ENGINEERING SENIOR DESIGN PROJECT

MOWBOTIC

THE NEXT GENERATION OF LAWN MOWING

BY XIANG CHEN, MIGUEL PONTE, CHANDRA SWAROOP REDDY, PONELARAJAN ELAVARASAN

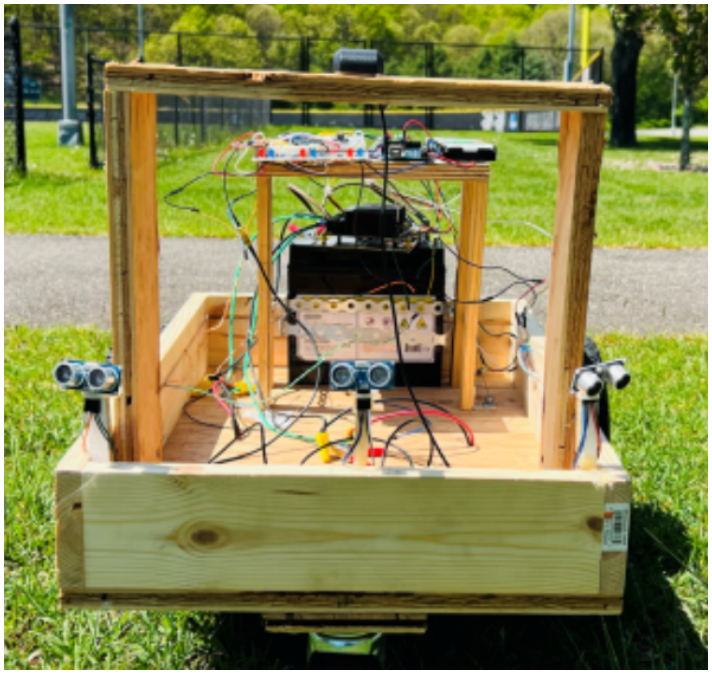


Table of Contents

Abstract

Problem:

Millions of people living in the US mow their lawn or use mowing services to do it for them. Where they may face the following problems:

- Lack of funds
    - Unable to pay for basic needs
    - Unable to pay for the service
        - Lawn grows long
            - Fines from the town
- Lack of availability
    - Unable to make time for the work

Solution:

To create an autonomous lawn mower that will greatly aide solve these problems. This autonomous lawnmower will perform these tasks and allow the user to mow their lawn with minimal effort on their part.

Benefits:

By creating this autonomous lawnmower, we will be able to see the following:

- Free your time
- Less money spent
- Decrease in labor
- Increase efficiency in lawn maintenance

Introduction

The field of engineering has had an amazing period of growth where our everyday lives are being changed for the better. This has been for the large part due to both growth in robotics and autonomous systems. Robotics uses a combination of sensors, motors, and digital systems to allow for implementation of autonomy. Implementation of some form of autonomy in our everyday lives and have become an ever-growing part of society. This implementation can range in complexity from purchasing a product off a store page to cleaning your home. Besides this, they can also differ in use cases as in where the product is designed to operate. Is it designed to be used in a factory

setting, or at your home. Many of these autonomous devices are being used in factory settings to reduce costs and improve efficiency. However, during the past few years, we have seen these robots designed for factory use be redesigned for the consumer. These products allow for many mundane tasks we do daily to be automated. The purpose of creating an autonomous lawn mower is to create a better system for robotic lawn mowers. In today's market, there are only lawn mowers that turn at random to cover the area, our design will minimize time spent mowing, energy used, and money spent in frequent lawnmowing services. Autonomous lawn mowers with the ability to be self-guiding would open a new market and revolutionize how a lawn is mowed. Tasks we can free up this time and allow the robot to perform the task more efficiently. The project of creating this autonomous lawnmower will be difficult but it will allow us to learn and improve our skill of engineering gained at Farmingdale State College.

Motivation

There are about 70 million households in the US living in the suburban area. They need to mow their lawns biweekly, weekly or monthly. The average lawn mowing time is 1-2 hours or the cost is $20-$50 if you are hiring someone to

do it for you. By automating this task, it allows individuals to better spend their time and allow the automated lawn mower to take over this mundane task. We began our project researching several autonomous lawn mowers that have already been created. Taking important concepts from each of these projects allows us to recreate an even better mower. Some of the main improvements would be in cost, and efficiency of the mower. Being able to see previous mistakes where others have tried in similar projects has put us in a great advantage in our own endeavors. The inspiration for this project came from wanting to innovate on the idea of a Roomba, by applying its technology to a different application. It is to create an autonomous lawn mower that meets the needs of modern standards. The design will focus on minimizing the time and money spent on lawn mowing. By automating this task, it allows individuals to better spend their time and allow the automated lawn mower to take over this mundane task. There are already automated lawn mowers that are available commercially, but their prices are usually over $1000. One of our main goals is to create an affordable lawn mower as automation is something that is becoming more commonplace rather than a luxury. The way we plan on bringing down our cost is by innovating on a bought electric lawn mower while keeping the automation as simple as we can. If we are successful, the

autonomous lawn mower can be cost efficient, by allowing the owner to pay a one-time fee, instead of a more frequent service from a lawnmowing compan

Power Rating

The secondary on-board power source consists of two 22Ah 12V Mighty Max batteries connected in a series configuration to power both Jazzy Electric drive motors at 30A each to full Power specifications for a desirable duration of 25mins. The computation for the optimal drive wheel linear speed is calculated by converting rev/m to rad/s followed by the application of the angular to linear Link equation to metamorphose angular speed into linear speed; V=RW (where W=Omega). The ATmega2560 microprocessor specifies; an overall DC current limit for all IO pins put together at 200mAmax. Its on-board voltage regulator is rated for 1Amax and will temporarily shut-off if the operating-tempmax is exceeded. Therefore, all components and accessories driven by the microprocessors should draw a quantifiable 400mAmax.

General Power Calculation

Power Consumption

- Arduino Mega: 2.5 Wh

- Arduino Nano: 0.0627 Wh

- Ultrasonic Sensor x3: 0.075 Wh

- Bluetooth Module: 0.5 Wh

- Motors (x2): 72-960 Wh

- MDDS30 Motor-driver: 7-35 Wh

- GPS Module: 0.5 Wh

Power Supply

- Mighty Max Battery: 1056 Wh

Minimum Run Time: 1 hour

Components List

- Jazzy Electric Drive Motors/Wheelchair Motor 12-24 VDC
- Arduino Mega 2560

- Cytron MDDS30 Motor Driver
- BlueFruit LE Bluetooth
- SparkFun Neo_M9N
- Primo Drive Wheels
- Mighty Max Battery
- Battery Charger
- Power Supply
- Ultrasonic Sensors

Jazzy electric drive Motors / Wheelchair motor 12-24VDC

Price: $ 55.00
Quantity: 2
Operating Voltage: 12-24VDC
Operating Current: 3 amps no load, 25-40 amps with load
Torque: 50ft-lbs
RPM: 120 rpm at 24VDC, 60 rpm at 12VDC

For the actual driving of our autonomous lawnmower, we are using a pair of used brushed dc Jazzy electric Wheelchair motors rated at 12-24 VDC. Two brake assembly wires were already removed to further simplify the motors for our use.

The motors will run at 3 amps under a no-load condition, and up to 25-40 amps with load. For precise operation of the motors, we are wiring them up to the sabertooth dual 32A motor driver which will dial down the amperage as previously discussed. The RPMs of the motors are listed at roughly 120 rpm at 24Vdc / 60 rpm at 12vdc, with around 50 ft-lbs of torque at a max load. The two motors, combined with the motor driver, will enable both the driving and steering of the mower, as well as braking. With these specifications in mind, the maximum total wattage per motor would be (P = V * C), and estimating that the motors will run at about 20-25 amps each bring it to around 600 Watts continuous. One of our next steps is going to be testing how the motor driver operates with the motors, and the power and amperage consumption of them.

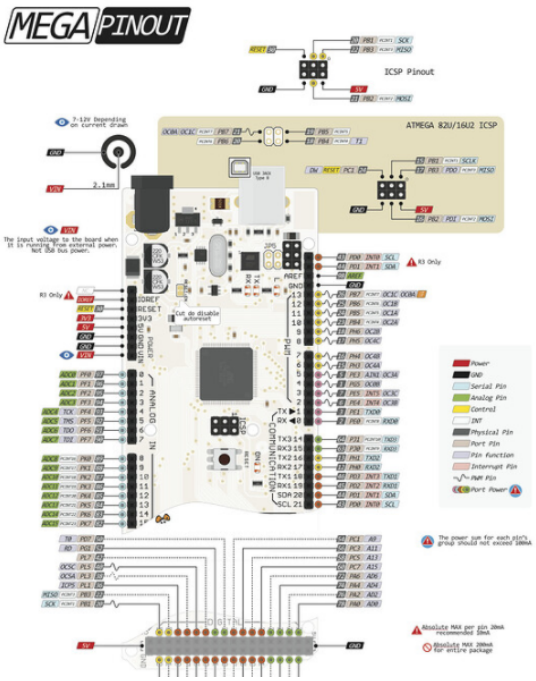
Arduino Mega 2560

- Microcontroller: Atmega2560
- Operating Voltage: 5 - 24V
- Operating Current: 200 mA to 500 mA
- Max Current Output: 800 mA to 1A
- Clock Frequency: 16MHZ
- Digital I/O: 54 (14 which provide PWM output)
- Analog I/O: 16
- Memory Size: 256 KB
- Price: $12
- Quantity: 1
- Protocol: Serial, SPI, I2C, I2S, UART

The microcontroller we will be using as the brains of the autonomous lawn mower is the Arduino Mega 2560. The reason we picked this microcontroller was due to the memory size, clock frequency and amount of digital I/O pins.

The clock frequency of the Atmega2560 is 16 MH.

This allows for quick logical decisions which will help the



lawn mower be fully autonomous. With the on board memory being at 256 KB allows us to have the flexibility to write code that might not be as optimized as it could be due to experience which would then require more space.

Cytron MDDS30 Motor Driver

Price: $71.50

Quantity: 1

Input and Output Voltage: 7-35v

Input and Output Current: 0-30A

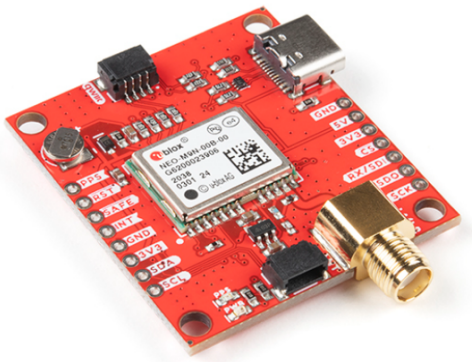
SmartDriveDuo-30 is one of the latest smart series motor drivers designed to drive medium power brushed DC motor with current capacity up to 80A peak (few seconds) and 30A continuously, each channel. This driver is designed especially for controlling differential drive mobile robot using RC

controller. Nevertheless, MDDS30 also can be controlled using analog joystick or microcontroller (PWM, Serial). MOSFETs are switched at 18 KHz to ensure quiet operation and no annoying whining sound. Besides, it also equipped with a microcontroller unit to provide smart features such as multiple input mode and thermal protection.

SparkFun Neo_M9N GPS Module

- Receivers Type: 92 Channels
- Time-To-First-Fix: cold start 26 S, hot start 1 S
- Sensitivity for Tracking: -160 dBm
- Maximum Navigation Update rate

  5-8 Hz
- Horizontal Position Accuracy: 1.5 M
- Price $74.95



The SparkFun NEO-M9N GPS Breakout is a high-quality GPS board with equally impressive configuration options including SMA. The NEO-M9N module is a 92-channel u-blox M9 engine GNSS receiver, meaning it can receive signals from the GPS, GLONASS, Galileo, and BeiDou constellations with ~1.5-meter accuracy. This breakout supports concurrent reception of four GNSS. This maximizes position accuracy in challenging conditions increasing, precision and decreases lock time; and thanks to the onboard rechargeable battery, you'll have backup power enabling the GPS to get a hot lock within seconds! Additionally, this u-blox receiver supports I^2C (u-blox calls this Display Data Channel) which makes it perfect for the Qwiic compatibility so we don't have to use up our precious UART ports. Utilizing our handy Qwiic system, no soldering is required to connect it to the rest of your system. However, we still have broken out 0.1"-spaced pins in case you prefer to use a breadboard.

Mighty M Voltage: 12 Volt

Amperage: 22 AH

Chemistry: GEL

Terminal: Internal Thread

Battery Dimensions:

7.13 in x 3.01 in x 6.57 in

Weight: 13.01 Lbsax Battery

Delivering power when you need it, the Mighty Max ML22-12GEL 12 Volt 22 AH (Maintenance Free) battery. Requires no addition of water during the life of the battery. The Mighty Max ML22-12 GEL is a TRUE DEEP CYCLE battery that can be mounted in any position, requires no maintenance. When a Gel Cell battery is charged, no hazardous fumes escape the battery case, gases are processed within the battery itself. Mighty Max GEL batteries are utilized in a wide variety of applications including; Consumer Electronics, Electric Vehicles, Engine Starters, Golf Carts, Hunting, Lawn and Garden Tools, Medical Mobility, Motorcycles, Power sports, Portable Tools, Solar, Toys and Hobby, Access Control Devices, Emergency Lighting, Security and more

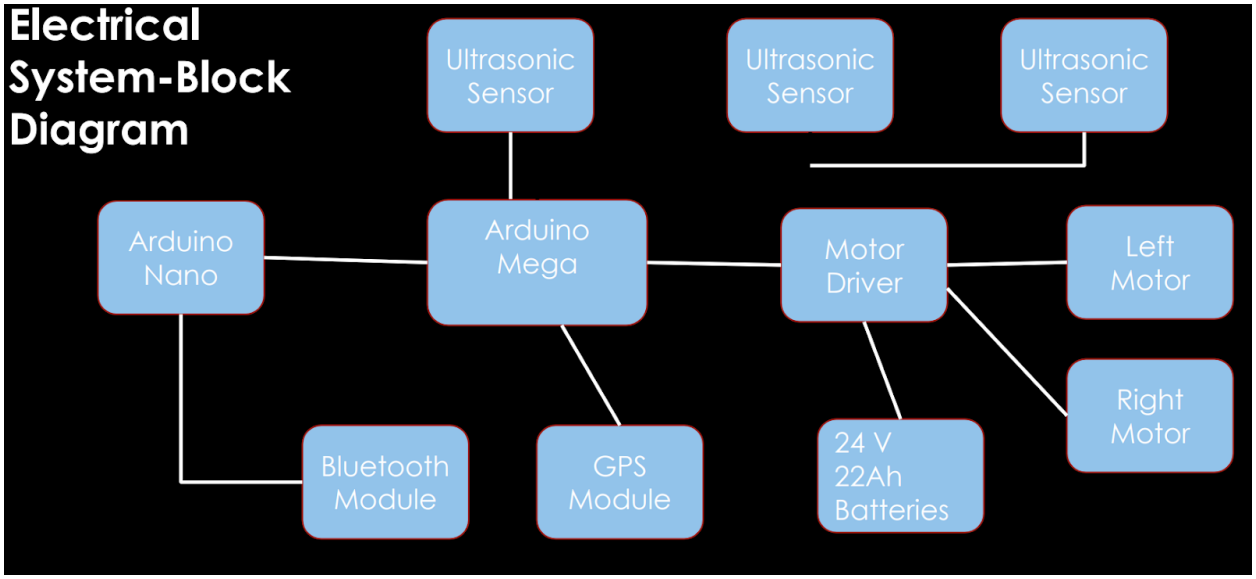
Considerations for Product Development

Being we are not the first group of people trying to design an autonomous mower likes the one we get here. Some of design in the market are quite successful. From the market point of view, we need to constraint our budget to maintain a certain some degree of competition. Originally, we intend to use high precision GPS

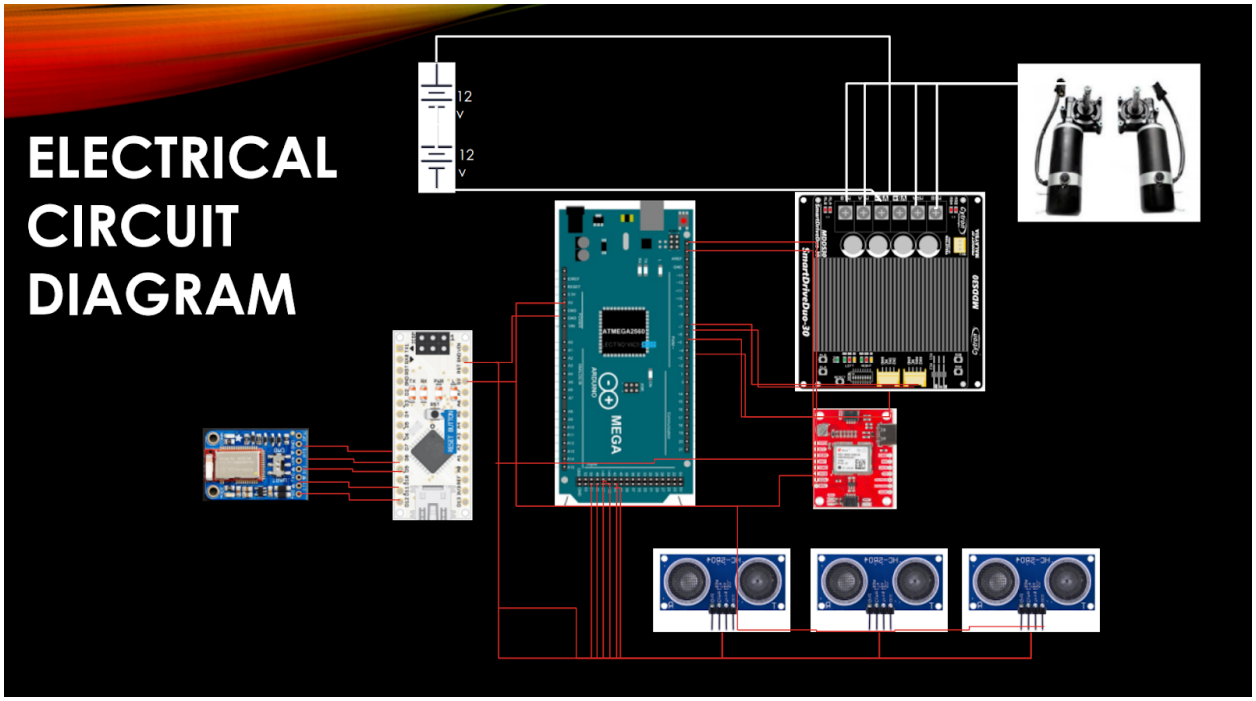
module namely, SparkFun GPS-RTK Board-NEO-M8P-2, but the cost is 4 times more than the SparkFun NEO_M9N. These considerations are as follows:

- Compatibility of systems in conjunction with one another
- Heat dissipation of the electrical system and processing components
- Wire gages needed to sufficiently allow for the proper current
- Voltage requirements for each subsystem and component
- Power requirement for the system
- Chassis modifications of existing lawn mower that are required
- Size of each component
- Weather the project may be subjected to when in operation

Electrical System-Block Diagram



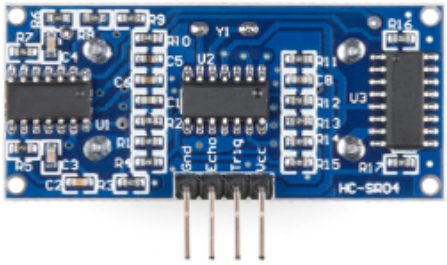
Electrical Circuit Diagram



Microcontroller

In order to process the information from the peripheral devices such as the GPS and the remote control and make decisions for the control section of the system, we need to have a central processing unit mounted on our system. For this system, we will be using the Arduino Mega 2560 microcontroller as well as the Microchip Starter Board that comes with the unit. Both of these devices are made by Microchip. We decided on this setup as opposed to other microcontrollers based on the fact that the compiler and board setup is easier to build on and understand than other choices. The board can run on 3.3 Vdc, and we will drop the voltage off of the battery to support it.

Ultrasonic Sensors:



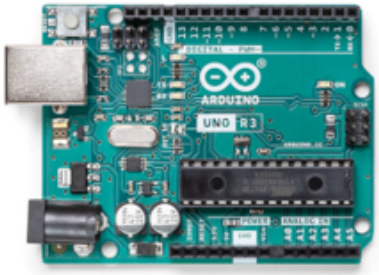
Operating Voltage - 5V

Operating Current -15mA

Distance Range - 2cm to 4m

Measure Angle - 15^{o}

Other Components



Arduino

Battery And
Battery Holder

Jumper Cables

LED's

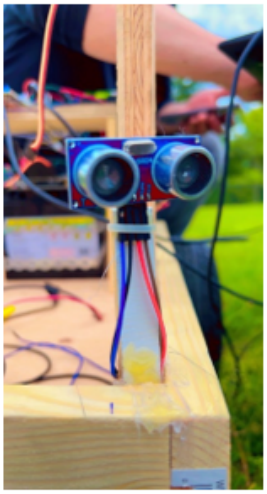
INITIAL IDEA FOR POSITION OF IR SENSORS –

- 2 IR Sensors will be used
- One will be placed in front center, and another will be placed on right side center

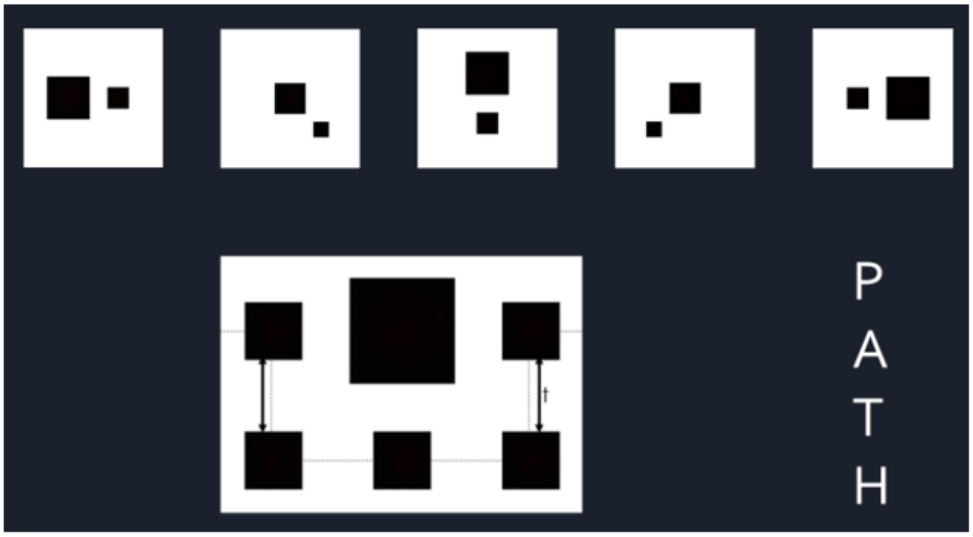
FAILED!! – Due to the length of face of the mower

POSITION OF ULTRASONIC SENSORS -

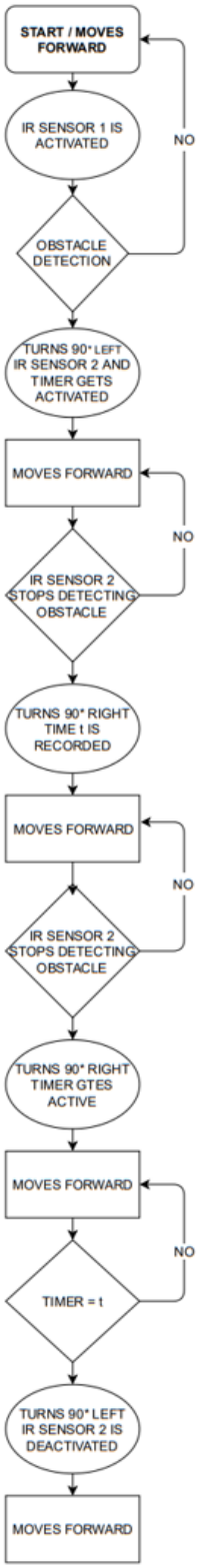
- 3 Ultrasonic Sensors are used
- One is placed at the left corner, one is placed in the center and one is placed in the right corner
- 3 are used because of the angle detected by ultrasonic sensors is 15°



INITIAL IDEA FOR OBJECT DETECTION USING IR SENSORS –



- Whenever the motor starts moving the IR Sensor 1 gets activated
- If IR Sensor 1 detects an obstacle, it stops and makes a 90° LEFT turn and activates IR Sensor 2 and a timer
- If IR Sensor 2 stops detecting an obstacle, it stops and records the timer 't' and then make a 90° RIGHT turn
- If IR Sensor 2 stops detecting an obstacle, it stops and make a 90° RIGHT turn and activates a timer
- If timer = t, then it stops and makes a 90° LEFT turn


START / MOVES FORWARD
IR SENSOR 1 IS ACTIVATED
NO
OBSTACLE DETECTION
TURNS 90° LEFT IR SENSOR 2 AND TIMER GETS ACTIVATED
MOVES FORWARD
NO
IR SENSOR 2 STOPS DETECTING OBSTACLE
TURNS 90° RIGHT TIME t IS RECORDED
MOVES FORWARD
NO
IR SENSOR 2 STOPS DETECTING OBSTACLE
TURNS 90° RIGHT TIMER GTES ACTIVE
MOVES FORWARD
NO
TIMER = t
TURNS 90° LEFT IR SENSOR 2 IS DEACTIVATED
MOVES FORWARD

INITIAL IDEA – FAILED!!

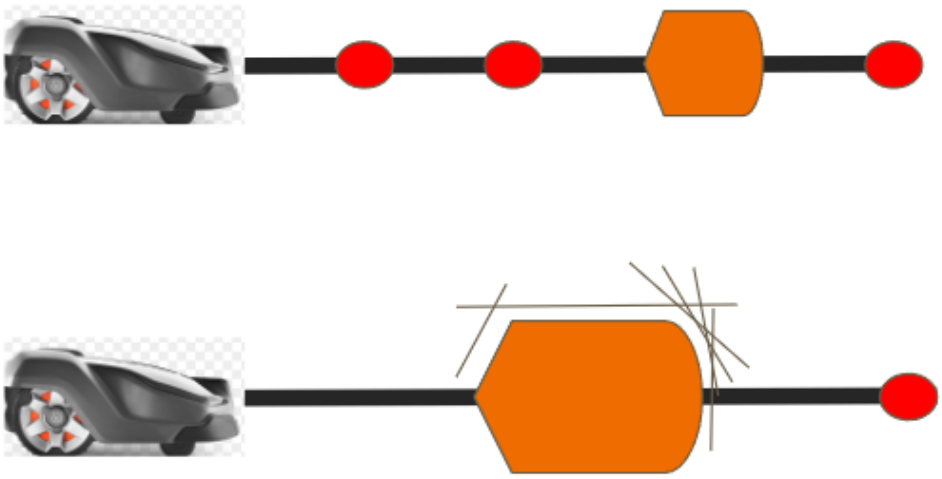
Limited for object which are in the form of

1. Square

2. Circle (<3m radius)

Doesn't work for objects which have ARBITRARY shapes

REVISED IDEA FOR OBJECT DETECTION –



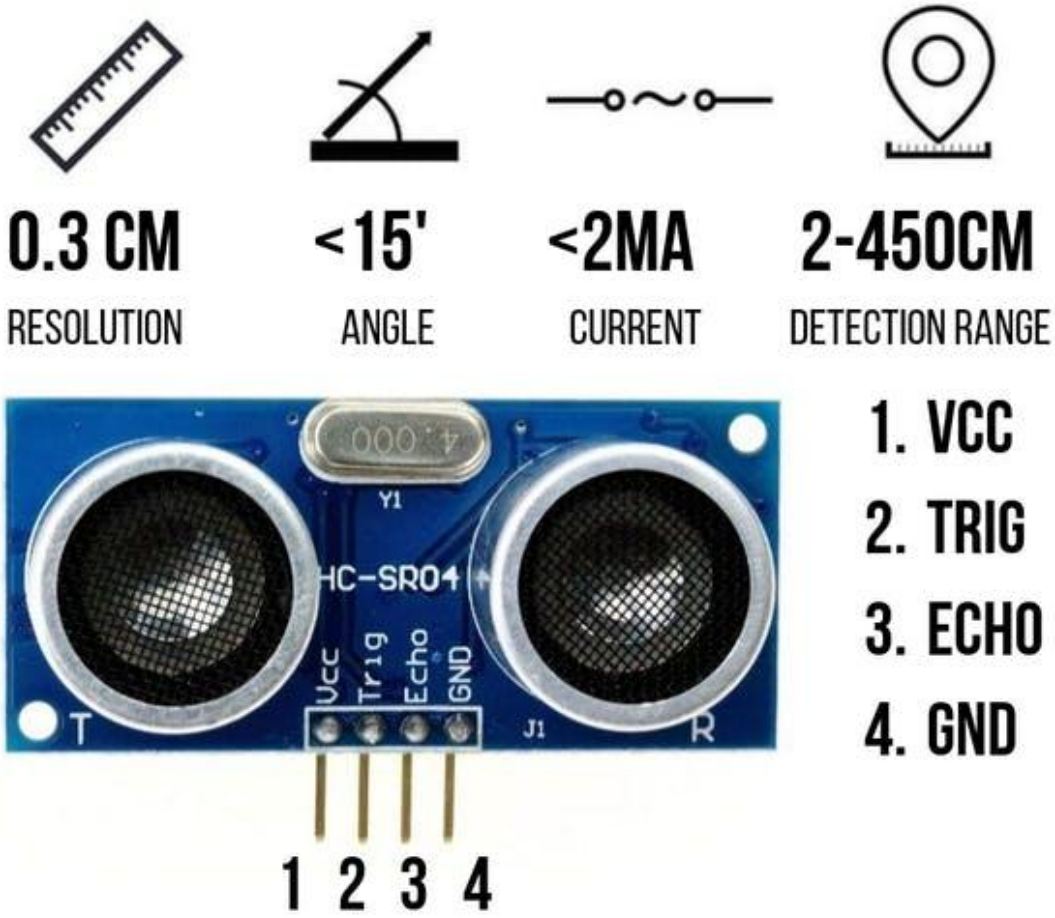
- Whenever the motor starts moving, all the Ultrasonic sensors get activated
- If any Ultrasonic sensor detects an obstacle, the mower comes to rest and turns left until all the ultrasonic sensors stops detecting obstacle

- The mower calculates its slope using GPS data, so the mover uses slope and turns around the object as shown in above figure.

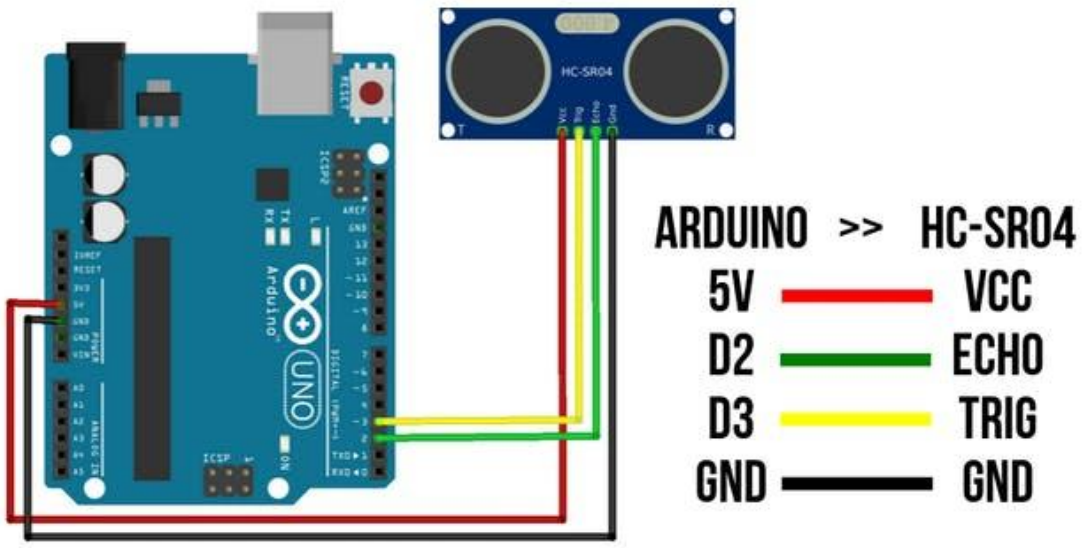
ULTRASONIC SENSORS :

Ultrasonic Sensor HC-SR04 is a sensor that can measure distance. It emits an ultrasound at 40 000 Hz (40kHz) which travels through the air and if there is an object or obstacle on its path It will bounce back to the module. Considering the travel time and the speed of the sound you can calculate the distance.

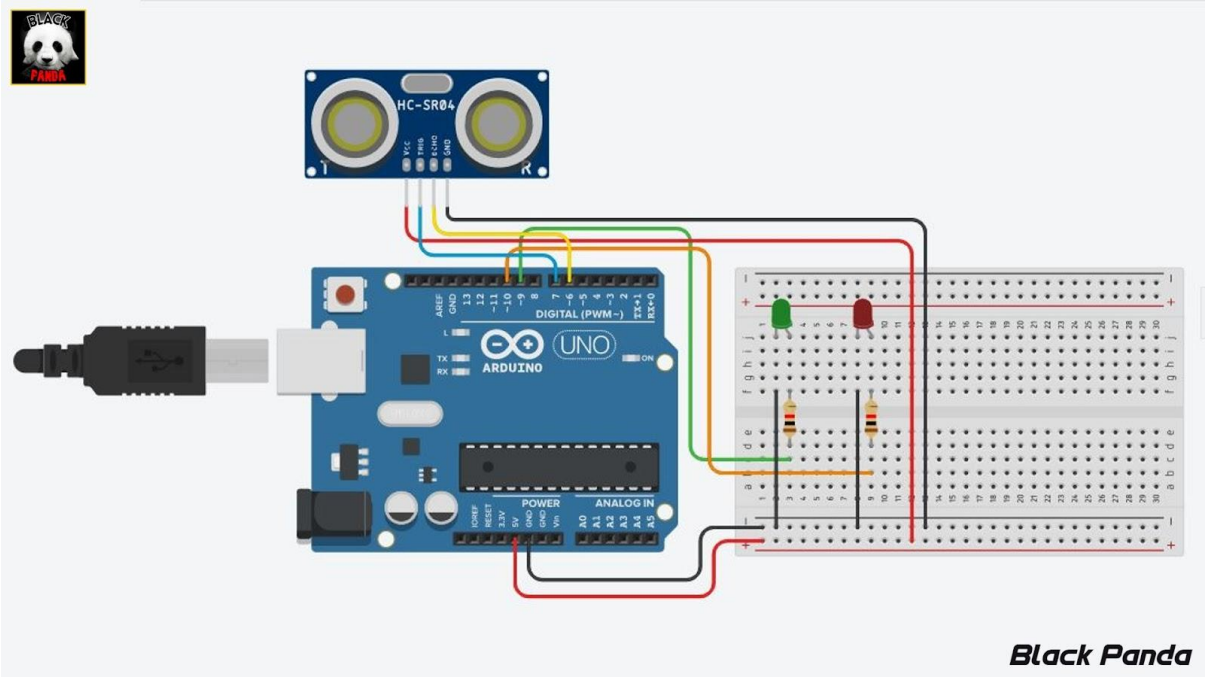
The configuration pin of HC-SR04 is VCC (1), TRIG (2), ECHO (3), and GND (4). The supply voltage of VCC is +5V and you can attach TRIG and ECHO pin to any Digital I/O in your Arduino Board.



CONNECTIONS:

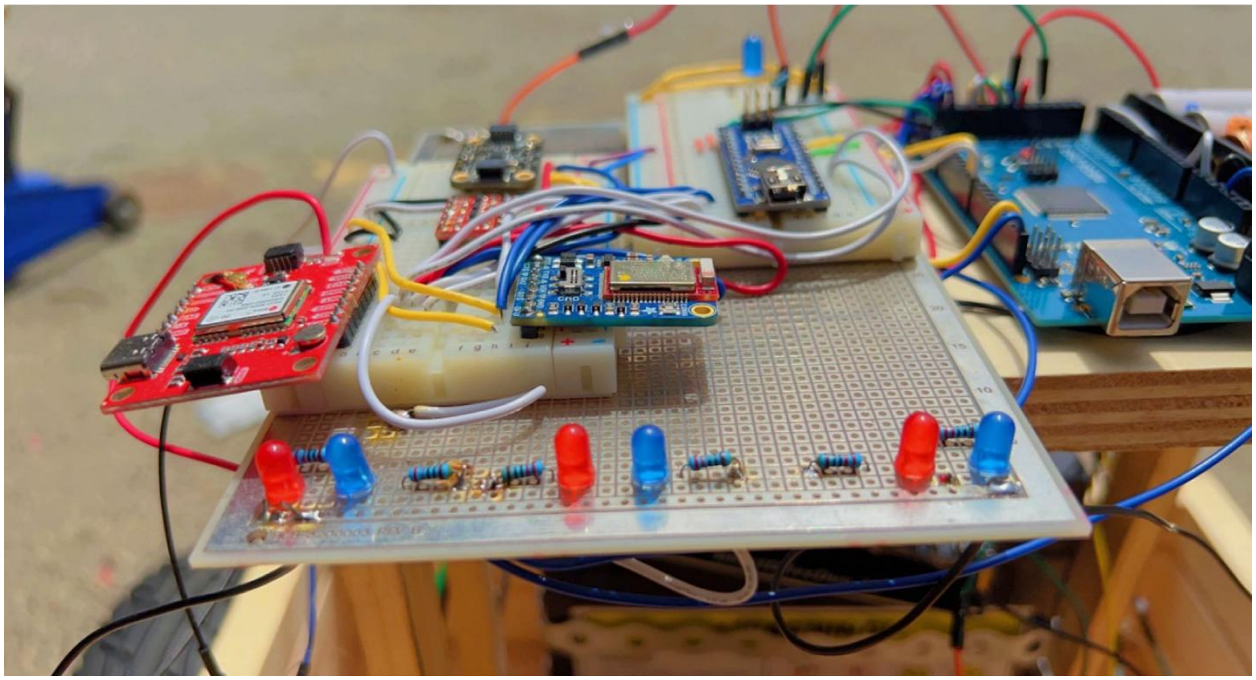


CONNECTION OF ULTRASONIC SENSOR HC-SR04 & ARDUINO BOARD

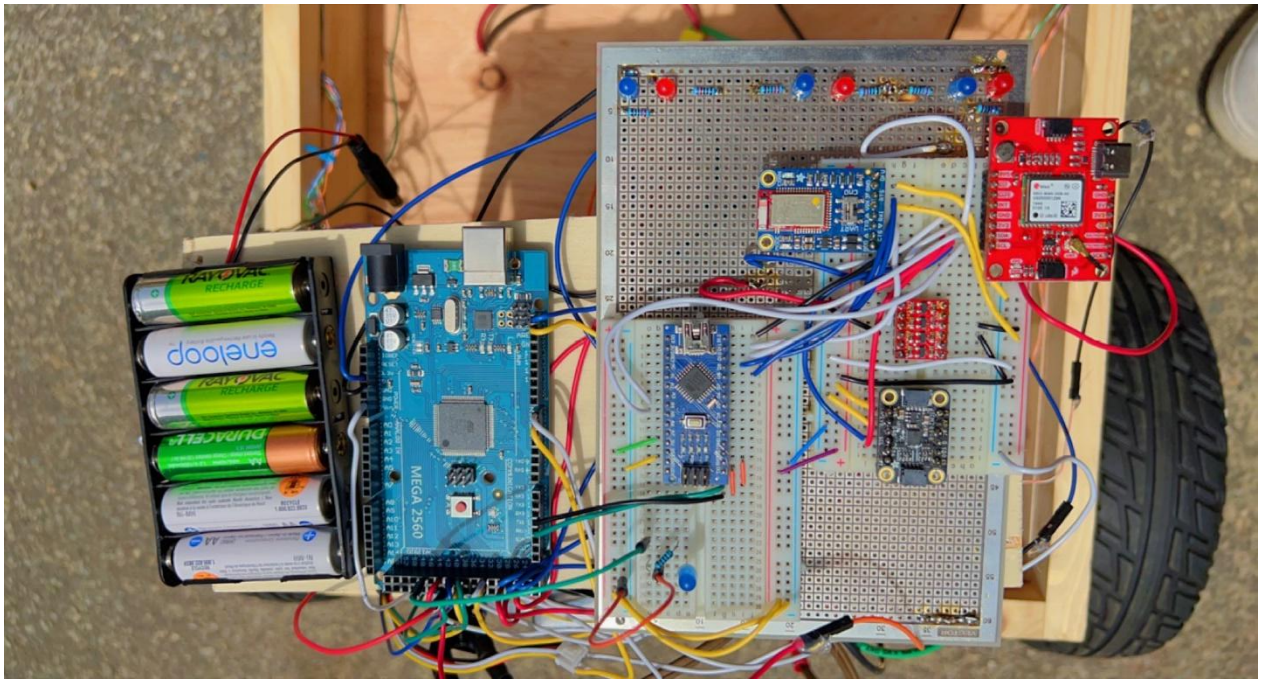


CONNECTION OF ULRTASONIC SENSORS, LED LIGHTS AND ARDUINO BOARD

IMPLEMENTATION

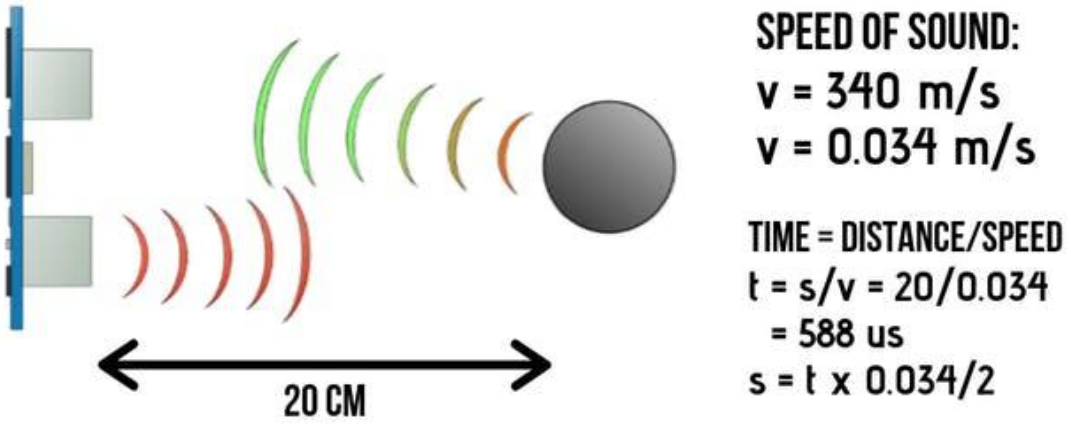


The LED lights are connected separately to each ultrasonic sensor. So, when the middle ultrasonic sensor detects an object only the middle red led glows and when there is no object, the blue led glows.



CALCULATION:

For example, if the object is 20 cm away from the sensor, and the speed of the sound is 340 m/s or 0.034 cm/µs the sound wave will need to travel about 588 microseconds. But what you will get from the Echo pin will be double that number because the sound wave needs to travel forward and bounce backward. So in order to get the distance in cm we need to multiply the received travel time value from the echo pin by 0.034 and divide it by 2.



ADVANTAGE OF THIS METHOD:

No need to configure the inside area like object shown in figures apart from the borders in the beginning.

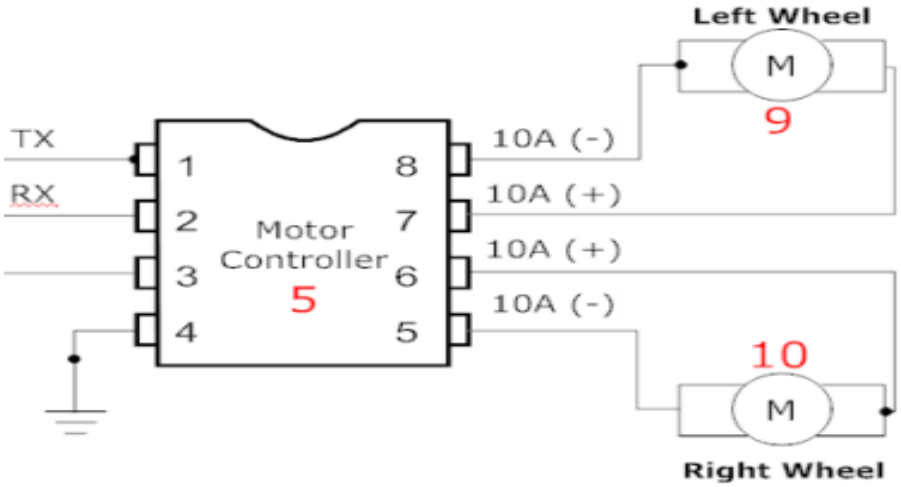
ARDUINO CODE FOR OBJECT DETECTION:

```
#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04

// defines variables
long duration; // variable for the duration of sound wave travel
int distance; // variable for the distance measurement

void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
  Serial.begin(9600); // // Serial Communication is starting with
9600 of baudrate speed
  Serial.println("Ultrasonic Sensor HC-SR04 Test"); // print some
text in Serial Monitor
  Serial.println("with Arduino UNO R3");
}
void loop() {
  // Clears the trigPin condition
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in
microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2; // Speed of sound wave divided by
2 (go and back)
  // Displays the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
}
```

Motor Controller and Wheels

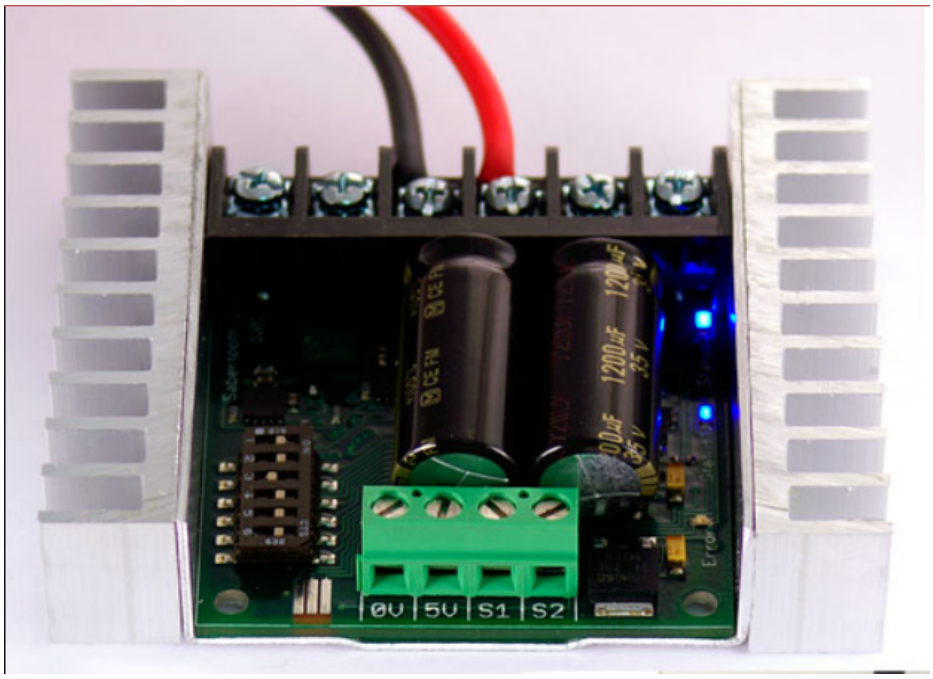


Motor controllers are devices that direct the activity of an electric engine. In fake lift applications, engine regulators for the most part allude to those gadgets utilized related to switchboards or variable recurrence drives to control the activity of the prime mover.

Motor controllers frequently incorporate a manual or programmed implies for beginning and halting the engine, choosing forward or switch pivot, accelerating or dialing back, and controlling other functional boundaries. Also, engine regulators can give assurance to the counterfeit lift framework by controlling or restricting the force, and safeguarding against over-burdens and blames. Many

engine regulators contain extra abilities, for example, information assortment and information logging as well as application-explicit control rationale.

Sabertooth 2x25:-



Product Sabertooth 2X25:-

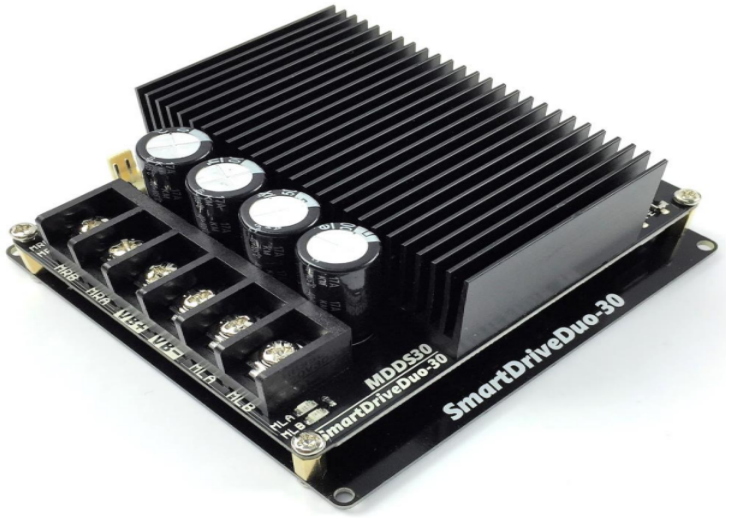
* The Sabertooth 2X25 is one of the most versatile, efficient and easy to use dual motor drivers on the market. It is suitable for high powered robots - up to 100lbs in combat or 300lbs for general purpose robotics.

* Out of the box, the Sabertooth can supply two DC brushed motors with up to 25A each. Peak currents of 50A per channel are achievable for a few seconds.

*At first we used Sabertooth 2x25 as our Motor Controller. It cost around 100$, But due to Over Heating issues, STATUS LED is not turning on, if it's turn on also no Motor Output from the Mobotic. So unfortunately this Motorcontroller is not useful for thi.s Lawnmover.

* So, we moved on with MDDS30 - Cytron SmartDriveDuo-30.

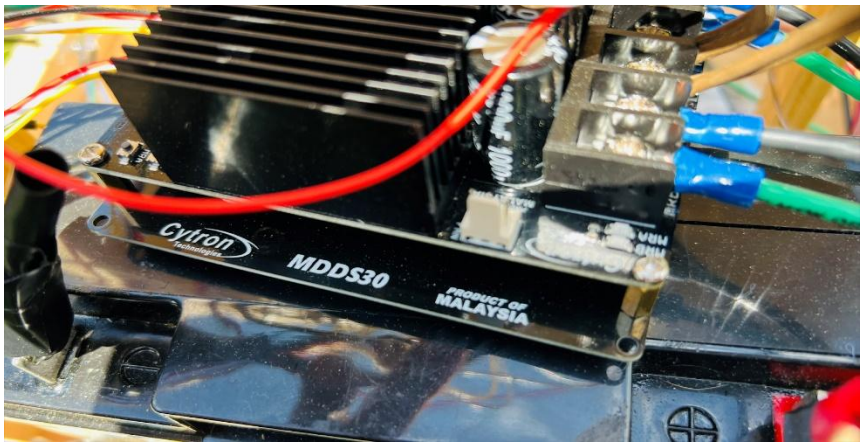
MDDS30 - Cytron SmartDriveDuo-30



We know DC brushed motor is widely used for many different applications; from conveyor, AGV (Automated Guide Vehicle), mobile robot and combat robot. One of the reasons is the ease to use it, just connect the two terminals of motor to power respectively, it will start rotating, so simple. Yet, to control the speed and direction from a controller, a motor driver is needed.

→ SmartDriveDuo-30 is one of the latest smart series motor drivers designed to drive medium power brushed DC motor with current capacity up to 80A peak (few seconds) and 30A continuously, each channel. This driver is designed especially for controlling differential drive mobile robot using RC controller. Nevertheless, MDDS30 also can be controlled using analog joystick or microcontroller (PWM, Serial). MOSFETs are switched at 18 KHz to ensure quiet operation and no annoying whining sound. Besides, it also equipped with a

microcontroller unit to provide smart features such as multiple input modes and current limit and thermal protection.



MDDS30 - Cytron SmartDriveDuo-30 is one of the Latest smart series motor drivers with the Current capacity up to 80A peak and it is designed especially for controlling differential drive mobile robot using the RC controller. And the MDDS30 - Cytron SmartDriveDuo-30 can be controlled using an analog joystick or microcontroller. And it has some Smart features such as multiple input modes,current limit and thermal protection.

Features of Cyron Motor Driver:-

- Bi-directional control for dual brushed DC motor.
- Support motor voltage from 7V to 35V.
- Maximum current up to 80A peak (1 second), 30A continuously, each channel.
- On board MOSFETs are switched at 18 KHz for quiet operation.
- Battery low voltage indicator.
- Battery over voltage indicator.
- Thermal protection.

- Current limit protection.

- Multiple input modes: RC, Analog/PWM, Serial Simplified and Serial Packetized.

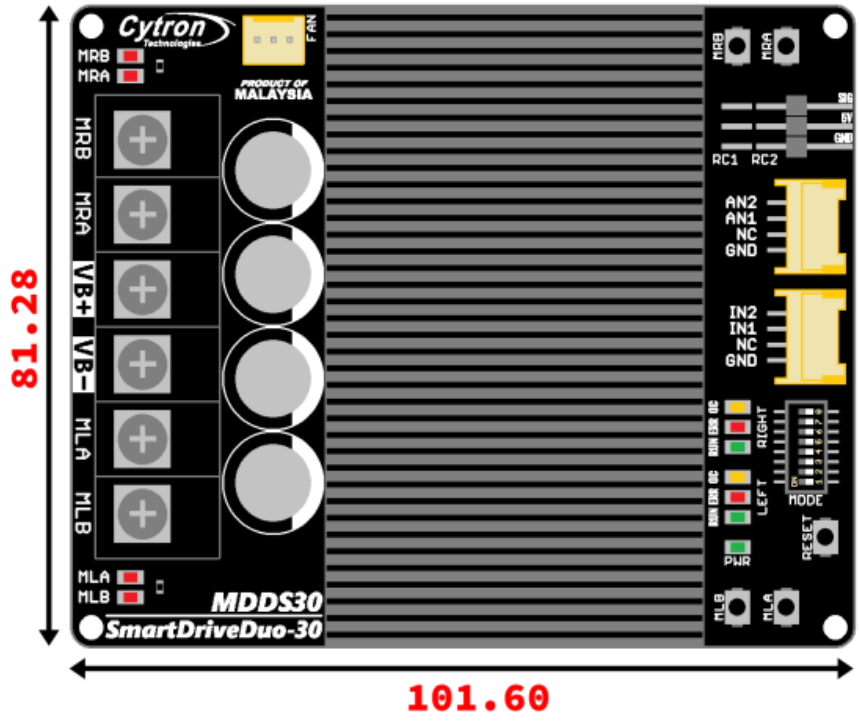
- On board push buttons for fast test and manual operation.

- No polarity protection for v motor.

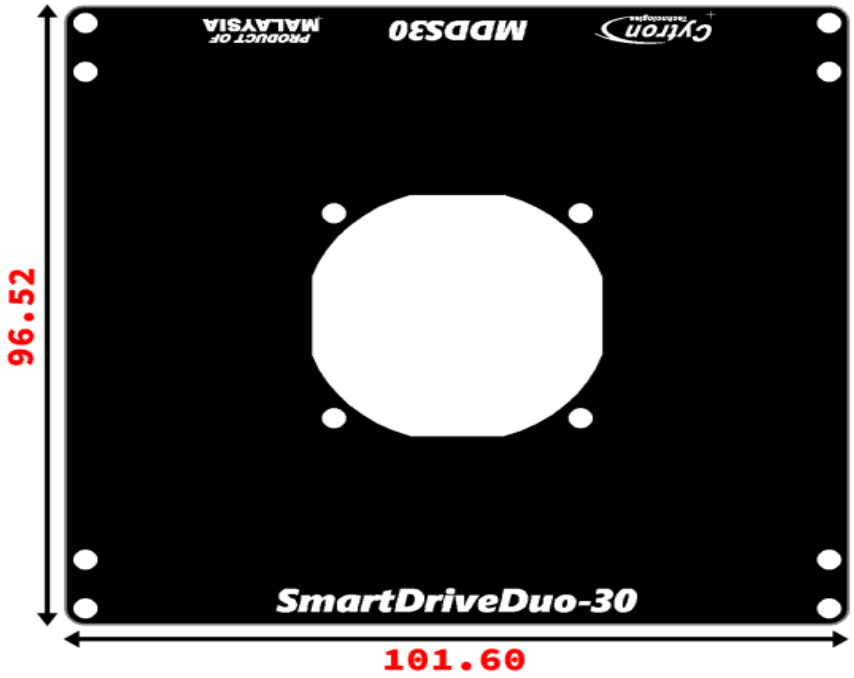
Product specifications:-

Dimension (unit in mm).

- MDDS30 main board.



- MDDS30 bottom cover.



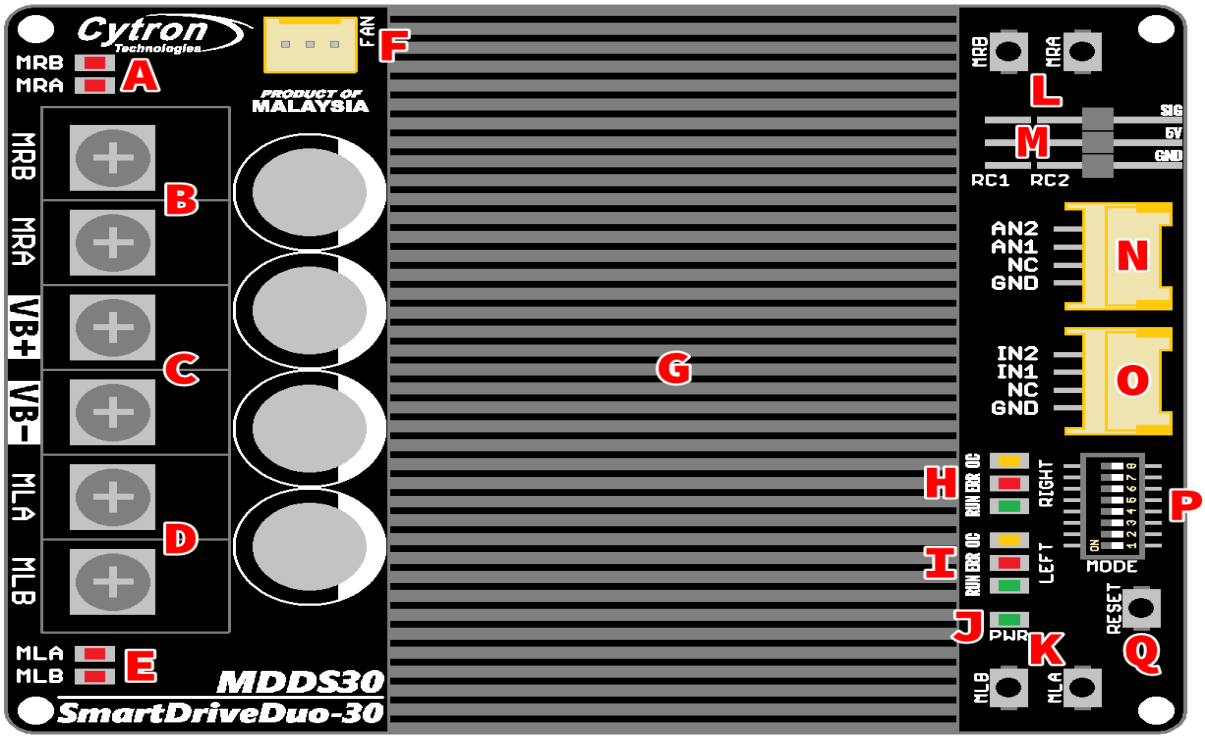
The Cyron Motor Driver MDDS30 is 81.28mm height, 101.60mm breadth and 96.52mm length. It is one of the slimmest motor driver. It's very easy and compact.

Absolute Maximum Rating of SmartDriveDuo-30:-

No	PARAMETERS	Min	Typ	Max	Unit
1	Input Voltage (Motor Supply Voltage)	7	-	35	V
2	IMAX (Max Continuous Motor Current)	-	-	30	A
3	PEAK (Peak Motor Current)	-	-	80	A

4	VIOH (Logic Input – HIGH Level)	1.3	-	5	V
5	VIOL (Logic Input – LOW Level)	0	-	0.7	V

CYTRON MDDS30 BOARD LAYOUT:-



Above image is the CYTRON MDDS30 BOARD layout, I had included the parts name detailed. And one is the name of the mentioned letters in the image.

	MOTOR RIGHT LED INDICATOR

A	
B	**MOTOR RIGHT TERMINAL BLOCK**
C	**POWER SUPPLY TERMINAL BLOCK**
D	**MOTOR LEFT TERMINAL BLOCK**
E	**MOTOR LEFT LED INDICATOR**
F	**COOLING FAN CONNECTOR**
G	**HEAT SINK**
H	**RIGHT CHANNEL LED INDICATOR**
I	**LEFT CHANNEL LED INDICATOR**
J	**POWER LED INDICATOR**
K	**MOTOR LEFT TEST BUTTON**

L	**MOTOR RIGHT TEST BUTTON**
M	**RC INPUT PIN**
N	**ANALOG/PWM INPUT PIN (GROVE)**
O	**DIGITAL/SERIAL INPUT PIN (GROVE)**
P	**MODE SELECTION DIP SWITCH**
Q	**RESET BUTTON**

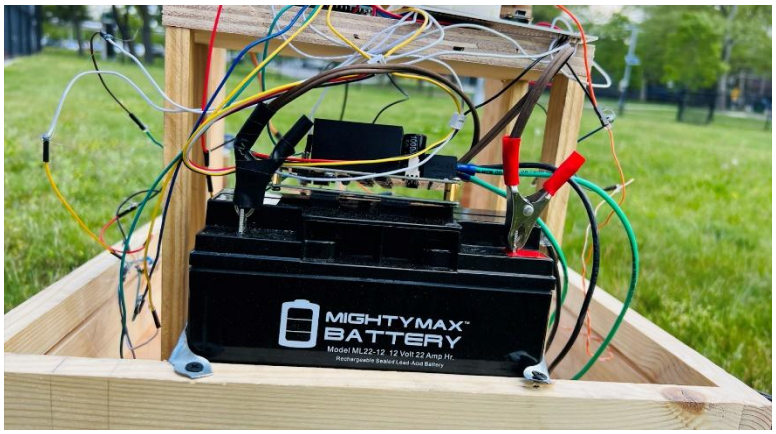
Mounting CYTRON MDDS30 in the Lawn Mover:-

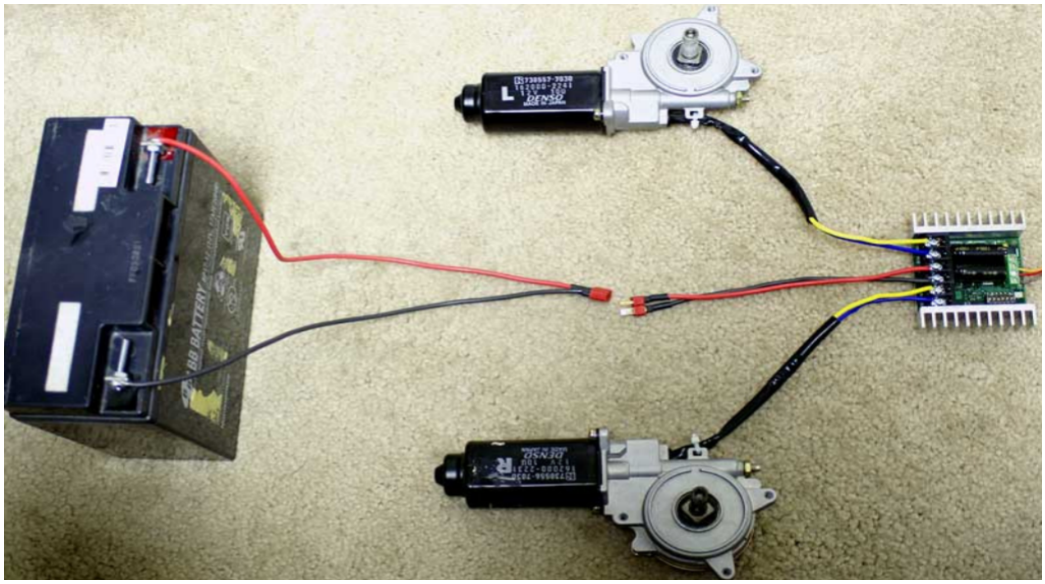
- With four mounting holes.
- These can be used to attach it to your robot.
- The centers of the mounting holes form a 1.75” x 2.25” rectangle. The holes are .125 inches in diameter.
- The proper size screw is a 4-40 round head machine or wood screw. Four 5/8” long machine screws and nuts are included.

- Necessary to mount the Cytron on standoffs to allow air to circulate.
- Better to attach the bottom heat spreader of the Cytron directly to the frame, without standoffs.

By using the CYTRON MDDS30 Motor driver we successfully connected this to battery terminal and the audrino board. It's perfectly working compared to sabretooth motor driver.

Battery Terminals B+ and B:-





This above image has included 12V Battery, Cytron MDDS30 Motor Driver, wheelchair motor and wires.

➔ We Connected CYTRON MDDS30 Motor driver to terminals B- and B+ of the battery.

➔ It is very Easy to unplug the battery for charging.

➔ By using a battery connector to connect/disconnect power to Cytron.

Speed Test in Wheels:

The below data where we got from the Digital Photo Sensor Tachometer device.

12 Volts Battery							
Both Motors				**Separate Motor**			
Forward Motion		**Reverse Motion**		**Left Wheel**		**Right Wheel**	
Left	52.9	**Forward**	51.3	**Forward**	51.5	**Forward**	54.8
Right	54	**Reverse**	54.5	**Reverse**	51.5	**Reverse**	54.6

24 Volts Battery							
Both Motors				**Separate Motor**			
Forward Motion		**Reverse Motion**		**Left Wheel**		**Right Wheel**	
Left	110.6	**Forward**	109.4	**Forward**	114	**Forward**	114.2
Right	112.2	**Reverse**	111.1	**Reverse**	113	**Reverse**	114

Digital Photo Sensor Tachometer is a device which measures accurate

RPM without contacting the spinning object. Able to measure 6 to 99,999 RPM

with ± 0.05% accuracy, the tachometer stores the last, minimum and maximum readings.

By using this we got some data were,

In 12Volt battery :

→ Using both the motor in Forward Motion for Left value 52.9, for Right 54 where as in Reverse Motion in Forward 51.3 and Reverse 54.5.

→ In separate Motor for Left wheel in Forward 51.5 and reverse 51.5 where as in Right Wheel Forward we got 54.8 and Reverse 54.6.

In 24Volt battery:

→ Using both the motor in Forward Motion for Left value 110.6, for Right 112.2 where as in Reverse Motion in Forward 109.4 and Reverse 111.1

→ In separate Motor for Left wheel in Forward 114 and reverse 113 where as in Right Wheel Forward we got 114.2 and Reverse 114.

→ The main reason for this due to different internal resistance of the DC motor cause variation in RPM.

Wheel Chair Motor:-

→ A motorized wheelchair, powerchair, electric wheelchair or electric-powered wheelchair (EPW) is a wheelchair that is propelled by means of an electric motor (usually using differential steering) rather than manual power.

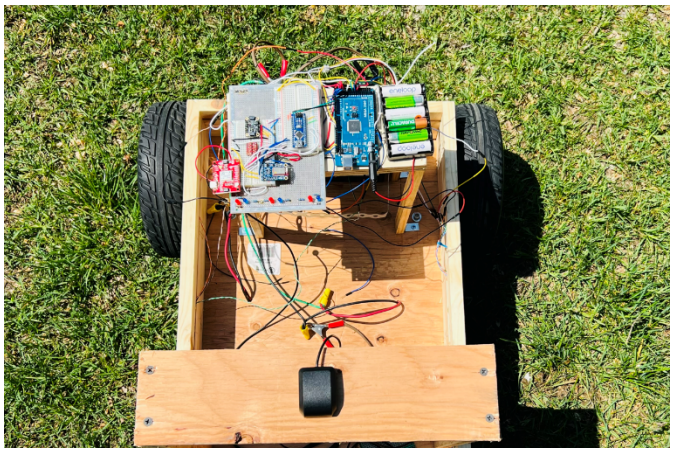
→ This Wheel chair Motor was are useful for those unable to propel a manual wheelchair.

→ Most wheel- chairs utilize two permanent magnet DC motors (PM motors), with two 12 V batteries providing a 24 V supply.

→ A power wheelchair has two motors. Each motor operates a drive wheel on either side of the power wheelchair. The motors draw power from the batteries and move the drive wheels.

→ A wheelchair Motor was tested with Digital Photo Sensor Tachometer to measures accurate RPM without contacting the spinning object.

→ we have attached the Test values or data which we got, while we conducting the Test on wheel.



This is our final image of our Mobotic (Lawnmover).

Programming

Arduino Mega

```
#include <SoftwareSerial.h>

#include <Wire.h>

#include <string.h>
```

```
#include <EEPROM.h>
#include <Arduino.h>
#include <SparkFun_u-blox_GNSS_Arduino_Library.h>
#include "Mowbotic.h"
#include <SPI.h>
//#include <Servo.h>

#define blue1 24
#define red1  25
#define blue2 26
#define red2  27
#define blue3 28
#define red3  29
#define Interupt 46
#define T1 38
#define T2 40
#define T3 42
#define E1 39
#define E2 41
#define E3 43
//int servoPin = 9;
```

```
SFE_UBLOX_GNSS myGNSS;
//Servo servo;

int autoMode=0;
int RCMode= 0;
long lastTime = 0;
long xArr[4];
long yArr[4];
long Pdelta[4];
int countOfItemsX=0;
int countOfItemsY=0;
int CountOfItemsDel=0;
int GPSAdd=66;
int PxRegis=8693;
int PyRegis=8689;
uint8_t *p=NULL;
long Heading;
double M1;
double M4;
//int angle = 0;

Mow Mowing;
```

```
void printFun(long* dummyArr);
//int findOrgin(long* dummyArr,int size);
void pushX(long dummyVar, long* dummyArr);
void pushY(long dummyVar, long* dummyArr);
double findM1(int index);
//int nextIndex(int index);
void transmitData();

void loop()
{
  int orginOfIndex;
  int size=4;
  long x1,y1, heading;
    long theSmallestLat;
    long theLargestLat;
    long theSmallestLong;
    long theLargestLong;



      transmitData();
      x1=myGNSS.getLatitude();
```

```
 y1=myGNSS.getLongitude();
 heading=myGNSS.getHeading();
 if(RCMode==2){
  Mowing.ultra();
 Mowing.task();
while(RCMode==2){
 if(theSmallestLat<x1<theLargestLat && theSmallestLong<y1<theLargestLong){

 }
 if(x1<theSmallestLat ||x1>theLargestLat ||y1<theSmallestLong||y1>theLargestLong){
  digitalWrite(Interupt, HIGH);
  delay(200);
  digitalWrite(Interupt, LOW);
 }
}
if(RCMode!=2){
digitalWrite(red1,LOW);
digitalWrite(blue1,LOW);
digitalWrite(red2, LOW);
digitalWrite(blue2,LOW);
digitalWrite(red3,LOW);
digitalWrite(blue3,LOW);
```

```
  }
 }
else{
if(RCMode==1){
  Mowing.Stop();
}
if(RCMode==5){
  Mowing.straight();
}
if(RCMode==6){
  Mowing.reverse();
}
if(RCMode==7){
  Mowing.leftTurn();
}
if(RCMode==8){
  Mowing.rightTurn();
}
if(RCMode==4){
  Mowing.fullStraight();
}
  //orginOfIndex=findOrgin(xArr, size);
```

```
 if(RCMode==3){
  Serial.print("3 is pressed");Serial.println(RCMode);
   transmitData();
   long x=myGNSS.getLatitude();
   long y=myGNSS.getLongitude();
  delay(500);
   pushX(x,xArr);     //store the Latitude data into the X array
   pushY(y,yArr);     //store the longitude data into the y array
  theSmallestLat=findTheSmallest(xArr);
  theLargestLat=findTheLargest(xArr);
  theSmallestLong=findTheSmallest(yArr);
  theLargestLong=findTheLargest(yArr);
 }


 }
 printFun(xArr);
 printFun(yArr);


  //printFunDou(Pdelta);
//  Serial.print("The orginOfIndex is:");Serial.println(orginOfIndex);
//  Serial.println(nextIndex(orginOfIndex));
 // Serial.println(nextIndex(nextIndex(nextIndex(orginOfIndex))));
```

```
  Serial.print("The smallest lat is:");Serial.println(theSmallestLat);
  Serial.print("The largest lat is:");Serial.println(theLargestLat);
  Serial.print("The smallest long is:");Serial.println(theSmallestLong);
  Serial.print("The largest long is:");Serial.println(theLargestLong);
  Serial.print("new lat value: ");Serial.println(x1);
  Serial.print("new long value: ");Serial.println(y1);
  Serial.print("heading value: ");Serial.println(heading);
}
void requestEvent(){
  Wire.write(uint8_t(p));
}

void receiveEvent(int Byte){
  RCMode=Wire.read();
}

void transmitData(){
  Wire.beginTransmission(66);
  Wire.write(255);
  Wire.endTransmission();
  Wire.requestFrom(66,8);
  while(Wire.available()==0);
```

```
}

long findTheSmallest(long* dummyArr){
  long theSmallest=dummyArr[0];
  for(int i=0;i<4;i++){
    if(theSmallest>dummyArr[i]){
      theSmallest=dummyArr[i];
    }
  }
  return theSmallest;
}

long findTheLargest(long* dummyArr){
  long theLargest=dummyArr[0];
  for(int i=0;i<4;i++){
    if(theLargest<dummyArr[i]){
      theLargest=dummyArr[i];
    }
  }
  return theLargest;
}
void pushX(long dummyVar, long* dummyArr){
```

```
  if(countOfItemsX<4){       //if the x array is not full
  dummyArr[countOfItemsX]=dummyVar; //assign gps latitude data to x array
  if(dummyArr[countOfItemsX]!=0){   //check if the memory location is empty
    digitalWrite(blue1,HIGH);       //set blue LED to indicate is being occupied
    delay(1000);
    digitalWrite(blue1,LOW);
  }
  countOfItemsX++;                  //increase the count of element for X-array
  if(countOfItemsX==4){             //check if the X array is full
    digitalWrite(red2, HIGH);
  }
  }
}

void pushY(long dummyVar, long* dummyArr){
  if(countOfItemsY<4){
  dummyArr[countOfItemsY]=dummyVar;
  if(dummyArr[countOfItemsY]!=0){
    digitalWrite(red1,HIGH);
    delay(1000);
    digitalWrite(red1,LOW);
  }
```

```
  countOfItemsY++;
  if(countOfItemsY==4){
    digitalWrite(red2, HIGH);
  }
  }
}

void printFun(long* dummyArr){
  for(int i=0;i<4;i++){
    Serial.print(i);
    Serial.print("=");
    Serial.print(dummyArr[i]);
    Serial.println();
  }
}

void AutoMode_ISR(){
  Serial.print("Interrupt ");
  Serial.println();
  Mowing.leftTurn();
       //delay(200);
}
```

```
void printFunDou(double* dummyArr){
  for(int i=0;i<4;i++){
    Serial.print(i);
    Serial.print("=");
    Serial.print(dummyArr[i]);
    Serial.println();
  }
}

int findOrgin(long* dummyArr, int size){
  int index=0;
  for(int i=1;i<size;i++){
    if(dummyArr[i]<dummyArr[index])
      index=i;
  }
  return index;
}

void gpsData(){
  if (millis() - lastTime > 1000)
  {
```

```
lastTime = millis(); //Update the timer

long latitude = myGNSS.getLatitude();
Serial.print(F("Lat: "));
Serial.print(latitude);

long longitude = myGNSS.getLongitude();
Serial.print(F(" Long: "));
Serial.print(longitude);

long speed = myGNSS.getGroundSpeed();
Serial.print(F(" Speed: "));
Serial.print(speed);
Serial.print(F(" (mm/s)"));

long heading = myGNSS.getHeading();
Serial.print(F(" Heading: "));
Serial.print(heading);
Serial.print(F(" (degrees * 10^-5)"));

int pDOP = myGNSS.getPDOP();
Serial.print(F(" pDOP: "));
```

```
        Serial.print(pDOP / 100.0, 2); // Convert pDOP scaling from 0.01 to 1


        Serial.println();
       }
     }
```

Arduino Nano Code

```
#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#include <Wire.h>
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"
#include "BluefruitConfig.h"

#if SOFTWARE_SERIAL_AVAILABLE
  #include <SoftwareSerial.h>
#endif

/*=========================================================================
    APPLICATION SETTINGS
```

FACTORYRESET_ENABLE Perform a factory reset when running this sketch

Enabling this will put your Bluefruit LE module in a 'known good' state and clear any config data set in previous sketches or projects, so running this at least once is a good idea.

When deploying your project, however, you will want to disable factory reset by setting this value to 0. If you are making changes to your Bluefruit LE device via AT commands, and those changes aren't persisting across resets, this is the reason why. Factory reset will erase the non-volatile memory where config data is stored, setting it back to factory default values.

Some sketches that require you to bond to a central device (HID mouse, keyboard, etc.) won't work at all with this feature enabled since the factory reset will clear all of the bonding data stored on the chip, meaning the

```
                    central device won't be able to reconnect.
  MINIMUM_FIRMWARE_VERSION  Minimum firmware version to have some new features
  MODE_LED_BEHAVIOUR        LED activity, valid options are
                    "DISABLE" or "MODE" or "BLEUART" or
                    "HWUART"  or "SPI"  or "MANUAL"
  ------------------------------------------------------------------------*/
  #define FACTORYRESET_ENABLE        1
  #define MINIMUM_FIRMWARE_VERSION    "0.6.6"
  #define MODE_LED_BEHAVIOUR          "MODE"
/*=========================================================================*/
//SoftwareSerial Serial1;

SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN,
BLUEFRUIT_SWUART_RXD_PIN);

Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
                BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);

// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  while (1);
```

```
}

// function prototypes over in packetparser.cpp
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout);
float parsefloat(uint8_t *buffer);
void printHex(const uint8_t * data, const uint32_t numBytes);

// the packet buffer
extern uint8_t packetbuffer[];
uint8_t *p=NULL;

/**************************************************************************/
/*!
    @brief  Sets up the HW an the BLE module (this function is called
            automatically on startup)
*/
/**************************************************************************/
void setup(void)
{
  while (!Serial);  // required for Flora & Micro
  delay(500);
```

```
Serial.begin(9600);
Serial.println(F("Adafruit Bluefruit App Controller Example"));
Serial.println(F("-----------------------------------------"));

/* Initialise the module */
Serial.print(F("Initialising the Bluefruit LE module: "));

if ( !ble.begin(VERBOSE_MODE) )
{
  error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check wiring?"));
}
Serial.println( F("OK!") );

if ( FACTORYRESET_ENABLE )
{
  /* Perform a factory reset to make sure everything is in a known state */
  Serial.println(F("Performing a factory reset: "));
  if ( ! ble.factoryReset() ){
    error(F("Couldn't factory reset"));
  }
}
```

```
/* Disable command echo from Bluefruit */
ble.echo(false);

Serial.println("Requesting Bluefruit info:");
/* Print Bluefruit information */
ble.info();

Serial.println(F("Please use Adafruit Bluefruit LE app to connect in Controller mode"));
Serial.println(F("Then activate/use the sensors, color picker, game controller, etc!"));
Serial.println();

ble.verbose(false);  // debug info is a little annoying after this point!

/* Wait for connection */
while (! ble.isConnected()) {
    delay(500);
}

Serial.println(F("******************************"));

// LED Activity command is only supported from 0.6.6
```

```
  if ( ble.isVersionAtLeast(MINIMUM_FIRMWARE_VERSION) )
  {
    // Change Mode LED Activity
    Serial.println(F("Change LED activity to " MODE_LED_BEHAVIOUR));
    ble.sendCommandCheckOK("AT+HWModeLED=" MODE_LED_BEHAVIOUR);
  }

  // Set Bluefruit to DATA mode
  Serial.println( F("Switching to DATA mode!") );
  ble.setMode(BLUEFRUIT_MODE_DATA);

  Serial.println(F("******************************"));
  Wire.begin();
}

void DataTran(int Byte){
  Wire.requestFrom(9,1,100);
  Wire.beginTransmission(9);
  Wire.write(Byte);
  Wire.endTransmission();
  //delay(1000);
}
```

```
/**************************************************************************/
/*!
    @brief  Constantly poll for new command or response data
*/
/**************************************************************************/
void loop(void)
{
  /* Wait for new data to arrive */
  uint8_t len = readPacket(&ble, BLE_READPACKET_TIMEOUT);
  if (len == 0) return;

  /* Got a packet! */
  // printHex(packetbuffer, len);

  // Color
  if (packetbuffer[1] == 'C') {
    uint8_t red = packetbuffer[2];
    uint8_t green = packetbuffer[3];
    uint8_t blue = packetbuffer[4];
    Serial.print ("RGB #");
    if (red < 0x10) Serial.print("0");
    Serial.print(red, HEX);
```

```
  if (green < 0x10) Serial.print("0");
  Serial.print(green, HEX);
  if (blue < 0x10) Serial.print("0");
  Serial.println(blue, HEX);
 }

 // Buttons
 if (packetbuffer[1] == 'B') {
  uint8_t buttnum = packetbuffer[2] - '0';
  boolean pressed = packetbuffer[3] - '0';
  p=&buttnum;
  if(buttnum==2){
   DataTran(buttnum);
  }
   if(buttnum==1){
    DataTran(buttnum);
   }
   if(buttnum==5){
    DataTran(buttnum);
   }
   if(buttnum==6){
    DataTran(buttnum);
```

```
  }
  if(buttnum==7){
    DataTran(buttnum);
  }
  if(buttnum==8){
    DataTran(buttnum);
  }
  if(buttnum==4){
    DataTran(buttnum);
  }
  if(buttnum==3){
    DataTran(buttnum);
    delay(420);
    DataTran(0);
  }
 Serial.print ("Button "); Serial.print(buttnum);
 //Serial.print("Button register:");Serial.print(uint8_t(p),HEX);Serial.println();
 if (pressed) {
  Serial.println(" pressed");
 } else {
  Serial.println(" released");
 }
```

```
}

// GPS Location
if (packetbuffer[1] == 'L') {
  float lat, lon, alt;
  lat = parsefloat(packetbuffer+2);
  lon = parsefloat(packetbuffer+6);
  alt = parsefloat(packetbuffer+10);
  Serial.print("GPS Location\t");
  Serial.print("Lat: "); Serial.print(lat, 4); // 4 digits of precision!
  Serial.print('\t');
  Serial.print("Lon: "); Serial.print(lon, 4); // 4 digits of precision!
  Serial.print('\t');
  Serial.print(alt, 4); Serial.println(" meters");
}

// Accelerometer
if (packetbuffer[1] == 'A') {
  float x, y, z;
  x = parsefloat(packetbuffer+2);
  y = parsefloat(packetbuffer+6);
  z = parsefloat(packetbuffer+10);
```

```
  Serial.print("Accel\t");
  Serial.print(x); Serial.print('\t');
  Serial.print(y); Serial.print('\t');
  Serial.print(z); Serial.println();
 }

 // Magnetometer
 if (packetbuffer[1] == 'M') {
  float x, y, z;
  x = parsefloat(packetbuffer+2);
  y = parsefloat(packetbuffer+6);
  z = parsefloat(packetbuffer+10);
  Serial.print("Mag\t");
  Serial.print(x); Serial.print('\t');
  Serial.print(y); Serial.print('\t');
  Serial.print(z); Serial.println();
 }

 // Gyroscope
 if (packetbuffer[1] == 'G') {
  float x, y, z;
  x = parsefloat(packetbuffer+2);
```

```
    y = parsefloat(packetbuffer+6);
    z = parsefloat(packetbuffer+10);
    Serial.print("Gyro\t");
    Serial.print(x); Serial.print('\t');
    Serial.print(y); Serial.print('\t');
    Serial.print(z); Serial.println();
  }

  // Quaternions
  if (packetbuffer[1] == 'Q') {
    float x, y, z, w;
    x = parsefloat(packetbuffer+2);
    y = parsefloat(packetbuffer+6);
    z = parsefloat(packetbuffer+10);
    w = parsefloat(packetbuffer+14);
    Serial.print("Quat\t");
    Serial.print(x); Serial.print('\t');
    Serial.print(y); Serial.print('\t');
    Serial.print(z); Serial.print('\t');
    Serial.print(w); Serial.println();
  }
}
```

Testing

One of the most difficult parts of this project was the testing and debugging phase. In this phase there were three important tests that we needed to run. The general movement test was pretty straight forward in which the Mega sent predetermined instructions to the motor drive in order for us to see if the Mowboitc moved forward in a straight line or not. The conclusion of the first test was that it did not move in a straight line. This is because of the variation of internal resistance of the motors. This caused the Mowbotic to drift in either direction depends on different road surface. The only way to correct the drifting is to implement a pair of motor encoders which provide feedback signals to the Arduino Mega. Based on the signals, the Mega will determine which motor need more power to compensate the motor deficiencies. The second way to do that is by adding a IMU to the system and programming an algorithm which would have the mega use the IMU to correct the movement of the Mowbotic if it started to drift. Unfortunately, due to budget and time constraints this fix could not be implemented. Instead, some hard coded values were added to change the power and angle of the Sun Joe very slightly.

The following test was to check whether the Mowbotic could travel a square. This test was utilized to address assuming the Mowbotic could make a trip a region given to it.

The primary region it got was a 12 by 12 square and it fizzled with the main code. This test required a great deal of experimentation because of turning speed. Once in a while it would turn excessively or too little thus the time it turned would should be changed. The test had similar inconveniences as the first as the engines didn't generally arrange as they should which brought about the Mowbotic drifting. Remedies were hard coded into the computerization code however didn't generally address the floating at time.

The last test was the item location test. This is a significant safety test as we expected to ensure in case something came quite close to the Mowbotic, that it would know to stop immediately. In Theory, this test has been straight forward as the code was already written, when testing however, it demonstrated in any case. This test required around three days to run and was perhaps the most irritating one. This is a result of another strange error we had where when the bot is turning, and it goes to actually look at the ultrasonic sensors, it stops the engines. So, for the brief moment it takes to actually look at the two sensors, the engines

would stop, which isn't great just as really looking at the sensors many times each second. This made us need to take object discovery out when utilizing the turn work.

Conclusion

Access to Support

Thankfully the pandemic has not been as much of an issue as it would have been in previous semesters; Although we still have to be precautious and follow certain school regulations or protocols. Throughout the semester we were able to use the library and labs on campus as our meeting location as well as using the equipment from the lab. That was the extent of support provided by the school. As previously mentioned in our budget, our project most will most likely exceed our budget which is astronomical given that we weren’t provided with funding of any kind.

Components:

We also had difficulties with finding components that would fit on our project as we were not very familiar with what is available in the market. We keep learning every day and have envisioned what we want to have as the final product. But there are also enhancements we would like to add if the time allow us but are not

Integral to the project:

- Solar Power Charging - Allow us to forego a charging station altogether and run the lawn mower without as much interruption
- Extend Battery Life - Being hard capped to a 1 hour run time is unfortunate, if we are able to adjust/modify the battery of the lawn mower itself we can have a much smoother experience.
- Modify the build - Because the lawn mower is automated there is no need for it to be as massive as it is. We would like to remove the handle to start off and potentially have a custom box for the components to fit in/on.

Time Constraint:

The biggest issue we found is time constraint on our project. While moving forward into our research we found it more difficult to finish in its entirety, as our project has a deep amount of complexity. With efforts and contributions from all our members we are able to progress as far as we can while meeting through zoom calls or in-person every week, and possibly make further enhancements.

Budget:

Budget is also a big factor in our project. We are trying to minimize our expenses as much as we can since the entire project will be out of our pocket. Theoretically we have reached $800 as a group in components.

Refference

https://create.arduino.cc/projecthub/ruchir1674/how-to-interface-gps-module-neo-6m-with-arduino-8f90ad

https://www.gabrielcsapo.com/arduino-web-server-mega-2560-r3-built-in-esp8266/

https://www.amazon.com/Mighty-Max-Battery-Electric-Scooter/dp/B00RZHIG70

https://www.snowjoe.com/products/sun-joe-14-inch-28-volt-cordless-lawn-mower

https://www.waveshare.com/w/upload/2/2c/NEO-6-DataSheet.pdf

https://store.arduino.cc/usa/arduino-nano

https://www.ebay.com/itm/253376031415?mkevt=1&mkcid=28&chn=ps

https://www.husqvarna.com/us/products/robotic-lawn-mowers/models/automower-115h/

https://www.dimensionengineering.com/datasheets/Sabertooth2x32.pdf

https://robotdyn.com/mega-wifi-r3-atmega2560-esp8266-flash-32mb-usb-ttl-ch340g-mic

https://www.reddit.com/r/mildlyinteresting/comments/6g2275/the_pattern_on_this_lawn_makes_it_look_3d/

https://www.brickhousesecurity.com/gps-trackers/gps-accuracy/

https://www.scag.com/pro-tip/lawn-striping-and-lawn-patterns/

https://www.lawnstarter.com/ny

http://ceng786-hw1-e162702.blogspot.com/2013/10/tangent-bug-planning-algorithm-with.html