SENSORS AND MOTORS CONTROL LAB

INDIVIDUAL LAB REPORT [ILR01]

OCTOBER 16, 2015

TUSHAR CHUGH

TEAM B-AUTOPIRATES
SHIYU DONG, BIKRAMJOT HANZRA, TAE-HYUNG KIM, WILLIAM SETO

SENSORS AND MOTORS CONTROL LAB

INDIVIDUAL PROGRESS

My primary role for sensors and motor control lab assignment was to develop and integrate the Graphical User Interface (GUI). I also defined the protocols for communication which we use to exchange serial data between Arduino and the computer. In addition to it, I handled the modes which can also be changed by interrupts fired from push button on Arduino (GUI acts as a master and governs these states for Arduino).

GRAPHICAL USER INTERFACE (GUI)

Background

Our user interface (GUI) is developed using C# and XAML. I used Visual Studio as the Integrated Development Environment (IDE). I had worked on C# in my job so the experience helped in developing the robust interface in less amount of time.

Visible Features of GUI

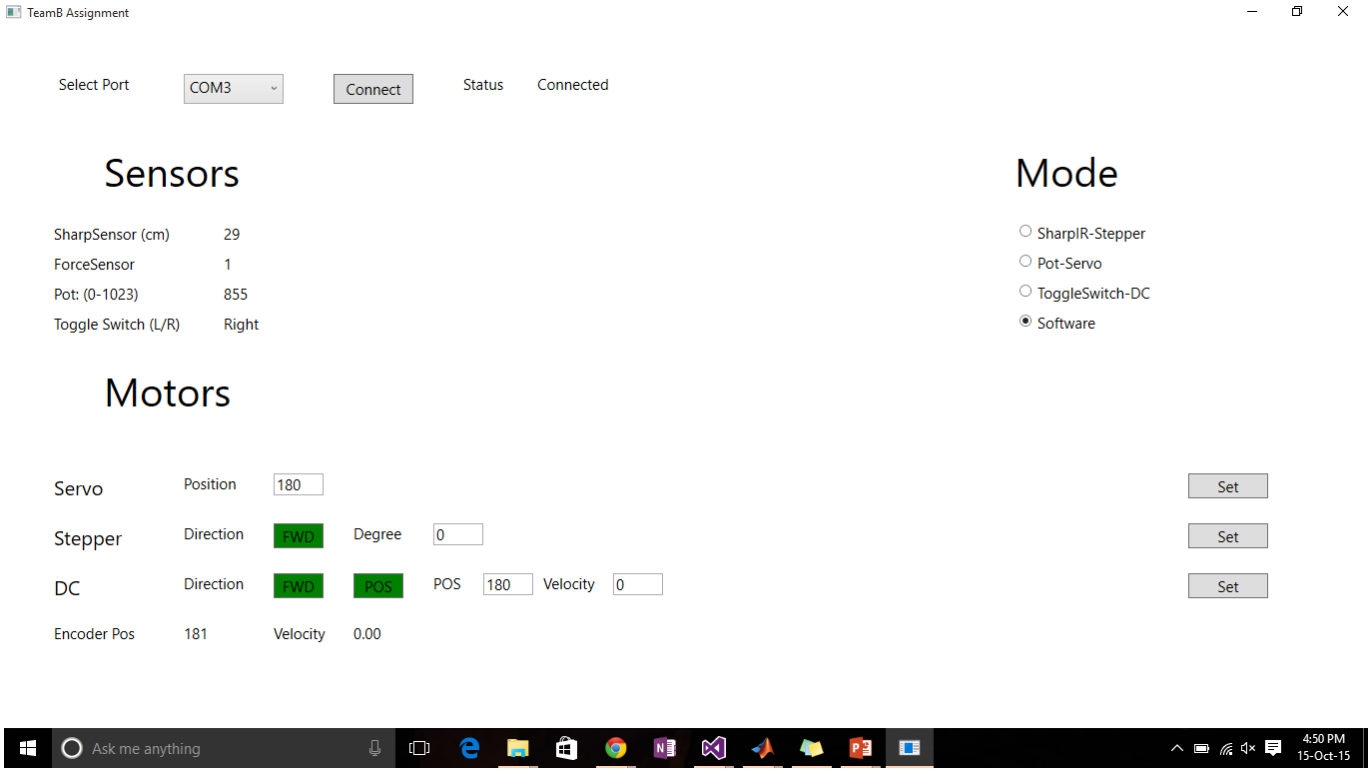
We knew from our previous experience that controlling all the motors simultaneously by attaching each one of them to a different sensors input would be a daunting task and might cause trouble. For example, there could have been latency issues or data loss issues we could have faced when interrupt from DC motor encoder would have fired at the same time while receiving serial data. So, we decided to have 4 states/modes for our circuit which are explained in Modes section.

Our GUI is dividing into 4 section:

a. **Connectivity Handle:** This can be seen on the top of the UI. Here, we can select the port on which Arduino is connected and see the status if it is connected or if there is an error.
b. **Sensor Inputs:** Once the Arduino is connected, this section displays the value of all four sensors which we have used. Value of sharp sensor is in cm, of force resistive sensor is range 0-1023 (depending on the force applied, of potentiometer is also in range 0-1023 (as we have the 10 bit ADC on Arduino), of Toggle switch is Left or Right. It is to be noted that real time values of sensors are displayed on screen in all the 4 modes. This section is in middle-left of the UI.
c. **Mode Select:** This section allows us to select and toggle between 4 available modes. The first mode ‘SharpIR-Stepper’ wherein the stepper motor responds to the distance of the object from IR sensor. The second mode ‘Pot-Servo’ enables servo motor’s position to be controlled with the rotation of the potentiometer knob. ‘Toggle-DC’ mode allows to control the direction (forward and backward) of the DC motor with respect to the state of the toggle switch (Left-Right). The final mode ‘software’ allows to control the motors directly from the UI.

d. **Motor Control:** This section only works when 'Software mode' is enabled. We can only control one motor at a time. The 'set' button on the UI has to be pressed to send the command. Below are the controls that are available for 3 motors:
    - **Servo:** Position (0-180 degree)
    - **Stepper:** Direction (Forward/Backward), Degrees to rotate (0-999)
    - **DC:** Direction (Forward/Backward), Mode (Position or Velocity PID control), Position (0-999, it is relative), Velocity (0-22). Encoder position and velocity are also displayed.

Figure 1: Screen shot of the GUI. Arduino is connected to COM3. We can see the values of all 4 sensors. Software mode is selected which means that the motors can be controlled directly from the GUI in this case.



BackEnd Features:

a. **Serial Communication**

We are communicating with Arduino through serial port at the Baud rate of 11,5200 and with parameters as Parity None, 8-bit data, 1 Stop-bit. I have written a separate class *serialHelper* which handles all the communication with serial port. It performs following functions: get available ports, connect to port, send data to port, receive data from port and

parse the received data. This class can be re-used with any other program with minimum modifications.

Below is the sample of the protocol which I defined for the communication between Arduino and computer. It is to be noted that we neglect the data which is not having the start marker at the beginning and end marker at the end of the data stream that is received or sent.

Send Data (To Arduino)

APMDXXXVK

AP is the start marker

M is the mode (0->Servo, 1->Stepper, 2->DC, 3->SharpIR-Stepper Mode, 4-> None, 5->Pot-Sevo Moder, 6->ToggleSW-DC Mode, 7->SoftwareMode)

D is direction select (0 is forward, 1 is backward)

XXX is either degree/position/velocity (depends on the other select values)

V is velocity or position PID control mode selector for DC mode selector

K is the end marker

Receive Data (From Arduino)

AP SH XXX XXX FO XXX PO XXX TO XXX BT Y EN XXX VL XXX K

AP is the start marker

SH is sharp sensor initial, FO is force sensor, PO is potentiometer, TO is toggle switch, BT is button interrupt status EN is position value of encoder, VL is velocity of motor

Y is interrupt status from push button (1 if interrupt is fired)

XXX can be any value from 0-65536 (Length of this string can be 0-5 but usually it is 0-4)

K is the end marker

It is to be noted that the received data as spaces ‘ ‘ in between each marker. So, it can be thought of as a concatenation of substrings. I am splitting the long string using these spaces to get substrings.

b. Writing to CSV File

CSVHelper class handler all functions related to writing to CSV. This includes creating a CSV file with appropriate headers and inserting the sensors data in the file. Right now, we are writing ADC input value (0-1023) and distance (in cms) of the sharp IR sensor to the file. The file can be imported in matlab to plot the transfer function or the graph of the last run

c. UI Elements Update

This is one of the most important feature of our UI. We are not explicitly refreshing the UI every time when we receive sensor data but we have used *data binding*. There is a binding between Elements on the UI and the variables of the *serialHandler* class. Whenever a new value is received and assigned to these variable the *eventHandlers* fire the interrupt which is handled by the *MainWindow* class. For more details, see property change events and how these are implemented in the *serialHelper* class.

d. **Cloud Connectivity**

I have also made a proof-of-concept (in a separate code) wherein we can send data of the sensors to the cloud. I used Microsoft Azure as the platform. The data is first ingested to *Azure Event Hub* from a C# application, then *Azure Stream Analytics* subscribes to this input data and transmits the most recent data to PowerBi. We can see the graph of sensor value (like distance for Sharp IR) vs time on PowerBi chart.

CHALLENGES

a. **Communication**

Initially we were sending data from Arduino separately for all the sensors and the data was passed instantly when the value were read from sensors. The end results was that the input data stream was very fast and there was data loss between computer and the Arduino. Also, it was difficult to handle data separately for all the sensor values. We overcame this problem using two solutions. First, we embedded all the send and receive data respectively in single packets which are explained in the serial communication section above. Second, we only sent the data from Arduino only after 200 milliseconds. This was enough for us to process and display the data in the GUI. Earlier, with fast data rate the display value on the GUI was changing so fast that it was very hard to read.

We also tried with *json* as the string format but we realized that it has too much of overhead for us to exchange so many headers with the values.

b. **Mode switching through push button:**

To make the state handling easy we handled the mode switching between various stated initiate by push buttons through GUI as computer doesn't have memory constraints and code is easier to debug with breakpoints. I faced a technical issue to fire interrupts for this in *MainWindow* class as for this I would have had to create an object of *MainWindow* in *serialHelper* which is not a practical solution. I overcame this by using *UIDispatcher* and *propertyChange Handler.* For more information refer to code.

TEAM WORK

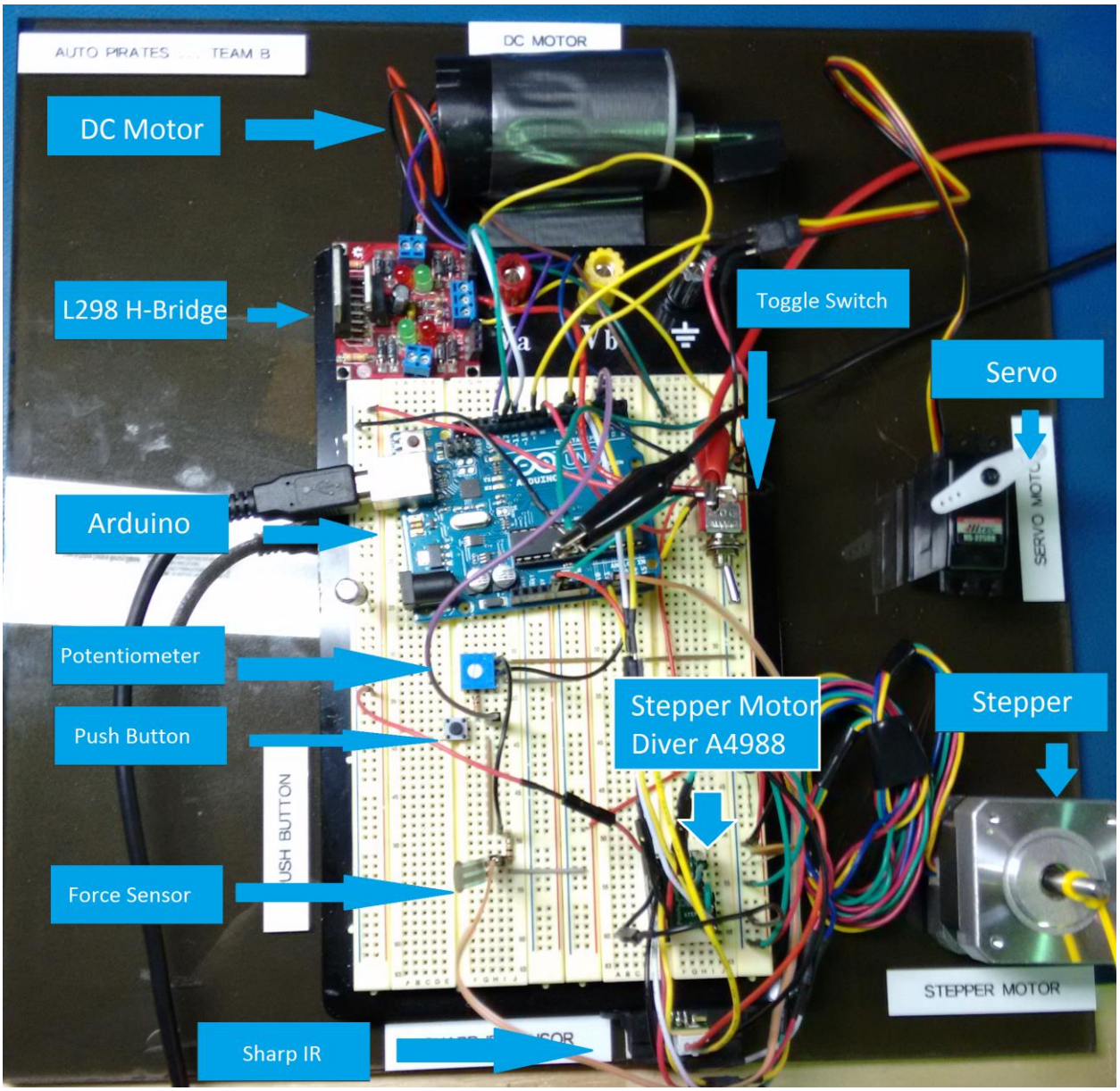
a. **Shiyu Dong:** He integrated force resistive sensor with Arduino and PID control with velocity for DC motor. He also worked on position control for DC motor with William.
b. **Bikram Hanzra:** He integrated Toggle Switch and servo motor with Arduino. He also wrote the matlab function to plot the graph of ADC output vs Distance for Arduino.
c. **Tae-Hyung Kim:** Tae-Hyung integrated potentiometer and stepper motor with Arduino.
d. **William Seto:** William was instrumental in binding/integrating all the separate components of Arduino code to work together as the single unit. He also integrated Sharp IR sensor with Arduino and worked with Shiyu on tuning the position control of DC motor.

FUTURE WORK

a. **For this assignment**
   - Sample data correctly and plot the transfer function in matlab

b. **For Water-Taxi Project (For next 2 weeks)**
   - Gather raw data from Radar
   - Parse data of radar using 'OpenCPN' library
   - Study state of art path-planning algorithms and select 2-3 to experiment for performance
   - Define functions (of code as it is a requirement from our sponsor)
   - Draft detailed software architecture

Figure: Circuit with all peripherals integrated (labelled in blue boxes)



APPENDIX

Arduino Code

```
/* BEGIN DC MOTOR DEFINES */

#define InA1          11                // INA motor pin
#define InB1          12                // INB motor pin
#define PWM1           6                 // PWM motor pin

#define FORWARD        1                 // direction of rotation
#define BACKWARD       2                 // direction of rotation

#define encoderPinA    2
#define encoderPinB    13

#define button_left    8
#define button_right   7


volatile long encoderPos = 0; //gives tick position from 0 reference
volatile long last_encoder_pos = 0;
volatile float velocity = 0.0;
volatile long desiredMotorPos = 0;

/* POSITION PID CONSTANTS */
volatile float p_error=0;     //current error
volatile float pre_error=0;   //previous p error
volatile float d_error=0;     // current p error - previous p error
volatile float i_error=0;     // i error, integrate p errors
const float kp           =25; //P parameter
const float ki           =0.00; //I parameter
const float kd           =30; //D parameter

/* VELOCITY PID CONSTANTS */
volatile float p_error_vel  =0;    //current error
volatile float pre_error_vel=0;    //previous p error
volatile float d_error_vel  =0;    // current p error - previous p error
volatile float i_error_vel  =0;    // i error, integrate p errors
const float kp_vel        =25;    //P parameter
const float ki_vel        =0.4;   //I parameter
const float kd_vel        =5;     //D parameter

volatile int Motor_speed_value=0;
const int max_speed=255;
/*** END DC MOTOR DEFINES ***/



/*** BEGIN STEPPER MOTOR DEFINES ***/

#include <Stepper.h>
#include <SharpIR.h>

long stepsPerRevolution = 800;

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 4, 5);

volatile long stepCount = 0;  // number of steps the motor has taken
SharpIR sharp(A0, 1, 50, 1080);

/*** END STEPPER MOTOR DEFINES ***/
```

```
/*** BEGIN SERVO MOTOR DEFINES ***/

#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = 2;  // analog pin used to connect the potentiometer

volatile int servoAngle = 90; //servo resets itself to 90 in the beginning

/*** END SERVO MOTOR DEFINES ***/


/*** serial stuff ***/
char mode = '7';
char dir = '0';
long val = 0;
char type = '0';
int stateChanged = 0;

volatile unsigned long last_gui_update_time = 0;

/* push button stuff */
#define pushbutton 3
volatile int pushbutton_state = 0;
volatile unsigned long last_pushbutton_high_time = 0;
static int debounce_delay = 100;


volatile unsigned long last_time = 0;


void setup() {
 /* BEGIN DC MOTOR CONTROL SETUP */
 pinMode(InA1, OUTPUT);
 pinMode(InB1, OUTPUT);
 pinMode(PWM1, OUTPUT);
 pinMode(encoderPinA, INPUT);
 pinMode(encoderPinB, INPUT);
 pinMode(button_left, INPUT_PULLUP);   //toggle switch
 pinMode(button_right, INPUT_PULLUP);
 digitalWrite(encoderPinB, HIGH);       // turn on pullup resistor
 digitalWrite(encoderPinA, HIGH);
 attachInterrupt(digitalPinToInterrupt(encoderPinA), read_encoder, CHANGE); // encoder pin on interrupt 0 (pin 2)
 /* END DC MOTOR CONTROL SETUP */

 myservo.attach(9);  // attaches the servo on pin 9 to the servo object

 attachInterrupt(digitalPinToInterrupt(pushbutton), pushbutton_interrupt, RISING);


 Serial.begin (115200);
}

void loop() {

 if ((digitalRead(pushbutton) == HIGH)) {
   //in case button is held (which won't trigger interrupt)
   last_pushbutton_high_time = millis();
 }

 sendData();

 stateChanged = 0;

 if (Serial.available() > 0) {
```

```
  String input = Serial.readString();

  if (input[0] == 'A' && input[1] == 'P' && input[8] == 'K') {

    //if we got a new command, reset values
    //
    resetValues();

    //Serial.println(input);
    mode = input[2];
    dir = input[3];
    val = input.substring(4,7).toInt();
    //Serial.println(val);
    type = input[7];
    stateChanged = 1;
  }
 }


 //process commands
 switch (mode) {
  case '0':    //servo  , ok
    processServo(val, 1);
    break;
  case '1':    //stepper      //fix going backwards
    processStepperMotor(dir, val, 1);
    break;
  case '2':    //dc motor    kind of ok
    processDCMotor(dir, val, type);
    break;
  case '3':    //ir sensor -> stepper   , ok
    processStepperMotor(dir, val, 2);
    break;
  case '4':    //force
    break;
  case '5':    //potentiometer -> servo , ok
    processServo(val, 2);
    break;
  case '6':    //switch -> dc motor
    processDCMotor(dir, val, '2');
    break;
 }
}

void pushbutton_interrupt() {

 unsigned long interrupt_time = millis();
 // ignore interrupt unless button has been low for some time
 if (interrupt_time - last_pushbutton_high_time > debounce_delay)
 {
  pushbutton_state = 1;
 }
 last_pushbutton_high_time = interrupt_time;

}

void sendData(void) {

 unsigned long curr_time = millis();

 if (curr_time - last_gui_update_time > 250) {

  //first, read data
  int irReading = analogRead(A0);
  int irDistance = sharp.distance();
  int forceReading = analogRead(A1);
```

```
  int potReading = analogRead(potpin);
  int toggleValue = 1;
  if(digitalRead(button_left)==HIGH) {
   toggleValue = 0;
  }

  String message = "AP";
  message = message + " SH " + irReading + " " + irDistance;
  message = message + " FO " + forceReading;
  message = message + " PO " + potReading;
  message = message + " TO " + toggleValue;
  message = message + " BT " + pushbutton_state;
  message = message + " EN " + encoderPos;
  message = message + " VL " + velocity;
  message = message + " OK";

  Serial.println(message);
  /*
  Serial.print("SharpSensor: "); Serial.print(irReading); Serial.print(" "); Serial.println(irDistance);
  Serial.print("FlexSensor: "); Serial.println(forceReading);
  Serial.print("Pot: "); Serial.println(potReading);
  Serial.print("Toggle: "); Serial.println(toggleValue);
  //Serial.print("FlexSensor: ");
  */
  last_gui_update_time = curr_time;
  pushbutton_state = 0;
 }



}
void resetValues(void) {
 //reset encoder values so we can keep sending position commands to motor
 encoderPos = 0;
 last_encoder_pos = 0;
 //stop dc motor
 Motor_speed_value = 0;
 run_motor();

 i_error = 0;
 i_error_vel = 0;

}


/*** BEGIN DC MOTOR FUNCTIONS ***/


void processDCMotor(char dir, int value, char mode) {

 int vel;
 String message = "";
 if (dir == '1') {
  value = value * -1;
 }

 switch (mode) {
  case '0':    //position control

    if (stateChanged) {
     desiredMotorPos = encoderPos + value;
    }
    positionPID(desiredMotorPos);
    run_motor();

    /*
```

```
    Serial.print("pwm value: "); Serial.print(Motor_speed_value);
    Serial.print(" encoder: "); Serial.print(encoderPos);
    Serial.print(" error: "); Serial.print(p_error);
    Serial.print(" pre_error: "); Serial.println(pre_error);
    */
    //message = message + "encoder: " + encoderPos;
    //Serial.println(message);
    //delay(5);

    break;
   case '1':
    //value = value / 20; //convert to rate of our control loop

    /*
    if (stateChanged) {
      desiredMotorPos = 0;
    }
    vel = encoderPos - last_encoder_pos;
    last_encoder_pos = encoderPos;
    Serial.println(vel);
    desiredMotorPos += value;
    positionPID(desiredMotorPos);
    */

    velocityPID(value);
    run_motor();
    /*
    Serial.print(" velocity: "); Serial.print(velocity);
    Serial.print(" vel error: "); Serial.print(p_error_vel);
    Serial.print(" vel pre_error: "); Serial.println(pre_error_vel);
    */


    delay(48);
    break;

   case '2':
    if(digitalRead(button_left)==HIGH) {
      Motor_speed_value = 150;
    } else {
      Motor_speed_value = -150;
    }
    run_motor();
    break;

  }
}

int positionPID(int desired_value) {
  p_error = desired_value - encoderPos;
  d_error = p_error - pre_error;
  i_error = i_error + p_error;

  pre_error = p_error; //store current error

  Motor_speed_value = kp*p_error + ki*i_error + kd*d_error; //PID calculation

  if (Motor_speed_value>=max_speed) {
    Motor_speed_value = max_speed;
  }
  else if(Motor_speed_value<=-max_speed)
  {
    Motor_speed_value = -max_speed;
  }

}
```

```
void velocityPID(int desired_value){

  velocity = encoderPos - last_encoder_pos; // counts per ~50ms,
  last_encoder_pos = encoderPos;

  p_error_vel = desired_value - velocity;
  d_error_vel = p_error_vel - pre_error_vel;
  i_error_vel = i_error_vel + p_error_vel;

  pre_error_vel = p_error_vel; //store current error

  Motor_speed_value = kp_vel*p_error_vel + ki_vel*i_error_vel + kd_vel*d_error_vel;//PID calculation

  if (Motor_speed_value>=max_speed) {
    Motor_speed_value = max_speed;
  }
  else if(Motor_speed_value<=-max_speed) {
    Motor_speed_value = -max_speed;
  }

}


// ENCODER INTERRUPT
void read_encoder(){

  if (digitalRead(encoderPinA) == HIGH) {   // found a low-to-high on channel A
    if (digitalRead(encoderPinB) == LOW) {  // check channel B to see which way
                                    // encoder is turning
      encoderPos = encoderPos + 1;         // CW
    } else {
      encoderPos = encoderPos - 1;         // CCW
    }
  }
  else                                    // found a high-to-low on channel A
  {
    if (digitalRead(encoderPinB) == LOW) {   // check channel B to see which way
                                    // encoder is turning
      encoderPos = encoderPos - 1;         // CCW
    } else {
      encoderPos = encoderPos + 1;         // CW
    }
  }
}

void run_motor(){
  if(Motor_speed_value>=0) {
    analogWrite(PWM1, Motor_speed_value);
    digitalWrite(InA1, LOW);
    digitalWrite(InB1, HIGH);
  } else {
    analogWrite(PWM1, -Motor_speed_value);
    digitalWrite(InA1, HIGH);
    digitalWrite(InB1, LOW);
  }
}
/*** END DC MOTOR CONTROL FUNCTIONS ***/

/*** BEGIN STEPPER MOTOR FUNCTIONS ***/

void processStepperMotor(char dir, long deg, int mode) {

  int dis;
  int motorSpeed;
  int real_dir = 1;
```

```
 if (dir == '1') {
  real_dir = -1;   //convert our char
 }

 switch (mode) {
  case 1:   //user-defined
   if (stateChanged) {
    stepCount = deg * stepsPerRevolution;   //convert desired degrees to steps
    stepCount = stepCount / 360;
   }
   //Serial.println(stepCount);
   motorSpeed = 100 * real_dir;   //arbitrary speed
   if (stepCount > 0 ) {   //will block main loop
    run_stepper_motor(motorSpeed);
    stepCount -= 1;
   }
   break;
  case 2:   //sensor control
   dis = sharp.distance();
   //Serial.println(dis);
   if (dis > 75) {
    dis = 75;
   }

   if (dis < 5) {
    dis = 5;
   }
   // map it to a range from 0 to 300 (max speed = 300)
   motorSpeed = map(dis, 5, 75, 0, 300);
   //Serial.println(dis);
   motorSpeed = motorSpeed * real_dir;
   run_stepper_motor(motorSpeed);
   break;

 }
}

void run_stepper_motor(int motorSpeed) {
 if (motorSpeed > 0) {  // CCW
  myStepper.setSpeed(motorSpeed);
  // step 1/100 of a revolution:
  myStepper.step(1);
 }
 else if (motorSpeed < 0)
 {  // CW
  motorSpeed = abs(motorSpeed);
  myStepper.setSpeed(motorSpeed);
  myStepper.step(-1);
 }
}
/*** END STEPPER MOTOR FUNCTIONS ***/

/*** BEGIN SERVO MOTOR FUNCTIONS ***/
void processServo(int pos, int mode) {
 int pot_val = 0;

 switch (mode) {
  case 1:    //user-defined

   myservo.write(pos);
   servoAngle = pos;
   delay(15);
   break;
  case 2:    //sensor-defined
   pot_val = analogRead(potpin);          // reads the value of the potentiometer (value between 0 and 1023)
```

```
    pot_val = map(pot_val, 0, 1023, 0, 180);     // scale it to use it with the servo (value between 0 and 180)
    myservo.write(pot_val);                  // sets the servo position according to the scaled value
    servoAngle = pos;
    delay(15);
    break;

 }
}

/*** END SERVO MOTOR FUNCTIONS ***/
```

GUI Code

XAML

```
<Window x:Class="ArduinoConnect.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:ArduinoConnect"
        mc:Ignorable="d"
        Title="TeamB Assignment" Height="750" Width="900">
    <Grid>
        <TextBlock Text="Select Port" HorizontalAlignment="Left" FontSize="15" TextAlignment="Center"
VerticalAlignment="Top" Height="30" Width="80" Margin="50,50,0,0"/>
        <ComboBox x:Name="PortsList" HorizontalAlignment="Left" VerticalAlignment="Top" Height="30"
Width="100" FontSize="15" Margin=" 180, 50, 0, 0" />
        <Button x:Name="ConnectPorts" Content="Connect" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Height="30" Width="80" Margin="330,50,0,0" Click="ConnectPorts_Click"/>
        <TextBlock  Text="Status" FontSize="15" HorizontalAlignment="Left" TextAlignment="Center"
VerticalAlignment="Top" Height="30" Width="40" Margin="460,50,0,0"/>
        <TextBlock x:Name="IsConnected" Text= "{Binding PortStatus}" FontSize="15" TextAlignment="Center"
HorizontalAlignment="Left" VerticalAlignment="Top" Height="30" Width="120" Margin="510,50,0,0"/>

        <TextBlock Text="Mode" FontSize="40" HorizontalAlignment="Right"  VerticalAlignment="Top"
Margin="0,120,250,0"/>
        <RadioButton x:Name="SharpIRMode" Content="SharpIR-Stepper" Width="200" FontSize="15" Height="30"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,200,150,0" Click="SharpIRMode_Click" />
        <RadioButton x:Name="PotMode" Content="Pot-Servo" Width="200" FontSize="15" Height="30"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,230,150,0" Click="PotMode_Click" />
        <RadioButton x:Name="ToggleSwMode" Content="ToggleSwitch-DC" Width="200" FontSize="15" Height="30"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,260,150,0" Click="ToggleSwMode_Click" />
        <RadioButton x:Name="SoftwareMode" Content="Software" Width="200" FontSize="15" Height="30"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,290,150,0" Click="SoftwareMode_Click" />

        <TextBlock Text="Sensors" FontSize="40" HorizontalAlignment="Left"  VerticalAlignment="Top"
Margin="100,120,0,0"/>


        <TextBlock Text="SharpSensor (cm)" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="50,200,0,0"/>
        <TextBlock x:Name="SharpValue" Text="{Binding SharpSensor}" FontSize="15"
HorizontalAlignment="Left" VerticalAlignment="Top" Margin="220,200,0,0"/>

        <TextBlock Text="ForceSensor" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="100" Margin="50,230,0,0"/>
        <TextBlock x:Name="FlexValue" Text="{Binding FlexSensor}" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="220,230,0,0"/>

        <TextBlock Text="Pot: (0-1023)" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Margin="50,260,0,0"/>
```

```
        <TextBlock x:Name="EncoderValue" Text="{Binding EncoderValue}" FontSize="15"
HorizontalAlignment="Left" VerticalAlignment="Top" Margin="220,260,0,0"/>

        <TextBlock Text="Toggle Switch (L/R)" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="50,290,0,0"/>
        <TextBlock x:Name="ButtonStatus" Text="{Binding ButtonStatus}" FontSize="15"
HorizontalAlignment="Left" VerticalAlignment="Top" Margin="220,290,0,0"/>


        <TextBlock Text="Motors" FontSize="40" HorizontalAlignment="Left"  VerticalAlignment="Top"
Margin="100,340,0,0"/>

        <TextBlock Text="Servo" FontSize="20" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="100" Margin="50,450,0,0"/>
        <TextBlock Text="Position" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="100" Margin="180,450,0,0"/>
        <TextBox x:Name="ServoMotorPOS" Text="0" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="50" Margin="270,450,0,0"/>
        <Button x:Name="ServoButton" Content="Set" FontSize="15" Height="25" Width="80"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,450,100,0" Click="ServoButton_Click"/>

        <TextBlock Text="Stepper" FontSize="20" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="100" Margin="50,500,0,0"/>
        <TextBlock Text="Direction" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="100" Margin="180,500,0,0"/>
        <ToggleButton x:Name="StepperToggle" Content="FWD" FontSize="15" Background="Green"
HorizontalAlignment="Left" VerticalAlignment="Top" Width="50" Height="25" Margin="270,500,0,0"
Click="StepperToggle_Click"/>
        <TextBlock Text="Degree" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="100" Margin="350,500,0,0"/>
        <TextBox x:Name="StepperMotorDegree" Text="0" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="50" Margin="430,500,0,0"/>
        <Button x:Name="StepperButton" Content="Set" Height="25" FontSize="15" Width="80"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,500,100,0" Click="StepperButton_Click"/>

        <TextBlock Text="DC" FontSize="20" HorizontalAlignment="Left" VerticalAlignment="Top" Width="100"
Margin="50,550,0,0"/>
        <TextBlock Text="Direction" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="100" Margin="180,550,0,0"/>
        <ToggleButton x:Name="DCDirToggle" Content="FWD" FontSize="15" Background="Green"
HorizontalAlignment="Left" VerticalAlignment="Top" Width="50" Height="25" Margin="270,550,0,0"
Click="DCDirToggle_Click" />
        <ToggleButton x:Name="DCModeToggle" Content="POS" FontSize="15" Background="Green"
HorizontalAlignment="Left" VerticalAlignment="Top" Width="50" Height="25" Margin="350,550,0,0"
Click="DCModeToggle_Click" />
        <TextBlock Text="POS" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top" Width="50"
Margin="430,550,0,0"/>
        <TextBox x:Name="DCMotorPOS" Text="0" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="50" Margin="480,550,0,0"/>
        <TextBlock Text="Velocity" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Width="60" Margin="540,550,0,0"/>
        <TextBox x:Name="DCMotorPWM" Text="0" FontSize="15" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="50" Margin="610,550,0,0"/>
        <Button x:Name="DCButton" Content="Set" Height="25" FontSize="15" Width="80"
HorizontalAlignment="Right" VerticalAlignment="Top" Margin="0,550,100,0" Click="DCButton_Click"/>

        <TextBlock Text="Encoder Pos" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Height="25"  Margin="50,600,00,0"/>
        <TextBlock x:Name="EncoderPos" Text="{Binding EncoderPosition}" FontSize="15"
HorizontalAlignment="Left" VerticalAlignment="Top" Height="25"  Margin="180,600,00,0"/>

        <TextBlock Text="Velocity" FontSize="15" HorizontalAlignment="Left" VerticalAlignment="Top"
Height="25"  Margin="270,600,00,0"/>
        <TextBlock x:Name="VelocityTrack" Text="{Binding DCVelocity}" FontSize="15"
HorizontalAlignment="Left" VerticalAlignment="Top" Height="25"  Margin="350,600,00,0"/>

    </Grid>
</Window>
```

serialHelper.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO.Ports;
using System.ComponentModel;


namespace ArduinoConnect
{
    class SerialHelper : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        private SerialPort port1;

        CsvHelper csvHelper = new CsvHelper();

        private string portStatus = "Not Connected";

        private string sharpSensor = "0";
        private string buttonStatus = "LEFT";
        private string encoderValue = "0";
        private string flexSensor = "0";
        private string encoderPosition = "0";
        private string dcVelocity = "0";
        private int tempCount = 0;
        private int[] temp = new int[100];
        private bool interruptFired;

        public bool InterruptFired
        {
            get { return interruptFired; }
            set
            {
                interruptFired = value;
                OnInterruptChanged(new PropertyChangedEventArgs("InterruptFired"));

            }
        }

        public string EncoderPosition
        {
            get
            {
                return encoderPosition;
            }
            set
            {
                encoderPosition = value;
                OnPropertyChanged("EncoderPosition");
            }
        }

        public string DCVelocity
        {
            get
            {
                return dcVelocity;
            }
            set
            {
                dcVelocity = value;
                OnPropertyChanged("DCVelocity");
            }
```

```
}

public string SharpSensor
{
    get
    {
        return sharpSensor;
    }
    set
    {
        sharpSensor = value;
        OnPropertyChanged("SharpSensor");
    }
}

public string EncoderValue
{
    get
    {
        return encoderValue;
    }
    set
    {
        encoderValue = value;
        OnPropertyChanged("EncoderValue");
    }
}

public string FlexSensor
{
    get
    {
        return flexSensor;
    }
    set
    {
        flexSensor = value;
        OnPropertyChanged("FlexSensor");
    }
}

public string ButtonStatus
{
    get
    {
        return buttonStatus;
    }
    set
    {
        buttonStatus = value;
        OnPropertyChanged("ButtonStatus");
    }
}

public String PortStatus
{
    get
    {
        return portStatus;
    }
    set
    {
       portStatus = value;
       OnPropertyChanged("PortStatus");
    }
}

public string[] GetPorts()
{
```

```
            //store all available port names
            string[] ports = SerialPort.GetPortNames();

            return ports;
        }

        public void initializePorts(string portName)
        {

            try
            {
                //create serial port as configured below
                //BaudRate = 115200 bps, Parity:None, 8-bit data, 1 Stop-bit
                port1 = new SerialPort(portName, 115200, Parity.None, 8, StopBits.One);

                //create asynchronous event(it's a secondary threat of the main program)
                //will activate every time our serial device sends data to computer
                port1.DataReceived += new SerialDataReceivedEventHandler(port1_DataReceived);
                //finally open port
                port1.Open();
                PortStatus = "Connected";
            }
            catch (Exception e)
            {
                PortStatus = "Error";
            }
        }

        public void WriteData(string serialData)
        {

            try
            {
                port1.WriteLine(serialData);
            }
            catch (Exception e)
            {
                //Do nothing
            }
        }

        private void port1_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            try
            {
                string message = port1.ReadLine();
                string[] messages = message.Split(' ');

                    if (messages[0] == "AP" && messages[1] == "SH" && messages[4] == "FO" && messages[6]
== "PO" && messages[8] == "TO" && messages[10] == "BT" && messages[12] == "EN" && messages[14]=="VL")
                    {
                        SharpSensor = messages[3];
                        FlexSensor = messages[5];
                        EncoderValue = messages[7];
                        EncoderPosition = messages[13];
                        DCVelocity = messages[15];

                        if (Convert.ToInt32(messages[9]) == 1)
                            ButtonStatus = "Right";
                        else
                            ButtonStatus = "Left";
                        if (Convert.ToInt32(messages[11]) == 1)
                        {

                            InterruptFired = true;
                        }

                        csvHelper.addRow(messages[2], messages[3]);
                    }
```

```
            }
            catch (Exception)
            {

                throw;

            }
        // Create the OnPropertyChanged method to raise the event
        protected void OnPropertyChanged(string name)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null)
            {
                handler(this, new PropertyChangedEventArgs(name));
            }
        }

        public event PropertyChangedEventHandler InterruptChanged;

        public void OnInterruptChanged(PropertyChangedEventArgs e)
        {
            if (InterruptChanged != null)
            {
                InterruptChanged(this, e);
            }
        }
    }
}
```

csvHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace ArduinoConnect
{
    class CsvHelper
    {
       private  string filePath = @"C:\Users\Tushar\Desktop\TeamBAssignment5.csv";
       private string delimiter = ",";

        public  void createFile()
        {
               string[][] output = new string[][]{
               new string[]{"ADCOutput", "Distance"},
               };
               int length = output.GetLength(0);
               StringBuilder sb = new StringBuilder();
              for (int index = 0; index < length; index++)
                  sb.AppendLine(string.Join(delimiter, output[index]));

             File.WriteAllText(filePath, sb.ToString());
        }

        public void addRow(string column1, string column2)
        {
            string[][] output = new string[][]{
               new string[]{column1, column2},
               };
            StringBuilder sb = new StringBuilder();
            sb.AppendLine(string.Join(delimiter, output[0]));
            File.AppendAllText(filePath, sb.ToString());
        }
    }
```

```
}
```

MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.ComponentModel;
using System.Windows.Threading;
using System.Threading;

namespace ArduinoConnect
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>

    public partial class MainWindow : Window
    {
        SerialHelper serialHelper = new SerialHelper();
        bool isSoftwareMode = false;

        public MainWindow()
        {
            this.DataContext = serialHelper;
            InitializeComponent();
            InitializeSerial();
            CsvHelper csvhelper = new CsvHelper();
            csvhelper.createFile();
            serialHelper.InterruptChanged += serialHelp_InterruptFired;
        }

        void serialHelp_InterruptFired(object sender, PropertyChangedEventArgs e)
        {
            Dispatcher.BeginInvoke(DispatcherPriority.Input, new ThreadStart(() =>
            {

                var serialobj = sender as SerialHelper;

            if (SharpIRMode.IsChecked == true)
                {
                    string serialData = "AP500000K";
                    serialHelper.WriteData(serialData);
                    PotMode.IsChecked = true;
                }
                else if (PotMode.IsChecked == true)
                {
                    string serialData = "AP600000K";
                    serialHelper.WriteData(serialData);
                    ToggleSwMode.IsChecked = true;
                }
                else if (ToggleSwMode.IsChecked == true)
                {
                    string serialData = "AP700000K";
                    serialHelper.WriteData(serialData);
```

```
            isSoftwareMode = true;
            SoftwareMode.IsChecked = true;
        }
        else if (SoftwareMode.IsChecked == true)
        {
            isSoftwareMode = false;
            string serialData = "AP300000K";
            serialHelper.WriteData(serialData);
            SharpIRMode.IsChecked = true;
        }
    }));
}

void InitializeSerial()
{
    serialHelper.PortStatus = "Not Connected";
    string[] ports = serialHelper.GetPorts();
    List<string> portsList = new List<string>();
    for (int i = 0; i < ports.Count(); i++)
    {
        portsList.Add(ports[i]);
    }
    PortsList.ItemsSource = portsList;
    PortsList.SelectedIndex = 0;
}

private void ConnectPorts_Click(object sender, RoutedEventArgs e)
{
    string port = PortsList.SelectedValue.ToString();
    serialHelper.initializePorts(port);
}

private void ServoButton_Click(object sender, RoutedEventArgs e)
{
    if (isSoftwareMode != true)
    {
        MessageBox.Show("Enable Software Mode");
        return;
    }
    if (serialHelper.PortStatus == "Connected")
    {
        string serialData;
        string init = "AP00";

        string temp = ServoMotorPOS.Text.ToString();

        if (Convert.ToInt32(temp) < 0 || Convert.ToInt32(temp) > 180)
        {
            MessageBox.Show("Value needs to be in between 0-180");
            return;
        }
        else
        {
            if (temp.Length == 1)
            {
                init = init + "00";
            }
            else if (temp.Length == 2)
            {
                init = init + "0";
            }
            init = init + temp;
        }

        init = init + "0K";

        serialData = init;
        serialHelper.WriteData(serialData);
    }
```

```
        else
        {
            MessageBox.Show("Port not connected");
        }
    }

    private void StepperButton_Click(object sender, RoutedEventArgs e)
    {
        if (isSoftwareMode != true)
        {
            MessageBox.Show("Enable Software Mode");
            return;
        }
        if (serialHelper.PortStatus == "Connected")
        {
            string serialData;
            string init = "AP1";
            if (StepperToggle.Content.ToString() == "FWD")
            {
                init = init + "0";
            }
            else
            {
                init = init + "1";
            }

            string temp = StepperMotorDegree.Text.ToString();

            if (temp.Length == 0 || temp.Length > 3)
            {
                MessageBox.Show("Value needs to be in between 1-3 digits");
                return;
            }
            else
            {
                if (temp.Length == 1)
                {
                    init = init + "00";
                }
                else if (temp.Length == 2)
                {
                    init = init + "0";
                }
                init = init + temp;
            }

            init = init + "0K";

            serialData = init;
            serialHelper.WriteData(serialData);
        }
        else
        {
            MessageBox.Show("Port not connected");
        }
    }

    private void DCButton_Click(object sender, RoutedEventArgs e)
    {
        if (isSoftwareMode != true)
        {
            MessageBox.Show("Enable Software Mode");
            return;
        }
        if (serialHelper.PortStatus == "Connected")
        {
            string serialData;
            string init = "AP2";
            if (DCDirToggle.Content.ToString() == "FWD")
```

```
            {
                init = init + "0";
            }
            else
            {
                init = init + "1";
            }

            if (DCModeToggle.Content.ToString() == "POS")
            {
                string temp = DCMotorPOS.Text.ToString();

                if (temp.Length == 0 || temp.Length > 3)
                {
                    MessageBox.Show("Value needs to be in between 1-3 digits");
                    return;
                }
                else
                {
                    if (temp.Length == 1)
                    {
                        init = init + "00";
                    }
                    else if (temp.Length == 2)
                    {
                        init = init + "0";
                    }
                    init = init + temp;

                    init = init + "0K";
                }
            }
            else
            {
                string temp = DCMotorPWM.Text.ToString();

                if (temp.Length == 0 || temp.Length > 3)
                {
                    MessageBox.Show("Value needs to be in between 1-3 digits");
                    return;
                }
                else
                {
                    if (temp.Length == 1)
                    {
                        init = init + "00";
                    }
                    else if (temp.Length == 2)
                    {
                        init = init + "0";
                    }
                    init = init + temp;
                    init = init + "1K";
                }
            }

            serialData = init;
            serialHelper.WriteData(serialData);
        }
        else
        {
            MessageBox.Show("Port not connected");
        }
    }

    private void StepperToggle_Click(object sender, RoutedEventArgs e)
    {
        if (StepperToggle.Content.ToString() == "FWD")
        {
```

```
                StepperToggle.Content = "BWD";
            }
            else
            {
                StepperToggle.Content = "FWD";
                StepperToggle.Background = Brushes.Green;
            }
        }

        private void DCDirToggle_Click(object sender, RoutedEventArgs e)
        {
            if (DCDirToggle.Content.ToString() == "FWD")
            {
                DCDirToggle.Content = "BWD";
            }
            else
            {
                DCDirToggle.Content = "FWD";
                DCDirToggle.Background = Brushes.Green;
            }
        }

        private void DCModeToggle_Click(object sender, RoutedEventArgs e)
        {
            if (DCModeToggle.Content.ToString() == "POS")
            {
                DCModeToggle.Content = "VEL";
            }
            else
            {
                DCModeToggle.Content = "POS";
                DCModeToggle.Background = Brushes.Green;
            }
        }

        private void ToggleSwMode_Click(object sender, RoutedEventArgs e)
        {
            isSoftwareMode = false;
            string serialData = "AP600000K";
            serialHelper.WriteData(serialData);
        }

        private void PotMode_Click(object sender, RoutedEventArgs e)
        {
            isSoftwareMode = false;
            string serialData = "AP500000K";
            serialHelper.WriteData(serialData);
        }

        private void SharpIRMode_Click(object sender, RoutedEventArgs e)
        {
            isSoftwareMode = false;
            string serialData = "AP300000K";
            serialHelper.WriteData(serialData);
        }

        private void SoftwareMode_Click(object sender, RoutedEventArgs e)
        {
            string serialData = "AP700000K";
            serialHelper.WriteData(serialData);
            isSoftwareMode = true;
        }

    }

}
```