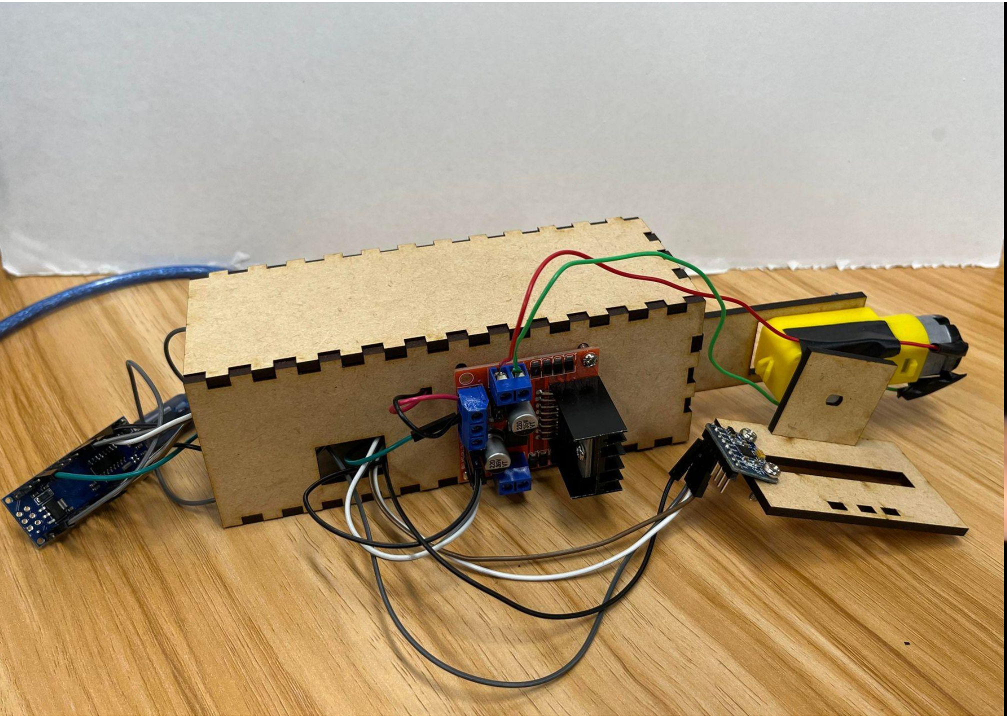
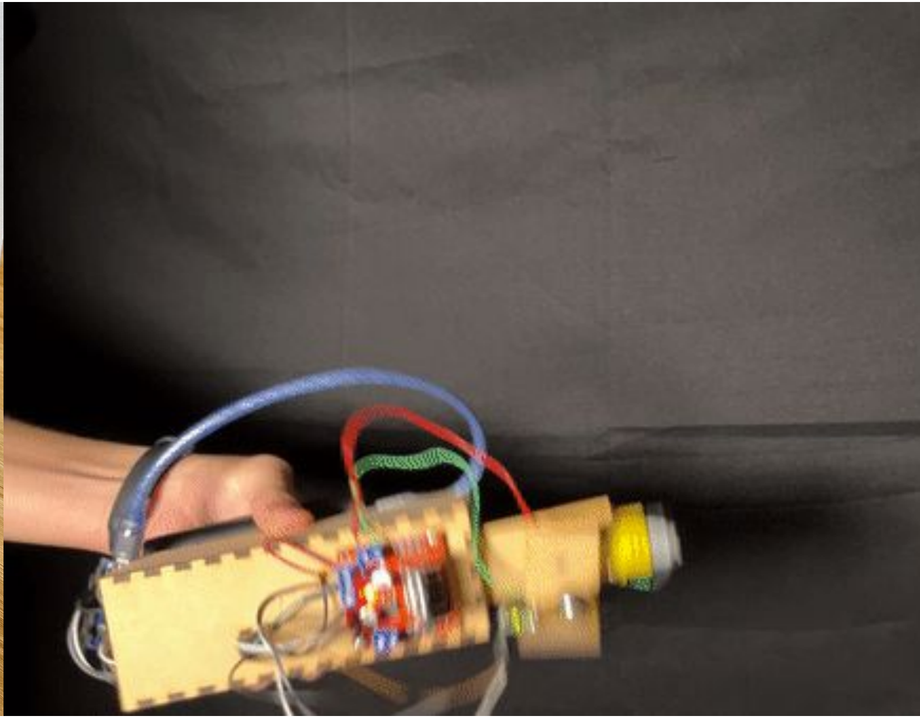
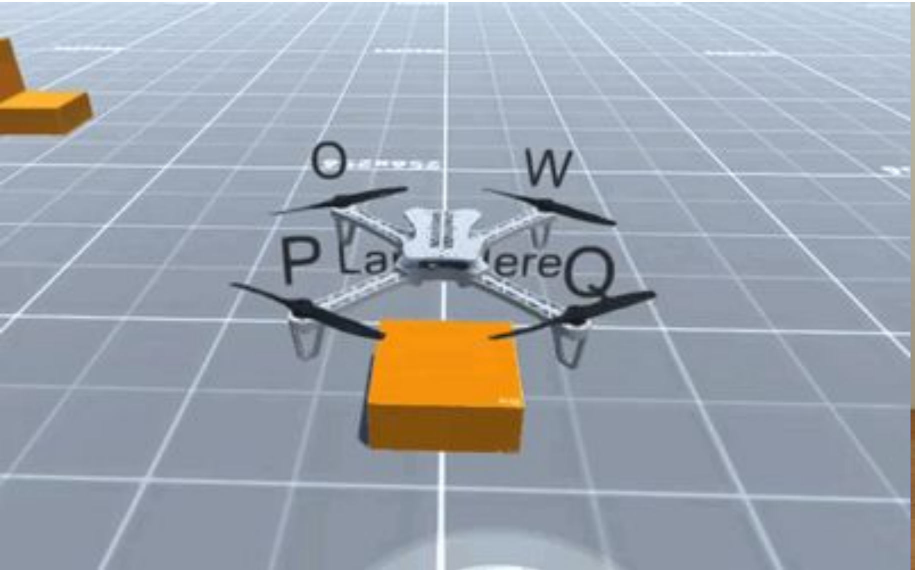
PID Control - Stabilizer

An 1-axis stabilizer using PID control

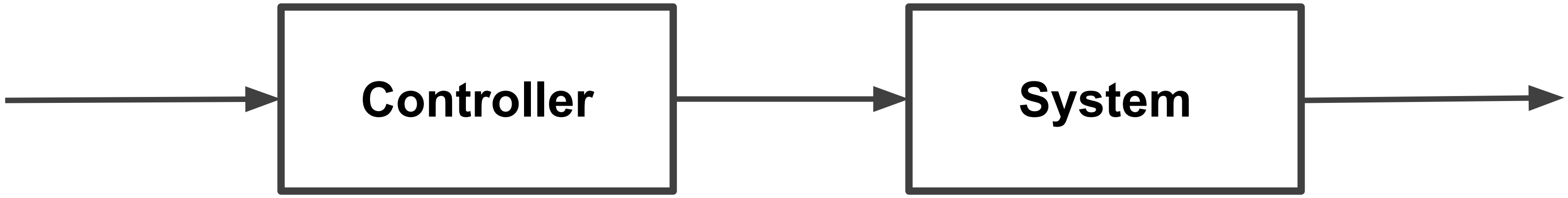




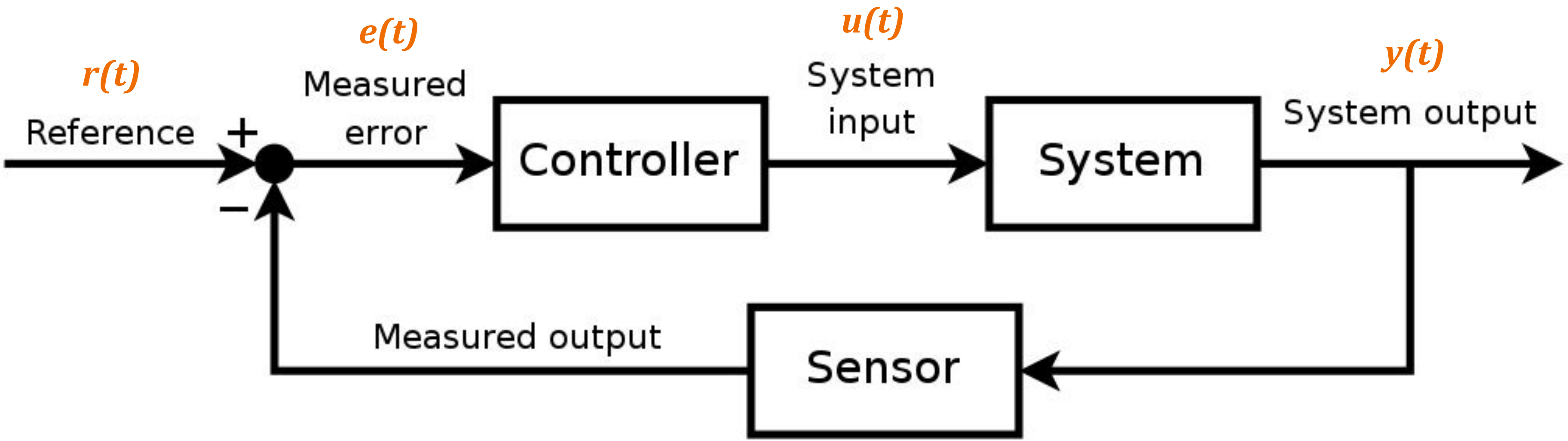
PID Control



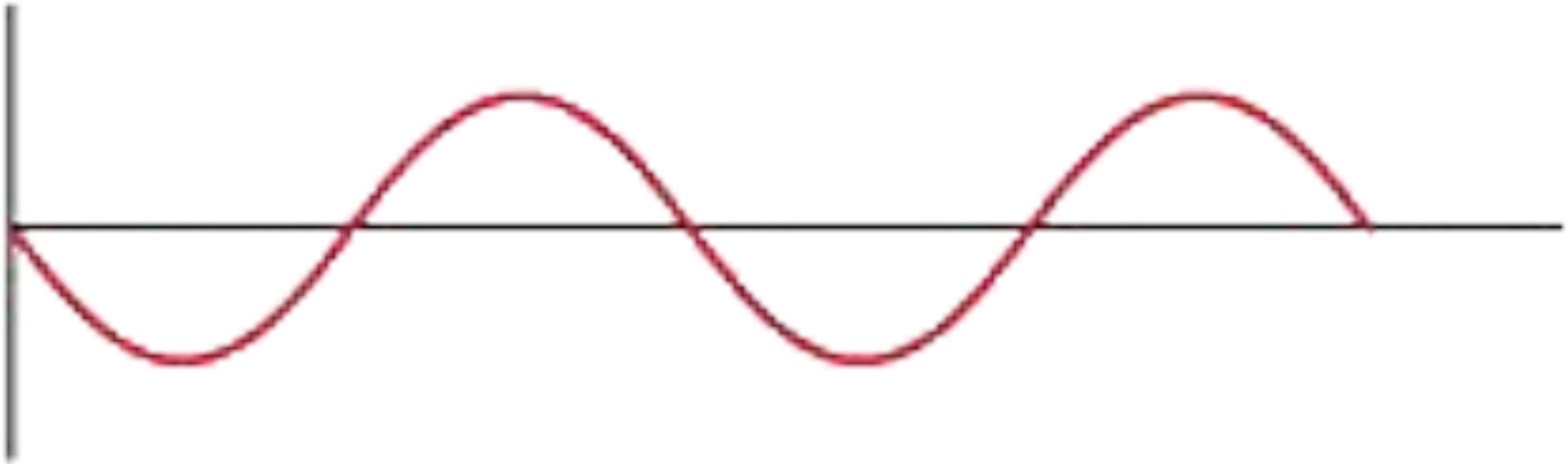
Open-Loop Control



Closed-Loop Control

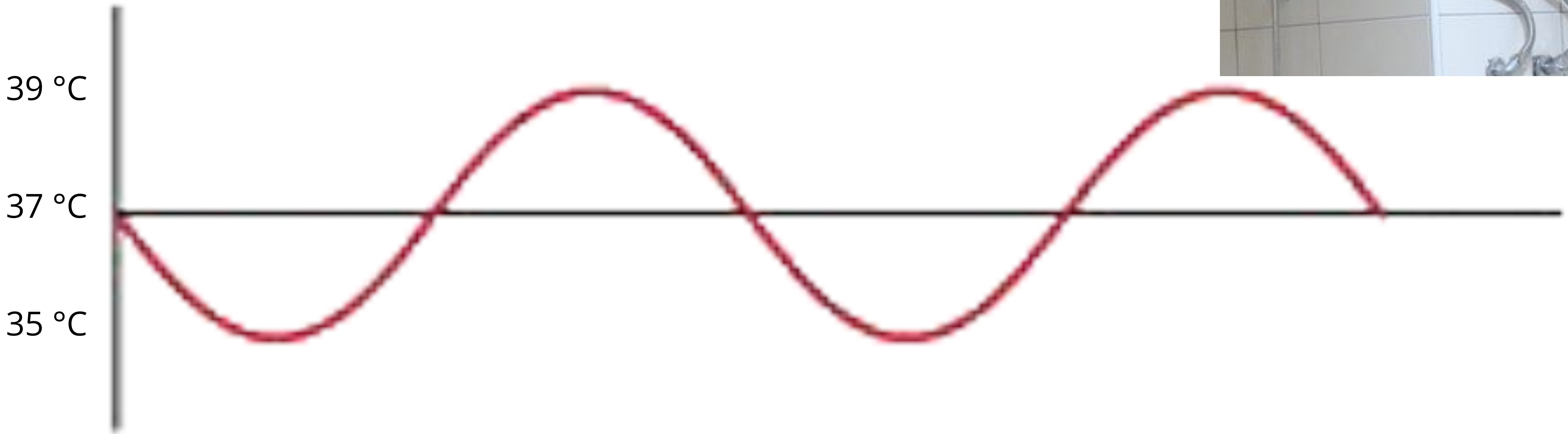


Bang-Bang (on/off) Control

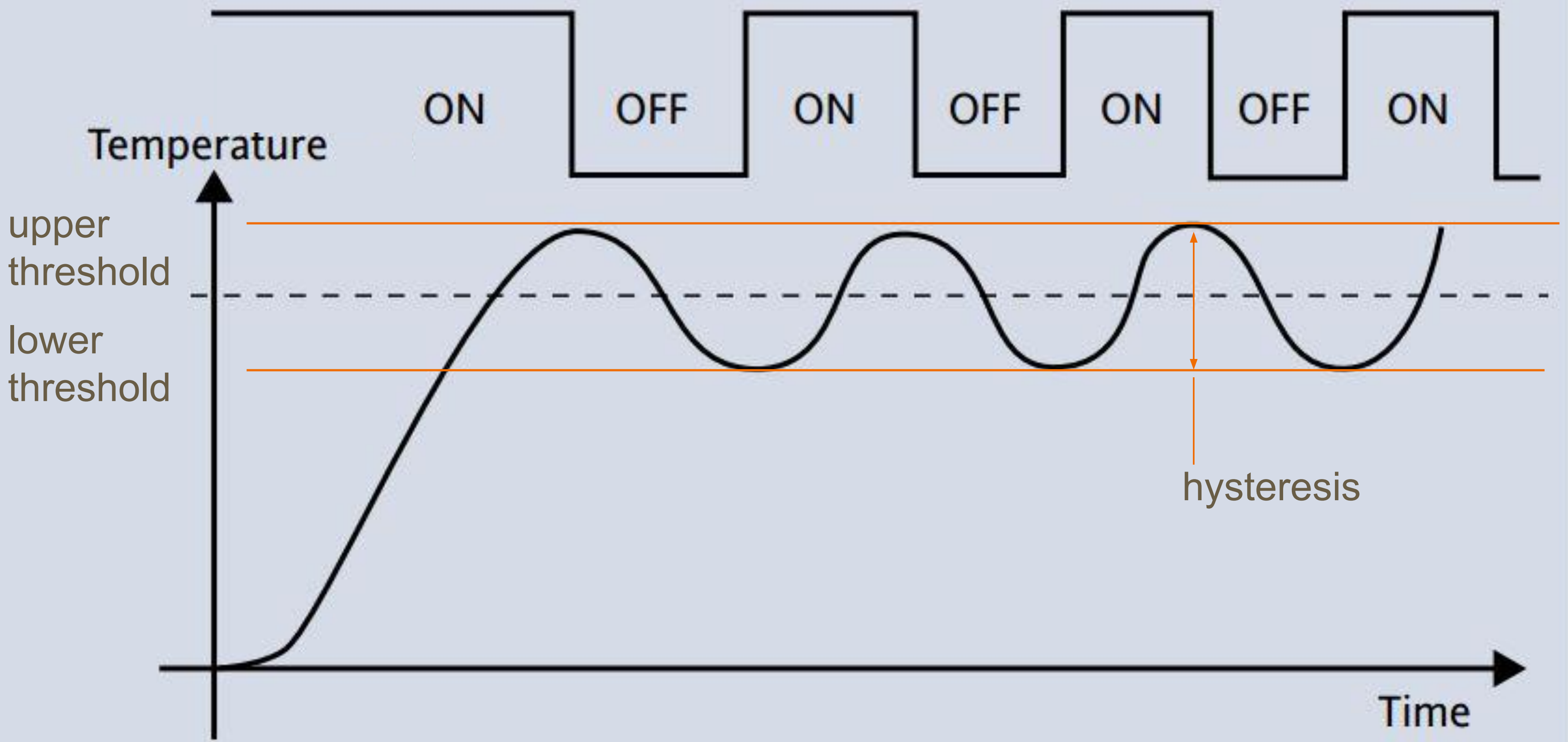


hysteresis + oscillations

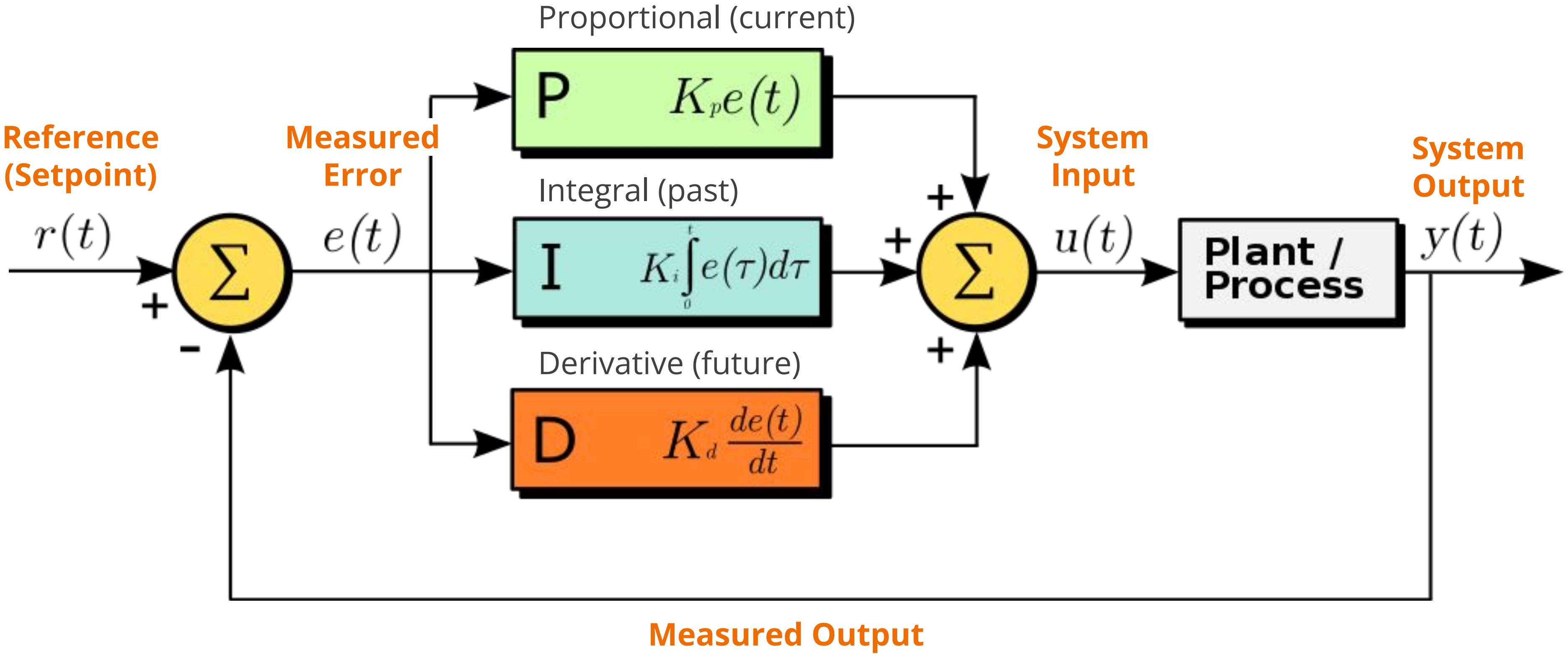
Bang-Bang (on/off) Control



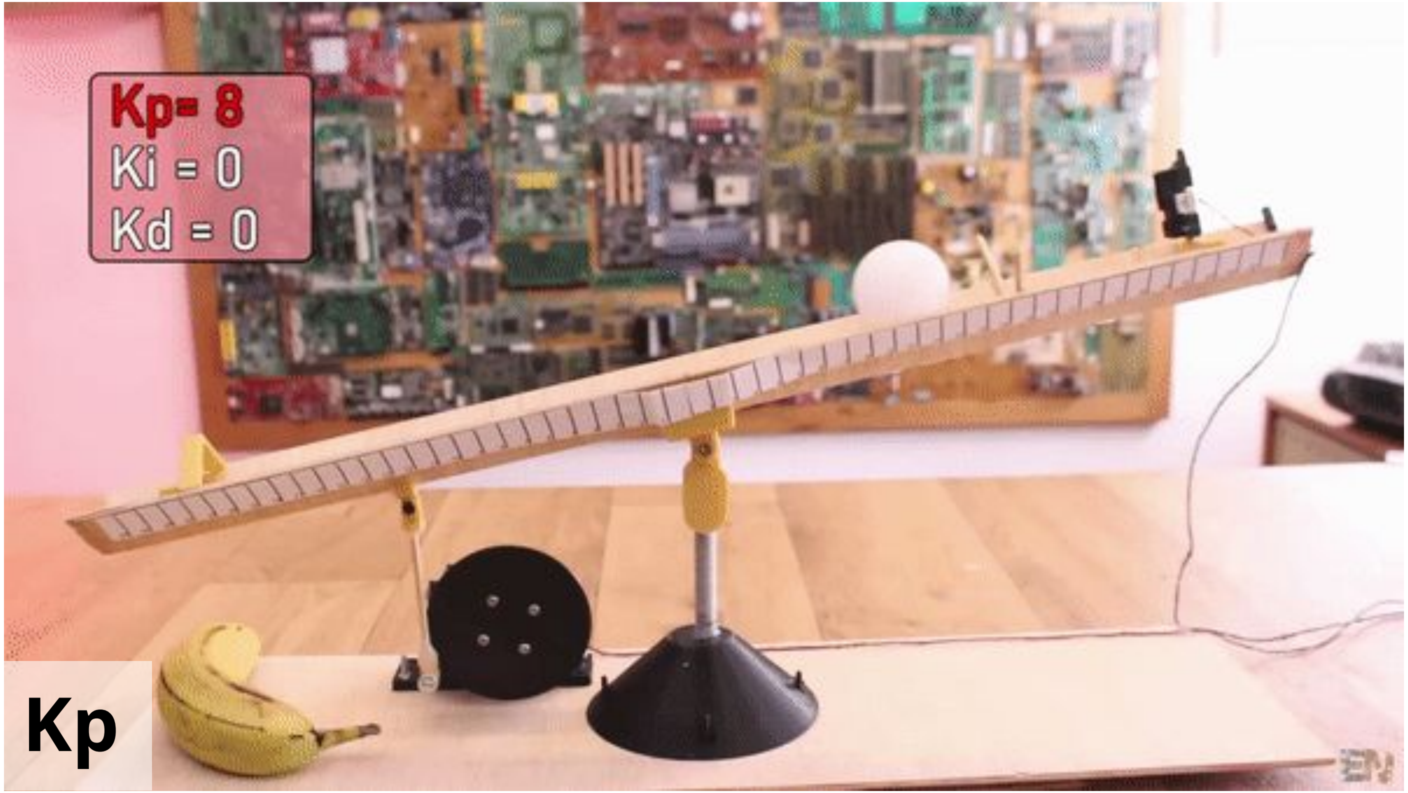
hysteresis + oscillations


ON
OFF
ON
OFF
ON
OFF
ON
Temperature
upper threshold
lower threshold
hysteresis
Time

PID Control



Proportional (current)
P
$K_p e(t)$
Reference
(Setpoint)
Measured
Error
Integral (past)
System
Input
System
Output
$r(t)$
$e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
$u(t)$
Plant /
Process
$y(t)$
+
-
+
+
+
Derivative (future)
D
$K_d \frac{de(t)}{dt}$
Measured Output

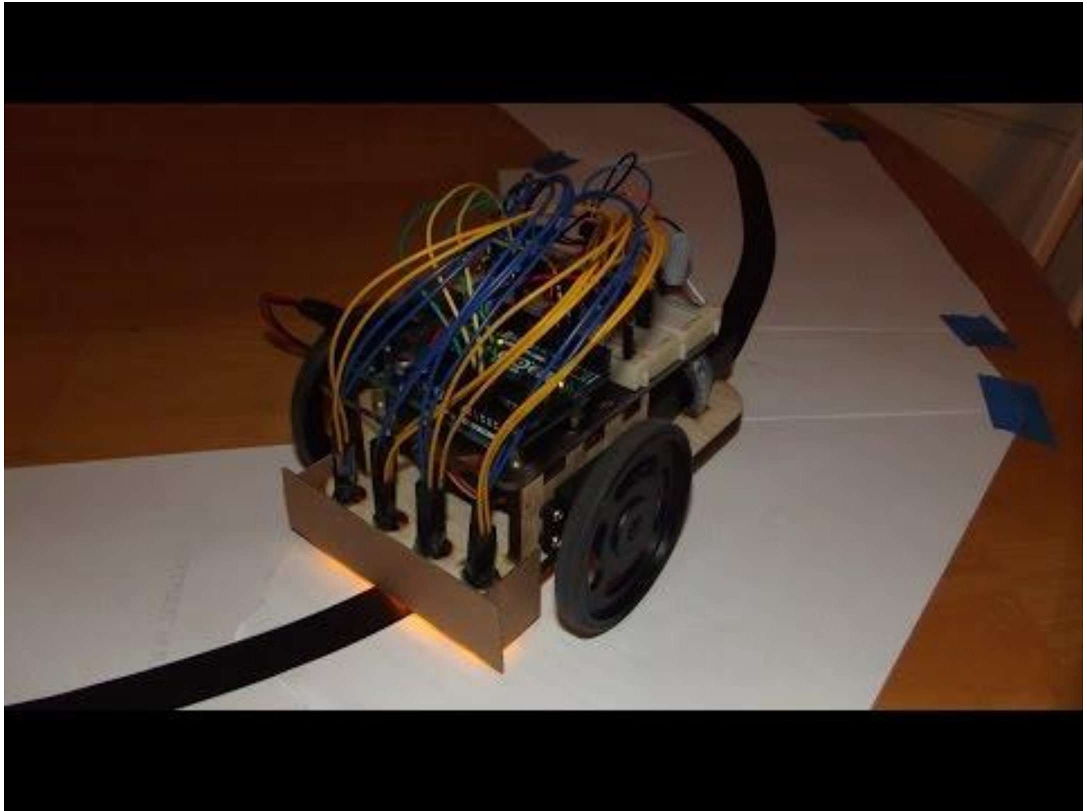


https://youtu.be/JFTJ2SS4xyA


TIME TRAVEL STUFF
Kp + Kd


Kp + Kd + Ki

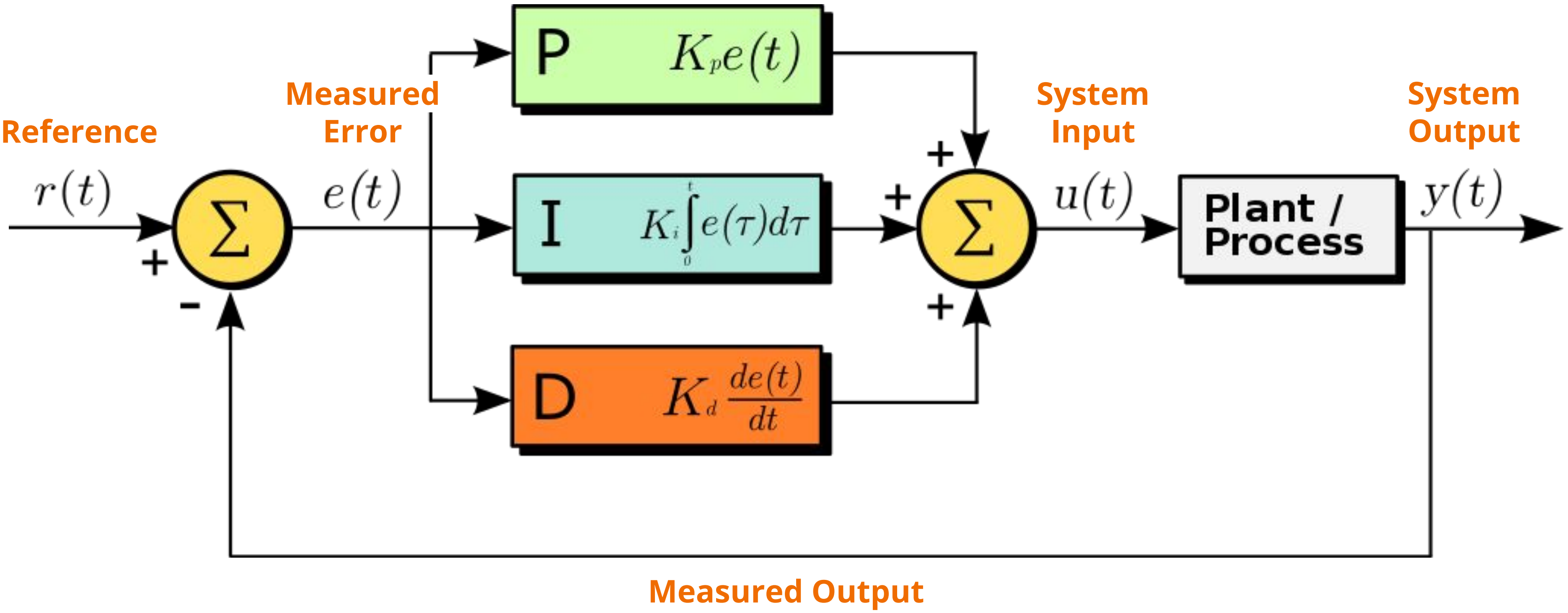
PID vs Bang-Bang



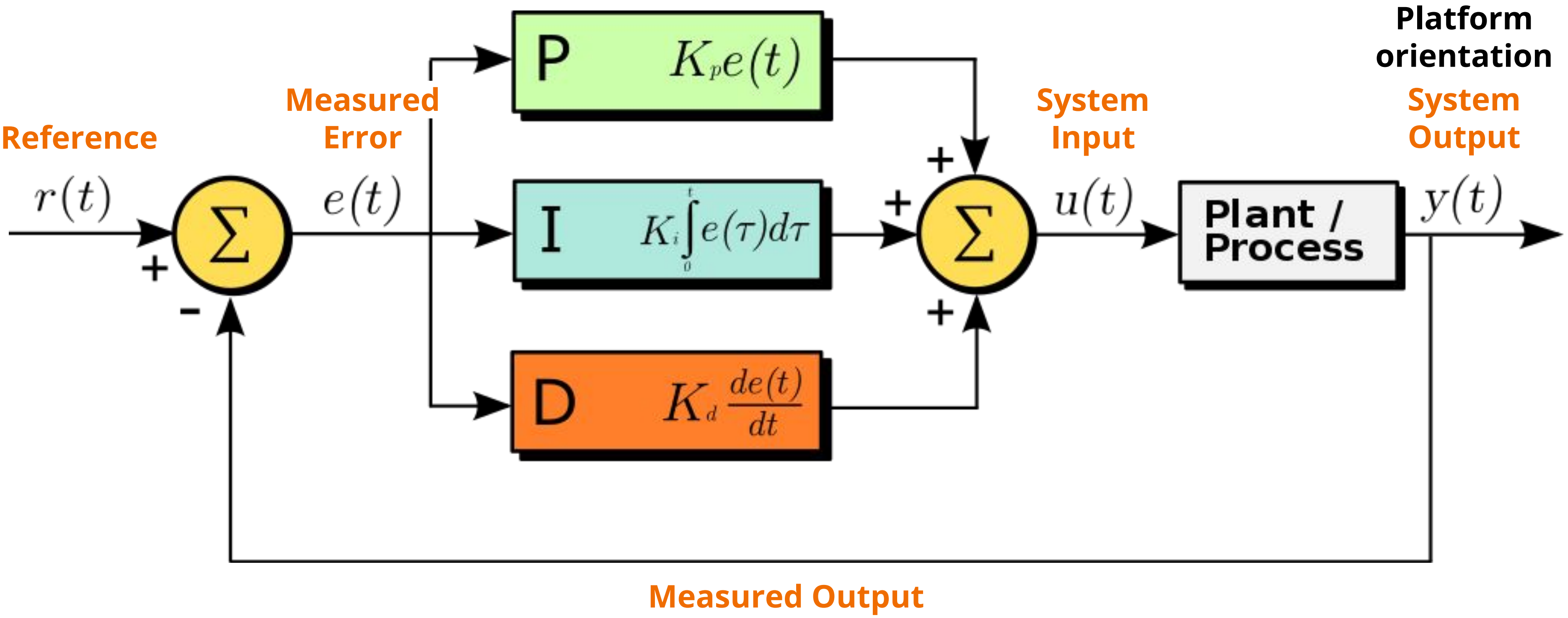
The Goal

Keep the platform leveled
(control the speed & direction of the motor based on PID values)

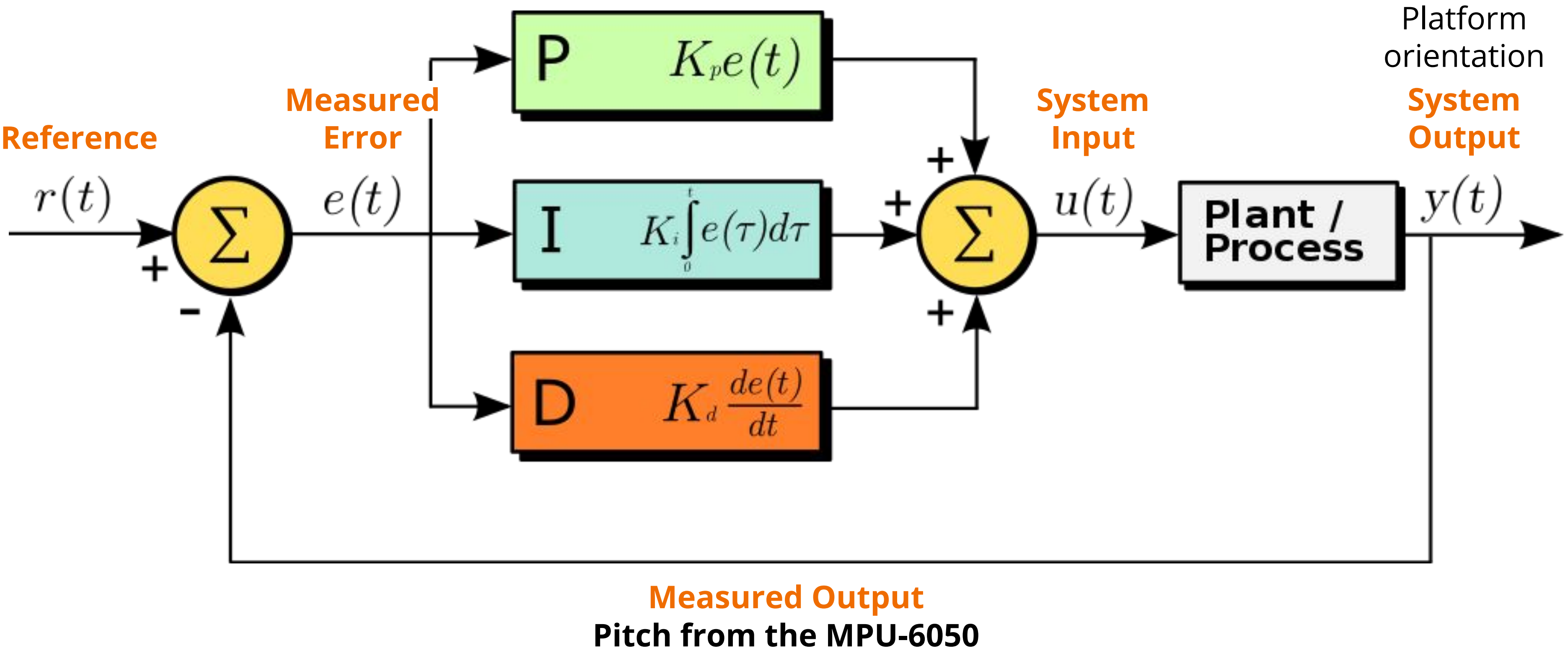
PID Control


Reference
$r(t)$
Measured
Error
$e(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau) d\tau$
D
$K_d \frac{de(t)}{dt}$
System
Input
$u(t)$
Plant /
Process
System
Output
$y(t)$
Measured Output

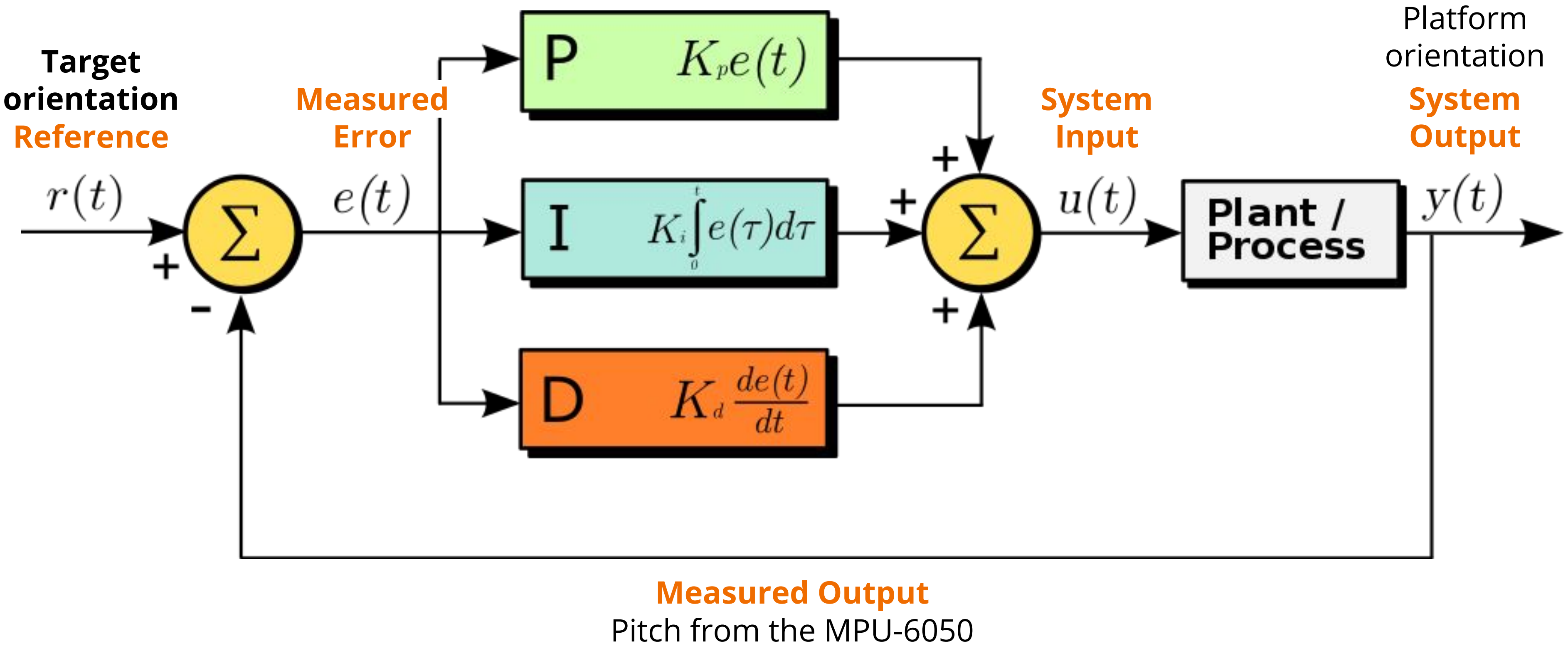
PID Control


Platform orientation
Reference
$r(t)$
+
-
Σ
Measured Error
$e(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
+
+
+
Σ
System Input
$u(t)$
Plant / Process
System Output
$y(t)$
Measured Output

PID Control

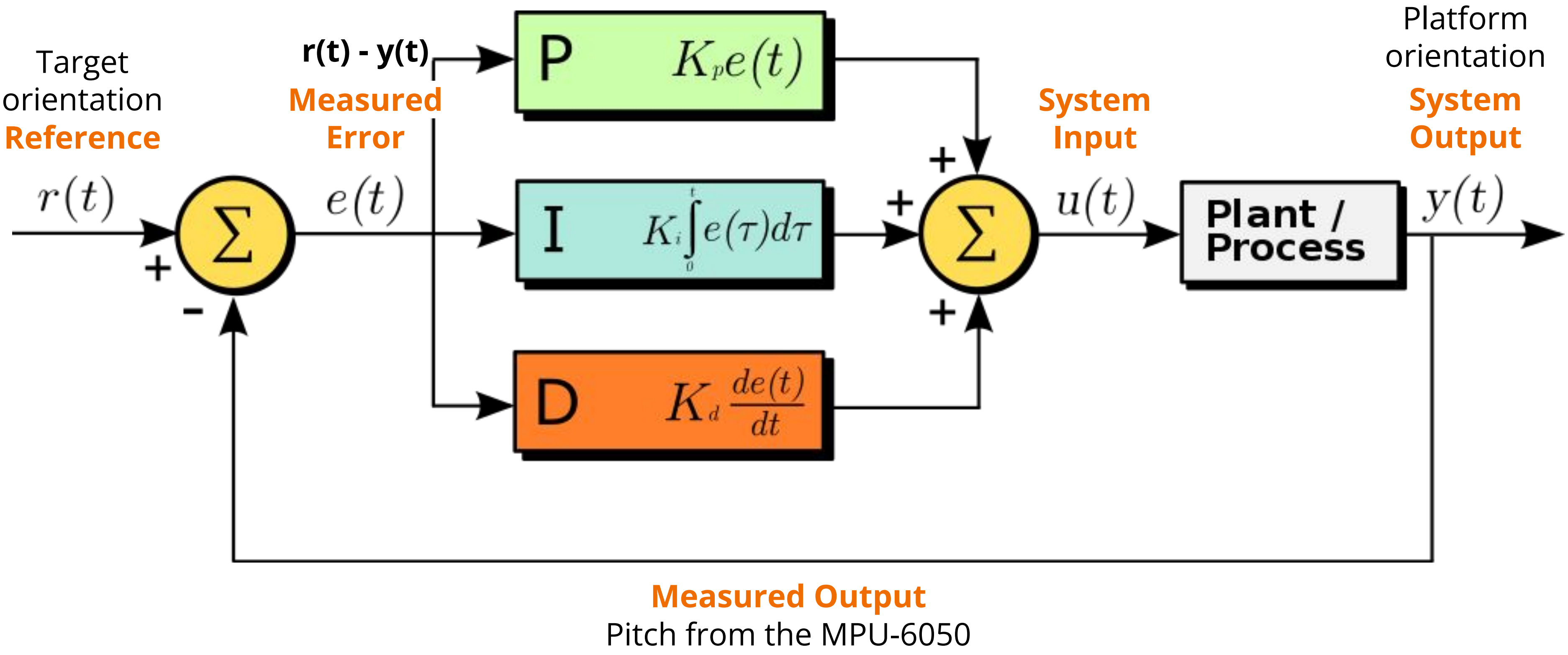

Platform
orientation
System
Output
Measured
Error
Reference
System
Input
$r(t)$
$e(t)$
$u(t)$
$y(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
Plant /
Process
+
-
Measured Output
Pitch from the MPU-6050

PID Control



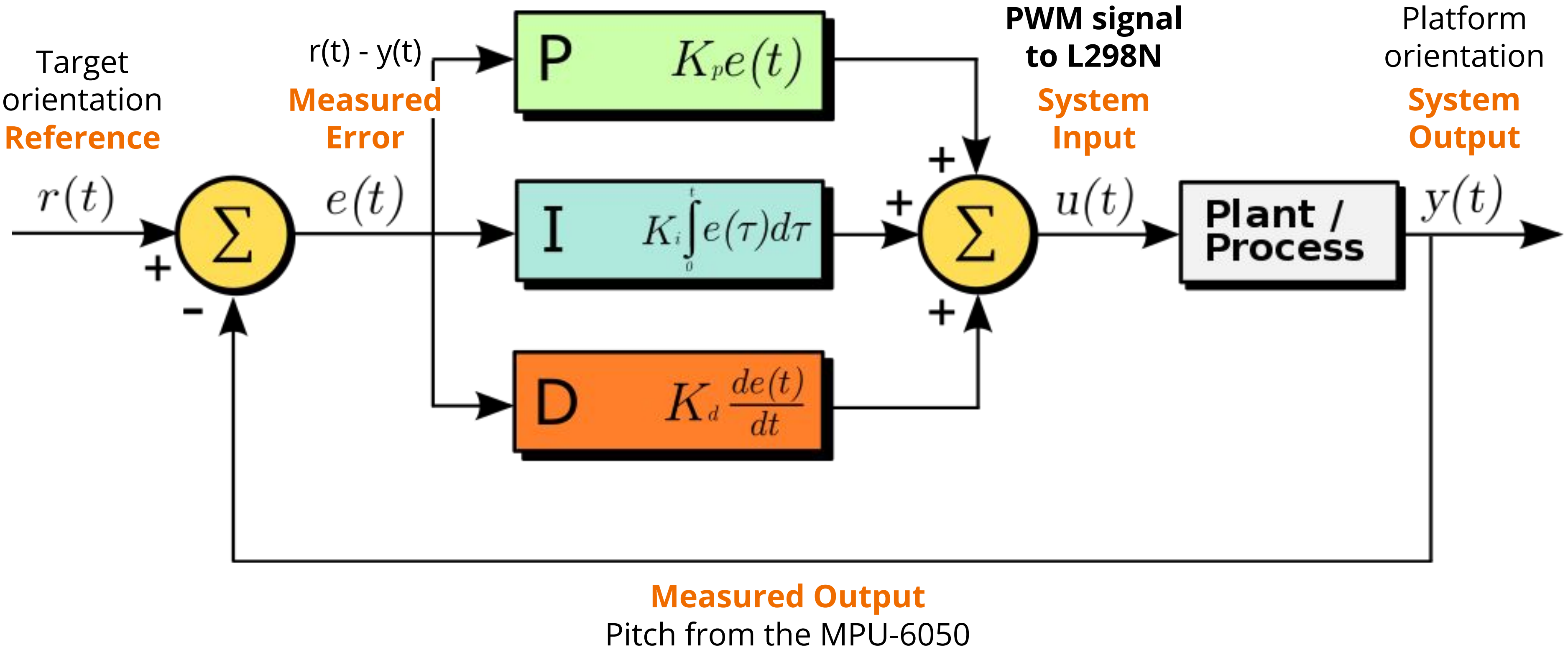
Target orientation
Reference
$r(t)$
+
-
Measured Error
$e(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
+
+
+
System Input
$u(t)$
Plant / Process
Platform orientation
System Output
$y(t)$
Measured Output
Pitch from the MPU-6050

PID Control



Target orientation
Reference
$r(t)$
+
-
Σ
r(t) - y(t)
Measured Error
$e(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau) d\tau$
D
$K_d \frac{de(t)}{dt}$
+
+
+
Σ
System Input
$u(t)$
Plant / Process
Platform orientation
System Output
$y(t)$
Measured Output
Pitch from the MPU-6050

PID Control

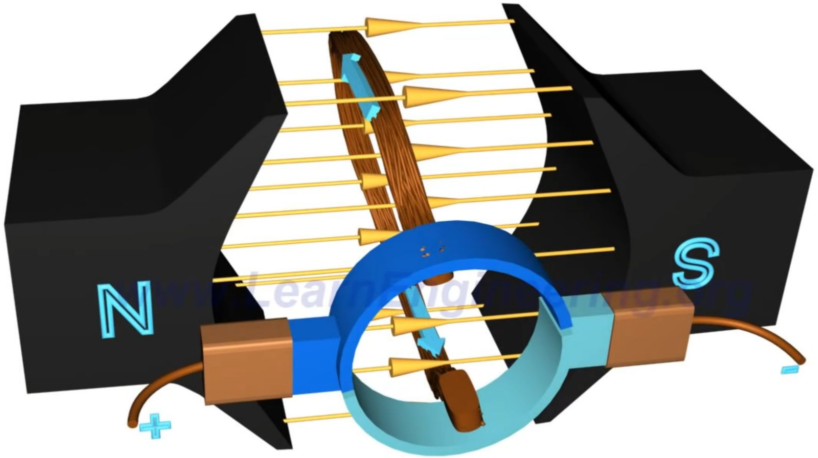

Target
orientation
Reference
r(t) - y(t)
Measured
Error
PWM signal
to L298N
System
Input
Platform
orientation
System
Output
$r(t)$
$e(t)$
P
$K_p e(t)$
I
$K_i \int_0^t e(\tau)d\tau$
D
$K_d \frac{de(t)}{dt}$
$u(t)$
Plant /
Process
$y(t)$
+
-
Measured Output
Pitch from the MPU-6050

Implementation

Materials

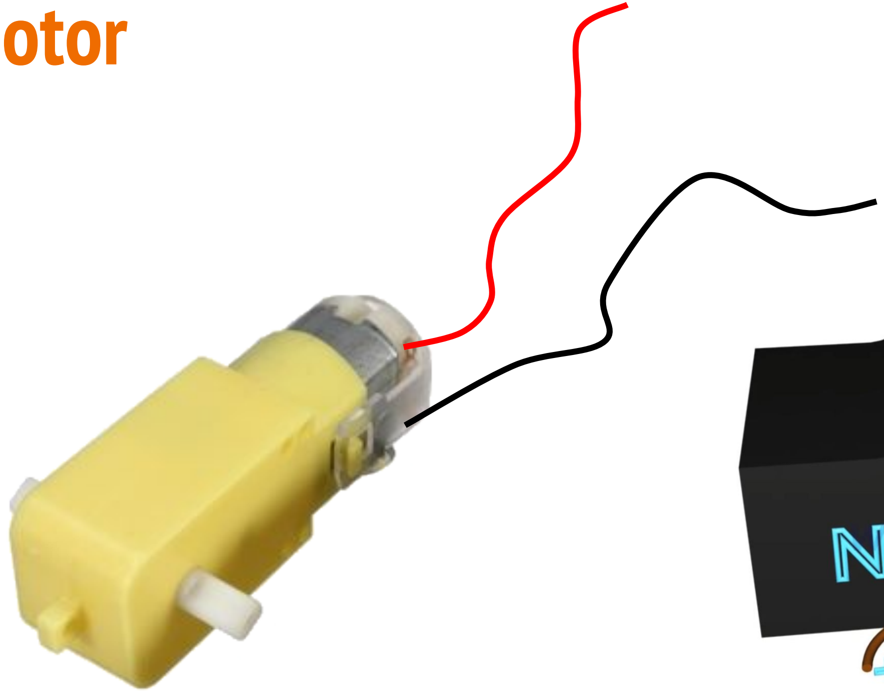
1. Arduino Nano x1
2. DC Motor x1
3. 9V Battery & Case x1
4. IMU (GY-521) x1
5. Motor Driver (L298N) x1
6. M3 Screws and Nuts (10, 15, 30) x2
7. DuPont Wire (Female-Female x7, Male-Female x1)
8. Laser Cutting Pieces
9. Steel Ball x1 **(please return it next week)**

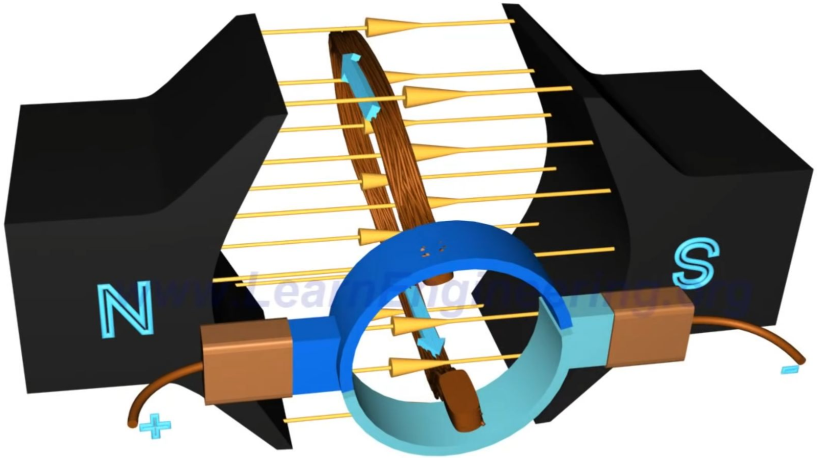
DC motor



https://youtu.be/LAtPHANEfQo

DC motor

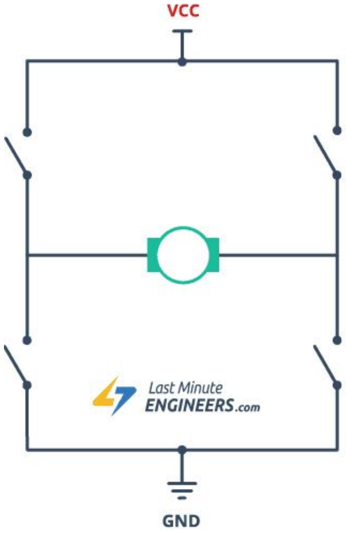




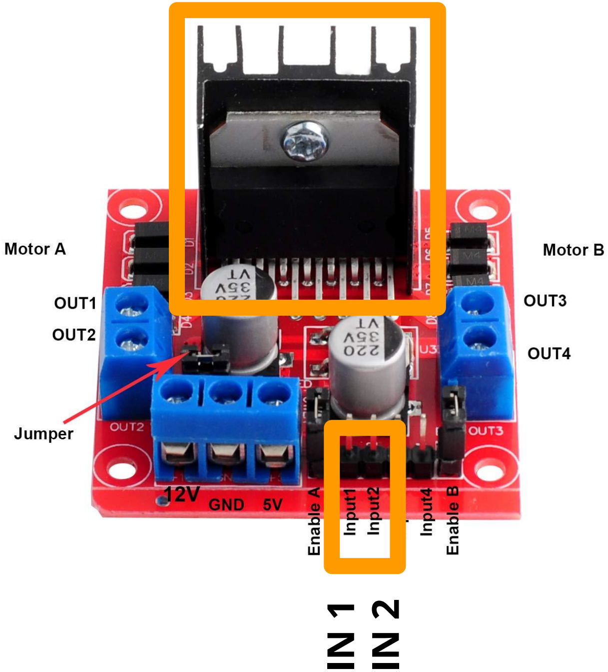
How do we reverse the polarity?

https://youtu.be/LAtPHANEfQo

L298N Motor Driver



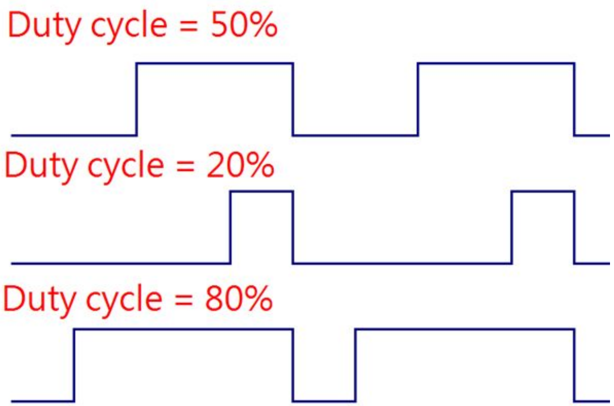
H-Bridge

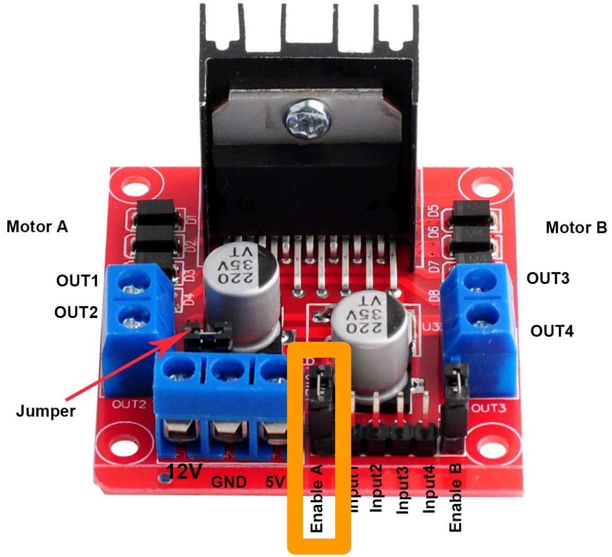


Behavior	IN1	IN2
Forward	HIGH	LOW
Backward	LOW	HIGH
Brake	LOW	LOW

L298N Motor Driver

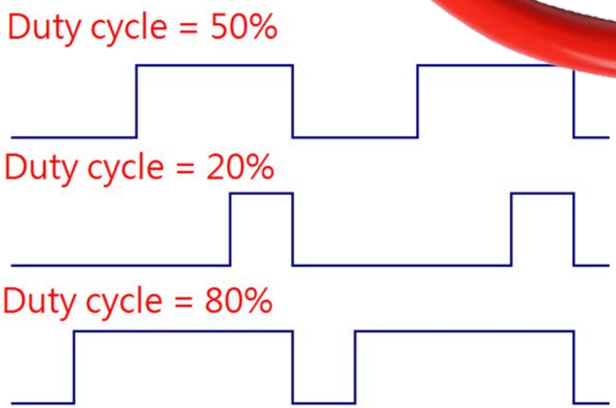
When the jumper is in place, the motor spins at full speed (5V).
Change the motor speed by sending PWM signals to the EN pins (ENA/ENB).

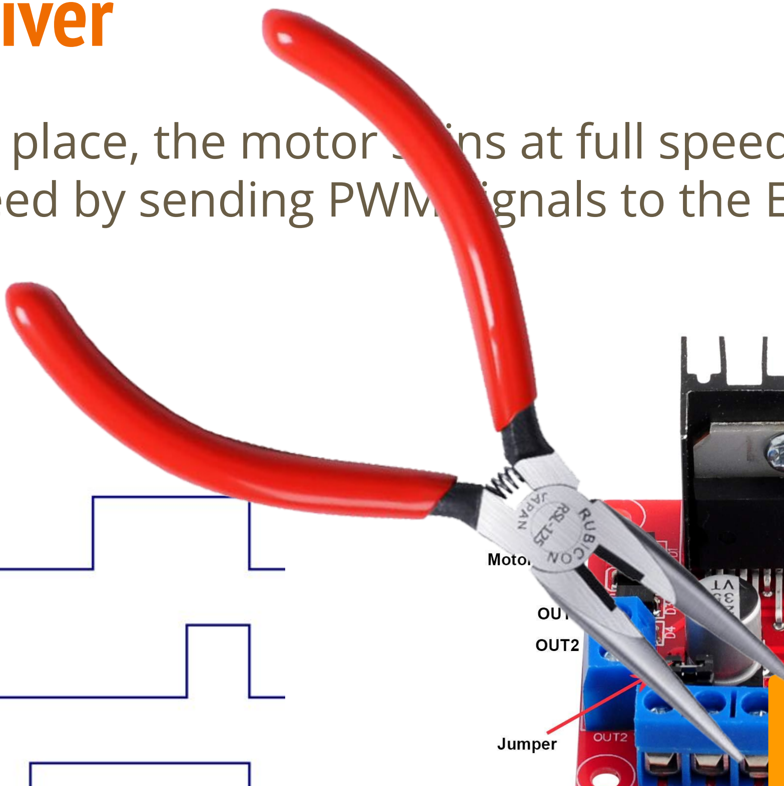


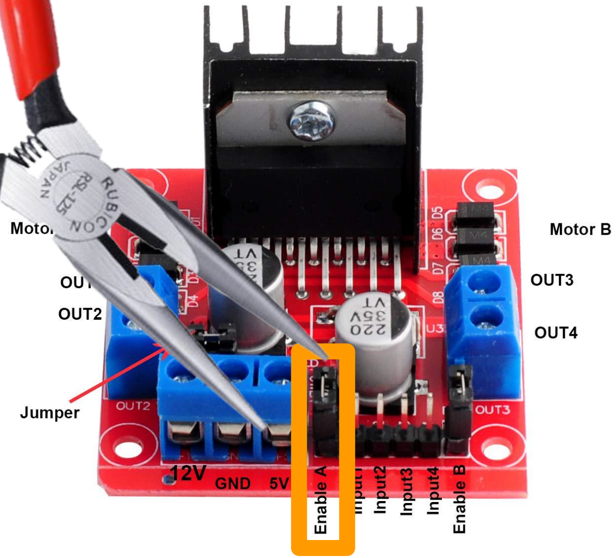


L298N Motor Driver

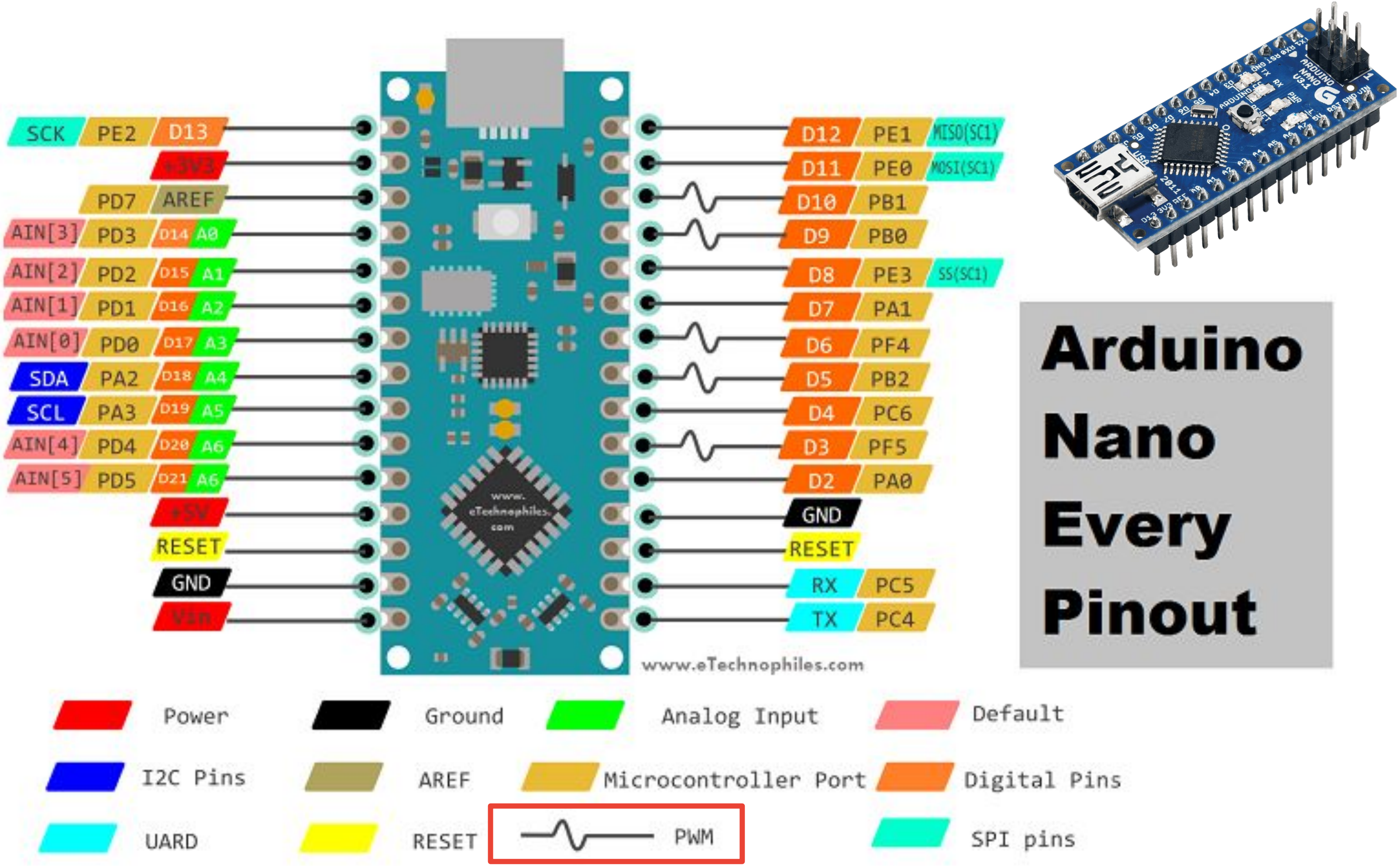
When the jumper is in place, the motor spins at full speed (5V).
Change the motor speed by sending PWM signals to the EN pins (ENA/ENB).

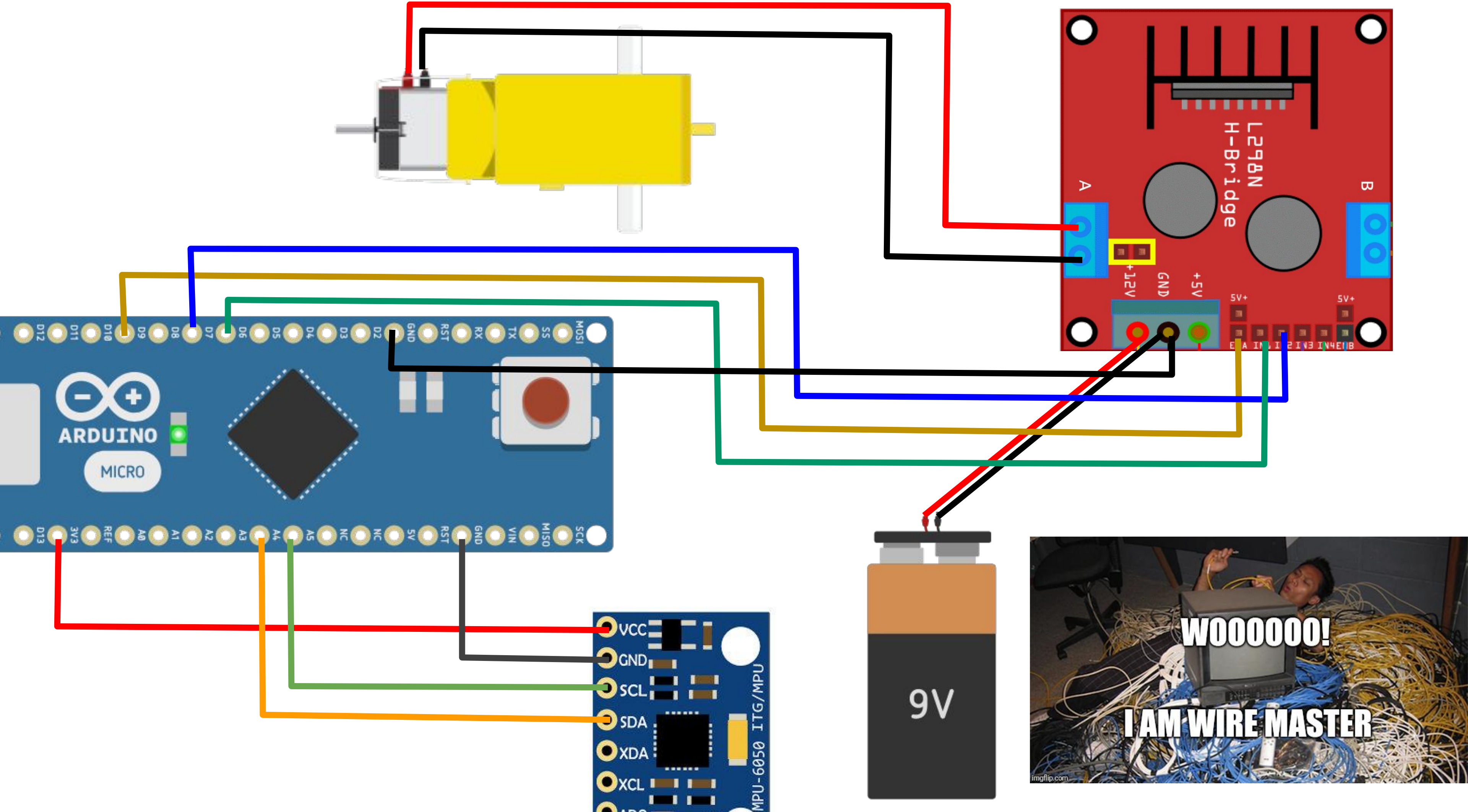






Arduino Nano Every Pinout

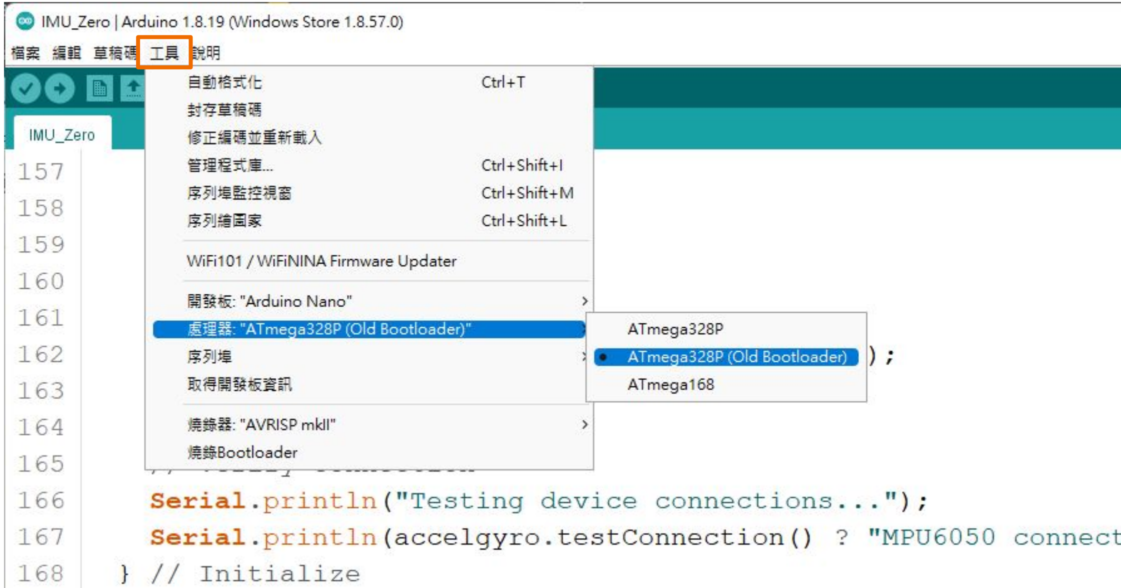



L298N
H-Bridge
A
B
+12V
GND
+5V
5V+
ENA IN1 IN2 IN3 IN4 ENB
ARDUINO
MICRO
D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX TX SS MOSI
D13 3V3 REF A0 A1 A2 A3 A4 A5 NC NC 5V RST GND VIN MISO SCK
VCC
GND
SCL
SDA
XDA
XCL
MPU-6050 ITG/MPU
9V
WOOOOOO!
I AM WIRE MASTER
imgflip.com

Troubleshooting for Arduino Nano

Make sure you select the right board and the right port. **If the sketch still can’t be uploaded…**

Tools > Processor > "ATMega328P (Old Bootloader)"



Control the DC Motor

```
int in1Pin = 6;
int in2Pin = 7;
int pwmPin = 9;

void setup() {
  pinMode(pwmPin, OUTPUT);
  pinMode(in1Pin, OUTPUT);
  pinMode(in2Pin, OUTPUT);

  SetDirection(0);
  analogWrite(pwmPin, 0);
}
```

Direction

Speed

```
void loop() {
  analogWrite(pwmPin, 50);
  delay(2000);
  analogWrite(pwmPin, 100);
  delay(2000);
  analogWrite(pwmPin, 255);
  delay(2000);
}

void SetDirection(int dir){
  if (dir == 0){
    digitalWrite(in1Pin, HIGH);
    digitalWrite(in2Pin, LOW);
  } else if (dir == 1){
    digitalWrite(in1Pin, LOW);
    digitalWrite(in2Pin, HIGH);
  }
}
```

NTU COOL > 文件 > Lab > Lab05_Stabilizer > Motor_Speed.ino

Check the MPU-6050

lab05 | Arduino 1.8.19 (Windows Store 1.8.57.0)

檔案 編輯 草稿碼 工具 說明

lab05 §

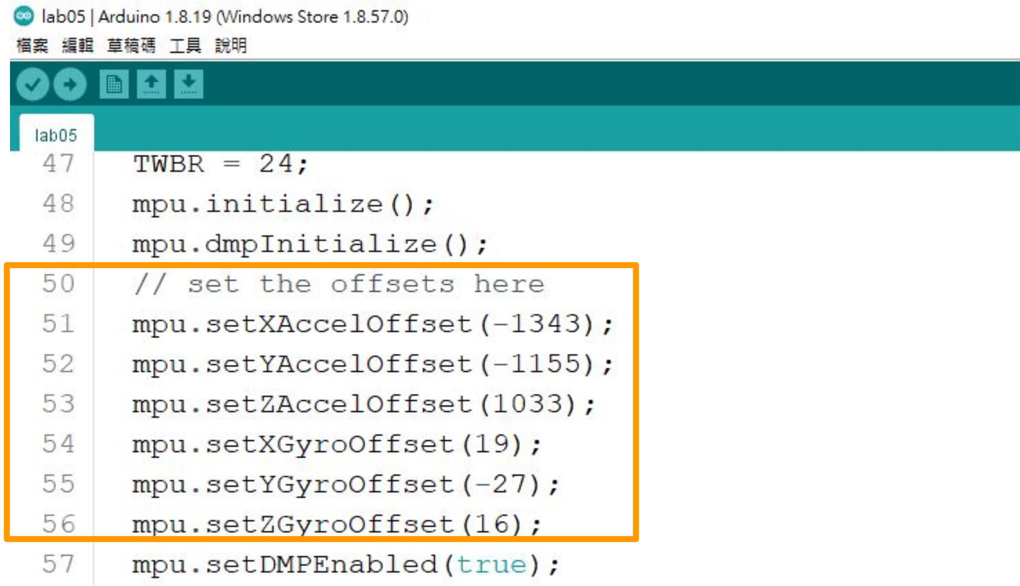
```
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

        Serial.print("ypr\t");
        Serial.print(ypr[0]*180/PI);
        Serial.print("\t");
        Serial.print(ypr[1]*180/PI);
        Serial.print("\t");
        Serial.print(ypr[2]*180/PI);
        Serial.println();
```

NTU COOL > 文件 > Lab > Lab05_Stabilizer > lab05.ino

Calibrate the MPU-6050

You can get the MPU-6050 offsets using the sample code (navigate to “File > Examples > MPU6050 > IMU_Zero”). [Reference]

Then set the offsets accordingly.



Control the DC Motor using PID Control

```
float error;
float prev_error;
float kp = 0;
float ki = 0;
float kd = 0;
int dt = 10;              // Sampling period
float setPoint = 0;       // Desired pitch value
float P, I, D, PID;
float time;
```

Tune the PID coefficients here

NTU COOL > 文件 > Lab > Lab05_Stabilizer > lab05.ino

Control the DC Motor using PID Control

Compute the PID coefficients based on measured errors. Constrain the PID outputs within [-255, 255] (**sign** → direction; **number** → PWM signal).

```
        error = (ypr[1] * 180 / M_PI) - setPoint;
        P = kp * error;
        I += ki * error;
        D = kd * ((error - prev_error) / dt);
        PID = constrain(P + I + D, -255, 255);
```

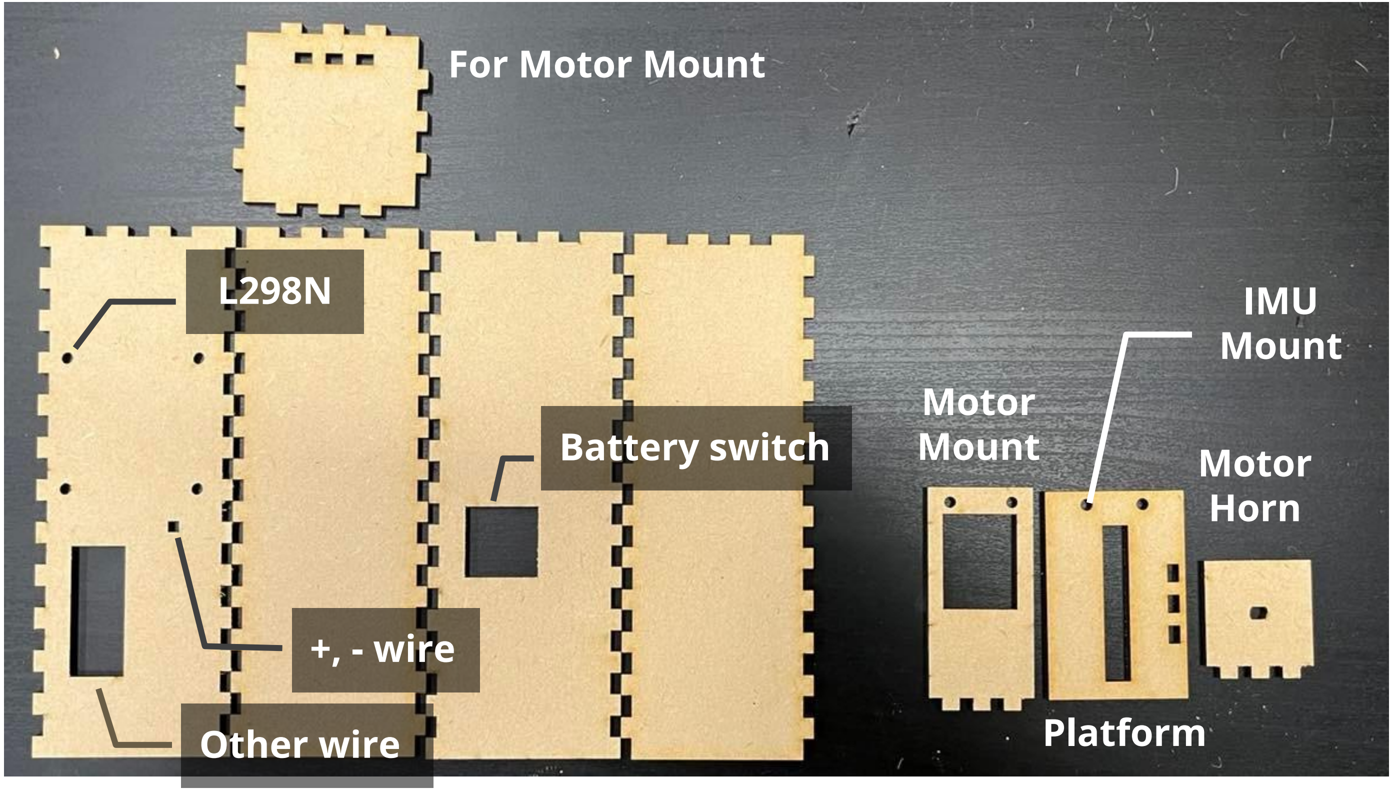
NTU COOL > 文件 > Lab > Lab05_Stabilizer > lab05.ino

Control the DC Motor using PID Control

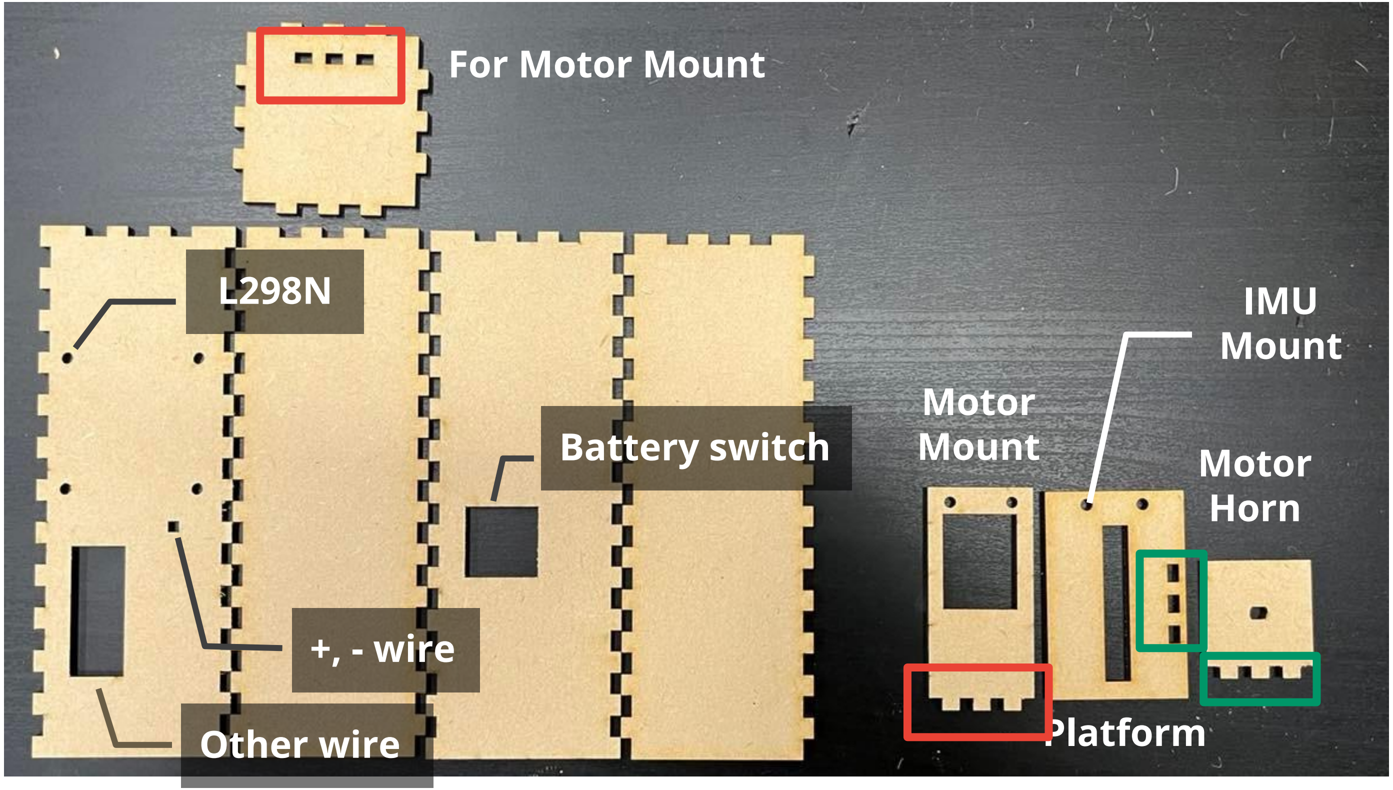
```
    // handle the direction
    if (PID > 0) {
      digitalWrite(in1, LOW);
      digitalWrite(in2, HIGH);
    } else {
      digitalWrite(in1, HIGH);
      digitalWrite(in2, LOW);
    }

    // send PWM signal
    analogWrite(enA, speed);   // actuate the motor
  }
```

NTU COOL > 文件 > Lab > Lab05_Stabilizer > lab05.ino

Laser Cutting Pieces



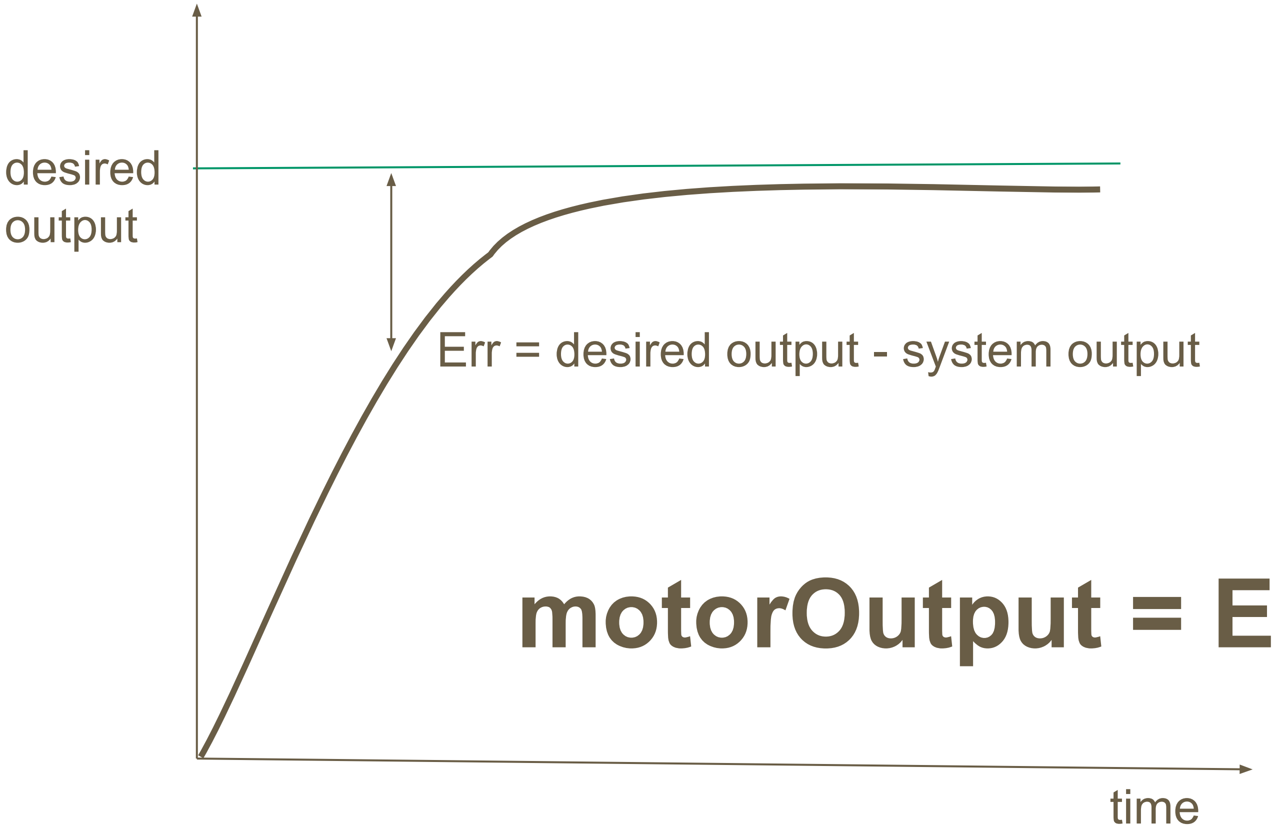
Laser Cutting Pieces



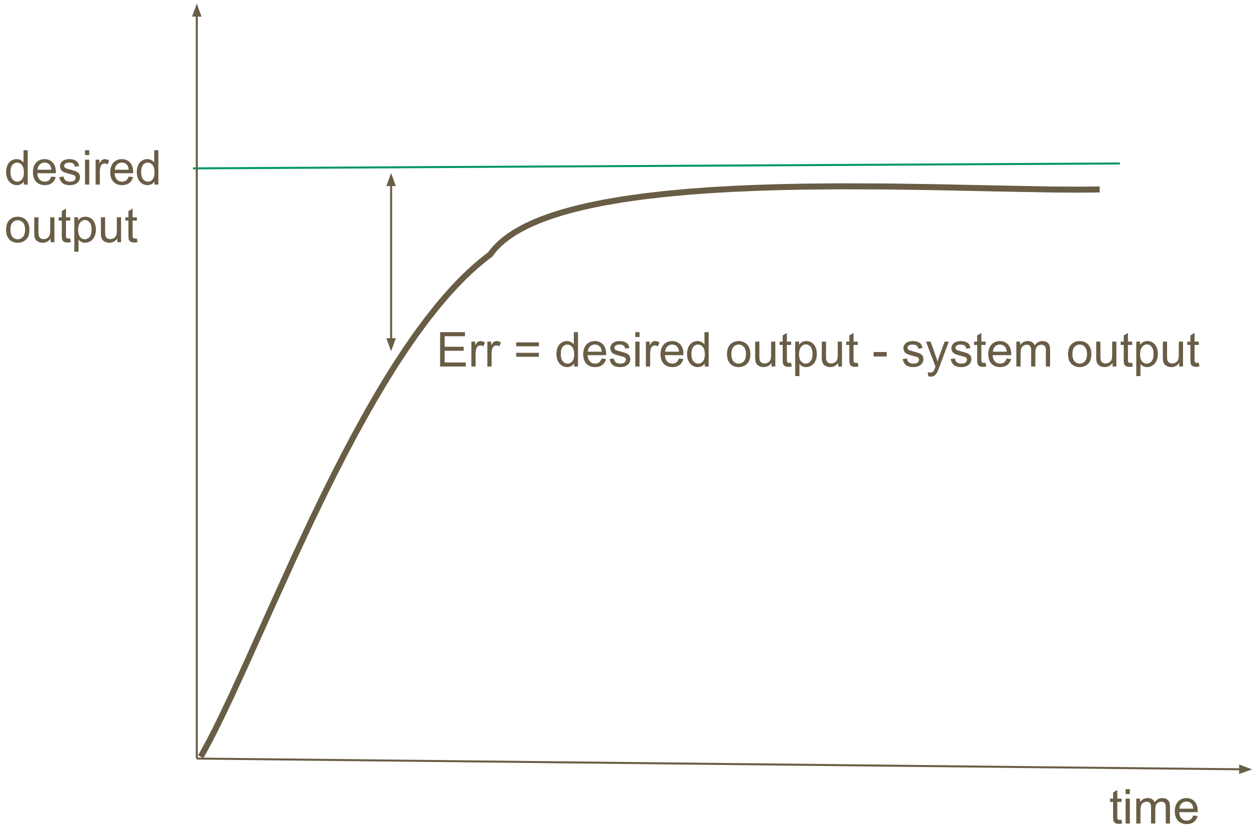
Let's start with a P controller

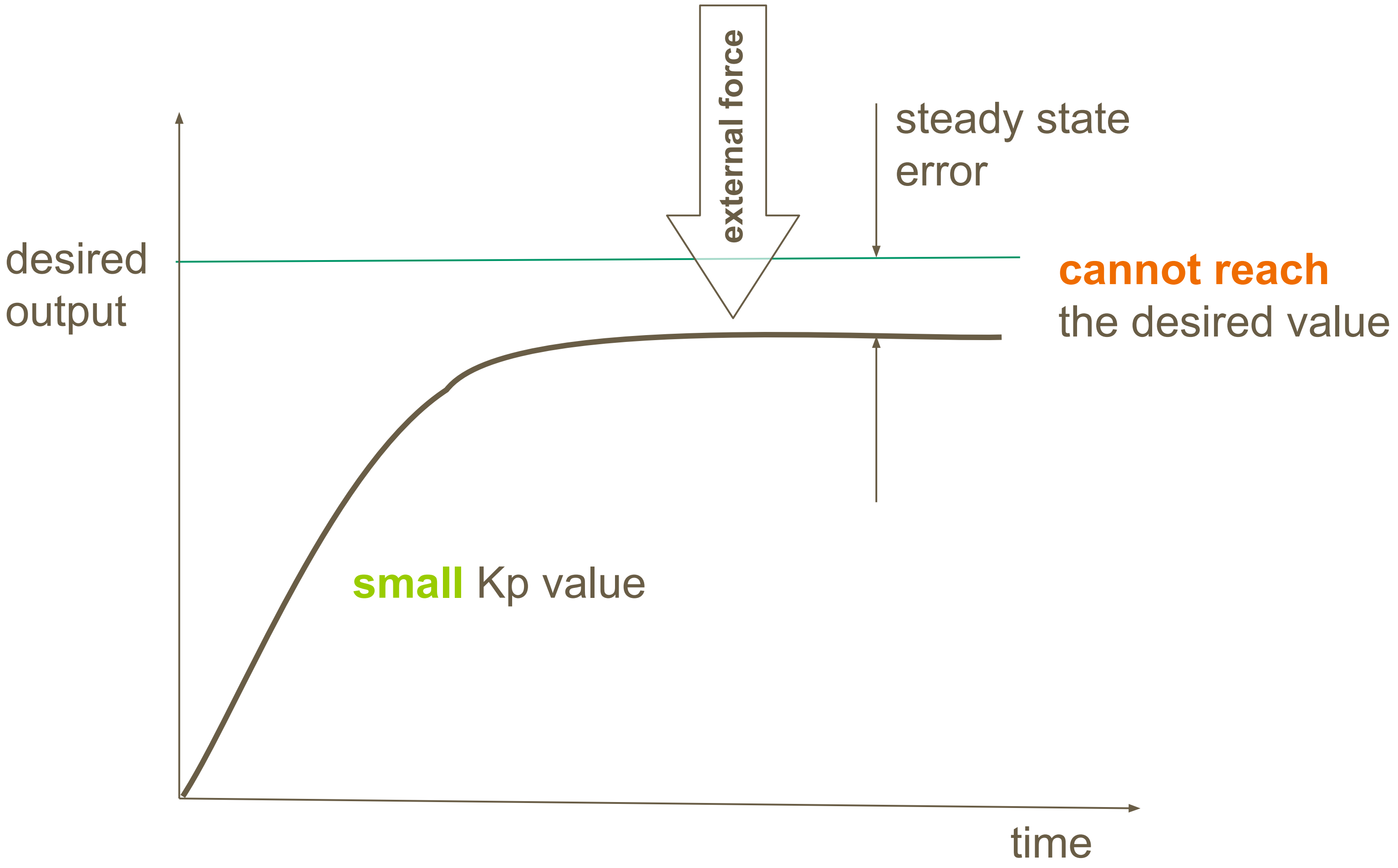
$\mathbf{K_p > 0}$, $K_i = 0$, $K_d = 0$

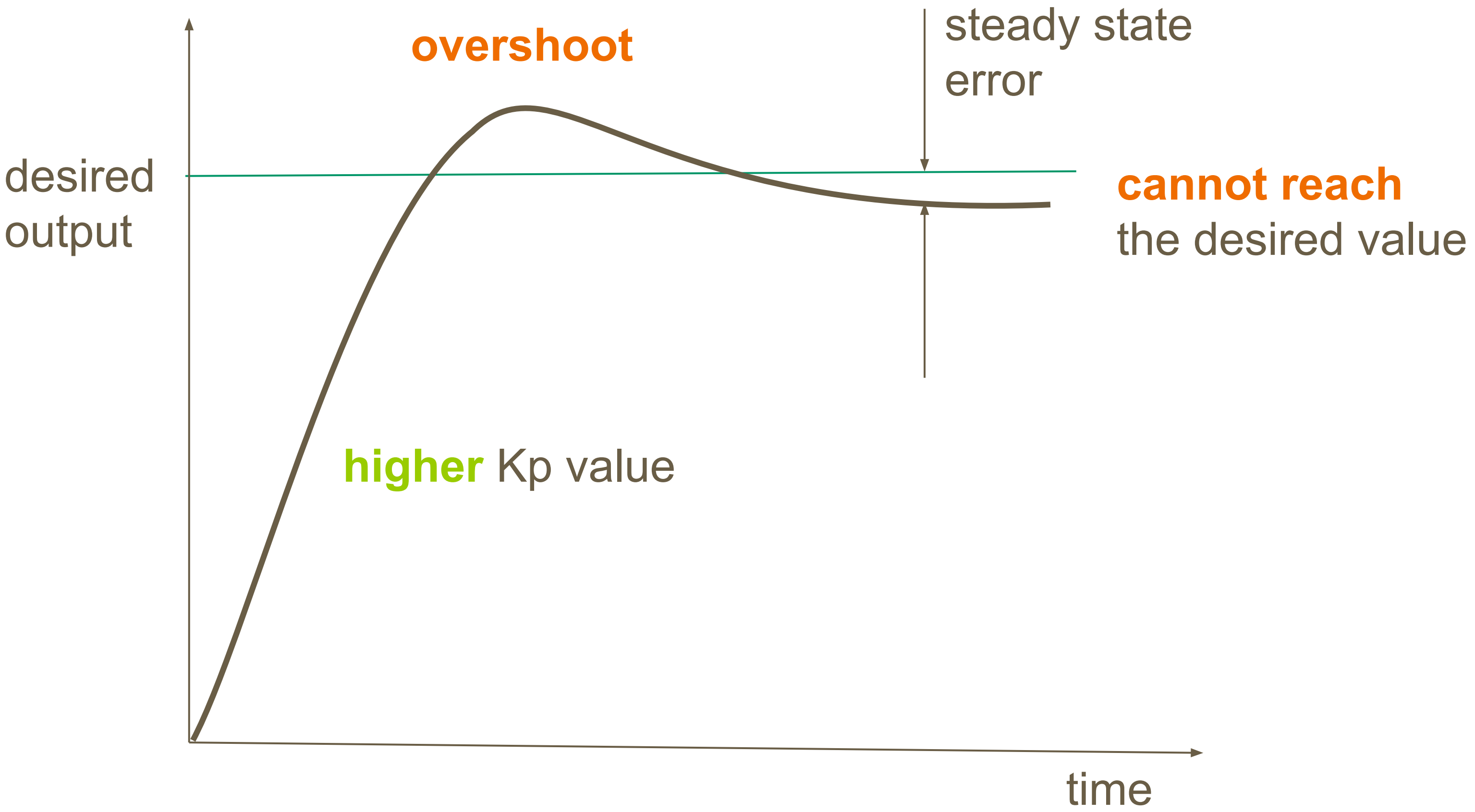
P:: the controller output is proportional to the **current error**

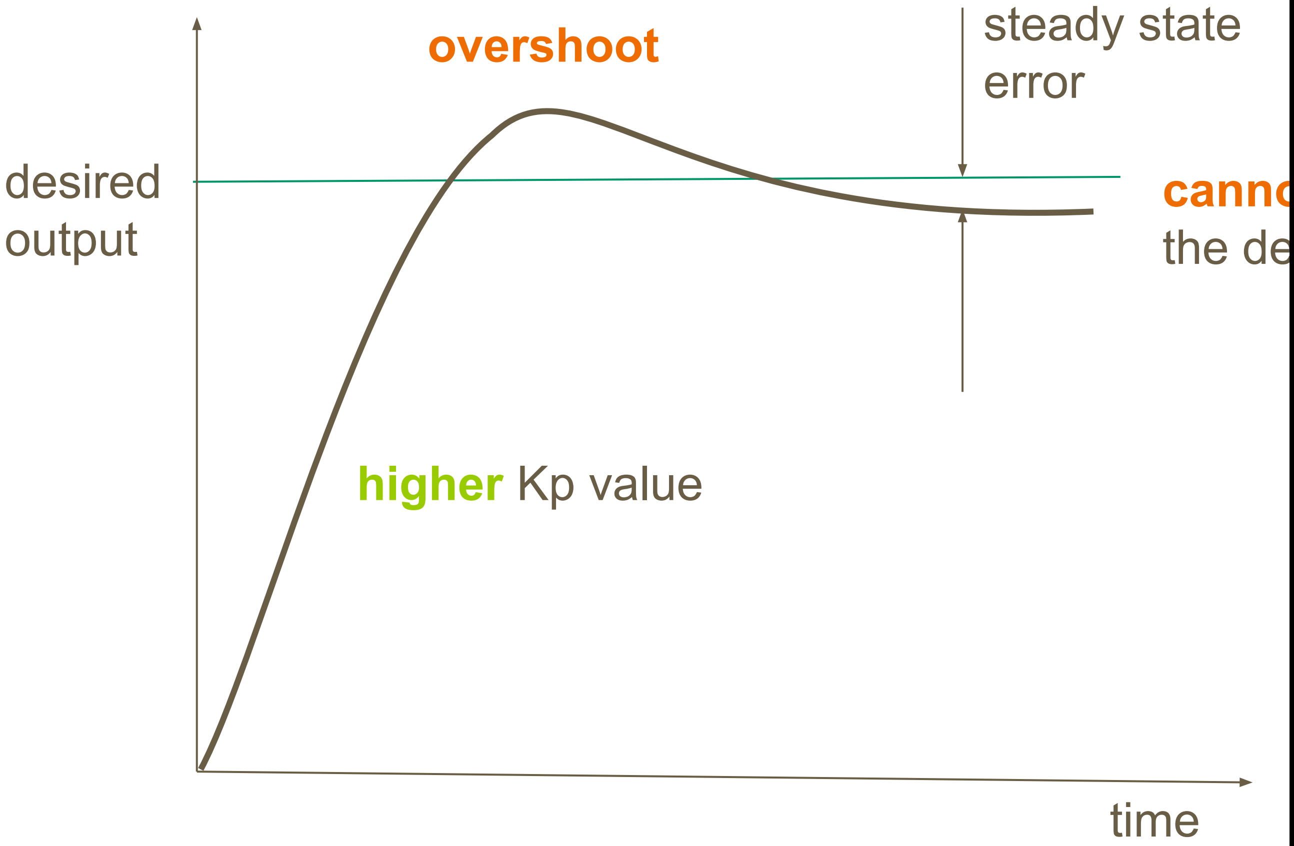


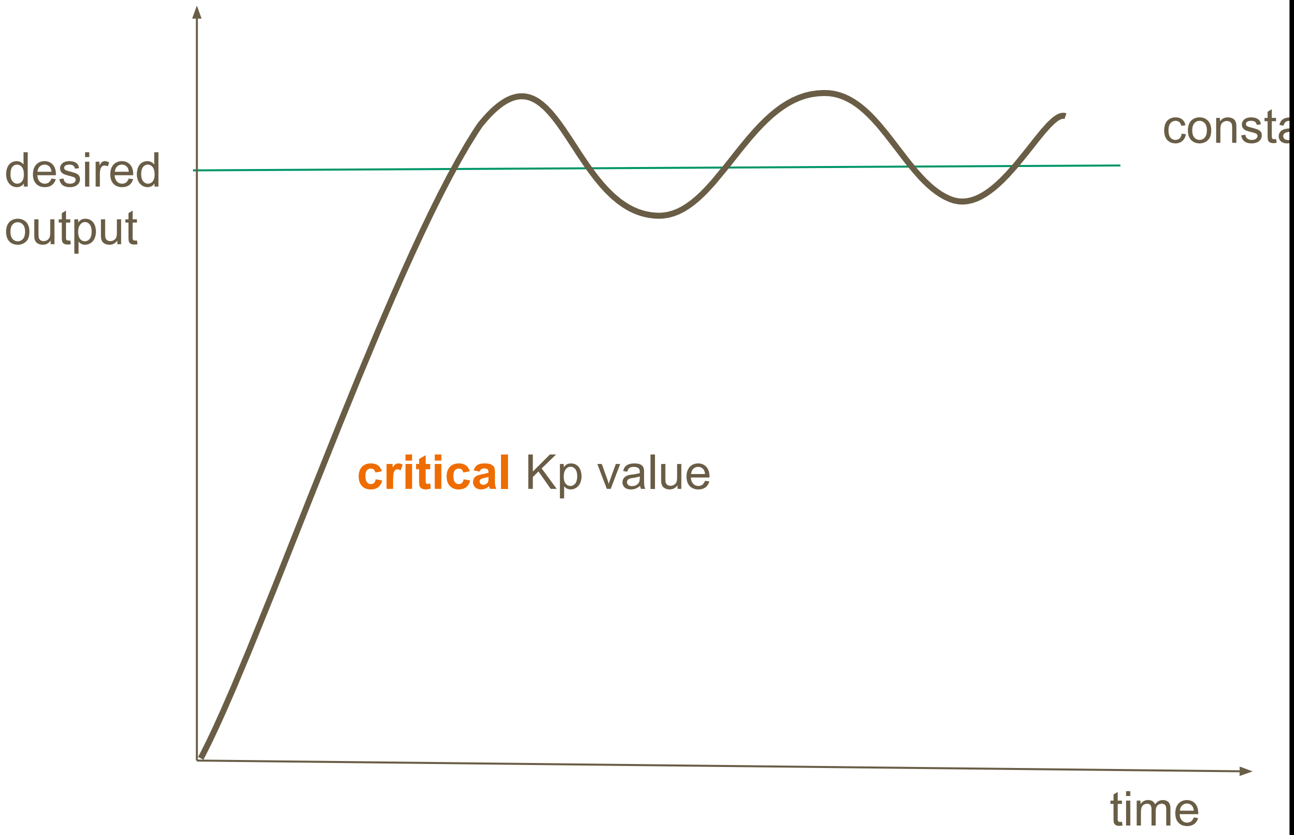
motorOutput = Err * Kp


desired output
Err = desired output - system output
time


external force
steady state
error
desired
output
cannot reach
the desired value
small Kp value
time


overshoot
steady state
error
desired
output
cannot reach
the desired value
higher Kp value
time


overshoot
steady state error
desired output
[redacted]
higher Kp value
time


consta[redacted]
desired
output
critical Kp value
time

bang-bang (on/off) control?

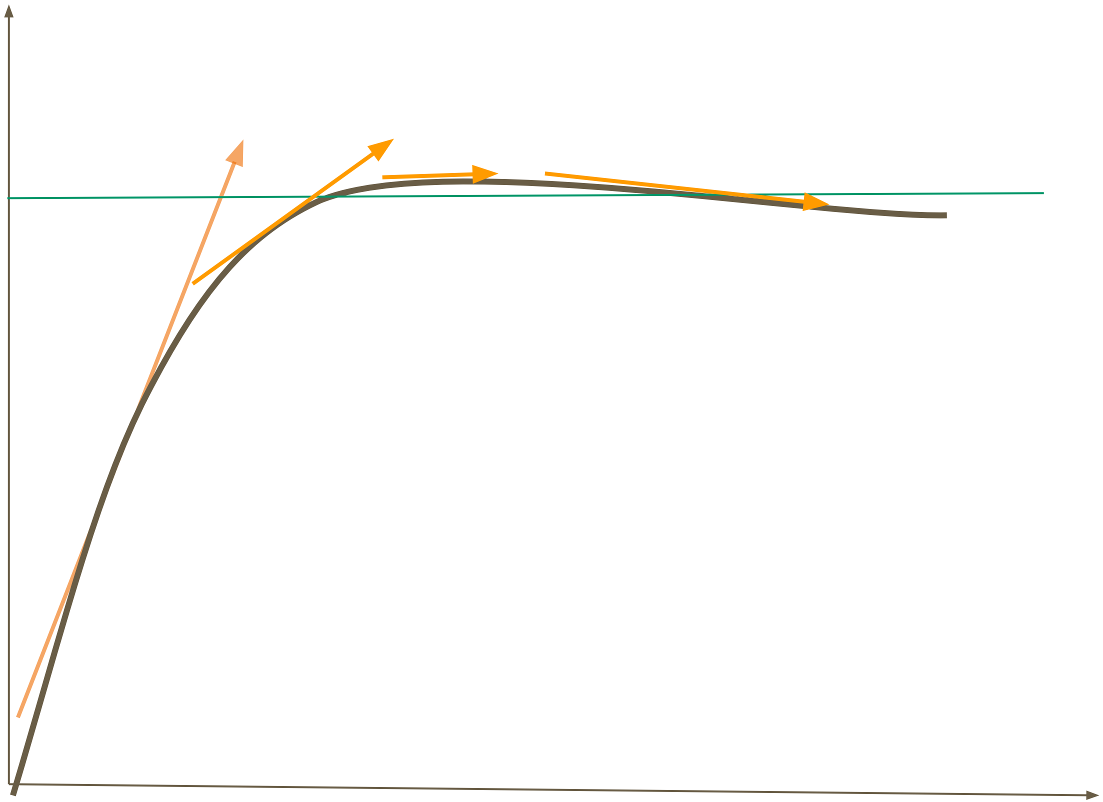
Set $K_p \sim \infty$, $K_i = 0$, $K_d = 0$ to simulate an on/off control

PD control - reduce overshoot

$K_p > 0$, $K_i = 0$, **$K_d > 0$**

provides damping and flattens the *e(t)* into a horizontal line

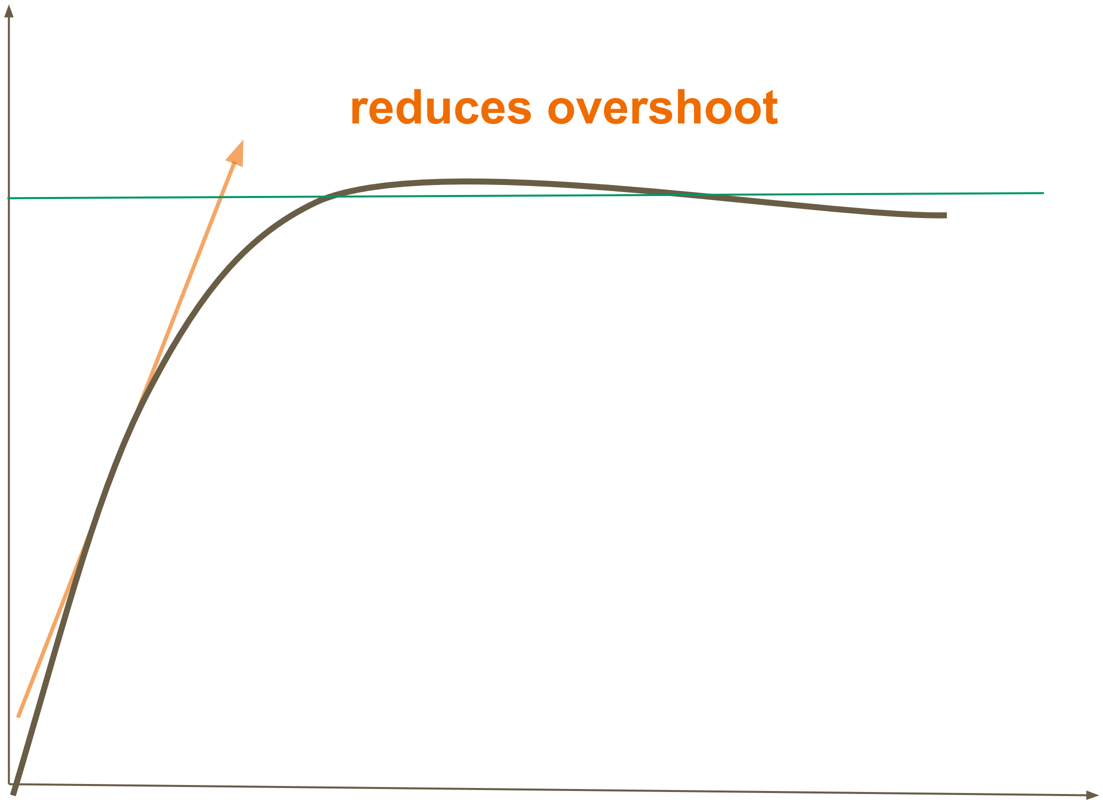
D:: predicts **future errors** based on the current rate of change



as before, P = Err * Kp

D = (error – prevError)/dt * Kd

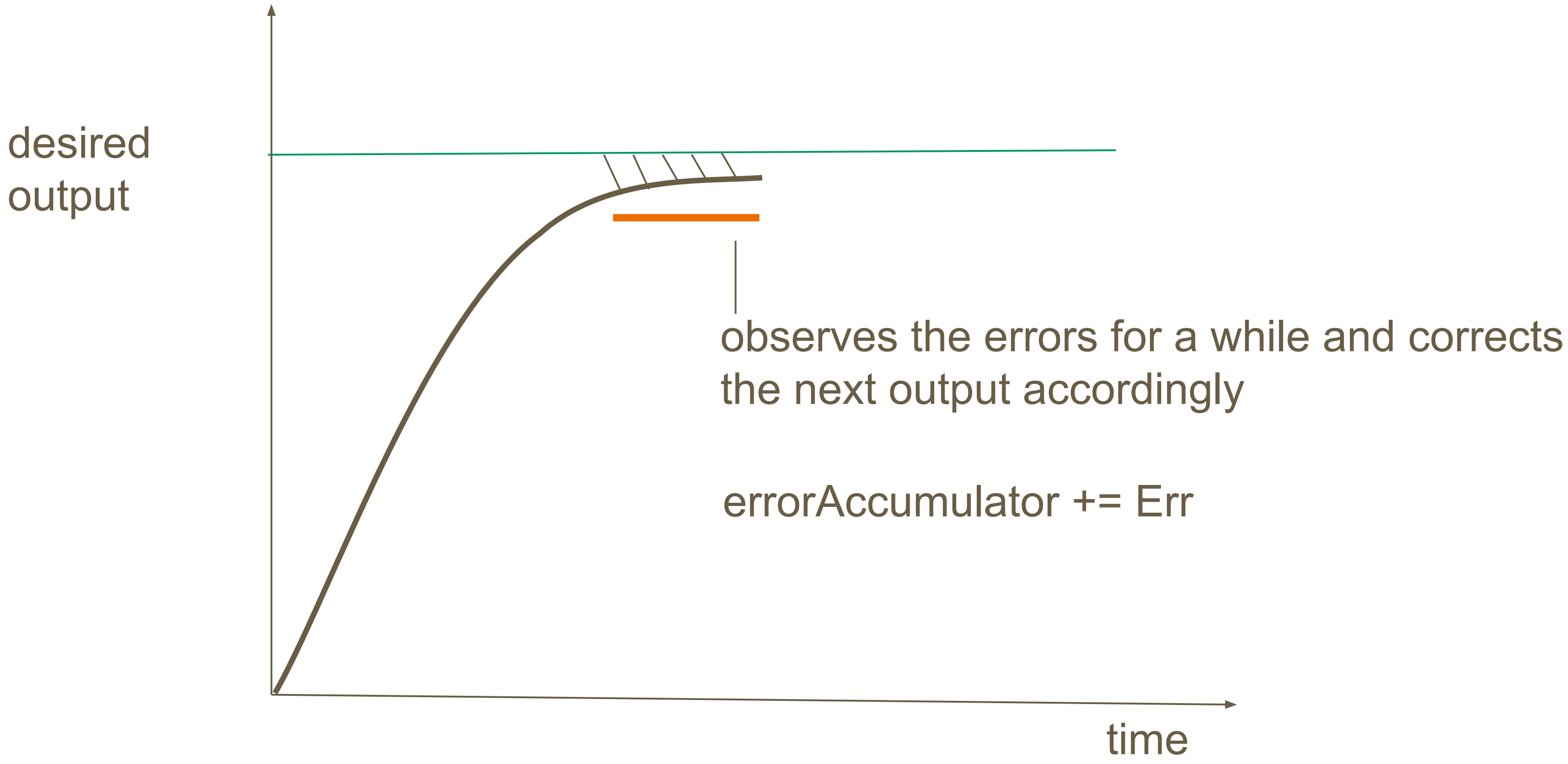
motorOutput = P + D


reduces overshoot

PI control to eliminate steady state error

$K_p > 0, K_i > 0, K_d > 0$

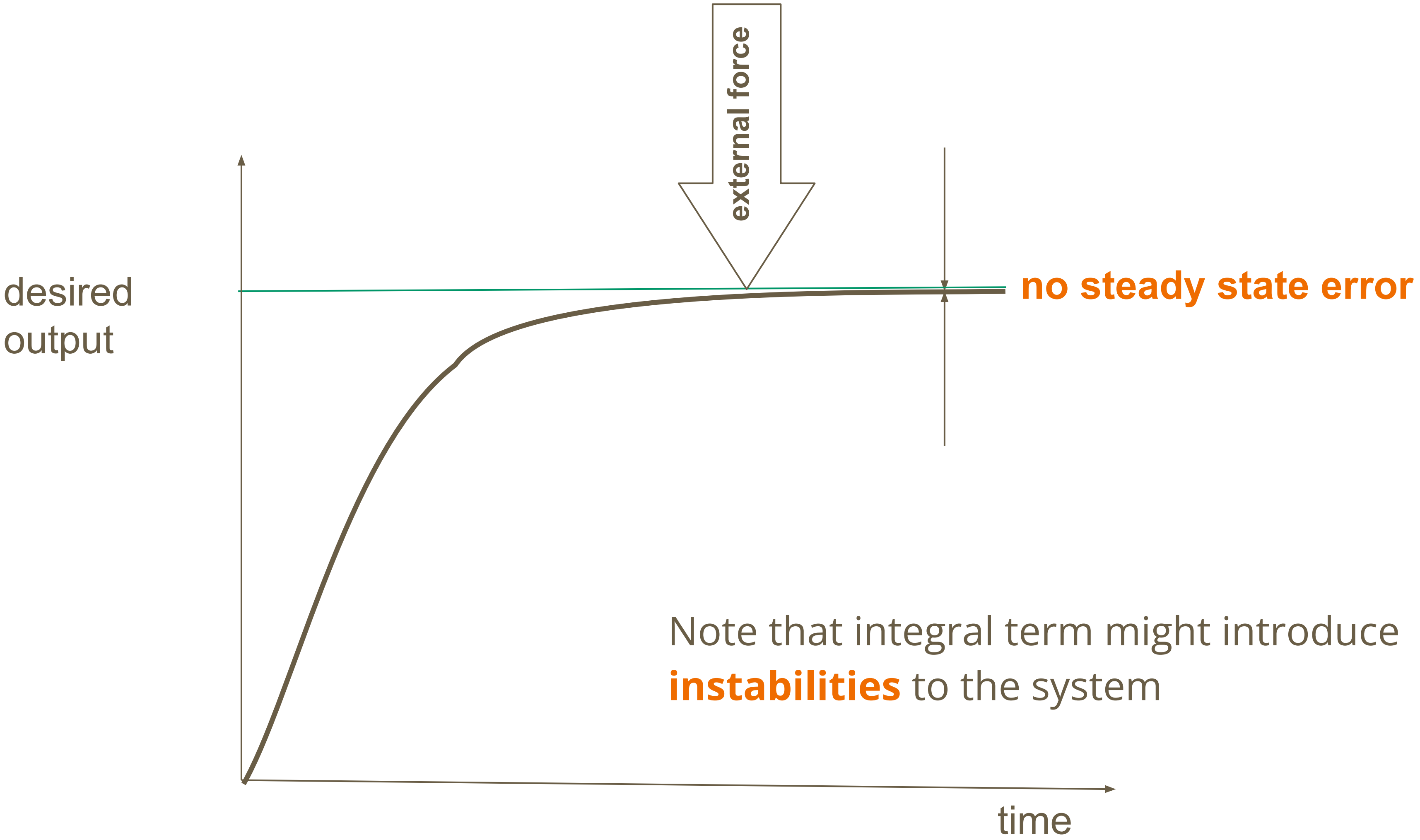
I:: responsible for past errors and can eliminate the steady state error


desired
output
observes the errors for a while and corrects
the next output accordingly
errorAccumulator += Err
time

as before, P = Err * Kp

motorOutput = P + I

errorAccumulator += Err



Note that integral term might introduce **instabilities** to the system

Tune the PID Controller

Tuning the parameters

1. Set all gains to zero
2. Increase the **P** gain until the system starts to oscillate
3. Increase the **D** gain until the the oscillations go away
4. Repeat steps 2 and 3 until increasing the **D** gain does not stop the oscillations
5. Now, if you need to eliminate the steady state error, increase **I** gain slowly until it brings you to the setpoint. Note that this will again introduce some oscillations in the system so you might have to reduce **P** and increase **D** a bit
6. Iterate...

There are many other tuning methods, but the best parameters vary with different power, motors, load...

Notes

1. Test the output values before actually connecting the motor
2. Turn off the battery when the stabilizer is not in use
3. Quickly turn off the battery when the motor is abnormal

Assignment Requirements

- Basic Requirements (video)
  - Demonstrate (1) oscillation & overshoot, (2) steady state error, and (3) proper control (no oscillation and steady state error) of the platform orientation
  - Optimize the parameters to minimize settling time and reduce overshoot
  - Submit the YouTube Link (less than 5 min) to NTU COOL
  - Please add description and timestamps to tell us what you did
- Bonus (competition)
  - Rotate the stabilizer **(0° → 90° → 0°**, similar to the GIF shown on page 2) while balancing the steel ball **7 times**, and tell us the time used in the description.
- **Deadline: 11/23 23:59**

Lab Satisfaction Survey

Lab 05 Feedback