Smart Robot Navigation with Computer Vision and EEG Control

Justin May, Salman Omer, Forest Song,
Gaurav Sethi, Daniel Chong, and Samuel Minkin

***Abstract*—This paper describes the design process towards designing a smart robot that can be controlled using an electroencephalograph (EEG). The robot will be able to autonomously navigate around obstacles using computer vision and an ultrasonic sensor. Using an EEG, we will measure brain activity by interpreting basic alpha and beta waves. While professional grade EEGs can detect many different brain waves, we can make a cheaper EEG that is capable of reading alpha and beta waves. Alpha waves occur at around 8-12 Hz and when measured from the frontal lobe provide an estimate of how relaxed a person is, while beta waves occur around 12-30 Hz and correspond to how much a person is concentrating or how alert they are. This data will be interpreted by a machine learning model that will be able to identify distinct thoughts. By optimizing the EEG design and software, specific thoughts can be more accurately mapped. The brain imaging software will be integrated with the robot. By reading certain level of brain waves, certain commands can be given to the robot. For navigation purposes, the robot needs more maneuverability to strafe around obstacles than a simple tank drive. Different maneuverable drivetrains, called holonomic drivetrains, were examined. Ultimately, a mecanum drivetrain was decided on; the process of designing this drive is detailed in this paper. In addition, the robot requires sensory that will make it suitable for navigation. This paper examines the use of both OpenCV for computer vision and an ultrasonic sensor to help the robot with navigation. .**

I. Introduction

WHEN coming up with a project idea, we wanted to create something that would allow us to challenge ourselves while maximizing our skill sets. Our group consists of one mechanical engineering student and five electrical/computer engineering students. On the electrical/computer side, this group has a strong foundation in software and circuit principles. Furthermore, many group members have basic experience with signal processing principles from coursework or outside projects. On the mechanical side, our mechanical student (Forest Song) has extensive experience with robotics concepts. As a whole, our group has the foundation to tackle many technical problems, so it became important to find a problem that would allow everyone to contribute meaningfully. Each group member approaches this project with an open mind and willingness to learn, allowing us to take on a problem that we were not as familiar with going in. We eventually landed on this idea after accounting for our current abilities as well as what kinds of skills and technologies we wanted to explore further.

One of our main inspirations for this project is based on a video published by the University of Florida which demonstrates many of our project objectives [1]. In the video, the researchers are able to control the movement of a drone using a neural interfacing device (presumably an EEG). The user is able to think forward and the drone inches forward. While it may take some time for that thought to actually translates to an action on the drone, this video still demonstrates that this kind of neural control is possible. Since this videos publication in 2015, many organizations have tried to accomplish similar objectives with much progress [2]. A successful neural interface to control robots has the potential greatly enhance human interaction with technology and has many diverse applications.

The robot was designed to be a simple chassis that would allow us to perform tasks by receiving information from the EEG and camera. To navigate easily and effectively, it was decided that the robot would have a holonomic drivetrain. A holonomic drivetrain is a drive base that has the ability to move in any direction and rotate independently [3]. As such, it can strafe from side to side without needing to rotate. This is useful for navigation, because the robot will not have to change its orientation in order to move around an object.

There are several different types of holonomic drivetrains. The three main types are omni drive, swerve drive, and mecanum drive. Each have their own characteristics make them useful for their own purposes.

A B C

Figure 1: A) Omni Drive, B) Swerve Drive, C) Mecanum Drive

An omni drive consists of omni wheels placed at 45 degrees at the corners of the robot. Each omni wheel has rollers on the side, which allow the robot to strafe and go in a direction perpendicular to the wheel face.

A swerve drive consists of 4 tank wheels mounted at the corners of the robot. Each wheel module consists of two motors: a swerve motor and a drive motor. The drive motor rotates the wheel, while the swerve motor turns the wheel module, allowing the robot to move in any direction

regardless of rotation.

A mecanum drive has 4 mecanum wheels aligned and attached the same way that a tank drive works. Each mecanum wheel has rollers that are placed at a 45 degree angle. Based on the direction that each motor rotates, the robot can strafe in different directions.

The pros and cons of each drivetrain were analyzed when deciding which drivetrain to use:

	Pros	Cons
Omni	• Cheapest wheels • Simple to program • Lightweight	• Lack of torque/traction • Bad for outdoor transportation • Frame is not space efficient • Sensitive to floor irregularities
Swerve	• Good for outdoor transportation • Best torque out of 3 options • Challenging to design(good learning experience) • High load capacity • Simple wheels	• Extremely complex to design and program • Expensive • Heavy
Mecanum	• Somewhat good torque • Easy to design frame • Compact • High load capacity	• Somewhat difficult to program • Wheels are expensive • Complex conceptually • Sensitive to floor irregularities

Table 1: Pros and Cons of Different Holonomic Drivetrains

For the purposes of a navigation robot, it is necessary for the robot to be able to travel on a variety of surfaces. As such, omni drive was ruled out, because its lack of torque makes it difficult to go over sloped surfaces. When comparing swerve and mecanum, swerve overall would be the better choice. Mecanum wheels would wear out over time and has less torque overall, making it less suited for outdoor navigation. As such, a swerve drivetrain was initially designed.

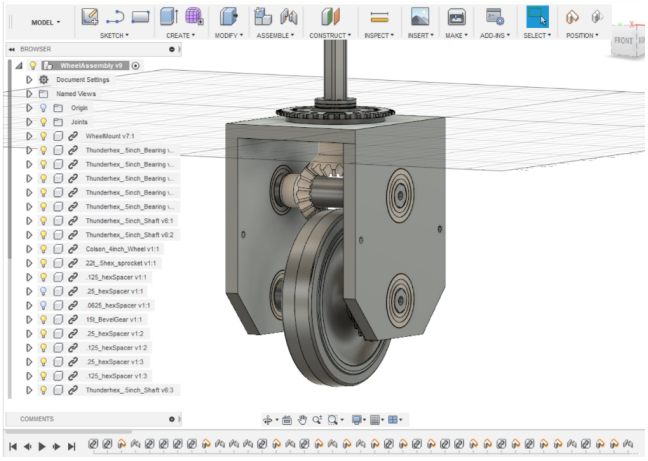


Figure 2: Initial Swerve Module

However, after doing research on the costs of a swerve module, ultimately it was deemed too expensive. The drivetrain itself would cost upwards of $500, while the budget allocated for the project was given as $200. As such, we decided to design the robot using a mecanum drivetrain, even though it was not as optimal as using a swerve drive. To further reduce our costs, we decided that we would design and 3D print our own mecanum wheels, which would be a lot cheaper than buying them.

One other integral part to this project is the EEG circuit itself. This is a simple circuit that was built based off the schematic that is explained in the next section. Since it is relatively cheap, it has a limited capacity in collecting brain-wave data when compared to traditional, commercial EEGs. This EEG is tuned to read alpha and beta waves which encompasses all brain waves that have a frequency from 8 to 40 Hz. These waves are the most important as they are the waves associated with conscious thought. Theta waves and delta waves are the waves below 8 Hz. Because these waves are typically associated with being at rest or being asleep, they are not useful for the purpose of this project.

As a whole, this project has many potential applications outside of what we are exploring. It aims to create a hands-free, interactive experience in which a user can control a robot without directly interfacing with it. While this project is bare bones in many aspects, it aims to demonstrate a proof of concept that it is possible to have thought based control of a robot on a low budget.

Equipment and Methodology

Equipment

Budget and Parts				
Item	Cost	Quantity	Total	Description
Wheel(3D Printed)	$17.55	4	$60.00	Mecanum wheel
Wheel Screws(25)	$8.96	2	$17.92	Screws for mecanum rollers
Bearings(10)	$15.23	4	$60.92	Bearings for rollers for free rotation
DC Motor(Drive Wheel)	$14.99	4	$59.96	Motors to drive the wheels
Motor Hub	$4.99	4	$19.96	Connects motor to frame
Raspberry Pi	$35.00	1	$35.00	For robot control
Motor Drivers	$16.99	2	$33.98	For motor control
Battery	$21.98	1	$21.98	Powering robot
Wheel Lock Nuts(100)	$4.18	1	$4.18	Nuts for mecanum rollers
Wheel Washers(100)	$3.86	1	$3.86	Spacers for mecanum rollers
AD620 Instrumentation Amp	$2.59	2	$5.18	The main IC for the EEG
3130 Op Amp	$10.06	2	$20.12	Needed for circuit design
1M Trimpot	$7.38	1	$7.38	Variable resistance for fine tuning circuit
Polarized Capacitors	$9.99	1	$9.99	Assorted capacitors for circuit design
Breadboard	$9.99	1	$9.99	Circuit prototyping
Arduino	$14.99	1	$14.99	Reads EEG data
Unpolarized Capacitors	$14.85	1	$14.85	Assorted capacitors for circuit design
Battery Holder	$8.99	1	$8.99	control voltage levels in circuit
Spectra 360 Electrode Gel	$6.29	1	$6.29	For readings when interfacing with electrodes
Gold Cup Electrodes	$43.23	1	$43.23	For interfacing with brain
		Total	$468.97	

Table 2: Initial Budget

Our budget and parts list is based off the research that we did in the methodology section below. Because we knew that we had a limited budget, we had to make several design changes to account for the lack of money that we had. We were able to quickly do this research and determine what equipment we needed based off our previous experiences in this field.

II. Methodology

Our project was broken down into discrete groups: the EEG group, the hardware group, and the integration group. Our deliverables are broken down in our Gantt Chart here:

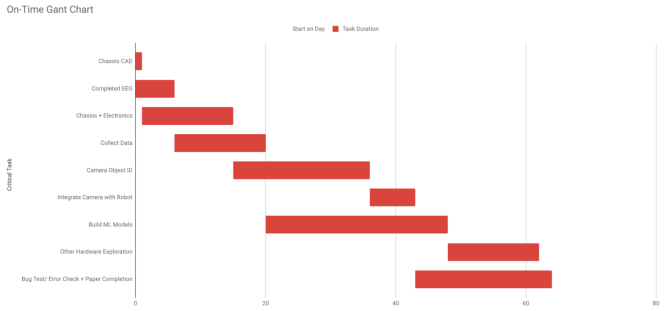


Figure 3: Gantt Chart Detailing our Timeline

This Gantt chart served as a basic timeline for when we would try to accomplish our tasks. However, we knew that with how busy we would get during semester, that things would not go to plan. As such, we made sure to evaluate our progress weekly and determine what we wanted to accomplish each week. We also split into three groups: Salman and Justin would work on the EEG, Forest would work on the robot, and Dan, Sam and Gaurav would work on integration between teams and computer vision. We decided to split up the work this way to maximize each of our strengths. We also realized that it would be difficult for our schedules to line up, so it would be more efficient if we met in smaller groups.

1) EEG Group: The EEG circuit is based on a simple schematic online. This circuit relies heavily on the AD620 instrumentation amplifier which is used to directly read in the input from the active electrodes. This instrumentation amp is designed so as to mitigate the effect of environmental noise that is always present. On the bottom of the schematic is a simple op-amp circuit which uses a 3130 amp. This circuit creates a floating ground at pin 6 of 2.5 volts which can be used at any other point in the circuit. The 3130 amp that is in the main circuit serves two purposes: to filter out high frequency waveforms and to amplify the output of the circuit. Since the waveforms read from the brain are at very low amplitudes, it becomes important to amplify those values through a combination of amplification from the AD620 and 3130 amp. Furthermore, the 3130 ensures that only low frequency values are outputted through the implementation of a low-pass filter. This circuit is specifically designed for use with and Arduino Uno.

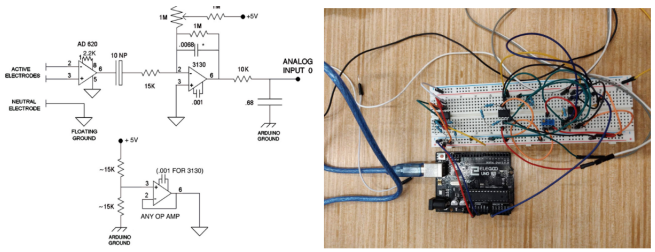


Figure 4: EEG Schematic for Arduino and EEG Prototype [4]

A. Hardware Group

Mecanum wheels were designed by a Swedish inventor named Bengt Ilon in 1975 [5]. The mecanum wheel consists of two main parts: the central wheel, and the outside rollers. These rollers are usually placed at 45 degrees with respect to the central wheel. Depending on the wheel direction and speed at which each wheel moves, each wheel produces a force vector.

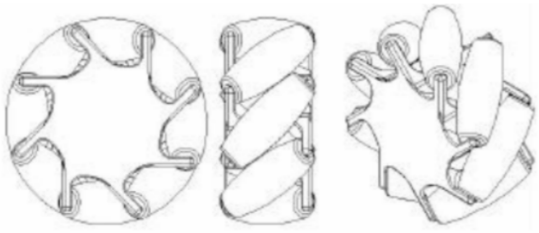


Figure 5: Mecanum Wheel

The sum of these vectors determines where the drivetrain will move; the combined force vector can be pointed at any direction, allowing the drive base to move in any direction regardless of direction. This is quite useful in tight spaces, and useful in our case for maneuvering around objects without changing orientation.

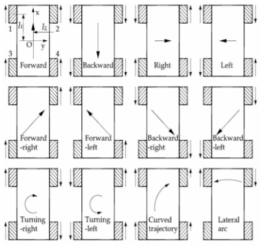
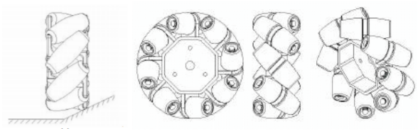


Figure 6: Options for Vehicle Motion

The drivetrain is able to translate in any direction, from forward to backward and also from left to right. For strafing from left to right and vice versa, the wheels have to rotate in different directions. It is able to go at diagonals by applying the same principles but also adjusting for the speed of each wheel.

There are some flaws with mecanum wheels that need to be accounted for. One such problem is that mecanum struggles to strafe up an incline when rollers are held in place from the outside. In this case, the rim of the mecanum wheel will often hit the ground surface, preventing the wheel from moving [6]. To avoid this, we will design our wheels to have centrally mounted rollers.



A B

Figure 7: A) Mecanum Struggles Strafing on Incline, B) Centrally Mounted Rollers

Centrally mounted rollers split up the peripheral roller into two parts, so the rim will not come into contact with the surface. This allows the robot to have more contact with the ground on uneven surfaces, at the cost of worse load carrying capacity.

The next step was to design the mecanum wheel itself. To start off, we had to figure out roller curvature. In order to strafe properly, the mecanum wheel has to be as close to circular as possible. If the wheels are not circular, the drivetrain will not be able to strafe without noticeable vertical deflection. As such, each roller has to have the perfect geometry in order to the wheel to remain circular. The values for the curvature will be calculated in Excel. These values will then be imported as a spline curve into Autodesk Fusion 360, which will allow us to create a 3D model of the roller.

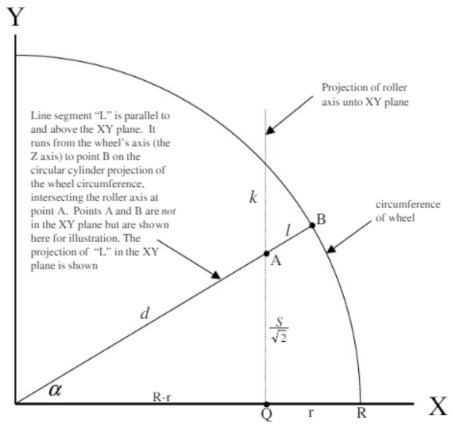


Figure 8: Algorithm for Roller Profile [7]

For a given R(wheel radius) and r(roller maximum radius), the curvature of a roller can be determined. Let us define parameter S as the distance along roller axis to intersection with Line L. Line L is the line from the wheels axis to a point on the projection of the wheels circumference, as shown in Figure x above. S will vary from 0 to half of the roller maximum length. This gives us the following equations:

$$\alpha = \tan^{-1}\left(\frac{S/\sqrt{2}}{(R-r)}\right)$$

$$d = \frac{R-r}{\cos(\alpha)}$$

$$l = R - d$$

$$\theta = \cos^{-1}\left(\frac{\sin(\alpha)}{\sqrt{2}}\right)$$

$$h = S + \frac{l\sin(\alpha)}{\sqrt{2}} \qquad Rr = l\sin(\theta)$$

Figure 9: Roller Formulas

Upon calculating these values, we can find out the given roller radius(Rr) at a given distance from the center(h). These values are defined in the figure above. Compiling these values in an Excel spreadsheet and graphing the results gives the curve shown in the figure below.

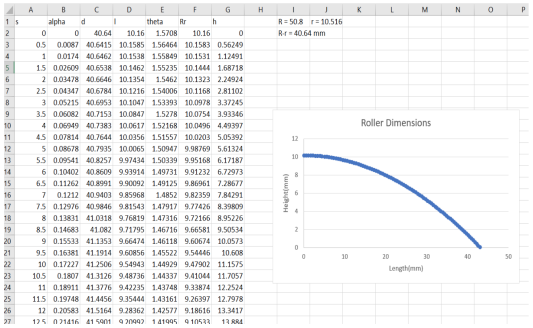
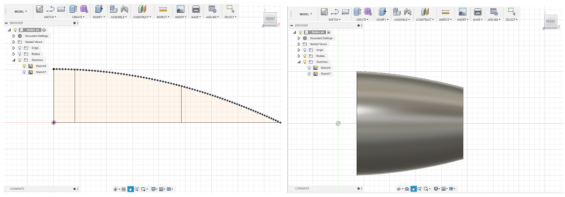


Figure 10: Roller Calculations and Graph for R = 2" (50.8 mm) and r = 0.413" (10.516 mm)

The values for Rr and h were then imported into Autodesk Fusion 360 as a spline curve. This resulting curve is shown in the figure below. This was then revolved to create a singular roller, with ends cut off to account for screw size, nut attachment, and washer spacing.



A B

Figure 11: A) Imported Spline Curve in Fusion, B) Created 3D model

The next step was to design the central wheel. The central wheel was designed with several things in mind: Having a certain thickness to ensure structural integrity, having a hole pattern that would allow it to be adapted to the motor hub that we bought, and having 8 holes for the rollers, which would have a centrally mounted bearing. The bearing ensures that the rollers rotate smoothly and make strafing for the wheels a lot easier.

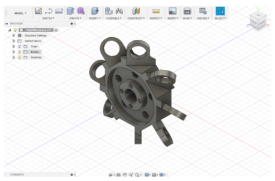
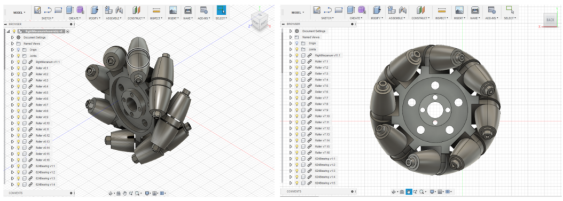


Figure 12: Central Wheel Design



A B

Figure 13: A) Full wheel assembly isometric view, B) Front view

The front view of the mecanum wheel assembly indicates that the wheel is indeed a perfect circle and the calculations of the rollers done previously were correct. We could then proceed

on to the next section, which was actually manufacturing the wheels, along with designing the physical chassis that would house the electronics, wheels, and motors.

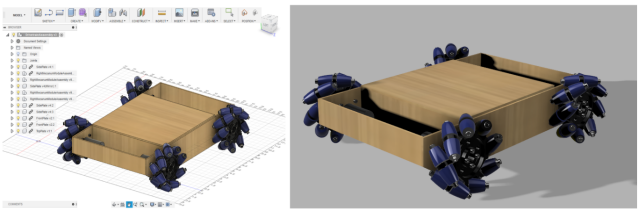


Figure 14: Robot Chassis Full CAD

The chassis frame was designed to be made out of several sheets of x 8 x 8 plywood. This would allow to be lightweight, strong, and cost effective. The center of the robot consists of a 2.5 x 8 x 8 volume box that would house all of the electronics. The motors and wheels would be mounted outside of that, making the robot approximately 14 long and 12 wide.

We also needed to choose an appropriate motor for our purposes. The robot needed to have enough torque to support its own weight and get over various surfaces, but also be fast enough to serve its purpose as a navigation robot.

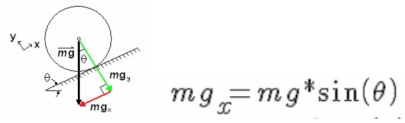


Figure 15: Free Body Diagram of Forces [8]

Given the forces acting on the robot on an incline, the necessary torque of the robots motors can be calculated. Through Newtons Second Law:

$$ma = f - mgsin$$
$$ma = T/r - mgsin$$
$$T = (\frac{100}{efficiency}) * (\frac{mr(a+g\sin\theta)}{N})$$

Where m = robot mass, f = force needed to move robot, = angle oF incline, T = torque needed to move robot, r = radius of wheel, a = acceleration of robot, and N = number of wheels

$$\text{By setting: } m = 25lbs = 11.34kg$$
$$r = 2" = 0.0508m$$
$$\alpha = 0.2m/s2$$
$$g = 9.81m/s2$$
$$\theta = 15degrees$$
$$N = 4$$
$$e = 50\%$$
$$\text{Torque} = 0.789N * m = 111.7oz * in$$

For the required speed of the robot, a persons walking speed was used as a measure. The average human walking speed is 1.4 m/s [9]. For a robot with wheel radius r = 2 = 0.0508m, the circumference of the wheel is c = 2πr = 2π(0.0508) = 0.318 m per rotation. As such, to travel at 1.4 m/s, the motors have to spin at a rate of 1.4/0.319= 4.486 rotations per second = 263.2 rpm. Therefore, the motors that we choose should have the requirements of having a torque greater than 111.7 oz*in and a speed of greater than 263.2 rpm. Based on these requirements, we decided on 170 rpm economy gear motors from ServoCity. They have a stall torque of 306 oz*in, and while the 170 rpm is less than 263.2 rpm required for walking speed, we decided having a higher torque was more important than the speed level. The motors were also mainly chosen for their relatively cheap price when compared with their counterparts of the same torque and rpms.

After doing these calculations, we built the chassis and 3D printed the wheels, as shown in the figures below.

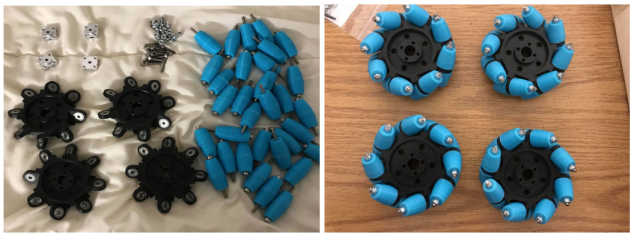
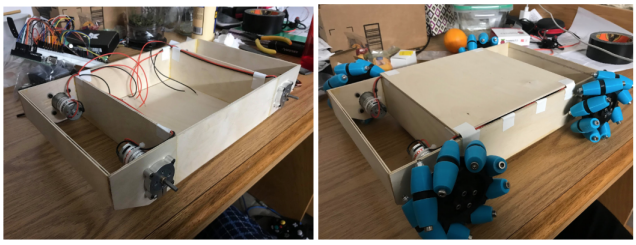


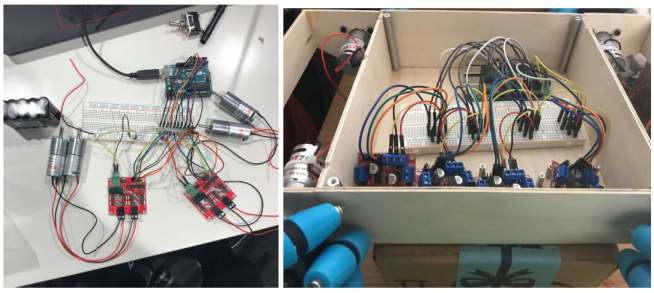
Figure 16: Printed Wheel Parts and Assembly



A B

Figure 17: A) Chassis with Motors Mounted, B) Full Chassis with Wheels

We also needed to wire the robot. The main setup consists of motors, motor drivers, a battery, and a brain to control everything. Initially, we used an Arduino to test the motors and make sure everything was working. We then switched to Raspberry Pi, to make integration with other parts of the project, namely computer vision, go more smoothly. This also required changing to the smaller L298N motor driver, as the larger motor driver was not compatible with the Raspberry Pi GPIO pins.



A B

Figure 18: A) Initial Wiring with Arduino, B) Wiring with Raspberry Pi

Figure x: Arduino Code

After we confirmed that the motor controllers worked, our next step was to make sure that the robot could be manually controlled. This involved connecting a PS3 controller to the Raspberry Pi through bluetooth, then reading joystick values. Based on the joystick values, the motor speeds and directions were determined by summing the positions of the left joysticks x and y axis for translation and the x axis for rotation.

```
#importing necessary functions
import pygame
import time
import RPi.GPIO as gpio

#Defining front right GPIO pin values
frENA = 2
frIN1 = 3
frIN2 = 4
#Defining front left GPIO pin values
flENA = 17
flIN1 = 27
flIN2 = 22
#Defining back left GPIO pin values
blENA = 10
blIN1 = 9
blIN2 = 11
#Defining back right GPIO pin values
brENA = 5
brIN1 = 6
brIN2 = 13

#setup for board and setting pins to output
gpio.setmode(gpio.BCM)
gpio.setwarnings(False)
gpio.setup(frENA,gpio.OUT)
gpio.setup(frIN1,gpio.OUT)
gpio.setup(frIN2,gpio.OUT)
gpio.setup(flENA,gpio.OUT)
gpio.setup(flIN1,gpio.OUT)
gpio.setup(flIN2,gpio.OUT)
gpio.setup(blENA,gpio.OUT)
gpio.setup(blIN1,gpio.OUT)
gpio.setup(blIN2,gpio.OUT)
gpio.setup(brENA,gpio.OUT)
gpio.setup(brIN1,gpio.OUT)
gpio.setup(brIN2,gpio.OUT)

#setting pwm frequency of pwm/ena pins
frPWM = gpio.PWM(frENA,1000)
flPWM = gpio.PWM(flENA,1000)
blPWM = gpio.PWM(blENA,1000)
brPWM = gpio.PWM(brENA,1000)

#duty cycle start of pwm pins
frPWM.start(25)
flPWM.start(25)
blPWM.start(25)
brPWM.start(25)

#designating axis numbers
leftJoyY = 1
leftJoyX = 0
#rightJoyY = 4
rightJoyX = 3

#initializing pygame for joystick control
pygame.init()
pygame.joystick.init()

#creating and initializing joystick object, 6 axis, 17 buttons
j= pygame.joystick.Joystick(0)
j.init()
'''
x = pygame.joystick.get_count()
print(x)
y = j.get_numaxes()
print(y)
z = j.get_numbuttons()
print(z)
'''

def motorVal(leftY,leftX,rightX):
    leftF = (leftY + leftX + rightX)
    rightF = (leftY - leftX - rightX)
    leftB = (leftY - leftX + rightX)
    rightB = (leftY + leftX - rightX)

    maxVal = max(abs(leftF),abs(leftB),abs(rightF),abs(rightB))
    if maxVal > 1:
        leftF = leftF/maxVal
        rightF = rightF/maxVal
        leftB = leftB/maxVal
        rightB = rightB/maxVal
    print (maxVal)
    flPWM.ChangeDutyCycle(abs(leftF*70))
    frPWM.ChangeDutyCycle(abs(rightF*70))
    blPWM.ChangeDutyCycle(abs(leftB*70))
    brPWM.ChangeDutyCycle(abs(rightB*70))

    if leftF < 0:
        gpio.output(flIN1,False) #reverse
        gpio.output(flIN2,True)
    else:
        gpio.output(flIN1,True)  #forward
        gpio.output(flIN2,False)

    if rightF < 0:
        gpio.output(frIN1,True)
        gpio.output(frIN2,False)
    else:
        gpio.output(frIN1,False)
        gpio.output(frIN2,True)

    if leftB < 0:
        gpio.output(blIN1,False)
        gpio.output(blIN2,True)
    else:
        gpio.output(blIN1,True)
        gpio.output(blIN2,False)

    if rightB < 0:
        gpio.output(brIN1,True)
        gpio.output(brIN2,False)
    else:
        gpio.output(brIN1,False)
        gpio.output(brIN2,True)

while True:
    events = pygame.event.get()
    for event in events:
        if event.type == pygame.JOYAXISMOTION:
            if event.axis == 1:
                leftJoyYPosition = j.get_axis(leftJoyY)
                leftJoyXPosition = j.get_axis(leftJoyX)
                rightJoyXPosition = j.get_axis(rightJoyX)
                rightJoyYPosition = -j.get_axis(rightJoyY)

                #print("leftX: ", str(round(leftJoyXPosition,2)), "leftY: ", str(round(leftJoyYPosition,2)), "rightX: ", str(round(rightJoyXPosition,2)), "rightY: ", str(round(rightJoyYPosition,2))
```

Figure 19: Raspberry Pi Code

B. Integration Group

The main objective of the integration group is to connect all of the components to a web server where we can display our data. The web framework we are choosing to use is Django, which is a python based framework and a model-view-template architecture. The web server needs to be run on a host computer, but we cannot use our laptops or desktops as hosts because we need our host to be attached to the robot. Therefore, we are using a Raspberry Pi mini-computer as our host.

The problem we faced with using the Raspberry Pi is that it is typically used by connecting it to a monitor, keyboard, and mouse, which would not be possible to attach to the robot. Hence, we had to figure out a way to make the Raspberry Pi headless, meaning we had to just have the pi as well as a power source and nothing else. The most obvious solution was to access the Pi via SSH. SSH is a networking protocol which allows you to establish a network between two computers and then access another computers terminal from your terminal.

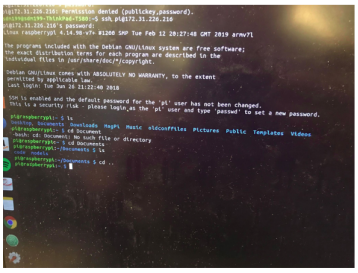


Figure 20: SSH into Raspberry Pi from Laptop

We faced several barriers when trying to make the Pi headless. The first one was that we didnt have a network manager installed on the device, so we werent able to connect it to the WiFi. As a result we had to install the manger, update the Pis software, and also edit the interfaces and dhcpcd files of the Pi so that we could actually connect it to the internet. The next problem we faced was getting the Pis IP address, which we needed to have in order to SSH into it. Any device which is connected to the internet has an IP address. When a device connects to a network, it connects through a router, and as a result its IP address is a subnet of the routers address. So when we connect to different networks with different routers, the IP address of the Pi will change to be a subnet of the router its connected to. Therefore, we will never know in advance what the Raspberry Pis IP address will be so we had to figure out a way to find it.

Finding the Pis IP address is contingent on the Pi being connected to the internet. If it is not connected to the internet, then it does not have an IP address and we would not be able to SSH into it as desired. This turned out to be the biggest issue we faced in making the Pi headless, namely figuring out how exactly to ensure that the Pi would connect to the internet on boot without us having to manually connect it to a monitor, open up the network manager, and connect it to the WiFi manually. After much research, we were able to get it to connect to the WiFi on boot by specifying the network configurations in the wpa-supplicant module and telling it to connect to that particular network automatically in the interfaces module.

Figure 21: Headless Pi Accessed via SSH

Finally, once we were able to ensure that the Pi would connect to the internet on boot we were almost done making the Pi completely headless. The last issue we had to figure out was how to find the IP address of the Pi. The solution we came up was to just make the computer list all devices whose addresses were subnets of the router on the WiFi network we wanted to connect to. Since our Pi was connected to the WiFi, we knew that it had to be a member of this list. So

once we would locate the Raspberry Pi on the list we would know its IP address which is listed along with the devices hostname. With the Pis IP address, we were then able to SSH into the Pi through our computers and able to run files on the Pi remotely as we desired.

Using our ability to SSH into the Pi, we then began to integrate our components. One of the main reasons that we needed to use the Pi in the first place was because we were using a Pi camera, which is simply a small eight megapixel camera attached directly into the Pi, as well as an ultrasonic distance sensor, which calculates the distance from itself to an object by shooting a wave at it and then receiving it back, and is also wired to the Pi. We needed to be able to interface with these devices and the only way that we could do that was to execute programs locally within the Pi. The purpose of the camera is for the robot to be able to see what is in front of it and be able to communicate with a person that there is an object in front of it, and the purpose of the distance sensor is to tell the person that it is within some distance of an object. In order to make these devices execute, we needed a multithreading program on the Pi that concurrently took a picture and read a distance.

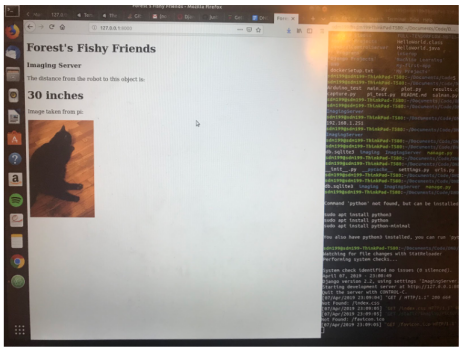


Figure 22: Simple Webpage to Display Data

We then wanted to be able to display our results on a webpage where we could see what the robot was seeing, and also see other useful information like the distance. We used the Django web framework because we already had some familiarity with it and it is pretty easy to use. The simple web page displayed above only shows the distance read by the sensor and a picture taken by the Pi. These data are displayed by running the program mentioned above which reads the data and then sends post requests to the server, telling it to display the data it has read. We are still working on our server, not all components are connected as of yet. Also, we will improve the design of the webpage and also increase the servers functionality to show even more useful information.

The next part that we worked on was reading and receiving distances. In order to accomplish this, we were initially planning on using OpenCV and machine learning to determine how far away objects were by comparing how much bigger an object becomes as more pictures are taken over time. However, we determined a better solution was to use an ultrasonic sensor called the HC-SR04 that is able to sense how far away the closest object is to it.

The HC-SR04 accomplishes this by sending ultrasound waves at 40 kHz through the air. If the waves hit an object, then they bounce back. So in order to calculate how far away an object is, we check to see how long it takes for the waves to bounce back and multiply it by the speed of the ultrasound wave.

The HC-SR04 ultrasonic sensor is only capable of sending out waves and detecting when ultrasound waves are received back [10]. Therefore, any kind of math that is used to calculate how far away the object is is done in a python script.

First, the start time of the ultrasound wave is recorded in a variable called pulse-start-time. Then the time that the ultrasound wave is received is also recorded in a variable called pulse-end-time. We subtract the two values and that is the amount of time that the ultrasound wave spent in the air. This value is stored in a variable called pulse-duration. Using our knowledge that the speed of an ultrasound wave is approximately 34300 cm/s, we use a formula 34300*pulse-duration/2 to find the distance in cm of the object that we are detecting [11]. We divide our result by 2 because we only need the distance between the object and the sensor, and the pulse-duration accounts for both the time it takes to get to the object and back.

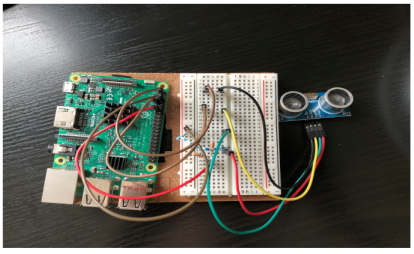
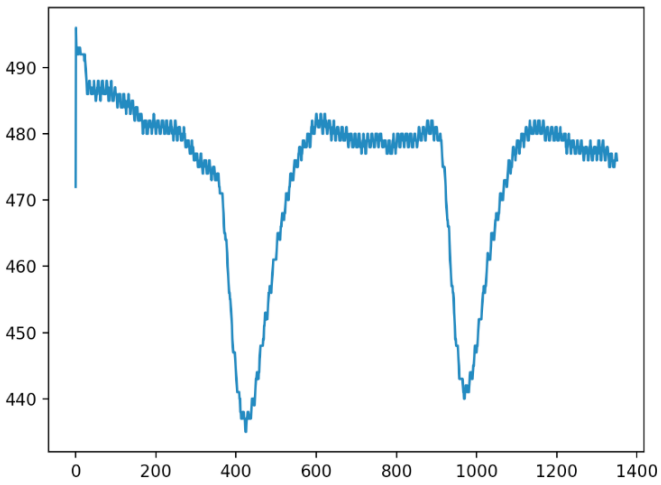


Figure 23: Sensor Circuit

In order for the robot to navigate through obstacles it would have to process the images it received from the camera and we determined that using OpenCV was the best way to give the robot vision. We installed the OpenCV 3 libraries onto the Pi and looked into how they could be used for the robot to understand the orientation of his surroundings. Using our webcam, visual studios, and the windows version of OpenCV, we were able to track objects using a variety of functions. For starters, we used the cvtColor() function to convert the BGR (blue green red) binary input feed from our camera into HSV (hue saturation value). Using HSV made it much easier to filter different types of colors allowing us to seperate objects in front of the robot from objects in the background. In addition, the function createTrackbar() was used in conjunction with this color conversion to allow us to filter out different colors and work with the sliders, in real time, to see which settings would work best to track certain colors. We used the erode() and dilate() functions to get rid of any filtered holes within the object we were trying to track and remove any extra random

pixels outside of the object that were not filtered out. Next up, we used the findContours() function on the image outputted by the color filtering which returned a vector of contours of the white spaces in the filtered image. From these vectors we would choose the largest contour using the moments() function to find the contour with the largest enclosed area. Then, finally, were able to track an object using our webcam. More research must be done into ways of tracking objects regardless of their color but this work has given us a strong basis in OpenCV to go off of when we attempt to integrate this software with the robot's camera.



III. Results and Discussion

A. EEG Group

Throughout the process of creating the EEG, we faced many issues such as being unable to view the data on our computers, not having any output at all from the circuit, or too much noise in the data. Currently, the circuit works and we are able to extract the data from it for use in other programs. We have not yet developed the programs which rely on the EEG data.

Initially, we were able to view the data using an Arduino and Processing script that was made available with the EEG schematic. From these scripts, we were able to produce a visual output as shown in the image below. This output was shown through the Processing script which displayed the waveform in real time. However, this output would not be good enough for us to use since we cannot actually extract the data from the output. Furthermore, the script is old and written in C which would not be a good language to use moving forward. Thus, we found it important to interact with the data in a Python script. This will help us later when it comes to integrating all parts of the project since Python is the most versatile language to use with extensive support. The Python data outputs are also shown below.

We were able to successfully read data from the EEG. Previously we were misreading the data. We were able to detect blinks using simple calculus to find local minima. Using the blinks, we were able to codify the the blinks into actions.

B. Hardware Group

The robot chassis, wheel design, arm end effector, and wiring are all complete. As such, the robot is successfully able to move forward/backward, and strafe from side to side. This is essential for our navigation. Additionally, the robot can be controlled manually using a PS3 controller, which can be useful for demonstrations and testing.

Results from testing show that the mecanum wheels perform quite well when moving, and can strafe quite effectively. This illustrates that the calculations done in the methodology section proved effective. The robot is also able to move well with the motors chosen, showing that the torque calculations done in that sections were also effective.

Similarly, the end effector is effective at picking up tennis balls. This is important, as it can give firm results in making sure that the software is working well and integrated effectively with the hardware.

C. Integration Group

We have made some progress on three fronts: making the Raspberry Pi headless, reading pictures and distances from the Raspberry Pi camera and HC-SR04 ultrasonic sensor, and using openCV algorithms on pictures. We have gone over much of our progress so far in the methodology section, so in this section we will give some more detail on our results and what we plan on doing with them.

Our first task was to make the Raspberry Pi headless, which we have mostly accomplished. We are now able to connect the Pi to the internet on boot, find its IP address, and most importantly SSH into it from a remote computer.

The next step was to be able to receive data coming from the sensor and camera on the Pi. We made a Django web server which contains web pages to display the data. So far, we have implemented a homepage that introduces our project and an imaging server which contains the pictures

and distances read by the Pi.

We have also been working on server logic. The point of the server is to respond to http requests made to it. We have a script which uses multithreading to concurrently take the picture and get the reading, and then sends a post request to our server which then handles that request by posting the image and distance to the imaging webpage.

Lastly, we have been learning about computer vision algorithms and have implemented a few basic algorithms that are described in the methodology section. Ultimately, we hope to be able to use classification algorithms so that the robot can detect certain objects in front of it.

IV. Conclusion

Our primary objective was to facilitate communication between all the disparate components, discern a clear and reproducible output from the EEG, and begin to autotomize, however crudely, the hardware components.

We developed a web server to communicate between all three parts. We have been working on receiving information from the pi to the server and our next step was to communicate with the robot. Based on the pictures and readings gathered by the Pi, we were able to communicate to the robot that its path is obstructed so that it knows it has to move in another direction. After we are able to communicate between the robot and the server, we were also able to communicate between the EEG and the server. This is a separate task from communicating between the EEG and the robot. In this case, we gather the number of blinks our user did in a certain time period, and pushed the data to our server . The purpose of our web server, after all, is to convey the data to the user. Finally, we need to establish a communication line between the EEG and robot. However, it may make more sense to not use a web server for this line of communication because web servers are used to handle HTTP requests, not just general communication that would be used between the two. As a result, we are considered some other client-server model such as interfacing between sockets to communicate between the two. Integrating all three parts allowed us to seamlessly create a nice end user experience, working towards the social good objective.

A major task in the EEG space is to reduce noise. We investigated the potential impedance of the circuit in the first place, and introduce a filter to remove the noise. Furthermore, our physical circuit was cleaned up by shortening wires, cleaning contacts, and re-calibrating the trim-potentiometers.

The hardware components have been completed for the most part. To make the robot more user friendly, the electronics were cleaned up, and several switches were added. This will made it easier to power the robot and charge the batteries. Additionally, some hardware advancements were designed in order to make the uses of the robot more apparent.

In conclusion, good progress has been made in reference to each individual project group, and we achieved our objective of integrating all three parts of our project with a common goal of having our robot move. Our robot is able to move

Acknowledgment

The authors would like to thank Dean Antoine, Professor Yates, Supraja, and the rest of the Design Advisers for their hard work and dedication towards supporting the Design and Development course.

References

[1] https://www.youtube.com/watch?v=hLjxMjBlB9k
[2] https://www.thedrive.com/tech/19498/emotivs-headset-reads-your-brain-and-lets-you-control-drones-with-your-mind
[3] http://ftckey.com/build/drive-trains/
[4] https://sites.google.com/site/chipstein/home-page/building-the-amplifier-2
[5] https://en.wikipedia.org/wiki/Mecanum_wheel
[6] http://www.incdmtm.ro/mecahitech2011/articole/Pp112-123.pdf
[7] https://dokumen.tips/documents/contour-of-a-bumpless-mecanum-roller.html
[8] https://www.robotshop.com/community/blog/show/drive-motor-sizing-tool
[9] https://en.wikipedia.org/wiki/Walking
[10] https://pimylifeup.com/raspberry-pi-distance-sensor/
[11] https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/