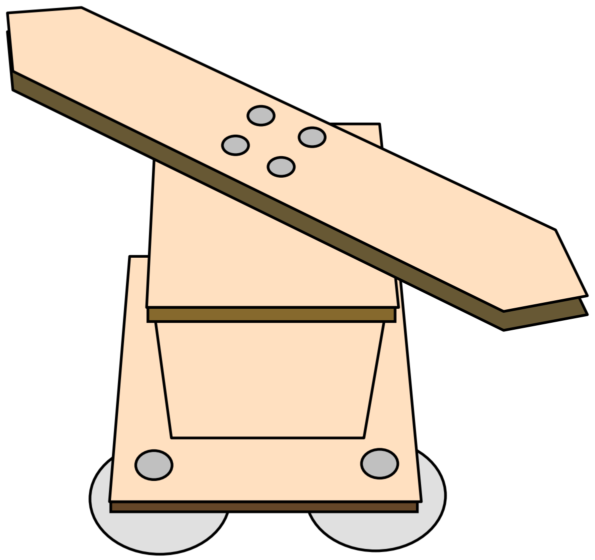
LAB 9 - PID CONTROL DC MOTOR



INTRO / EXERCISE

In prior exercises we learned to move a DC motor and we learned to read encoders so we can sense the motor's angular position. Now we will use the sensor information to control how the motor moves! We will use a PID controller - probably the most commonly used control law. The PID controller is fairly simple but we must TUNE the controller to optimize performance.

ITEMS NEEDED

1. Computer with Arduino IDE, Arduino, USB cable
2. DC motor - E-S motor (Mfr #: 22PG 2230 4.75 EN 12V, Robot Shop)
   12V, 22 mm planetary gear box, 1900 rpm no load speed; 4.75:1 gear ratio; 4 mm shaft
   16 pulse/revolution encoder, 0.2 kg-cm torque, 1.3 kg-cm stall torque
   2.5 A stall current, 90 mA no load current, 330 mA "rated" current
   Note - this a "speedy" motor, not "torquey" (less encoder resolution, but more back-driveable)
3. Motor driver/controller – Cytron MDD10, dual channel (can control 2 motors), 10A, 7-30V ($21)
4. Power supply (12V, bench, power brick, or battery)
5. Motor holding fixture

BACKGROUND

PID control is the most commonly used control law. It is fairly simple and it works under most circumstances. However the GAINS must be TUNED which can be a challenge, and that is whatwe will do in this lab.

REFER to the CT GUIDE on PID CONTROL.

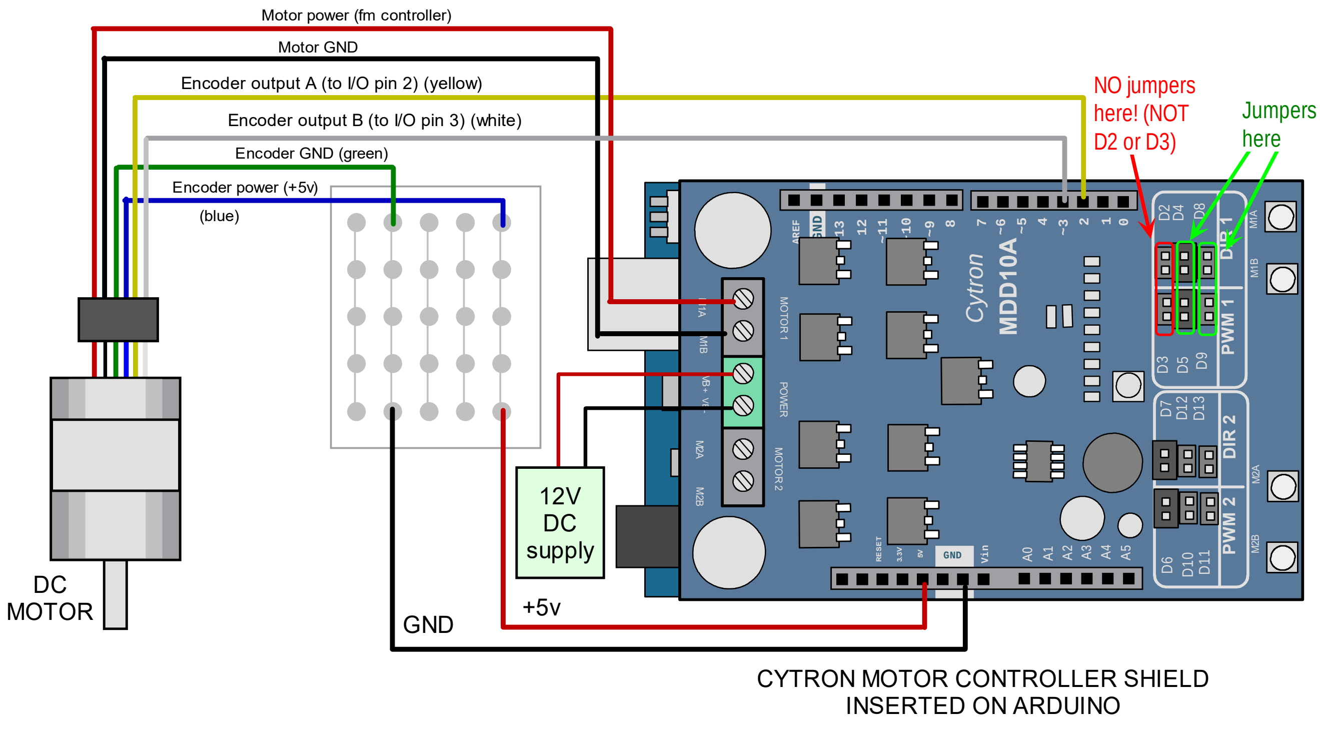
We will basically be duplicating the exercises done in the video below.
https://www.youtube.com/watch?v=fusr9eTceEo

Note that this MOTOR has low gear reduction, making in FAST but not "torquey". This improves motor back-driveability (the ability to turn the output shaft easily with the motor off), but it reduces encoder resolution. We previsouly used Pololu motors had higher gear ratio and more gear backlash (slop).

The newer E-S motors use a PLANETARY GEAR system which has less back-lash or "slop". Slop means you can take the motor shaft and jiggle it around (motor off) and the internal motor's rotor does not move. This is caused by small gaps between the meshing gear teeth. Higher precision gears on more expensive motors can reduce backlash. With backlash the output shaft (ie - the lever) can make small movements but the encoders do not register the movement. This prevents us from making corrections with the controller. This makes it tougher to tune the gains.

CONNECTIONS

Connections will be the same as prior motor-encoder exercises.
You must hook up both the motor and its encoders.
Use the fixture provided.


Motor power (fm controller)
Motor GND
Encoder output A (to I/O pin 2) (yellow)
Encoder output B (to I/O pin 3) (white)
Encoder GND (green)
Encoder power (+5v)
(blue)
NO jumpers here! (NOT D2 or D3)
Jumpers ok here
DC MOTOR
12V DC supply
GND
+5v
Cytron
MDD10A
CYTRON MOTOR CONTROLLER SHIELD
INSERTED ON ARDUINO

CODE

Use the encoder code that uses INTERRUPTS. Then add motor control code. The PID controller lives in a new function called "computePID()". The function "getDesiredAngle()" gets the desired angle. We pass getDesiredAngle a 0 if we want a constant desired angle. We pass it a 1 if we want a square wave.

The square wave is achieved using the statement below

```
   x = (tnow/dur)%2;       // 0=even, 1=odd (better than checking elapsed time)
```

How does it work? Tnow is the current time in milliseconds (eg 1000 at 1 sec). "dur" is the duration in ms (eg - 2000 = 2 seconds). For time from 0 to 2 seconds "(tnow/dur)" is between 0 and 1 (eg - 0.90), but since it's integer division it is truncated to be 0. The "%2" gets the remainder after dividing by 2. So for time 0 to 2 seconds, you get 0. For time 2 - 4 seconds, "(tnow/dur)" is between 1 and 2 but truncates to 1. And the remainder is 1. So "x" above switches from 0 to 1 and back to 0 (and so on) every 2 seconds. It does so without us having to program elapsed time.

INTEGRAL ANTI-WIND UP

I control helps correct for error that won't go away (persistent error). But we must be especially careful with I control which can create instability and un-predictable responses. Accumulated error can easily "WIND UP" causing instability. This happens is our actuator SATURATES (we are asking it to exert more force than it is capable of). Thus we must include ANTI-WIND UP code. This includes

1. Capping accumulated error.
2. ZERO out integral correction is ERROR SWITCHES SIGN
3. STOP integrating (adding) error if actuator is saturated.

```
// PID_DCmotor.ino -----------------------------------
// use DC gearhead motor with encoder (ES motor)
// implement PID position control, tune gains
// shaftAngle is (+) CW viewing onto shaft
// (+) motor speed is CW

int pinA = 2;           // enc pin A, only pins 2 & 3 for external interrupts
int pinB = 3;           // enc pin B

int motorCmdPin = 5;    // pwm
int dirPin = 4;         // for motor dir

int encoderCPR = 64;       // with quadrature (16*4)
float gearRatio = 4.75;    // for e-s motor 1900 rpm
volatile long count = 0;
float shaftAngle = 0.0;
float xd;


void setup() //---------------------------------------------------------
{
  //set up external interrupts for 2 encoders pins
  pinMode(pinA, INPUT_PULLUP);
  pinMode(pinB, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(pinA), encAchanged, CHANGE);
  attachInterrupt(digitalPinToInterrupt(pinB), encBchanged, CHANGE);

  pinMode(motorCmdPin, OUTPUT);
  Serial.begin(9600);
}

void loop() //-----------------------------------------------------------
{
  float u, motorCmd;
  xd = getDesiredAngle(0);            // input: 0 = const, 1 = square wave

  shaftAngle = 360 * count / (encoderCPR * gearRatio);  // convert to degrees

  u = computePID(shaftAngle);
  motorCmd += u;

  driveMotor(motorCmd);

  //outputToSM(motorCmd, 200);        //output to serial monitor
}


float computePID (float x) //-----------------------------------------------------
{
  float Kp = 0.1;          //P gain 10 ideal?
  float Kd = 0.2;          //I gain 350 ideal?
  float Ki = 0.01;         //D gain 0.1 ideal?

  float error, de, u;
  static float eo = 0.0;         // error fm last loop ("static" - value retained
  static float esum = 0.0;       // static - var's also not re-initialized

  unsigned long t;
  static unsigned long to = 0;
  float esumCap = 1500.0;         // limit for summed error (anti-wind up)
  xd = 0.0;
  t = millis();

  error = x - xd;                              // get error
  de = (float) (error - eo) / (t - to);      // get deriv of error
  esum += error;                               // add up error

  //I anti-windup code

  esum = constrain(esum, -esumCap, esumCap);  // clamp integral of error
//  eee = error * eo;  // needed?
  if ((error * u) > 0)     // if e & u have same sign (not good)
  {
    esum = 0.0;                  // zero out added (accumulated) error
    aaa = 1;                     // nec?
  }

  u = Kp * (-error) + Kd * (-de) + Ki * (-esum);   // compute control

  eo = error;                //reset vals for next loop
  to = t;

  // Serial.println(esum);      // write to SM if nec

  return u;
}

float getDesiredAngle(int desiredMode) // --------------------------
{
  unsigned long tnow;
  int dur = 3000;                // time @ ea position

  float xd;
  float xd1 = 30.0;        // 2 angles for square wave
  float xd2 = -30.0;
  unsigned long x;
  tnow = millis();         // get current time
  x = (tnow/dur)%2;        // 0=even, 1=odd (better than checking elapsed time)

  if (!desiredMode)              // desiredMode=0 --> xd constant
  {
     xd = 0.0;
  }
   else if (desiredMode)         // desiredMode=1 --> xd = square wave
   {

     if (x) { xd = xd1;}
     else  { xd = xd2; }
     }
   }

  //Serial.println(xd);
  return xd;
}

void driveMotor(int motorCmd) // ---------------------------------------
{
  int motorDir;
  int speedVal;
  if (motorCmd >= 0)
  {
    motorDir = 0;
  }
  else if (motorCmd << 0)
  {
    motorDir = 1;
  }
  motorCmd = abs(motorCmd);
  speedVal = map(motorCmd, 0, 100, 0, 140);         // cap at 230ish, 255 max
  digitalWrite(dirPin, motorDir);
  analogWrite(motorCmdPin, speedVal);               // analog write is 0-255
}

void encAchanged() // ----------------------------------------
{
  if (digitalRead(pinA) != digitalRead(pinB))
  {
    count++;
  }
  else
  {
    count--;
  }
}

void encBchanged() // ---------------------------------------------
{
  if (digitalRead(pinA) == digitalRead(pinB))
  {
    count++;
  }
  else
  {
    count--;
```

```
void outputToSM (float x, int period) //-----------------
{
  static unsigned long tlast;
  if (millis() - tlast >= period)  //print to SM ev 0.3 sec
  {
    Serial.println(x);
    tlast = millis();
  }
}

//end ---------------------------------------------------------------
```

EXERCISES

9.1 SERVO TO POSITION with DISPLACEMENT

Program the motor to follow a CONSTANT motor angle (0 degrees). The angle that the motor is at when you start the program will be 0 degrees.

Hold the lever to an initial position and start your program. This initializes the motor position to be "zero degrees" at this location. Your code should set up the PID controller so that the desired motor shaft angle is this position. You will produce a "disturbance" by displacing the lever (slowly!) about 40 to 90 degrees. The PID controller should try to drive the lever back to the original position. The "response" - that is the movement of the lever - will be based on the physics of the system and the Kp, Ki, and KD gains used. The goal is to change (tune) the gains so the response is the way we want: to move back to the desired position quickly with almost no oscillation.

Start with P control.

Start with a lower gain (Kp from ~ 0.1 to 10) - feels like soft spring & response is sluggish (lever may not reach desired angle.

Increase Kp - spring stiffer, faster response, should get closer to desired angle, may have oscillations

Increase Kp more - until system starts becoming unstable. (Kp ~ 18?) Lever may vibrate on its own.

Drop Kp - quick response, not unstable, but have oscillations

Add Kd (D control) - to reduce oscillations. Start Kd low. Then increase to remove oscillations.
    Error may not go to 0.

Add Ki (I control) - CAREFULLY. Start with low Ki, and SLOWLY increase it since I control can easily make the system unstable.

Note in the video, the update frequency affects the controller performance. At the end he has his updating at 800 Hz. We may have to check how fast our loop function is running (eg code in ARDUINO GUIDE on how to check loop speed).

CAREFUL - this experiment can cause instability. Keep your hand near the adjustable power knob. Be ready to switch it off quickly. Make sure the lever cannot hit anything (especially your hand and fingers), even if it swings a full 360 degrees.

9.2 SQUARE WAVE

Program the motor to follow a square wave. That is, it will try to move repeatedly between 2 angular positions. The PID controller will attempt to follow that square wave. The steps for tuning the PID gains will be similar here.