

Department of Mechatronics Engineering

Critical Design Review

Autonomous Wheelchair System (AWC)

Lucky Patel, Austin Clark, Duc Nguyen

Mechatronics System Design – Summer 2021

Mechatronics Engineering

Kennesaw State University

07/28/2021

**Table of Contents**

List of Figures

List of Tables

Executive Summary (Austin)

1. Introduction

1.1 Project Overview (Austin)

1.2 Major Objectives and Developments (Austin)

1.3 Team Chart and Major Responsibility (Austin)

2. System Requirements & Specifications

2.1 Technical Requirements and Specifications (Austin)

2.2 Functional Features (Austin)

2.3 Minimum Success Criteria (Austin)

2.4 Design Verification Plan (Austin)

3. System Overview

3.1 Configuration (Lucky and Austin)

3.2 Standard Compliance (Lucky)

3.3 Major Subsystem and Components (Lucky)

3.4 Trade Study for Major Parts (Lucky)

4. Prototype Development

4.1 Introduction of Major Works (Duc)

4.2 Hardware (Duc)

4.3 Software (Duc)

4.4 Budget and Project Schedule (Austin)

4.5 Analysis and Test Results (Duc)

5. Conclusions

5.1 Achievements and Lessons Learned (Austin)

5.2 Future Improvements and Design Optimization (Austin)

References

Appendices

Appendix A: CAD Drawings (Lucky)

Appendix B: Codes (Austin)

Appendix C: Team member contribution Table (Austin)

**List of Figures**

|  |  |
| --- | --- |
| **Figure Name** | **Page #** |
| Figure 1: Block Diagram | 13 |
| Figure 2: Circuit Diagram | 14 |
| Figure 3: Project Schedule | 19 |
| Figure 4: 22 mm joint | 23 |
| Figure 5: LCD RPI Mount | 24 |
| Figure 6: RPI Shell | 25 |
| Figure 7: RPI to Leg | 26 |
| Figure 8: Ultrasonic Sensor Mount | 27 |
| Figure 9: Wheelchair Arm | 28 |

**List of Tables**

|  |  |
| --- | --- |
| **Figure Name** | **Page #** |
| Table 1: Team Member Roles | 9 |
| Table 2: Verification Approach/Plan | 12 |
| Table 3: Base Model vs New Installation | 12 |
| Table 4: Arduino Weighted Rating | 16 |
| Table 5: Sensor Weighted Rating | 16 |
| Table 6: Bill of Materials | 19 |
| Table 7: Optimizations | 21 |
| Table 8: Team Technical Contribution | 53 |

**Executive Summary (Austin)**

In this paper the process of articulating and building an autonomous obstacle avoiding an electric wheelchair is described. A Raspberry pi cam is connected to a Raspberry pi 4 model B to detect follow lines and obstacles in its path. The raspberry pi is then connected to both a touchscreen for gui implementation and an arduino for motion controls and reading in ultrasonic sensor data. The ultrasonic sensors detect objects as well as measure a distance of 50 inches from the wall and using a PI controller keep the wheelchair within 10 inches of that distance. These items are secured to various 3D printed mounts. It operates inside an area designed to take advantage of its object detection, line tracking, and wall following capabilities. The goal of this project is for the chair to be able to travel autonomously through an area safely without issue for at least 60 seconds and detect objects within 50 inches of collision.

1. **Introduction**
   1. **Project Overview (Austin)**

The autonomous Wheelchair system implements obstacle avoidance, wall following, and line following using ultrasonic sensors and a raspberry pi camera. The 3D printed structures are created as a kit, therefore allowing it to be versatile. The electric system consists of a Raspberry Pi 4B microprocessor, 2 ultrasonic sensors, a Raspberry Pi cam, and other important components. The Pi cam is used to track the preplaced lines and the ultrasonic sensors are used for obstacle avoidance as well as wall tracking.

* 1. **Major Objectives and Developments (Austin)**

The creation of the AWC consisted of 3 main categories: electrical, mechanical, and programming. The mechanical portion consists of creating mounts for the sensors, chassis for the electrical components, and a case for the touchscreen. This section is 75% done. Most of these parts have been 3D modeled; however they still need printing and slight measurement adjustments. The next section is the electrical engineering portion. This section consists of wiring the components and choosing the best pieces for the job. In AWC a capacitor and pull down resistor is necessary in order to keep the system powered on after autonomous mode. The best resistor and capacitor for the job still needs to be determined and the cables need more support and hiding. Therefore the electrical portion is at 60%. The final category in the AWC is programming. In this system programming needs to be done on both the arduino as well as the raspberry pi making this the most lengthy and difficult section in the AWC. Currently the programming is around 60% done. Using line-tracking on the raspberry pi and the pi cam we are currently able to follow a straight line however the coding needs more adjustment because it is not within the ideal success rate. The next portion of programming is on the arduino for wall following. Currently using ultrasonic sensors the AWC is able to follow a wall given a desired range and maintain that distance within 10 inches of that value using PI control.

* 1. **Team Chart and Major Responsibility (Austin)**

**Table 1: Team member roles**

|  |  |  |
| --- | --- | --- |
| **Team Member** | **Major Roles** | **Support Roles** |
| Lucky Patel | Team Leader, CAD designer, Physical Assembly | Electrical Assembly |
| Austin Clark | Wall Following and Object detection programmer, Bluetooth controller programmer, Electrical Assembly, Parts analyst, Documentation leader | Physical Assembly |
| Duc Nguyen | Line Following Programmer |  |

1. **System Requirements and Specifications**

**2.1 Technical Requirements and Specifications (Austin)**

The AWC has a base of an electric wheelchair of dimensions (LxWxH) 38in x 27in x 38in. It is able to travel manually via bluetooth control or joystick until the wall following or line following buttons are pressed. Once one of the autonomous modes is activated the AWC will either move via line tracking or wall following throughout the environment. Both modes contain object detection allowing safe movement throughout the environment. The bluetooth functionality is implemented using an HC-08 bluetooth module and controlled on a phone using the elegoo bluetooth tool app. In order to properly track lines the raspberry pi camera is mounted at least 25 inches above the top of the chair. The mounts for the microcontrollers and sensors are included below in the appendix.

**2.2 Functional Features (Austin)**

The autonomous functionality of the AWC allows for a minimum of 60 seconds of movement. In wall following mode the AWC stays a distance of 50 inches from the wall with an error range of 10 inches. During the line following phase, the AWC follows within 2 preplaced lines until stop or run time ends. The user is able to not only manually control the AWC via bluetooth control but both autonomous modes can be triggered this way as well.

**2.3 Minimum Success Criteria (Austin)**

In order for the AWC to be successful it must meet the minimum criteria as determined by the design team. The first of such criteria is that in addition to having an autonomous mode it must keep its manual functionality. Keeping this will provide a greater number of environments for the AWC to travel. In addition the purpose of this project is to improve upon a normal electric wheelchair, removing the manual functionality would be a step backward and not forward. The next requirement is that in wall following autonomous mode the AWC should keep a distance of 60 in from the wall with a 10 in margin of error, while in line-tracking mode the AWC should remain inside the lines it is tracking. The last criteria is that the AWC must be capable of remaining operational for at least 60 seconds without running into objects or people.

**2.4 Design Verification Plan (Austin)**

While designing the AWC it is necessary to determine what can be done in order to better plan out the different systems; this data is shown below.

**Table 2: Verification Approach/Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Systems** | **Analysis** | **Model** | **Simulation** | **Testing** |
| Microcontrollers | **✔** |  |  | **✔** |
| Batteries | **✔** |  |  | **✔** |
| Sensors | **✔** |  | **✔** | **✔** |
| Camera | **✔** |  | **✔** | **✔** |
| Mounts | **✔** | **✔** | **✔** | **✔** |

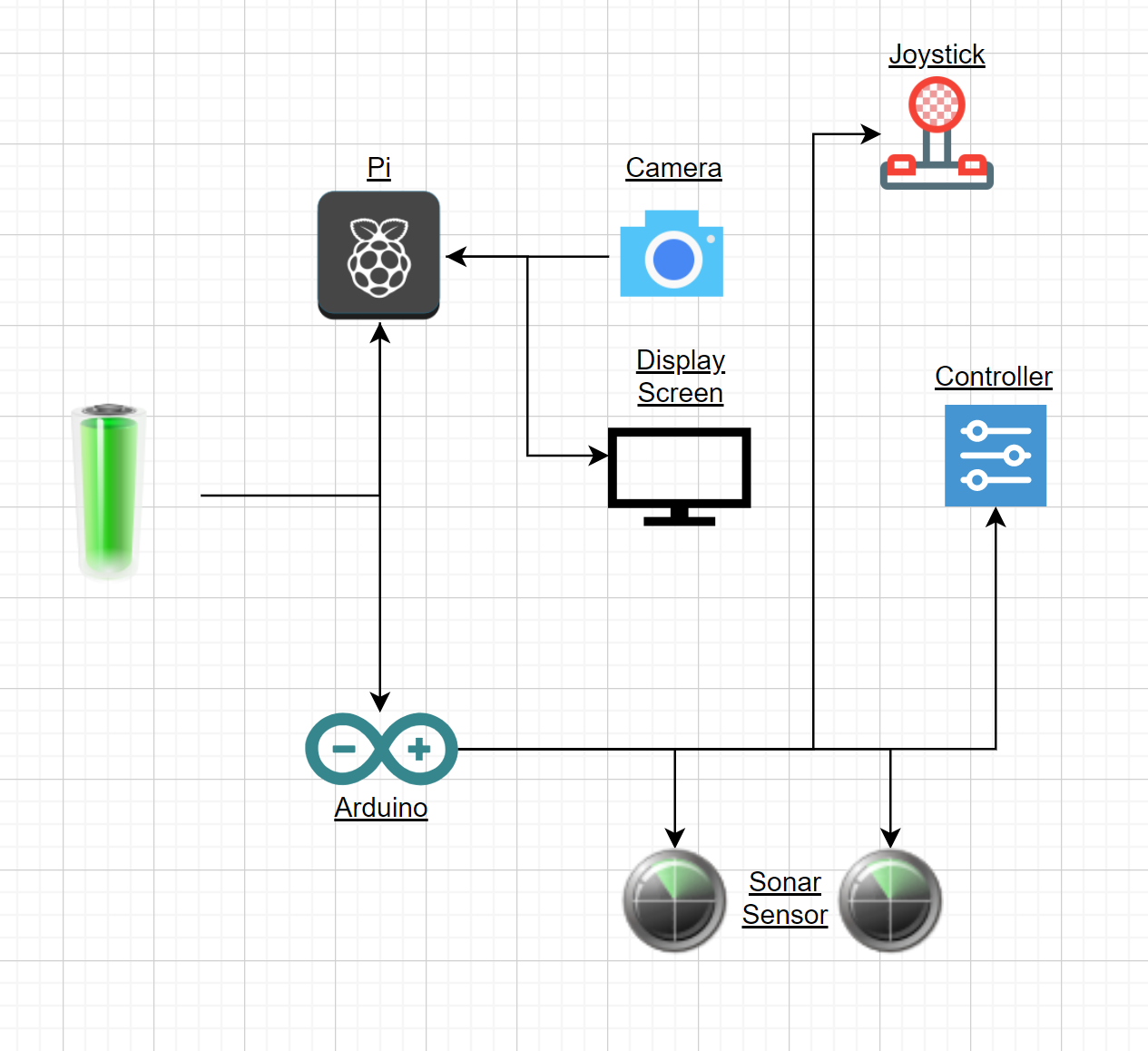
**Table 3: Base Model vs New Installation**

|  |  |
| --- | --- |
| **Using the pre-existing components** | |
| **Pros** | **Cons** |
| Contains a voltage Limiter | Less flexible |
| Less intrusive | Restricted to used components |
| Less wiring | Guess work in programming |
| Less new parts installed |  |

1. **System Overview**

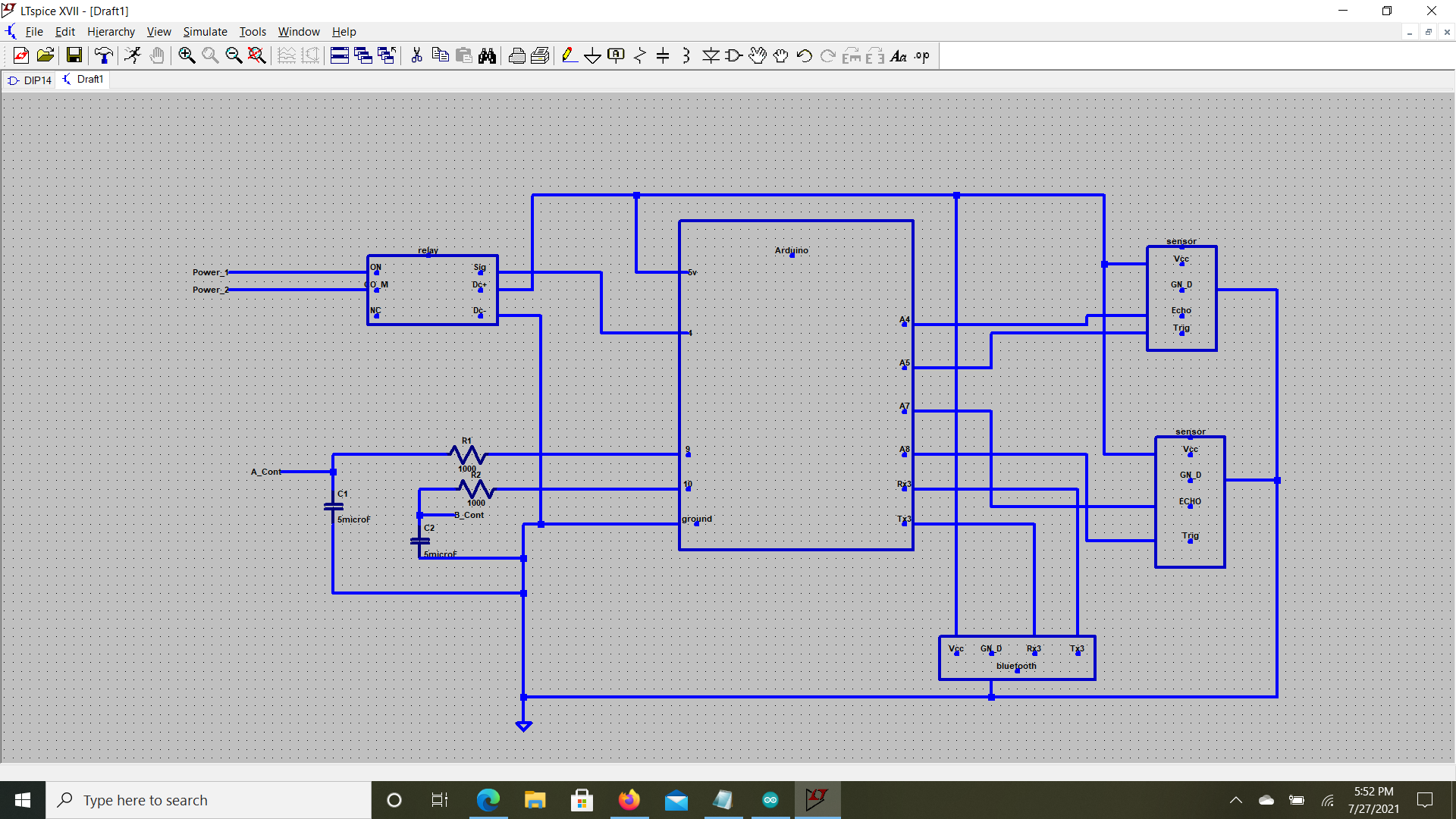
**3.1 Configuration (Austin and Lucky)**

**Figure 1: Block Diagram**



In the AWC the battery pack supplies power to the Raspberry pi which in turn is connected to the pi camera, the touchscreen, and the arduino atmega 2560. Next the arduino connects the 2 ultrasonic sensors to the physical joystick, and the bluetooth module for the control.

**Figure 2: Circuit Diagram**

****

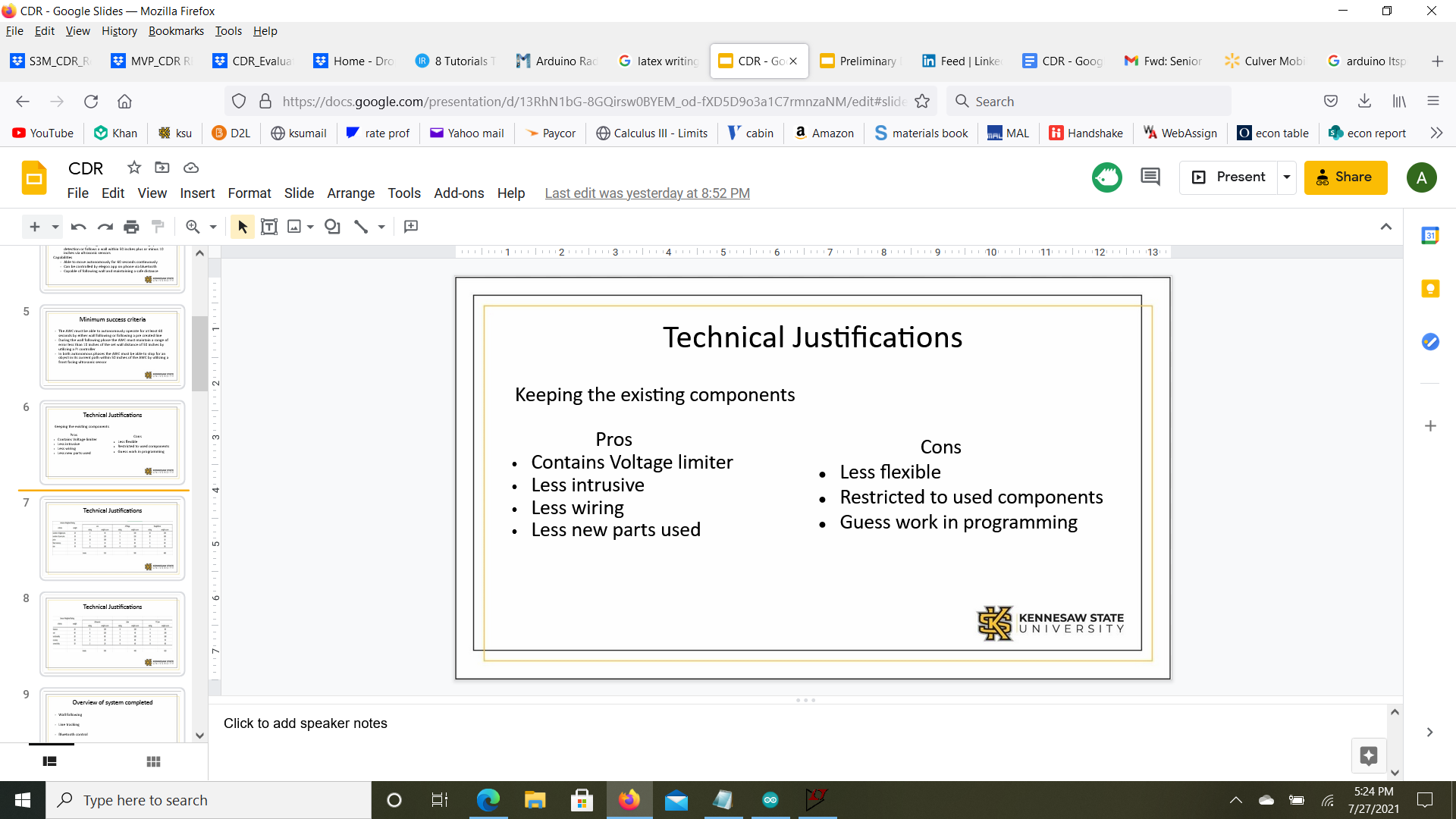
**3.2 Standard Compliance (Lucky)**

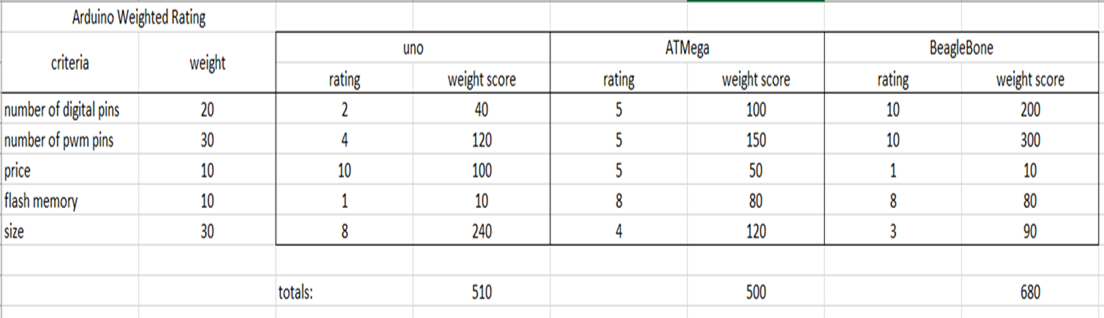
* The minimum clear width for single wheelchair passage shall be 32 in (815 mm) at a point and 36 in (915 mm) continuously.
* The space required for a wheelchair to make a 180-degree turn is a clear space of 60 in (1525 mm) diameter or a T-shaped space.

**3.3 Major Subsystem and Components (Lucky)**

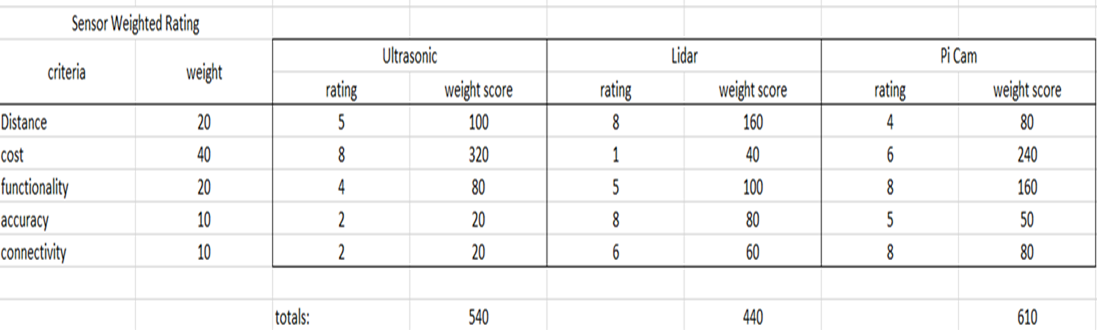
The major subsystems allowing this system to run would be the mobile wheelchair. It contains components capable of running Linux OS, bluetooth, 2 motors, usb c to usb a cable and battery power cable. For the autonomous wheelchair we chose to use Raspberry pi 4 and Arduno mega since it was the best for the budget and capabilities. additional components were added to the wheelchair to make it a complete prototype which include 2 sonar sensors for wall follow, 1 raspberry Pi cam for line tracking, joystick for manual control, powerbank to power the Arduino and Raspberry pi, A touch screen to control and run diagnosis on the chair, relays for power, And 3d printed components to hold all the extra components. without all the subsystems in the wheelchair there wouldn't be a complete prototype.

**3.4 Trade Study for Major Parts (Lucky)**

****

**Table 4: Arduino Weighted Rating**

**Table 5: Sensor Weighted Rating**

****

1. **Prototype Development**

**4.1 Introduction of Major Works (Duc)**

* Mechanical assembly completed:

- Designed, 3D printed, and installed designs for: a camera mount, ultrasonic sensors, the raspberry pi, the pi touchscreen, the arduino, and the breadboard.

* Electrical System completed:

- Wires labelled, soldered, and secured.

- Battery pack for Raspberry Pi chosen and is in use.

* Programming completed:

- OpenCv line tracking following white lines in an area with adequate lighting

- Ultrasonic wall following works with a 50 in distance from the wall with 10 inches error range.

- Bluetooth control is implemented for basic control and wall follow mode.

- Object detection in use during ultrasonic mode.

**4.2 Hardware (Duc)**

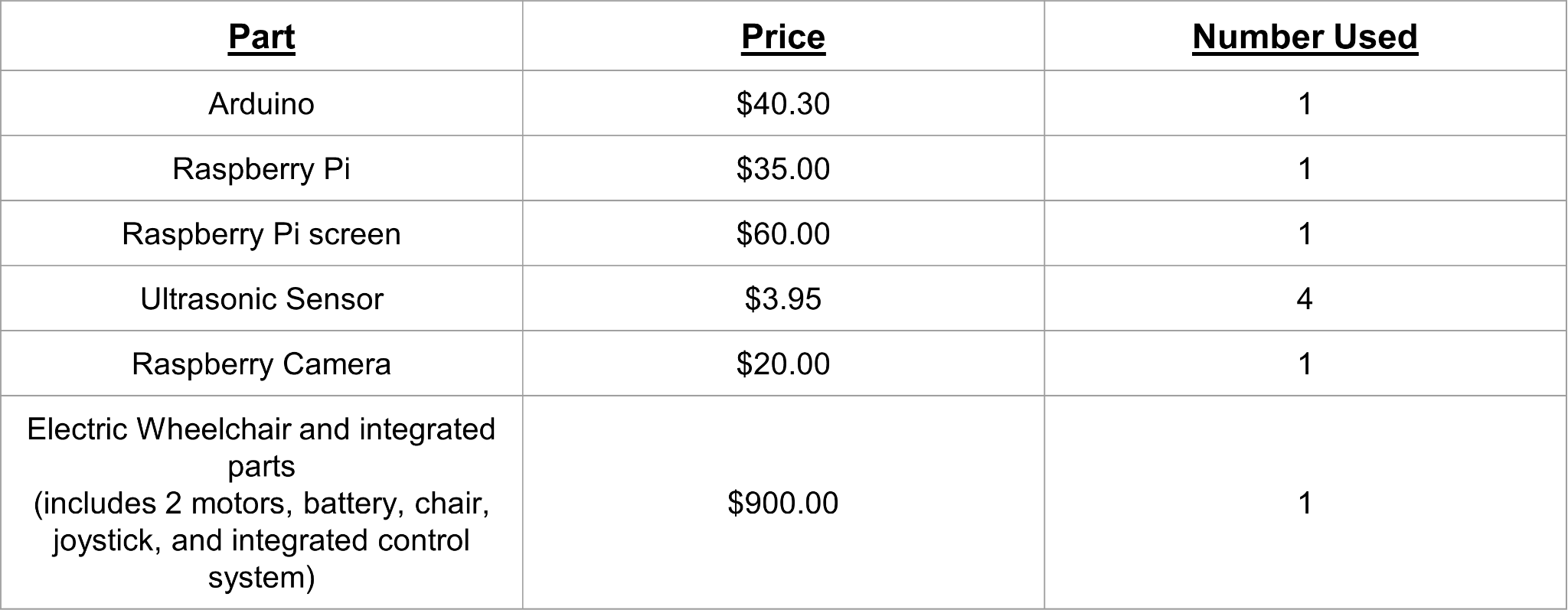
* Raspberry pi 4(Model B);
* Arduino(Mega);
* Camera;
* Ultrasonic sensors;
* Wires;
* PPC pipes;
* Breadboard;
* 3d Printed PLA;
* Power bank;
* Screws and nuts;
* Powered tools;
* Lubricant;
* Miscellaneous electronic components;
* Wheelchair Integrated system.

**4.3 Software (Duc)**

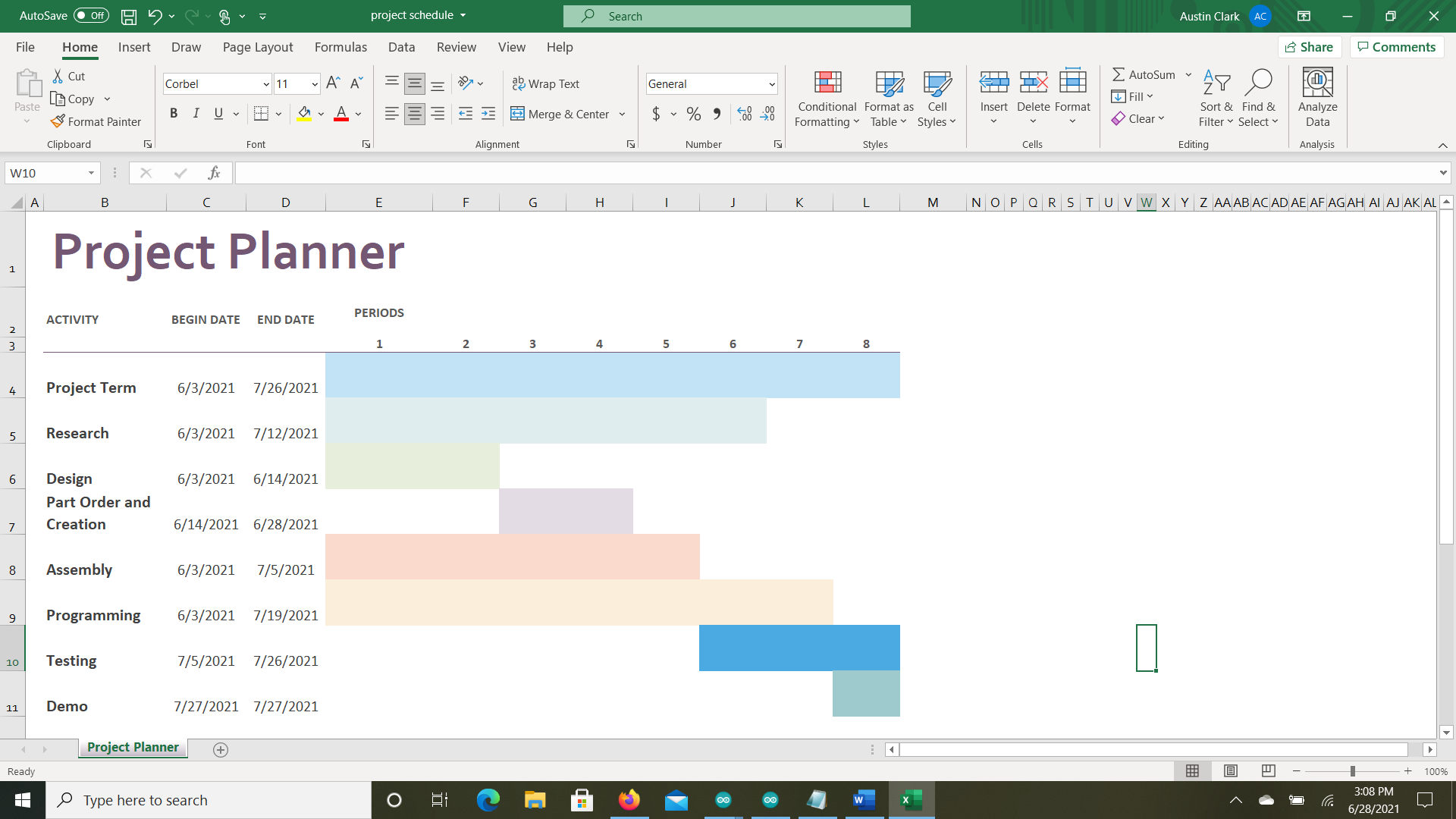
* C++(Geany);
* OpenCV;
* Python;
* Arduino(IDE);
* Solidworks;
* TinkerCAD;
* Thinkgiverse;
* Elegoo Bluetooth App;
* CircuitLab.

**4.4 Budget and Project Schedule (Austin)**

**Table 6: Bill of Materials**

****

**Figure 3: Project Schedule**

****

**4.5 Analysis and Test Results (Duc)**

**•Wall method:**

* Best testing area
* Adjusting ultrasonic sensor readings
* PI controller testing and creation
* Object detection

**•Line method:**

* Lanes created
* Battery used
* Area of Interest size
* Lane recognition

1. **Conclusions**

**5.1 Achievements and Lessons Learned (Austin)**

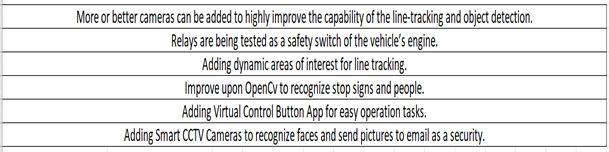
During the course of this project many lessons were learned below is a short list of the most valuable lessons.

* Teamwork is essential
* Break big tasks down into smaller more manageable tasks
* Document everything
* Don’t be afraid to ask for advice
* There is more than 1 way to accomplish a task
* Workspace organization is key

**5.2 Future Improvements and Design Optimization (Austin)**

With any project there are time and budget constraints which limit the use of certain products or methods that may improve the functionality of the system. In the list below there are options for possible improvements should the resources allow or should another group take over the project.

**Table 7: Optimizations**

****

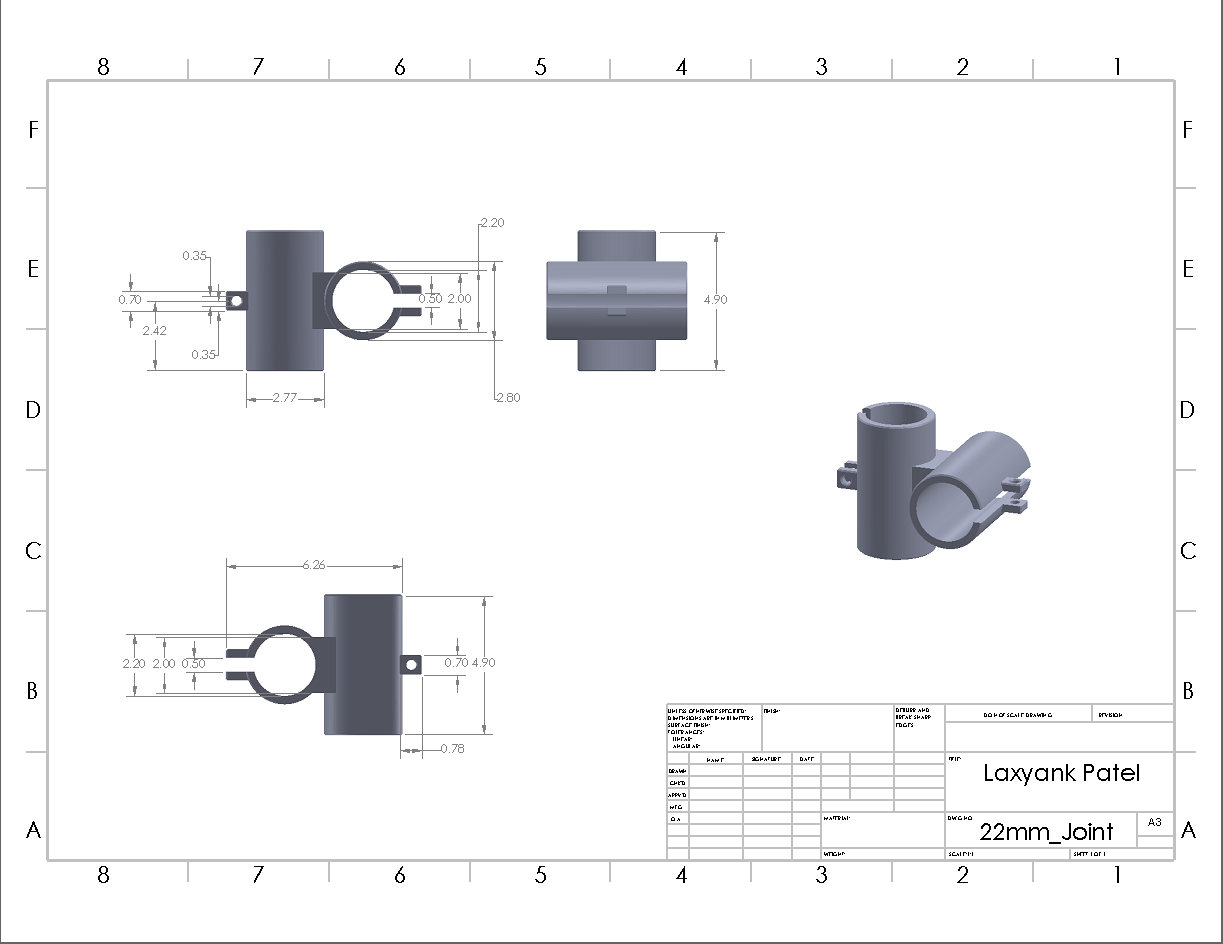
**References**

* Mentors
  + Chan Ham
  + Chris Voicu
* Facility
  + Engineering Lab room Q118
* Hardware
  + Pi Camera
  + Raspberry pi
  + Electric Wheelchair
  + Arduino
* Software
  + C++/Python/Arduino
  + TinkerCAD

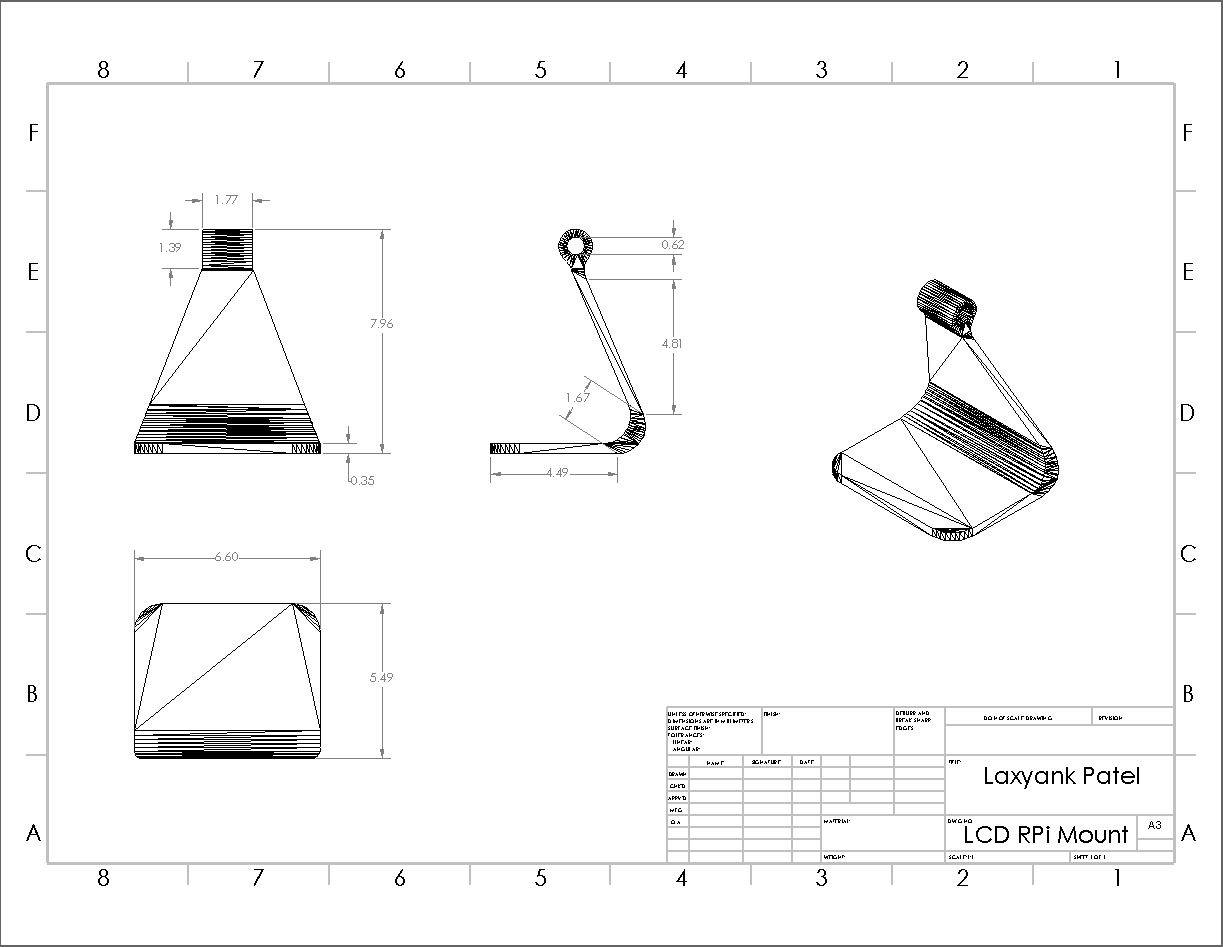
**Appendix**

**Appendix A: CAD Drawings**

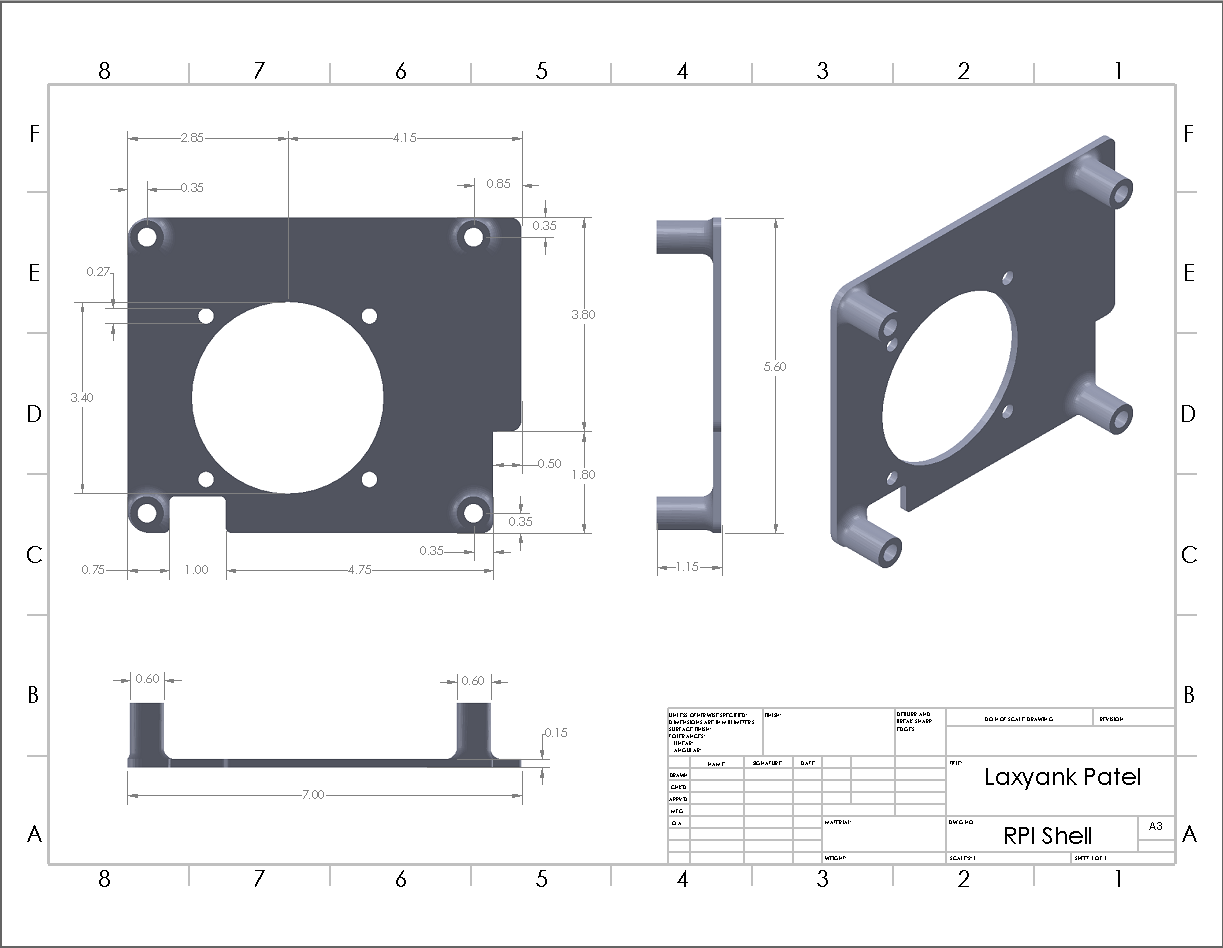
**Figure 4: 22 mm joint**

****

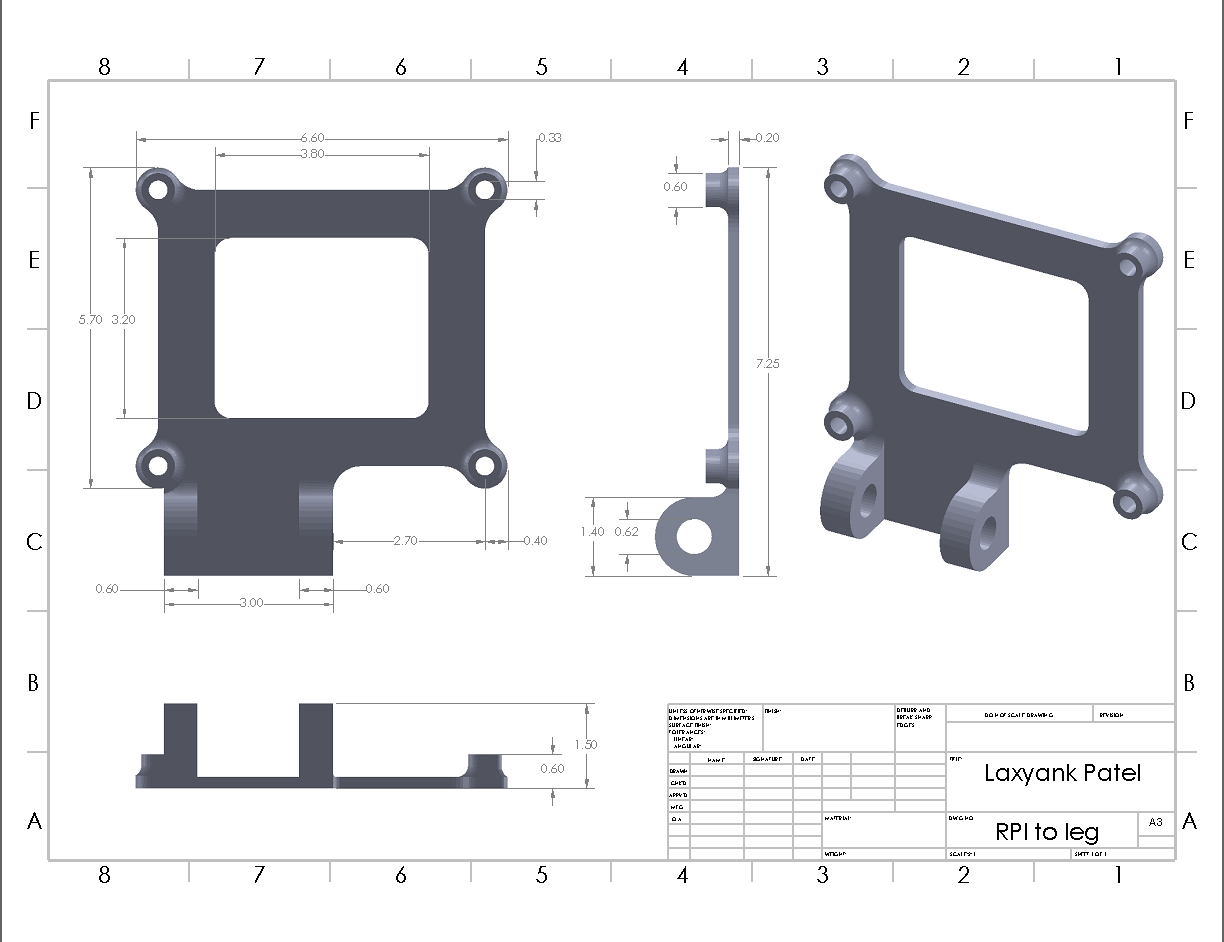
**Figure 5: LCD RPI Mount**

****

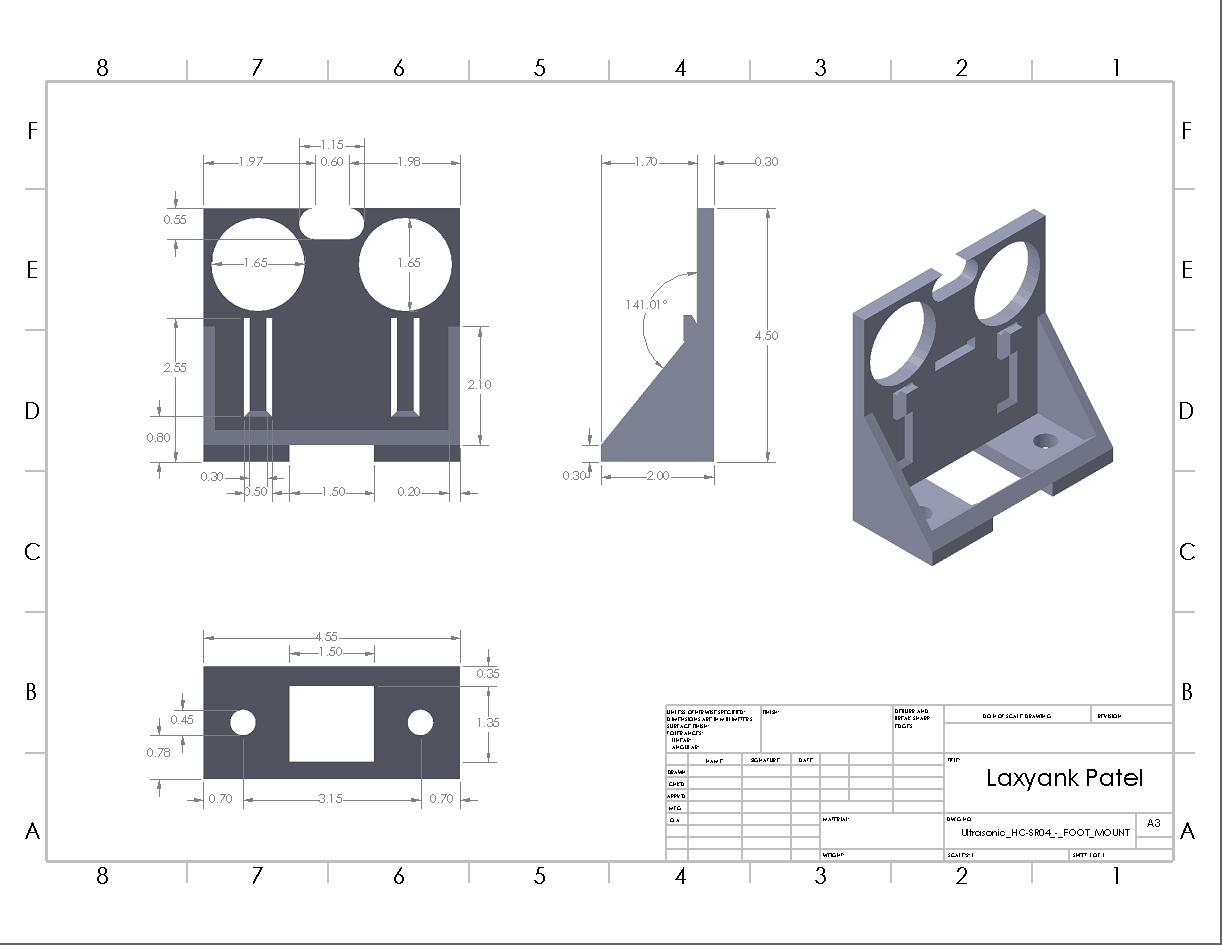
**Figure 6: RPI Shell**

****

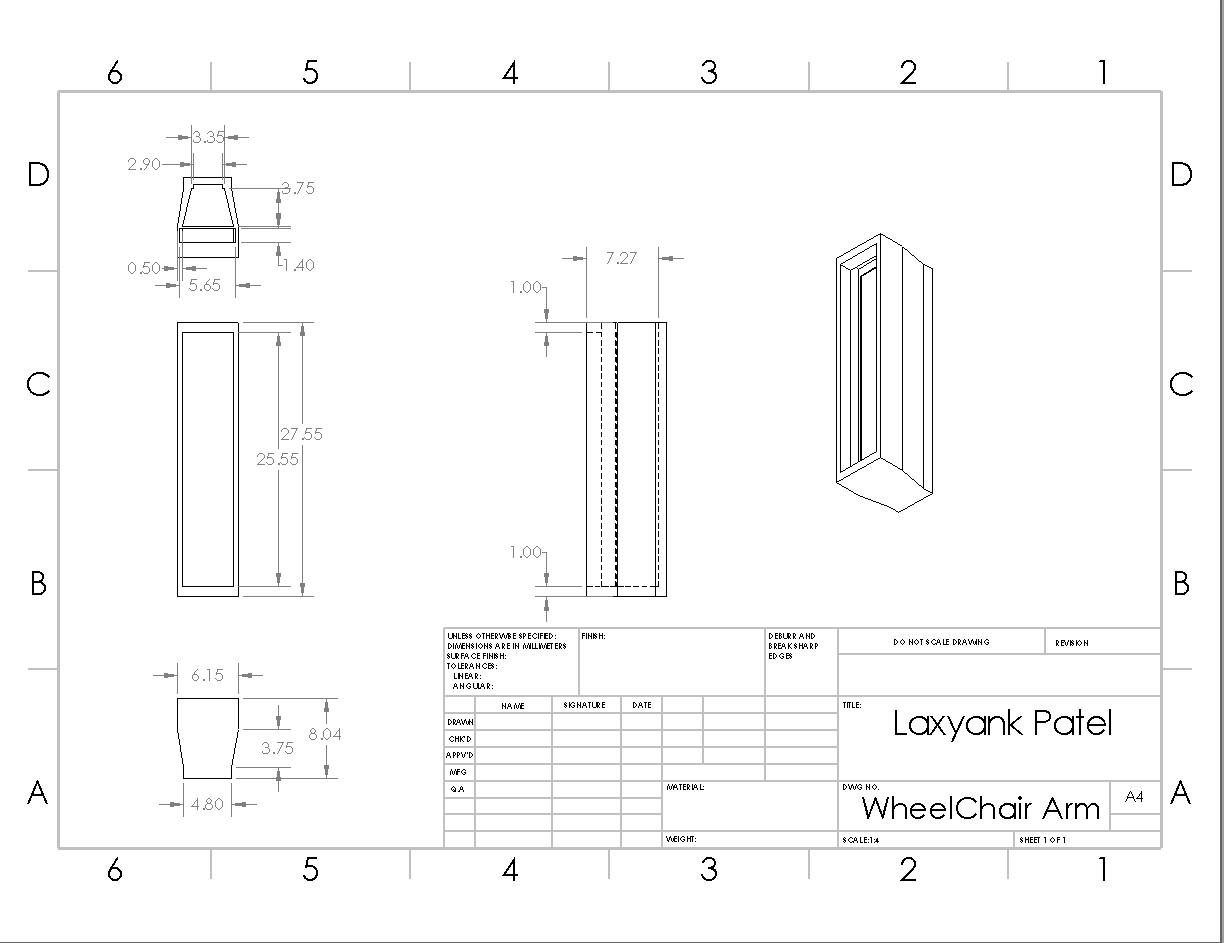
**Figure 7: RPI to Leg**

****

**Figure 8: Ultrasonic Sensor Mount**

****

**Figure 9: Wheelchair Arm**

****

**Appendix B: Code**

**Raspberry Pi Code**

#include <opencv2/opencv.hpp>

#include <raspicam\_cv.h>

#include <iostream>

#include <chrono>

#include <ctime>

#include <wiringPi.h>

#include <cstdio>

#include <cstdlib>

#include <fcntl.h>

#include <string>

#include <sys/ioctl.h>

#include <termios.h>

#include <unistd.h>

#include <stdexcept>

#include <thread>

using namespace std;

using namespace cv;

using namespace raspicam;

Mat frame, Matrix, framePers, frameGray, frameThresh, frameEdge, frameFinal, frameFinalDuplicate;

Mat ROILane;

int LeftLanePos, RightLanePos, frameCenter, laneCenter, Result;

RaspiCam\_Cv Camera;

stringstream ss;

string input = "a";

bool written = false;

vector<int> histrogramLane;

Point2f Source[] = {Point2f(30,150),Point2f(330,150),Point2f(20,210),Point2f(345,210)};

Point2f Destination[] = {Point2f(80,0),Point2f(280,0),Point2f(80,240),Point2f(280,240)};

void Setup ( int argc,char \*\*argv, RaspiCam\_Cv &Camera )

{

Camera.set ( CAP\_PROP\_FRAME\_WIDTH, ( "-w",argc,argv,360 ) );

Camera.set ( CAP\_PROP\_FRAME\_HEIGHT, ( "-h",argc,argv,240 ) );

Camera.set ( CAP\_PROP\_BRIGHTNESS, ( "-br",argc,argv,50 ) );

Camera.set ( CAP\_PROP\_CONTRAST ,( "-co",argc,argv,50 ) );

Camera.set ( CAP\_PROP\_SATURATION, ( "-sa",argc,argv,50 ) );

Camera.set ( CAP\_PROP\_GAIN, ( "-g",argc,argv ,50 ) );

Camera.set ( CAP\_PROP\_FPS, ( "-fps",argc,argv,100));

}

void commandThread(){

while (1){

cin >> input;

written = false;

}

}

void Capture()

{

Camera.grab();

Camera.retrieve( frame);

cvtColor(frame, frame, COLOR\_BGR2RGB);

}

void Perspective()

{

line(frame,Source[0], Source[1], Scalar(0,0,255), 2);

line(frame,Source[1], Source[3], Scalar(0,0,255), 2);

line(frame,Source[3], Source[2], Scalar(0,0,255), 2);

line(frame,Source[2], Source[0], Scalar(0,0,255), 2);

Matrix = getPerspectiveTransform(Source, Destination);

warpPerspective(frame, framePers, Matrix, Size(360,240));

}

void Threshold(){

cvtColor(framePers, frameGray, COLOR\_RGB2GRAY);

inRange(frameGray, 100, 255, frameThresh);

Canny(frameGray,frameEdge, 100, 500, 3, false);

add(frameThresh, frameEdge, frameFinal);

cvtColor(frameFinal, frameFinal, COLOR\_GRAY2RGB);

cvtColor(frameFinal, frameFinalDuplicate, COLOR\_RGB2BGR); //used in histrogram function only

}

void Histrogram()

{

histrogramLane.resize(360);

histrogramLane.clear();

for(int i=0; i<360; i++) //frame.size().width = 400

{

ROILane = frameFinalDuplicate(Rect(i,140,1,100));

divide(255, ROILane, ROILane);

histrogramLane.push\_back((int)(sum(ROILane)[0]));

}

}

void LaneFinder()

{

vector<int>:: iterator LeftPtr;

LeftPtr = max\_element(histrogramLane.begin(), histrogramLane.begin() + 100);

LeftLanePos = distance(histrogramLane.begin(), LeftPtr);

vector<int>:: iterator RightPtr;

RightPtr = max\_element(histrogramLane.begin() +275, histrogramLane.end());

RightLanePos = distance(histrogramLane.begin(), RightPtr);

line(frameFinal, Point2f(LeftLanePos, 0), Point2f(LeftLanePos, 240), Scalar(0, 255,0), 2);

line(frameFinal, Point2f(RightLanePos, 0), Point2f(RightLanePos, 240), Scalar(0,255,0), 2);

}

void LaneCenter()

{

laneCenter = (RightLanePos-LeftLanePos)/2 +LeftLanePos;

frameCenter = 181.5;

line(frameFinal, Point2f(laneCenter,0), Point2f(laneCenter,240), Scalar(0,255,0), 3);

line(frameFinal, Point2f(frameCenter,0), Point2f(frameCenter,240), Scalar(255,0,0), 3);

Result = laneCenter-frameCenter;

}

int fd;

void setupComPort(const string comPort){

fd = open(comPort.c\_str(), O\_RDWR | O\_NOCTTY | O\_NDELAY);

if (fd < 0)

{

throw std::runtime\_error("Failed to open port!");

}

struct termios config;

tcgetattr(fd, &config);

// Set baudrate

cfsetispeed(&config, B9600);

cfsetospeed(&config, B9600);

// 9600 8N1

config.c\_cflag &= ~PARENB;

config.c\_cflag &= ~CSTOPB;

config.c\_cflag &= ~CSIZE;

config.c\_cflag |= CS8;

// Disable hardware based flow control

config.c\_cflag &= ~CRTSCTS;

// Enable receiver

config.c\_cflag |= CREAD | CLOCAL;

// Disable software based flow control

config.c\_iflag &= ~(IXON | IXOFF | IXANY);

// Termois Non Cannoincal Mode

config.c\_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

// Minimum number of characters for non cannoincal read

config.c\_cc[VMIN] = 1;

// Timeout in deciseconds for read

config.c\_cc[VTIME] = 0;

// Save config

if (tcsetattr(fd, TCSANOW, &config) < 0)

{

close(fd);

throw std::runtime\_error("Failed to configure port!");

}

// Flush RX Buffer

if (tcflush(fd, TCIFLUSH) < 0)

{

close(fd);

throw std::runtime\_error("Failed to flush buffer!");

}

}

void write(std::string message){

int length = message.size();

if (length > 100)

{

throw std::invalid\_argument("Message may not be longer than 100 bytes!");

}

char msg[101];

strcpy(msg, message.c\_str());

int bytesWritten = write(fd, msg, length);

}

void stop(){

cout <<"stop";

}

void autonomous( ){

while(input == "a")

{

auto start = std::chrono::system\_clock::now();

Capture();

Perspective();

Threshold();

Histrogram();

LaneFinder();

LaneCenter();

if (Result >= -3 && Result <= 3){

cout<<"Forward"<<endl;

write("0");

}

else if (Result > 3){

cout<<"Right"<<endl;

write("1");

}

else if (Result <-3){

cout<<"left"<<endl;

write("2");

}

else{

cout<<"stop"<<endl;

write("2");

}

ss.str(" ");

ss.clear();

ss<<"Result = "<<Result;

putText(frame, ss.str(), Point2f(1,50), 0,1, Scalar(0,0,255), 2);

namedWindow("orignal", WINDOW\_KEEPRATIO);

moveWindow("orignal", 0, 100);

resizeWindow("orignal", 640, 480);

imshow("orignal", frame);

namedWindow("Perspective", WINDOW\_KEEPRATIO);

moveWindow("Perspective", 640, 100);

resizeWindow("Perspective", 640, 480);

imshow("Perspective", framePers);

namedWindow("Final", WINDOW\_KEEPRATIO);

moveWindow("Final", 1280, 100);

resizeWindow("Final", 640, 480

);

imshow("Final", frameFinal);

waitKey(1);

auto end = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed\_seconds = end-start;

float t = elapsed\_seconds.count();

int FPS = 1/t;

cout<<"FPS = "<<FPS<<endl;

sleep(.001);

}

}

void mainThread(){

while(1){

if(input == "s" && written == false){

cout<<"I received a stop command"<<endl;

write("s");

written = true;

}

if(input == "b" && written == false){

cout<<"I received a Begin command"<<endl;

write("b");

written = true;

}

if(input == "a" && written == false){

cout<<"I received an auto command"<<endl;

autonomous();

written = true;

}

if(input == "i" && written == false){

cout<<"I received an on command"<<endl;

write("i");

written = true;

}

if(input == "o" && written == false){

cout<<"I received an off command"<<endl;

write("o");

written = true;

}

sleep(1);

}

}

int main(int argc,char \*\*argv)

{

wiringPiSetup();

//Setup ComPort

string comportString = "/dev/ttyACM0";

setupComPort(comportString);

Setup(argc, argv, Camera);

cout<<"Connecting to camera"<<endl;

if (!Camera.open())

{

cout<<"Failed to Connect"<<endl;

}

cout<<"Camera Id = "<<Camera.getId()<<endl;

thread command(commandThread);

thread mainProgram(mainThread);

command.join();

mainProgram.join();

return 0;

}

**Arduino Code**

#define joyA 9

#define joyB 10

#define ECHO\_PIN A4

#define TRIG\_PIN A5

#define TRIG\_PIN2 A7

#define ECHO\_PIN2 A8

const int ObstacleDetection = 35;

const int powerPin = 4;

const int powerFeedback = 5;

int SW\_state = 0;

char mode = 'm';

char data = '\*';

int line\_A = 130;

int line\_B = 130;

int speed = 190;

int neutral = 140;

float kp = 2.5;

float ki = .1;

int wallDist = 50;

int counter;

float distance[]={0,0,0};

bool runOnce = false;

void setup() {

pinMode(ECHO\_PIN, INPUT);

pinMode(TRIG\_PIN, OUTPUT);

pinMode(ECHO\_PIN2, INPUT);

pinMode(TRIG\_PIN2, OUTPUT);

pinMode(A0, INPUT);

pinMode(A1, INPUT);

pinMode(powerPin,OUTPUT);

pinMode(powerFeedback,INPUT);

pinMode(11, OUTPUT);

pinMode(12, OUTPUT);

pinMode(13, OUTPUT);

pinMode(2, INPUT\_PULLUP);

Serial.begin(9600);

// Serial3 is for bluetooth in pins TX3 and RX3

Serial3.begin(9600);

}

//Ultrasonic distance measurement Sub function

void Distance\_test() {

digitalWrite(TRIG\_PIN, LOW);

delayMicroseconds(2);

digitalWrite(TRIG\_PIN, HIGH);

delayMicroseconds(20);

digitalWrite(TRIG\_PIN, LOW);

distance[0] = pulseIn(ECHO\_PIN, HIGH)/74/2;

digitalWrite(TRIG\_PIN2, LOW);

delayMicroseconds(2);

digitalWrite(TRIG\_PIN2, HIGH);

delayMicroseconds(20);

digitalWrite(TRIG\_PIN2, LOW);

distance[1] = pulseIn(ECHO\_PIN2, HIGH)/74/2;

}

//starts up chair on run

void startupProcedure() {

stop();

delay(300);

digitalWrite (powerPin,HIGH);

delay(100);

digitalWrite (powerPin,LOW);

delay(2500);

}

//F(neutral,speed),B(neutral,30),R(50,neutral),L(220,neutral),S(neutral,neutral)

void move(int lineA, int lineB){

analogWrite(joyA,lineA);

analogWrite(joyB,lineB);

}

void stop(){

move(140,140);

// analogWrite(joyA,neutral);

// analogWrite(joyB,neutral);

}

//PI controller calculations with limiters

float Pcalc(){

Distance\_test();

if(distance[0] > 200){

return 130;

}

if (distance[0] > 110){

return 185;

}

delayMicroseconds(20);

if(distance[0] > 70 && distance[0] < 110){

return 130;

}

float err = distance[0] - wallDist;

float alpha = kp \* err + (ki \* err) + 130;

if (alpha > 220){

alpha = 220;

}

if (alpha < 30){

alpha = 30;

}

//Serial.println(distance[0]);

return alpha;

}

//toggles between auto and manual

void toggleRead(){

if (digitalRead(2) == LOW){

if (mode == 'u'){

mode = 'm';

}

else{

mode = 'u';

}

delay(300);

}

}

void lineTracking(){

while (mode == 'a'){

reading();

//delay(1);

readingSerial();

//Serial.println(data);

// Serial.println(mode);

if(data=='0'){ //forward

move(neutral, 190);

}

else if(data=='1'){

move(113,neutral);

}

else if(data=='2'){

move(177, neutral);

}

else if (data == 's'){

stop();

}

Distance\_test();

objectDetection('a');

}

}

void objectDetection(char type){

while(distance[1] < ObstacleDetection && distance[1] > 1 && mode == type){

stop();

reading();

Distance\_test();

delay(10);

toggleRead();

}

}

void ultra(){

move(neutral, speed);

while(counter<3600 && mode == 'u'){

Distance\_test();

line\_A = Pcalc();

move(line\_A,speed);

delay(10);

// Serial.print("line\_A: " );

//Serial.println(line\_A);

counter++;

//This reads toggle switching for auto or manual

toggleRead();

reading();

// Obstacle detection and stop until object moves

objectDetection('u');

}

stop();

counter = 0;

}

void readingSerial(){

while(Serial.available()){

data = Serial.read();

switch(data){

case 'i': startupProcedure(); data = '\*'; break;

case 'm': mode = 'm'; break;

case 'a': mode = 'a'; break;

default: break;

}

}

//Serial.println(data);

}

void reading(){

if(Serial3.available()){

mode = Serial3.read();

switch(mode){

case 'p': startupProcedure(); mode = '\*'; break;

case 'f': move(neutral,speed); break;

case 'b': move(neutral,30); break;

case 'l': move(180,neutral); break;

case 'r': move(80,neutral); break;

case 's': stop();

mode = 's';

break;

case 'u': ultra(); break;

case 'a': mode = 'a'; break;

case 'y': speed += 10;

if (speed > 250) {

speed = 250;

}

break;

case 'z': speed -= 10;

if (speed < 180) {

speed = 180;

}

break;

default: break;

}

}

//Serial.println(mode);

//mode = 'm';

}

void manualMode() {

int xPosition = analogRead(A0);

line\_A = xPosition/8.0+80;

int yPosition = analogRead(A1);

line\_B = yPosition/4.6+40;

SW\_state = digitalRead(2);

delay(1);

move(line\_A,line\_B);

// Serial.print("lineA=");

// Serial.println(line\_A);

}

void loop() {

if(runOnce == false){

startupProcedure();

runOnce = true;

}

reading();

//delay(10);

readingSerial();

//delay(10);

//Serial.println(speed);

toggleRead();

if(mode == 'm'){

manualMode();

Distance\_test();

Serial.println(distance[0]);

Serial.println(distance[1]);

}

else if(mode == 'u'){

ultra();

while(mode == 'u'){

reading();

toggleRead();

delay(300);

}

}

else if(mode == 'a'){

lineTracking();

}

}

**Appendix C: Team member contribution table**

**Table 8: Team technical contribution**

|  |  |
| --- | --- |
| **Name** | **Contributions** |
| Lucky Patel | 3D design, print and installation; wire organization and soldering |
| Austin Clark | Documentation; Wall Following Programming; Bluetooth Implementation; Movement Programming |
| Duc Nguyen | Line Following Programming |