ME102b Team 13 Final Project: Mech-E

Team Members: Adam Stewart, Ben Shoemaker, Jaeyun Ha, Seong Ho Yang

Opportunity:

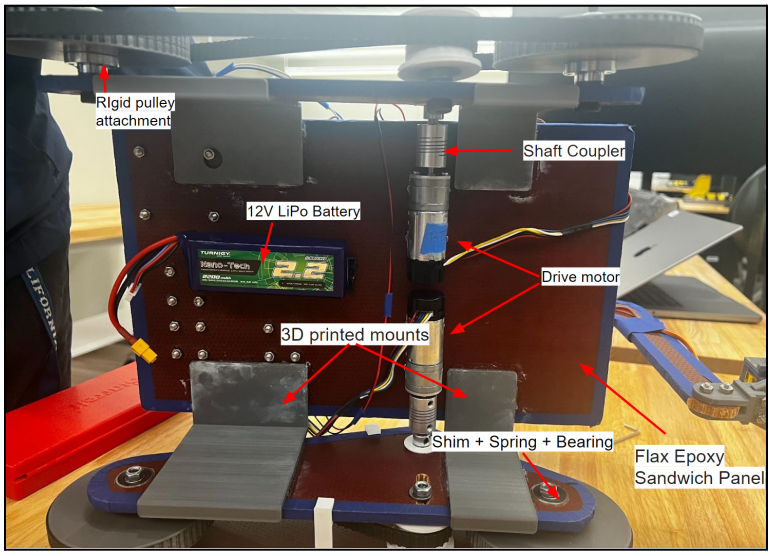
Our team wanted to design a robot that could be remote controlled in order to enter potentially hazardous or dangerous areas to pick up and retrieve objects without human contact.

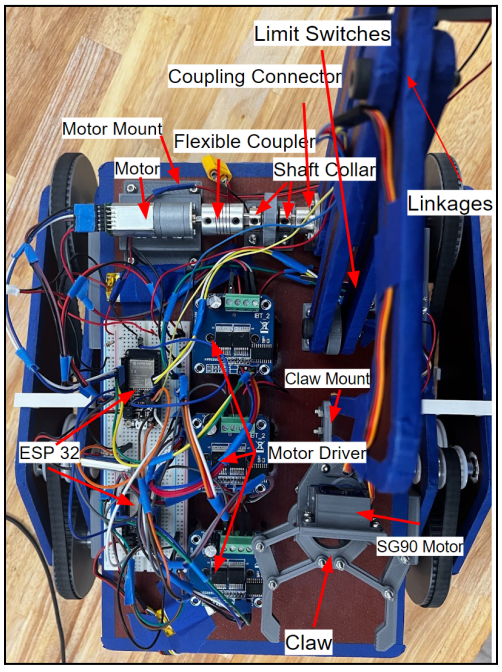
High-Level Strategy:

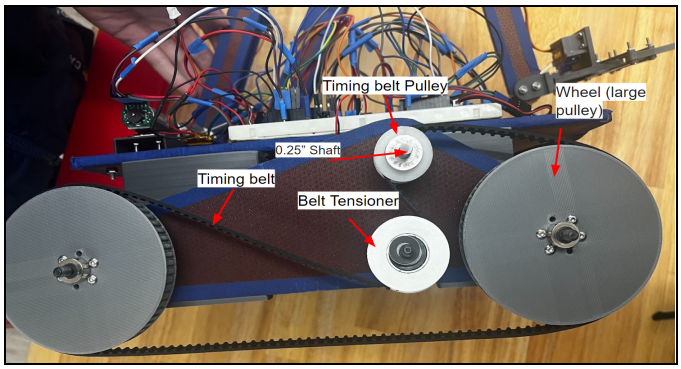
Our solution is "Mech-E," a robot that consists of two primary systems: a treaded drivetrain for both linear movement and turning, and a 4-bar linkage transmission that moves a retractable claw. There are 2 drive motors for the treads—one on each side—-a third motor to drive the 4-bar linkage, and finally a small servo motor to open and close the claw. The treads allow for movement over a variety of surfaces, and the 4-bar linkage can position the claw at different heights for grabbing objects. The user controls Mech-E wirelessly over WIFI using the ESP-NOW protocol with Arduino IDE. This is done via an external control panel with four button inputs and three potentiometers. The first button turns Mech-E "on," allowing movement of the linkage arm and priming the robot for subsequent actions. The next three buttons toggle the different states on or off: linear motion (in which the drive motors move in the same direction), turning in place (where the drive motors move in opposite directions), and control of the claw. Finally, the 3 potentiometers allow for continuous adjustment of the speed of the drive motors, the position of the linkage, and the expansion/retraction of the claw.

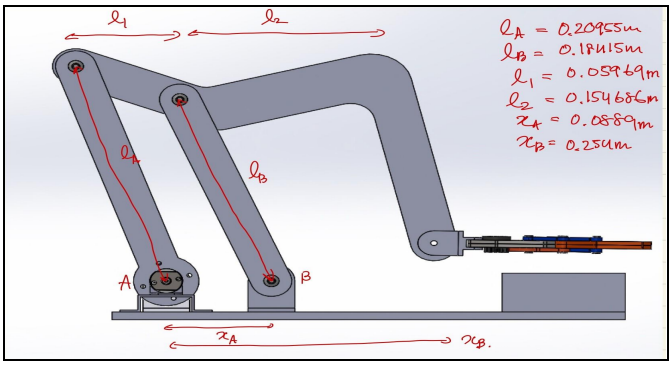
Our original plan was to have the claw be mounted to a fully flexible robotic arm that could rotate about its base as well as have two separate pivot joints that could individually be controlled, which would have required 3 motors. However, we then realized that these extra degrees of freedom are unnecessary, because the drive system already allows Mech-E to turn and orient itself accordingly. Thus, we simplified the claw transmission to a 4-bar linkage that can extend and retract the claw in a single plane, which only requires one motor to drive the linkage system. We also originally planned to include additional sensors such as an adjustable camera with live feed as well as gas sensors to detect the presence of CO or CO_2 in hazardous environments. We decided these were out of the scope of the project, and instead focused our time on making robust and refined transmission systems for the treads and claw.

Physical Device and Integrated Components:









Function-Critical Calculations:

Linkage Calculations: The purpose of these is to find the approximate reaction forces at the pins where we have ball bearings, as well as estimate the external load that the linkage system can sustain (see the length definitions in the diagram above). The max motor torque was based on the Pololu spec sheet. We calculated the max torque by taking 60% of the stall torque (25 kg*mm), which was 15 kg*mm. We used the highest-torque 12V 20D motor that Pololu sells in order to maximize the allowed mass of the linkage.

M = 0.150 kg, $L_a = 21cm$, $L_b = 19cm$

Force components: *In x direction:* $F_{Ax} + F_{Bx} = 0$; *In y direction:* $F_{Ay} + F_{By} = F_{motor}$

At point A: $F_{Ay}X_a + F_{Ax}L_a + F_{Bx}L_b + F_{By}X_b - F_{motor}L_2 + \tau_{motor} = 0$; $F_{Ay} = F_{By} = 0$

$F_{Ax} = \frac{F_{motor}L_2 + \tau_{motor}}{L_a - L_b}$ = 0.5995 N which suggests that $F_{Bx} = -0.5995N$

Momentum analysis :

$-F_{motor}L_5 - F_{By}L_{AB} + \tau_{motor} = 0 \Rightarrow$ *Reaction forces:* $F_{By} = -6.96N$, $F_{Ay} = 7.107N$

These forces can easily be handled by typical skateboard bearings.

Finding the max weight the linkage can handle:

Stall torque of motor = 15 kg*m = 1.472 Nm

Using SolidWorks, we were able to find the new coordinate of the motor once our linkage was extended.

$r = <0.327, 0.010, 0> m$

So the moment balance becomes: $i(zF_y - yF_Z) - j(F_xy - F_yx) + k(F_xy - F_yx) = \tau k$

Thus, $F_xy - F_{load}x = \tau$ where $F_y = F_{load}$ and the horizontal component of the force is 0.

Finally, $F_{load} = 4.508N$. This suggests that the linkage can hold up to about 0.460 kg.

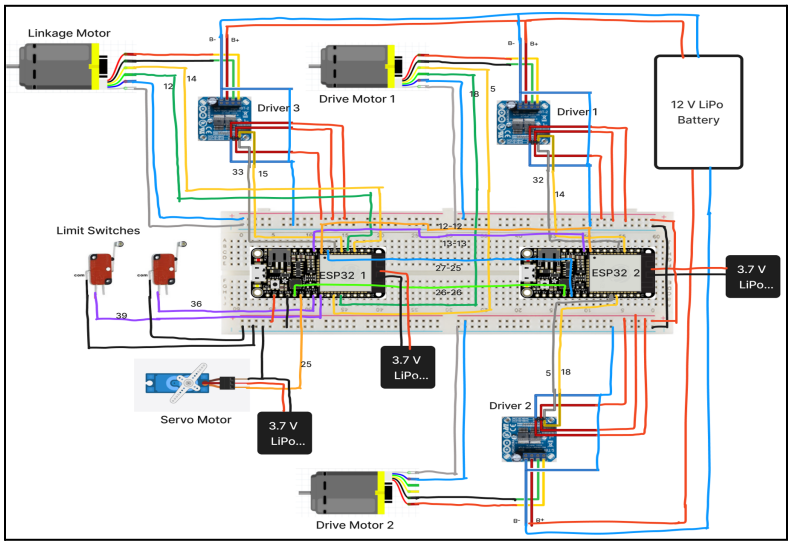
Finding the max speed of the vehicle:

As the vehicle wheels are placed on bearings we have assumed that the friction of this device is negligible. As we only care about the top speed and not the acceleration we can look solely at the RPM of the motor and our timing belt configuration. We pulled from the Pololu spec sheet that our motors under light load should have approximately an RPM of 400.

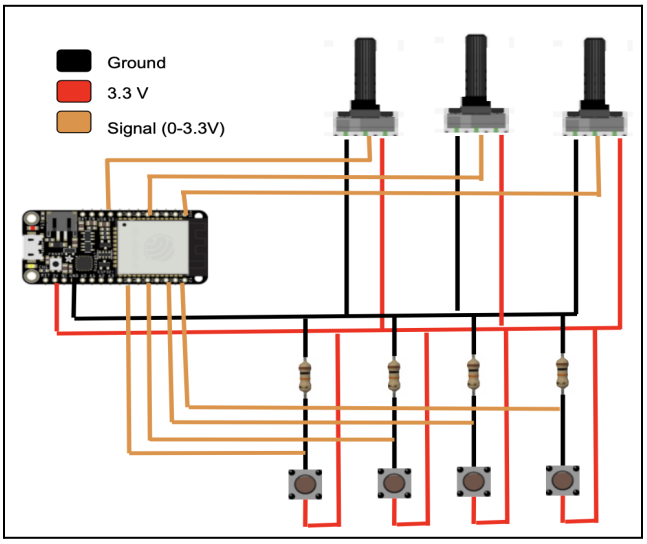
$$Desired\ Max\ Drive\ Motor\ RPM = \frac{Desired\ Speed\ in\ Meters\ per\ second * 60\ seconds * Gear\ Ratio}{Diameter\ of\ the\ Drive\ Pulley * \pi}$$

We arbitrarily set the target speed as 0.5 meters per second as this is reasonably fast for a treaded system of this size. The designed gear ratio was 4, and the diameter of the large pulley is 0.097 meters. This requires a target RPM of 395, so we consider our motor to be well chosen. We then did physical testing of the vehicle and over 7.625 meters the vehicle traveled at a max speed of 0.545 meters/second.

Circuit Diagrams:

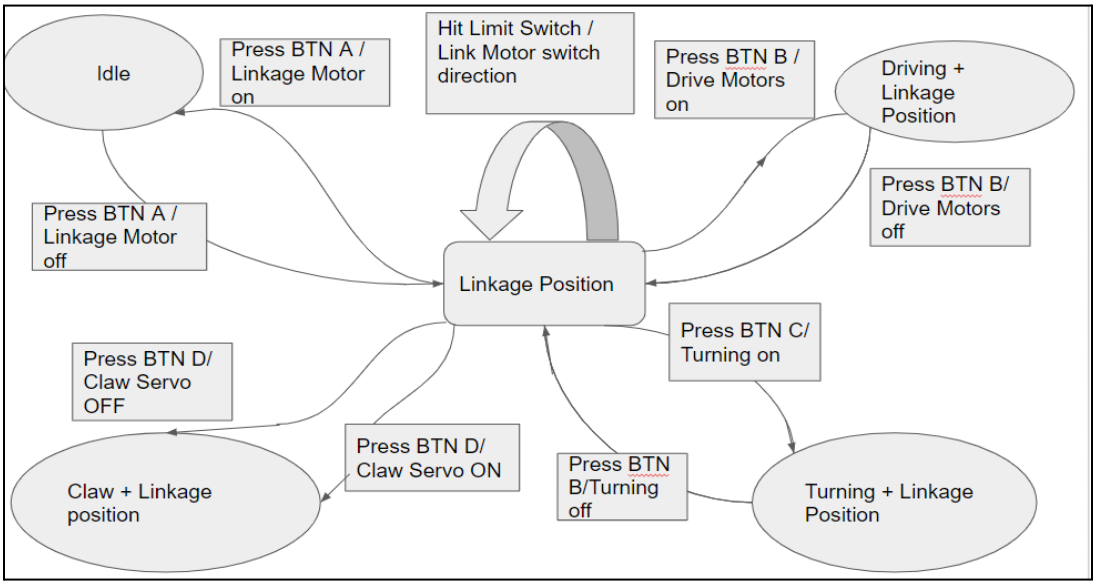


Vehicle Circuit Diagram



Controller Circuit Diagram

State Transition Diagram:



Reflection:

We believe the project was a great learning experience where we learnt to mix all the different components of the course together. We were able to create a robust transmission system integrated with a fully functional linkage system with attached servo. However, we do think we could have further improved our project. For the calculations, we neglected various factors such as linkage mass which could have contributed to the error of the calculated figures and the actual figure. In the future, it would be best to account for mass. For the linkage, we initially used acrylic but once manufactured it was too heavy so we switched to composite sandwich panels. In addition, we had issues with the linkage where it would oscillate during position control. This was mainly a hardware issue, since the set screws and mounting screws both easily became loose, so using Loctite or another adhesive would have been helpful.

Appendix

Bill of Materials:

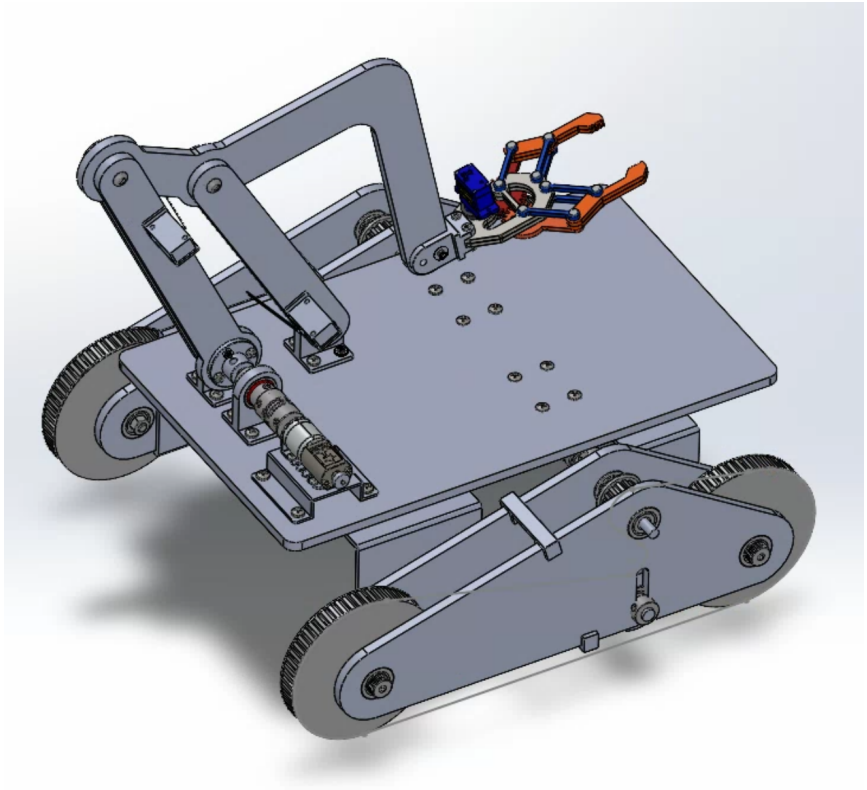
Team 13 BOM and Orders							
Item Description	**Vender**	**Quantity Bought**	**Total Cost**	**Who Bought it**	**Reimbursed?**	**Arrived?**	**LINK:**
Motor Drivers	Amazon	1	$18.99	Adam	No	Yes	Here
Shaft flanged connector	Amazon	1	$9.99	Adam	No	Yes	Here
Flexible shaft coupler	Amazon	1	$11.03	Adam	No	Yes	Here
Rotary shaft	Amazon	1	$7.49	Adam	No	Yes	Here
Shaft collars	Amazon	1	$8.99	Adam	No	Yes	Here
Assorted M2 - M5 screw set	Amazon	1	$19.89	Adam	No	Yes	Here
Set Screw Shaft Collar	McMaster	4	$5.44	Adam	No	Yes	Here
Lightweight Timing Belt Pulley	McMaster	2	$20.22	Adam	No	Yes	Here
40mm Steel Shoulder Screws	McMaster	3	$6.18	Adam	No	Yes	Here
8mm Steel Shoulder Screws	McMaster	6	$14.10	Adam	No	Yes	Here
Zinc-Plated Steel Hex Nut	McMaster	1	$3.14	Adam	No	Yes	Here
Ring Shim	McMaster	1	$9.14	Adam	No	Yes	Here
Belleville Disc Spring	McMaster	2	$7.04	Adam	No	Yes	Here
1/4" Rotary shaft	Amazon	1	$8.88	Adam	No	Yes	Here
Alminum bar stock (total price with above)	Amazon	1	$16.89	Adam	No	Yes	Here

2x 1/4" Flanged Bearings	Amazon	2	$15.42	Ben	No	Yes	Here
25D Brushed DC Motor	Pololu	2	$127.66	Ben	No	Yes	Here
20D Brushed DC Motor	Pololu	1	$29.95	Ben	No	Yes	Here
Encoder for 20D Motor	Pololu	1	$8.95	Ben	No	Yes	Here
20D Motor Mounting Bracket	Pololu	1	$7.95	Ben	No	Yes	Here
25D Motor Mounting Bracket	Pololu	1	$7.95	Ben	No	Yes	Here
Universal Mounting Hub	Pololu	2	$19.90	Ben	No	Yes	Here
370XL Timing Belt	McMaster	2	$12.37	Ben	No	Yes	Here
Flax-Epoxy-Nomex Honeycomb Composite Sandwich Panel	Ben	N/A	$0.00	Ben	N/A	Yes	
3D Printed Components	Jaeyun	N/A	$0.00	Jaeyun	N/A	Yes	
ESP32	Microkit	3	$0.00	N/A	N/A	Yes	
Jumper Wires	MicroKit	20+	$0.00	N/A	N/A	Yes	
Breadboard	Microkit	2	$0.00	N/A	N/A	Yes	
10K Ohm Potentiometers	Microkit	3	$0.00	N/A	N/A	Yes	
10KOhm Resistors	Microkit	4	$0.00	N/A	N/A	Yes	
Push Button	Microkit	4	$0.00	N/A	N/A	Yes	
Limit Switch	Ben	2	$0.00	Ben	N/A	Yes	
TOTALS FOR EACH ORDER (includes the shipping and tax, which is not listed above)							
Amazon Order 1	$84.21						

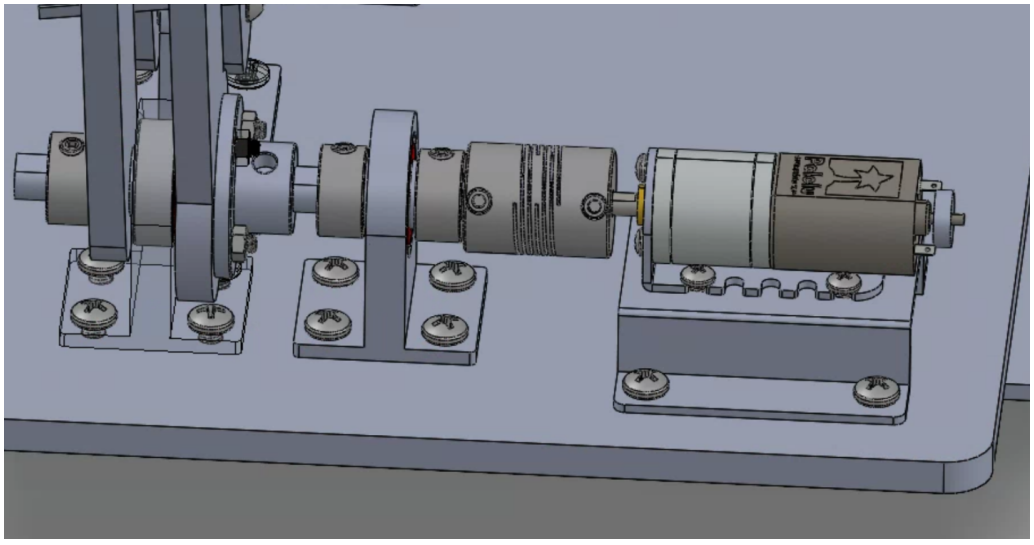
Amazon Order 2	$27.49						
McMaster Order 1	$81.74						
Amazon Order 3	$15.42						
Pololu Order	$202.36						
McMaster Order 2	$12.37						
TOTAL SPENT:	$423.59						

CAD:

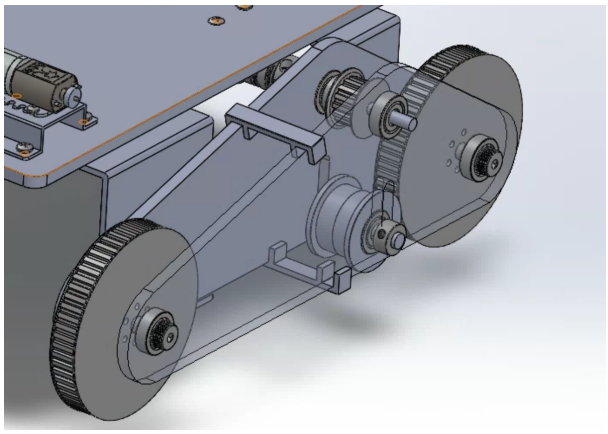
1. Overall view of assembly (not including the wiring, motor drivers, or ESP32):



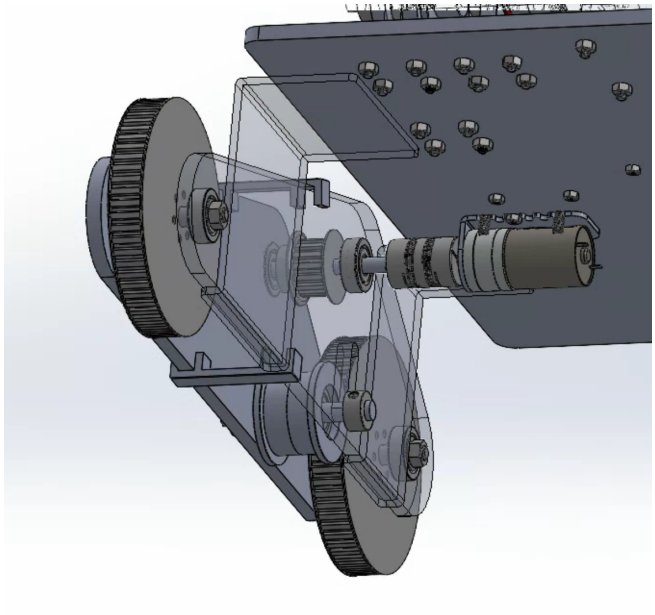
2. Close-up of linkage transmission subassembly:



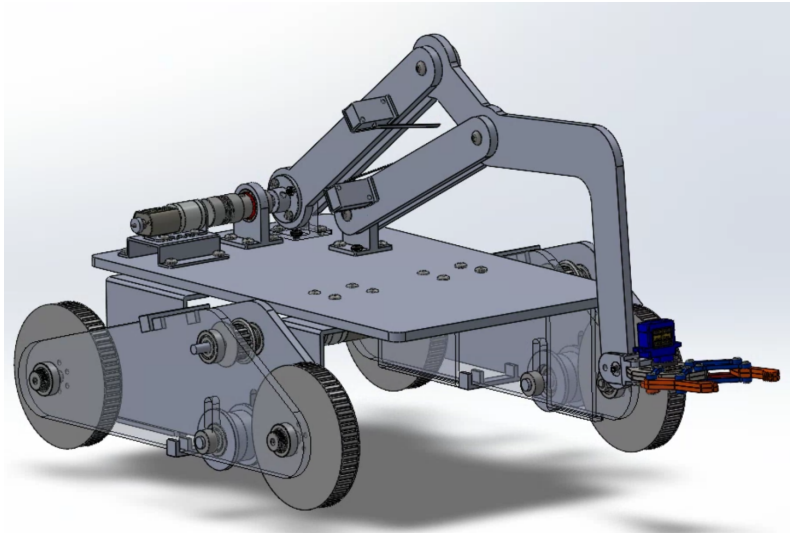
3. Close-up of drivetrain tread transmission system:



4. Drive motor mounting and transmission on the underside of the base:



5. Front isometric view with linkage extended:



Code:

Code 1 : Remote controller (external/separate ESP 32)

```
// download libraries
#include <esp_now.h>
#include <WiFi.h>
#include <Arduino.h>

// defining the constants
#define BTNA 36
#define BTNB 4
#define BTNC 5
#define BTND 18

#define led 21
#define POT 34
#define POT2 39
#define POT3 33

volatile bool buttonIsPressedA = false;
volatile bool buttonIsPressedB = false;
volatile bool buttonIsPressedC = false;
volatile bool buttonIsPressedD = false;
volatile bool debounceT = false;
int state = 1 ;
hw_timer_t * timer0 = NULL;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// initialize the variables
void IRAM_ATTR isrA() {  // the function to be called when interrupt is triggered
  buttonIsPressedA = true;
  timerStart(timer0);
}

void IRAM_ATTR isrB() {  // the function to be called when interrupt is triggered
  buttonIsPressedB = true;
  timerStart(timer0);
}
void IRAM_ATTR isrC() {  // the function to be called when interrupt is triggered
  buttonIsPressedC = true;
  timerStart(timer0);
}
void IRAM_ATTR isrD() {  // the function to be called when interrupt is triggered
  buttonIsPressedD = true;
  timerStart(timer0);
}
/// potval init
int potvals = 0 ;
int potvals2 = 0 ;
int potvals3 = 0 ;

void IRAM_ATTR onTime() {
  portENTER_CRITICAL_ISR(&timerMux);
```

```
  debounceT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux);
  timerStop(timer0);
}
void TimerInterruptInit() {  //The timer simply counts the number of Tic generated by the quartz. With a quartz clocked at 80MHz, we will have 80,000,000 Tics.
  timer0 = timerBegin(0, 80, true); // divides the frequency by the prescaler: 80,000,000 / 80 = 1,000,000 tics / sec
  timerAttachInterrupt(timer0, &onTime, true);    // sets which function do you want to call when the interrupt is triggered
  timerAlarmWrite(timer0, 250000, true);         // sets how many tics will you count to trigger the interrupt
  timerAlarmEnable(timer0); // Enables timer
  timerStop(timer0);
}
// REPLACE WITH YOUR RECEIVER MAC Address
//uint8_t broadcastAddress[] = {0x0C, 0xDC, 0x7E, 0xCB, 0x09, 0x64};
uint8_t broadcastAddress[] = {0x40, 0xF5, 0x20, 0x45, 0xE4, 0xEC};

// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
  //char a[32];
  int a;
  int b;
  int c;
  int d;

} struct_message;

// Create a struct_message called myData
struct_message myData;

esp_now_peer_info_t peerInfo;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  //Serial.print("\r\nLast Packet Send Status:\t");
  //Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}
void setup() {
  pinMode(BTNA, INPUT);  // configures the specified pin to behave either as an input or an output
  pinMode(BTNB, INPUT);
  pinMode(BTNC, INPUT);
  pinMode(BTND, INPUT);

  attachInterrupt(BTNA, isrA, RISING);
  attachInterrupt(BTNB, isrB, RISING);
  attachInterrupt(BTNC, isrC, RISING);
  attachInterrupt(BTND, isrD, RISING);
```

```
  Serial.begin(115200);
  TimerInterruptInit(); // Initiates timer interrupt

  // Init Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
}

void loop() {
  switch (state) {
    case 1:  //idle
    dataPack() ;
    // Event checker
      if (CheckForButtonPressA() == true) {
        // service
        Serial.println("Position only ");
        state = 2;
      }
      break;

    case 2:
    dataPack();
    // Event checker
      if (CheckForButtonPressA() == true) {
        // Service
        Serial.println("idling");
        state = 1;
      }
      // Event checker
```

```
      if (CheckForButtonPressB() == true) {
        // service
        Serial.println("Drive");
        state = 3;
      }
      // Event checker
      if (CheckForButtonPressC() == true) {
        // Service
        Serial.println("Turn ");
        state = 4;
      }
      if (CheckForButtonPressD() == true) {
        Serial.println("Claw");
        state = 5;
      }
      break;

    case 3:
    dataPack();
    // Event checker
      if (CheckForButtonPressB() == true) {
        // Service
        Serial.println("Pos");
        state = 2;
      }

      break;

      case 4: //Turn + Pos
      dataPack() ;
      // Event checker
      if (CheckForButtonPressC() == true) {
        // Service
        Serial.println("Pos");
        state = 2;
      }

      case 5: //Claw + Pos
      dataPack();
      // Event checker
      if (CheckForButtonPressD() == true) {
        // Service
        Serial.println("Pos");
        state = 2;
      }
      break;
  //ESP.restart() ;
}
}
void dataPack() {
  potvals = analogRead(POT) ;
  potvals2 = analogRead(POT2) ;
```

```
  potvals3 = analogRead(POT3);

  // Set values to send
  //strcpy(myData.a, "THIS IS A CHAR");
  myData.a = potvals;
  myData.b = potvals2;
  myData.c = potvals3;
  myData.d = state ;
  Serial.println(state) ;
//  if ((potvals2 > 0)&&(potvals2 < 2048)){
//    myData.b = 0 ;
//  }
//  else if ((potvals2 > 2048)&&(potvals2 < 4095)){
//    myData.b = 1 ;
//  }
  // Send message via ESP-NOW
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));

  if (result == ESP_OK) {
    //Serial.println("Sent with success");
  }
  else {
   // Serial.println("Error sending the data");
  }
  delay(50);
}
bool CheckForButtonPressA (){
  // Event checker
  if (debounceT && buttonIsPressedA){
    // Service
    debounceTimerReset();
    clearAflag();
  return true ;
  }
  else{
  return false;
  }
}

bool CheckForButtonPressB (){
  // Event
  if (debounceT && buttonIsPressedB){
    // Service
    debounceTimerReset();
    clearBflag();
  return true ;
  }
  else{
  return false;
  }
}
```

```
bool CheckForButtonPressC (){
  // Event checker
  if (debounceT && buttonIsPressedC){
    // Service
    debounceTimerReset();
    clearCflag();
  return true ;
  }
  else{
  return false;
  }
}

bool CheckForButtonPressD(){
  // Event
  if (debounceT && buttonIsPressedD){
    // Service
    debounceTimerReset();
    clearDflag();
  return true ;
  }
  else{
  return false;
  }
}
void clearAflag() {
  buttonIsPressedA = false;
}

void clearBflag() {
  buttonIsPressedB = false;
}

void clearCflag() {
  buttonIsPressedC = false;
}

void clearDflag() {
  buttonIsPressedD = false;
}


void debounceTimerReset(){
  portENTER_CRITICAL(&timerMux);
    debounceT = false;
    portEXIT_CRITICAL(&timerMux);
    timerRestart(timer0);
}
```

Code 2 : Receiver code for main ESP32 (ESP32 1 in the vehicle circuit diagram)

```
//library
#include <esp_now.h>
#include <WiFi.h>
#include <ESP32Servo.h>
#include <ESP32Encoder.h>
Servo myservo ;

// pins and all
#define RXD2 13
#define TXD2 12
int trans = 0;
int endmes = 97;
int state = 1 ;
bool driveMotorFlag = false ;
bool turnFlag = false ;
bool idleFlag = true ;
const int deadRange = 15 ;

//define pins and all
#define backLimit 36
#define frontLimit 39
#define DRIVE 27
#define TURN 26
#define servopin 25
#define BIN_1 15
#define BIN_2 33
#define LED_PIN 21     // CHANGE POTENTIOMETER PIN HERE TO MATCH CIRCUIT IF NEEDED

ESP32Encoder encoder;
ESP32Encoder encoderDM;

// defining the variables ; ;;

int potvals = 0 ;
int potvals2 = 0 ;
int potvals3 = 0 ;

int theta = 0;
int thetaDes = 0;
int thetaMax = 1000;     // 488.3 * 6 counts per revolution
int D = 0;
//int potReading = 0;
int potMax = 3020; // max potentiometer reading

int Kp = 8;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
int Ki = 0.1;
int KiMax = 50;
int e_accum = 0;

//Setup Drive Motor Variables
volatile int countDM = 0;
```

```
volatile bool deltaTDM = false;
int D1 = 0;
int KpD = 8;
int KiD = 1;
int accError = 0;
int Imax = 100;
int omegaSpeedDM = 0;
int omegaMax = 4536; //Change this to max open loop speed
int omegaDes = 0;
int dir = 1;
bool backHit = false ;
bool frontHit = false ;
int forwardRate = 10 ;
int backwardRate = -10 ;

//Setup interrupt variables ----------------------------
volatile int count = 0; // encoder count
volatile bool deltaT = false;     // check timer interrupt 2
hw_timer_t * timer0 = NULL;
hw_timer_t * timer1 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
portMUX_TYPE timerMux1 = portMUX_INITIALIZER_UNLOCKED;

// setting PWM properties ---------------------------
const int freq = 5000;
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int ledChannel_d = 4 ;
//const int ledChannel_3 = 3;
//const int ledChannel_4 = 4;
//const int ledChannel_5 = 5;
//const int ledChannel_0 = 0;
const int resolution = 8;
const int MAX_PWM_VOLTAGE = 255;
const int NOM_PWM_VOLTAGE = 150;
//define the structure for sending pot values
typedef struct struct_message {
    //char a[32];
    int a;
    int b;
    int c;
    int d;
} struct_message;
struct_message myData;

//callback func when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&myData, incomingData, sizeof(myData));
// FOR DEBUGGING ONLY
  //vals = Serial.read() ;
  //m ,,,,,,Serial.print("Bytes received: ");
  //Serial.println(len);
```

```
//  Serial.print("potentiometer_value1: ");
//  Serial.println(myData.a);
//  Serial.print("potentiometer_value2: ");
//  Serial.println(myData.b);
//  Serial.print("potentiometer_value3: ");
//  Serial.println(myData.c);
    //Serial.print("potentiometer_value4: ");
    //Serial.println(myData.b);
//  Serial.println();
}

//Initialization -----------------------------------
void IRAM_ATTR onTime0() {
  portENTER_CRITICAL_ISR(&timerMux0);
  countDM = encoderDM.getCount( );
  encoderDM.clearCount ( );
  deltaTDM = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux0);
}
void IRAM_ATTR onTime1() {
  portENTER_CRITICAL_ISR(&timerMux1);
  count = encoder.getCount( );
  encoder.clearCount ( );
  deltaT = true; // the function to be called when timer interrupt is triggered
  portEXIT_CRITICAL_ISR(&timerMux1);
}
void IRAM_ATTR isrBack() {  // the function to be called when interrupt is triggered
  backHit = true ;
  D = forwardRate ;
}
void IRAM_ATTR isrFront() {  // the function to be called when interrupt is triggered
  frontHit = true ;
  D = backwardRate  ;
}
//HardwareSerial SerialPort(2);
void setup() {
  pinMode(frontLimit, INPUT);
  pinMode(backLimit, INPUT);

  Serial2.begin(115200, SERIAL_8N1, RXD2, TXD2);
  // put your setup code here, to run once:
  //ESP32PWM::allocateTimer(2) ;
  ESP32PWM::allocateTimer(3) ;
  myservo.attach(servopin) ;
  pinMode(TURN,OUTPUT);
  pinMode(DRIVE,OUTPUT);

  WiFi.mode(WIFI_STA) ;

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error with initializing ESP-NOW") ;
    return ;
```

```
  }
  esp_now_register_recv_cb(OnDataRecv) ;

  Serial.begin(115200);
  ESP32Encoder::useInternalWeakPullResistors = UP; // Enable the weak pull up resistors
  encoder.attachHalfQuad(14, 32); // Attache pins for use as encoder pins
  encoder.setCount(0);  // set starting count value after attaching

  encoderDM.attachHalfQuad(18, 5); // Attache pins for use as encoder pins
  encoderDM.setCount(0);  // set starting count value after attaching

  // configure LED PWM functionalitites
  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);
  //ledcSetup(ledChannel_d, freq, resolution);

//  ledcSetup(ledChannel_3, freq, resolution);
//  ledcSetup(ledChannel_4, freq, resolution);
//  ledcSetup(ledChannel_5, freq, resolution);
//  ledcSetup(ledChannel_0, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);
  //ledcAttachPin(data1, ledChannel_d);
//  ledcAttachPin(BIN_3, ledChannel_3);
//  ledcAttachPin(BIN_4, ledChannel_4);
//  ledcAttachPin(BIN_5, ledChannel_5);
//  ledcAttachPin(BIN_6, ledChannel_0);

  // initilize timer
  timer0 = timerBegin(0, 80, true);  // timer 0, MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer0, &onTime0, true); // edge (not level) triggered
  timerAlarmWrite(timer0, 10000, true); // 5000000 * 1 us = 5 s, autoreload true

  timer1 = timerBegin(1, 80, true);  // timer 1, MWDT clock period = 12.5 ns *
TIMGn_Tx_WDT_CLK_PRESCALE -> 12.5 ns * 80 -> 1000 ns = 1 us, countUp
  timerAttachInterrupt(timer1, &onTime1, true); // edge (not level) triggered
  timerAlarmWrite(timer1, 10000, true); // 10000 * 1 us = 10 ms, autoreload true

  // at least enable the timer alarms
  timerAlarmEnable(timer0); // enable
  timerAlarmEnable(timer1); // enable
}

void encodersLoop(){
  if (deltaTDM) {
      portENTER_CRITICAL(&timerMux0);
      deltaTDM = false;
      portEXIT_CRITICAL(&timerMux0);
      omegaSpeedDM = countDM;
```

```
      //Serial.println(deltaTDM);
  }
  if (deltaT) {
      portENTER_CRITICAL(&timerMux1);
      deltaT = false;
      portEXIT_CRITICAL(&timerMux1);
      if (bool(digitalRead(frontLimit))== false && bool(digitalRead(backLimit)) == false){
      theta += count;
      //Serial.println(theta) ;
      }
}
}
void driveMotor() {
      potvals3 = myData.c;

      //Serial.print("omegaSpeed");
      //Serial.println(omegaSpeedDM);
      omegaDes = dir*map(potvals3, 0, 4095, -omegaMax, omegaMax); // PLEASE SPECIFY OMEGAMAX
VALUE ABOVE

      accError = accError + (omegaDes - omegaSpeedDM);
      //Serial.print("accError");
      //Serial.println(accError);
      if (accError >= Imax){
        accError = Imax;
      }
      else if (accError <= -1*Imax){
        accError = -1*Imax;
      }

       D1 = KpD*(omegaDes - omegaSpeedDM) +KiD*(accError);
       //Serial.print("D1T");
       //Serial.println(D1T);
  if (D1 > MAX_PWM_VOLTAGE) {
          D1 = MAX_PWM_VOLTAGE;
      }
      else if (D1 < -MAX_PWM_VOLTAGE) {
          D1 = -MAX_PWM_VOLTAGE;
      }
//      if (D1 > 0 && D1 < deadRange) {
//        D1 = 0 ;
//      }
//      if (D1 < 0 && D1 > -deadRange) {
//        D1 = 0 ;
//      }
//

  trans = map(D1, -MAX_PWM_VOLTAGE, MAX_PWM_VOLTAGE, 1000, 1511);
  Serial2.println(trans);
  Serial.println(trans) ;
  Serial2.write(97);
  delay(50); //This delay is needed to prevent the data being sent too quickly to the aux
```

```
esp32 and filling up the buffer
}
void turnMotor() {

      potvals3 = myData.c;

      //Serial.print("omegaSpeed");
//      Serial.print("OmegaSpeedDM");
//      Serial.println(omegaSpeedDM);
      omegaDes = dir*map(potvals3, 0, 4095, -omegaMax, omegaMax); // PLEASE SPECIFY OMEGAMAX
VALUE ABOVE

      accError = accError + (omegaDes - omegaSpeedDM);
      //Serial.print("accError");
      //Serial.println(accError);
      if (accError >= Imax){
        accError = Imax;
      }
      else if (accError <= -1*Imax){
        accError = -1*Imax;
      }

       D1 = KpD*(omegaDes - omegaSpeedDM) +KiD*(accError);
       //Serial.print("D1T");
       //Serial.println(D1T);
  if (D1 > MAX_PWM_VOLTAGE) {
          D1 = MAX_PWM_VOLTAGE;
      }
      else if (D1 < -MAX_PWM_VOLTAGE) {
          D1 = -MAX_PWM_VOLTAGE;
      }
//      if (D1 > 0 && D1 < deadRange) {
//        D1 = 0 ;
//      }
//      if (D1 < 0 && D1 > -deadRange) {
//        D1 = 0 ;
//      }
//

  trans = map(D1, -MAX_PWM_VOLTAGE, MAX_PWM_VOLTAGE, 1000, 1511);
  Serial2.println(trans);
  //Serial.println(D1) ;
  Serial2.write(97);
  delay(50);

}
void driveMotorsOn() {
  digitalWrite(DRIVE,HIGH) ;
}
void driveMotorsOff() {
  digitalWrite(DRIVE,LOW);
}
```

```
void turnMotorsOn() {
  digitalWrite(TURN,HIGH) ;
}
void turnMotorsOff() {
  digitalWrite(TURN,LOW);
}

void loop() {

  state = myData.d ;
  Serial.print("State : ");
  Serial.println(state) ;
  encodersLoop() ;
  // event checker
  if (state == 1 ) {
    idleFlag = true ;
  }
  // Event checker
  if (state == 2 && idleFlag ) {
    // Service
    LinkageMotor();
    idleFlag = false ;
  }
  // Event checker
  if (state == 2 && !idleFlag) {
    // Service
    LinkageMotor() ;
    driveMotorsOff();
    turnMotorsOff();
  }
  // Event checker
  if (state == 3 ) {
//    Service checker
    LinkageMotor();
    driveMotor() ;
    driveMotorsOn();
  }
  // Event checker
  if (state == 4 ) {
    // Service
    LinkageMotor() ;
    turnMotor() ;
    driveMotorsOn() ;
    turnMotorsOn();

  }
  // Event checker
  if (state == 5) {
    // Service
    LinkageMotor();
    servoClaw() ;
  }
```

```
  //delay(15);
}
//Other functions
void servoClaw() {
  potvals = myData.a ;
  potvals = map(potvals,0,4095,0,180) ;
  myservo.write(potvals) ;
}
void LinkageMotor() {
      potvals2 = myData.b;
      Serial.print("linkage num : ");
      Serial.println(myData.b);

      if (bool(digitalRead(frontLimit))== true){
        theta = thetaMax;
      }

      if (bool(digitalRead(backLimit))== true){
        theta = 0;
      }

      thetaDes = map(potvals2, 0, potMax, 0, thetaMax);
      Serial.println(thetaDes) ;
      //A6 CONTROL SECTION
      //CHANGE THIS SECTION FOR P AND PI CONTROL
      int e = (thetaDes - theta);
      int e_accum_max = 0.5;

      e_accum = e_accum + e;
      if (e_accum > e_accum_max) {
        e_accum = e_accum_max;
      }
      else if (e_accum < -e_accum_max) {
        e_accum = -e_accum_max;
      }

      D = Kp*e + Ki*e_accum;


      //END A6 CONTROL SECTION

      //Ensure that you don't go past the maximum possible command
      if (D > MAX_PWM_VOLTAGE) {
          D = MAX_PWM_VOLTAGE;
      }
      else if (D < -MAX_PWM_VOLTAGE) {
          D = -MAX_PWM_VOLTAGE;
      }

      //Map the D value to motor directionality
      //FLIP ENCODER PINS SO SPEED AND D HAVE SAME SIGN
```

```
    if (D > 0) {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, D);
    }
    else if (D < 0) {
        ledcWrite(ledChannel_1, -D);
        ledcWrite(ledChannel_2, LOW);
    }
    else {
        ledcWrite(ledChannel_1, LOW);
        ledcWrite(ledChannel_2, LOW);
    }
    //plotControlData();
}
```

Code 3: Auxiliary ESP32 code (ESP32 2 in the vehicle circuit diagram)

```
//Pin Definitions
#define RXD2 13 //Data receive pin for serial data transfer
#define TXD2 12 //Data transmist pin for serial data transfer
#define DRIVE 26 //Pin to read state of vehicle
#define TURN 25 //Pin to read state of vehicle

//Motor Pins
#define BIN_1 14
#define BIN_2 32
#define BIN_3 18
#define BIN_4 5



const int MAX_PWM_VOLTAGE = 255;
int incomingByte = 0;
int endmes = 97;
int i = 1;
int finalint = 0;
int thou = 0;
int hund = 0;
int tens = 0;
int ones = 0;
int D1 = 0;
int D2 = 0;
bool drive = false;
bool turn = false;


// defining the channels
const int ledChannel_1 = 1;
const int ledChannel_2 = 2;
const int ledChannel_3 = 3;
const int ledChannel_4 = 4;
const int freq = 5000 ;
```

```
const int resolution = 8;

void setup() {
  pinMode(TURN, INPUT);
  pinMode(DRIVE, INPUT);
  Serial.begin(115200);
  Serial2.begin(115200, SERIAL_8N1, RXD2, TXD2);
  Serial.println("Start");

  ledcSetup(ledChannel_1, freq, resolution);
  ledcSetup(ledChannel_2, freq, resolution);
  ledcSetup(ledChannel_3, freq, resolution);
  ledcSetup(ledChannel_4, freq, resolution);

  ledcAttachPin(BIN_1, ledChannel_1);
  ledcAttachPin(BIN_2, ledChannel_2);
  ledcAttachPin(BIN_3, ledChannel_3);
  ledcAttachPin(BIN_4, ledChannel_4);
}

void loop() {
  // Reading State Pins
  drive = bool(digitalRead(DRIVE));
  turn = bool(digitalRead(TURN));

  //Reading serial data and converting from bytes back to commanded motor integer
while (Serial2.available() > 0 && incomingByte != endmes) {
    // read the incoming byte:
    incomingByte = Serial2.read();

    if (i == 1){
      thou = 1000*(((char)incomingByte)-'0');
    }
    if (i == 2){
      hund = 100*(((char)incomingByte)-'0');
    }
    if (i == 3){
      tens = 10*(((char)incomingByte)-'0');
    }
    if (i == 4){
      ones = (((char)incomingByte)-'0');
      finalint = thou+hund+tens+ones;
    }
    i = i +1;
  }
  D1 = map(finalint, 1000, 1511, -MAX_PWM_VOLTAGE, MAX_PWM_VOLTAGE);
  Serial.println(finalint);
  incomingByte = 0;
  i = 1;
if (!drive){
  ledcWrite(ledChannel_1, LOW);
  ledcWrite(ledChannel_2, LOW);
```

```
delay(15);
  ledcWrite(ledChannel_3, LOW);
  ledcWrite(ledChannel_4, LOW);
}

// Sending PWM values to motor drivers based on the commanded D values
if (drive){
  if (!turn){
    D2 = D1;
    if (D1 > 0) {

    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, D1);
   }
  else if (D1 < 0) {
    ledcWrite(ledChannel_1, -D1);
    ledcWrite(ledChannel_2, LOW);

   }
  else {
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, LOW);
   }

   if (D2 > 0) {
          ledcWrite(ledChannel_3, LOW);
          ledcWrite(ledChannel_4, D2);
      }
      else if (D2 < 0) {
          ledcWrite(ledChannel_3, -D2);
          ledcWrite(ledChannel_4, LOW);
      }
      else {
          ledcWrite(ledChannel_3, LOW);
          ledcWrite(ledChannel_4, LOW);
      }
  }
  if (turn){
    D2 = D1;
    if (D1 > 0) {

    ledcWrite(ledChannel_1, D1);
    ledcWrite(ledChannel_2, LOW);

   }
  else if (D1 < 0) {
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, -D1);
   }
  else {
    ledcWrite(ledChannel_1, LOW);
    ledcWrite(ledChannel_2, LOW);
```

```
    }

    if (D2 > 0) {
            ledcWrite(ledChannel_3, LOW);
            ledcWrite(ledChannel_4, D2);
        }
        else if (D2 < 0) {
            ledcWrite(ledChannel_3, -D2);
            ledcWrite(ledChannel_4, LOW);
        }
        else {
            ledcWrite(ledChannel_3, LOW);
            ledcWrite(ledChannel_4, LOW);
        }
   }
}
}
```