MANUAL: DEVICE FOR LINEAR POSITION CONTROL

INTRODUCTION

The system serves the purpose of practicing on linear position control. A full program of PID control is provided with it as an examle. User can also write a program and download to the controller and test the system.

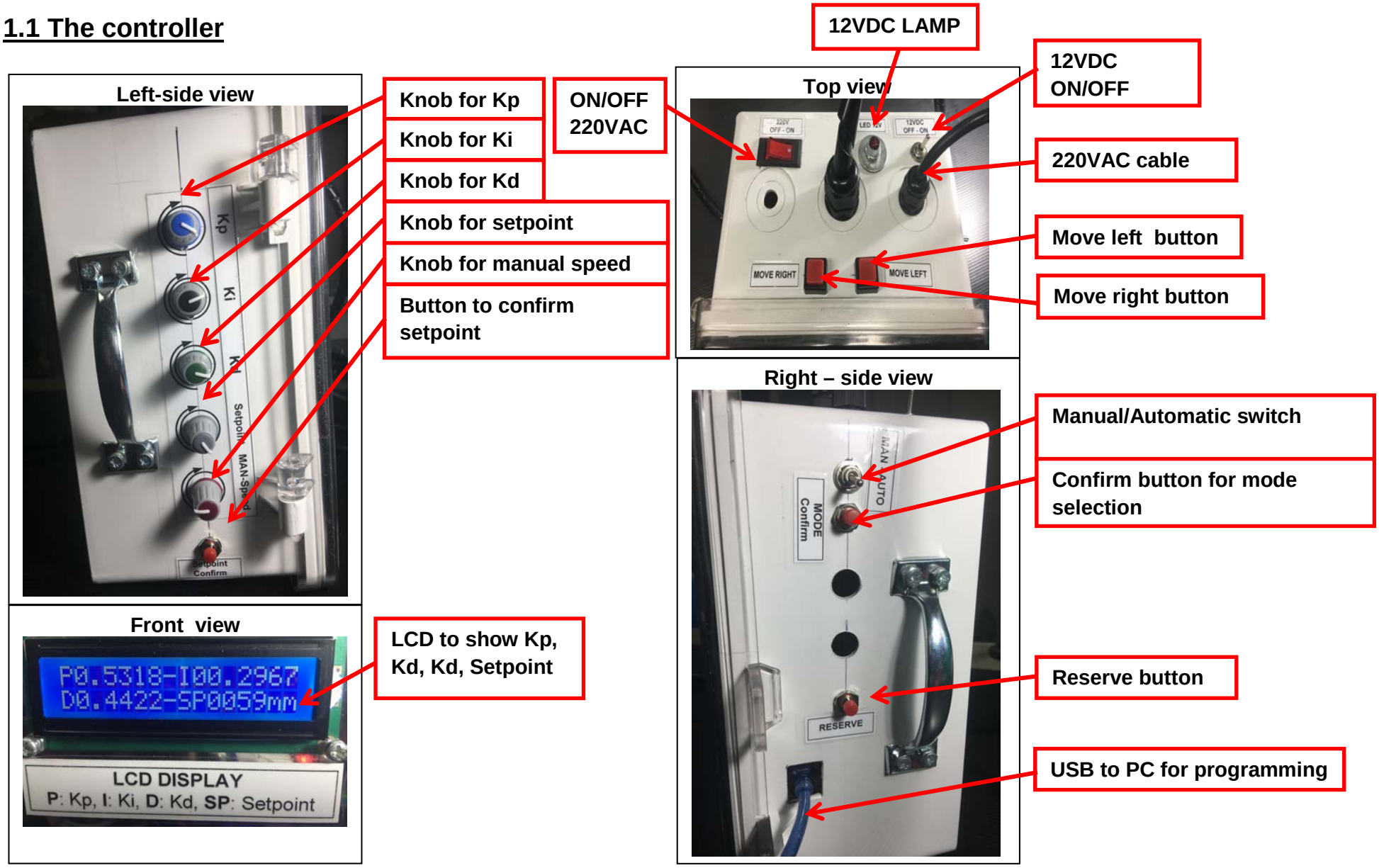
The system has 02 mode: Manual and Auto. At manual mode, the user can set manual speed and move the motor by buttons. The system can be homed automatically by pressing Setpoint Confirm and Mode Confirm buttons.

At automatic mode, the user will set the setpoint and the motor will automatically move the system to the target. The parameters Kp, Ki, Kd can be adjusted from the panel (by turning knobs).

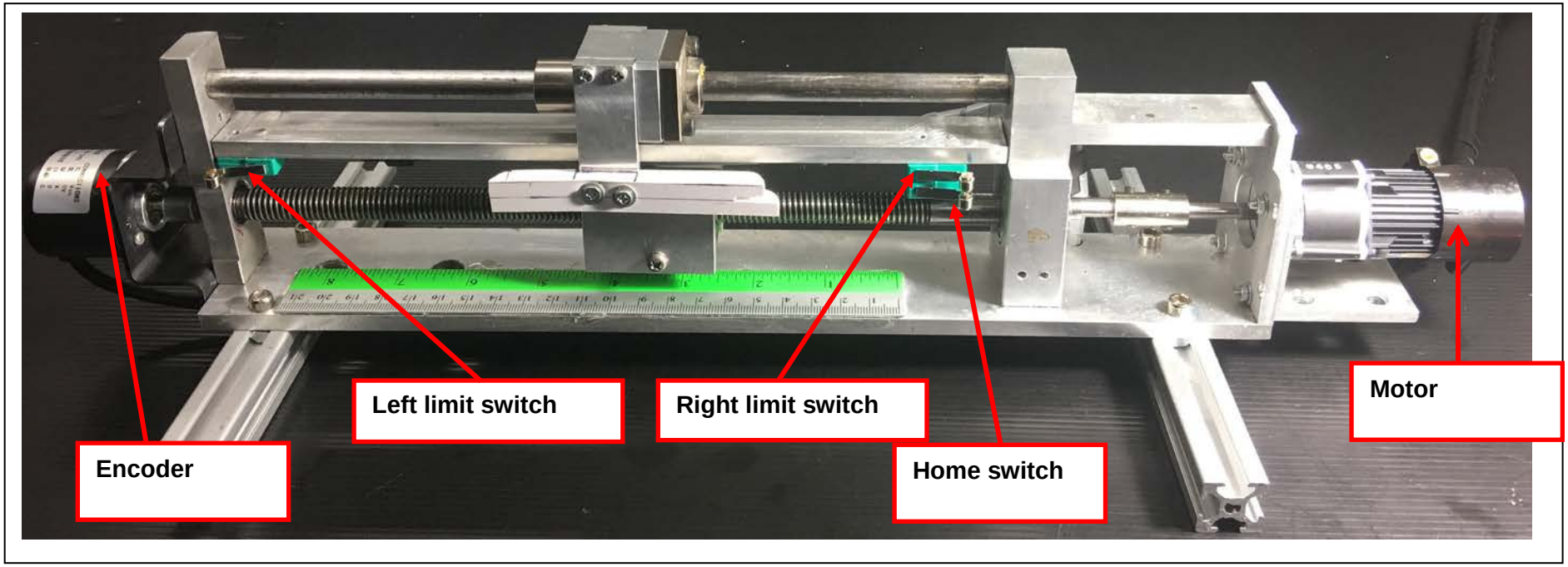
The system can work alone. If the user needs to observe the graph of process variable, control variable, etc. then the programming IDE Arduino needs to be installed (download from *http://arduino.cc/en/Main/Software*).

I. THE SYSTEM DESCRIPTION AND OPERATION

1.1 The controller



1.2 The mechanism



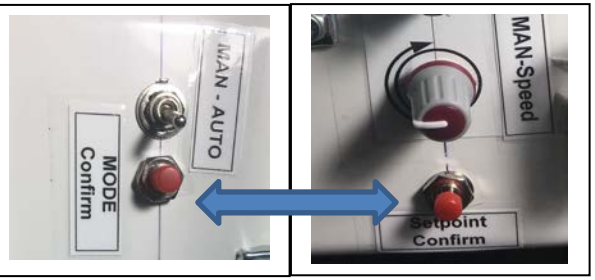
1.3 Operation

1.3.1 Power on for testing the sytem without PC

STEP 1. Plug power cable into 220VAC.
STEP 2. Turn on the main switch for 220VAC. The lamp of the switch is on.
STEP 3. Turn on the switch for 12VDC. The 12 VDC lamp is on.

Power on for testing the sytem with PC

Running the Arduino programming software V1.8.5. and plug the USB cable into the PC then. Press **Ctrl + Shift + L** for ploter.
STEP 1. Plug power cable into 220VAC.
STEP 2. Turn on the main switch for 220VAC. The lamp of the switch is on.
STEP 3. Turn on the switch for 12VDC. The 12 VDC lamp is on.

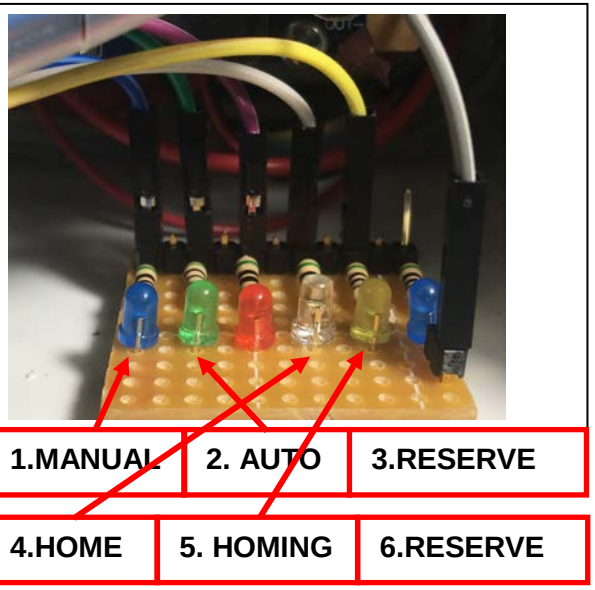


1.3.2 Manual Mode

STEP 4. Turn the mode selection switch to **MAN position** and press the **MODE Confirm button** to confirm the mode change. The **MANUAL LED** will be ON.

STEP 5. Pressing at the both **MODE Confirm button** & **Setpoint Confirm button** for automatic homing. During homing, the **HOMING LED** will be flashed. When being at home, the home switch will be activated and the **HOMED LED** is ON and stay ON showing that the home position is defined.

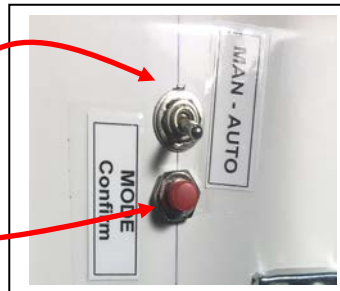
The user can adjust the manual speed knob to change the motor speed and then using either MOVE LEFT or MOVE RIGHT. If using the PC, the user can observe the plot of process variable, the control variable from the plots. The PID parameters can be changed by coresponding knobs and observe the value from the LCD.

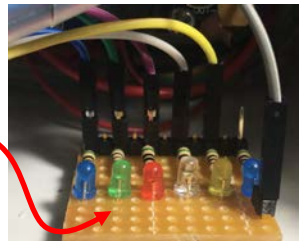


1.3.3 Automatic Mode

After homing, the system can run at automatic mode (PID control).If the user has a PC with Arduino programing software then run it and press Ctrl + Shift + L for serial plot.

STEP 6. Turn the switch to AUTO position and press the confirm button to change the system to automatic mode. At this time, the GREEN led should be ON.

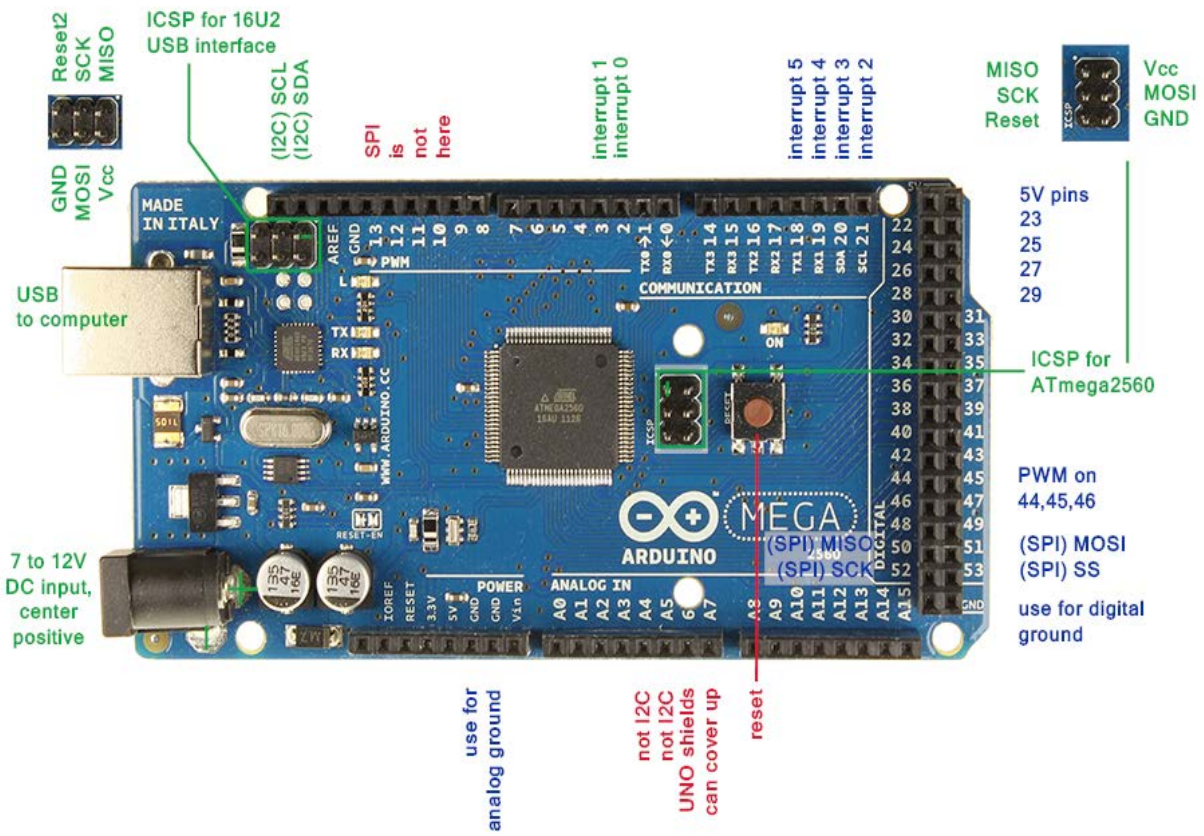




STEP 7. Change the setpoint knob to the desired position (LCD shows it) and press the setpoint confirm button. The motor will move the disk to that position.

STEP 8. The user can observe the plots of setpoint, process variable, and the control variable. PID parameters Kp, Ki, Kd can be adjusted and the change can be reflected from the plot. BLUE => setpoint, RED=> process variable, GREEN => control variable.

II. MICRO-CONTROLLER: ARDUINO MEGA 2560 V3.



2.1 Power Pins

VIN: The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

5V: This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

3V3: A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

Brand	Name (link)	Processor	Operating Voltage	Input Voltage Range	CPU Speed	Analog In	Digital IO	PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	SERIAL UART
Arduino	Mega 2560	ATmega2560	5 V	7 -12 V	16 MHz	16	54	15	4	8	256	Regular	4

2.2. Input And Output

Each of the 54 digital pins and 16 analog pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. The analog pins will sample voltage levels and report a digital value using analogRead(). Analog pins and a subset of the digital pins do Pulse Width Modulation (PWM) using analogWrite(). Each pin operates at 0 to 5 volts, can provide or receive an absolute maximum of 40 mA (20mA recommended), and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms.

2.3 Specialized Functions

Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX). Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the USB-to-TTL Serial chip.

External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

PWM: 2 to 13 and 44 to 46, provide 8-bit PWM output with the analogWrite() function.

SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.

LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

I2C/TWI : 20 (SDA) and 21 (SCL). Support I2C aka IIC aka TWI communication using the Wire library. Note that these pins are not in the same location as the TWI/I2C pins on the Duemilanove or Diecimila.

2.4 Analog And Other Pins

The Mega has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

All analog pins do analog output using PWM. You can't set an analog pin to a specific voltage; PWM just adjusts the duty cycle of a square wave alternating between 0v and +5v so that the average over time appears between 0 and +5v. The Analog input pins can also be used as Digital Input or Output pins. Just use **A0** through **A15** for a pin name where you would use a digital pin number.

AREF: Reference voltage for the analog inputs. Used with analogReference().

RESET: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

2.5 Pin Connections

A. LCD connecitons

```
LCD_RS = 53;
LCD_E = 51;
LCD_DB4 = 49;
LCD_DB5 = 47;
LCD_DB6 = 45;
LCD_DB7 = 43;
```

B. Encoder connections

```
ENCODER_PINA = 3;//pin for encoder's channel A.
ENCODER_PINB = 2;//pin for encoder's channel B.
```

C. Motor driver connections

```
ENA = 7;//PWM to driver.
DIR_RIGHT = 36;//DIR_RIGHT=1,DIR_LEFT=0, CW
rotation.
DIR_LEFT =38;//DIR_RIGHT=0,DIR_LEFT=1, CCW rotation.
```

D. Limit switches connections

```
LMS_LEFT = 40;//left limit switch.
LMS_RIGHT = 42;//right limit switch.
```

E. Button connections

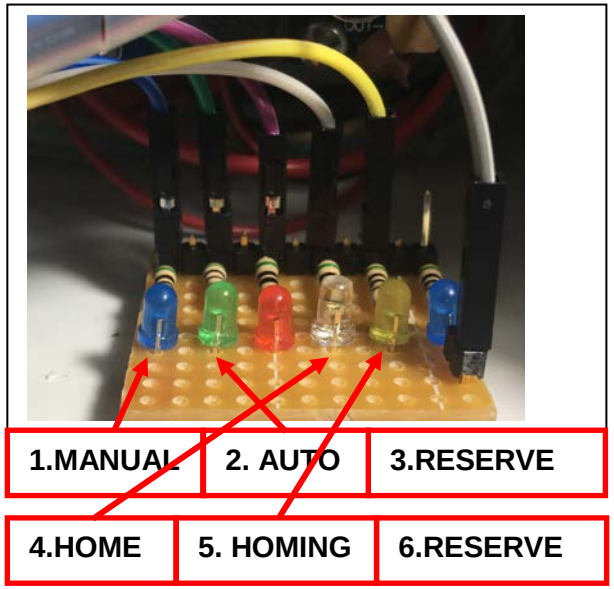
```
BUTTON_LEFT = 22;
BUTTON_RIGHT = 24;
BUTTON_CONFIRM = 26;//confirm mode change.
BUTTON_MODE = 28;//mode: FALSE = MAN, TRUE = AUTO.
SWITCH_HOME = 30;//home limit switch.
BUTTON_SET = 32;//press to confirm set point.
BUTTON_RES = 34;//reserve button.
```

F. LED connections

```
(LED1)LED_MAN = 25;//on at manual mode
(LED2)LED_AUTO = 27;//on at auto mode.
(LED3)LED_LIMITS = 29;//reserved.
(LED4)LED_HOME = 31;//on if homed.
(LED5)LED_AUTOHOME = 33;//flashing if homing.
(LED6)RESERVED is not connected.
```

G. Knobs connections

```
Kp = analogA1);
Ki = analog(A2);
Kd = analog(A3);
Setpoint = analog(A4);
manualSpeed = analog(A5);
```



III. PROGRAMMING SOFTWARE

3.1 Software

The Arduino Conroller can be programmed with the open-source Arduino software. The programming platform enables you to write and edit your program. It can cross compile your code for Atmega2560. It can also upload your program using the same software.

The *μ*C on the Arduino comes with a *pre-burned boot-loader*, which allows uploading new code to it without the use of an external hardware programmer. Instead of using a physical "Reset" button the Arduino is designed in a way that allows it to be reset by software running on a connected computer.

The Arduino programs can be categorized in 3 parts: structure, values and functions. For details on programming please refer to **http://arduino.cc/en/Reference/HomePage**. The basic structure of the programming language is simple. It will always contain 2 parts/functions, one is the setup () function, which is the preparation part. The other is the loop () function which is the execution part. Therefore a simple sketch of your program will look like as:

```
void setup(){
\\statements  here
}
void  loop(){
\\statements  here
}
```

3.2. Arduino Software Platform Installation

- Steps to install Arduino programming platform on our laptop and to communicate it with the Arduino.
- Go to ***http://arduino.cc/en/Main/Software*** to choose and download the compressed packaged based on your Operating System.
- Based on your operating system selection a zipped file will be downloaded on your machine.
- Run ./arduino shell script on your terminal and it will open the Arduino platform on your machine.
- Connect one end of Arduino board USB cable with your machine
- You are now ready to write, compile and upload your program for Arduino board.

IV. ENCODERS

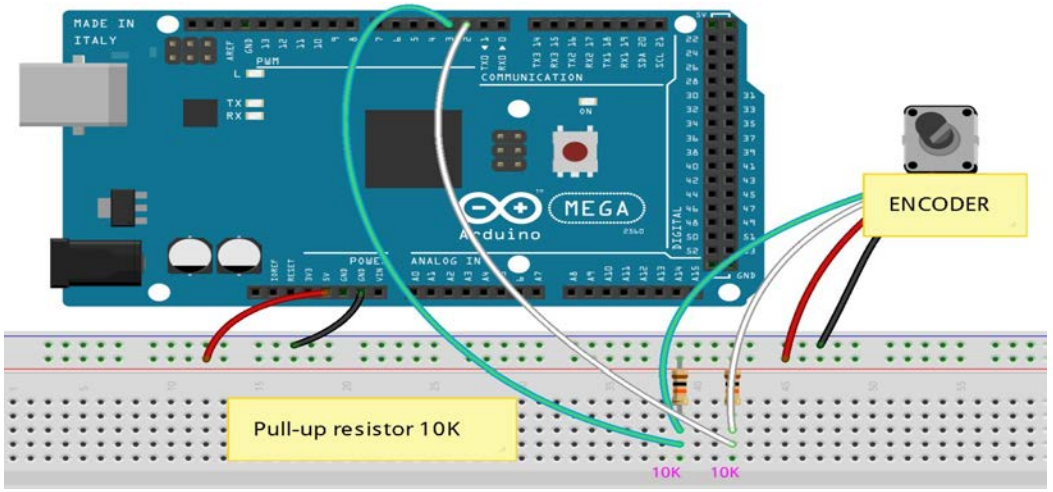
4.1. Rotary Incremental Optical Encoders

Since rotary incremental optical encoders are used most widely in industries. In this part we will go into details about how to integrate this kind of encoder to a microcontroller and decode the given information to get position. The index pulse is normally used for position reference or homing. In case the encoder doesn't have index pulse then one input to the controller will be used as home reference. When this input is activated at homing mode, the counter will be initialized to zero or to a specific number of pulses so that the system knows where to start from.

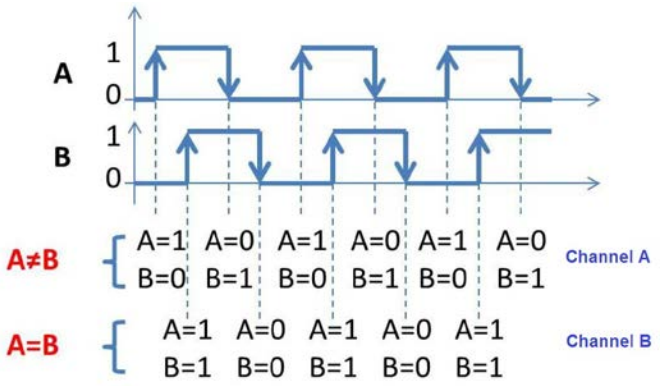
4.2. Encoder type: LPD3806-600BM-G5-24C

- AB two-phase incremental optical rotary encoder, NPN open collector output .
- Performance: 600 pulses / rev.
- Operating voltage: DC5-24V
- Maximum mechanical speed: 5000 rev / min
- The electrical response frequency: 20K / sec
- Integrated speed: 2000 rev / min

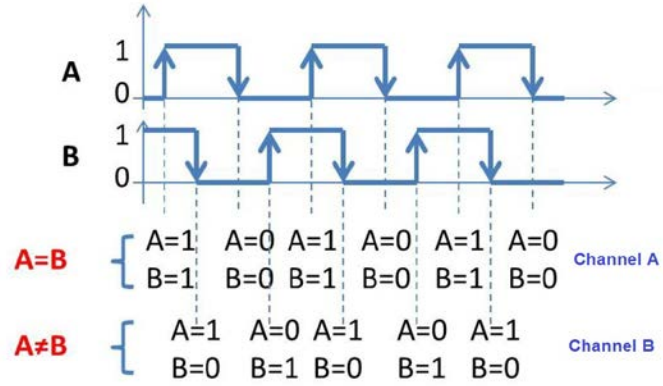
- Output: AB rectangular two-phase quadrature pulse output circuit output NPN open collector output type, this output type can and with internal pull-up resistor connected directly to the microcontroller (open collector output when there is no pull-up resistor, there is no voltage output)
- Connection: Green = A phase; White = B phase; Red = VCC positive supply; Black = GND



4.3. Algorithms for Position Calculation



a) Clockwise Rotation



b) Counter Clockwise Rotation

4X decoding : If an encoder comes with a specification of 600 PPR (or sometimes CPR) meaning that a channel will generates 600 pulses per revolution then the counter number will be increased / decreased by 2400 per revolution. Following is the algorithm to determine the encoder's rotating direction:

```
If A has a transition either from 0 – 1 or 1 – 0
then
    If A ≠ B then { position = position + 1;}
//clockwise rotation
    If A = B then { position = position – 1;}
//counter clockwise rotation

If B has a transition either from 0 – 1 or 1 – 0
then
    If A = B then {position = position + 1;}
//clockwise rotation
    If A ≠ B then {position = position – 1;} //counter
clockwise rotation
```

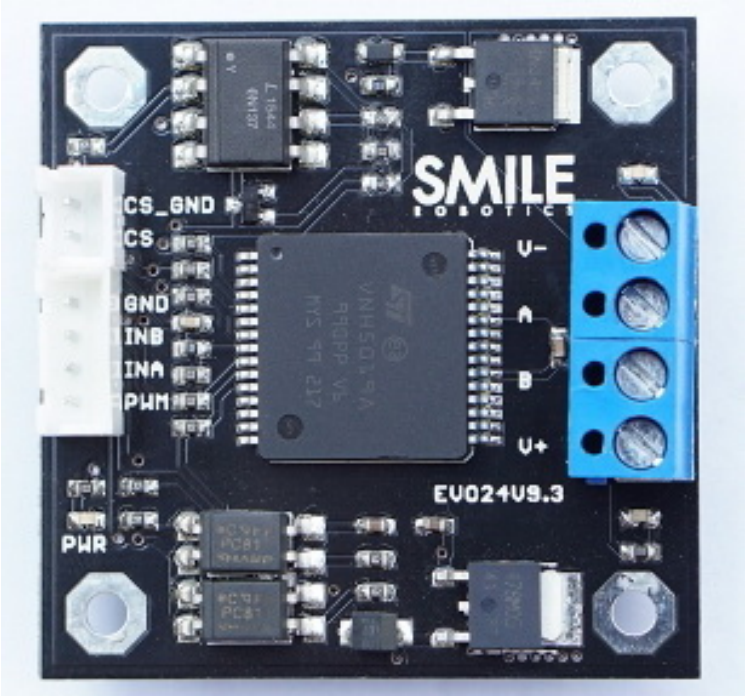
Both channels will be connected to interrupt pin of the controller. In our case, channel A => pin 3, channel B => pin 2. The following program is used to read pulses from the encoder and store them in a variable.

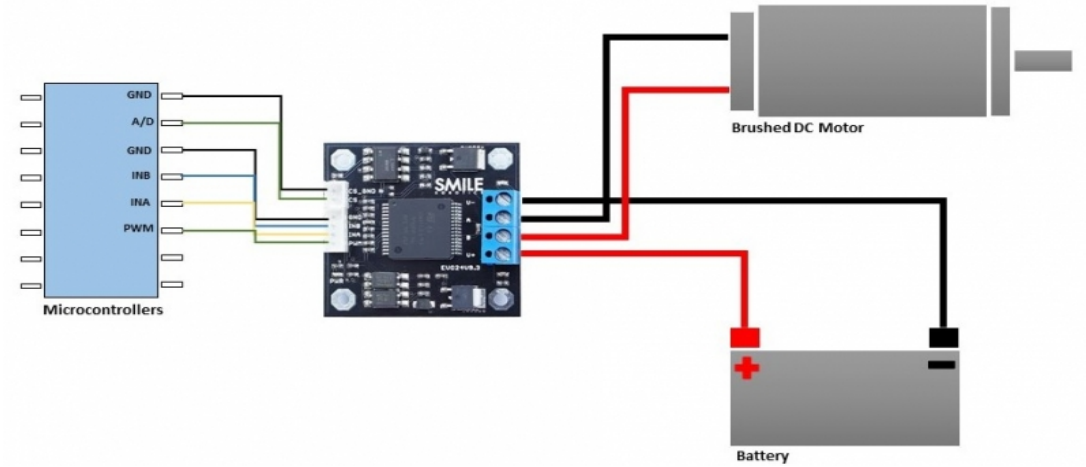
There are 02 functions and they will be implemented once there is an interrupt from the pin.

The variable currentPosition is used to store the number of pulses and must be declared as volatile. Two channel pins are set up as INPUT. Once there is an interrupt (ENCODER_PINA or ENCODER_PINB changes status from 0 to 1 or 1 to 0. The variable will be updated. This variable will be used in side the main program to convert to the angle position.

```
const int ENCODER_PINA = 3;//pin for encoder's channel A.
const int ENCODER_PINB = 2;//pin for encoder's channel B.
volatile int currentPosition = 0;//number of pulses at time of calculate

void setup(){
pinMode(ENCODER_PINA, INPUT);//set pin ENCODER_PINA as input.
pinMode(ENCODER_PINB, INPUT);//set pin ENCODER_PINB as input.
attachInterrupt(0, doEncoderB, CHANGE);//set up ISR for interrupt 0 - pin 2
attachInterrupt(1, doEncoderA, CHANGE);//set up ISR for interrupt 1 - pin 3.
}
void loop() {
}
/*=============================================================================*/
/*INTERRUPT SUBROUTINE CHANNEL A: CW OR CCW BELLOWING ARE OF MOTOR, OPPOSITE TO THE ENCODER.
When an interrupt on channel A occurs then: CW rotation: A != B, CCW rotation: A = B.8*/
void doEncoderA() {

  if (digitalRead(ENCODER_PINA) == digitalRead(ENCODER_PINB)){
    currentPosition++;}
  else {currentPosition--;}}
/*=============================================================================*/
/*INTERRUPT SUBROUTINE CHANNEL B
When an interrupt on channel B occurs then: CW rotation: A = B, CCW rotation: A != B.*/
void doEncoderB() {
  if (digitalRead(ENCODER_PINA) != digitalRead(ENCODER_PINB)){
    currentPosition++;}
  else {currentPosition--;}}
```

V. MOTOR DRIVER (EVO24V9.3) (http://smile-robotics.com)

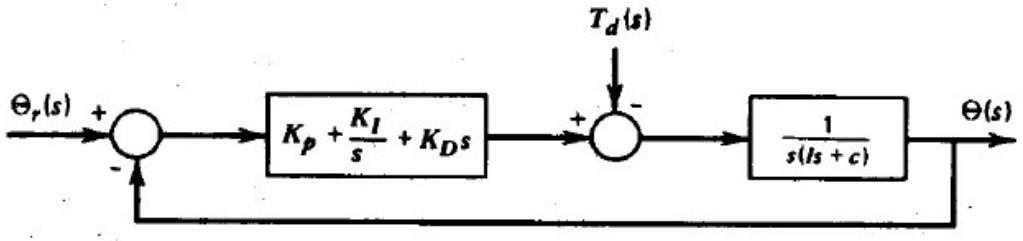


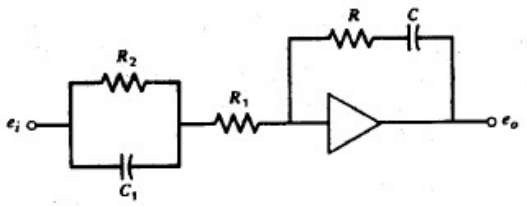


EVO24V9.3 is a motorized brush motor type IC bridge full bridge motor. For small motors up to 150 W, it can drive up to 10 A continuous current and 30 A at a maximum voltage of 24 V. Using opto-Isolator to prevent potential interference from the motor. And it can also prevent reverse currents to the control sector. It also has protection systems such as protection against polarity. The system is cut off when the temperature is too high. And the system can prevent damage if the main control system is a problem. Motor speed control by adjusting the width of the PWM signal at a frequency not exceeding 10 kHz.

PIN	Description
V-	Ground (Ground) of power supply such as Battery or Power Supply
A	Connector to Motor Line 1
B	Connector to Motor Line 2
V+	Anode of the power supply such as Battery or Power Supply, which has a voltage in the range of 7-28VDC.
GND	Ground of Control Signal
INB	Directional Logic Signal Rotate direction 1
INA	Directional Logic Signal Rotate direction 2
PWM	Pulse Width Modulation Connection
CS	Output signal for load current measurement
CS_GND	Ground for measuring the load current

VI. PID ALGORITHM





$$e_o = -\left(K_p e_i + K_I \int_0^t e_i dt + K_D \frac{de_i}{dt}\right) - \beta R_1 C_1 \frac{de_o}{dt}$$

$$\beta = \frac{R_2}{R_1 + R_2} \qquad K_p = \beta \frac{RC + R_2 C_1}{R_2 C}$$

$$K_I = \frac{\beta}{R_2 C} \qquad K_D = \beta R C_1$$

Figure 6.6-1 Simplified Form of Position Control System Using PID Control

108 Manukid Parnichkun

The algorithm and parameters were developed based on the lecture notes from Prof. Manukid Parnichkun as seen from the above picture.

$$lmn = Kp * ER + Ki \int ERdt + Kd * \frac{dER}{dt}$$

$$lmn = lmnP + lmnI + lmnD$$

$$lmnP = Kp * ER$$

$$lmnI = Ki \int ERdt$$

$$lmnD = Kd * \frac{dER}{dt}$$

Where,

lmn = loop manipulated variable or controller output signal.
lmnP = proportional portion.
lmnI = integral portion.
lmnD = derivative portion.
ER = current controller error, defined as setPoint – processVariable.
Kp = proportional gain, a tuning parameter (no unit).
Ki = Integral gain, a tuning parameter (unit 1/s)
Kd = derivative gain, a tuning parameter (unit is s).
setPoint = set point.
processVariable = measured process variable.

The PID forms written above are in continuous form and used by analog controller. Using microcontroller, forms should be converted into digital type. In microcontroller, the calculation will be done at fixed interval called CYCLE.

6.1 PID Portions

- **Proportional Portion**

ER = setPoint - processVariable
lmnP = Kp * ER

- **Integral Portion**

lmnI = $Ki \int ERdt$
d(lmnI)/dt = Ki * ER
d(lmnI) = Ki * ER * dt
lmnI – lastLmnI = ER * Ki * dt
lmnI = lastLmnI + ER * Ki * dt
lmnI = lastLmnI + ER * Ki * CYCLE

- **Derivative Portion**

lmnD = $Kd * \frac{dER}{dt}$ = (Kd/dt) * d(ER)
lmnD = (Kd/dt) * (ER – lastER)
lmnD = (Kd/dt) * (ER – lastER)
lmnD = (Kd/CYCLE) * (ER – lastER)

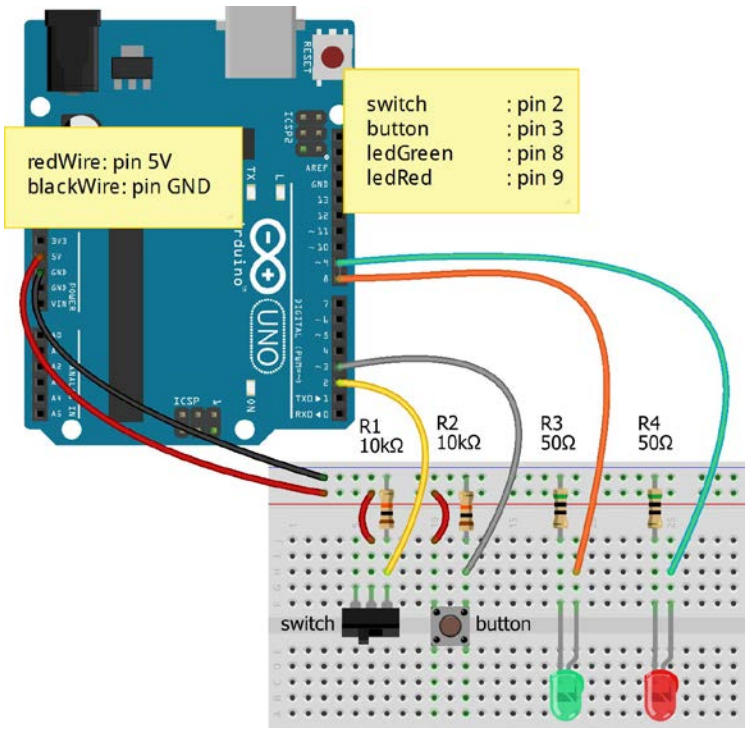
Where,
lastLmnI : *Integral portion at the last interval.*
lastER : *Error at the last interval.*

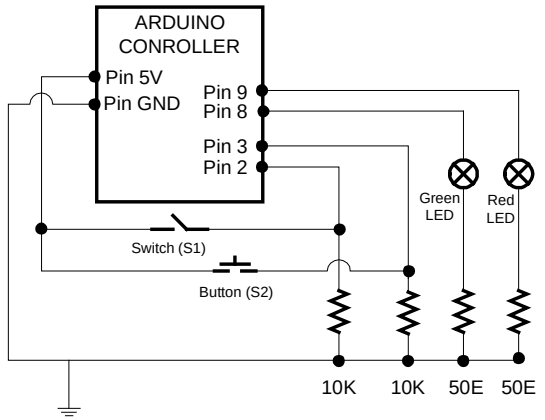
The PID computation is written as a function and it will be called in the main program at interval CYCLE. Two variables lastError and lmnI must be declared as static so their value will be remained after the cycle.

```
static double lastError;//last value of error.
static double lmnI;//integral portion. Static vars for remembering last value.
```

```
void PID()
 {
  static double lastError;//last value of error.
  double lmn;//loop manipulated variable
  double lmnP;//proportional portion.
  double lmnD;//derivative portion.
  double integralDiff;//Integral difference
  static double lmnI;//integral portion. Static vars for remembering last value.

  currentError = setPoint - processVariable;
  //Proportional portion calculation
  lmnP = Kp * currentError;

  //Integral portion calculation.
  if (((integralDiff > 0) && (lmn > HIGH_LIMIT)) || ((integralDiff < 0) && (lmn < LOW_LIMIT))){
        integralDiff = 0.0;}
  else { integralDiff = currentError * CYCLE * Ki;}
  if (currentError == 0.0) {lmnI = 0;}
  else {lmnI += integralDiff;}
  lmnIntegral = lmnI;
  //Derivative portion calculation.
  lmnD     = (Kd/CYCLE) * (currentError - lastError);
  lastError = currentError;//update error.
  lmn      = lmnP + lmnI + lmnD;//sum of portions

  //Safety limit
  if ((lmn > HIGH_LIMIT) && (lmnI > HIGH_LIMIT)){lmnI = lmnI - (lmn - HIGH_LIMIT);}
  if ((lmn < LOW_LIMIT) && (lmnI < LOW_LIMIT)){lmnI =  lmnI + (LOW_LIMIT - lmn);}
  if (lmn > HIGH_LIMIT) {lmn = HIGH_LIMIT;}
  if (lmn < LOW_LIMIT) {lmn = LOW_LIMIT;}
  if (currentError == 0) {controlVariable = 0;} else {controlVariable = lmn;}
  }//END PID
```

VII. BUTTONS AND LED CONNECTIONS TO THE CONTROLLER.





A button needs a resistor (10K is OK) to connect to GROUND so that when it is not activated the ZERO will go to input. It is because the microcontroller inputs are floating and can pick some voltage level and can be interpreted as HIGH logic.

VIII. EXAMPLE OF A PID CONTROL PROGRAM

```
/*=============================================================================================*/
/*PROGRAM INTRODUCTION
Motor speed control by PID controller. //Both channel A and B of the encoder are connected to
interrupt 0(pin3) and interrupt 1 (pin3) to calculate the pulses. The formula is as belowing:
processVariable = currentPosition * FEED_CONSTANT/NUMBER_PULSE_REVERLUTION. FEED_CONSTANT is 2 mmm.
The number of pulses for a revolution is 600x4 = 2400 (taking both rising and falling edges of both channels).
Constants and parameters are denoted by capital letters. E.g.: ENCODER_PINA = 3.
HHManh, Mechatronics, AIT Thailand.*/
/*=============================================================================================*/
/*CONSTANTS AND GLOBAL VARIABLE DECLARATION*/
#include <LiquidCrystal.h> // include the library code.
// initialize the library with the numbers of the interface pins
const int LCD_RS = 53;
const int LCD_E = 51;
const int LCD_DB4 = 49;
const int LCD_DB5 = 47;
const int LCD_DB6 = 45;
const int LCD_DB7 = 43;
const int ENCODER_PINA = 3;//pin for encoder's channel A.
const int ENCODER_PINB = 2;//pin for encoder's channel B.
const int ENA = 7;//PWM to driver.
const int DIR_RIGHT = 36;//DIR_RIGHT=1,DIR_LEFT=0, CW rotation.
const int DIR_LEFT =38;//DIR_RIGHT=0,DIR_LEFT=1, CCW rotation.
const int LMS_LEFT = 40;//left limit switch.
const int LMS_RIGHT = 42;//right limit switch.
const int BUTTON_LEFT = 22;
const int BUTTON_RIGHT = 24;
const int BUTTON_CONFIRM = 26;//confirm mode change.
const int BUTTON_MODE = 28;//mode: FALSE = MAN, TRUE = AUTO.
const int SWITCH_HOME = 30;//home limit switch.
const int BUTTON_SET = 32;//press to confirm set point.
const int BUTTON_RES = 34;//reserve button.
const int LED_MAN = 25;//on at manual mode
const int LED_AUTO = 27;//on at auto mode.
const int LED_LIMIT = 29;//on at limits.
const int LED_HOME = 31;//on if homed.
const int LED_AUTOHOME = 33;//flashing if homing.
const double HIGH_LIMIT = 255.0;//high limit.
const double LOW_LIMIT = -255.0;//low limit.
```

```
double floatKp = 0;
double floatKi = 0;
double floatKd = 0;
int knobSetpoint = 0;
int knobSpeed = 0;

/*variables used for PID*/
double setPoint = 50;//setting speed at manual mode.
double controlVariable = 0;
double processVariable = 0.0;//current angle
double currentError = 0;
double lmnIntegral = 0;
int manualValue = 50;
double Kp;//proportional gain.
double Ki;//intergral gain.
double Kd;//derivative gain.
double CYCLE = 0.05;//PID processing interval.
boolean manMode = false;
boolean autoMode = false;

//vars in Interrupt Subroutine need to be volatile.
volatile long currentPosition = 0;//number of pulses at time of calculate
int buttonMode, buttonConfirm, buttonRight, buttonLeft, buttonSet, flashBit;
int lmsLeft, lmsRight, switchHome, homeSet, autoHome;
double readKp, readKi, readKd, readSetpoint, readSpeed;
double EMA_a = 0.6;
int EMA_S = 0;
unsigned long lastLCDTime = 0;
unsigned long lastFlashingTime = 0;
unsigned long lastSerialTime = 0;
unsigned long lastPIDTime = 0;//to store the real-time when previous execution is done.
LiquidCrystal lcd(LCD_RS, LCD_E, LCD_DB4, LCD_DB5, LCD_DB6, LCD_DB7);

/*=============================================================================================*/
void setup() {
  //TCCR4B = (TCCR4B & 0xF8) | 0x01; //change PWM frequency to 31.374Kz (no noise)
  Serial.begin(9600);//serial speed 9600kb/s.
  pinMode(BUTTON_MODE, INPUT);
  pinMode(BUTTON_RIGHT, INPUT);
  pinMode(BUTTON_LEFT, INPUT);
  pinMode(BUTTON_CONFIRM, INPUT);
  pinMode(BUTTON_SET, INPUT);//to confirm setting Kp, Ti, Kd, Setpoint.
  pinMode(LMS_LEFT, INPUT);
  pinMode(LMS_RIGHT, INPUT);
  pinMode(BUTTON_RES, INPUT);
```

```
  pinMode(SWITCH_HOME, INPUT);
  pinMode(ENCODER_PINA, INPUT);//set pin ENCODER_PINA as input.
  pinMode(ENCODER_PINB, INPUT);//set pin ENCODER_PINB as input.

  pinMode(LED_AUTO, OUTPUT);//set pin LED_PIN as output.
  pinMode(LED_MAN, OUTPUT);
  pinMode(LED_HOME, OUTPUT);
  pinMode(LED_LIMIT, OUTPUT);
  pinMode(LED_AUTOHOME, OUTPUT);//On when set button is pressed.
  pinMode(DIR_RIGHT, OUTPUT);//set pin DIR1 as output.
  pinMode(DIR_LEFT, OUTPUT);//set pin DIR2 as output.
  pinMode(ENA, OUTPUT);//set pint ENA as output
  digitalWrite(DIR_RIGHT, LOW);
  digitalWrite(DIR_LEFT, LOW);
  digitalWrite(LED_HOME, LOW);
  //FilterOnePole lowpassFilter(LOWPASS, 5.0);
  attachInterrupt(0, doEncoderB, CHANGE);//set up ISR for interrupt 0 - pin 2.
  attachInterrupt(1, doEncoderA, CHANGE);//set up ISR for interrupt 1 - pin 3.
  lcd.clear();
  lcd.begin(16, 2);// set up the LCD's number of columns and rows.
  lcd.print("Linear position control");// Print a message to the LCD.
 }
/*=============================================================================================*/
void loop() {

/*Reading inputs and conversion*/
readKp = analogRead(A1);
readKi = analogRead(A2);
readKd = analogRead(A3);
readSetpoint = analogRead(A4);
readSpeed = analogRead(A5);
// Conversion from analog read.
floatKp = fmap(readKp, 0, 1023.0, 0.0, 2.0);
floatKi = fmap(readKi, 0, 1023.0, 0.0, 2.0);
floatKd = fmap(readKd, 0, 1023.0, 0.0, 2.0);
knobSetpoint = map(readSetpoint, 0, 1023, 0, 160);
knobSpeed = map(readSpeed, 0, 1023, 0, 255);
Kp = floatKp; Ki = floatKi; Kd = floatKd;

buttonMode = digitalRead(BUTTON_MODE);
buttonConfirm = digitalRead(BUTTON_CONFIRM);
buttonRight = digitalRead(BUTTON_RIGHT);
buttonLeft = digitalRead(BUTTON_LEFT);
buttonSet = digitalRead(BUTTON_SET);
lmsLeft = digitalRead(LMS_LEFT);
```

```
lmsRight = digitalRead(LMS_RIGHT);
switchHome = digitalRead(SWITCH_HOME);



/*LCD displays resutls from knobs*/
if (millis()-lastLCDTime > 500){
lcd.setCursor(0, 0); lcd.print("P"); lcd.print(floatKp, 4);
lcd.print("-I");
if (floatKi < 10) {lcd.print("0"); lcd.print(floatKi,4);}
else {lcd.print(floatKi, 4);}
lcd.setCursor(0, 1); lcd.print("D"); lcd.print(floatKd, 4); lcd.print("-SP");
if (knobSetpoint < 10) {lcd.print("000"); lcd.print(knobSetpoint); lcd.print("mm");}
else if ((knobSetpoint > 9) && (knobSetpoint < 100)) {lcd.print("00"); lcd.print(knobSetpoint); lcd.print("mm");}
else if ((knobSetpoint > 99) && (knobSetpoint < 1000)) {lcd.print("0"); lcd.print(knobSetpoint); lcd.print("mm");}
else {lcd.print(knobSetpoint); lcd.print("mm");}
lastLCDTime = millis();}

if (millis() - lastFlashingTime > 500){flashBit = !flashBit; lastFlashingTime = millis();}//for flashing.

/*Mode change*/

if ((buttonMode == LOW) && (buttonConfirm == HIGH)) {autoMode = false; manMode = true;}
if ((buttonMode == HIGH) && (buttonConfirm == HIGH)){manMode = false; autoMode = true;}
digitalWrite(LED_MAN, manMode); digitalWrite(LED_AUTO, autoMode);
processVariable = ((double)currentPosition * 2)/2400 ; //2400 pulses/rev = 2mm

/*Manual rotation*/
if (manMode == true) {
      controlVariable = knobSpeed; setPoint = processVariable;
      if (buttonRight == HIGH) {digitalWrite(DIR_LEFT, LOW); digitalWrite(DIR_RIGHT, HIGH);
      analogWrite(ENA, controlVariable);}
    else if (buttonLeft == HIGH) {digitalWrite(DIR_RIGHT, LOW); digitalWrite(DIR_LEFT, HIGH);
      analogWrite(ENA, controlVariable);}
    else {analogWrite(ENA,0);}

 if (buttonSet == HIGH && buttonConfirm == HIGH) {autoHome = HIGH;}
 if (autoHome == HIGH && homeSet == LOW){digitalWrite(LED_AUTOHOME, flashBit);
  digitalWrite(DIR_LEFT, LOW);digitalWrite(DIR_RIGHT, HIGH);analogWrite(ENA, controlVariable);}

 if ( autoHome == HIGH && switchHome == HIGH) {homeSet = HIGH;
  digitalWrite(LED_HOME, homeSet);digitalWrite(LED_AUTOHOME, LOW);digitalWrite(DIR_RIGHT, LOW);
  digitalWrite(DIR_LEFT, LOW); analogWrite(ENA,0);
  delay(500);
  autoHome = LOW;
```

```
  currentPosition = 0;}}


/*Automatic mode (PID)
Automatic mode is executed only when the system is at home and the mode is at AUTO
PID is processed at interval in ms (1000*CYCLE)*/
if ((homeSet == true) && (autoMode == true)) {
  if (buttonSet == HIGH){setPoint = (double)knobSetpoint;}
  if (millis()- lastPIDTime > 1000 * CYCLE) {PID(); lastPIDTime = millis();}
int intControlVariable = int(controlVariable); //Convert to integer.
/*If the error is > 0 the motor moves CW, other moves CCW.*/
if (currentError > 0) {digitalWrite(DIR_LEFT,HIGH); digitalWrite(DIR_RIGHT,LOW);}
if (currentError < 0) { digitalWrite(DIR_LEFT,LOW); digitalWrite(DIR_RIGHT,HIGH);}
if (currentError == 0){intControlVariable = 0; digitalWrite(DIR_RIGHT,LOW); digitalWrite(DIR_LEFT,LOW);}
analogWrite(ENA, abs(intControlVariable));}//Write pulse train to PWM output pin

/*This part is for plotting: setPoint(BLUE), processVariable(RED), controlVariable(GREEN)*/

if (millis() - lastSerialTime > 100) {
 Serial.print(setPoint);
 Serial.print(" ");
 Serial.print(processVariable);
 Serial.print(" ");
 Serial.print(controlVariable);
 Serial.print(" ");
 Serial.println(lmnIntegral);
 lastSerialTime = millis();}
}//END OF LOOP
```

```
/*=============================================================================================*/
/*PID ALGORITHM
lmn is output of the PID controller and consists of 03 portions: proportional, integral, & derivative.
lmn (called loop-manipulated variable): lmn = lmnP + lmnI + lmnD.
lmn = Kp*ER + Ki*integral(ER*dt) + Kd*d(ER)/dt) (dt = CYCLE: processing interval).
lmnP = Kp*ER = Kp * (setPoint - processVariable).
lmnI = Ki*integral(ER*dt) <=> d(lmnI) = Ki*ER*dt <=> lmnI-lastLmnI = ER* Ki* CYCLE
<=> lmnI = lastLmnI + ER*CYCLE*Ki.
lmnD = Kd*d(ER)/dt = Kd* d(ER)/dt = Kd * (ER - lastER)/CYCLE.
lmnD = (Kd/CYCLE)*(ER-lastER)
Bumpless manual-to-auto switch: in manual mode, setPoint = processVariable; then back-calculate lmnI.
Preventing integral-windup: when lmn is out of limit we set lmn to limit and back calculate lmnI.*/
 void PID()
 {
  static double lastError;//last value of error.
  double lmn;//loop manipulated variable
  double lmnP;//proportional portion.
  double lmnD;//derivative portion.
  double integralDiff;//Integral difference
  static double lmnI;//integral portion. Static vars for remembering last value.

  currentError = setPoint - processVariable;
  //Proportional portion calculation
  lmnP = Kp * currentError;

  //Integral portion calculation.
  if (((integralDiff > 0) && (lmn > HIGH_LIMIT)) || ((integralDiff < 0) && (lmn < LOW_LIMIT))){
        integralDiff = 0.0;}
  else { integralDiff = currentError * CYCLE * Ki;}
  if (currentError == 0.0) {lmnI = 0;}
  else {lmnI += integralDiff;}
  lmnIntegral = lmnI;
  //Derivative portion calculation.
  lmnD     = (Kd/CYCLE) * (currentError - lastError);
  lastError = currentError;//update error.
  lmn      = lmnP + lmnI + lmnD;//sum of portions

  //Safety limit
  if ((lmn > HIGH_LIMIT) && (lmnI > HIGH_LIMIT)){lmnI = lmnI - (lmn - HIGH_LIMIT);}
  if ((lmn < LOW_LIMIT) && (lmnI < LOW_LIMIT)){lmnI =  lmnI + (LOW_LIMIT - lmn);}
  if (lmn > HIGH_LIMIT) {lmn = HIGH_LIMIT;}
  if (lmn < LOW_LIMIT) {lmn = LOW_LIMIT;}
  if (currentError == 0) {controlVariable = 0;} else {controlVariable = lmn;}
  }//END PID
```

```
/*=============================================================================================*/
/*INTERRUPT SUBROUTINE CHANNEL A
When an interrupt on channel A occurs then: CW rotation: A != B, CCW rotation: A = B.8*/
void doEncoderA() {

  if (digitalRead(ENCODER_PINA) == digitalRead(ENCODER_PINB)){
    currentPosition++;}
  else {currentPosition--;}}
/*=============================================================================================*/
/*INTERRUPT SUBROUTINE CHANNEL B
When an interrupt on channel B occurs then: CW rotation: A = B, CCW rotation: A != B.*/
void doEncoderB() {
  if (digitalRead(ENCODER_PINA) != digitalRead(ENCODER_PINB)){
    currentPosition++;}
  else {currentPosition--;}}
/*=============================================================================================*/
double fmap(double x, double in_min, double in_max, double out_min, double out_max)
{ if (x < 3) {x = 0;}
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;}
/*END OF PROGRAM
=============================================================================================*/
```

===END OF DOCUMENT===
APENDIXES…
REFERENCES…