the user.

```
def multi2(temp10):
        unknown_image =
face_recognition.load_image_file("image{}.jpg".format(temp10))
        try:
                unknown_encoding =
face_recognition.face_encodings(unknown_image)[0]
                matches = face_recognition.compare_faces (faces_encodings,
unknown_encoding)
                if(matches[0] == True):
                        os.remove('image{}.jpg'.format(temp10))
                        sock.sendto(b'stop', (UDP_IP, UDP_PORT))
                        store = msgid
                        print('Known person found')
                        bot.sendMessage(text = 'Known Person found',
chat_id=chat_id)
                        sock.sendto(b'reset', (UDP_IP, UDP_PORT))
                        time.sleep(100)
                os.remove('image{}.jpg'.format(temp10))
        except:
                print('Not able to detect face')
```

Here the robot detects the image and while it sends the image to the user via telegram it starts following and simultaneously checks whether the person is known or unknown. If the person is known, it stops following and returns to the original position by itself whereas if the person is unknown, it continues to follow and wait for the commands from the user and performs the action accordingly.

```
for i in np.arange(0, detections.shape[2]):
                confidence = detections[0, 0, i, 2]
                if confidence > args["confidence"]:
                        idx = int(detections[0, 0, i, 1])
                        #print(idx)
                        if idx == 15:
                                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                                (startX, startY, endX, endY) = box.astype("int")
                                rects.append(box.astype("int"))
                                label = "{}: {:.2f}%".format(CLASSES[idx],
                                        confidence * 100)
                                cv2.rectangle(frame, (startX, startY), (endX, endY),
                                        (0, 255, 0), 2)
                                y = startY - 15 if startY - 15 > 15 else startY + 15
                                cv2.putText(frame, label, (startX, y),
                                        cv2.FONT_HERSHEY_SIMPLEX, 0.5,
COLORS[idx], 2)

        objects = ct.update(rects)
        # loop over the tracked objects
        for (objectID, centroid) in objects.items():
                text = "ID {}".format(objectID)
                cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)
```

In few cases every moving object may not be a threat. So as a precaution we first identify the object and classify whether it’s a person or not.

```
if objects:
                    ret , img = cam.read()
                    out.write(img)
                    if temp10<2:
                              cv2.imwrite('image{}.jpg'.format(temp10), frame)
                              threading.Thread(target=multi(temp10)).start()
                              threading.Thread(target=multi2(temp10)).start()
                              temp10+=1
                    if centroid[0] > 280:
                              print("Turning Right")
                              sock.sendto(b'clear', (UDP_IP, UDP_PORT))
                              sock.sendto(b'right', (UDP_IP, UDP_PORT))
                    elif centroid[0] < 120:
                              print("Turning left")
                              sock.sendto(b'clear', (UDP_IP, UDP_PORT))
                              sock.sendto(b'left', (UDP_IP, UDP_PORT))
                    else:
                              print("Forward")
                              sock.sendto(b'clear', (UDP_IP, UDP_PORT))
                              sock.sendto(b'forward', (UDP_IP, UDP_PORT))
          else:
                    print("Search")
                    sock.sendto(b'clear', (UDP_IP, UDP_PORT))
                    sock.sendto(b'search', (UDP_IP, UDP_PORT))
```

After classifying the object as a person the robot starts following and simultaneously checks whether its known or unknown and also sends the exact directions in order to follow the person.

Chapter 5

CONCLUSION

Conclusion:

Raspberry-pi based devices can help implementing various robots, to increase quality of life of people. These inexpensive machines when programmed correctly can save a lot of money and can be of great use. It is also to be noted that robots are sooner or later going to be an important part of our life just like mobile phones now.

The safety of a house is important aspect of one’s life and this model with can be used in real life situations to cater those needs. The model is also able to communicate with the owner and record things, hence it can be called an addition to the modern day CCTV cameras, it has lower price and better functionality than them.

Project Implementation using the waterfall model was a good choice for a small team, a structured approach was followed during the implementation of this project involving exchange of ideas, sharing of data, implementation and coding which resulted in successful completion of project.

REFERENCES

[1] Wei Liu1 , Dragomir Anguelov, “SSD: Single Shot MultiBox Detector”

[2] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, Sergey Ioffe “Scalable, High-Quality Object Detection”

[3] https://wdprogrammer.tistory.com/52

[4] http://www.scholarpedia.org/article/Local_Binary_Patterns
Matti Pietikäinen, Machine Vision Group, Department of Electrical and Information Engineering, University of Oulu, Finland

[5] Rahul Madbhavi, Shreepad Potadar, Sylvester Jerome D’souza, “Object Follower and Barrier Escaping Robot Using Image Processing”, IJIRSET 2015.

[6] Kirti Bhagat, Sayalee Deshmukh, Shraddha Dhonde, Sneha Ghag, “Obstacle Avoidance Robot”, IJIRSET 2016.

[7] Rupa Gurram, SweathaSuresh.B., Sneha.B.R., Sushmitha.R, “Object Tracking Robot on Raspberry Pi using Opencv”, IJETT 2016.

[8] Chinmayi R, Yogesh kumar Jayam, Venkatesh Tunuguntla, Jaideep venkat Dammuru, Harshith Nadella, “Obstacle Detection and Avoidance Robot”, IEEE 2018

[9] https://towardsdatascience.com/ssd-single-shot-detector-for-object-detection-using-multibox-1818603644ca