Introduction to Biomedical Engineering

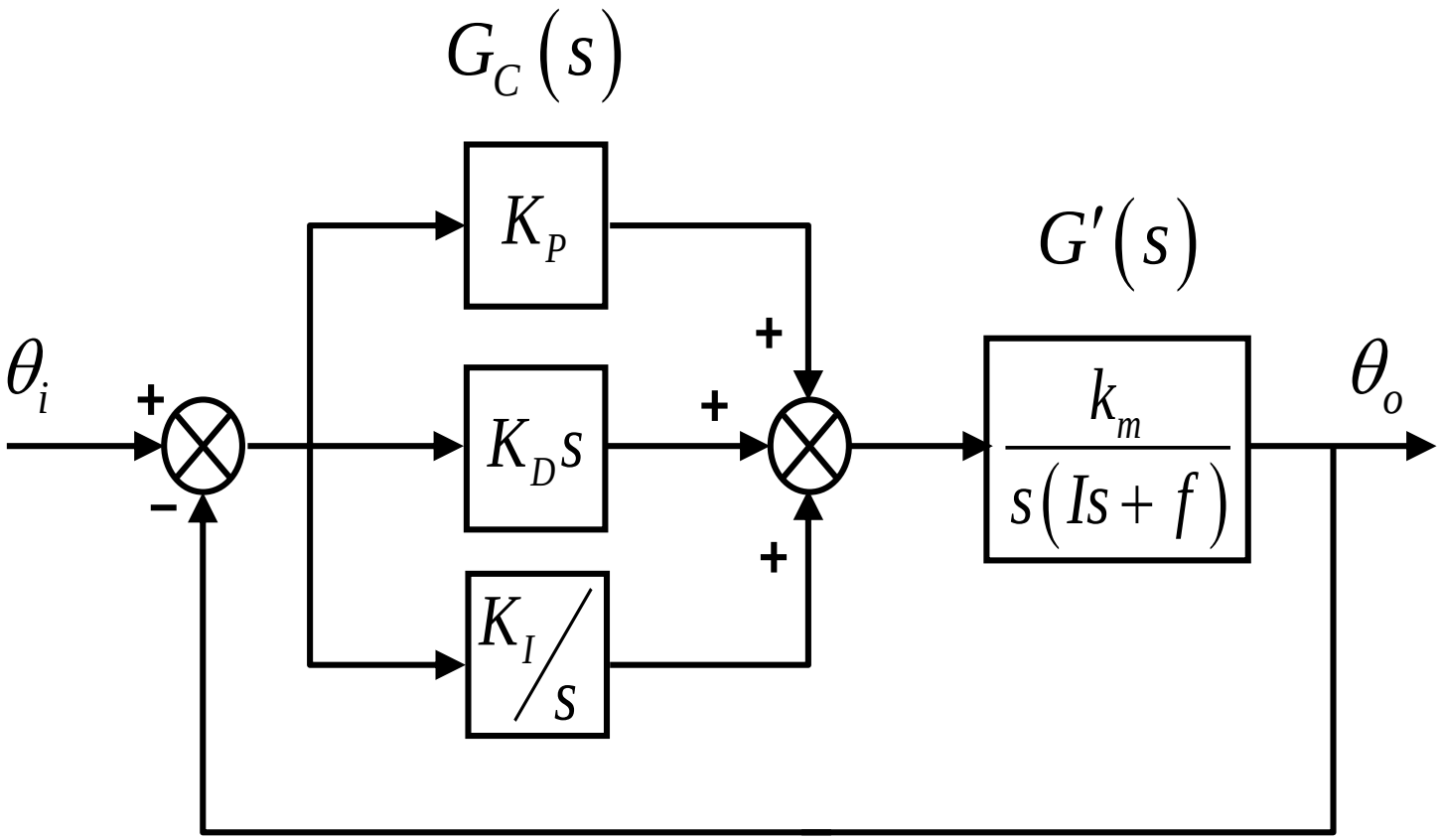
Section 3: **Microcontrollers/Arduino**

Lecture 3.3: PID controller

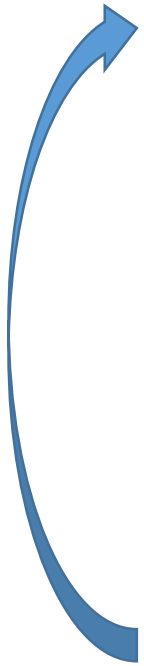
POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH

PID controller

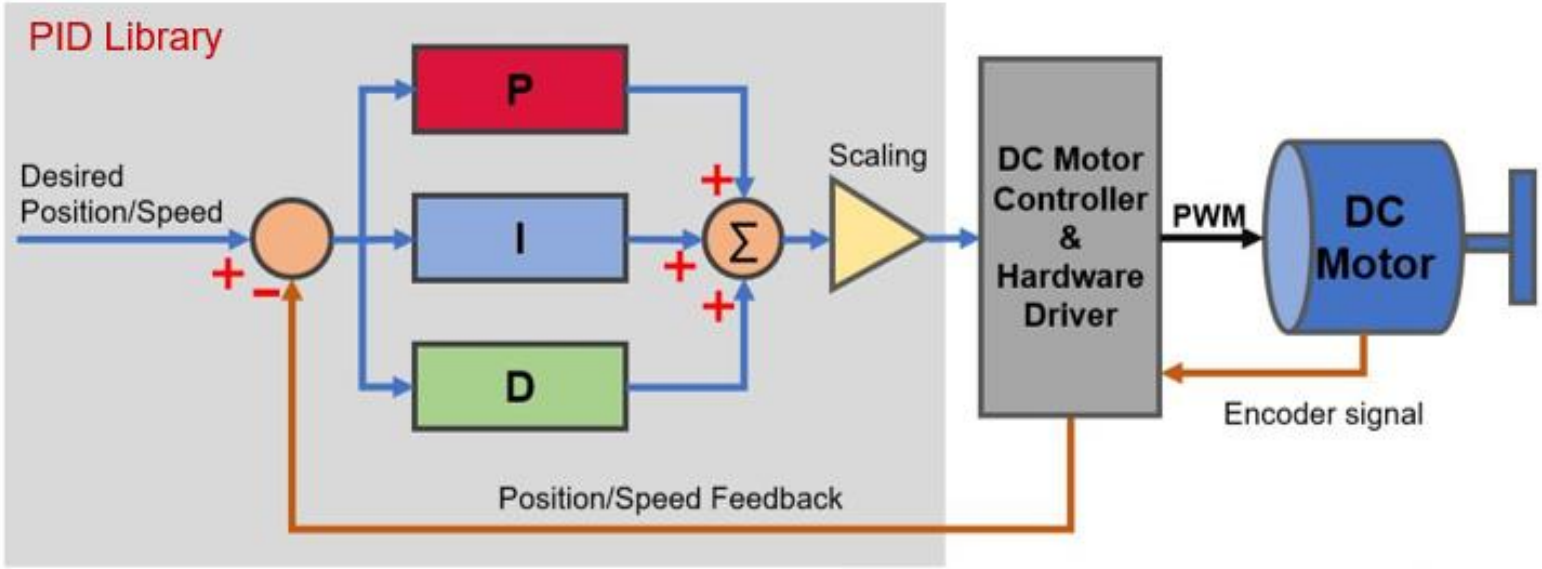


Basic algorithm



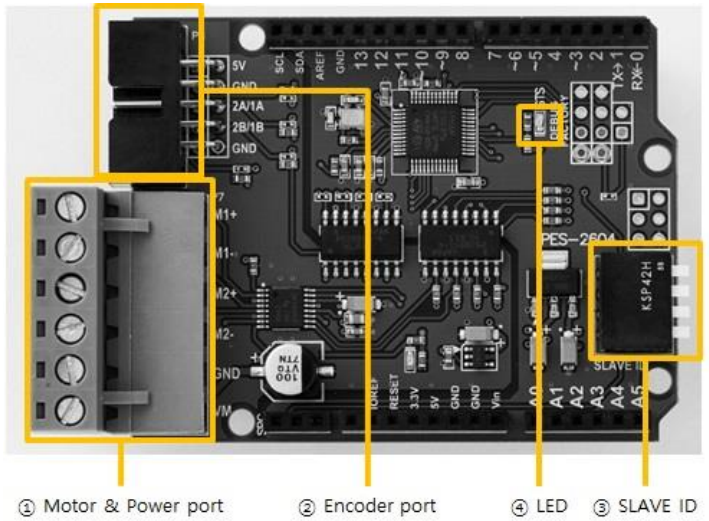
- Compute error
  - E = Set_value – actual_value
- Compute differential error
  - dE = (E-E_previous)/time_between_measurements
- Compute integral error
  - sE = sE+E*time_between_measurements
- Correct output
  - OUTPUT = OUTPUT + Kp*E + Kd*dE + Ki*sE

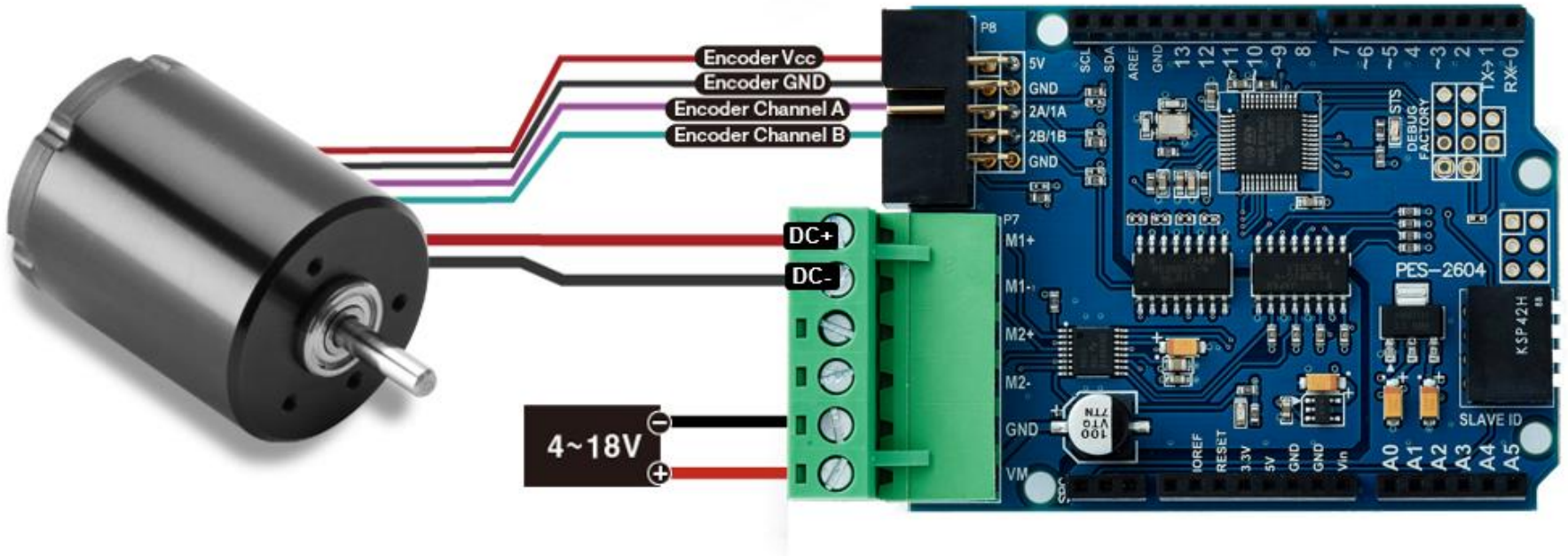
One way of doing it



Possible quick and reliable solution

https://www.phpoc.com/support/manual/pes-2604_user_manual/contents.php?id=layout

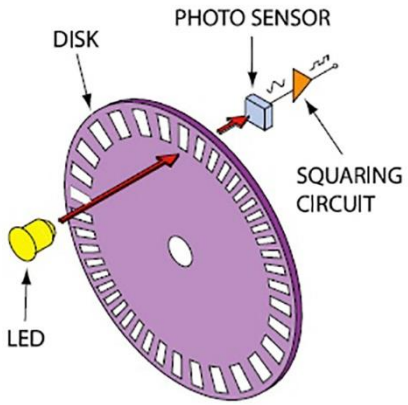


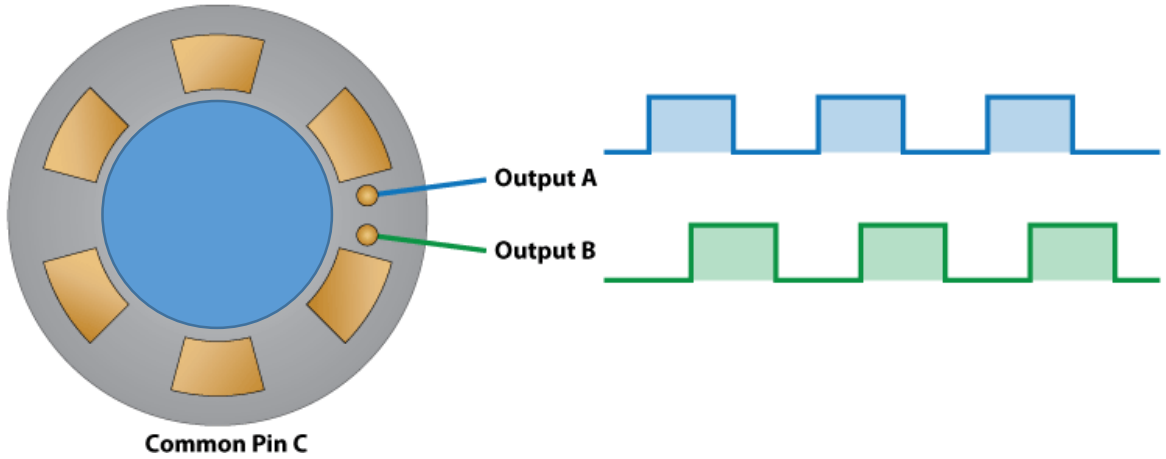


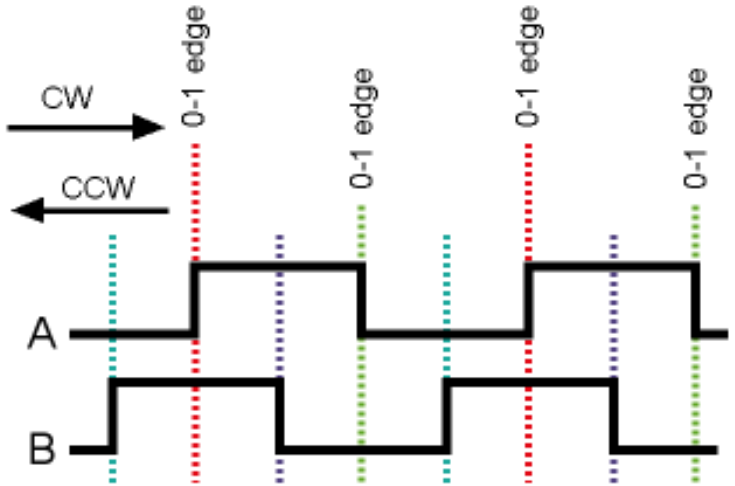
Components we need

- Arduino Uno
- Motor
- Rotary encoder
- Power supply
- TIP120 transistor (for PWM control of speed) and resistor

Rotary encoder

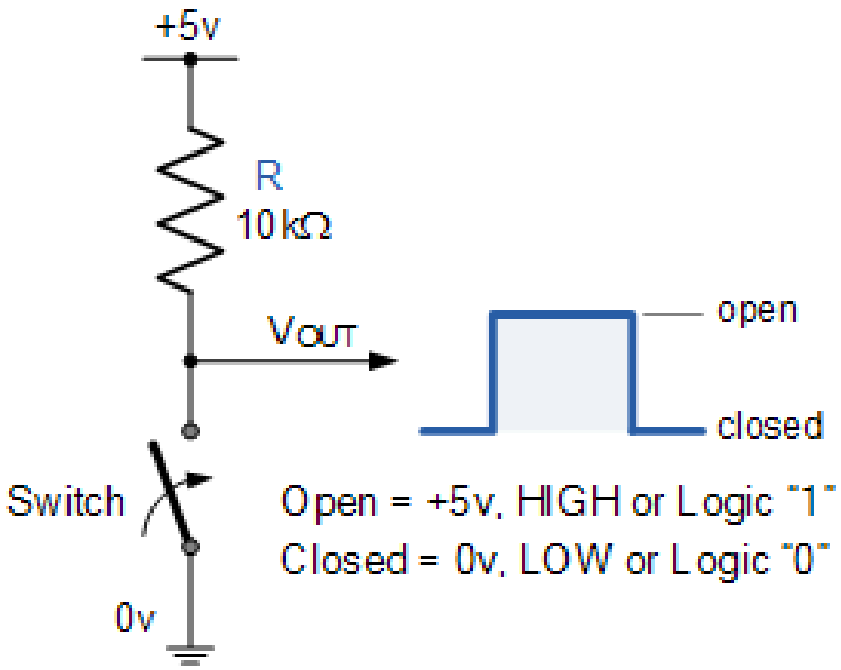




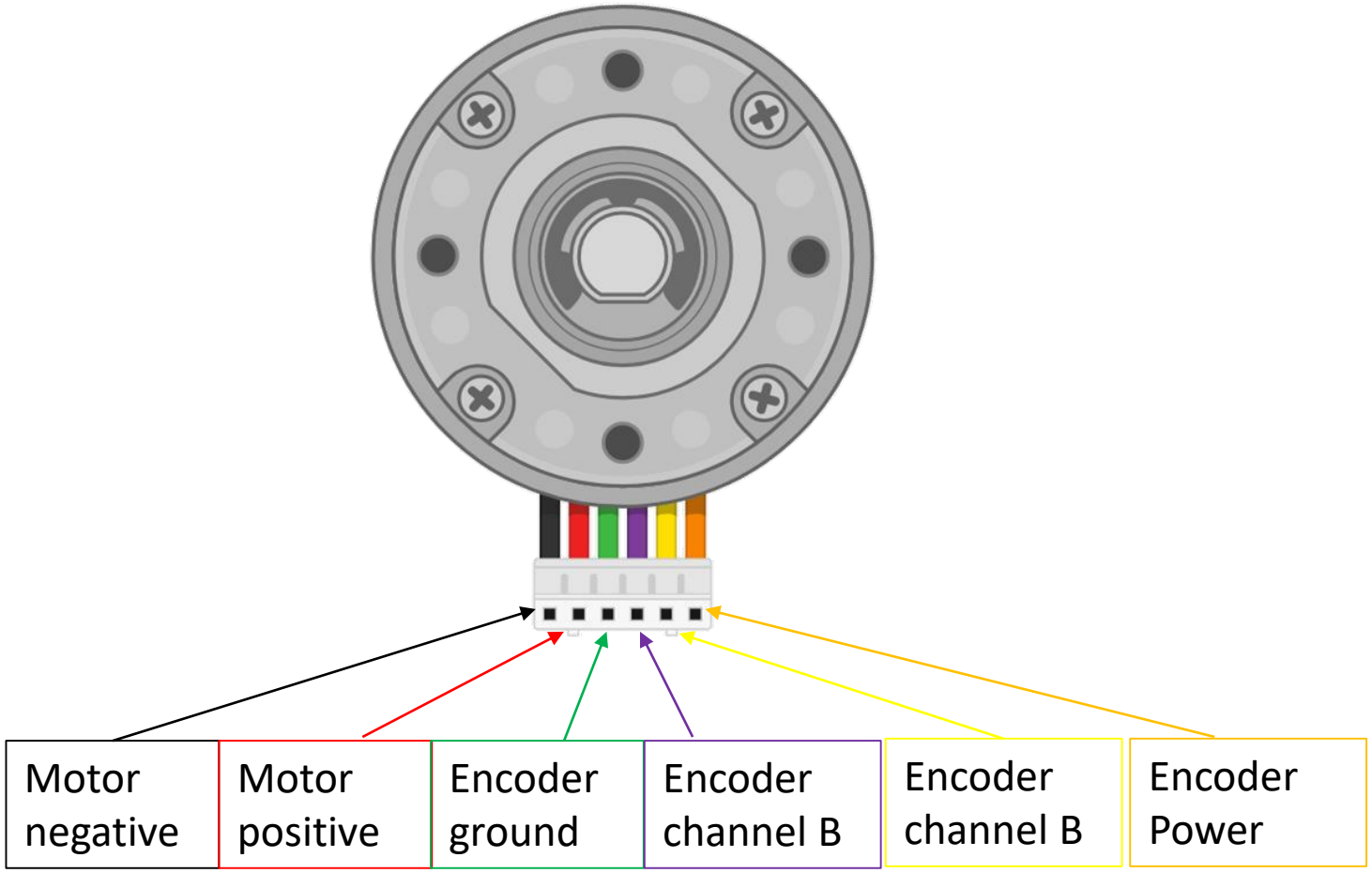


Simplest way to read the speed (not the best, but simplest)

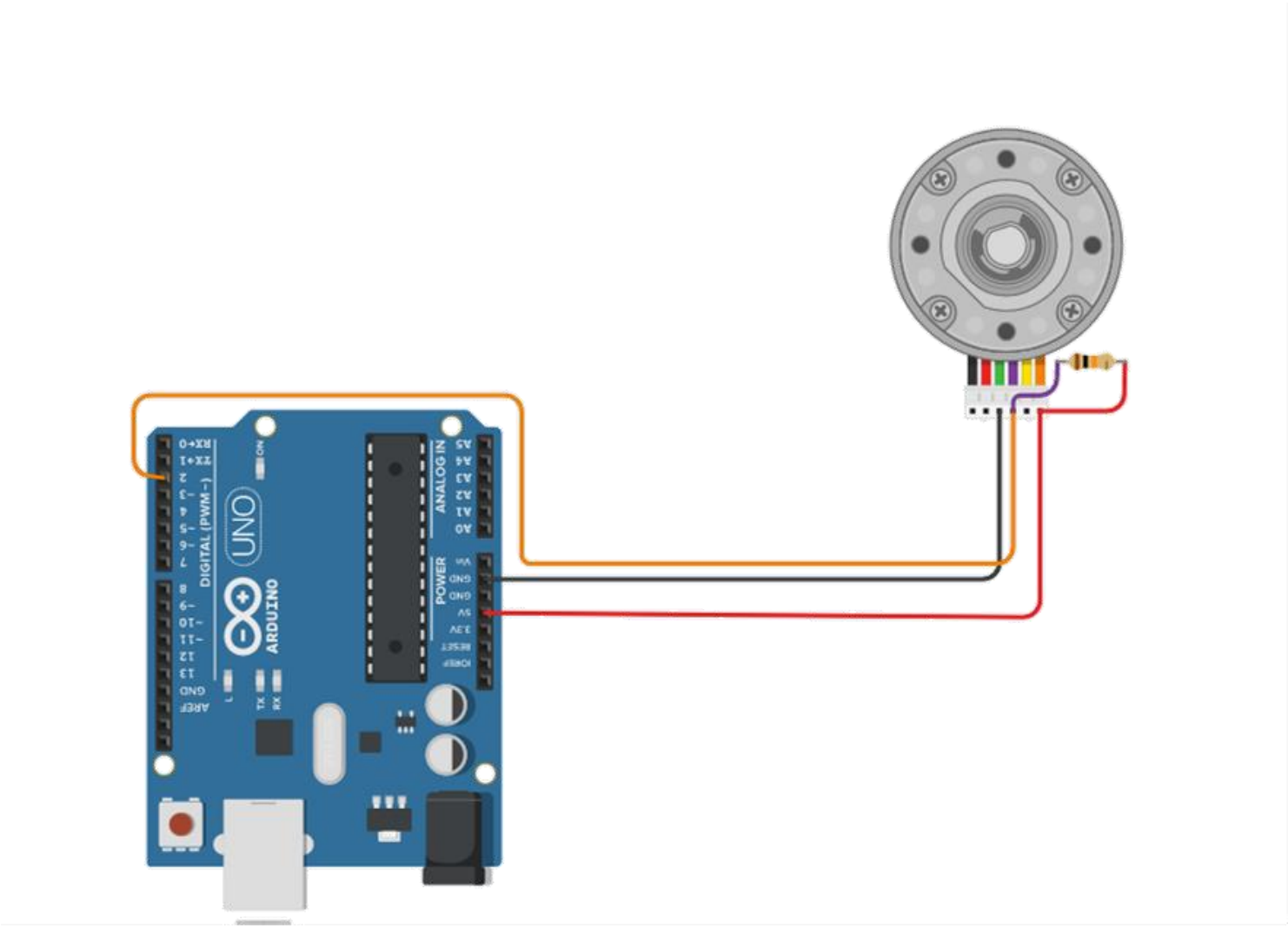
- Just use one counter output A



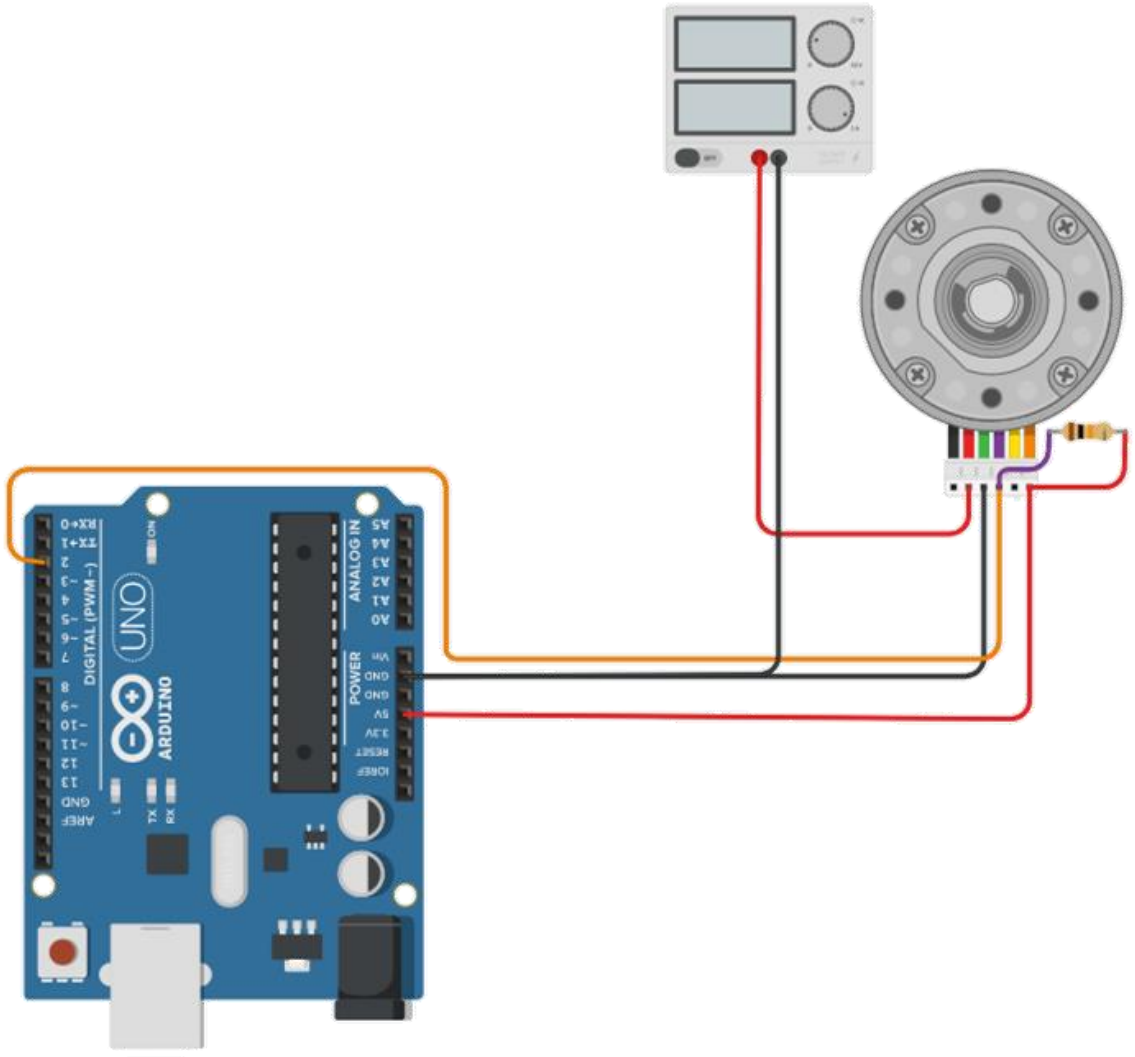
Motor with encoder



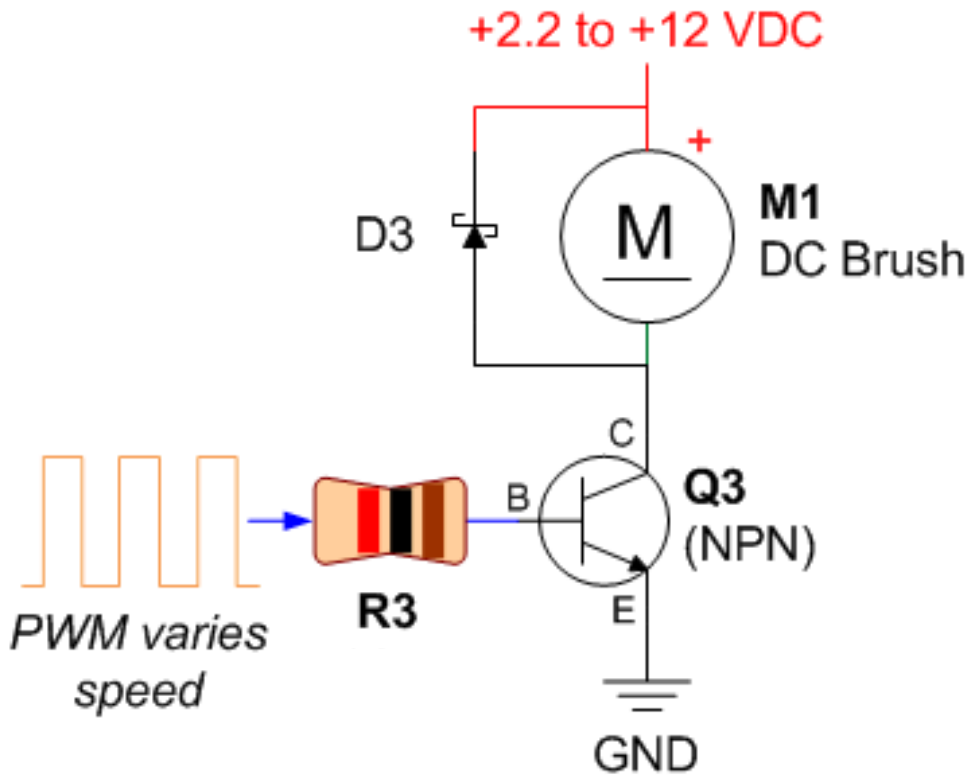
Lets start playing

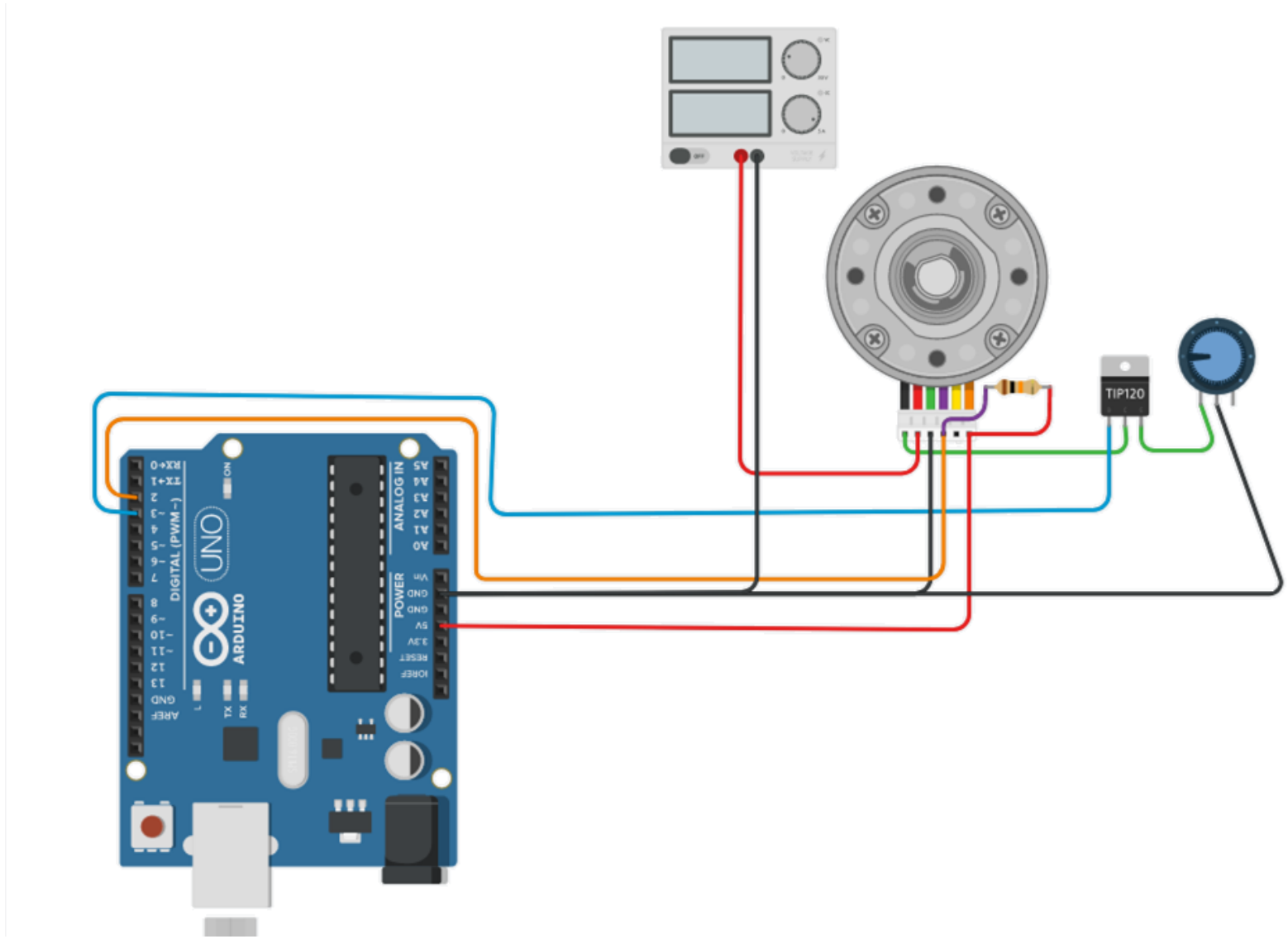


Add some power

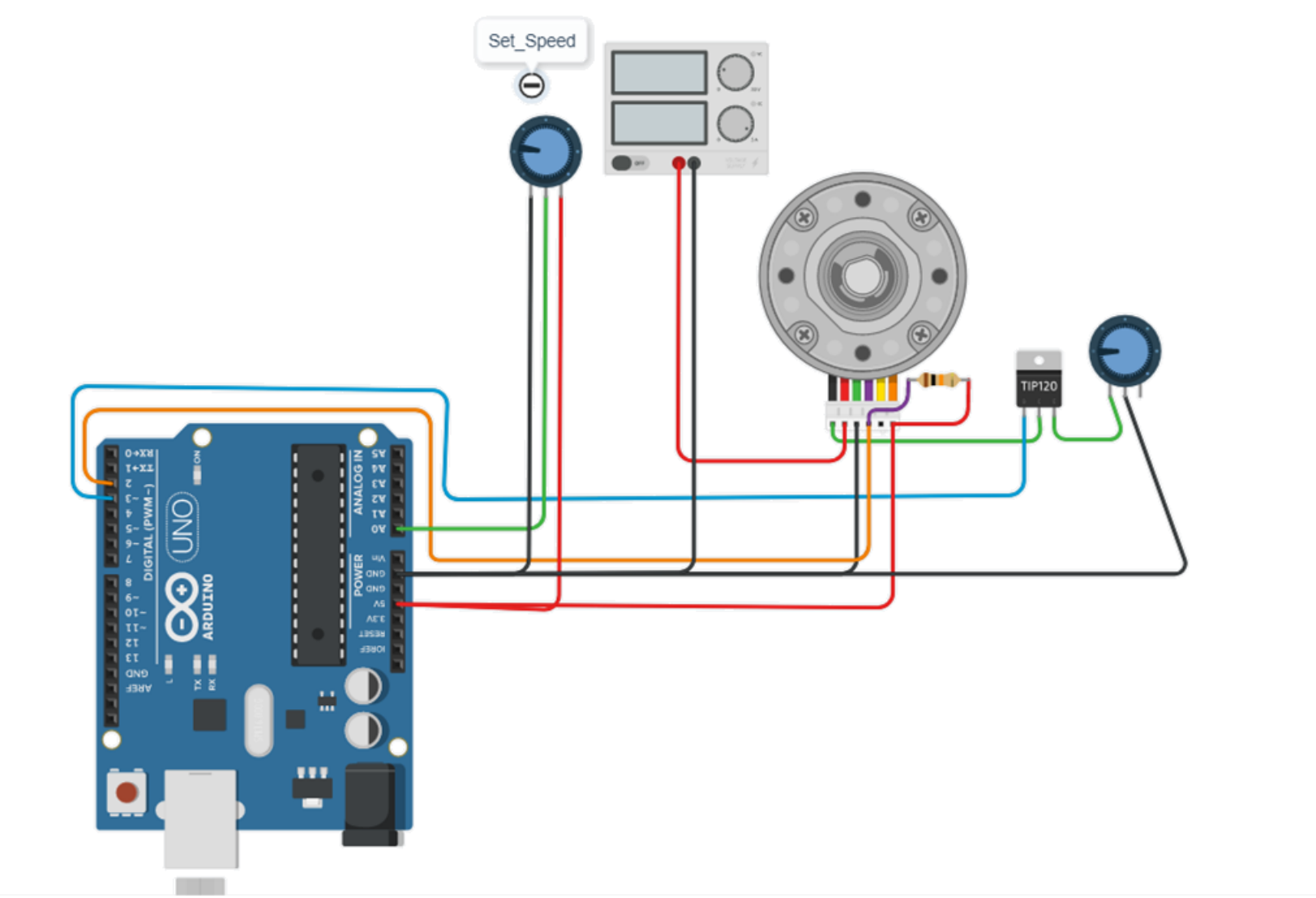


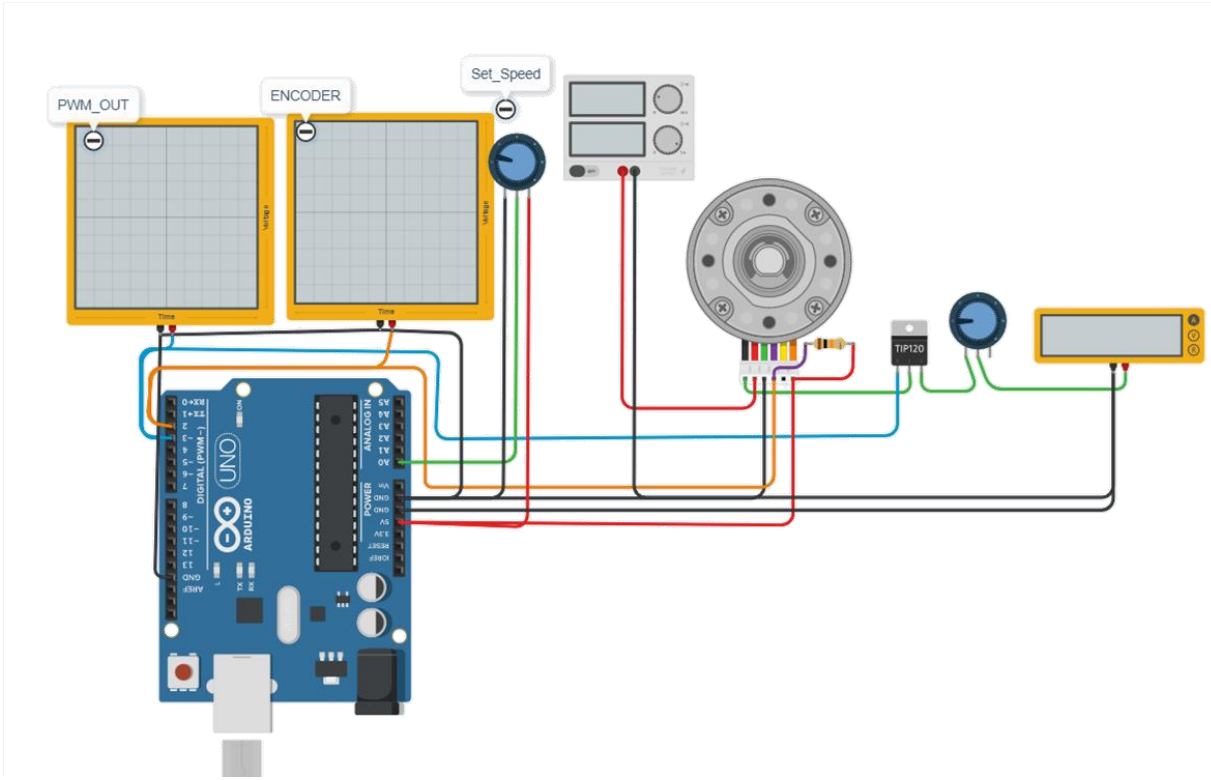
Output to the motor: PWM




TIP120
UNO
ARDUINO
DIGITAL (PWM~)
ANALOG IN
POWER

Final Assembley





https://www.tinkercad.com/things/cHusPUHIboN-dc-motor-pid-velocity-control

```
#define pwm_out       3
#define encoder_in    2
#define potentiometer A0

int PWM_value;
int rpm_set, rpm;
int past_error=0;
int error=0;
int I_error=0;
int D_error=0;
float KP=0.1; //try values: kp=0.1, ki=0.000001, kd=0,001
float KI=0.0001;
float KD=0.000;

void setup()
{
  Serial.begin(9200);

  pinMode(encoder_in,    INPUT);
  pinMode(potentiometer,INPUT);
  pinMode(pwm_out,       OUTPUT);

  PWM_value=10;
  analogWrite(pwm_out,   PWM_value);
}

void loop()
{
  rpm_set=map(analogRead(potentiometer),0,1013,10,600);

  rpm=9.55*((60*1000*10)/pulseIn(encoder_in,HIGH));

  //compute error
  past_error = error;
  error=rpm_set - rpm;
  I_error=I_error+error ;
  D_error=past_error-error;

  //pwm
  PWM_value =
    PWM_value
    +
    (KP*error)
    +
    (KI*I_error)
    +
    (KD*D_error);

  if(PWM_value>254){PWM_value=255;}else{if(PWM_value<10){PWM_value=10;}}

  analogWrite(pwm_out,   PWM_value);

  Serial.println(rpm);

  //Serial.print("error=");Serial.println(error);
  //Serial.print("D_error=");Serial.println(D_error);

}
```

Tuning the PID controller

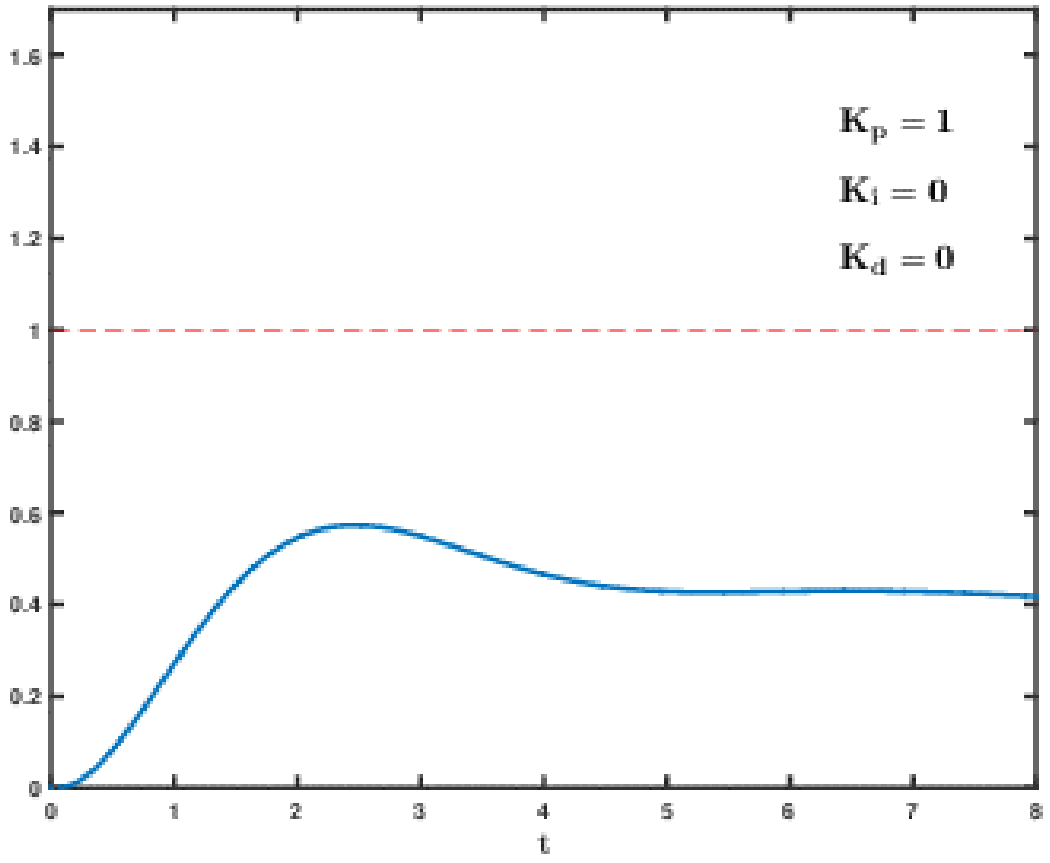
Choosing a tuning method

Method	Advantages	Disadvantages
Manual tuning	No math required; online.	Requires experienced personnel.[citation needed]
Ziegler–Nichols [b]	Proven method; online.	Process upset, some trial-and-error, very aggressive tuning.[citation needed]
Tyreus Luyben	Proven method; online.	Process upset, some trial-and-error, very aggressive tuning.[citation needed]
Software tools	Consistent tuning; online or offline - can employ computer-automated control system design (*CAutoD*) techniques; may include valve and sensor analysis; allows simulation before downloading; can support non-steady-state (NSS) tuning.	Some cost or training involved.[20]
Cohen–Coon	Good process models.	Some math; offline; only good for first-order processes.[citation needed]
Åström-Hägglund	Can be used for auto tuning; amplitude is minimum so this method has lowest process upset	The process itself is inherently oscillatory.[citation needed]

Manual tuning

- Set Ki=0 and Kd=0
- ↑ Kp until oscillates, set to half that value
- ↑ Ki until any offset is corrected in sufficient time (too much will cause instability)
- ↑ Kd until acceptably quick to reach the reference (too much will cause excessive response and overshoot)

Tuning the PID controller



Thank you for your attention!

Some graphic material used in the course was taken from publicly available online resources that do not contain references to the authors and any restrictions on material reproduction.

POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

OPEN
POLYTECH