Introduction

This project is about developing a wirelessly controlled Robot based on Raspberry pi single board computer. This project has the following main functionality :

- Accepting wireless movement controls from client app in real-time for robot movements.
- Transmitting video feed to the client application in real-time.
- Digital connectivity with HD video transmission.

Purpose of the project :

- **Rescue Missions :** This robot can be used by NDRF teams to rescue people during a natural disaster for example: Rescuing people from the rubles of a collapsed building after a massive earthquake.
- **Recon mission :** This robot can be used by Indian army to carry out recon missions to locate enemies for example : Indian army can use this robot to carry out recon mission during close quarter combat in urban environment and locate the enemies from relatively safe distance.
- **Mobile home surveillance :** This robot can be used by people who want to keep an eye on their pets while they are away from home.

Tech Stack used :

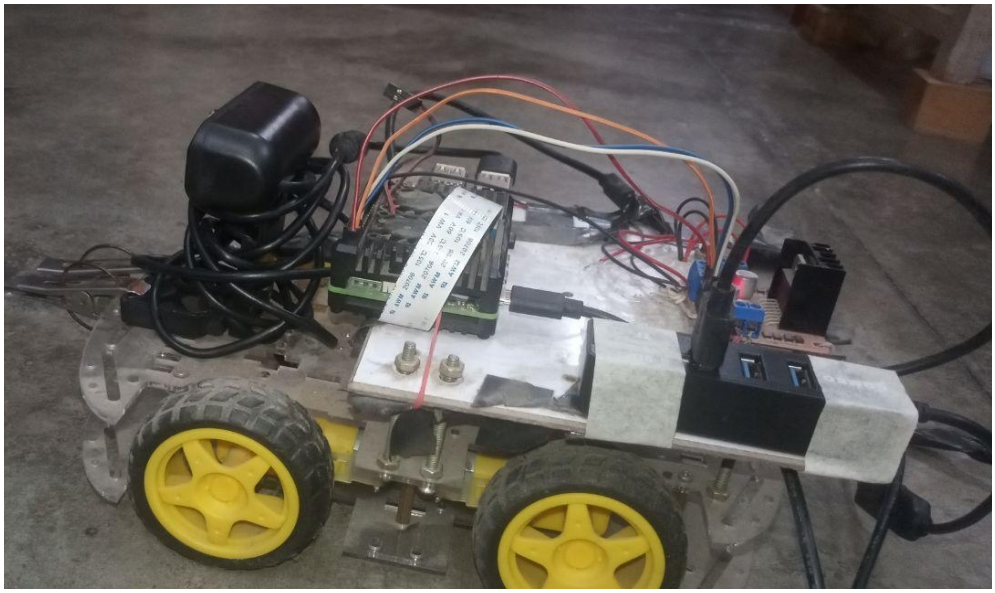
Hardware :

- Raspberry pi
- Jumper wires
- H bridge motor controller LN298 IC
- Camera module
- Electric DC motors
- Robot Chassis , switches, multi-power delivery usb system
- Power supply

Software :

- Atom IDE (Ubuntu Linux workstation)

- Python programming language
- Libraries used
- Linux OS (Raspbian OS)
- Geany IDE (Raspberry pi)
- Only Office for documentation (Ubuntu Linux workstation)



Hardware

Raspberry pi :

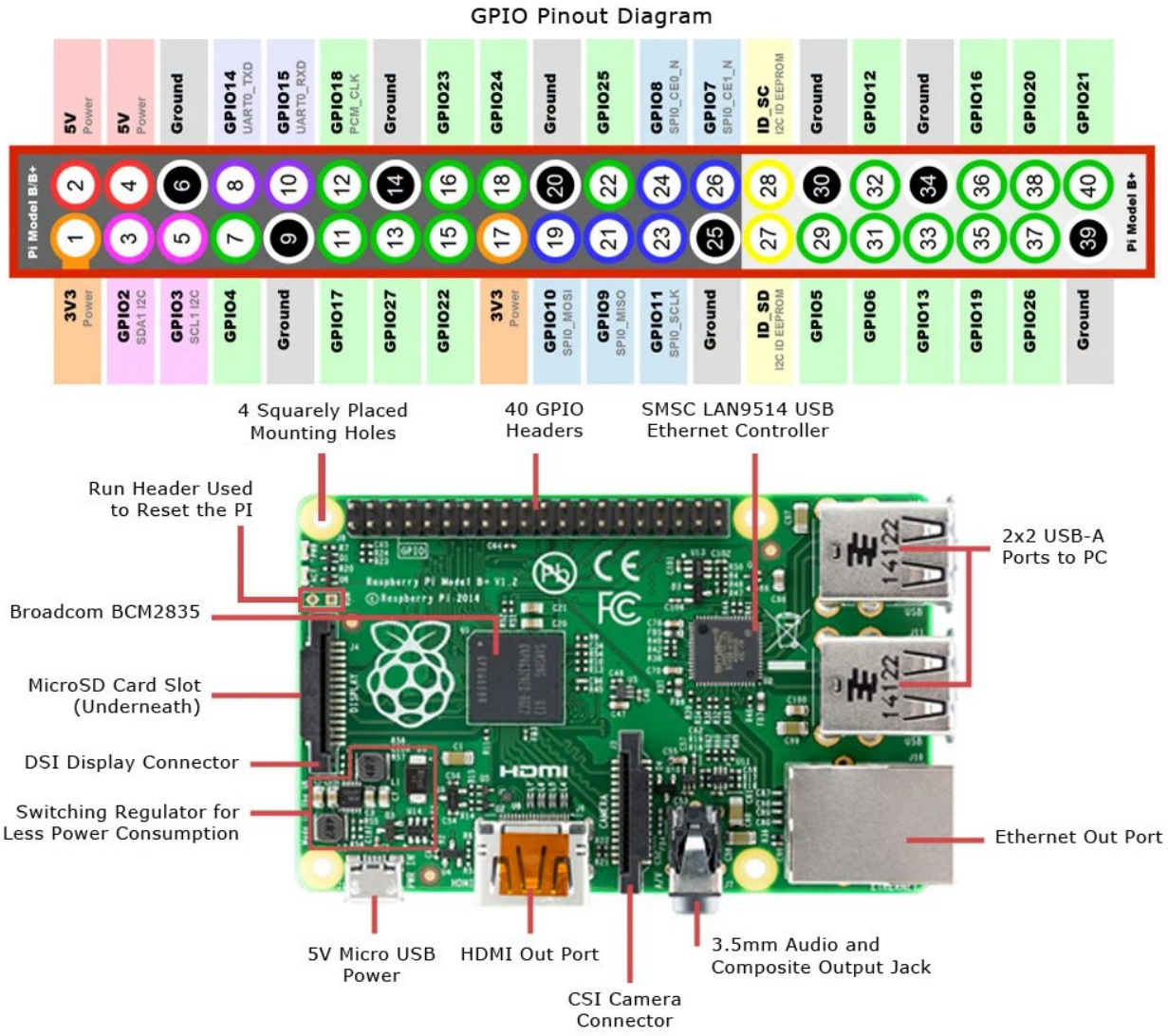
The Raspberry Pi is a low cost, **credit-card sized computer** that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do. What's more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations etc.

Raspberry pi GPIO pins :

The GPIO pins are one way in which the Raspberry Pi can control and monitor the outside world by being connected to electronic circuits.

The Pi can control LEDs, turning them on or off, drive motors, and interact with many other objects. It can also detect the pressing of a switch, change in temperature, or light, etc, by attaching kinds of

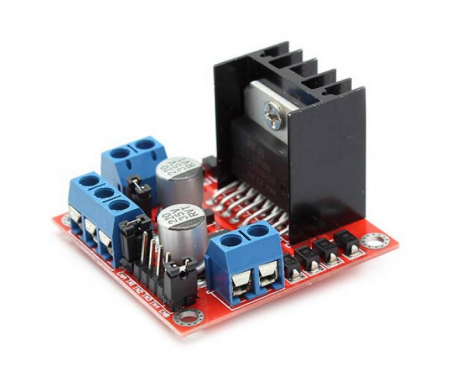
sensors. We refer to all these activities, and more, as **physical computing**.



Micro SD card :

Micro SD card is used to store the operating system for raspberry pi (Raspbian OS) which is one of many flavors of Linux OS.

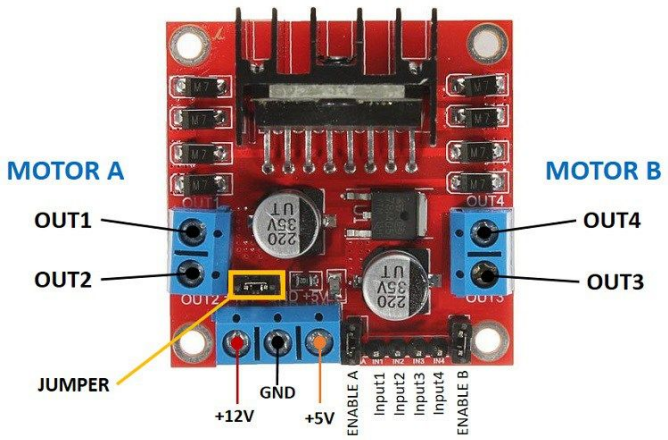
Motor controller :



The Motor controller is a special circuit that allows raspberry pi to control motors using GPIO pins. Motor controller acts as an interface hardware circuit between raspberry pi and DC motors of the robot.

The Motor controller used in this project is LN298N Motor controller driver circuit. The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

The module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output.



This depends on the voltage used at the motors VCC. The module have an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V we can enable the 5V regulator and the 5V pin can be used as output, for example for powering our Arduino board. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need connect it to a 5V power supply in order the IC to work properly. We can note here that this IC makes a voltage drop of about 2V. Next are the logic control inputs. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled.

Jumper Wires :

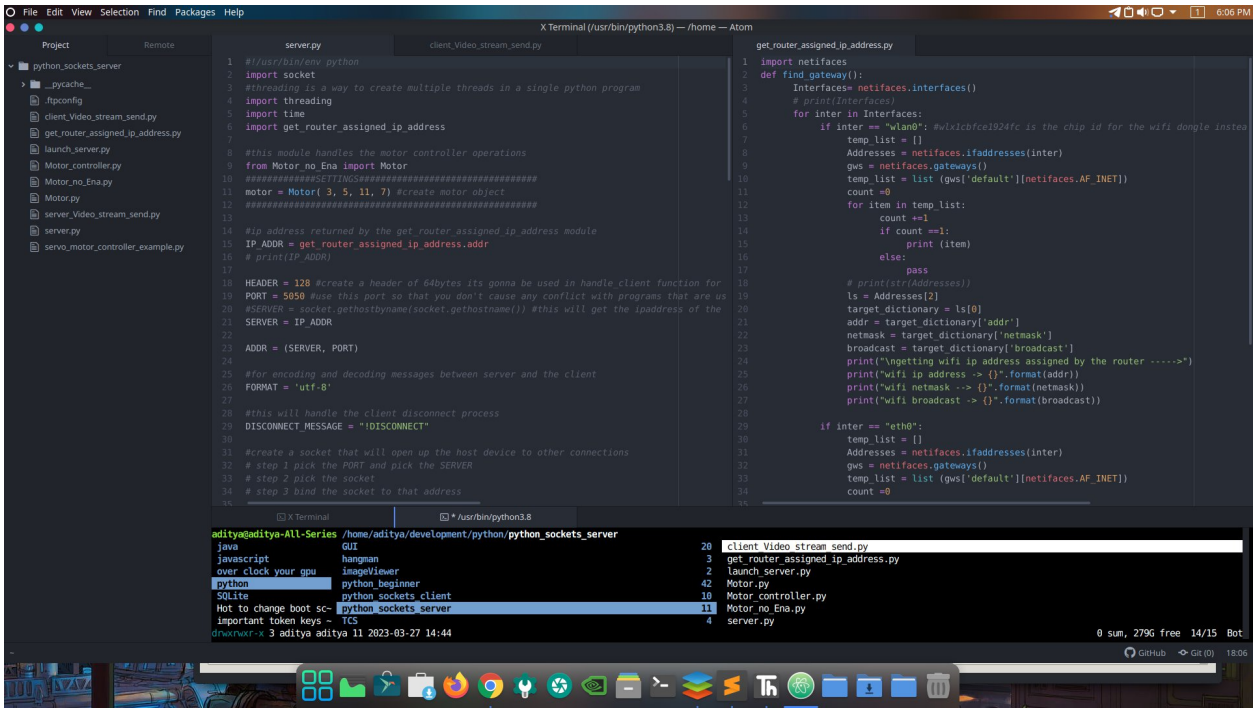
Jumper wires are used to make it easier to manage circuits built on a breadboard. They allow the components of the circuit to be spread out more, allowing easier access, and also allows them to be laid out in a more logical fashion. The wires are usually cut to lengths such that they fit neatly between two holes in the breadboard, but it is easy enough to create your own using a length of wire and some wire strippers.

Here we are using it to connect Raspberry pi to Motor Controller and we are also using it to connect Motor controller to all four dc motors of the robot.

Software

IDE :

In this project the software code for raspberry pi and client app to control the robot is written in Atom IDE.



Python programming language :

Python is a popular programming language. It was created by Guido van Rossum and released in 1991.

It is used for :

- Web development (Server side).
- Software development.
- Mathematics.
- System Scripting.
- Machine Learning and AI.

Here in this project we are using python programming language to create servers that will operate on raspberry pi. Python programming language is

also used in writing a GUI client application. The server running on raspberry pi will accept commands sent from the client app. This entire system will work together and allow us to operate the robot wirelessly using a laptop or a desktop computer and see the video live stream in real-time simultaneously while controlling the robot itself.

Libraries used :

Server side Libraries (raspberry pi):

Socket : This module provides access to the BSD socket interface. It is available on modern UNIX systems, Linux OS, Windows OS etc. Some behaviors may be platform dependent, since calls are made to the operating system socket API.

The Python interface is a transliteration of the UNIX system call and library for socket to Python's object-oriented style : the socket() function returns a socket object whose methods implements various socket system calls. Parameter types are read() and write() operations just like Python files, buffer allocation on receive operations are automatic and buffer length is implicit in send operations.

Out of many socket families like AF_UNIX, AF_INET6, AF_NETLINK, AD_TIPC etc.. We are only using AF_INET in this a pair of (host, port) is used for address family, where the host is a string representing either a host-name in the internet domain notation like 'www.adityaSarthak.com' or an IPv4 address like '100.50.200.5' , and port integer.

Open CV : Open CV is an open source computer vision library which is commonly used in conjunction with Machine Learning and AI.

Here we are using Open CV to access the camera of raspberry pi and capture the video footage in real-time.

Pickle : The pickle library implements binary protocols for serializing and de-serializing a Python object structure. "Pickling" is a process where a Python object is converted into a byte stream and " unpickling " is the inverse operation where byte stream is converted back into an object hierarchy.

Here we are using Pickle Library to serialize every frame of the video captured by the raspberry pi camera before sending it to the client machine via python socket library.

Netifaces : Netifaces is a third-party library in python programming language to enumerate network interfaces on local machine. Historically it has been difficult to straightforwardly get the network addresses of the machine on which your python scripts are running without compromising the portability of the script. Netifaces takes care of enumerating interfaces , network addresses and also preserve the portability of the python script.

Here we are using Netifaces Library to get access to the onboard wifi card of the raspberry pi and get the IP address of the raspberry pi on the network.

This IP address is later used in the socket library so that we can send and receive messages between client application running on laptop and servers running on raspberry pi.

Threading : Threading library in python allows you to have different parts of your program run concurrently and can simplify your design.

Here we have used threading to receive the packets of data from the client application in our server running on raspberry pi so that we do not stop other threads running on the server in the terminal application.

GPIO : GPIO library provides basic interactions with the GPIO pins of the raspberry pi but no implementation of any connection protocol.

Here we have used this library to interface with the GPIO pins of raspberry pi so that we can control motors via LN298N H-Bridge Motor controller.

Sleep : Python has a library called time which provide several methods to handle time-related tasks. One of most popular method is sleep(). The sleep() method suspends execution of the current thread for a given number of seconds.

We have used this library in the test main method of Motor_no_Ena.py file in order to test the motors of the raspberry pi robot during initial development process.

Client side Libraries (raspberry pi):

Tkinter : Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications.

Here we have used Tkinter library to write a GUI client application to control the raspberry pi robot and to view the received video stream sent by the robot in our application window.

Multiprocessing : Multiprocessing library in python allows you to have different parts of your program run concurrently and can simplify your design. Multiprocessing package offers both local and remote concurrency.

Here we have used multiprocessing to run two processes simultaneously in our Tkinter GUI client application. One process is responsible for making a connection via sockets to the first server which controls the Motors of the robot and the second process is responsible for making a connection to the second server that is responsible for transmitting video feed from raspberry pi camera to our GUI application running on the laptop.

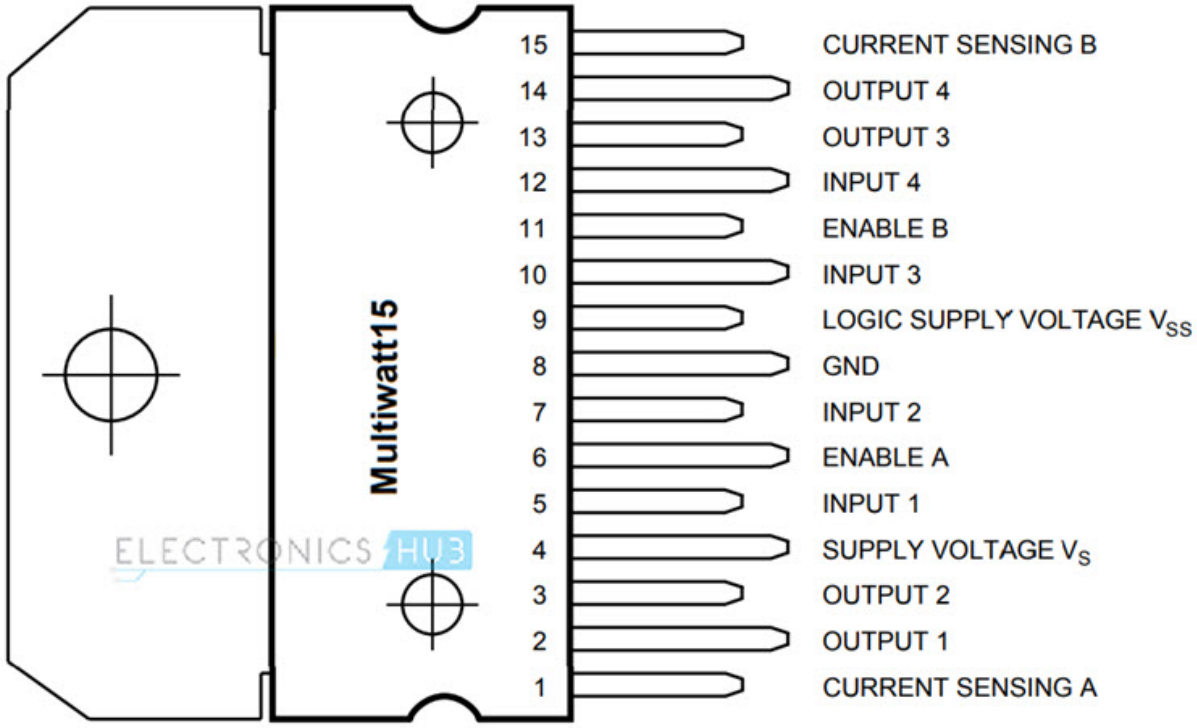
[The rest of the libraries are the same as server side libraries]

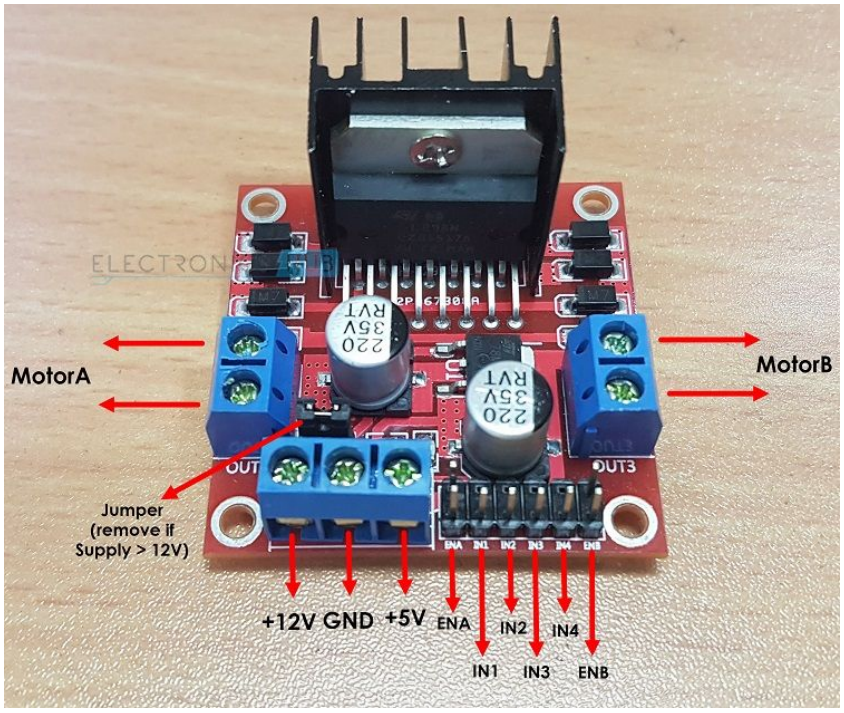
System Design

Introduction :

System design is a crucial aspect of developing complex software applications, systems, or processes that efficiently meet specific requirements and solve real-world problems. It encompasses the entire process of defining, planning, and implementing a well-structured and functional system architecture. Whether it's designing a web application, building a large-scale distributed system, or creating an embedded software solution, system design plays a pivotal role in ensuring the success of the final product.The primary goal of system design is to create a scalable, robust, and maintainable system that fulfills the desired objectives. It involves analyzing requirements, identifying components, defining interfaces, allocating resources, and determining the overall structure of the system. Effective system design takes into account various factors, including performance, reliability, security, extensibility, usability, and cost.

Circuit and Pin diagram:

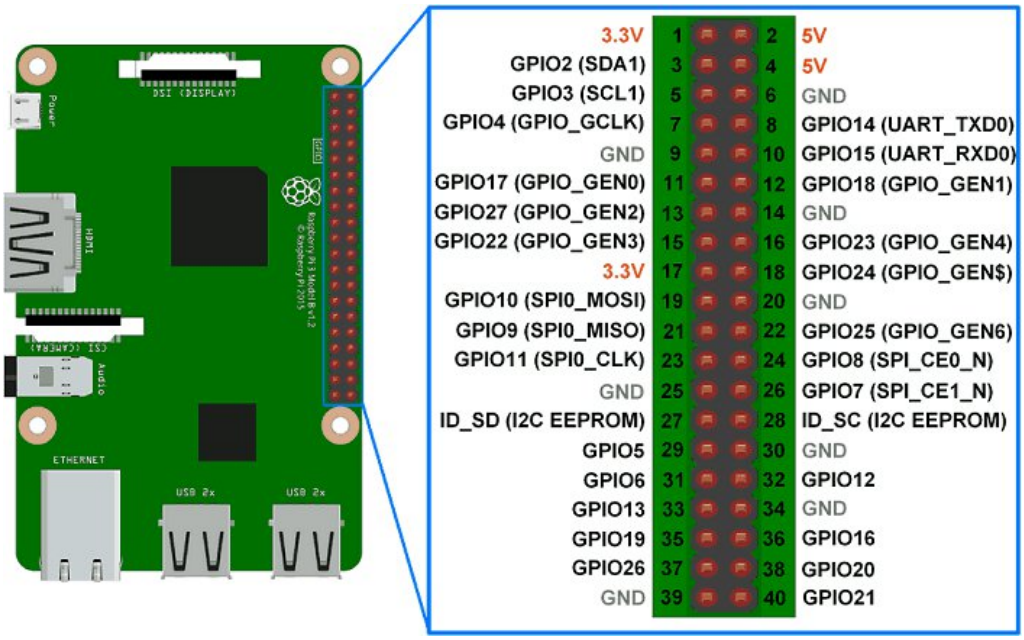




Step 1 : Connect the two motors left and right of the robot to each slots Motor A and Motor B as shown above.

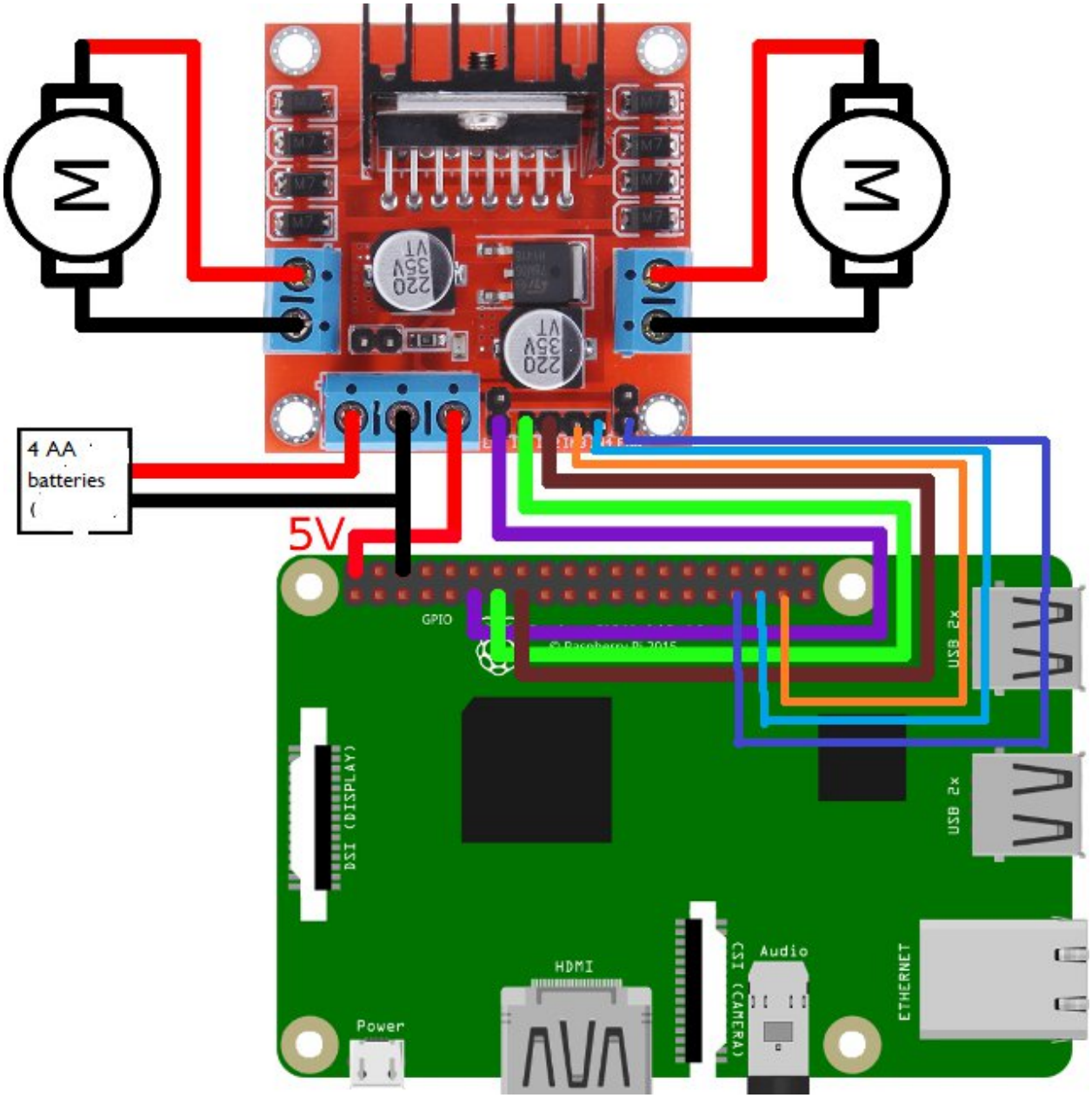
Step 2 : Connect the positive +12V, ground wire from the power supply to the slot of the Motor controller. NOTE that the ground wire will split in two one of which will go to the raspberry pi GPIO ground pin. Now attach the +5V wire from raspberry pi to the Motor Controller.

Step 3 : Connect the ENA, IN1, IN2, IN3, IN4, ENB pins of Motor Controller to the raspberry pi GPIO pins 3, 5, 7, 11 . Raspberry pi GPIO pins are shown as below

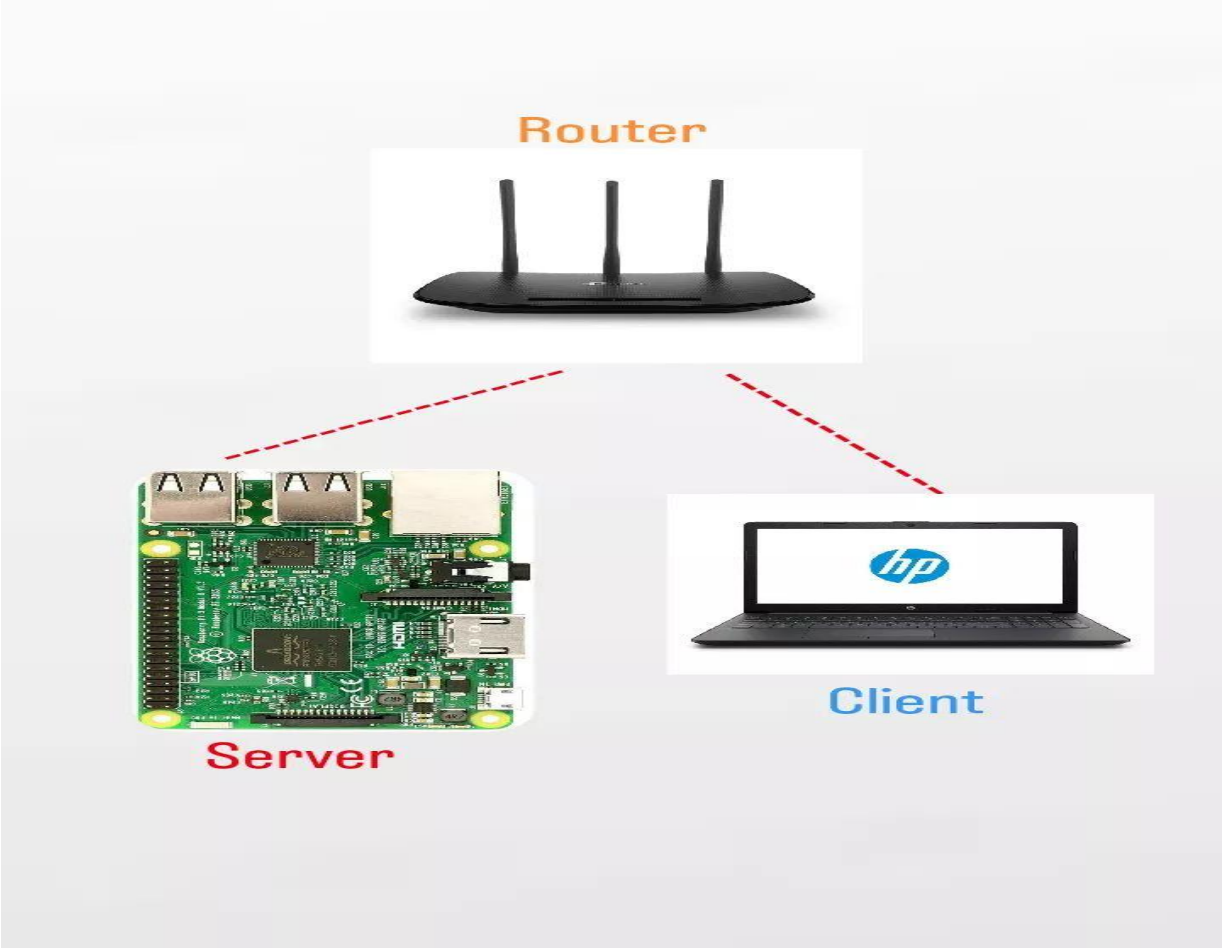


Here is an example Circuit diagram for raspberry pi robot.

[Note : The circuit diagram shown here can be different from the real world model that we have used in our project]



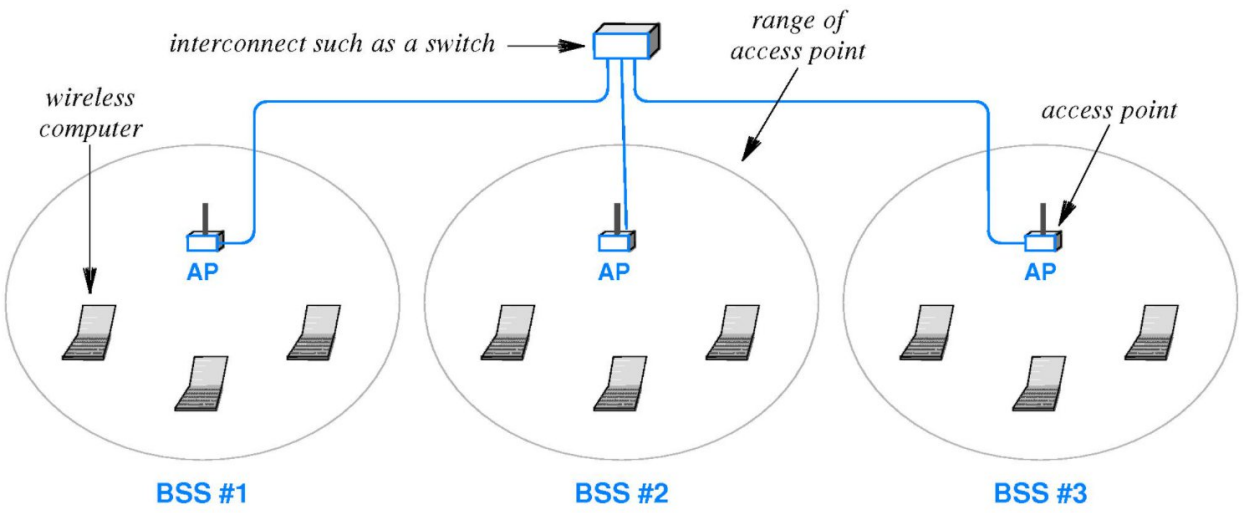
Network Architecture :



A WLAN (wireless local area network) is a network that enables devices to connect and communicate wirelessly. In contrast to a standard wired LAN, where devices connect via Ethernet cables, WLAN devices interact via Wi-Fi. While a WLAN differs from a standard LAN in appearance, it performs the same services. DHCP is commonly used to add and set up new devices. They can communicate with other network devices in the same manner as wired devices can. The fundamental distinction is in the manner in which data is conveyed. In a LAN, data is sent in a sequence of Ethernet packets through physical connections. Packets are transmitted over the air in a WLAN.

Stations : Stations are network components that communicate wirelessly. They might be access points or endpoints, and each has its own network address.

Basic Service Set : A Basic Service Set is a network that connects a group of stations. An Independent BSS is a set of stations in ad hoc network (IBSS). An Extended Service set (ESS) is a collection of connected BSSs , such as those found in a network with many access points.



Distribution System : In an ESS the distribution system connects access points. Wired or Wireless connectivity are available. Mesh or its own WDS protocol can be used by a wireless distribution system (WDS). Fixed wireless is a type of radio transmission used to connect two geographically separated access points.

- **Bridge :** A WLAN bridge connects a WLAN to a LAN or an access point
- **Endpoint :** It's a computer, mobile or Internet of things used by user
- **Access point :** The base station that serves as a connection point for other stations is known as the access point.

Robot Implementation and testing

Implementation :

Client GUI Application (running on client's computer):

Client.py : This file is responsible for managing connection with the server (running on raspberry pi) that controls the DC motors of the robot by interfacing with the LN298N Motor controller.

Source code :

```
import socket

programs that are using the network on your device
class client_manager():
    def __init__(self, HEADER, PORT, FORMAT, SERVER):
        self.HEADER = HEADER #64

        self.PORT = PORT #5050
        self.FORMAT = FORMAT #'utf-8'
        self.SERVER = SERVER #"192.168.29.222"
    def connect_to_server(self):
        ADDR = (self.SERVER, self.PORT)
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        client.settimeout(20)   # set the timeout for the client of 5 seconds
throw an exception if server is not found after 5 seconds
        global ERR
        ERR = False
        try:
            client.connect(ADDR)
            ERR = False
            return client
        except socket.error:
            print("Cannot find server on the network")
```

```
            print("make sure you are on the same network as your raspberry pi robot")
            ERR = True
            return ERR
    def send(self, msg, client_id):
        client = client_id
        message = msg.encode(self.FORMAT) #it will convert the messages from string to bytes format
        msg_length = len(message)
        if ERR == False: # this if statement will anly allow client to send message when it successfully connects to the server
            send_length = str(msg_length).encode(self.FORMAT)
like the HEADER message
            send_length += b' ' * (self.HEADER - len(send_length))
            client.send(send_length)
            client.send(message)
            print(client.recv(800000).decode(self.FORMAT))
#declaring main function
def main():
    client_id = client_class.connect_to_server()
    client_class.send("Ilove ironman rpi from MSI", client_id)
    client_class.send("!DISCONNECT", client_id)
if __name__ == '__main__':
    client_class = client_manager(64, 5050, 'utf-8', "192.168.29.222")
    main()
```

Client_video.py : This file is responsible for managing connection with the server (running on raspberry pi) that captures the video from the camera attached to the raspberry pi and send it to the client application so that the operator of the robot can see the real-time video stream while controlling the robot.

Source code :

```
import socket

DISCONNECT_MESSAGE = "!DISCONNECT"

class client_video_manager():

    def __init__(self, HEADER, PORT, FORMAT, SERVER):

        self.HEADER = HEADER #64

        self.PORT = PORT #5050

        self.FORMAT = FORMAT #'utf-8'

        self.SERVER = SERVER #"192.168.29.222"

    def connect_to_server(self):

        ADDR = (self.SERVER, self.PORT)

        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

      client.settimeout(60)   # set the timeout for the client of 5 seconds throw an exception if server is not found after 5 seconds

        global ERR

        ERR = False

        try:

            client.connect(ADDR)

            ERR = False

            return client

        except socket.error:

            print("Cannot find server on the network")

robot") print("make sure you are on the same network as your raspberry pi
```

```
            ERR = True
            return ERR


    def send(self, msg, client_id):
        client = client_id
        message = msg.encode(self.FORMAT) #it will convert the messages from string to bytes format
        msg_length = len(message)
      if ERR == False: # this if statement will anly allow client to send message when it successfully connects to the server
            send_length = str(msg_length).encode(self.FORMAT)
            send_length += b' ' * (self.HEADER - len(send_length))
            client.send(send_length)
            client.send(message)
            #recieving message from the server testing ONLY
            #print(client.recv(800000).decode(self.FORMAT))


#declaring main function
def main():
    client_id = client_class.connect_to_server()
    client_class.send("e", client_id)
```

```
    client_class.send("!DISCONNECT", client_id)



if __name__ == '__main__':

    client_class = client_manager(128, 5051, 'utf-8', "192.168.29.235")

    main()
```

Video_recv.py : This file is responsible for receiving live video stream sent by the server running on raspberry pi.

#This is a client which will send the video feed to the host server from rpi -> to another computer where the server is Listening

Source code :

```
import socket, cv2, pickle, struct, numpy

class video_reciever():

    def __init__(self,PORT,SERVER):

        self.PORT = PORT #5050

        self.SERVER = SERVER #"192.168.29.222"

    def connect_to_server(self):

        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        Addr_server = (self.SERVER, self.PORT)

        client_socket.settimeout(20)
```

```
        global ERR
        ERR = False
        try:
            client_socket.connect(Addr_server)
            ERR = False
            return client_socket
        except socket.error:
            print("Cannot find server on the network")
            print("make sure you are on the same network as your raspberry pi
robot")
            ERR = True
            return ERR
    def recv(self, client_socket):
        client_socket = client_socket
        #initialize the data variable as an empty string
        data = b""
        payload_size = struct.calcsize("Q")
        try:
            while True:
```

```
        while len(data)<payload_size:
            #here we will use 4kb buffer
            packet = client_socket.recv(4*1024)
            if not packet: break
            data+=packet
        packed_msg_size = data[:payload_size]
        #the rest of the data contains our video frame
        data = data[payload_size:]
        msg_size = struct.unpack("Q",packed_msg_size)[0]
        while len(data)< msg_size:
            data +=client_socket.recv(4*1024)
        frame_data = data[:msg_size]
        data = data[msg_size:]
        frame = pickle.loads(frame_data)
        cv2.imshow("Recieved", frame)
        key = cv2.waitKey(1) & 0xFF #display
except Exception:
    client_socket.close()
```

Keyboard_input_tk.py : This is the main file which is responsible for creating the actual GUI elements that the user will see on his/her desktop screen in the front end and also holds all the backend functionality related to raspberry pi robot controls and video receiving functions.

Source code :

```
from tkinter import *

from tkinter import messagebox

from client import client_manager

from client_video import client_video_manager

from video_recv import video_reciever

import multiprocessing

import webbrowser



import dev_info

from PIL import ImageTk,Image

main = Tk()

main.title("Raspberry pi Robot Controller Application")

main.geometry("1024x142+0+0")

main.minsize(1024, 142)

main.maxsize(1024, 142)
```

```
Display_frame_background = "#155263"

Display_frame = LabelFrame(main,text="Output",font = ("times new roman",20,"bold"),padx=5,pady=5, bd=5, relief=RIDGE,fg = "white", bg = Display_frame_background)

Display_frame.place(x=390,y=0,width= 635, height= 80)

Label(Display_frame, text = "Command sent to the server : ", bg = Display_frame_background, font = ("times new roman",15,"bold"), fg="white").grid(row=0,column=0)

output_text = StringVar()

output_field = Entry(Display_frame, font = ("times new roman",15,"bold"),bg="#f6e4ad", fg="black", textvariable = output_text)

output_field.grid(row=0,column=1,sticky = W+E)

#this function will show developer information in a new window

def openNewWindow():

    print("developer info")

    developer_window = Toplevel(main)

    developer_window.title("Developer Info")

    developer_window.geometry("530x500")

    developer_window.minsize(628, 500)

    developer_window.maxsize(628, 500)

    Developer_frame_backgrount_color = "#3e4a61"
```

```
    Developer_frame = LabelFrame(developer_window,text="Developer Information",bd=7,relief=GROOVE,font=("times new roman", 15, "bold"),fg = "white", bg = Developer_frame_backgrount_color)

    Developer_frame.place(x=0,y=0,width=628,height=500)

    Dev_name_Label = Label(Developer_frame,text="Name:-",font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=0,column=0,padx=5,pady=5,sticky="w")

    Dev_name_aditya_Label = Label(Developer_frame,text=dev_info.Name,font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=0,column=1,padx=5,pady=5,sticky="w")

    Roll_Number_Label = Label(Developer_frame,text="Roll no:-",font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=1,column=0,padx=5,pady=5,sticky="w")

    Roll_number_Label = Label(Developer_frame,text=dev_info.Roll_Number,font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=1,column=1,padx=5,pady=5,sticky="w")

    College_code_Label = Label(Developer_frame,text="College code:-",font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=2,column=0,padx=5,pady=5,sticky="w")

    College_code_Label = Label(Developer_frame,text=dev_info.College_code,font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=2,column=1,padx=5,pady=5,sticky="w")

    #Branch
```

```
   Branch_Label = Label(Developer_frame,text="Branch:-",font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=3,column=0,padx=5,pady=5,sticky="w")

    Branch_2_Label = Label(Developer_frame,text=dev_info.Branch,font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=3,column=1,padx=5,pady=5,sticky="w")

 Course_Label = Label(Developer_frame,text="Course:-",font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=4,column=0,padx=5,pady=5,sticky="w")

    Course_2_Label = Label(Developer_frame,text=dev_info.Course,font=("times new roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,fg="white").grid(row=4,column=1,padx=5,pady=5,sticky="w")

    global canvas

    global img

    canvas = Canvas(Developer_frame, width =229, height = 355)

    canvas.grid(row=0,column=2,rowspan=5)

    img = ImageTk.PhotoImage(Image.open("dev.png"))

    canvas.create_image(2, 2, anchor=NW, image=img)

    Dev_Search_Url = "https://radiant-bastion-62859.herokuapp.com/profile/3947f970-07f0-4863-bdf6-0d247c0b2e82/"

    GitHub_Url = "https://github.com/aryan68125?tab=repositories"

    Linked_in_Url = "https://www.linkedin.com/in/aditya-kumar-74700520a/"

    new = 1
```

```
    def openDevSearch():

        webbrowser.open(Dev_Search_Url,new=new)

    def openGithub():

        webbrowser.open(GitHub_Url,new=new)

    def openLinkedin():

        webbrowser.open(Linked_in_Url,new=new)



    social_media_Label = Label(Developer_frame,text="Social Media Links:-
",font=("times new
roman",12,"bold"),pady=3,padx=5,bg=Developer_frame_backgrount_color,f
g="white").grid(row=5,column=0,padx=5,pady=5,sticky="w")

    Open_devsearch = Button(Developer_frame, text = "Dev
search",command=openDevSearch)

    Open_devsearch.grid(row=6,column=0,padx=5,pady=5,sticky="w")

    Open_github = Button(Developer_frame, text =
"Github",command=openGithub)

    Open_github.grid(row=6,column=1,padx=5,pady=5,sticky="w")

    Open_Linkedin = Button(Developer_frame, text =
"Linkedin",command=openLinkedin)

    Open_Linkedin.grid(row=6,column=2,padx=5,pady=5,sticky="w")

def openHelpWindow():

    print("Controls Information")
```

```
    Help_window = Toplevel(main)

    Help_window.title("Help window")

    Help_window.geometry("650x715")

    Help_window.minsize(650, 715)

    Help_window.maxsize(650, 715)

    global img1,img2, imageList

    img1 = ImageTk.PhotoImage(Image.open("controls.png"))

    img2 = ImageTk.PhotoImage(Image.open("camera_controls.png"))

    imageList=[img1,img2]

    Help_frame_backgrount_color = "#8a1253"

    Help_frame = LabelFrame(Help_window,text="Help",bd=7,relief=GROOVE,font=("times new roman", 15, "bold"),fg = "white", bg = Help_frame_backgrount_color, padx= 5,pady=5)

    Help_frame.place(x=0,y=0,width=650,height=715)

    Canvas_background_color = "#f0ab8d"

    Canvas_Frame = LabelFrame(Help_frame,text="",bd=7,relief=GROOVE,font=("times new roman", 15, "bold"),fg = "white", bg = Canvas_background_color, padx= 5,pady=5)

    Canvas_Frame.place(x=0,y=50,width=625,height=625)

    global canvas
```

```
    canvas = Canvas(Canvas_Frame, width =600, height = 600)

    canvas.grid(row=0,column=0,rowspan=5)

    canvas.create_image(2, 2, anchor=NW, image=img1)

    def OpenRobotControlsHelp(image_list_index):

        canvas = Canvas(Canvas_Frame, width =600, height = 600)

        canvas.grid(row=0,column=0,rowspan=5)

        canvas.create_image(2, 2, anchor=NW, image=imageList[image_list_index])

    def OpenRobotCameraControlsHelp(image_list_index):

        canvas = Canvas(Canvas_Frame, width =600, height = 600)

        canvas.grid(row=0,column=0,rowspan=5)

        canvas.create_image(2, 2, anchor=NW, image=imageList[image_list_index])

    Movement_Help_button = Button(Help_frame,text="Robot Controls", command = lambda : OpenRobotControlsHelp(0), font = ("times new roman",15,"bold"),bg="#c51350", fg="black")

    Movement_Help_button.grid(row=0,column=0, sticky="w", pady=5 ,padx=5)

    Camera_Help_button = Button(Help_frame,text="Camera Controls", command = lambda : OpenRobotCameraControlsHelp(1), font = ("times new roman",15,"bold"),bg="#c51350", fg="black")

   Camera_Help_button.grid(row=0,column=1, sticky="w", pady=5 ,padx=5)
```

```
devInfo_frame_background = "#9fd3c7"

devInfo_frame = LabelFrame(main,text="",font = ("times new roman",20,"bold"),padx=5,pady=5, bd=5, relief=RIDGE,fg = "white", bg = devInfo_frame_background)

devInfo_frame.place(x=390,y=80,width= 635, height= 60)

dev_button = Button(devInfo_frame,text="Developer Information", command = openNewWindow, font = ("times new roman",15,"bold"),bg="#e0ffcd", fg="black")

dev_button.grid(row=0,column=0, sticky="w", pady=2)

help_button = Button(devInfo_frame,text="Help", command = openHelpWindow, font = ("times new roman",15,"bold"),bg="#e0ffcd", fg="black")

help_button.grid(row=0,column=1, sticky="w", pady=2 ,padx=5)

def start_camera():

    def start_camera():

        #initializing video recv class that will actually recieve video from the raspberry pi

        global video_reciever_class

        ip_addrc = ip_addr_text.get()

        print("server_vido_ip : ", ip_addrc)

        video_reciever_class = video_reciever(9000,str(ip_addrc))

        client_socket= video_reciever_class.connect_to_server()
```

```
        video_reciever_class.recv(client_socket)

    output_text_command = output_text.get()

    if output_text_command != "Video streaming: ON":

      if output_text_command != "Oops! Try again" and output_text_command != "Turn Left" and output_text_command != "Turn Right" and output_text_command != "Forward" and output_text_command != "Reverse" and output_text_command != "Stop":

            try:

                print("starting video transmission")

                output_field.configure(state='normal')

                output_field.delete(0,"end")

            output_field.insert(END,"Video streaming: ON") #commands_Sent are the commands that is sent to the server

                output_field.configure(state='disabled')

                client_video_class.send("e", client_video_id)

                global proc

                proc = multiprocessing.Process(target=start_camera, args=())

                proc.start()

            except:

                print("Oops Try again")

                output_field.configure(state='normal')
```

```
            output_field.delete(0,"end")

            output_field.insert(END,"Oops! Try again") #commands_Sent are the commands that is sent to the server

            output_field.configure(state='disabled')

        else:

         messagebox.showerror("ERROR (69x420)","Camera is already on!!")

            return

    else:

        messagebox.showerror("ERROR (69x420)","Camera is already on!!")

        return

def stop_camera():

    output_field.configure(state='normal')

    output_field.delete(0,"end")

    output_field.insert(END,"Video streaming: OFF")

    output_field.configure(state='disabled')

    proc.terminate()

camera_ON_button = Button(devInfo_frame,text="Camera On", command = start_camera, font = ("times new roman",15,"bold"),bg="#e0ffcd", fg="black")

camera_ON_button.grid(row=0,column=2, sticky="w", pady=2 ,padx=5)
```

```
camera_OFF_button = Button(devInfo_frame,text="Camera Off", command = stop_camera, font = ("times new roman",15,"bold"),bg="#e0ffcd", fg="black")

camera_OFF_button.grid(row=0,column=3, sticky="w", pady=2 ,padx=5)

def leftKey(event):

    print ("LeftArrow key pressed")

    Command_sent = "a"

    client_class.send(Command_sent, client_id)

    output_field.configure(state='normal')

    output_field.delete(0,"end")

    output_field.insert(END,"Turn Left")

    output_field.configure(state='disabled')



def rightKey(event):

    print ("RightArrow key pressed")

    Command_sent = "d"

    client_class.send(Command_sent, client_id)

    output_field.configure(state='normal')

    output_field.delete(0,"end")

    output_field.insert(END,"Turn Right")
```

```
    output_field.configure(state='disabled')

def upKey(event):

    print ("upArrow key pressed")

    Command_sent = "w"

    client_class.send(Command_sent, client_id)

    output_field.configure(state='normal')

    output_field.delete(0,"end")

    output_field.insert(END,"Forward") #commands_Sent are the commands
that is sent to the server

    output_field.configure(state='disabled')

def downKey(event):

    print ("downArrow key pressed")

    Command_sent = "s"

    client_class.send(Command_sent, client_id)

    output_field.configure(state='normal')

    output_field.delete(0,"end")

    output_field.insert(END,"Reverse") #commands_Sent are the commands
that is sent to the server

    output_field.configure(state='disabled')

def halt(event):
```

```
    print ("downArrow key pressed")

    Command_sent = "b"

    client_class.send(Command_sent, client_id)

    output_field.configure(state='normal')

    output_field.delete(0,"end")

   output_field.insert(END,"Stop") #commands_Sent are the commands that
is sent to the server

    output_field.configure(state='disabled')

frame = Frame(main, width=1, height=1)

main.bind('<Left>', leftKey)

main.bind('<Right>', rightKey)

main.bind('<Up>', upKey)

main.bind('<Down>', downKey)

main.bind('b', halt)

frame.pack()

DISCONNECT_MESSAGE = "!DISCONNECT"

def disconnect():

    print("disconnecting from raspberry pi....")

    #close Robot motor controller server
```

```
    client_class.send(DISCONNECT_MESSAGE, client_id)
    client_video_class.send(DISCONNECT_MESSAGE, client_video_id)
    connect_button["state"]="normal"
    disconnect_button["state"]="disabled"
    output_field.delete(0,"end")

def connect():
  print("connecting to raspberry pi....")
  global server_ip
  global ip_addr
  ip_addr = ip_addr_text.get()
  if ip_addr== "":
       messagebox.showerror("Required Fields", "Input Fields are NULL!")
       return
  else:
      server_ip = ip_addr
  print(ip_addr)
  global client_class
  client_class = client_manager(128, 5050, 'utf-8', str(ip_addr))
```

```
  global client_id

  client_id = client_class.connect_to_server()

  ERR = client_id

  client_class.send("start_signal_from_client", client_id)

  if ERR==True:

      messagebox.showerror("ERROR", "Cannot find your Robot on the network \nMake sure you and the server are on the same LAN network")

      connect_button["state"]="normal"

      disconnect_button["state"]="disabled"

      return

  print(ip_addr)

  global client_video_class

  client_video_class = client_video_manager(128, 5000, 'utf-8', str(ip_addr))

  global client_video_id

  client_video_id = client_video_class.connect_to_server()

  ERR = client_video_id

  client_video_class.send("start_signal_from_client", client_video_id)

  if ERR==True:

      messagebox.showerror("ERROR", "Cannot find your Robot on the network \nMake sure you and the server are on the same LAN network")
```

```
        connect_button["state"]="normal"

        disconnect_button["state"]="disabled"

        return

    connect_button["state"]="disabled"

    disconnect_button["state"]="normal"

connect_frame_background = "#5b446a"

connect_frame = LabelFrame(main,text="",padx=5,pady=5, bd=5, relief=RIDGE,fg = "white", bg = connect_frame_background)

connect_frame.place(x=0,y=0, width= 390, height= 140)

ip_addr_text = StringVar()

ip_addr_input = Entry(connect_frame,font = ("times new roman",15,"bold"),bg="#f6e4ad", fg="black", textvariable = ip_addr_text).grid(row=0,column=0,pady=5,padx=7, columnspan=2, sticky=W+E)

connect_Label = Label(connect_frame, text="Connect to robot : ", font = ("times new roman",15,"bold"), bg = connect_frame_background)

connect_Label.grid(row=1,column=0, sticky="w")

connect_button = Button(connect_frame,text="Connect", command=connect, font = ("times new roman",15,"bold"),bg="#906387", fg="black")

connect_button.grid(row=1,column=1, sticky=W+E)

connect_Label = Label(connect_frame, text="Disconnect from robot : ", font = ("times new roman",15,"bold"), bg = connect_frame_background)
```

```
connect_Label.grid(row=2,column=0, sticky="w")

disconnect_button = Button(connect_frame,text="Disconnect", command=disconnect, font = ("times new roman",15,"bold"),bg="#906387", fg="black")

disconnect_button.grid(row=2,column=1, sticky="w")

disconnect_button["state"]="disabled"

main.mainloop()
```

Server (running on raspberry pi robot):

Get_router_assigned_ip_address.py : As we know that any wifi router assigns IP addresses randomly to the devices connected to it. But here we are using sockets library in Python so that we can make connection tunnel between the client and server so that we can control the robot. For this to happen we need to pass the IP address and port number in the socket() method but since the router randomly assigns the IP address to a device this may cause our code to explode. To mitigate this problem this module is written . This module fetches the IP address assigned to raspberry pi by the wifi router via Linux OS system api call and later we used this IP address fetched by this program into our socket() method.

Source code :

```
import netifaces

def find_gateway():

    Interfaces= netifaces.interfaces()

    for inter in Interfaces:

        if inter == "wlan0": #wlx1cbfce1924fc is the chip id for the wifi dongle instead of wlan0 which is the default wifi card
```

```
temp_list = []

Addresses = netifaces.ifaddresses(inter)

gws = netifaces.gateways()

temp_list = list (gws['default'][netifaces.AF_INET])

count =0

for item in temp_list:

    count +=1

    if count ==1:

        print (item)

    else:

        pass

ls = Addresses[2]

target_dictionary = ls[0]

addr = target_dictionary['addr']

netmask = target_dictionary['netmask']

broadcast = target_dictionary['broadcast']

print("\ngetting wifi ip address assigned by the router ----->")

print("wifi ip address -> {}".format(addr))

print("wifi netmask --> {}".format(netmask))
```

```
            print("wifi broadcast -> {}".format(broadcast))

        if inter == "eth0":

            temp_list = []

            Addresses = netifaces.ifaddresses(inter)

            gws = netifaces.gateways()

            temp_list = list (gws['default'][netifaces.AF_INET])

            count =0

            for item in temp_list:

                count +=1

                if count ==1:

                    print (item)

                else:

                    pass

    return  addr

addr = find_gateway()
```

Motor_no_Ena.py : This module controls the DC motor of the raspberry pi robot by interfacing with the L298N Motor controller via GPIO pins of raspberry pi. You can think of it as a driver module for L298N Motor controller.

Source code :

```
import RPi.GPIO as GPIO #import gpio pin library from raspbian OS

from time import sleep


GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)


class Motor():

    def __init__(self, In1A, In2A, In1B, In2B):

        self.In1A = In1A

        self.In2A = In2A

        GPIO.setup(self.In1A, GPIO.OUT)

        GPIO.setup(self.In2A, GPIO.OUT)

        self.In1B = In1B

        self.In2B = In2B

        GPIO.setup(self.In1B, GPIO.OUT)

        GPIO.setup(self.In2B, GPIO.OUT)

    def forward(self):

        GPIO.output(self.In1A, GPIO.HIGH)

        GPIO.output(self.In2A, GPIO.LOW)
```

```
        GPIO.output(self.In1B, GPIO.HIGH)
        GPIO.output(self.In2B, GPIO.LOW)
    def reverse(self):
        GPIO.output(self.In1A, GPIO.LOW)
        GPIO.output(self.In2A, GPIO.HIGH)
        GPIO.output(self.In1B, GPIO.LOW)
        GPIO.output(self.In2B, GPIO.HIGH)
    def left(self):
        GPIO.output(self.In1A, GPIO.HIGH)
        GPIO.output(self.In2A, GPIO.LOW)
        GPIO.output(self.In1B, GPIO.LOW)
        GPIO.output(self.In2B, GPIO.HIGH)
    def right(self):
        GPIO.output(self.In1A, GPIO.LOW)
        GPIO.output(self.In2A, GPIO.HIGH)
        GPIO.output(self.In1B, GPIO.HIGH)
        GPIO.output(self.In2B, GPIO.LOW)
    def stop(self):
        GPIO.output(self.In1A, GPIO.LOW)
```

```
        GPIO.output(self.In2A, GPIO.LOW)
        GPIO.output(self.In1B, GPIO.LOW)
        GPIO.output(self.In2B, GPIO.LOW)
def main():
    motor.forward()
    sleep(2)
    motor.stop()
    sleep(2)
    motor.left()
    sleep(2)
    motor.stop()
    sleep(2)
    motor.right()
    sleep(2)
    motor.stop()
    sleep(2)
    motor.reverse()
    sleep(2)
    motor.stop()
```

```
if __name__ == '__main__':

    #PASSING PIN NUMBERS TO THE MOTOR CLASS

    #left motors GPIO pin numbers Ena = 12, In1A = 3, In2A = 5

    #right motors GPIO pin numbers Enb = 13, In1B = 11, In2B = 7

    # Motor( In1A, In2A, In1B, In2B)

    motor = Motor( 3, 5, 11, 7)

    main()
```

Server_Video_stream_send.py : This is a server that is responsible for capturing video in real time via camera attached to the raspberry pi through open CV and send the video frame by frame through wireless communication network (in this case WLAN) to the client GUI application.

Source code :

```
#this code for the server which will be responsible for recieving video packets sent from raspberry pi so that we can stream videos from raspberry pi to our application

import socket

import cv2

import pickle

import struct

import get_router_assigned_ip_address
```

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)

host_name = socket.gethostname()

IP_ADDR = get_router_assigned_ip_address.Addr

host_ip = socket.gethostbyname(IP_ADDR)

print("[HOST IP] ", host_ip)

port = 9000

socket_address = (host_ip, port)

server_socket.bind(socket_address)

server_socket.listen(5)

print("[Video Stream Server Starting] Listening At....",socket_address)

while True:

    client_socket, addr = server_socket.accept()

    print("[Got Connection From] ",addr)

    if client_socket:

        vid = cv2.VideoCapture(0)

        while(vid.isOpened()):

            img, frame = vid.read()
```

```
        a = pickle.dumps(frame)

        message = struct.pack("Q", len(a)) + a

        client_socket.sendall(message)

        cv2.imshow('TRANSMITTING VIDEO',frame)

        key = cv2.waitKey(1) & 0xFF #display

        if key == ord('q'):

            client_socket.close()
```

Server.py : This server is responsible for accepting commands sent by client GUI application coming through WLAN and coordinate the motors in such a way using Motor_no_Ena.py module that the robot can move according to the operator's wishes.

Source code :

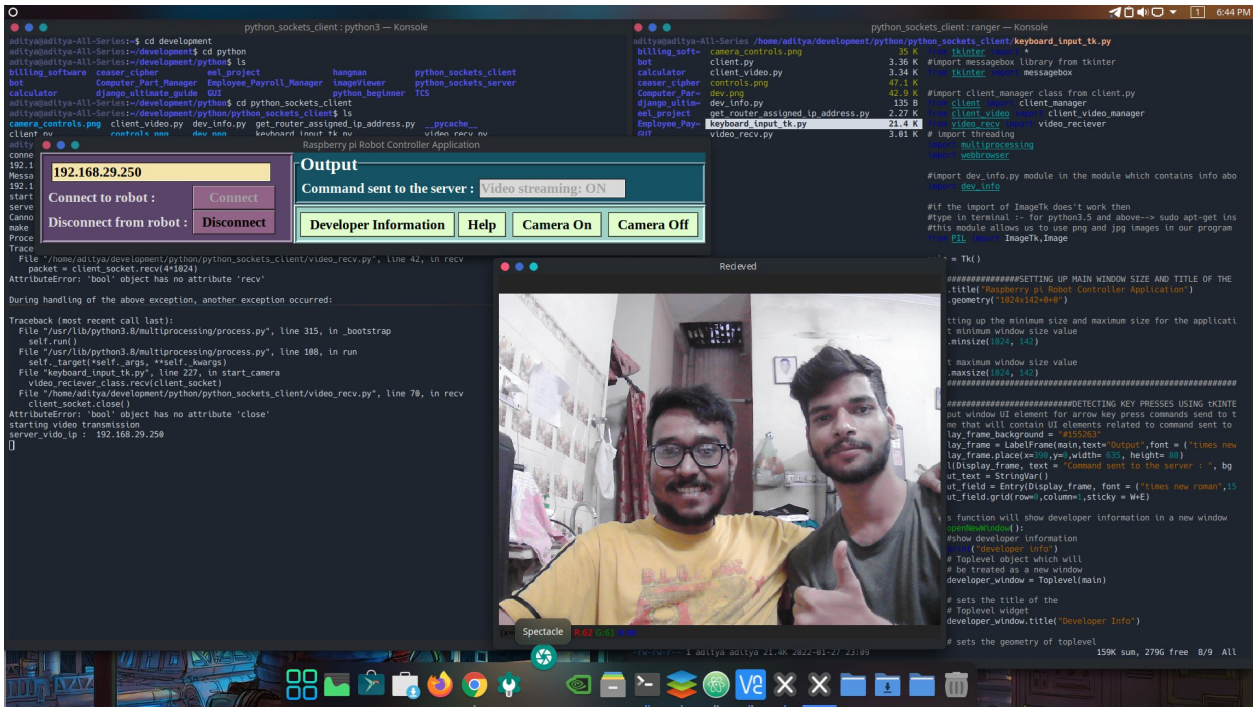
```
import socket

import threading

import time

import get_router_assigned_ip_address

from Motor_no_Ena import Motor

motor = Motor( 3, 5, 11, 7) #create motor object

IP_ADDR = get_router_assigned_ip_address.addr
```

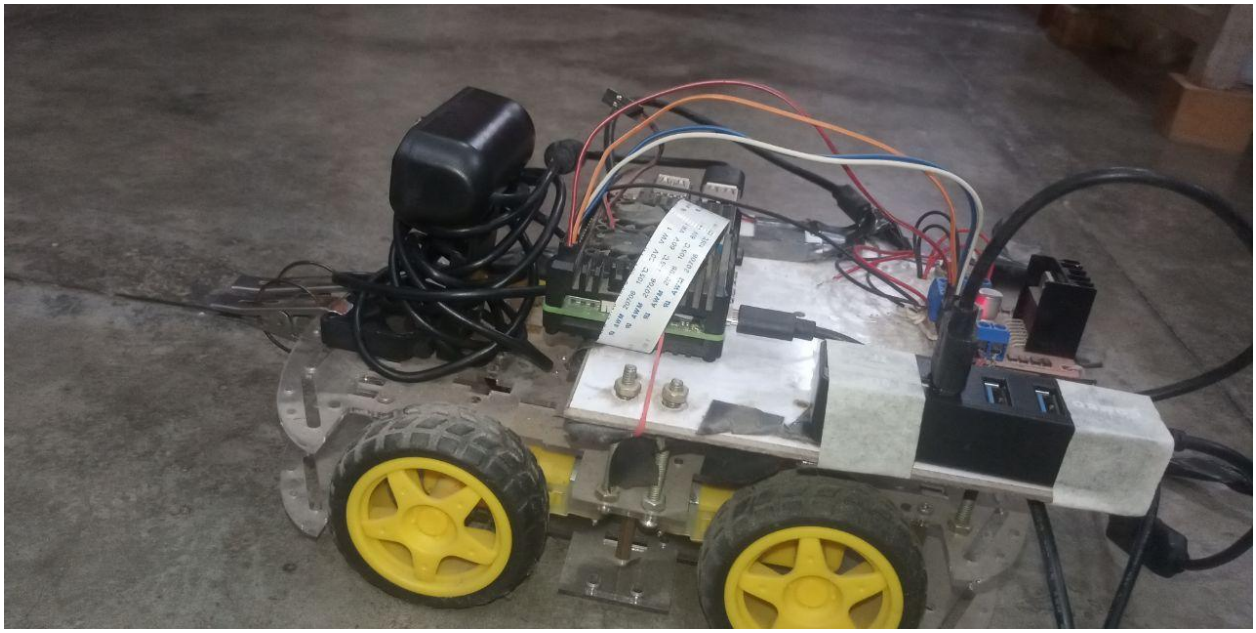
```
HEADER = 128 #create a header of 64bytes its gonna be used in handle_client function for guessing how many bytes we are gonna recieve from the client

PORT = 5050 #use this port so that you don't cause any conflict with programs that are using the network on your device

#SERVER = socket.gethostbyname(socket.gethostname()) #this will get the ipaddress of the computer this script is running on

SERVER = IP_ADDR

ADDR = (SERVER, PORT)

FORMAT = 'utf-8'

DISCONNECT_MESSAGE = "!DISCONNECT"

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) #this line allows the server to use the same Address otherwise it will throw an exception error address already in use

server.bind(ADDR)

def handle_client(conn, addr):

    print("[NEW CONNECTION] {} connected".format(addr))

    connected = True

    while connected:

        msg_length = conn.recv(HEADER).decode(FORMAT) #converting bytes into a string
```

```
if msg_length:
    msg_length = int(msg_length)
    msg = conn.recv(msg_length).decode(FORMAT)
    if msg == DISCONNECT_MESSAGE:
        connected = False
        print("[CLIENT LOGGED OUT] client left")
        print("connection closed")
    conn.send("You disconnected from the server".encode(FORMAT))
    elif msg == "w":
        motor.forward()
    elif msg == "s":
        motor.reverse()
    elif msg == "a":
        motor.left()
    elif msg == "d":
        motor.right()
    elif msg == "b":
        motor.stop()
    print("[{}] {}".format(addr, msg))
```

```
            conn.send("Message reieved".encode(FORMAT))

    conn.close()

def start():

    server.listen()

    print("[LISTENING] Server is listening on ipaddress {}".format(SERVER))

    while True:

        conn, addr = server.accept()

        thread = threading.Thread(target=handle_client, args = (conn,addr))

        thread.start()

        NumberOf_clients_connected = threading.activeCount() -1

        print("[ACTIVE                                    CONNECTIONS]
{}".format(NumberOf_clients_connected))

print("[STARTING] server is starting...")

start()
```

Output Screen





In this screen you can see that the live video is coming directly from the robot's camera and by using arrow key of the keyboard you can move the robot forward , backward and turn the robot left and right.