GESTURE BASED AUTOMATION FOR REMOTE ROBOT CONTROL

by

INDRAJEET R. KHATER 2010103668
SUBRAMANIAM KEDARNATH 2011103022
ADNAN ALI 2011103028

A project report submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY, CHENNAI – 25

MAY 2015

BONAFIDE CERTIFICATE

Certified that this project report titled **GESTURE BASED AUTOMATION FOR REMOTE ROBOT CONTROL** is the *bonafide* work of **INDRAJEET R. KHATER (2010103668)**, **SUBRAMANIAM KEDARNATH (2011103022)** and **ADNAN ALI (2011103028)** who carried out the project work under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place: Chennai
Date:

Dr. Arul Siromoney
Professor
Department of Computer Science and Engineering
Anna University, Chennai – 25

COUNTERSIGNED

Head of the Department,
Department of Computer Science and Engineering,
Anna University Chennai,
Chennai – 600025

ACKNOWLEDGEMENT

We would like to thank the entire committee consisting of Dr. V. Vetriselvi, Dr. S. Sudha, Dr. V. Mary Anita Rajam, Dr. A. P. Shanthi, Dr. Rajeswari Sridhar and Dr. D. Shiloah Elizabeth for constantly motivating us and identifying the areas of improvement on the project. We would like to extend our immense gratitude to our project guide, Dr. Arul Siromoney for his perpetual support and able guidance which was instrumental in taking the project to its successful conclusion. Finally, we would like to thank the Head of the department, Dr. D. Manjula for providing a conducive environment and amenities to facilitate our project work.

Indrajeet R. Khater **Subramaniam Kedarnath** **Adnan Ali**

ABSTRACT

Smarter and more convenient ways of user interaction are becoming a common affair in today's world. This holds true in the field of robotics especially HRI (Human Robot Interaction). This project is at the convergence of gesture recognition technology and robotics. It is envisioned that the robots of the future seamlessly interact with humans and there is no better way to achieve this than gestures.

Essentially, this project is aimed at setting up an arrangement that helps the user interact with a robot in a remote location through the Internet. Instead of using any traditional methods of input, a gesture-based interface is suggested to make the experience more intuitive and user friendly. The basic framework includes an input device capable of recognizing and processing gestures. These gestures are then communicated over the Internet to a robot, which follows the prescribed movement.

A research paper on the above mentioned arrangement was presented in the IEEE conference at the Student Research Symposium in Delhi (ICACCI-2014). The same is now a published work of research (ISBN: 978-981-09-2229-0). The implementation of the above arrangement using latest equipment is a first of its kind.

ABSTRACT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

RPI	Raspberry Pi
SDK	Software Development Kit
PC	Personal Computer
MP4	MPEG 4
SSH	Secure Shell

CHAPTER 1

INTRODUCTION

1.1 GENERAL OVERVIEW

Robotics is an emerging technology in the field of science. These days many types of wireless robots are being developed and are put to varied applications and uses. The broad objective of taking up this project is to create a intuitive and innovative method of controlling a remote object using simple gestures. This approach is intuitive because the usage of hand gestures comes naturally to any user without prior training unlike generic remote controls. The innovativeness lies in the fact that while implementing the project, we used technology that have never been used together to achieve this result. A gesture recognition device interfaced with a processing unit is placed at the user's location. A simple moving robot with four wheels is present at a remote location anywhere on the globe with the constraint being that internet connection is available. As the user makes simple gestures the device recognises them, processes them and transfers it to the robot over the internet. The robot then moves in an appropriate direction based on the gesture performed. For example if the user performs a simple swipe right gesture, the robot moves in the right direction. Thus, we have defined a set of gestures and the consequent robot movements based on the gesture.

1.2 ABOUT THE PROJECT

The project is divided into three main modules. The first module is the gesture recognition device interfaced with the processing unit. Gesture recognition is achieved using the Microsoft Kinect for Windows which is a motion sensing input device. While even a generic computer web cam is capable of gesture recognition, it is not the cutting edge technology that we aim to use. It lacks the basic accuracy, efficiency and performance. On the other hand, competitors like LeapMotion controller have a smaller observation area and Myo by Thalmic Labs is more suitable for muscular movement. Kinect's greatest merit lies in its ability to handle both high quality image and signal processing simultaneously. Not only does it have a greater field of observation, but is also developer friendly and popular in the market. These features led us to believe that the Kinect is the best possible gesture recognition device available in the scope of our project.

The second module is the robot which is present at the remote location. The robot is controlled by the Raspberry-Pi (RPi), a microcomputer which is interfaced with the DC motors using an L298N full bridge motor driver. For constructing the robot, we had other choices like the Arduino. However, compared to the Arduino which is a commonly used micro-controller, the RPI offers a whole platform to work on as it is a micro-computer. Since the RPI is also very cost-effective, the RPI is adopted as the robot controller.

The third module is basically interfacing the gesture recognition device with the robot over the internet. Since, we want the robot to be controlled remotely over a long distance, the internet is the best choice to achieve this long range communication. From any location on the

globe, we can control the robot provided there is internet connection available at the remote location. Other interfacing mechanisms like bluetooth provide a much shorter range of communication. One of the drawbacks of using the internet is that it is difficult to model the transmission efficiency at any time period, because the speed of the internet is never constant.

CHAPTER 2

RELATED WORK

Remote control of robotic systems has been applied in manufacturing, underwater manipulation, disaster recovery operations, teleoperations, space exploration, etc. With recent advances on the Internet technology, remote control of mobile robots presents new opportunities in resources sharing, long-distance learning, and remote experimentation. Kun Qian, Jie Niu and Hong Yang [1] developed a Gesture Based Remote Human-Robot Interaction System Using Kinect. They used complex algorithms like Camshaft Based hand tracking and Hand Motion Trajectory recognition to obtain the gestures from the kinect sensor. Then they constructed a bi-handed Cyton robot arm which would be controlled using the gestures recognized. While our project conceptually inclines to the objective of this paper, the fundamental difference is in the manner the robot is built. The gestures in [1] are literally being mimicked by the robotic arm and hence the gestures are limited to the robot's design. We aim to define custom gestures depending on the motion of the robot (4 wheeler). However, the preliminary architecture design is strikingly similar. While [1] uses a computer to control the robot, we aim to cut down the costs and use a micro-computer (RPI) instead. Silas F. dos Reis Alves, Alvaro J. Uribe-Quevedo, Ivan Nunes da Silva and Humberto Ferasoli Filho [2] developed Pomodoro, a Mobile Robot Platform for Hand Motion Exercising. Pomodoro is basically a device for encouraging hand motion exercise through flexion/extension and ulnar/radial deviation movements. It however makes use of

Leap Motion technology to recognise the gestures. Since the cost of acquiring and maintaining a mobile robot is expensive, the system was designed with a complimentary 3D virtual environment, which did not requires the physical robot, but allowed connectivity with the real robot if available. The leap motion sensor tracking is prone to errors when the hand and/or fingers are occluded, or when there is no line- of-sight between the aligned fingers and the sensor. The Pomodoro works only in a limited range because it uses bluetooth to interface with the robot. We aim to overcome the limited range by using the internet instead. Viren Pereira, Vandyk Amsdem Fernandes and Junieta Sequeira [3] developed a Low Cost Object Sorting Robotic Arm using Raspberry Pi. They basically automated the process of recognising objects and sorting it using a raspberry-pi. Moreover [3] uses a basic camera module for image recognition. The paper aims to sort objects based on colors. Our idea is similar as we aim to control robots using gestures, thus the end purpose is different but the general processes and methods used are similar. From this paper, we gained a good idea about the various merits of the RPI and how we could harness the Raspberry-pi in our project. The use of the Raspberry-pi reduces the cost of the project while providing an open source and flexible Linux based platform for experimenting. K. K. Biswas and Saurav Kumar Basu [4] conducted research on Gesture Recognition using Microsoft Kinect. They proposed a method to recognize human gestures using a Kinect depth camera. The paper explained how to obtain the depth profile of the subject and mentioned about the ability to recognize multiple human gestures using the Kinect. [4] programmed complex gestures like CLAP for Clapping, CALL for Hand gesture to call someone, WAVE for Waving hand and NO for Shaking head sideways. Thus, we understood that Kinect was one of the best gesture recognition devices available and we could use it in our project.

Patrick Benavidez and Mo Jamshidi [5] developed a Mobile Robot Navigation and Target Tracking System. They basically built an autonomous robot using an x86 based computer onboard the robot running Ubuntu Linux. Raspbian is also a flavor of Linux. Hence, the framework for the navigation of the robot was essentially the same. However, our robot is not autonomous and we will control it using hand gestures recognized by the Kinect. The problem of remote control of robotic systems has been the subject of much research in recent years. Crisman and Webb [6] developed an autonomous land vehicle for steering autonomously with high intelligence on the road under different conditions. Varaiya [7] proposed a smart vehicle for incorporating the core driver technique into intelligent ve- hicle highway systems. Akash S A, Akshay Menon, Arpit Gupta, Md Waheeb Wakeel, Praveen MN and P. Meena [8] designed A novel strategy for controlling the movement of a Smart Wheelchair using Internet of Things. The different modes of control for the movement of the chair are made based on the specific needs of the user. Apart from the controls like joystick, chin control, voice activation, control through head movement, through calls made using a mobile, this chair can also be remotely controlled through the internet. Raspberry Pi is used as a master controller for both joystick as well as internet control. We thus realised the power of IOT (Internet of Things). IOT goes beyond machine-to-machine communications (M2M) and covers a variety of protocols, domains and applications.

CHAPTER 3

REQUIREMENTS ANALYSIS

3.1 REQUIREMENTS

3.1.1 Robot

Hardware Requirements for the Robot

1) Raspberry pi microcomputer
2) L298N motor driver
3) Robot chassis
4) Wheels
5) 12V battery
6) Male-female connecting wires
7) Wi-Fi modem/ router
8) Portable charger
9) 3G dongle

Software Requirements for the Robot

1) Raspbian OS (Linux flavour for RPI)
2) XRDP (remote desktop) VNC viewer

Functional Requirements for Robot

1) Robot should be connected to the internet
2) Robot should be charged
3) Robot should be capable of receiving gesture signals
4) Robot should have the necessary resources to process the signals

5) Robot should move according to the processed gesture signals

3.1.2 Gesture recognition

Hardware Requirements for the Gesture Recognition Device

1) Microsoft Kinect
2) Laptop (Processing unit)

Software Requirements for the Gesture Recognition Device

1) Microsoft Kinect SDK

Functional Requirements for Gesture Recognition Device

1) Processing unit should be connected to the internet
2) The Microsoft Kinect should be interfaced with the processing unit
3) The user should be recognised on the processing unit screen
4) The movement of the users hand should be tracked by the sensors
5) For a recognised gesture, the program should trigger the python program

3.1.3 Internet Module

Software Requirements for the Internet module

1) SSH (Putty)
2) XRDP (remote desktop) VNC viewer
3) Terminal emulator for android

Functional Requirements for the Internet module

1) Activate the SSH tunnel
2) Forward the recognised gesture signals from the processing unit connected with the

3.1.4 Analysis

Robot is built using the raspberry-pi, the reason for choosing which over others is made clear in the subsequent chapters. The chassis is built with 4 wheels and an L298N motor driver that is interfaced with the RPI. The gesture signals received trigger the python program to drive the wheels of the robot. The Microsoft Kinect is connected to a Windows PC which hosts the C# program written to interact with the Kinect. The program can identify the user using the coordinates sent from the sensors and hence the movement tracking is done. On the completion of a gesture, the necessary signals are sent through the SSH tunnel which are then picked up by the robot program.

CHAPTER 4

SYSTEM DESIGN

4.1 COMPONENTS IN THE PROJECT

4.1.1 Hardware Components

1. Raspberry pi microcomputer
2. L298n motor driver
3. Robot chassis
4. Wheels
5. 12V battery
6. Male-female connecting wires
7. Wi-Fi modem/ router
8. Portable charger
9. 3G dongle
10. Microsoft Kinect
11. Raspberry Pi camera module

4.1.2 Software Components

1. Raspbian OS (Linux flavor for RPI)
2. Microsoft Kinect SDK
3. Microsoft visual studio VNC server
4. SSH (Putty)
5. XRDP (remote desktop) VNC viewer
6. Terminal emulator for android

4.1.3 Programs

a) Python program to move robot in the following directions:

1. Left
2. Right
3. Forward
4. Backward
5. Circular manner

b) Gesture detection and processing for the Kinect:

1. Swipe left
2. Swipe right
3. Mouse control
4. Swipe up
5. Swipe down
6. Circle

c) C# SSH Connection

4.2 TECHNOLOGY USED

4.2.1 Raspberry-pi

The robot is aimed to be controlled using the Raspberry-pi. Raspberry pi (RPI) is a credit card sized microcomputer that runs on a Linux based operating system loaded on its SD card . It is powered by the Broadcom BCM2835 system-on-a-chip (SoC) that includes a 32-bit ARM1176JZFS processor, clocked at 700MHz, and a Videocore IV GPU. It has 512 MB ram and can be over-clocked to 1000 MHz. It can connect to a network by using an external user-supplied USB Ethernet or Wi- Fi adapter. Furthermore, the raspberry-pi has two USB ports, an HDMI port, Audio In and out, RCA video output and GPIO Pins inbuilt. This contains an ARM1176JZFS (ARM11 using

an ARMv6-architecture core) with floating point, running at 700Mhz. Apart from this, the Raspberry-pi can have a 5MP camera module mounted on it. Being Linux based, it gives the programmer more freedom to tweak with the various flavors of the Raspbian operating system.

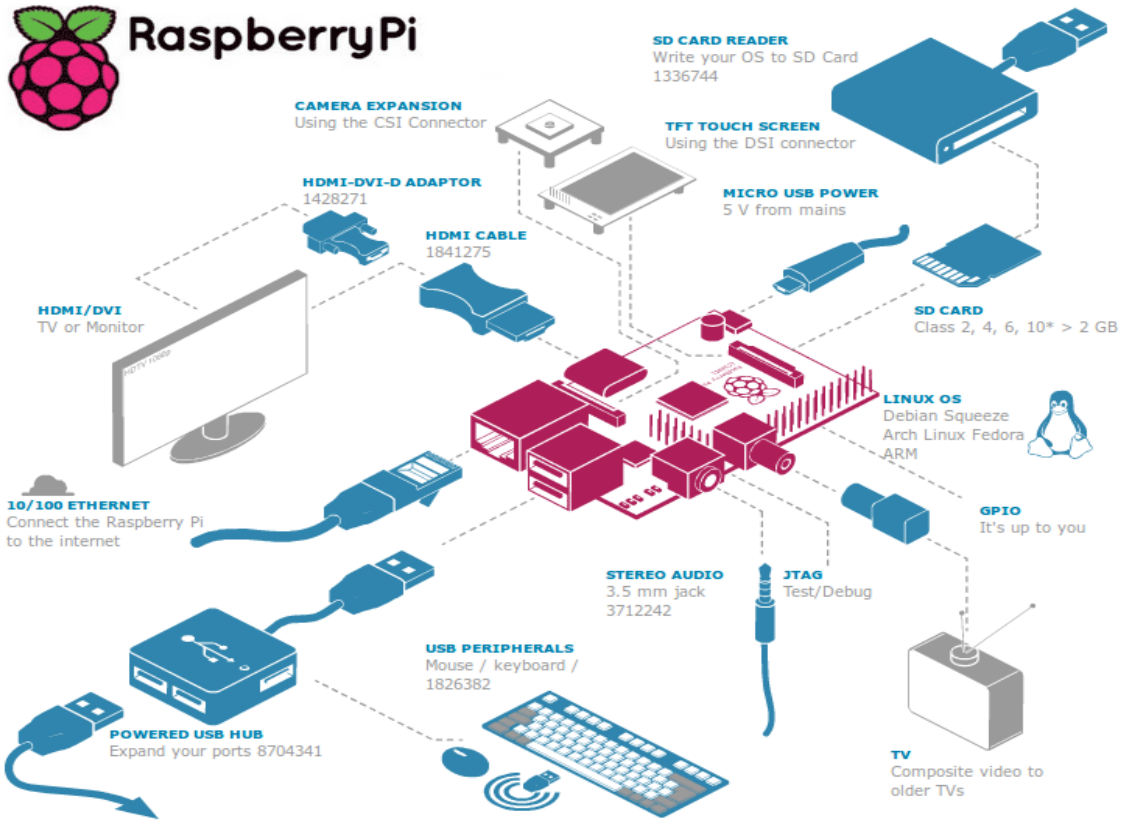

Figure 4.1 Raspberry Pi

4.2.2 L298N Motor Driver

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently

of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

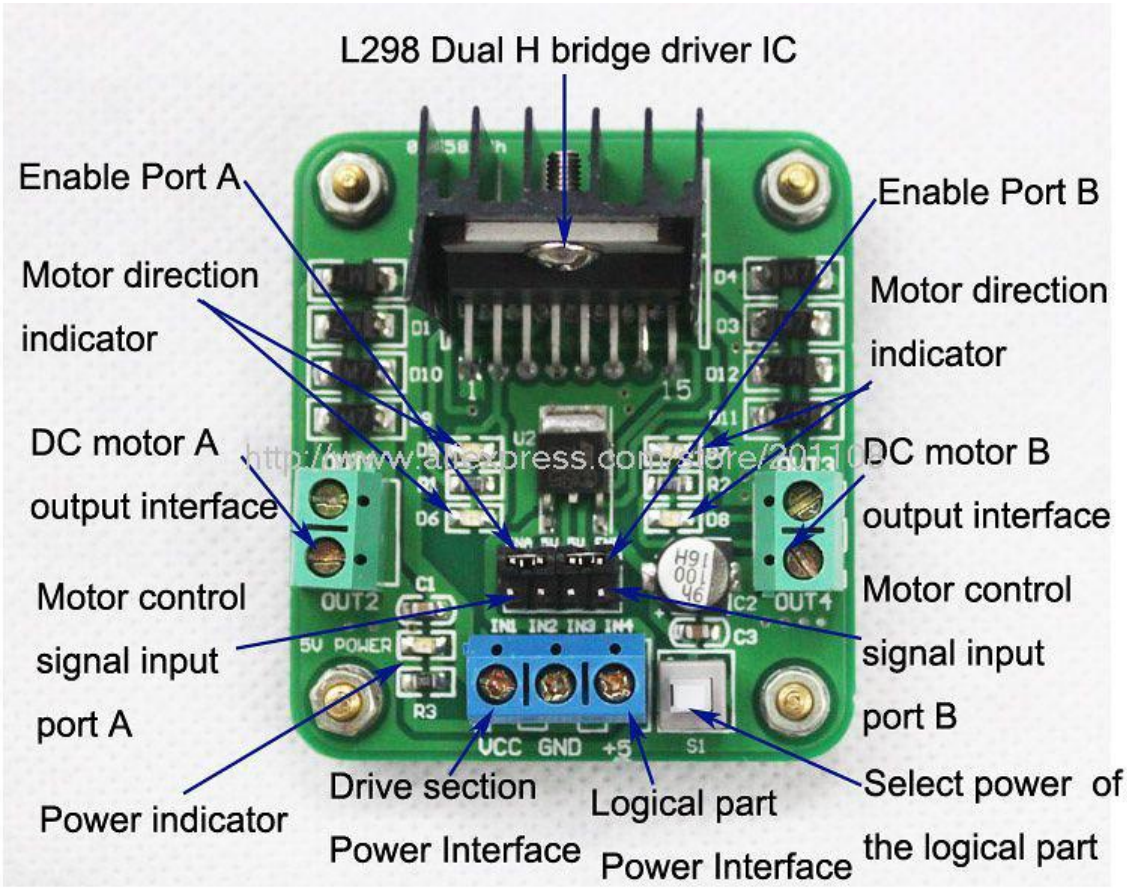

Figure 4.2 L298N Motor Driver

4.2.3 Microsoft Kinect

Gesture and voice recognition is achieved using the Microsoft Kinect for Windows which is a motion sensing input device. The Kinect contains an RGB camera that stores three channel data in a 1280x960 resolution. This makes capturing a color image possible. It has an infrared (IR) emitter and an IR depth sensor. The emitter emits infrared light beams and the depth sensor reads the IR beams reflected back to

the sensor. The reflected beams are converted into depth information measuring the distance between an object and the sensor. This makes capturing a depth image possible. The Kinect also has a multi-array microphone, which contains four microphones for capturing sound. Because there are four microphones, it is possible to record audio as well as find the location of the sound source and the direction of the audio wave. Finally, there is a 3-axis accelerometer configured for a 2G range, where G is the acceleration due to gravity. It is possible to use the accelerometer to determine the current orientation of the Kinect.

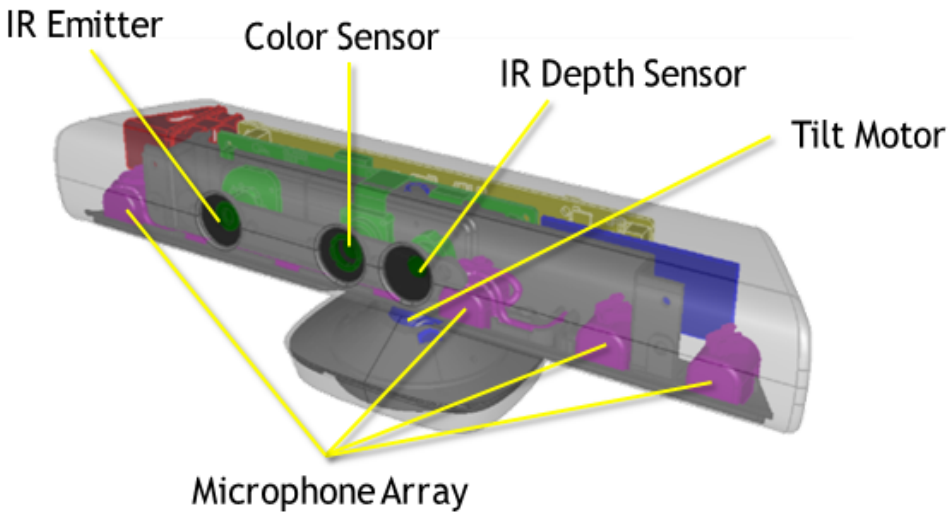

Figure 4.3 Schematic Diagram of Microsoft Kinect

4.2.4 RPI Camera

The Raspberry Pi camera board contains a 5 Mega Pixel sensor, and connects via a ribbon cable to the CSI connector on the Raspberry Pi. The video and still image quality is better than a USB webcam of similar price. The ”Pi NoIR” version of the camera was released is what we are using for the project. It has the same sensor with the IR filter removed, and a black PCB. With no IR filter, it can see near-IR wavelengths (700 - 1000 nm) like a security camera, with the tradeoff

of poor colour rendition. It is otherwise the same and uses the same software as the normal Pi camera.

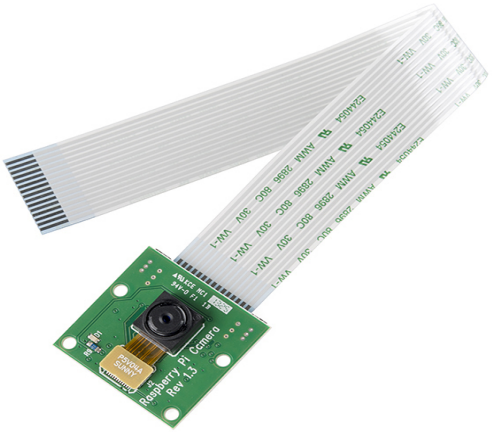

Figure 4.4 Raspberry-pi camera

4.3 BLOCK DIAGRAM

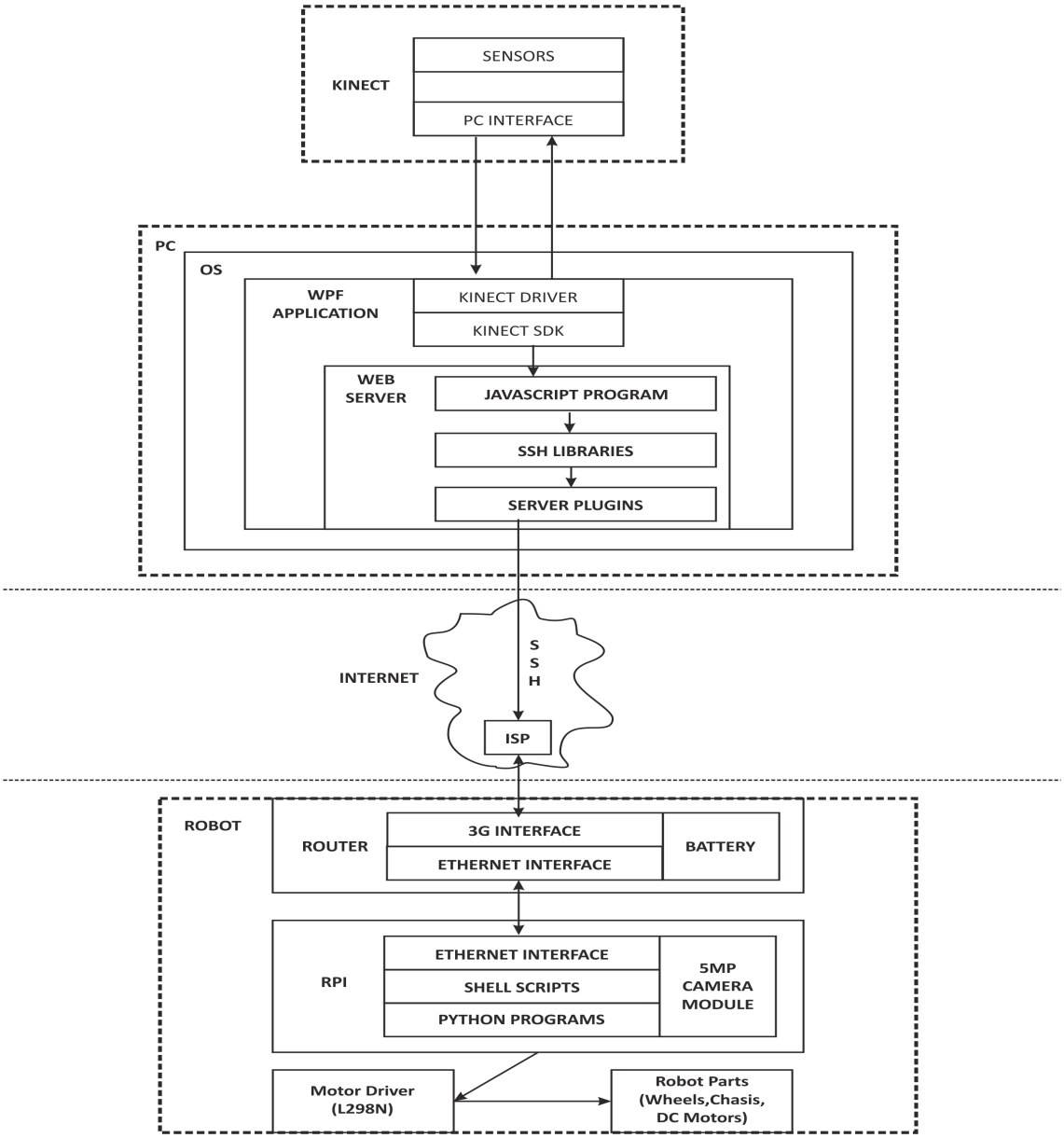

Figure 4.5 Block Digram

4.4 FLOW DIAGRAM

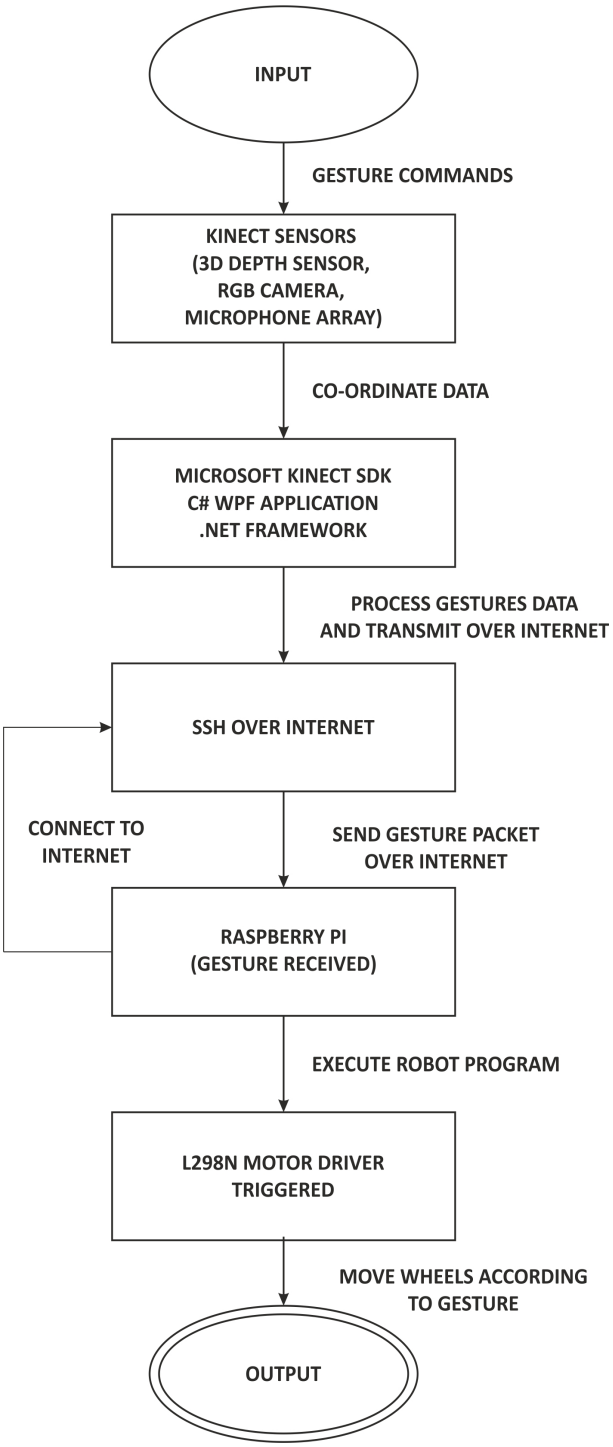

Figure 4.6 Flow Diagram from Input to Output

Description of Flow Diagram

The user provides a gesture or a voice command as input. This is

captured by the various sensors present on the Microsoft Kinect. The coordinates of the gestures are then processed by the SDK on the PC, to recognize the nature of the gesture i.e Swipe Left, Swipe Right etc. The confirmed gesture is then passed on to the WPF application which processes it and sends as gesture packets over the internet. These gesture packets are received by the Internet Web Server and sent to the Raspberry-pi client. At all times, the Raspberry -pi is connected to the internet to communicate with the web-server. Upon receiving the gesture, the Raspberry-pi executes the Robot program written in Python. This triggers the L2N8N motor driver which is responsible for controlling the wheels of the robot. Depending on the gesture, the robot moves in the desired direction. Thus the final output is the motion of the robot as directed by the user.

CHAPTER 5

SYSTEM DEVELOPMENT

5.1 INPUT AND OUTPUT TO SYSTEM

5.1.1 Input

Basic hand gestures are the input to the system. We have the following hand gestures

1) Swipe left
2) Swipe right
3) Swipe up
4) Swipe down
5) Circle

5.1.2 Output

The output of the system is an appropriate response from the robot to the gesture.

1) A swipe left gesture results in the robot moving to the left
2) A swipe right results in the robot moving to the right
3) A swipe up results in the robot moving forward
4) A swipe down results in the robot moving backward
5) A circle results in the robot moving in a circular manner

5.2 INPUT AND OUTPUT FOR EACH MODULE

5.2.1 Raspberry-pi controlled Robot

Input

The input to Raspberry Pi is the gesture packets from the internet.

Output

The output is the motion of the robot based on the Gestures.

Process

Code for Robot Control

Forward Movement

```
def forward(tf):
gpio.output(7,False)
gpio.output(11,True)
gpio.output(13,True)
gpio.output(15,False)
time.sleep(tf)
gpio.cleanup()
```

Backward Movement

```
def reverse(tf):
gpio.output(7,True)
gpio.output(11,False)
gpio.output(13,False)
gpio.output(15,True)
time.sleep(tf)
gpio.cleanup()
```

5.2.2 Kinect-PC module for gesture recognition

Input

The input here is the gestures like swipe left, swipe right, swipe up, swipe down, the ability to control the mouse and drawing a circle.

Output

The output is the gesture packets. These are stored in the system, after the program is run.

Process

Swipe to Right

```
{
if ((final vector.Y - initial vector.Y >maximum height) &&
(final vector.X - initial vector.X >0) &&
(swipe duration >minimum duration) &&
(swipe duration <maximum duration))
Swipe Right Gesture detected;
}
```

Swipe to Left

```
{
if ((final vector.Y - initial vector.Y >maximum height) &&
(final vector.X - initial vector.X <0) &&
(swipe duration >minimum duration) &&
(swipe duration >maximum duration))
Swipe Left Gesture detected;
}
```

5.2.3 Internet-Interfacing the Gesture and Voice Recognition Device with the Robot

Input

The input here is the gesture packets stored in the system.

Output

The output is the packets being transmitted over the internet through SSH to the robot, and the robot understanding each packet and performing the action

Process

Sample pseudo-code for SSH connection over C#

```
using (var client = new SshClient("hostnameOrIp", "username",
   "password"))
{
    client.Connect();
    client.RunCommand("etc/init.d/networking restart");
    client.Disconnect();
}
```

5.3 MODULES

5.3.1 Raspberry-pi controlled Robot

Since the aim is to build a prototype, the emphasis is not on a specific task the robot should accomplish. Hence, a simplistic robot with basic motor functions is built. The robot can navigate in all directions i.e left, right, front and back; capable of turning and manoeuvring on its path. We make use of an L298N motor driver. A full h bridge is used because we need to run four motors. A half bridge runs only 2 motors and L298N is more powerful than L298D (half bridge). We have 200 RPM DC motors which can rotate clockwise and anticlockwise. The RPI (raspberry pi) is set up by installing the Raspbian OS (flavour of Linux) and we dedicate an 8GB SD card to it as fast and better processing will be required in the future. We chose 512 MB ram also for the same reason. The GPIO pin is set up and so is the corresponding package bundle for the RPI. The remote desktop server

(Xbmc) and the SSH terminal is set up on the RPI. This is done so that the RPI can be remotely accessed via the DLink Wi-Fi adapter using the TP-Link wireless router. The Wi-Fi adapter is also installed with a powered USB hub because the RPI does not have enough power to directly handle a Wi-Fi adapter. This is set up and all the GPIO pins were connected to the motors. A corresponding python program is written to control the dc motors and rotate the wheels of the motor in the forward and backward directions. On completion, the program is modified to interface with the keyboard strokes by using the TKinter interface which identifies the keyboard strokes on press. Basic level of design includes communication of error free gesture signals to the computer which are subsequently forwarded through the Internet to the robot (raspberry-pi). At the first stage of development, primitive signals from the keyboard of the computer were successfully transmitted to the robot. These signals were correctly understood and the robot moved in the appropriate directions. For e.g on pressing A,W and D , the robot moved left, forward and right respectively. At the next stage of development, this was then connected to the laptop using the Microsoft remote desktop server by setting up a static IP in the RPI. The static IP is set up because of the constant change in IP every time it connects to the network. The static IP is used to remotely access the RPI and run the program on the laptop and use the laptop keys to move the robot. Directions like forward, backward, turn left, turn right, sharp left, sharp right have been programmed for specific key strokes. We are able to access the robot remotely from any location using mobile phone or pc. We have done this using VNC (virtual network controller) server and remote SSH (secure shell) connection. We first installed the VNC server and updated it. We then wrote a shell script that starts the VNC server as soon as the pi is turned on. We then configure the pi with a static

IP address over the internet. Initially, the IP address was dynamic and it kept changing every time the pi went on the internet. We solved this problem by using dynamic DNS from no-ip.com. This helped us assign dynamic DNS to the robot which we can use anytime. The VNC server makes use of port 5901 and SSH using port 22. We have moved our robot from other locations in Chennai, Bangalore, Delhi and Nigeria. At the final stage of development, a 5 megapixel NO IR camera was attached to the RPI camera slot. This was then updated and upgraded and the Camera module in the RPI was enabled so that we can take photos and videos from the Camera. A web based application was then created to stream the video from the RPI to the webpage. This was done using the MP4 conversion to stream lesser number of bytes over the internet for continuous streaming of the videos. The IP address of the RPI was put in the web browser for the video to start streaming. The video module code was written in Python on the RPI and this code gets triggered immediately after the booting of the RPI in the shell script. Thus the video is enabled and continuous MP4 videos are streamed to the browser over the internet.

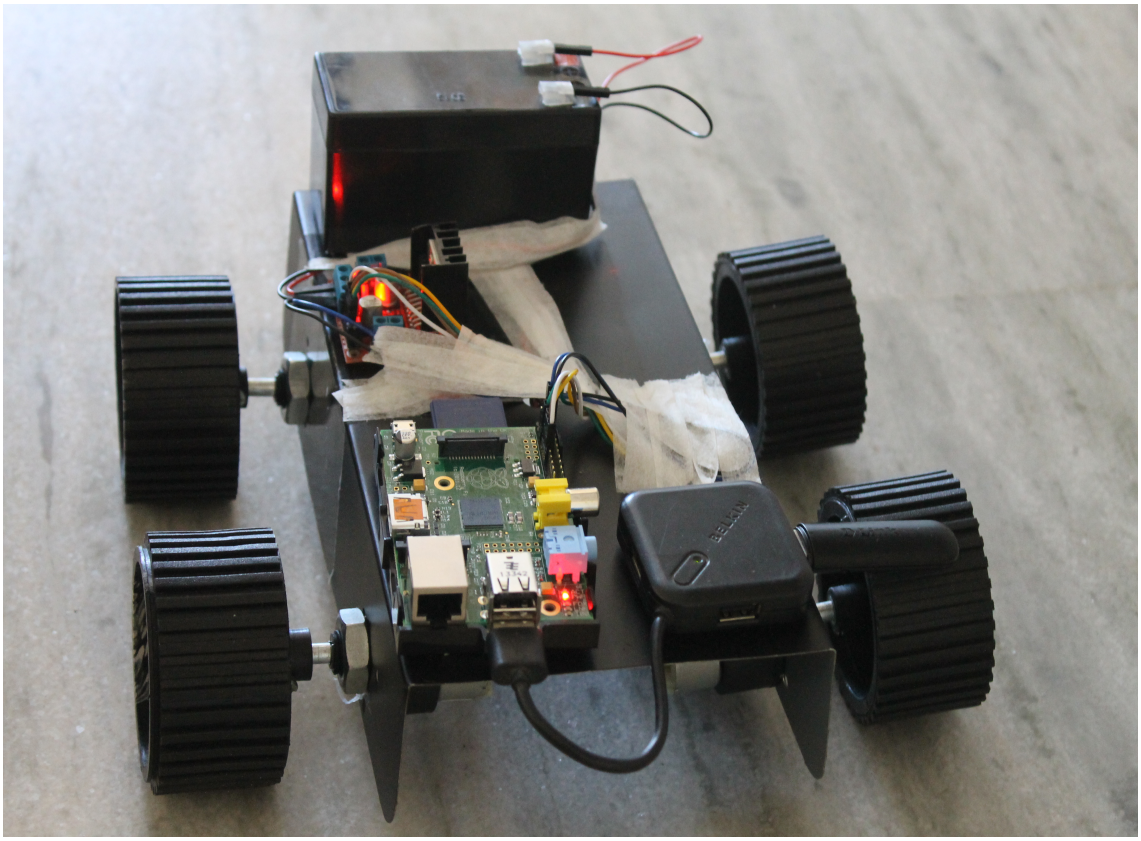

Figure 5.1 The Preliminary Robot which could Only be Moved through Computer or Phone

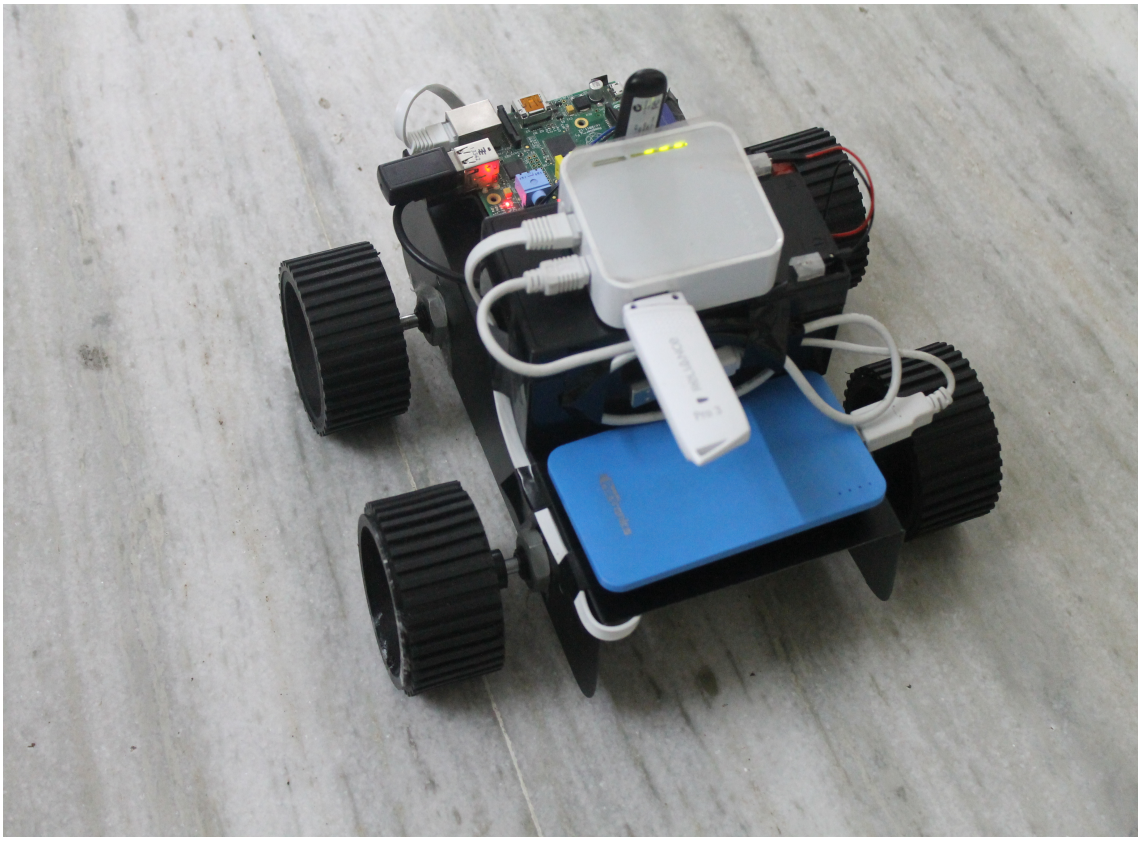

Figure 5.2 The Fully Functional Robot without the Camera

5.3.2 Kinect-PC module for gesture recognition

In order to control the robot, we require a gesture and voice recognition device. The Microsoft Kinect for Windows is the best suited device for our requirements. To recognise gestures from the Kinect, we purchased the Microsoft Kinect and downloaded the Kinect for Windows Software Development Kit (SDK). The SDK provides the tools and APIs, both native and managed, that we need to develop Kinect-enabled applications for Microsoft Windows. Developing Kinect-enabled applications is essentially the same as developing other Windows applications, except that the Kinect SDK provides support for the features of the Kinect, including colour images, depth images, audio input, and skeletal data. Programming the Kinect to recognise the gestures is done in C sharp programming language, in a .NET

framework. We then plan to make use of open source softwares like OpenKinect which will help us to use the Kinect?s hardware with other devices. We have gone in a step-wise manner while recognizing the gestures. We have recognized basic gestures like swipe left and swipe right. In order to achieve that, we have written an algorithm that takes each and every point on the hand and checks for the constraints. For the SwipeRight gesture, we will use constraints like:

1. Each new position should be to the right of the previous one
2. Each position must not exceed in height the first by more than a given distance (20 c

Similarly, we program the SwipeLeft gesture with another algorithm. Some of the other constraints in our program are that, the swipe left and swipe right gestures only work for the right hand. Moreover, to recognize the gesture accurately, the person must stand at a distance of at least 4 feet. After recognizing basic gestures, we have moved on to complex gestures. We are able to control the movement of the mouse using our left hand. We can basically move the mouse any where on the screen. To click, we draw a circle using the right hand. To double click, we need to draw two such circles. Thus we have been successful in recognizing even complex gestures.

Figure 5.3 Recognition of Skeleton using Kinect

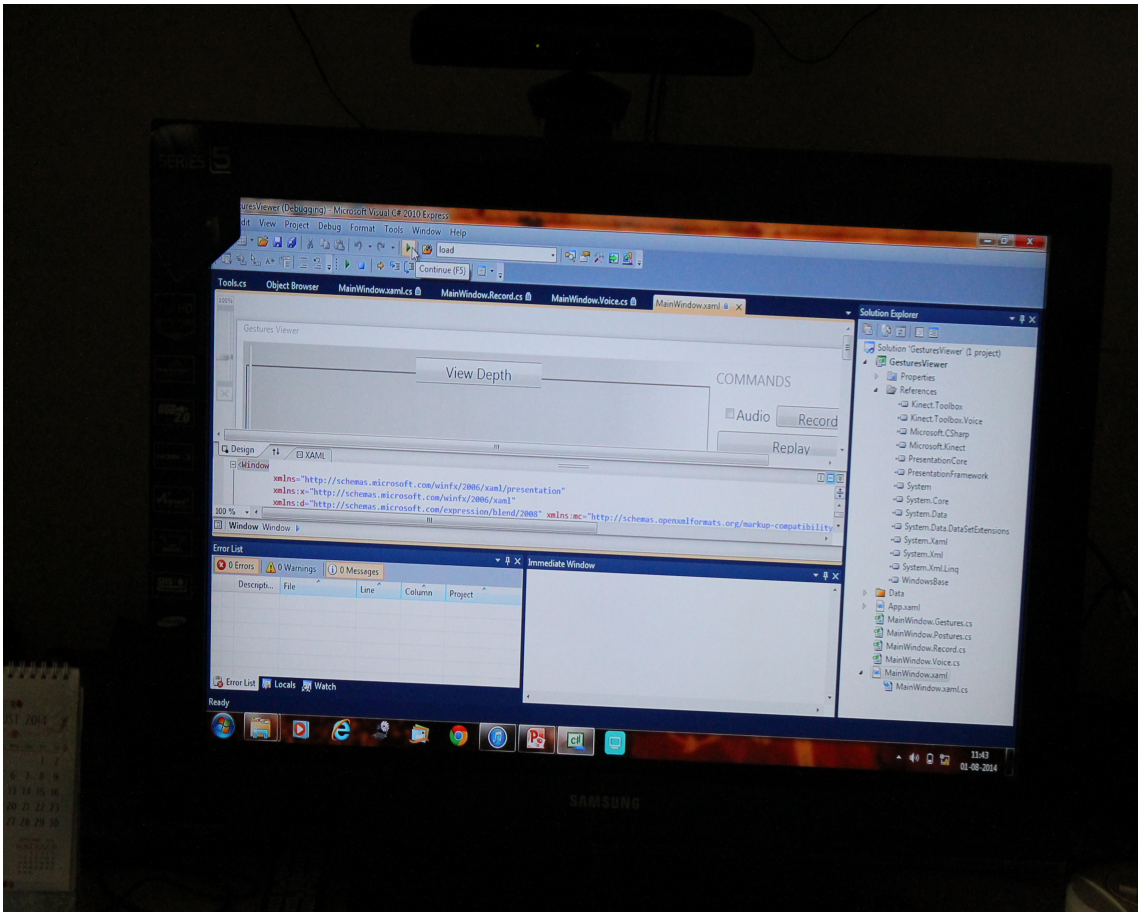

Figure 5.4 Programming for Kinect using C# WPF Application

5.3.3 Internet-Interfacing the Gesture and Voice Recognition Device with the Robot

The connectivity between the raspberry-pi and the Kinect can be established through the internet or bluetooth. This particular arrangement of connecting the raspberry-pi over the internet and making it interact with the Kinect is an example of the internet of things and has not been exploited to its absolute potential. Interaction between the Arduino and the Kinect has been carried out through various platforms like the internet and bluetooth, but this has not been carried forward to the raspberry-pi. Due to various power requirements and other factors most attempts at such an integration have been unsuccessful. For every gesture performed by the user in front of the kinect, a

separate code is written on robot that moves it according to the gesture. There are basically separate modules for each gesture.Once the gesture is recognized on the WPF(Windows Platform Framework) it is communicated using the SSH(Secure Shell Connection) over the internet to the robot. This is done using a noIP address, so that a static IP address is not required. Else, every time the robot connects to the internet, the IP address will change. Moreover, the SSH connection is persistent and open for all commands. Once the SSH connection has been established, we can perform all actions like SwipeRight, SwipeLeft etc and only upon program termination does the SSH connection close. This reduces time delay and improves performance. If the SSH connection was opened and closed for each individual gesture, it would be a serious bottleneck on the performance. Once the gesture packets reach the robot, it senses the command and executes the appropriate python program. Complex elliptical gestures like family of circles that make the robot rotate about itself are defined based on two simple previously defined gestures(swipe left immediately followed by a swipe right). Thus we have been able to successfully interface a gesture recognition device(Microsoft Kinect) and a robot controller(Raspberry-Pi) over the internet.

5.4 SOFTWARE IMPLEMENTATION FLOW

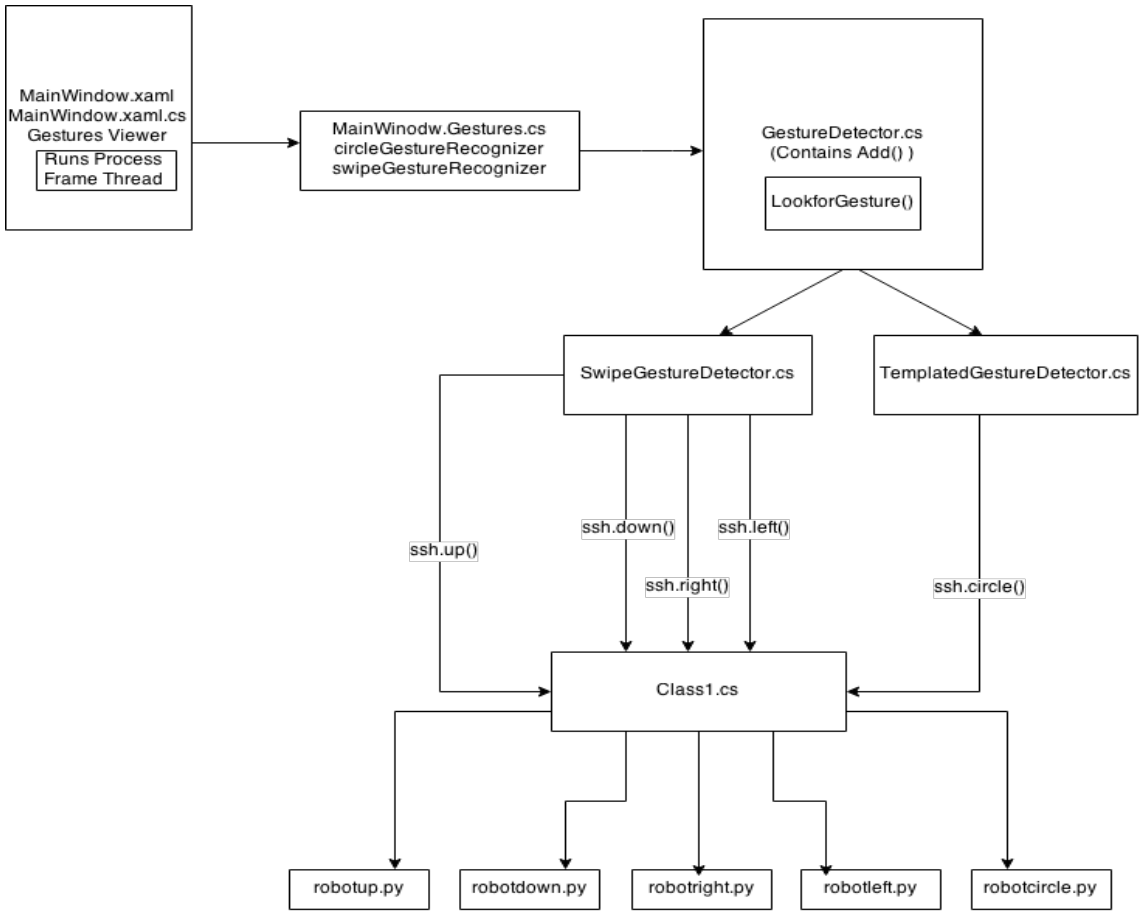

Figure 5.5 Software Flow Diagram

CHAPTER 6

RESULTS AND DISCUSSION

6.1 RESULTS

6.1.1 Assessment

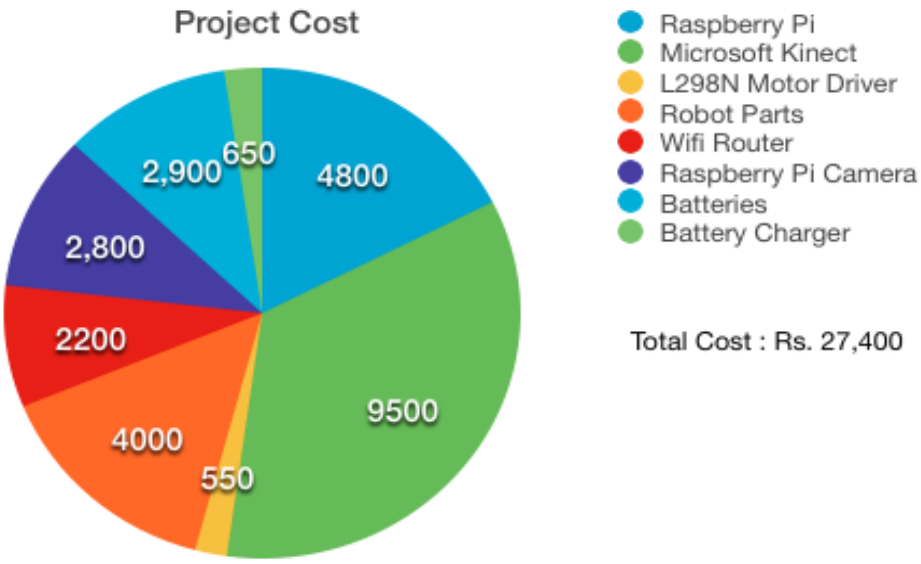

Figure 6.1 Project Cost for each Part

S.No	Test Case	Action	Result
1	SwipeRight	Swipe hand in the right direction	Robot moves in the right direction
2	SwipeLeft	Swipe hand in the left direction	Robot moves in the left direction
3	SwipeUp	Swipe hand in the upward direction	Robot moves in the forward direction
4	SwipeDown	Swipe hand in the downward direction	Robot moves in the backward direction
5	Circle	Swipe hand in a circular manner	Robot rotates
6	DiagonalSwipe	Swipe hand in a diagonal manner	Robot does not move
7	Left handed gesture	Any gesture made with the left hand	Robot does not move

Table 6.1 Test Cases and Results

6.1.2 Evaluation

There are several differences between the Internet-based remote control systems and other tele-operating systems. Most of the tele-operating systems are based on private media, by which the

transmission delay and data loss rate can be well modelled. The Internet in contrast is a public and shared resource in which various end users transmit data via the complex network simultaneously. The route for data transmission between two end points in a wide area is not fixed for different trials and the collision may be caused when two or more users transmit data via the same route simultaneously. Therefore the transmission efficiency of the Internet is difficult to be modelled at any time period.

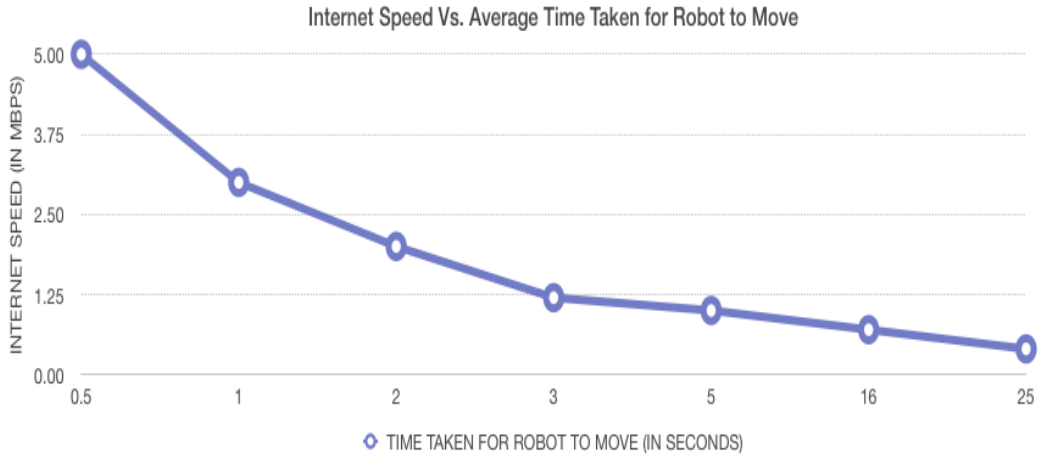

Figure 6.2 Time taken for Robot to Move based on Internet Speed

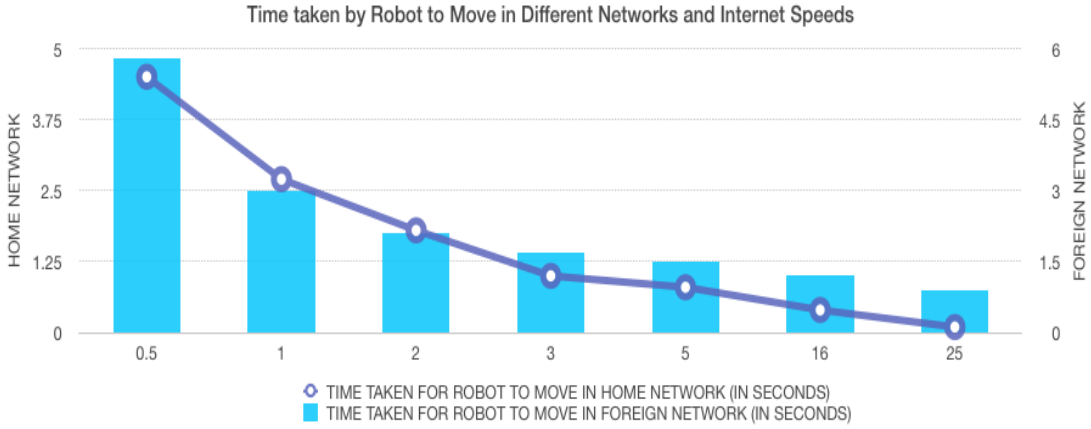

Figure 6.3 Time taken for Robot to Move in Different Networks based on Internet Speed

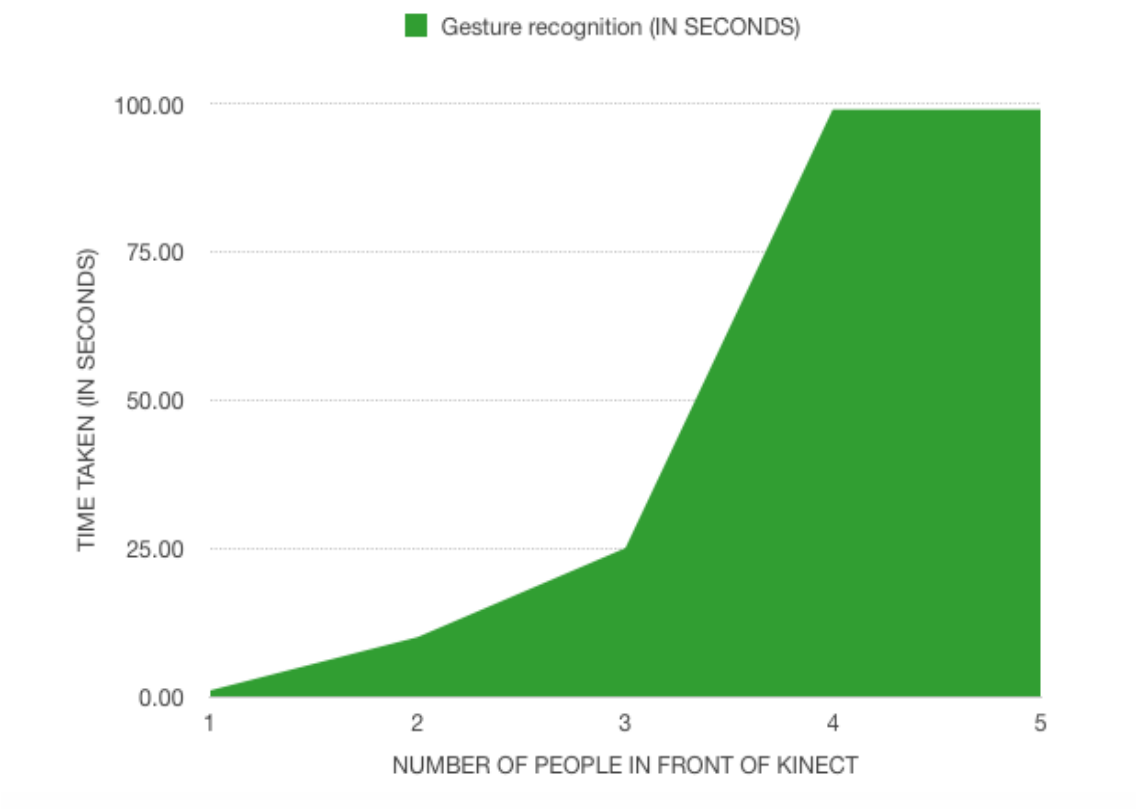

Figure 6.4 Time taken to Recognise Gesture based on Number of People in Front of Kinect

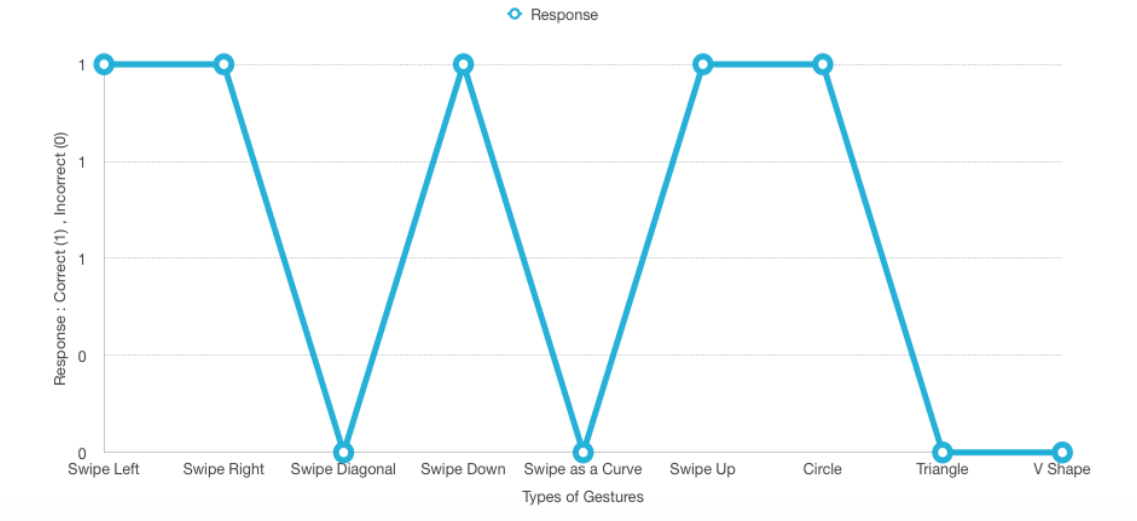

Figure 6.5 Response based on type of gesture

CHAPTER 7

CONCLUSIONS

7.1 CONTRIBUTIONS

This project establishes a method to integrate the various technologies that can be used for a host of applications in various fields. The interfacing of a flagship gesture recognition device like the Kinect with a low cost microcomputer like the RPI over the internet using SSH is a relatively new combination. This particular arrangement of connecting the raspberry-pi over the internet and making it interact with the Kinect is an example of the internet of things and has not been exploited to its absolute potential. Interaction between the Arduino and the Kinect has been carried out through various platforms like the internet and bluetooth, but this has not been carried forward to the raspberry-pi. Due to various power requirements and other factors most attempts at such an integration had been unsuccessful. We have successfully integrated the Kinect and the raspberry-pi. A research paper on the above mentioned arrangement was presented at the International Conference on Advances in Computing, Communication and Informatics. ICACCI is an IEEE conference conducted annually at different locations across the globe. Our research work was recognised at the Student Research Symposium of ICACCI-2014, Delhi. The same is now a published work of research (ISBN: 978-981-09-2229-0) in the Research Publishing Services, Singapore.The implementation of the above arrangement using latest equipment is a first of its kind.

7.2 FUTURE WORK

The basic arrangement can be extended to newer areas of application. The set of gestures defined can also be more specific to a particular task that the robot is designed to complete. More flexibility can be incorporated by allowing the user to define gesture-response pair. For example, the user can decide that the swipeRight action will make the robot move forward. The chassis and design of the robot is simple to incorporate the basic motion but can be modified according to requirements. For example, the replacement of DC motors with servo motors to enhance the manoeuvrability of the robot. It is possible to use the more advanced Raspberry-pi B+ version, rather than the B model, to reduce power consumption by the RPI. Moreover, with a bigger budget, the more Microsoft Kinect for Windows(Developer version) can be used instead of the one used currently. Better quality camera, with a higher megapixel value can be used for the robot. The Kinect can be programmed to receive gestures from multiple users simultaneously to perform complex tasks using the robot. Extending the same idea, in a more complex environment, multiple Kinects can be integrated to accommodate more users. Potential applications include but are not limited to defence, industrial appliances, smart home technology, technology for the differently abled individuals, adventure sports/gaming, telesurgery.

APPENDIX A

Secure Shell (SSH)

SSH uses public-key cryptography to authenticate the remote computer and allow it to authenticate the user, if necessary. There are several ways to use SSH; one is to use automatically generated public-private key pairs to simply encrypt a network connection, and then use password authentication to log on.

Another is to use a manually generated public-private key pair to perform the authentication, allowing users or programs to log in without having to specify a password. In this scenario, anyone can produce a matching pair of different keys (public and private). The public key is placed on all computers that must allow access to the owner of the matching private key (the owner keeps the private key secret). While authentication is based on the private key, the key itself is never transferred through the network during authentication. SSH only verifies whether the same person offering the public key also owns the matching private key. In all versions of SSH it is important to verify unknown public keys, i.e. associate the public keys with identities, before accepting them as valid. Accepting an attacker's public key without validation will authorize an unauthorized attacker as a valid user.

A.1 SAMPLE WORKING CODE FOR SSH

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Renci.SshNet;

namespace Kinect.Toolbox
{
    public static class ssh
    {
        public static void right()
        {

            {
                try
                {
                    using (var client = new
                       SshClient("192.168.0.104", "pi", "teddy"))
                    {
                        client.Connect();
                        client.RunCommand("sudo python
                           robotright.py");
                        client.Disconnect();
                    }


                }
```

```
        catch
        {

        }
    }
}
public static void left()
{

    {
        try
        {
            using (var client = new
               SshClient("192.168.0.104", "pi", "teddy"))
            {
                client.Connect();
                client.RunCommand("sudo python
                   robotleft.py");
                client.Disconnect();
            }


        }
        catch
        {

        }
    }
}
public static void circle()
{

    {
        try
        {
```

```
            using (var client = new
                SshClient("192.168.0.104", "pi", "teddy"))
            {
                client.Connect();
                client.RunCommand("sudo python
                    robotcircle.py");
                client.Disconnect();
            }


        }
        catch
        {

        }
    }
}
public static void up()
{

    {
        try
        {
            using (var client = new
                SshClient("192.168.0.104", "pi", "teddy"))
            {
                client.Connect();
                client.RunCommand("sudo python robotup.py");
                client.Disconnect();
            }


        }
        catch
        {
```

```
            }
        }
    }
    public static void down()
    {

        {
            try
            {
                using (var client = new
                   SshClient("192.168.0.104", "pi", "teddy"))
                {
                    client.Connect();
                    client.RunCommand("sudo python
                       robotdown.py");
                    client.Disconnect();
                }


            }
            catch
            {

            }
        }
    }
}
}
```

A.1.1 Sample Swipe Code

```
using System;
using Microsoft.Kinect;
```

```
namespace Kinect.Toolbox
{
    public class SwipeGestureDetector : GestureDetector
    {
        public float SwipeMinimalLength {get;set;}
        public float SwipeMaximalHeight {get;set;}
        public int SwipeMininalDuration {get;set;}
        public int SwipeMaximalDuration {get;set;}

        public SwipeGestureDetector(int windowSize = 20)
            : base(windowSize)
        {
            SwipeMinimalLength = 0.4f;
            SwipeMaximalHeight = 0.2f;
            SwipeMininalDuration = 250;
            SwipeMaximalDuration = 1500;
        }

        protected bool ScanPositions(Func<Vector3, Vector3, bool>
            heightFunction, Func<Vector3, Vector3, bool>
            directionFunction,
            Func<Vector3, Vector3, bool> lengthFunction, int
                minTime, int maxTime)
        {
            int start = 0;

            for (int index = 1; index < Entries.Count - 1; index++)
            {
                if (!heightFunction(Entries[0].Position,
                    Entries[index].Position) ||
                    !directionFunction(Entries[index].Position,
                    Entries[index + 1].Position))
                {
                    start = index;
                }
```

```
            if (lengthFunction(Entries[index].Position,
                Entries[start].Position))
            {
                double totalMilliseconds = (Entries[index].Time
                    - Entries[start].Time).TotalMilliseconds;
                if (totalMilliseconds >= minTime &&
                    totalMilliseconds <= maxTime)
                {
                    return true;
                }
            }
        }

        return false;
    }

    protected override void LookForGesture()
    {
        // Swipe to right
        if (ScanPositions((p1, p2) => Math.Abs(p2.Y - p1.Y) <
            SwipeMaximalHeight, // Height
            (p1, p2) => p2.X - p1.X > -0.01f, // Progression to
                right
            (p1, p2) => Math.Abs(p2.X - p1.X) >
                SwipeMinimalLength, // Length
            SwipeMininalDuration, SwipeMaximalDuration)) //
                Duration
        {
            RaiseGestureDetected("SwipeToRight");
            ssh.right();
            return;
        }

        // Swipe to left
```

```
if (ScanPositions((p1, p2) => Math.Abs(p2.Y - p1.Y) <
   SwipeMaximalHeight, // Height
    (p1, p2) => p2.X - p1.X < 0.01f, // Progression to
       right
    (p1, p2) => Math.Abs(p2.X - p1.X) >
       SwipeMinimalLength, // Length
    SwipeMininalDuration, SwipeMaximalDuration))//
       Duration
{
   RaiseGestureDetected("SwipeToLeft");
   ssh.left();
    return;
}
if (ScanPositions((p1, p2) => Math.Abs(p2.X - p1.X) <
   SwipeMaximalHeight, // Height
    (p1, p2) => p2.Y - p1.Y > -0.01f, // Progression to
       up
    (p1, p2) => Math.Abs(p2.Y - p1.Y) >
       SwipeMinimalLength, // Length
    SwipeMininalDuration, SwipeMaximalDuration)) //
       Duration
{
    RaiseGestureDetected("SwipetoUp");
    ssh.up();
    return;
}
if (ScanPositions((p1, p2) => Math.Abs(p2.X - p1.X) <
   SwipeMaximalHeight, // Height
    (p1, p2) => p2.Y - p1.Y < 0.01f, // Progression to
       down
    (p1, p2) => Math.Abs(p2.Y - p1.Y) >
       SwipeMinimalLength, // Length
    SwipeMininalDuration, SwipeMaximalDuration))//
       Duration
{
```

```
                RaiseGestureDetected("SwipeToDown");
                ssh.down();
                return;
            }
        }
    }
}
```

A.2 SAMPLE CIRCLE CODE

```
using System.Linq;
using System.IO;
using Microsoft.Kinect;

namespace Kinect.Toolbox
{
    public class TemplatedGestureDetector : GestureDetector
    {
        public float Epsilon { get; set; }
        public float MinimalScore { get; set; }
        public float MinimalSize { get; set; }
        readonly LearningMachine learningMachine;
        RecordedPath path;
        readonly string gestureName;

        public bool IsRecordingPath
        {
            get { return path != null; }
        }

        public LearningMachine LearningMachine
        {
            get { return learningMachine; }
        }
```

```
public TemplatedGestureDetector(string gestureName, Stream
    kbStream, int windowSize = 60)
    : base(windowSize)
{
    Epsilon = 0.035f;
    MinimalScore = 0.80f;
    MinimalSize = 0.1f;
    this.gestureName = gestureName;
    learningMachine = new LearningMachine(kbStream);
}

public override void Add(SkeletonPoint position,
    KinectSensor sensor)
{
    base.Add(position, sensor);

    if (path != null)
    {
        path.Points.Add(position.ToVector2());
    }
}

protected override void LookForGesture()
{
    if (LearningMachine.Match(Entries.Select(e => new
        Vector2(e.Position.X, e.Position.Y)).ToList(),
        Epsilon, MinimalScore, MinimalSize))
    {
        RaiseGestureDetected(gestureName);
        ssh.circle();
    }
}

public void StartRecordTemplate()
{
```

```
            path = new RecordedPath(WindowSize);
        }

        public void EndRecordTemplate()
        {
            LearningMachine.AddPath(path);
            path = null;
        }

        public void SaveState(Stream kbStream)
        {
            LearningMachine.Persist(kbStream);
        }
    }
}
```