# ME102B Final Report

*Team 20: Ping Pong Catcher*

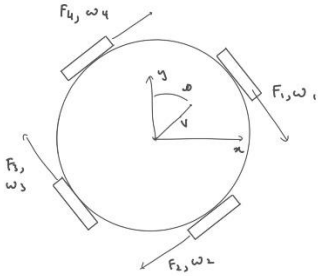
*Wong Rui Yue, Fang Yi Kuan, Ming Xiao Huo*

**Opportunity:** How might we make the retrieval of ping pong balls less tedious?

**High level strategy:** We wanted to create a robot able to catch ping pong balls as they fall after being hit by a player. We planned to use a camera to detect incoming balls and a chassis with omni directional wheels for quick movement along the ground plane, able to hit 2m/s within .5s and an IMU to correct for any rotation of the chassis.

Eventually we ditched the computer vision portion of the project and focused on tuning the controls and capabilities of the robot itself. Instead, we used a joystick to wirelessly control the chassis.

A user would input the desired speed and direction of the robot into 2 joysticks connected to an ESP32. The ESP32 would wirelessly communicate with another ESP32 on the robot. This ESP32 would communicate this information via UART to an Arduino Due on the robot. An IMU on the robot also takes in reading on the angle the robot is facing and communicates this with the Due via UART. The Due takes the desired speed and direction as well as the angle the robot is facing and converts it into motor instructions sent to motor drivers.



$$\omega_1 = \frac{-\mathrm{v} \times \mathbf{cos}(45+\theta)}{r_{wheel}} + f(\theta_{chassis})$$

$$\omega_2 = \frac{-\mathrm{v} \times \mathbf{cos}(45-\theta)}{r_{wheel}} + f(\theta_{chassis})$$

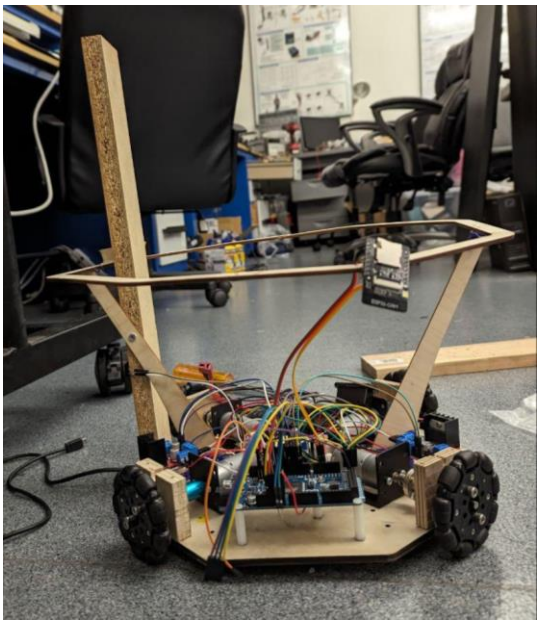
$$\omega_3 = \frac{\mathrm{v} \times \mathbf{cos}(45+\theta)}{r_{wheel}} + f(\theta_{chassis})$$

$$\omega_4 = \frac{\mathrm{v} \times \mathbf{cos}(45-\theta)}{r_{wheel}} + f(\theta_{chassis})$$

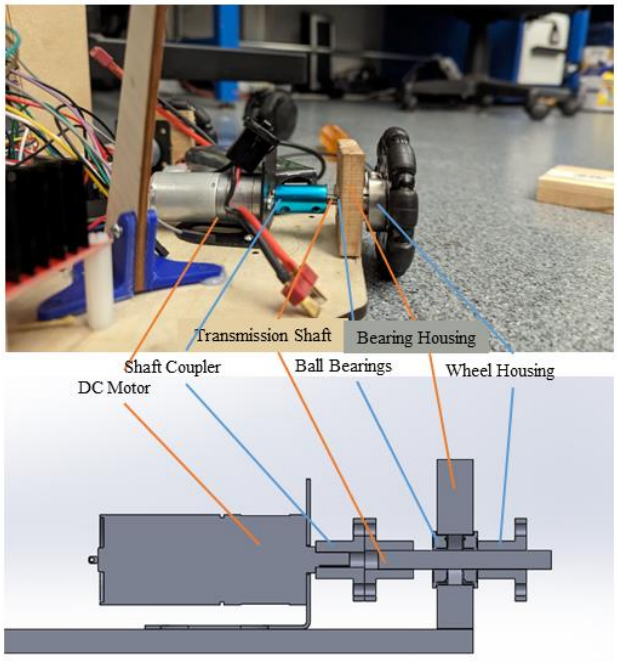
$f(\theta_{chassis})$ corrects the heading of the chassis

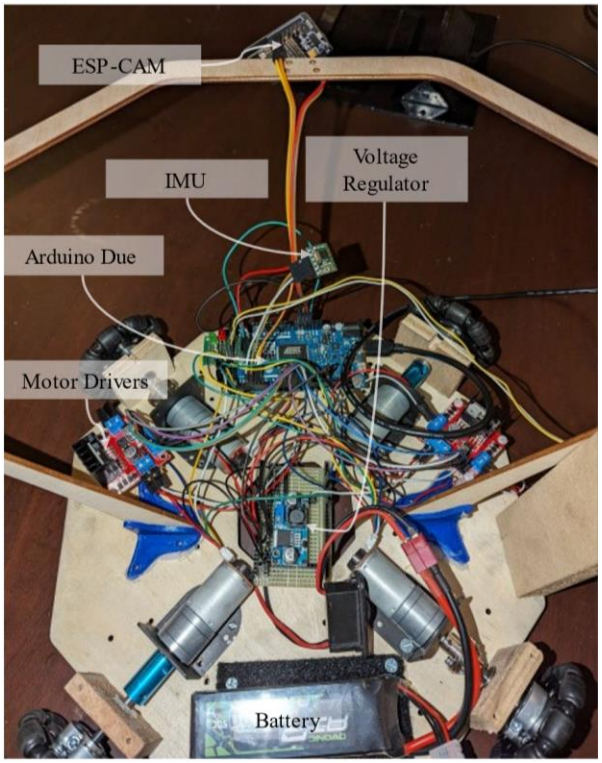
**Physical design:**

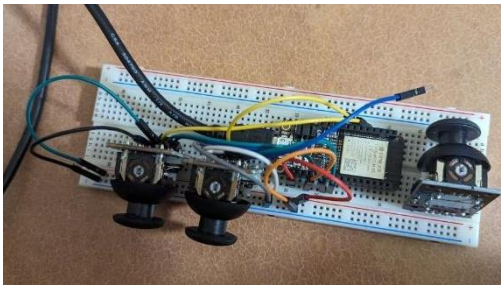
Side view



Wheel Transmission





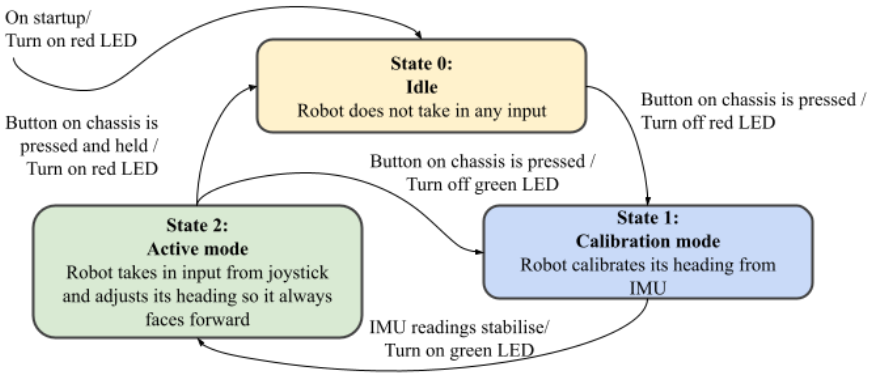
Joystick with ESP

**Function-critical decisions**

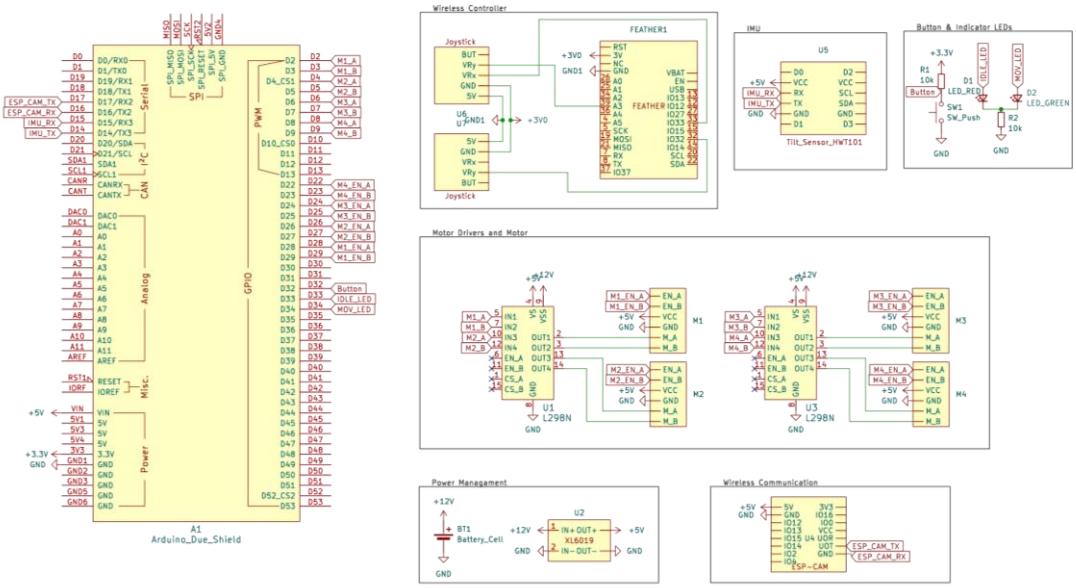
| ***Motor choice*** | ***Bearing forces*** |
|---|---|
| $D_{wheel} = 0.072m, V_{max} = 2ms^{-1}$<br>$\omega_{motor_{max}} = \frac{V_{max}}{D_{wheel}/2} = 83.33\, rads^{-1}$<br>$= 13.26\, rpm$<br>$m_{chassis} = 3kg, a_{max} = 4ms^{-2}$<br>$T_{motor_{max}} = \frac{m_{chassis} \cdot a_{max}}{2} \cdot \frac{D_{wheel}}{2}$<br>$= 0.216Nm = 2.201kgcm$<br>Motor chosen | Max static radial load: 346.3N<br>Max dynamic radial load: 916.7N<br>Source<br>$F_{static} = m_{chassis}g = 29.43N < F_{static_{max}}$<br>$F_{dynamic} = \sqrt{\left({F_{static}}^2 + \left(\frac{m_{chassis} \cdot a_{max}}{2}\right)^2\right)}$<br>$= 30.035N < F_{dynamic_{max}}$ |

**Electrical design**

***State Transition Diagram***



***Circuit Diagram***



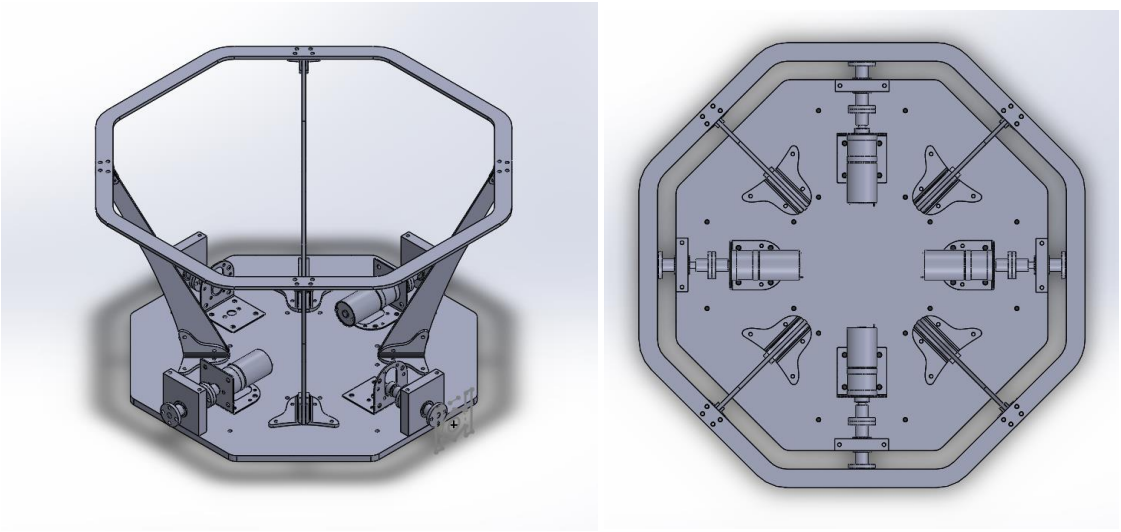

**Conclusion**

In building a robot with omnidirectional wheels, ensuring that all wheels touch the ground was crucial to the chassis behaving as intended. Our base board of the chassis is laser cut ¼" wood, and wood is not always level or square. This resulted in a lot of manual adjustments needed to be made to ensure all the wheels touch the ground. Additionally, if the ground the robot operates on is not perfectly flat, the robot also will not work as intended. As such we would recommend to other groups to use metal for the baseboard and build a suspension system into a similar robot.

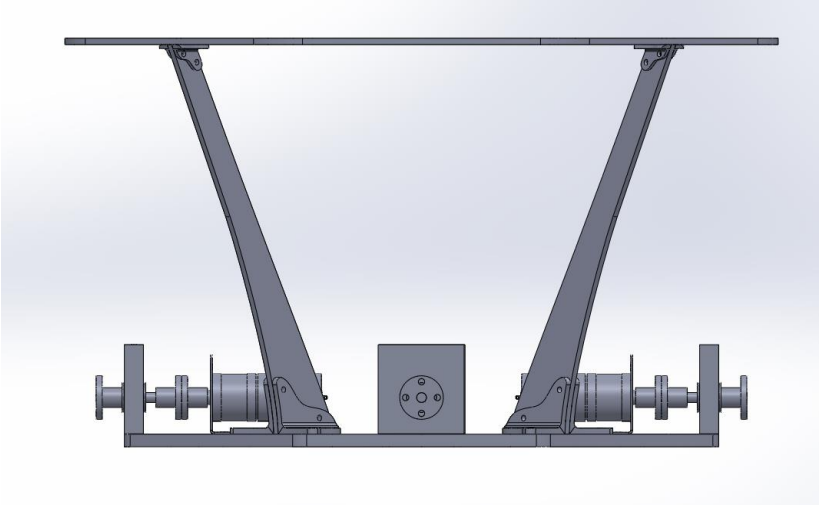
We also has a lot of difficulty tuning the PID control of all 4 motors. The PID of each motor needed to be tuned with the chassis moving on the ground, however we found that isolating the effects of tuning PID of each motor difficult as the performance of all the motors affected the movement of the chassis.

**Appendix**

***CAD***



Left: Orthogonal View, Right: Top View



Side view

Team 20

***BOM***

| | S/N | Item Name | Price | Quantity | Description | Subtotal | Source | |
|---|---|---|---|---|---|---|---|---|
| **Electronics** | 1 | 12V 3S Lipo Battery | $17.99 | 1 | 12V battery to power motors | $17.99 | link | |
| | 2 | LM2596 Buck Converter (2pcs) | $2.98 | 1 | Adjustable Voltage Regulator | $2.98 | link | |
| | 3 | L298N Motor Driver (4pcs) | $8.48 | 1 | Motor Driver | $8.48 | link | |
| | 4 | 12V DC Motors | $14.88 | 4 | DC motors | $59.52 | link | |
| | 5 | Arduino Due | $48.40 | 1 | Used to control motors using information from joystick an | $48.40 | link | |
| | 6 | ESP-CAM | $9.98 | 1 | Used to receive joystick information | $9.98 | link | |
| | 7 | HWT101 MEMS Tilt Angle Senso | $49.00 | 1 | IMU | $49.00 | link | |
| | 8 | Dual Axis Joystick | $1.26 | 2 | | $2.52 | link | |
| | 9 | LED | $0.00 | 2 | 1 red, 1 green (taken from lab) | $0.00 | link | |
| | 10 | Resistors | $0.00 | 3 | taken from lab kit | $0.00 | link | |
| | 11 | Adafruit Huzzah32 Feather | $0.00 | 1 | from lab kit, used to send joystick input | $0.00 | link | |
| **Chassis** | 1 | Laser cut bottom board | $3.98 | 1 | from Jacobs hall | $3.98 | link | |
| | 2 | Machined bearing holder | $1.25 | 4 | material cost, machined | $5.00 | link | |
| | 3 | Handle | $0.00 | 1 | scrapwood | $0.00 | link | |
| | 4 | Screws | $15.00 | 1 | from Home Depot | $15.00 | link | |
| | 5 | Motor Brackets (4pcs) | $10.95 | 1 | Machined to fit our motors | $10.95 | link | |
| | 6 | 5mm diameter 50mm shafts | $7.49 | 1 | | $7.49 | link | |
| | 7 | 4mm to 5mm Shaft Couplers | $6.49 | 4 | | $25.96 | link | |
| | 8 | 72mm Omniwheels | $8.99 | 4 | | $35.96 | link | |
| | 9 | 5mm Bore Sonic Hub | $6.99 | 4 | | $27.96 | link | |
| | | | | | **Total** | $331.17 | | |
| | | | | | | | | |

## *Arduino Due Code*

```
#include <Encoder.h>
#include <DueTimer.h>
#include <REG.h>
#include <wit_c_sdk.h>
// Defining stuff
  // Defining stuff for IMU
    #define ACC_UPDATE    0x01
    #define GYRO_UPDATE   0x02
    #define ANGLE_UPDATE  0x04
    #define MAG_UPDATE    0x08
    #define READ_UPDATE   0x80
    // Global variables for IMU
    static volatile char s_cDataUpdate = 0, s_cCmd = 0xff;
    static void CmdProcess(void);
    static void AutoScanSensor(void);
    static void SensorUartSend(uint8_t *p_data, uint32_t uiSize);
    static void SensorDataUpdata(uint32_t uiReg, uint32_t uiRegNum);
    static void Delayms(uint16_t ucMs);
    const uint32_t c_uiBaud[8] = {0,4800, 9600, 19200, 38400, 57600, 115200, 230400};
    float fAcc[3], fGyro[3], fAngle[3];
      // For IMU callibration
      float IMUinit = 0;
      float IMU_cal_range = 0.1;
      const int IMU_cal_length = 10;
      float IMUinitArr[IMU_cal_length];
      int count = 0;
  // Define the pins for the encoder
    #define encoderPin1A 28
    #define encoderPin1B 29
    #define encoderPin2A 26
    #define encoderPin2B 27
    #define encoderPin3A 24
    #define encoderPin3B 25
    #define encoderPin4A 22
    #define encoderPin4B 23
  // Define the pins for the motor
    #define M1_1 2
    #define M1_2 3
    #define M2_1 4
    #define M2_2 5
    #define M3_1 6
    #define M3_2 7
    #define M4_1 8
    #define M4_2 9
  // define states, buttons and LED pins
    #define button 32
    #define MOVLED 33
    #define IDLELED 34

    // define states
      #define IDLE 0
      #define CAL 1
      #define MOV 2
```

```
// Variables for state transition machine
  volatile bool buttonPressed = false;
  volatile bool buttonHold = false;
  volatile bool debounceFlag = true;
  int debouncePeriod = 200000;
  int holdPeriod = 2000000;
  int state = 0;
// To calibrate joystick
    #define LEFTX_CNT 122
    #define LEFTY_CNT 116
    #define RIGHTX_CNT 120

// Variables for chassis control
  float speed;
  float angle = 0;
// setting PWM properties ---------------------------
  const int MAX_PWM_VOLTAGE = 255;
  const int NOM_PWM_VOLTAGE = 220;

//Setup encoder counts ---------------------------
  const int roundCount = 420; //encoder counts per round
  volatile int count1 = 0; volatile int count2 = 0;
  volatile int count3 = 0; volatile int count4 = 0; // encoder count

//Setup encoder interrupt variables ---------------------------
  volatile bool encoderInt = false;    // check timer interrupt 1
  int encoderPeriod = 50000; //in micro seconds


//setting motor speed variables
  float currentSpeed1 = 0; float currentSpeed2 = 0; float currentSpeed3 = 0; float currentSpeed4 = 0;
  // float omegaDes1 = 0; float omegaDes2 = 0; float omegaDes3 = 0; float omegaDes4 = 0;
  int omegaMax = 10;   // in RPS

////setting PID values
  float Kp = 5;   // TUNE THESE VALUES TO CHANGE CONTROLLER PERFORMANCE
  float Ki= 5;
  int integralmax = 50;
  float errorsum[4]={0.0};
  int IMax = 0;
// Setup serial buffer and serial timer interrupt
  int UARTPeriod = 50000; //in micro seconds
  String UART_buffer;
// Variables for joystick readings
  int LeftX, LeftY, RightX, RightY;\
  String LeftXstr, LeftYstr, RightXstr, RightYstr;
  String RightYdebug;

// Create an Encoder object
  Encoder Encoder1(encoderPin1A, encoderPin1B);
  Encoder Encoder2(encoderPin2A, encoderPin2B);
  Encoder Encoder3(encoderPin3A, encoderPin3B);
  Encoder Encoder4(encoderPin4A, encoderPin4B);
```

```
  //Setting up timer for encoder readings
void onEncoderTimer(){
  count1 = Encoder1.read(); count2 = Encoder2.read();
  count3 = Encoder3.read(); count4 = Encoder4.read();
  Encoder1.write(0); Encoder2.write(0); Encoder3.write(0); Encoder4.write(0);
  encoderInt = true;
}

void setup() {
  // Start serial port
  Serial.begin(9600); //this port comunicates with the computer for debugging
  Serial2.begin(9600); //this port communicates with the ESP-CAM
  // Setting up IMU
    WitInit(WIT_PROTOCOL_NORMAL, 0x50);
    WitSerialWriteRegister(SensorUartSend);
    WitRegisterCallBack(SensorDataUpdata);
    WitDelayMsRegister(Delayms);
    AutoScanSensor();
  // Set pinmode and write motor pins to low
    pinMode(M1_1, OUTPUT); pinMode(M1_2, OUTPUT);
    pinMode(M2_1, OUTPUT); pinMode(M2_2, OUTPUT);
    pinMode(M3_1, OUTPUT); pinMode(M3_2, OUTPUT);
    pinMode(M4_1, OUTPUT); pinMode(M4_2, OUTPUT);
    digitalWrite(M1_1, LOW); digitalWrite(M1_2, LOW);
    digitalWrite(M2_1, LOW); digitalWrite(M2_2, LOW);
    digitalWrite(M3_1, LOW); digitalWrite(M3_2, LOW);
    digitalWrite(M4_1, LOW); digitalWrite(M4_2, LOW);

  // Set up button, interrupts and timers
      pinMode(button,INPUT_PULLUP);
      attachInterrupt(digitalPinToInterrupt(button), onButtonPress,FALLING);
      Timer2.attachInterrupt(Timer2ISR).setPeriod(debouncePeriod);
      Timer4.attachInterrupt(Timer4ISR).setPeriod(holdPeriod);
      // Setting up indicator LEDs
        pinMode(IDLELED, OUTPUT); pinMode(MOVLED, OUTPUT);
        digitalWrite(IDLELED, HIGH);

  //initialising timers
    Timer3.attachInterrupt(onEncoderTimer);
    Timer3.start(encoderPeriod);

}
```

```
void loop() {
  debounceLoop();
  switch(state){
    // robot enters into IDLE upon startup, goes into CAL when button is held down
    case IDLE:
      if(buttonPressed == true){buttonPressLoop();}
      if(buttonHold == true){Timer4.stop(); buttonHold = false; state = CAL; digitalWrite(IDLELED, LOW);}
    break;
    // robot waits until readings from IMU is stable and then takes an average of the intial values to get its intial heading
    // robot then goes into moving state
    case CAL:
      IMUcali();
      digitalWrite(MOVLED,HIGH);
      state = MOV;
    break;
    // robot listens to instructions from the joystick controller transmitted via wifi, received by the esp-cam
    // robot also corrects its heading if it is rotated
    case MOV:
      if(buttonPressed == true)
        {buttonPressLoop(); state = CAL;digitalWrite(MOVLED,LOW); stopMotor();}
      else if (buttonHold == true)
      {Timer4.stop(); buttonHold = false; state = IDLE;
      digitalWrite(MOVLED,LOW); digitalWrite(IDLELED, HIGH); stopMotor(); }
      else{
          SerialUpdate();
          getSpeedAngle();
          countToRPS();
          IMUloop();
          chassis(angle, speed, fAngle[2]);
      }
    break;}
}
```

```
  //This funcitons stops all the motors, used when changing states from the moving state to other states
void stopMotor(){
  digitalWrite(M1_1, LOW); digitalWrite(M1_2, LOW);
  digitalWrite(M2_1, LOW); digitalWrite(M2_2, LOW);
  digitalWrite(M3_1, LOW); digitalWrite(M3_2, LOW);
  digitalWrite(M4_1, LOW); digitalWrite(M4_2, LOW);
}

// Replaces arduino map function to return floats
float map(float x, float map_min, float map_max, float out_min, float out_max){
  float out = (x - map_min)/(map_max - map_min) * (out_max - out_min) + out_min;
  return out;
}

// Functions called for button and timer interrupts
  // This function is called when button intterupt is triggered
    void onButtonPress(){
      if (debounceFlag == true){
      buttonPressed = true;}}
  // This function is called when timer2 intterupt is triggered
  // This timer is for debouncing
    void Timer2ISR(){
      if(digitalRead(button) == HIGH){debounceFlag = true;}
    }
  // This function is called when timer4 intterupt is triggered
  // This timer is for detecting when the button is held down
    void Timer4ISR(){
      if(digitalRead(button) == LOW){buttonHold = true;}
    }
```

```
  // Service functions
    // This function is called in the main loop when the button is clicked
    void buttonPressLoop(){
    buttonPressed = false; debounceFlag = false;
    Timer2.start(); Timer4.start();}

    // This function services the debounce timer
    void debounceLoop(){
      if(debounceFlag == true){Timer2.stop();}}
// Converts joystick inputs to speed and angle of chassis
void getSpeedAngle(){

  float  x_vet, y_vet, r_vet; //local variables
  //turns joystick input of 0-255 to -127 to 127
  x_vet = LEFTX_CNT - LeftX;
  y_vet = LEFTY_CNT - LeftY;
  r_vet = RIGHTX_CNT - RightY;

  //turns vectors into decimals
  x_vet = x_vet / 127;
  y_vet = y_vet / 127;

  //right joystick controls speed, this scales the vector accordingly
  r_vet = r_vet / 127 *5;
  speed = r_vet ;
  angle = 0;

  // Get the desired heading of the chassis from the left joystick input
  if (abs(x_vet) < 0.03 && y_vet > 0)
  {
    angle = 0;
  }
  else if (abs(x_vet) < 0.03 && y_vet < 0)
  {
    angle = 180;
  }
  else if (abs(y_vet) < 0.03 && x_vet > 0)
  {
    angle = 90;
  }
  else if (abs(y_vet) < 0.03 && x_vet < 0)
  {
    angle = 270;
  }
  else if (x_vet > 0 && y_vet > 0){
      angle =atan(x_vet/y_vet) / 3.14 * 180;
  }
  else if (x_vet > 0 && y_vet < 0){
      angle =atan(x_vet/y_vet) / 3.14 * 180;
      angle = angle + 180;
  }
  else if (x_vet < 0 && y_vet > 0){
      angle =atan(x_vet/y_vet) / 3.14 * 180;
      angle = angle + 360;
  }
```

```
  else if (x_vet < 0 && y_vet < 0){
      angle =atan(x_vet/y_vet) / 3.14 * 180;
      angle = angle + 180;
  }
}


// returns PID value
float PID(int motor, float Kp, float Ki, float des, float act){
  float error = des-act;
  float pwm = 0.0;

  errorsum[motor-1] += error;
  pwm = float(Ki) * errorsum[motor-1] + float(Kp) * error;

  if (pwm > NOM_PWM_VOLTAGE){pwm = NOM_PWM_VOLTAGE;}
  else if (pwm < -NOM_PWM_VOLTAGE){pwm = -NOM_PWM_VOLTAGE;}
  if (errorsum[motor-1] > integralmax)
  {
    errorsum[motor-1] = integralmax;
  }
  else if (errorsum[motor-1] < -1* integralmax)
  {
    errorsum[motor-1] = -1*integralmax;
  }
  return pwm;

  }
//convert encoder counts to RPS
void countToRPS(){
  currentSpeed1 = float(count1)/float(roundCount)*1000000.0f/float(encoderPeriod);
  currentSpeed2 = float(count2)/float(roundCount)*1000000/float(encoderPeriod);
  currentSpeed3 = float(count3)/float(roundCount)*1000000.0f/float(encoderPeriod);
  currentSpeed4 = float(count4)/float(roundCount)*1000000/float(encoderPeriod);
}
```

```
// Sends motor driver instructions
void motor(int motor_num, float speed){
  switch(motor_num){
    case 1:
      if (speed >= 0){
        analogWrite(M1_1, speed);
        analogWrite(M1_2, 0);}
      else
      {analogWrite(M1_1, 0);
        analogWrite(M1_2, -1*speed);}
    break;
    case 2:
      if (speed >= 0){
        analogWrite(M2_2, 0);
        analogWrite(M2_1, speed);}
      else{
        analogWrite(M2_1, 0);
        analogWrite(M2_2, -1*speed);
      }
    break;
    case 3:
      if (speed >= 0){
        analogWrite(M3_2, 0);
        analogWrite(M3_1, speed);}
      else{
        analogWrite(M3_1, 0);
        analogWrite(M3_2, -1*speed);}
    break;
    case 4:
      if (speed >= 0){
        analogWrite(M4_2, 0);
        analogWrite(M4_1, speed);}
      else{
        analogWrite(M4_1, 0);
        analogWrite(M4_2, -1*speed);}
    break;}}
```

```
// Converts speed and angle of chassis into motor instructions
// IMU correction is here
void chassis(float angle, float speed, float corr){
  float motorspd = 0.0;
  float radangle = 0.0;
  radangle = (angle + 45) /180 *3.14; //convert angle to radians
  // This part of the code converts readings from the IMU to instructions to motors on how to correct it
  //get heading error, subtract off initial heading of robot
  float corrPWM; corr = corr - IMUinit;
  //map the heading error to rpm of motors
  if(corr>0){corrPWM = map(corr,0,90,0,3); } else {corrPWM = map(corr,-90,0,-3,-0);}

  // calculate desired speed of each motor
  motorspd = -1 * cos(radangle) * speed + corrPWM;
    motor(1, PID(1,Kp,Ki,motorspd,currentSpeed1));
  radangle = (angle + 45) /180 *3.14;
  motorspd =cos(radangle) * speed + corrPWM;
    motor(3, PID(3,Kp,Ki,motorspd,currentSpeed3));
  radangle = (45 - angle) /180 *3.14;
  motorspd = cos(radangle) * speed + corrPWM;
  motor(4, PID(4,Kp,Ki,motorspd,currentSpeed4));
  radangle = (45 - angle) /180 *3.14;
  motorspd = -1 * cos(radangle) * speed + corrPWM;
    motor(2, PID(2,Kp,Ki,motorspd,currentSpeed2));

}
```

```
// Gets joystick input from esp32
// Get communicaiton from the serial port until indicators, seperate
// and store them into appropriate functions
// Reading from serial port contains quite a bit of garbage from loss
void SerialUpdate(){
  if (Serial2.available()>0) {
      UART_buffer = Serial2.readStringUntil('\n');
      UART_buffer = "";
      UART_buffer = Serial2.readStringUntil('\n');
      int inx = UART_buffer.indexOf(',');
      String buf = "";
      buf = UART_buffer.substring(0,inx);
      // Reading from serial port contains quite a bit of garbage from loss
      // This if statement filters the garbae out, similiar is donw for other variables
      if (buf.length() ==5 && buf.charAt(0) == '0' && buf.charAt(4) == '0'){
      LeftX = buf.substring(1,4).toInt();
      LeftXstr = buf.substring(1,4);}
      UART_buffer = UART_buffer.substring(inx+1);
      inx = UART_buffer.indexOf(',');
      buf = "";
      buf = UART_buffer.substring(0,inx);
      if (buf.length() ==5 && buf.charAt(0) == '1' && buf.charAt(4) == '1'){
      LeftY = buf.substring(1,4).toInt();
      LeftYstr = buf.substring(1,4);}
      UART_buffer = UART_buffer.substring(inx+1);
      inx = UART_buffer.indexOf(',');
      buf = "";
      buf = UART_buffer.substring(0,inx);
      RightX = buf.toInt();
      RightXstr = buf;
      UART_buffer = UART_buffer.substring(inx+1);
      inx = UART_buffer.indexOf(',');
      buf = "";
      buf = UART_buffer.substring(0,inx);
      RightYdebug = buf;
      if (buf.length() ==6 && buf.charAt(0) == '3' && buf.charAt(4) == '3'){
      RightY = buf.substring(1,4).toInt();
      RightYstr = buf.substring(1,4);}
      UART_buffer = UART_buffer.substring(inx+1);

  }
```

```
411  // gets output from IMU
412  void IMUloop(){
413    while (Serial1.available())
414      {WitSerialDataIn(Serial1.read());}
415      while (Serial.available())
416      {CopeCmdData(Serial.read());}
417      CmdProcess();
418      if(s_cDataUpdate)
419      {for(int i = 0; i < 3; i++)
420        { fAcc[i] = sReg[AX+i] / 32768.0f * 16.0f;
421          fGyro[i] = sReg[GX+i] / 32768.0f * 2000.0f;
422          fAngle[i] = sReg[Roll+i] / 32768.0f * 180.0f;}
423        s_cDataUpdate = 0;}
```

```
424  //Takes IMU values and stors them into array
425  void IMUcaliarr(){
426    while (Serial1.available())
427      {WitSerialDataIn(Serial1.read());}
428      while (Serial.available())
429      {CopeCmdData(Serial.read());}
430      CmdProcess();
431      if(s_cDataUpdate)
432      {for(int i = 0; i < 3; i++)
433        { fAcc[i] = sReg[AX+i] / 32768.0f * 16.0f;
434          fGyro[i] = sReg[GX+i] / 32768.0f * 2000.0f;
435          fAngle[i] = sReg[Roll+i] / 32768.0f * 180.0f;}
436        if(s_cDataUpdate & ACC_UPDATE)
437        {s_cDataUpdate &= ~ACC_UPDATE;}
438        if(s_cDataUpdate & GYRO_UPDATE)
439        {s_cDataUpdate &= ~GYRO_UPDATE;}
440        if(s_cDataUpdate & ANGLE_UPDATE)
441        { Serial.print("angle:"); Serial.print(fAngle[2], 3); Serial.print("\r\n");
442          if (fAngle[2]< IMUinit + IMU_cal_range && fAngle[2]> IMUinit - IMU_cal_range){
443            IMUinitArr[count] = fAngle[2];count +=1;
444        } else {count = 0;IMUinit = fAngle[2];}
445          s_cDataUpdate &= ~ANGLE_UPDATE;}
446        if(s_cDataUpdate & MAG_UPDATE)
447        {s_cDataUpdate &= ~MAG_UPDATE;}
448        s_cDataUpdate = 0;
449      }
450  }
```

```
// This function takes IMU values into an array and finds the average
void IMUcali(){
      count = 0;
    while(count<IMU_cal_length){
      IMUcaliarr();}
    float IMU_sum;
    for (int i = 0; i<IMU_cal_length; i++){
      IMU_sum += IMUinitArr[i]/IMU_cal_length;
      Serial.print(IMU_sum);Serial.print(",");Serial.println(IMUinitArr[i]);}

    IMUinit = IMU_sum;
    Serial.print("IMU: "); Serial.println(IMUinit);
    delay(1000);
}
// Functions for IMU setup
  void CopeCmdData(unsigned char ucData){
    static unsigned char s_ucData[50], s_ucRxCnt = 0;

    s_ucData[s_ucRxCnt++] = ucData;
    if(s_ucRxCnt<3)return;                    //Less than three data returned
    if(s_ucRxCnt >= 50) s_ucRxCnt = 0;
    if(s_ucRxCnt >= 3)
    {
      if((s_ucData[1] == '\r') && (s_ucData[2] == '\n'))
      {
        s_cCmd = s_ucData[0];
        memset(s_ucData,0,50);
        s_ucRxCnt = 0;
      }
      else
      {
        s_ucData[0] = s_ucData[1];
        s_ucData[1] = s_ucData[2];
        s_ucRxCnt = 2;

      }
    }}
```

```
static void ShowHelp(void){
  Serial.print("\r\n************************   WIT_SDK_DEMO ************************");
  Serial.print("\r\n************************          HELP           ************************\r\n");
  Serial.print("UART SEND:a\\r\\n   Acceleration calibration.\r\n");
  Serial.print("UART SEND:m\\r\\n   Magnetic field calibration,After calibration send:   e\\r\\n   to indicate the end\r\n");
  Serial.print("UART SEND:U\\r\\n   Bandwidth increase.\r\n");
  Serial.print("UART SEND:u\\r\\n   Bandwidth reduction.\r\n");
  Serial.print("UART SEND:B\\r\\n   Baud rate increased to 115200.\r\n");
  Serial.print("UART SEND:b\\r\\n   Baud rate reduction to 9600.\r\n");
  Serial.print("UART SEND:R\\r\\n   The return rate increases to 10Hz.\r\n");
  Serial.print("UART SEND:r\\r\\n   The return rate reduction to 1Hz.\r\n");
  Serial.print("UART SEND:C\\r\\n   Basic return content: acceleration, angular velocity, angle, magnetic field.\r\n");
  Serial.print("UART SEND:c\\r\\n   Return content: acceleration.\r\n");
  Serial.print("UART SEND:h\\r\\n   help.\r\n");
  Serial.print("******************************************************************************\r\n");}

static void CmdProcess(void){
  switch(s_cCmd)
  {
    case 'a': if(WitStartAccCali() != WIT_HAL_OK) Serial.print("\r\nSet AccCali Error\r\n");
      break;
    case 'm': if(WitStartMagCali() != WIT_HAL_OK) Serial.print("\r\nSet MagCali Error\r\n");
      break;
    case 'e': if(WitStopMagCali() != WIT_HAL_OK) Serial.print("\r\nSet MagCali Error\r\n");
      break;
    case 'u': if(WitSetBandwidth(BANDWIDTH_5HZ) != WIT_HAL_OK) Serial.print("\r\nSet Bandwidth Error\r\n");
      break;
    case 'U': if(WitSetBandwidth(BANDWIDTH_256HZ) != WIT_HAL_OK) Serial.print("\r\nSet Bandwidth Error\r\n");
      break;
    case 'B': if(WitSetUartBaud(WIT_BAUD_115200) != WIT_HAL_OK) Serial.print("\r\nSet Baud Error\r\n");
              else
              {
                Serial1.begin(c_uiBaud[WIT_BAUD_115200]);
                Serial.print(" 115200 Baud rate modified successfully\r\n");
              }
      break;
    case 'b': if(WitSetUartBaud(WIT_BAUD_9600) != WIT_HAL_OK) Serial.print("\r\nSet Baud Error\r\n");
              else
              {
                Serial1.begin(c_uiBaud[WIT_BAUD_9600]);
                Serial.print(" 9600 Baud rate modified successfully\r\n");
              }
      break;
    case 'r': if(WitSetOutputRate(RRATE_1HZ) != WIT_HAL_OK)  Serial.print("\r\nSet Baud Error\r\n");
              else Serial.print("\r\nSet Baud Success\r\n");
      break;
    case 'R': if(WitSetOutputRate(RRATE_10HZ) != WIT_HAL_OK) Serial.print("\r\nSet Baud Error\r\n");
              else Serial.print("\r\nSet Baud Success\r\n");
      break;
    case 'C': if(WitSetContent(RSW_ACC|RSW_GYRO|RSW_ANGLE|RSW_MAG) != WIT_HAL_OK) Serial.print("\r\nSet RSW Error\r\n");
      break;
    case 'c': if(WitSetContent(RSW_ACC) != WIT_HAL_OK) Serial.print("\r\nSet RSW Error\r\n");
      break;
```

```
      case 'c': if(WitSetContent(RSW_ACC) != WIT_HAL_OK) Serial.print("\r\nSet RSW Error\r\n");
        break;
      case 'h': ShowHelp();
        break;
      default :break;
    }
    s_cCmd = 0xff;}
  static void SensorUartSend(uint8_t *p_data, uint32_t uiSize){
    Serial1.write(p_data, uiSize);
    Serial1.flush();}
  static void Delayms(uint16_t ucMs){
    delay(ucMs);}
  static void SensorDataUpdata(uint32_t uiReg, uint32_t uiRegNum){
    int i;
      for(i = 0; i < uiRegNum; i++)
      {
          switch(uiReg)
          {
              case AZ:
          s_cDataUpdate |= ACC_UPDATE;
              break;
              case GZ:
          s_cDataUpdate |= GYRO_UPDATE;
              break;
              case HZ:
          s_cDataUpdate |= MAG_UPDATE;
              break;
              case Yaw:
          s_cDataUpdate |= ANGLE_UPDATE;
              break;
              default:
          s_cDataUpdate |= READ_UPDATE;
        break;
          }
      uiReg++;
      }}

  static void AutoScanSensor(void){
    Serial1.begin(115200);
    Serial1.flush();}
```

***ESP-CAM code***

```
#define STASSID "ESP32_WiFi"
#define STAPSK "12345678"
#include <WiFi.h>
#include <WiFiUdp.h>

unsigned int localPort =8080;
unsigned int remotePort=8080;
char incomingPacket[537];
char A;
WiFiUDP Udp;

void setup(){
  Serial.begin(9600);
  WiFi.mode(WIFI_STA);
  WiFi.begin(STASSID, STAPSK);
  while (WiFi.status() != WL_CONNECTED)
  {
  Serial.print(".");
  delay(500);
  }
  delay(1000);
  Serial.println();
  Serial.print("Connected! IP address: ");
  Serial.println(WiFi.localIP());
  Serial.printf("UDP server on port  ", localPort);
  Udp.begin(localPort);
}

void loop(){
  int packetSize = Udp.parsePacket();
  if (packetSize)
  {
  int len = Udp.read(incomingPacket, 536);
  if (len > 0)
  {
  incomingPacket[len] = 0;
  String COM=incomingPacket;
    Serial.println(COM);
  }
  }

}
```

***Joystick ESP code***

```
#include <WiFi.h>
#include <WiFiUDP.h>



#define right_X 34
#define right_Y 32
#define left_X 39
#define left_Y 33
#define right_but 19

const char AP_NameChar[] = "ESP32_WiFi";
const char WiFiAPPSK[] = "12345678";
char payload[10];
unsigned int localPort =8080;
unsigned int remotePort=8080;
char incomingPacket[537];
char A;
WiFiUDP Udp;
volatile int x_value;
volatile int y_value;
volatile int key_value;

void setup(){
  Serial.begin(9600);
  WiFi.mode(WIFI_AP);
  WiFi.softAP(AP_NameChar, WiFiAPPSK);
  // Udp.begin(localPort);
  Serial.println();
  Serial.println("Started ap. Local ip: " + WiFi.localIP().toString());
  pinMode(right_Y, INPUT); pinMode(right_X, INPUT); pinMode(right_but, INPUT);
  pinMode(left_Y, INPUT); pinMode(left_X, INPUT);
}
int mappedLeftX; int mappedRightX; int mappedLeftY; int mappedRightY;
```

Team 20

```
void loop(){
  int leftx = analogRead(left_X);int rightx = analogRead(right_X);
  mappedLeftX = mapY(leftx, 2048); mappedRightX = mapY(rightx,2048);
  mappedLeftY = mapY(analogRead(left_Y),2048); mappedRightY = mapY(analogRead(right_Y),2048);
  Udp.beginPacket("192.168.4.255",remotePort);
  Udp.print(leadingZero(mappedLeftX,0) + String(",") + leadingZero(mappedLeftY,1) + String(",") + leadingZero(mappedRightX,2) + String(",") + leadingZero(mappedRightY,3));
  Udp.endPacket();
  Serial.print(leadingZero(mappedLeftX,0));Serial.print('\t');Serial.print(mappedLeftY, DEC);Serial.print(mappedRightY, DEC);
  Serial.print("\n");
  delay(10);

}
String leadingZero(int x, int index){
  String ret = "";
  if (x < 10){ ret = String(String(index)+String("00")+String(x)+ String(index));}
  else if (x < 100){ ret = String(String(index)+String("0")+String(x)+ String(index));}
  else{ret = String(index)+String(x)+String(index);}
  return ret;
}
int mapY(int reading, int middle){
  int ret = 0;
  if (reading > middle){
    ret = map(reading, middle,4095,127,255);
  } else {ret = map(reading, 0,middle,0,127);}
  return ret;}

```