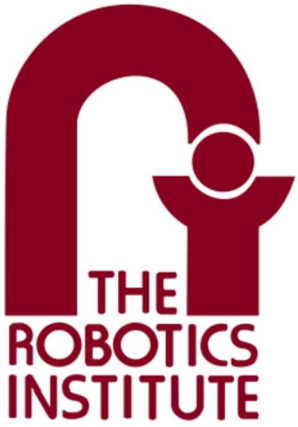

Individual Lab Report

MRSD Project Course I

Spring 2023

Shreyas Jha

Team E: OuterSense

Teammates: Atharv Pulapaka, Dhanesh Pamnani, Jash Shah, Ronit Hire

ILR01

Feb. 8, 2023

Contents

Individual Lab Report

Individual Progress

Sensors and Motor Lab

I worked on interfacing the potentiometer and stepper motor with a microcontroller board Arduino Mega 2560 (Figure 1 depicts the setup). The potentiometer is an analog sensor that was used as an input device in the setup to control the motion of the stepper motor. The potentiometer has 3 pins which were connected to +5V, GND, and an analog input pin on the microcontroller. By turning the knob on the potentiometer, the effective resistance through the potentiometer varied, resulting in a change in the voltage input provided to the microcontroller. Arduino Mega can accept up to 5V of voltage input on the analog pin, and this is read as an 8-bit value, effectively ranging from 0 to 1023. Based on this input, the change in the relative position of the potentiometer was calculated and sent as a command to the stepper motor. The stepper motor was a NEMA17 variant and was powered by a DRV8825 motor driver. The motor driver received commands from the microcontroller to move one step at a time in a particular direction via the step and direction pin outputs. The step length of the stepper motor could be controlled using the M0, M1, and M2 pins on the motor driver, and by default it was set at 1.8 degrees. The code is available in Appendix A.

Figure 1. Stepper motor and potentiometer interfaced with Arduino Mega

I also worked on integrating the individual pieces of code snippets written by all team members into one single executable code to be written on the microcontroller. This involved a state machine to switch between the sensor-based motor control and the GUI input-based motor control modes. In the sensor mode, all the sensor readings

(potentiometer, ultrasonic distance sensor, IR sensor, and temperature sensor) were continuously read and published to the GUI interface. If the distance measured by the IR sensor was less than 35cm and the temperature was below 80F, the servo motor would continuously sweep between 0 and 90 degrees. If the distance measured by the ultrasonic sensor was less than 7cm the stepper motor would spin continuously with step lengths of 1.8 degrees. Finally, the speed and direction of the DC motor were controlled by mapping the potentiometer reading to a corresponding RPM and direction of rotation for the DC motor. In the GUI mode, specific commands from the GUI were written to a data structure containing all the required parameters to select and drive the motors, using which appropriate methods to control different motors were invoked. The sensor readings were also continuously streamed to the GUI. The integrated code is available in Appendix B. The entire team setup for the sensors and motor control lab demonstration is shown in Figure 2.

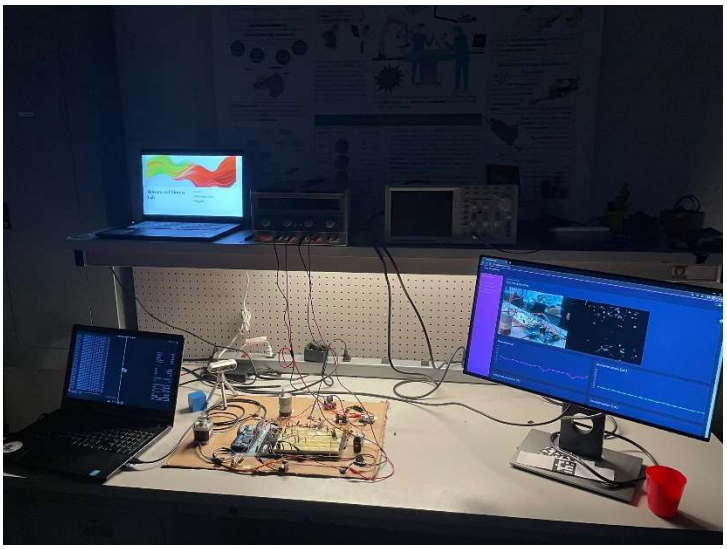

Figure 2. Team presentation setup for the sensors and motor control lab

(Picture credits – Atharv Pulapaka)

Capstone Project

I have primarily worked on the electronic hardware and embedded system for the RC Car. The first step was to study the datasheets of all the components to identify the interfacing opportunities (power ratings, baud rates, and communication protocols) of all the components. I studied the batteries and their chargers (originally purchased NiMh battery, 3S LiPos from the inventory, NiMh charger, and iMax B6AC charger), IMUs (UM7 orientation sensor, and Razor 9DOF IMU), microcontroller boards (Arduino Mega, Arduino Uno, Teensy 3.2, Teensy 3.6, Teensy 4.0, Raspberry Pi Pico WH, and Raspberry Pi 4B), motor (Velieon BLDC motor) and electronic speed controllers (VESC

and JMotors ESC), original servo motor on the car for steering control, hollow shaft quadrature encoders, camera ICs to serve as an optical flow based velocity sensor, ESP Wifi modules and real-time clock ICs.

I have completed the rudimentary control of the BLDC motor via the Arduino Mega, control the servo motor, and can read raw or processed data from the orientation sensor. Additionally, I have also converted all batteries, chargers, and ESCs to have appropriate male/female versions of the XT60 connectors.

Challenges

Multiple serial communication on Arduino Mega with UM7 sensor: Reading data from the orientation sensor via a different hardware serial fails when also performing other serial I/O tasks on the Arduino Mega. I tried debugging this in many ways by using different baud rates, hardware and software serials, microcontrollers, etc. Potential solutions could be to monitor and control the serial buffer or use a different supported communication protocol.

Configuring VESC for our BLDC motor and interfacing it with the microcontroller: The VESC is supposed to be a plug-and-play device with low-level motor control implemented, however, I have been struggling to configure it with the suggested GUI or reading sensor data from it on the microcontroller. It is likely that the VESC is damaged or faulty and we plan to contact the manufacturer for support.

Teamwork

Atharv Pulapaka: For the sensors and motor control lab, Atharv worked on the temperature sensor and a moving average filter for it, the servo motor, and the electrical wiring for the integrated system. With respect to the capstone project, Atharv has been primarily working on Simulink to develop initial simulations of our control architecture by building upon existing examples for lane detection and tracking. Atharv set up the electronic wiring for the sensors and motors lab and helped in debugging the integrated code.

Dhanesh Pamnani: For the sensors and motor control lab, Dhanesh worked on the IR proximity sensor, calculated it's transfer function, and wrote the interrupt routine with debounce for the push button. With respect to the capstone project, Dhanesh has been owning the mechanical design, identifying vendors, machining, and assembly of the hardware for the track and infrastructure sensing units. He helped me in soldering and building the connectors for the electronic components for the RC Cars.

Jash Shah: Jash has been working on a pose estimation method. He and Ronit helped me understand and integrate the code for the DC motor in the full code for the sensors and motor control lab and connect it to the GUI.

Ronit Hire: Ronit developed GUI and it with the rest of the code for the Sensors and Motor lab. With respect to the capstone project, Ronit has been working on procurement of all the components, aided in machining the mechanical hardware, partnered on the controller simulation in Simulink, helped in developing and debugging the pose estimation algorithm, and helped me in trying to calibrate the VESC.

Plans

My goal for the next lab demonstration is to attempt to teleoperate the RC car over WLAN and collect IMU data from it. This will involve interfacing the ESP8266 module with Arduino mega, calibrating the duty cycle for the BLDC motor, receiving, unpacking, and responding to the control inputs.

Quiz – Sensors and motor control lab

Reading a datasheet - ADXL335 accelerometer

- Sensor range: -3.6g to + 3.6g
- Sensor's dynamic range: 7.2g
- The purpose of the capacitor C_{DC} on the LHS of the functional block diagram on p. 1 is to decouple the accelerometer from the noise on the power supply. It is connected between the power supply and the accelerometer IC in parallel to one another and creates a low impedance path for the high-frequency signals to get shunted resulting in a clean DC signal.
- Equation for the sensor's transfer function: V_{out} = 1.5 V + 300 mV * a (At Vs = 3V)
- Largest expected nonlinearity error in g: 0.0216 g
- Sensor's bandwidth for the X- and Y-axes: 1600 Hz
- Expected noise at 25Hz measurement bandwidth for X and Y axis: 945 µg
- Steps to experimentally determine the root-mean-square (RMS) noise of this sensor:
    - Take multiple sensor readings by keeping other operating conditions the same (e.g., temperature) at regular intervals.
    - Calculate the mean of all the sensor readings and subtract it from all the readings to get a residual signal value.
    - Find the square of the residual signal and take the average of all obtained values.
    - By taking the square root of the average of the squared residual values find the RMS value of the noise
    - Repeat this for each axis of the accelerometer.
    - Record the sensor readings: Collect a large number of sensor readings with constant operating conditions (e.g., temperature, humidity, etc.) over a long period of time to ensure that the readings are statistically significant. The readings should be taken at regular intervals to avoid aliasing and to ensure that the noise is Gaussian.

  Assumptions:

    - There is white gaussian noise – it has a normal distribution and consecutive sensor readings are independent.
    - There is no aliasing effect and sufficient samples are collected.
    - The noise value is stationary, and its distribution doesn't change over time.

Signal conditioning

Filtering

- Problems with moving average filter:
    - It has an inherent lag which scales with the window size. For example, if we take the average of 200 samples, the reading will take time to reach the true state being measured.
    - It is not robust to outliers and large outliers can skew the average values.

- Problems with median average filter:
    - It is computationally expensive and requires sorting the data for each sample.
    - It might miss capturing correct readings where a large change occurred as it finds the median value in the window. This is a problem when trying to preserve the high-frequency components of the signal.

Op amps

- Case 1: Uncalibrated sensor has a range of -1.5 to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output)
    - V1 will be the reference input and V2 will be the input voltage.
    - Using a non-inverting configuration, $R_f/R_i = 1$, $V_{ref} = -3V$.

- Case 2: Uncalibrated sensor has a range of -2.5 to 2.5V (-2.5V should give a 0V output and 2.5V should give a 5V output)
    - This calibration is not possible as it would require a negative net ratio of R_f/R_i, resistance cannot be negative.

Control

- To implement a discrete version of PID control, calculate error ‘e’ after each time step duration of ‘t’ seconds.
    - Error is the difference between the desired position and current position.
    - Proportional term is the product of the proportional gain and the current error.
    - Derivative term is the product of the derivative gain and rate of change of error (the difference between the current error and previous error divided by the time between them).
    - Integral term is the product of the integral gain, cumulative sum of errors since start and the total time elapsed since start.

- To speed up the sluggish system I would use the proportional term and increase the proportional gain to reduce the rise time of the system.

- To reduce the steady state error, I would use the integral term and increase the integral gain to compensate the accumulated error. This will drive it towards the desired position.
- To reduce the overshoot, I would use the derivative term and increase the derivative gain to dampen the oscillations; effectively a smaller control input would be generated closer to the desired position.

Appendix

Appendix A: Code for control of stepper motor via potentiometer

```
#define pot_pin A0
#define stepper_step_pin 11
#define stepper_dir_pin 10
const int stepper_step_size = 1.8;
int pot_prev = 0;
float stepper_change;

void setup()
{
  Serial.begin(9600);
  pinMode(pot_pin, INPUT);
  pinMode(stepper_step_pin, OUTPUT);
  pinMode(stepper_dir_pin, OUTPUT);
  pot_prev = get_pot();
}

void loop()
{
  //Every two seconds, move the stepper motor proportional to the change in
potentiometer reading
  stepper_change = pot_prev - get_pot();
  stepper_change = map(stepper_change, -1023, 1023, -360, 360);
  pot_prev = get_pot();
  stepper_control(stepper_change);
  delay(2000);
}

int get_pot()
{return analogRead(pot_pin);}

void stepper_control(float stepper_pos)
{
  if (stepper_pos > 0)
    digitalWrite(stepper_dir_pin, HIGH); //rotate in positive direction
  else
    digitalWrite(stepper_dir_pin, LOW);

  stepper_pos = abs(stepper_pos);
  while(stepper_pos > 0)
  {
    digitalWrite(stepper_step_pin, HIGH);
    delayMicroseconds(2000);
    digitalWrite(stepper_step_pin, LOW);
    stepper_pos -= stepper_step_size;
  };
```

Appendix B: Integrated code for sensors and motor lab

```
//Arduino Mega - external interrupt pin 2,3,18,19,20,21
#define pi 3.1416

//Push button as E-stop interrupt for all motors
#define push_estop_pin 18


//Push button as state machine -- sensor based control and GUI
#define push_state_pin 21


//-------------------------Stepper motor------------------------------
#define stepper_step_pin 11
#define stepper_dir_pin 10
const int stepper_step_size = 1.8;
//--------------------------------------------------------------------


//-------------------------Servo motor---------------------------------
#define servo_pin 9 //PWM pin
#include <Servo.h>
Servo myservo;
//--------------------------------------------------------------------


//-------------------------DC motor-----------------------------------------
#define IN1 12
#define IN2 5
#define PWM 4

//Quadrature encoder
#define ENCA 3  // Quadrature encoder A pin - Interrupt
#define ENCB 2 // Quadrature encoder B pin - Interrupt

#define VMAX 11.7
#define VMIN -11.7

// PID gains
float kp_rpm = 0.001;
float kd_rpm = 0;
float ki_rpm = 0.0005;

float kp_pos = 30;
float kd_pos = 0.001;
```

```
float ki_pos = 0.001;

// motor modes
char dc_motor_mode = 'P';

long EncoderCount = 0;

volatile unsigned long count = 0;
unsigned long count_prev = 0;
float theta, RPM, RPM_d;
float theta_prev = 0;
int dt;
float RPM_max = 103;
int POS = 0;
float e;
float e_prev=0;
float e_int=0;
float e_int_prev=0;
float V = 0.1;
unsigned long t;
unsigned long t_prev = 0;

bool first_time = true;
bool switch_dc_motor_mode = false;

//--------------------------------------------------------------------------------
-

//IR sensor (Analog) -- Transfer function (10cm - 80cm range)
#include <SharpIR.h>
#define IRPin A0 //Analog pin
#define IRmodel 20150
SharpIR mySensor = SharpIR(IRPin, IRmodel);
//Ultrasonic sensor
#define ultrasonic_echo_pin 6
#define ultrasonic_trigger_pin 7
//Temperature sensor (Analog)
#define temperature_pin A1 //Analog pin
float temp_filter[10];

#define pot_pin A2

bool e_stop;        //STOP ALL MOTORS WHEN TRUE
bool state_machine; //FLIP ON TRIGGER - false is sensor control | TRUE is GUI
control
```

```
bool transmit = false;

//Variables
struct vars
{
  float pot;
  float ir_distance_cm;
  float ultrasonic_distance_cm;
  float temperature;
};
vars var;

struct gui_mot_commands
{
  int16_t stepper_pos;
  int16_t servo_pos;
  int16_t dc_val; //Ir then this is RPM -> velocity control else run pos control
with dc val
  char control_type; //R or P
};

gui_mot_commands gui_mc;


void setup()
{
  Serial.begin(9600);

  pinMode(push_estop_pin, INPUT);
  attachInterrupt(digitalPinToInterrupt(push_estop_pin), estop, FALLING);
  delay(200);
  pinMode(push_state_pin, INPUT);
  attachInterrupt(digitalPinToInterrupt(push_state_pin), change_mode, FALLING);

  pinMode(stepper_step_pin, OUTPUT);
  pinMode(stepper_dir_pin, OUTPUT);

  myservo.attach(servo_pin,1000,2000);
//https://www.arduino.cc/reference/en/libraries/servo/attach/

  //----------- dc motor config
  pinMode(ENCB, INPUT_PULLUP);
  pinMode(ENCA, INPUT_PULLUP);
  pinMode(IN2, OUTPUT);
  pinMode(IN1, OUTPUT);
```

```
  cli();
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT2 = 0;
  OCR2A = 12499; //Prescaler = 64
  TCCR2B |= (1 << WGM22);
  TCCR2B |= (1 << CS21 | 1 << CS20);
  TIMSK2 |= (1 << OCIE2A);
  sei();
  gui_mc.control_type = 'R';
  gui_mc.dc_val = 20;
  //---------------

  pinMode(temperature_pin, INPUT);
  pinMode(IRPin, INPUT);
  pinMode(ultrasonic_trigger_pin, OUTPUT);
  pinMode(ultrasonic_echo_pin, INPUT);
  pinMode(pot_pin, INPUT);

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  transmit=false;
  //establish_connection();

//Initialize var
  e_stop = false;
  state_machine = false;

  for(int i=0;i<10;i++)
  { temp_filter[i] =get_temperature();}
}

void send_data()
{
  if(transmit)
    Serial.write((byte*)&var, sizeof(var));
}

void establish_connection()
{
  bool rec = false;
  while(!rec)
  {
    if(Serial.available()>0)
    {
```

```
      char inp = Serial.read();
      if(inp=='S') // request to start data
      {
        transmit=true;
        digitalWrite(LED_BUILTIN, LOW);
        Serial.write('A'); //connection acknowledged, next byte will be data
        delay(100);
        rec=true;
      }
    }
  }
}

//ISR 1
void change_mode()
 {//Serial.println("here");
  static unsigned long last_interrupt_time_mode = 0;
  unsigned long interrupt_time_mode = millis();
  if (interrupt_time_mode - last_interrupt_time_mode > 200)
  {state_machine = !state_machine; }
  last_interrupt_time_mode = interrupt_time_mode;
  }

void estop()
 {
  static unsigned long last_interrupt_time_estop = 0;
  unsigned long interrupt_time_estop = millis();
  if (interrupt_time_estop - last_interrupt_time_estop > 200)
  { e_stop = !e_stop;   }
  last_interrupt_time_estop = interrupt_time_estop;
 }


void servo_control(int servo_pos)
{
    myservo.write(servo_pos);
}

void stepper_control(int stepper_pos)
{
  if (stepper_pos > 0)
    digitalWrite(stepper_dir_pin, HIGH); //rotate in positive direction
  else
    digitalWrite(stepper_dir_pin, LOW);
```

```
  stepper_pos = abs(stepper_pos);
  while(stepper_pos > 0)
  {
    digitalWrite(stepper_step_pin, HIGH);
    delayMicroseconds(2000);
    digitalWrite(stepper_step_pin, LOW);
    stepper_pos -= stepper_step_size;
  };
}


int get_pot()
{
  return analogRead(pot_pin);
}

int get_ir_distance()
{
  return mySensor.distance();
}

int get_ultrasonic_distance()
{
  digitalWrite(ultrasonic_trigger_pin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(ultrasonic_trigger_pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(ultrasonic_echo_pin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  double duration = pulseIn(ultrasonic_echo_pin, HIGH);
  // Calculating the distance
  float distance = duration * 0.034 / 2;
  return distance;
}

float get_temperature()
{
  int total = 0;
  for(int i = 0;i<9; i++)
  { temp_filter[i] =  temp_filter[i+1];
    total += temp_filter[i];
  }
  temp_filter[9] =  analogRead(temperature_pin)* 0.48828125;
  total += temp_filter[9];
```

```
  return total/10;

}

void ISR_EncoderA() {

  bool PinB = digitalRead(ENCA);
  bool PinA = digitalRead(ENCB);

  if (PinB == LOW) {
    if (PinA == HIGH) {
      EncoderCount++;
    }
    else {
      EncoderCount--;
    }
  }

  else {
    if (PinA == HIGH) {
      EncoderCount--;
    }
    else {
      EncoderCount++;
    }
  }

}

void ISR_EncoderB() {

  bool PinB = digitalRead(ENCB);
  bool PinA = digitalRead(ENCA);

  if (PinA == LOW) {
    if (PinB == HIGH) {
      EncoderCount--;
    }
    else {
      EncoderCount++;
    }
  }

  else {
```

```
    if (PinB == HIGH) {
      EncoderCount++;
    }
    else {
      EncoderCount--;
    }
  }

}

void readEncoder() {

  int b = digitalRead(ENCB);
  if (b > 0) {
    POS++;
  }
  else {
    POS--;
  }
}

void setMotor(int dir, int pwmVal, int pwm, int in1, int in2) {
  analogWrite(pwm, pwmVal);
  if (dir == 1) {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
  }
  else if (dir == -1) {
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
  }
  else {
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
  }
}

void WriteDriverVoltage(float V, float Vmax) {
  int PWMval = int(255 * abs(V) / Vmax);
  if (PWMval > 255) {
    PWMval = 255;
  }
  if (V > 0) {
    digitalWrite(IN2, HIGH);
    digitalWrite(IN1, LOW);
```

```
  }
  else if (V < 0) {
    digitalWrite(IN2, LOW);
    digitalWrite(IN1, HIGH);
  }
  else {
    digitalWrite(IN2, LOW);
    digitalWrite(IN1, LOW);
  }
  analogWrite(PWM, PWMval);
}

void dc_velocity_control(int RPM_d)
{
  // conditions:
  // count = 0
  // count_prev = 0 before starting
  // theta_prev = 0
  // encoder_count = 0
  // t_prev = 0
  // e_prev = 0
  // e_int_prev = 0
  // e_int = 0
  // Serial.println(RPM_d);
  if(switch_dc_motor_mode)
  {
    detachInterrupt(digitalPinToInterrupt(ENCA));
    delay(100);
    attachInterrupt(digitalPinToInterrupt(ENCB), ISR_EncoderA, RISING);
    attachInterrupt(digitalPinToInterrupt(ENCA), ISR_EncoderB, CHANGE);
    delay(500);
    count = 0;
    count_prev = 0;
    EncoderCount = 0;
    theta_prev =0;
    t_prev = 0;
    e_prev = 0;
    e_int_prev = 0;
    e_int = 0;
    switch_dc_motor_mode=false;
  }
  if (first_time)
  {
    attachInterrupt(digitalPinToInterrupt(ENCB), ISR_EncoderA, RISING);
    attachInterrupt(digitalPinToInterrupt(ENCA), ISR_EncoderB, CHANGE);
```

```
    first_time = false;
  }

  if (count > count_prev)
  {
    t = millis();
    theta = EncoderCount / 900.0;
    dt = (t - t_prev);

    // RPM_d = 20;
    RPM = (theta - theta_prev) / (dt/1000.0) * 60;
    e = RPM_d - RPM;

    e_int = e_int_prev + (dt * (e + e_prev) / 2);
    V = kp_rpm * e + ki_rpm * e_int + (kd_rpm * (e - e_prev) / dt);
    if (V > VMAX)
    {
      V = VMAX;
      e_int = e_int_prev;
    }
    if (V < VMIN)
    {
      V = VMIN;
      e_int = e_int_prev;
    }

    WriteDriverVoltage(V, VMAX);

    theta_prev = theta;
    count_prev = count;
    t_prev = t;
    e_int_prev = e_int;
    e_prev = e;
  }

}

void dc_position_control(int degree)
{
  // condition
  //pos =0
  //e_prev = 0
  //t_prev = 0
  //e_int = 0
  // Serial.println(degree);
```

```
if (switch_dc_motor_mode)
{
  // Serial.println("here");
  // remove velocity control interrupts
  detachInterrupt(digitalPinToInterrupt(ENCB));
  detachInterrupt(digitalPinToInterrupt(ENCA));
  delay(100);
  // setup interrupts for position control
  attachInterrupt(digitalPinToInterrupt(ENCA), readEncoder, RISING);

  // reset default states
  setMotor(-1, 0, PWM, IN1, IN2);
  delay(500);
  switch_dc_motor_mode = false;
  POS = 0;
  e_prev = 0;
  t_prev = 0;
  e_int = 0;
}
if (first_time)
{
  attachInterrupt(digitalPinToInterrupt(ENCA), readEncoder, RISING);
  first_time = false;
}
// Serial.println(POS);

int target = degree * 100 / 360;
// time difference
long t_pos = micros();
float delta_ts = ((float)(t_pos-t_prev))/ 1.0e6;
t_prev = t_pos;
// error
e = POS - target;
// derivative
float dedt = (e - e_prev) / delta_ts;
// integral
e_int = e_int + e * delta_ts;
// control signal
float u = kp_pos * e + kd_pos * dedt + ki_pos * e_int;
// Serial.println(u);

// motor power
float pwr = fabs(u);
if ( pwr > 255 ) {
  pwr = 255;
```

```
  }
  // motor direction
  int dir = 1;
  if (u < 0) {
    dir = -1;
  }
  // signal the motor
  setMotor(dir, pwr, PWM, IN1, IN2);
  // store previous error
  e_prev = e;

}

void receive_data()
{
   if(Serial.available())
   {
     char inp =Serial.read();
     if(inp =='I')
     {
       size_t nb = Serial.readBytes((byte *)&gui_mc, sizeof(struct
gui_mot_commands));
      //  Serial.println(gui_mc.control_type);
      //  Serial.println(gui_mc.dc_val);

     }
     //discard everything
     while(Serial.available()>0) Serial.read();
   }
}


void loop() {
  // put your main code here, to run repeatedly:
  char prev_mode = gui_mc.control_type;
  if (e_stop)
  {
    Serial.println("Estooppped");
    //E-stop is on
    //Stop DC motor, do not move other motors
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);

  }
  else
```

```
{ //Serial.println(state_machine);
  if (state_machine)
  {
    Serial.println("GUI Mode");
    //Inputs from GUI
    receive_data();
        //Serial read
          //move servo to set pos
          //spin dc motor at set speed
          //steppper motor change
          //set dc motor to pos
    //move motors accordingly
    if (gui_mc.control_type == 'P')
    {
      if(gui_mc.control_type!=prev_mode)
      {
        switch_dc_motor_mode=true;
      }
      dc_position_control(gui_mc.dc_val);
      prev_mode = 'P';
      // Serial.println(gui_mc.control_type);
      // Serial.println(gui_mc.dc_val);
    }
    else if (gui_mc.control_type == 'R')
    {
      if(gui_mc.control_type!=prev_mode)
      {
        switch_dc_motor_mode=true;
      }
      dc_velocity_control(gui_mc.dc_val);
      // Serial.println(gui_mc.control_type);
      prev_mode='R';
    }

    var.pot = get_pot();
    var.ir_distance_cm = get_ir_distance();
    var.ultrasonic_distance_cm = get_ultrasonic_distance();
    var.temperature = get_temperature();
    send_data();
  }
  else
  {
    //Inputs from Sensors
    Serial.println("Sensor Mode");
    var.pot = get_pot();
```

```
      var.ir_distance_cm = get_ir_distance();
      var.ultrasonic_distance_cm = get_ultrasonic_distance();
      var.temperature = get_temperature();
      send_data();

      if (var.ir_distance_cm < 35 && var.temperature < 80 )
      {
        for(int i=0;i<90;i++)
          {servo_control(i);  delay(10);}
        for(int i=90;i>=0;i--)
          {servo_control(i);  delay(10);}
      }

      while (get_ultrasonic_distance() < 7)
      {stepper_control(5);};

      if(var.pot>511)
         dc_velocity_control(map(var.pot,0,512,-103,0));
      else
         dc_velocity_control(map(var.pot,512,1023,0,103));
    }
  }
}

ISR(TIMER2_COMPA_vect) {
  count++;
  //  Serial.print(count * 0.05); Serial.print(" \t");
}
```