Gesture based Rover and Appliance Control

Virtual Company Name: Sigma Six

Team Info:

CEO - Praveen Padala - praveenreddypadala@gmail.com

CTO – Tauha Khan - tauhakhan@gmail.com

CFO – Soumya Rama - soumya.guriginjakunta@gmail.com

Zankhana Rana - zankhanarana9@gmail.com

Raj Shah - rajpareshshah14@gmail.com

Manasa Tempalli - manasatempalli@gmail.com

Table of Contents:

1.Background and description of problem

Robots have the potential to support elderly and disabled people and make them feel independent in doing their daily household activities. There have been many assistive robots developed over time which have helped reaching the basic needs of elderly people. But statistically the usage of such systems has been found to be very less compared to other household robots (like vacuum cleaners, floor cleaners). This is due to the high technology implementation, complex equipment to build the service robot which makes such machinery extremely expensive. Also, such highly advanced equipment may not be enough user friendly because it is hard for elderly people to actually analyse and understand their working. Therefore, we have come up with an idea to develop a cost effective basic version of a service robot to help the disabled to pick light weighted items based on hand gesture recognition (implemented using image processing). The robot involves a rover connected with a robotic arm that moves in different directions to pick items based on the persons hand gesture. The functionality and efficiency of this implementation could be extended further to remotely access other daily home equipment. Controlling the robot using gesture recognition makes it helpful for the disabled to pick items, turn on and off equipment remotely.

2.Objective

The main objective of this project is to help the elderly and disabled people in leading a hassle-free life. This robot helps the elderly in overcoming the daily difficulties such as picking up items on the floor, picking water bottle, turn on lights, AC, TV etc. without having to move from where they are. Also making it hand gesture recognising equipment, makes it further easy for them without having to press any buttons or having to operate the entire equipment.

3.Methodology

The gesture recognition is achieved using convolutional neural networks which is explained in detail in further sections. The robotic movement and arm controls are handled using a Raspberry pi. Also, the networking of the pi with the gesture recognition is done using the wifi connect in the pi. The communication is done using a local PAN of the two pi's. Also, a bulb switch controlling operation is achieved using a third pi. The overall circuitry and understanding of the embedded dev board is clearly explained in the further sections.

4. Embedded Development Board

Development and prototype boards are computer boards that are used to develop or test electronic modules. These are used to evaluate programs for embedded devices such as controllers, point-of-sale (PoS) terminals, kiosks and information appliances.

Development boards combine a processor, chipset, memory and onboard peripherals with debugging features. Specifications for development boards include bus type, processor type, form factor, number of ports, port type, memory and operating system. Raspberrypi 3 model has been using our implementation.

4.1 Reasons for choosing Rpi3

- This development board features a SoC, with an integrated ARM, compatible CPU and on Chip GPU
- It is an open source platform
- Promotes python and scratch programming
- It has on board Wifi, Bluetooth and USB boot capabilities.
- It is compatible with multiple operating systems such as Linux, Windows, IOT Core, RISC OS
- Low cost
- Low power
- High availability
- High reliability – Tested over millions of Raspberry Pis Produced to date – Module IO pins have 35u hard gold plating

4.2 Key Applications

Low cost PC/tablet/laptop , IoT applications , Media centre Robotics
• Industrial/Home automation
• Server/cloud server, Print server
• Security monitoring, Web camera
• Gaming
• Wireless access point
• Environmental sensing/monitoring (e.g. weather station)
Low cost PC/tablet/laptop , IoT applications , Media centre Robotics

4.3 Technical Specifications

Peripherals

• 48x GPIO

• 2x I2C

• 2x SPI

• 2x UART

• 2x SD/SDIO

• 1x HDMI 1.3a

• 1x USB2 HOST/OTG

• 1x DPI (Parallel RGB Display)

• 1x NAND interface (SMI)

• 1x 4-lane CSI Camera Interface (up to 1Gbps per lane)

• 1x 2-lane CSI Camera Interface (up to 1Gbps per lane)

• 1x 4-lane DSI Display Interface (up to 1Gbps per lane)

• 1x 2-lane DSI Display Interface (up to 1Gbps per lane) 2.3 Software

• ARMv6 (CM1) or ARMv7 (CM3, CM3L) Instruction Set

• Mature and stable Linux software stack – Latest Linux Kernel support – Many drivers upstreamed – Stable and well supported userland – Full availability of GPU functions using standard API

4.4 Electrical Specification

Symbol	Parameter	Minimum	Maximum	Unit
VBAT	Core SMPS Supply	-0.5	6.0	V
3V3	3V3 Supply Voltage	-0.5	4.10	V
1V8	1V8 Supply Voltage	-0.5	2.10	V
VDAC	TV DAC Supply	-0.5	4.10	V
GPIO0-27_VDD	GPIO0-27 I/O Supply Voltage	-0.5	4.10	V
GPIO28-45_VDD	GPIO28-27 I/O Supply Voltage	-0.5	4.10	V
SDX_VDD	Primary SD/eMMC Supply Voltage	-0.5	4.10	V

4.5 DC Characteristics

Symbol	Parameter	Conditions	Minimum	Typical	Maximum	Unit
V_{IL}	Input low voltage[a]	VDD_IO = 1.8V	-	-	0.6	V
		VDD_IO = 2.7V	-	-	0.8	V
V_{IH}	Input high voltage[a]	VDD_IO = 1.8V	1.0	-	-	V
		VDD_IO = 2.7V	1.3	-	-	V
I_{IL}	Input leakage current	TA = +85°C	-	-	5	µA
C_{IN}	Input capacitance	-	-	5	-	pF
V_{OL}	Output low voltage[b]	VDD_IO = 1.8V, IOL = -2mA	-	-	0.2	V
		VDD_IO = 2.7V, IOL = -2mA	-	-	0.15	V
V_{OH}	Output high voltage[b]	VDD_IO = 1.8V, IOH = 2mA	1.6	-	-	V
		VDD_IO = 2.7V, IOH = 2mA	2.5	-	-	V
I_{OL}	Output low current[c]	VDD_IO = 1.8V, VO = 0.4V	12	-	-	mA
		VDD_IO = 2.7V, VO = 0.4V	17	-	-	mA
I_{OH}	Output high current[c]	VDD_IO = 1.8V, VO = 1.4V	10	-	-	mA
		VDD_IO = 2.7V, VO = 2.3V	16	-	-	mA
R_{PU}	Pullup resistor	-	50	-	65	kΩ
R_{PD}	Pulldown resistor	-	50	-	65	kΩ

4.6 AC Characteristics

Pin Name	Symbol	Parameter	Minimum	Typical	Maximum	Unit
Digital outputs	t_{rise}	10-90% rise time[a]	-	1.6	-	ns
Digital outputs	t_{fall}	90-10% fall time[a]	-	1.7	-	ns
GPCLK	t_{JOSC}	Oscillator-derived GPCLK cycle-cycle jitter (RMS)	-	-	20	ps
GPCLK	t_{JPLL}	PLL-derived GPCLK cycle-cycle jitter (RMS)	-	-	48	ps

5.Gesture Detection

5.1 Convolutional Neural Networks

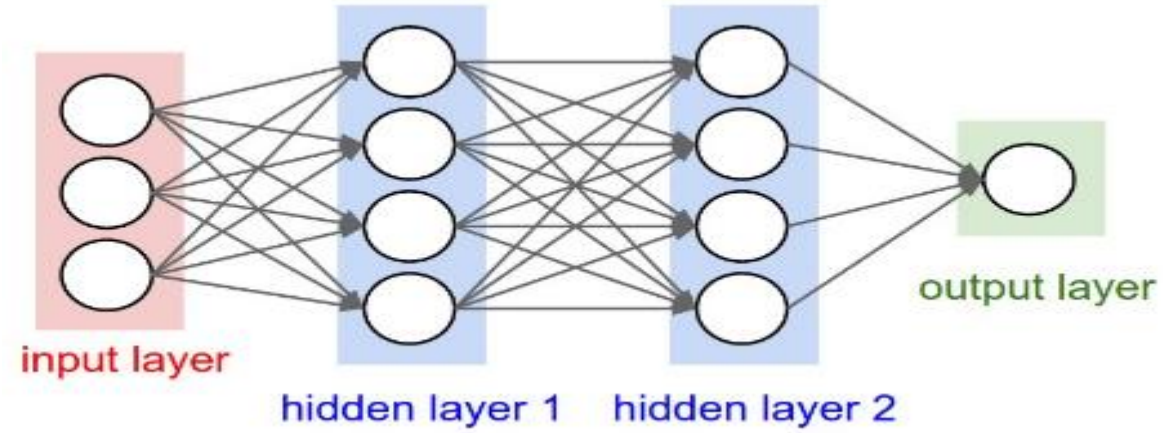

Fig1. Neural Networks

In machine learning, a convolution neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. It's a branch of artificial Neural Networks which is specifically used for image processing applications.
Convolutional Neural network is state of the art technique which tries to work in a similar fashion as human vision.

5.2 Architecture

A convolutional neural network is basically an arrangement of layers. every layer performs a set of actions that are transferred to another through a differentiable function. Convolution networks mainly comprises of three layers: Convolution layer, Pooling Layer, Fully connected Layer which are bundled upon each other to form a full Convolution network architecture. These layers which are stacked upon each other may or may not have different parameters. Every Layer in this network accepts input 3D volume and transforms it into an output 3D volume.

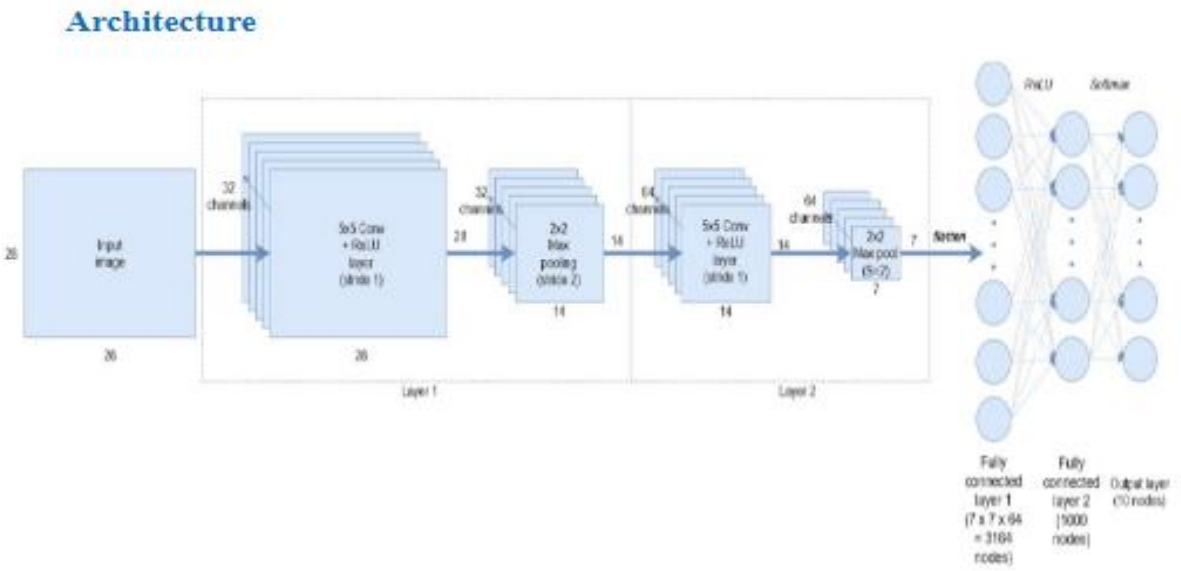

Fig 2. Architecture of convolutional neural networks

Feature maps learns about different features of the image by using convolution max pooling. Two Fully connected layer. Softmax function to get probabilities.

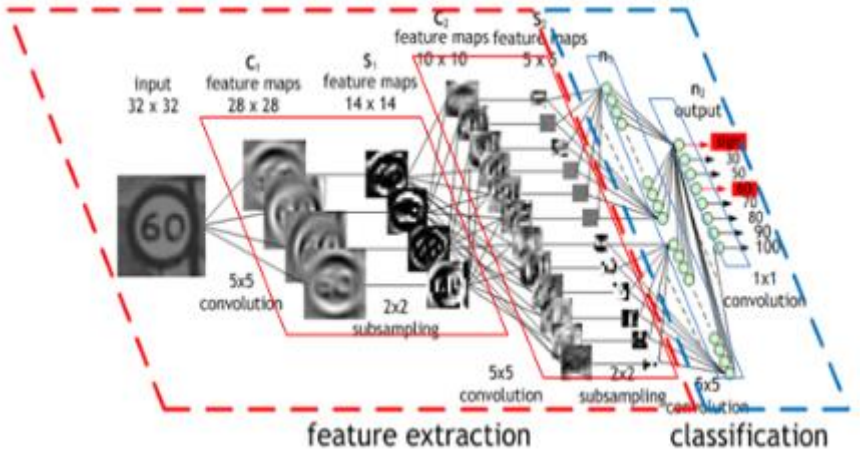

Fig 3. Feature extraction and classification of images

Traditional Methods using computer vision not very efficient and complex.

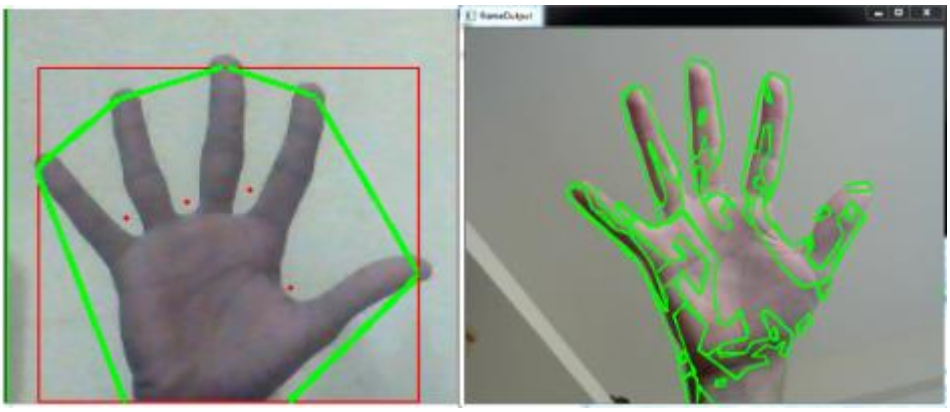

Fig 4. Problem with traditional methods

5.3 Hand-Gesture-Recognition-Using-CNN

Hand Gesture Recognition using Convolution Neural Networks Datacollection.py is used for collecting train data and test data. It takes 1000 images of each gestures and stores them as training set

The CNN model is trained and saved by using Traingest.py Meta and index file are created which are used in predictgest.py

Predictgest.py is used for real time prediction of the gestures. There are a total of 10 Different Gestures that are trained. You can see the different gestures in Training set. The Gestures corresponds to numbers 0-9.

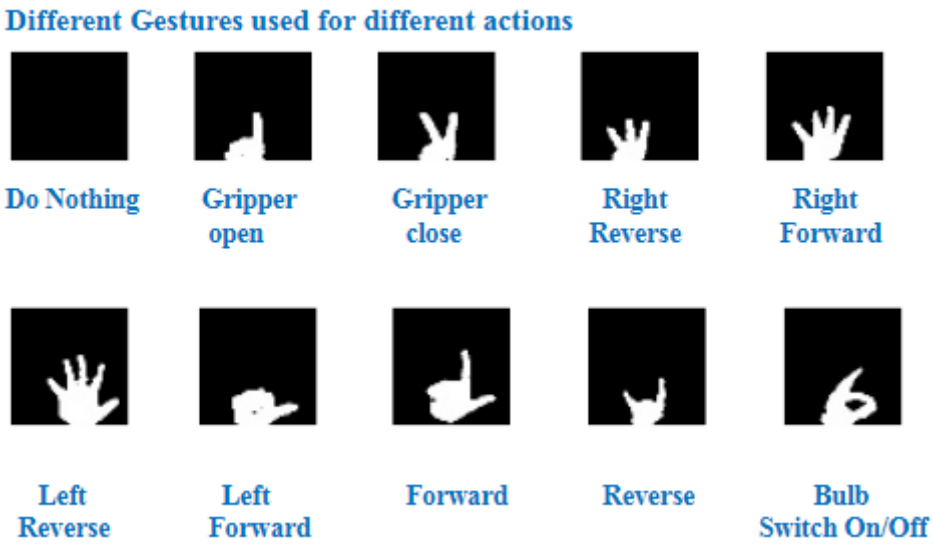

Fig 5. Working of hand gestures and corresponding function

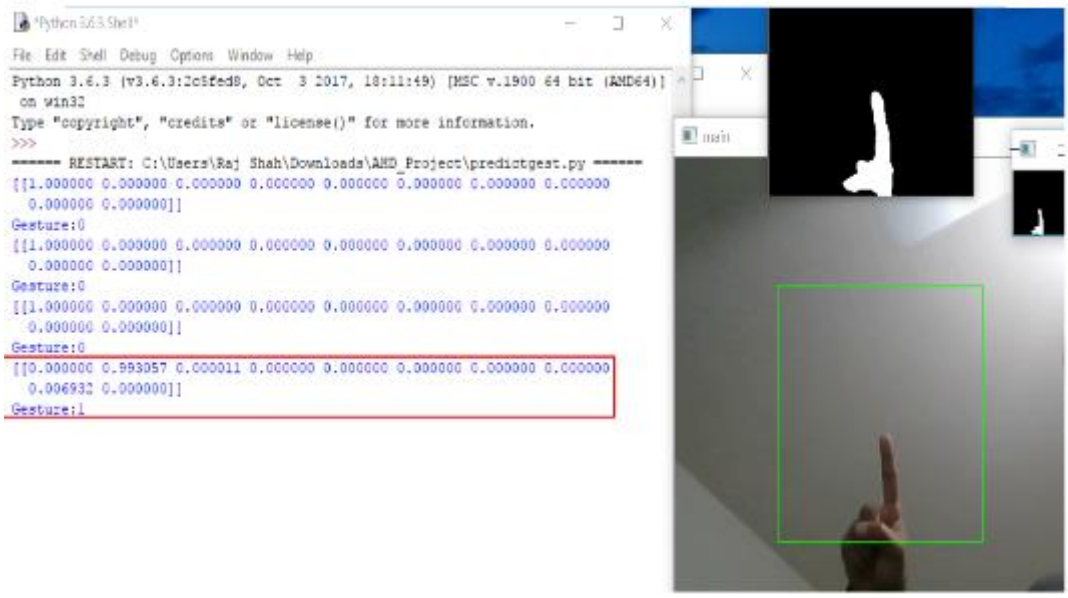

Fig 6. Woking module

6. Socket Connection

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.

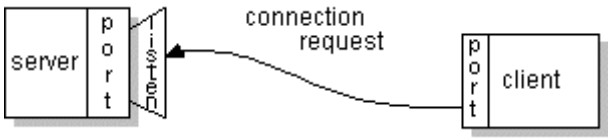

Fig 7. Server-client connection request

If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.

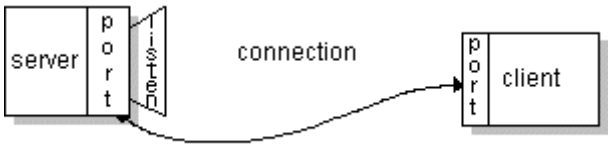

Fig 8. Server-client connection request

On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets. An endpoint is a combination of an IP address and a port number. Every TCP connection can be uniquely identified by its two endpoints. That way you can have multiple connections between your host and the server.

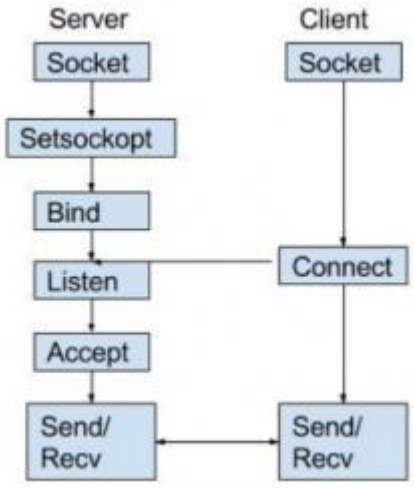

Fig 9. Flow of request

6.1 Current implementation

Socket connection explained in the above section is used in our project to communicate the gesture recognition results and the received signal at the motor between the Rpi's. The code implementation is in the next section

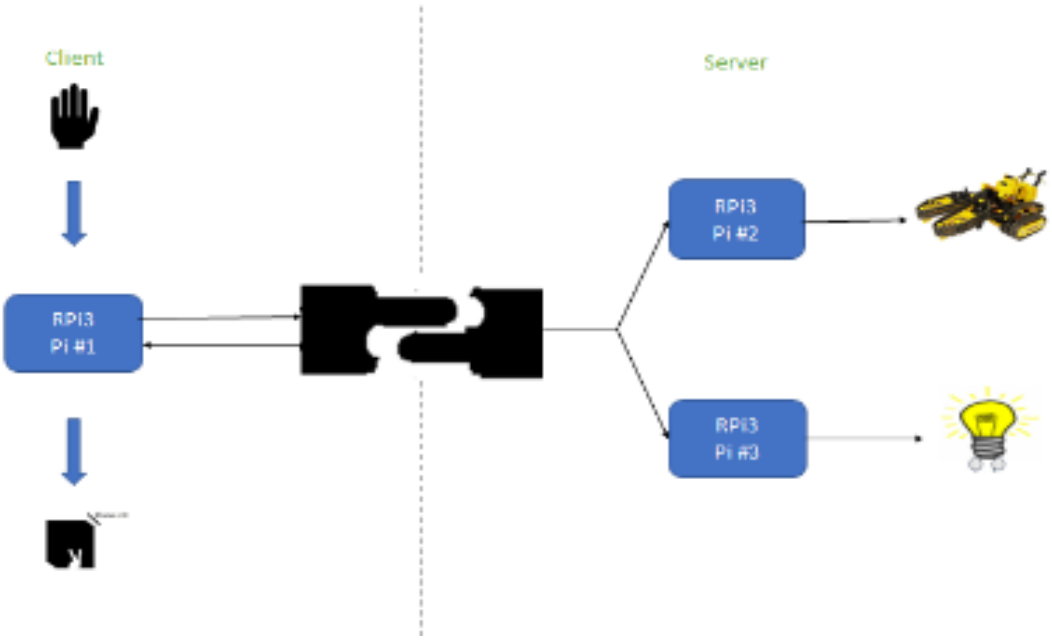

Fig 10. Current implementation

Receive code

```
#import socket
#
#UDP_IP = "192.168.1.98"
#
#UDP_PORT = 5006
#
#sock = socket.socket(socket.AF_INET, # Internet
#
#socket.SOCK_DGRAM) # UDP
#
#sock.bind((UDP_IP, UDP_PORT))
#
#while True:
#  data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
#  print "received message:", data

import RPi.GPIO as GPIO
import socket
import time



def init():
   GPIO.setmode(GPIO.BOARD)

   GPIO.setup(37,GPIO.OUT)

init()
GPIO.output(37,False)
```

```
def relay(r_flag):
   print r_flag
   init()
   if(r_flag==0):
     GPIO.output(37,True)
     print("Bulb On")
   elif(r_flag==1):
     GPIO.output(37,False)
     print("Bulb Off")



UDP_IP = "192.168.1.98"

UDP_PORT = 5006

UDP_IP_sender = "192.168.1.80"



sockrecv = socket.socket(socket.AF_INET, # Internet
socket.SOCK_DGRAM) # UDP


socksend = socket.socket(socket.AF_INET, # Internet
socket.SOCK_DGRAM) # UDP

sockrecv.bind(('', UDP_PORT))

#socksend.bind((UDP_IP_sender, UDP_PORT_reply))
r_flag=0
while True:
   send_flag = 0
   pi2_flag=0
   data, addr = sockrecv.recvfrom(1024) # buffer size is 1024 bytes
   print "received message:", data
   if data == '9':
     pi2_flag = 1
     print "Turning Bulb on"
     relay(r_flag)
     if(r_flag==0):
       r_flag = 1
     else:
       r_flag = 0

   else:
     print "pass"
   if(pi2_flag):
     sockrecv.sendto(str(UDP_IP), (UDP_IP_sender, UDP_PORT))
     send_flag=1
   if(send_flag == 1):
     print "sent"
```

Controlling motors

```
import RPi.GPIO as GPIO
import socket
import time



def init():
```

```
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(11,GPIO.OUT)
    GPIO.setup(13,GPIO.OUT)
    GPIO.setup(38,GPIO.OUT)
    GPIO.setup(40,GPIO.OUT)
    GPIO.setup(21,GPIO.OUT)
    GPIO.setup(23,GPIO.OUT)
    GPIO.setup(37,GPIO.OUT)

def m1m2rev(tf):
    init()
    GPIO.output(11,True)
    GPIO.output(13,False)
    GPIO.output(38,True)
    GPIO.output(40,False)
    GPIO.output(21,False)
    GPIO.output(23,False)
    time.sleep(tf)
    GPIO.cleanup()

def m1m2fwd(tf):
    init()
    GPIO.output(11,False)
    GPIO.output(13,True)
    GPIO.output(38,False)
    GPIO.output(40,True)
    GPIO.output(21,False)
    GPIO.output(23,False)
    time.sleep(tf)
    GPIO.cleanup()

def m2rev(tf):
    init()
    GPIO.output(11,True)
    GPIO.output(13,False)
    GPIO.output(38,False)
    GPIO.output(40,False)
    GPIO.output(21,False)
    GPIO.output(23,False)
    time.sleep(tf)
    GPIO.cleanup()

def m2fwd(tf):
    init()
    GPIO.output(11,False)
    GPIO.output(13,True)
    GPIO.output(38,False)
    GPIO.output(40,False)
    GPIO.output(21,False)
    GPIO.output(23,False)
    time.sleep(tf)
    GPIO.cleanup()
def m1rev(tf):
    init()
    GPIO.output(38,True)
    GPIO.output(40,False)
    GPIO.output(11,False)
    GPIO.output(13,False)
    GPIO.output(21,False)
    GPIO.output(23,False)
    time.sleep(tf)
    GPIO.cleanup()
```

```
def m1fwd(tf):
   init()
   GPIO.output(38,False)
   GPIO.output(40,True)
   GPIO.output(11,False)
   GPIO.output(13,False)
   GPIO.output(21,False)
   GPIO.output(23,False)
   time.sleep(tf)
   GPIO.cleanup()
def m3fwd(tf):
   init()
   GPIO.output(21,True)
   GPIO.output(23,False)
   GPIO.output(11,False)
   GPIO.output(13,False)
   GPIO.output(38,False)
   GPIO.output(40,False)
   time.sleep(tf)
   GPIO.cleanup()

def m3rev(tf):
   init()
   GPIO.output(21,False)
   GPIO.output(23,True)
   GPIO.output(11,False)
   GPIO.output(13,False)
   GPIO.output(38,False)
   GPIO.output(40,False)
   time.sleep(tf)
   GPIO.cleanup()

UDP_IP = "192.168.1.87"
UDP_PORT = 5006

UDP_IP_sender = "192.168.1.80"
#UDP_PORT_reply = 6789


sockrecv = socket.socket(socket.AF_INET, # Internet
socket.SOCK_DGRAM) # UDP


socksend = socket.socket(socket.AF_INET, # Internet
socket.SOCK_DGRAM) # UDP

sockrecv.bind((UDP_IP, UDP_PORT))

#socksend.bind((UDP_IP_sender, UDP_PORT_reply))
r_flag=0
while True:
   send_flag = 0
   pi1_flag=1
   data, addr = sockrecv.recvfrom(1024) # buffer size is 1024 bytes
   print "received message:", data
   if data == '1':
     print "Gripper open"
     m3fwd(4)
   elif data== '2':
     print "Gripper Close"
     m3rev(3)
   elif data == '3':
     print "Right rev"
```

```
  m1rev(0.5)
elif data == '4':
  print "Right fwd"
  m1fwd(0.5)
elif data == '5':
  print "Left rev"
  m2rev(0.5)
elif data == '6':
  print "Left fwd"
  m2fwd(0.5)
elif data == '7':
  print "Both Fwd"
  m1m2fwd(5)
elif data == '8':
  print "Both Rev"
  m1m2rev(5)
elif data == '1':
  print "pi 2 working"
  pi1_flag = 0
else:
  print "pass"
if(pi1_flag):
  sockrecv.sendto(str(UDP_IP), (UDP_IP_sender, UDP_PORT))
  send_flag=1
if(send_flag == 1):
  print "sent"
```

7. Rover and Appliance control

The rover with gripper module consist of 3 DC motors that help it traverse along the ground and lift objects using the gripper. Motors M1 and M2, based on the input signals, allows the rover to move left, right, and back and forth. Motor M3, based on the input signal, opens and closes the gripper.
An L298 Motor drive controller is used to control the motors. The driver ties in the input signals from RPI3. The motor drive module controls the speed and direction of the motors.
The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time.

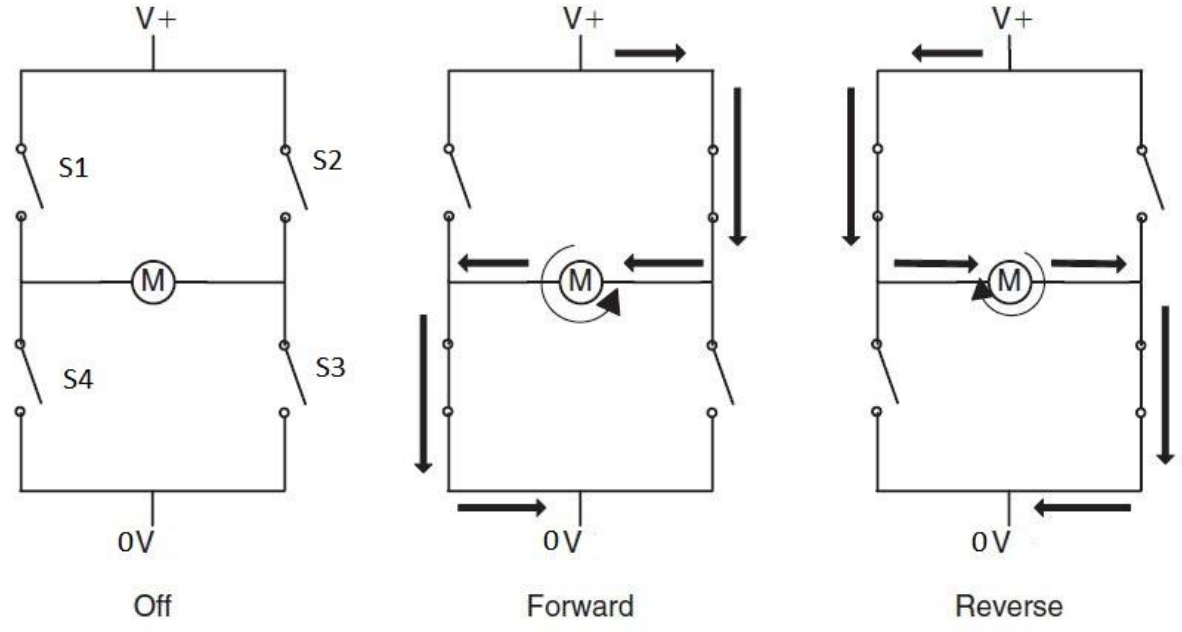

Fig 11. H-Bridge

Enable	Input 1	Input 2	Motor Direction
1	1 (S2,S4=close)	0 (S3,S4=open)	Forward Rotation
1	0 (S2,S4=open)	1 (S3,S4=close)	Reverse Rotation
1	0 (S3,S4=open)	0 (S3,S4=open)	Standstill
0	X	X	Standstill

Following is the table that shows the gesture commands and the corresponding control actions for motor, and the output that follows:

Gesture signal	Motor 1 (Gripper)	Motor 2 (Left)	Motor 3(Right)	Output action
Gesture 1	Open	0ff	Off	Gripper open
Gesture 2	Close	Off	Off	Gripper close
Gesture 3	Off	Rev	Off	Left reverse
Gesture 4	Off	Fwd	Off	Left forward
Gesture 5	Off	Off	Rev	Right Reverse
Gesture 6	Off	Off	Fwd	Right forward
Gesture 7	Off	Fwd	Fwd	Both forward
Gesture 8	Off	Rev	Rev	Both reverse

7.1 Controlling a remote appliance

- The control signal for turning on/off the light bulb comes from the RPI3 GPIO pin. This signal is based on a certain hand gesture.
- The bulb needs 120V supply to turn on. We use a 5V DC mechanical relay which takes the input signal at Pin ‘IN’ of relay module. Based on this input signal the bulb will turn on or off.
- The phase terminal of the AC source is connected to the ‘COM’ pin. The neutral terminal of AC source is connected to negative terminal the light bulb.
- In our case we use NO (normally open) contact to connect to phase terminal of the light bulb.
- When the ‘IN’ signal is high, the NO contact closes, providing 120V supply to bulb, thus completing the circuit and turning on the bulb.

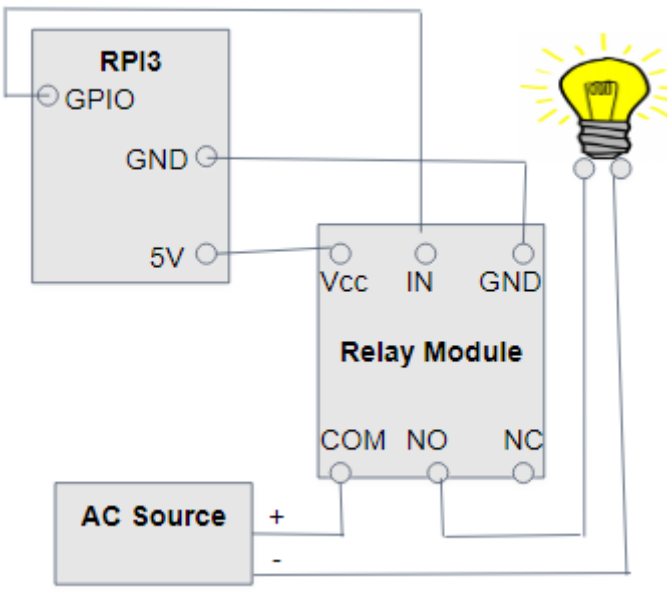

Fig 12. External devide control remotely

8. Conclusions

The basic set up for the actual implementation is achieved. The hand gesture recognition was successfully implemented. Also, the integration of hand gesture recognition with the motor functionality was made a success. Although this is a small scale implementation, the idea could be further extended with much efficient resources so as to achieve real time goals (much faster control of the robot and much more efficacy of the bot to carry heavier objects and control of entire house hold devices or any required device as such).

9. References

- Gesture Recognition Market Analysis By Technology (2D, 3D), By Application (Tablets & Notebooks, Smartphones, Gaming Consoles, Smart Televisions, Laptops & Desktops) And Segment Forecasts, 2014 - 2024
- http://cs231n.github.io/convolutional-networks/
- https://cs.nju.edu.cn/wujx/paper/CNN.pdf
- https://github.com/Hvass-Labs/TensorFlow-Tutorials
- https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html
- https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf

Attachments

- The file containing the code for gesture recognition using CNN is attached in the document folder