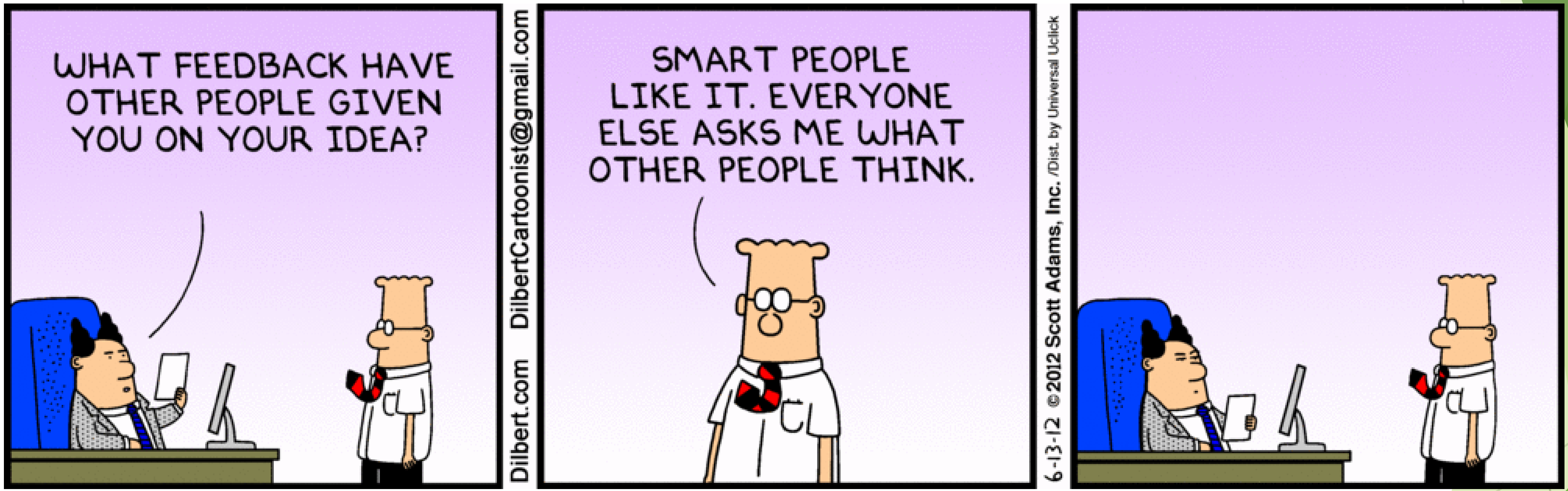
Feedback Control



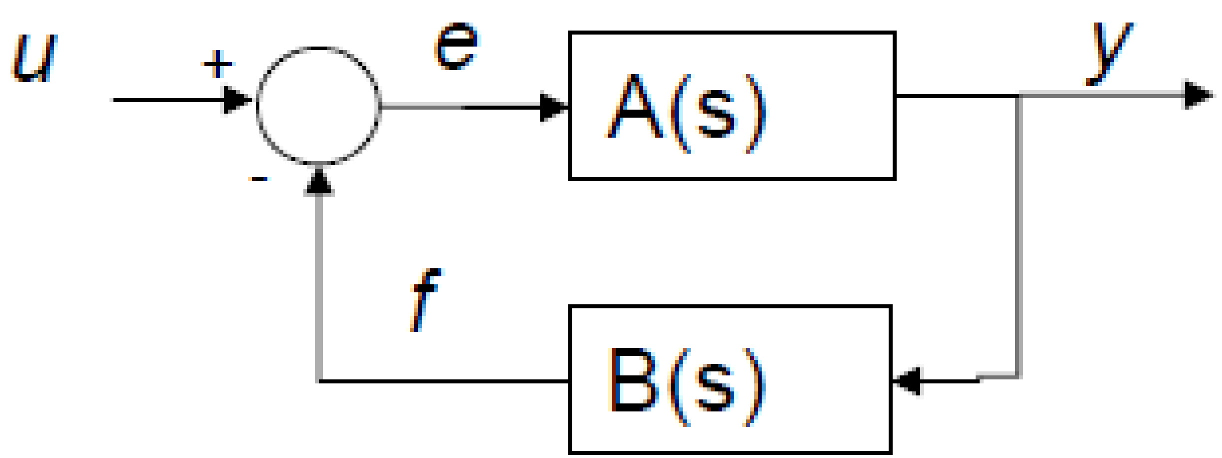
Outline

- Background & Motivation
- Theory
- Applications
- Takeaways & Questions

What is Feedback Control?

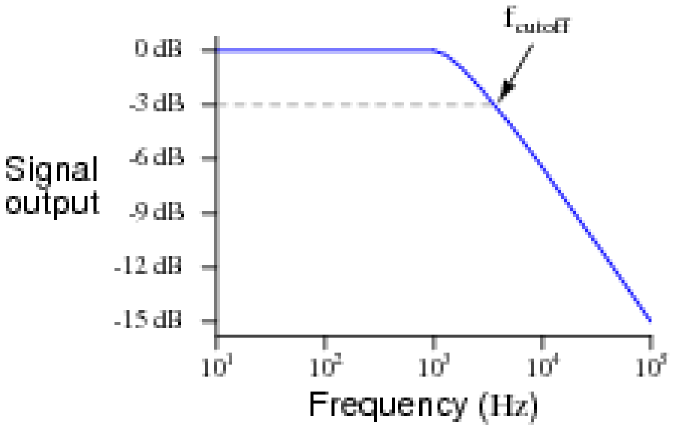
Definition

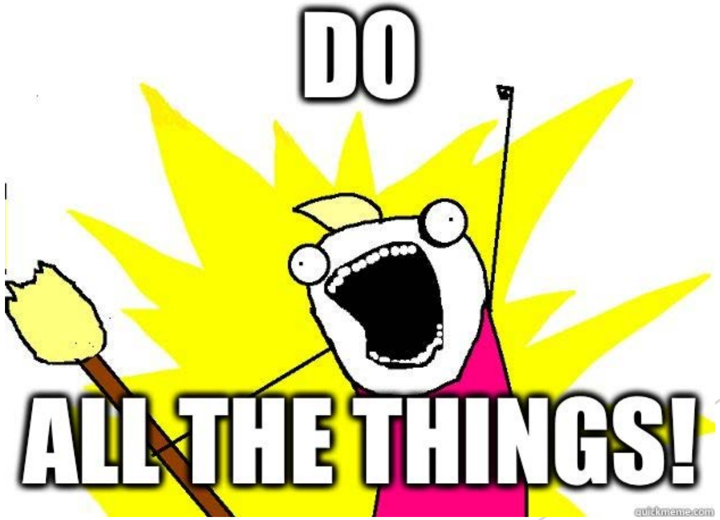
Optimizing a system's performance by feeding its output back into its input



Motivation

- Wider bandwidth
- Less interference
- More stability
- So much more!!



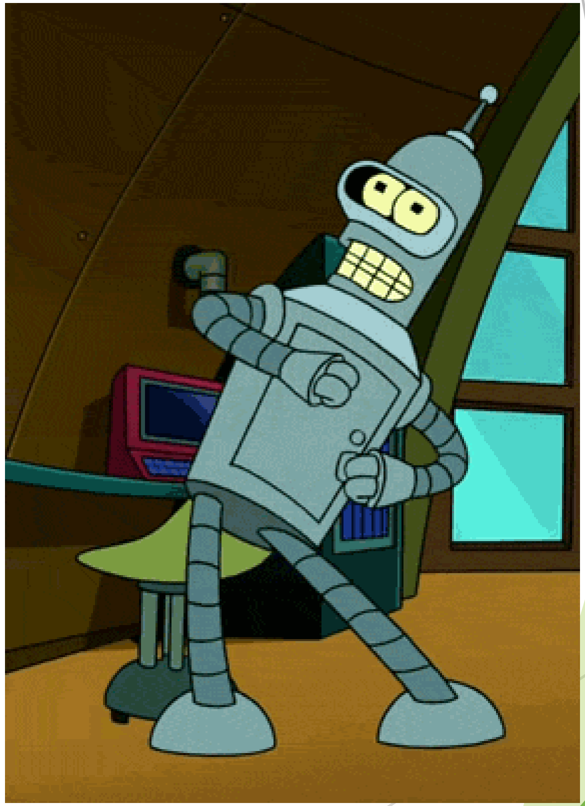


https://www.allaboutcircuits.com/worksheets/passive-filter-circuits/

What are applications for YOU?

Applications for YOU

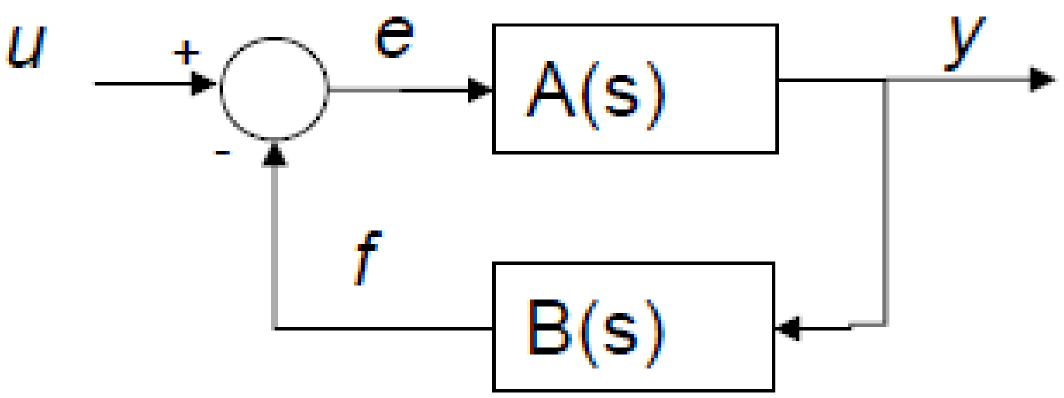
- Better treasure detection
- Improved line/wall sensors
- Faster robots
- So much more!!



https://giphy.com/gifs/dancing-happy-mIZ9rPeMKefm0

THEORY

Theory



$$y = A(s)e$$

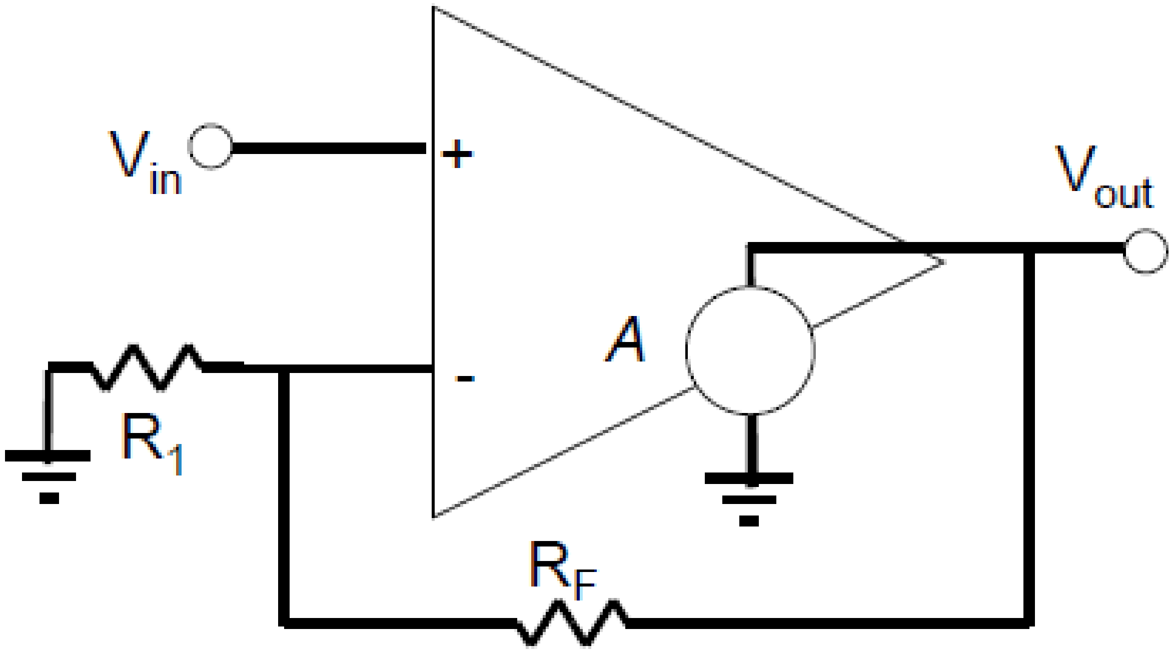
$$f = B(s)y$$

$$e = u - f$$

$$\begin{aligned} y &= A(s)e \\ &= A(s)\,[u - f] \\ &= A(s)\,[u - B(s)y] \\ &= A(s)u - A(s)B(s)y \end{aligned}$$

$$H(s) = \frac{y}{u} = \frac{A(s)}{1 + A(s)B(s)}$$

Non-inverting Op-amp



What's the equation for Vout?

Application of Theory

$$u = V_{in}$$
$$y = V_{out}$$
$$A(s) = A_v$$
$$B(s) = \frac{R_1}{R_1 + R_f}$$

$$u = V_{in}$$
$$y = V_{out}$$
$$A(s) = A_v$$
$$B(s) = \frac{R_1}{R_1 + R_f}$$

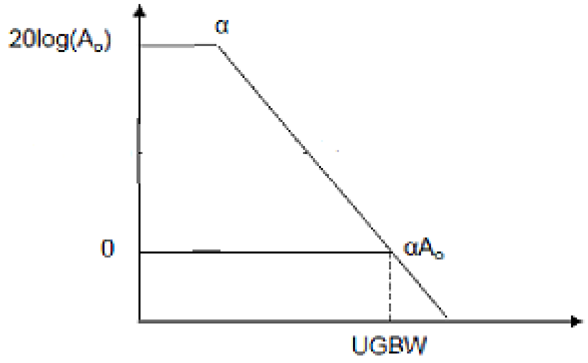
$$\frac{V_{out}}{V_{in}} = \frac{A(s)}{1 + A(s)B(s)}$$

$$= \frac{A_v\,(R_1 + R_f)}{(R_1 + R_f + A_v R_1)}$$

For large Av

$$H(s) = \frac{R_1 + R_f}{R_1} = 1 + \frac{R_f}{R_1}$$

Low-pass Filter



$$A(s) = \frac{A_0}{1 + \frac{s}{\alpha}}$$

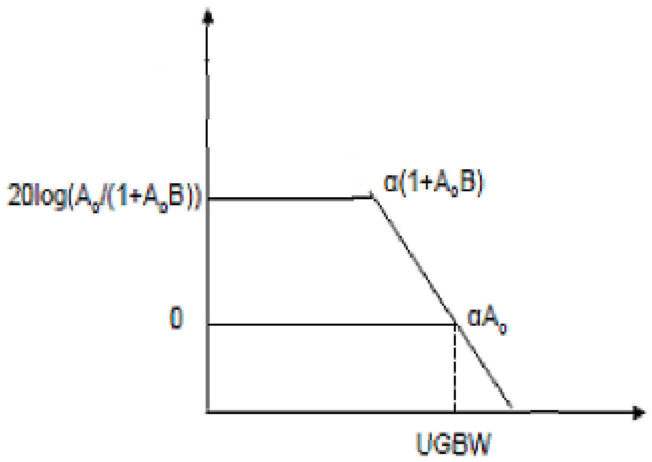
$$B(s) = 0$$

$$H(s) = \frac{A(s)}{1 \ + A(s)B(s)} = \frac{A_0}{1 + \frac{s}{\alpha}}$$

DC Gain : A_0

Pole: $\omega_p = \alpha$

Filter with Feedback



$$A(s) = \frac{A_0}{1+\frac{s}{\alpha}}$$

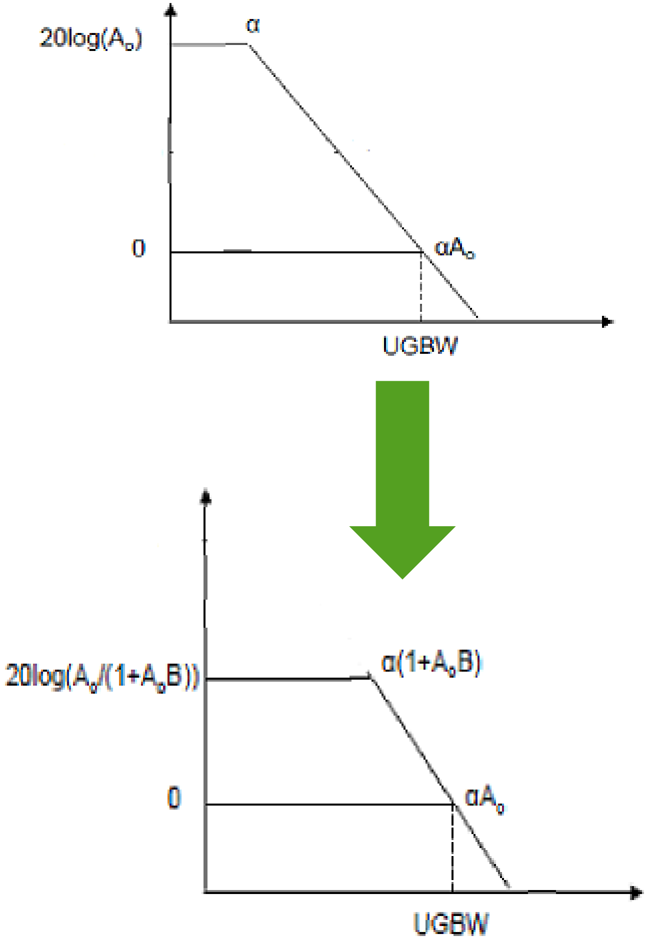
$$B(s) = 1$$

$$H(s) = \frac{A(s)}{1 + A(s)B(s)} = \frac{\frac{A_0}{1+\frac{s}{\alpha}}}{1+\frac{A_0 B}{1+\frac{s}{\alpha}}} = \frac{A_0}{A_0 B + 1 + \frac{s}{\alpha}}$$

DC Gain : $\frac{A_0}{A_0 B+1}$

Pole: $\omega_p = \alpha(1 + A_0 B)$

Feedback Consequences



DC Gain : A_0

Pole: $\omega_p = \alpha$

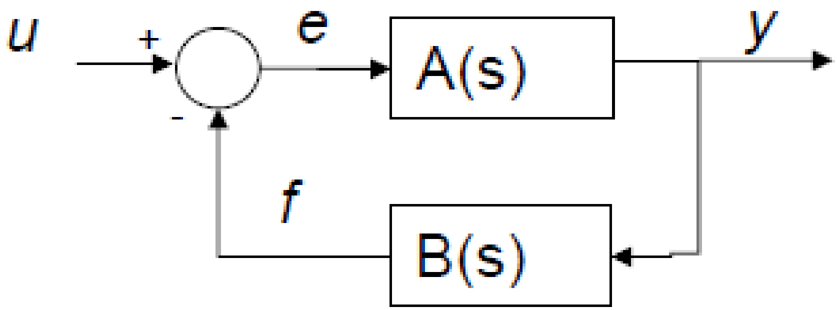
DC Gain : $\frac{A_0}{A_0 B + 1}$

Pole: $\omega_p = \alpha(1 + A_0 B)$

Feedback trades lower gain for higher bandwidth!

Theory Review

- Error Correction by feeding output back to input
- Applications for bandwidth, interference, and stability
- Can be used as a circuit analysis tool
- Trades lower gain for higher frequency range



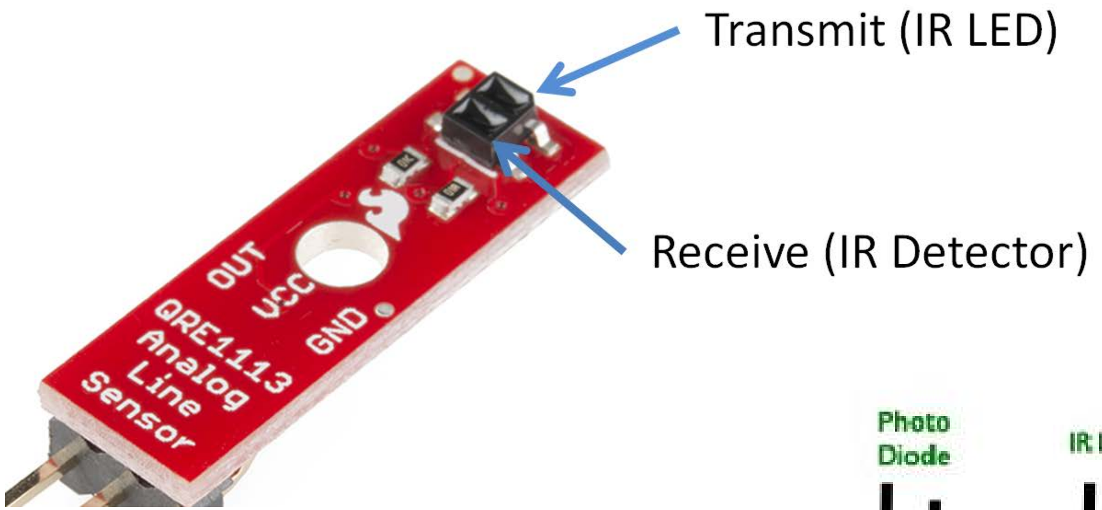
$$H(s) = \frac{y}{u} = \frac{A(s)}{1 + A(s)B(s)}$$

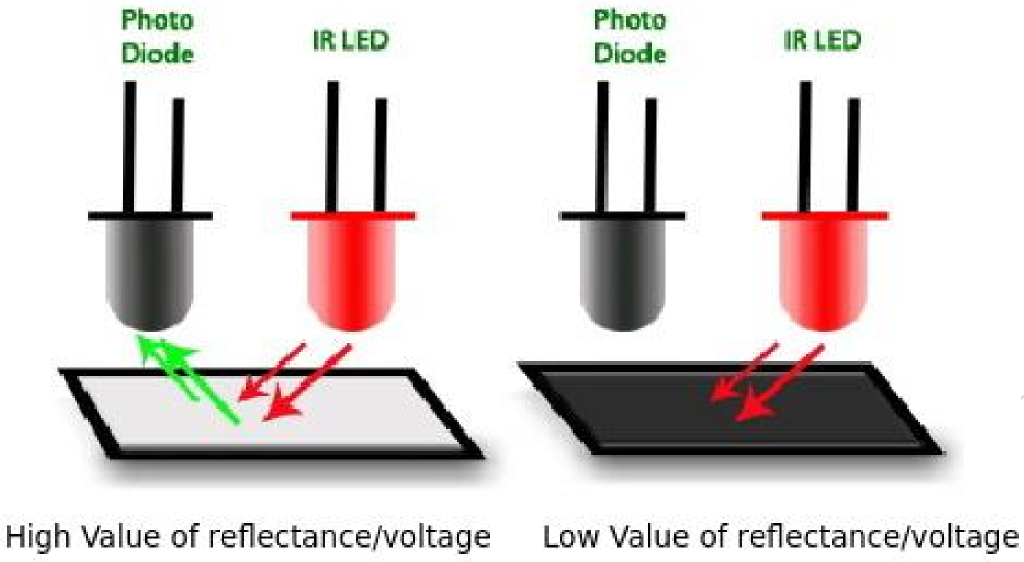
Line Following

https://giphy.com/gifs/arduino-RhdxqQ81tfURi

How Do Line Sensors Work?

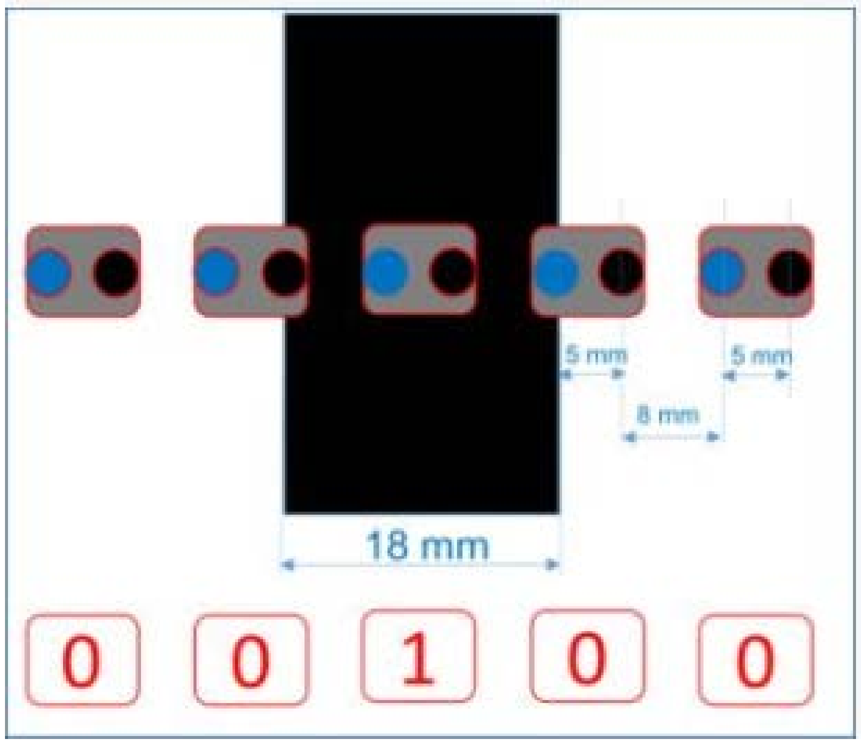
Line Sensor Hardware

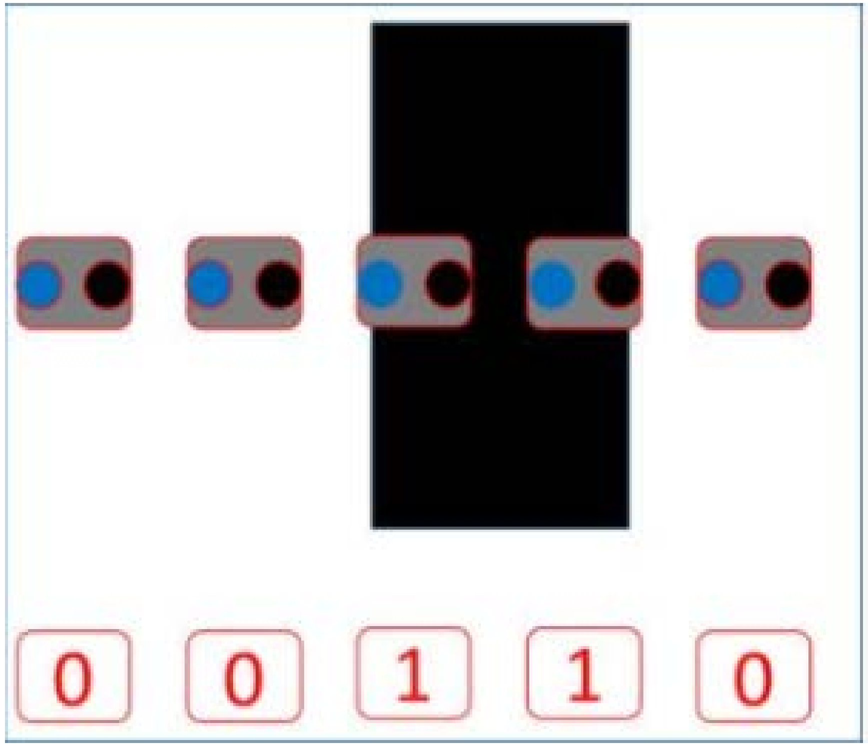




http://www.instructables.com/id/Line-Follower-Robot-PID-Control-Android-Setup/

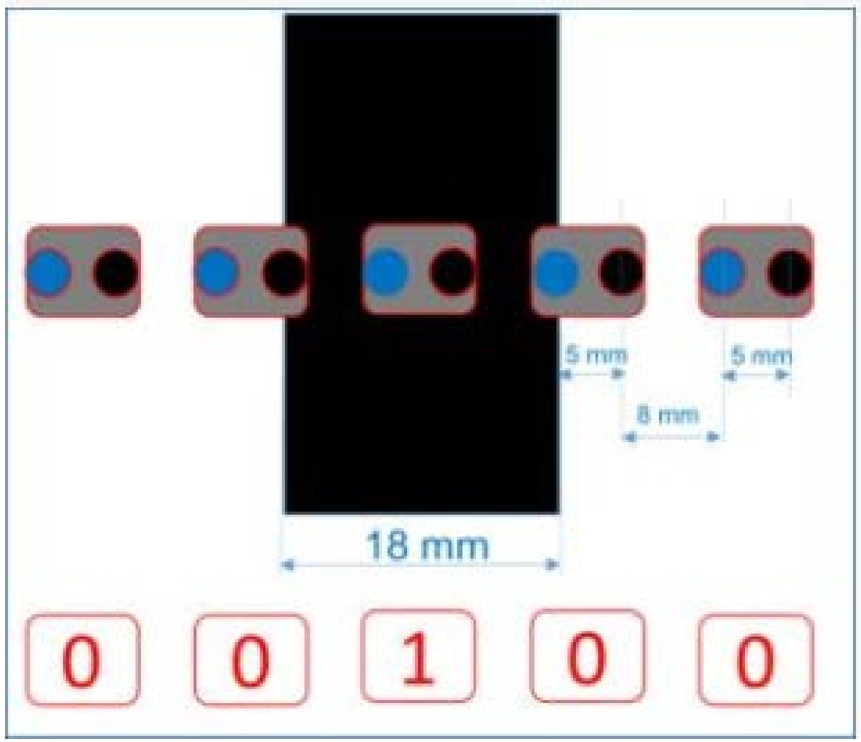
Digital Output

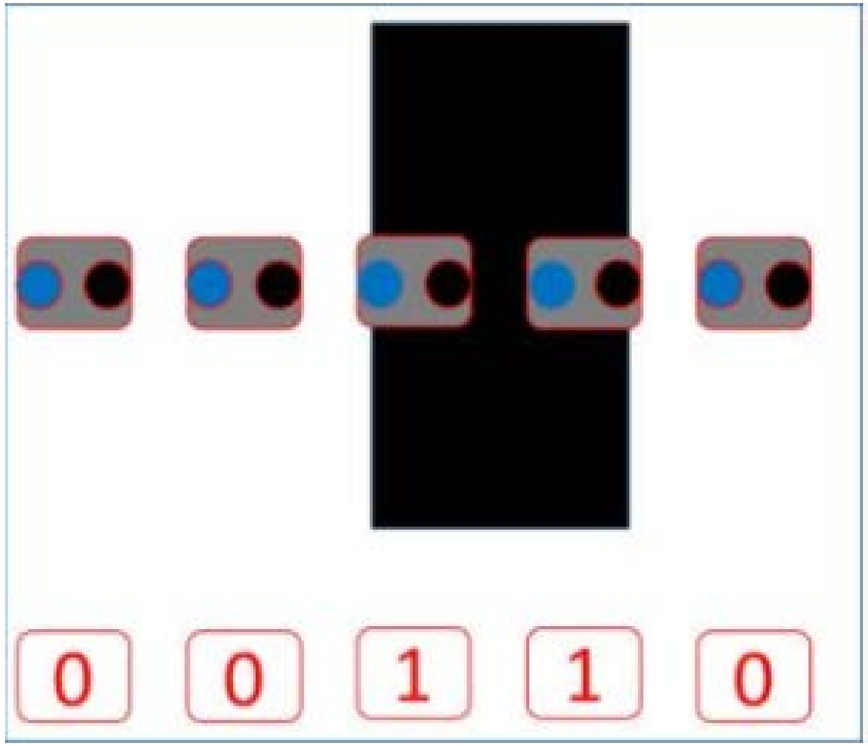




http://www.instructables.com/id/Line-Follower-Robot-PID-Control-Android-Setup/

Servo Correction



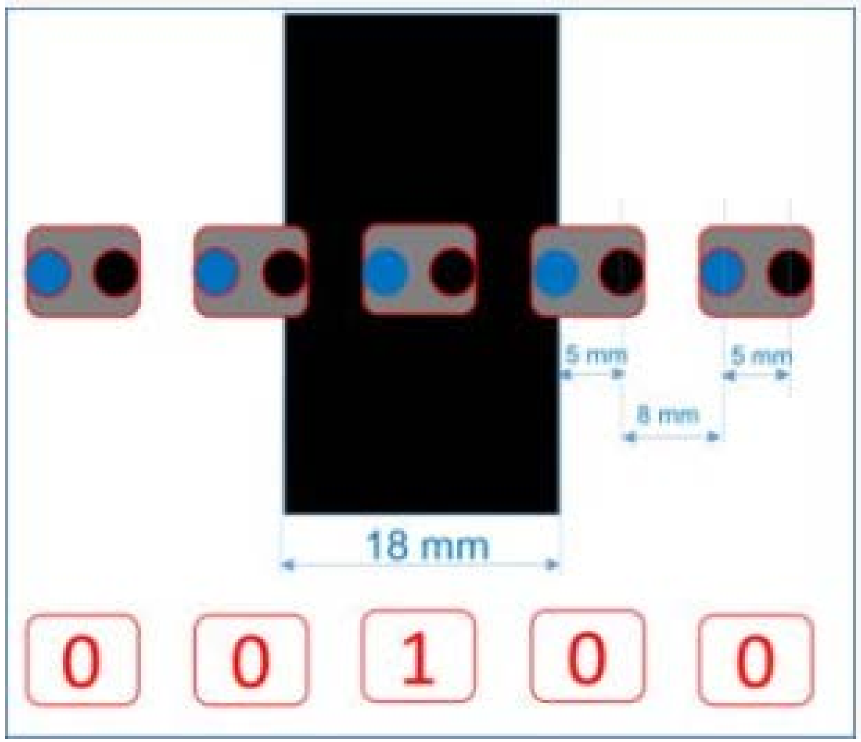


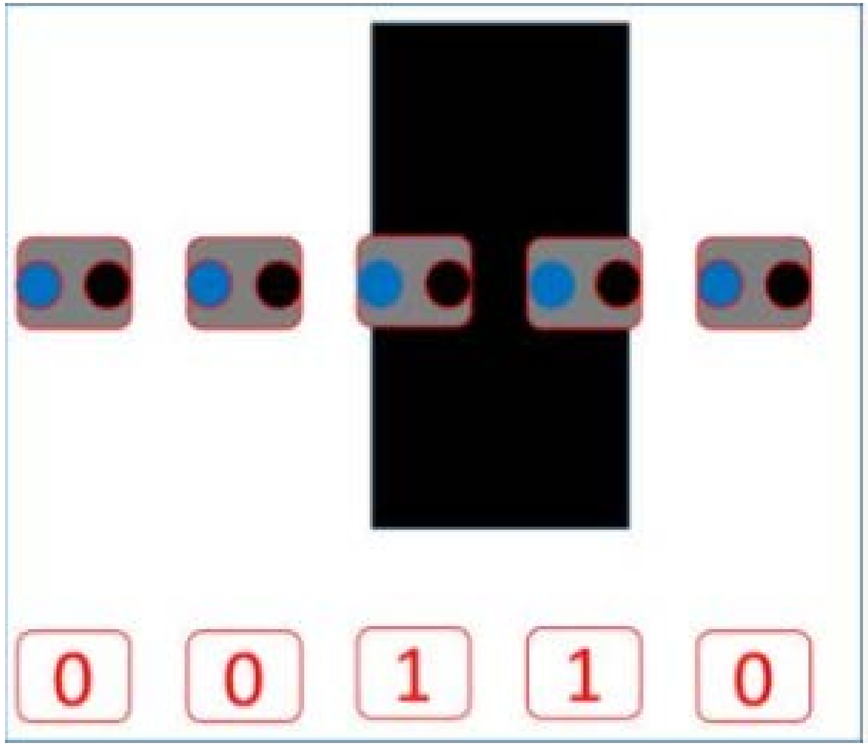
Left Servo Speed : 50
Right Servo Speed : 50

Left Servo Speed : ?
Right Servo Speed : ?

http://www.instructables.com/id/Line-Follower-Robot-PID-Control-Android-Setup/

Digital Output





Left Servo Speed : 50
Right Servo Speed : 50

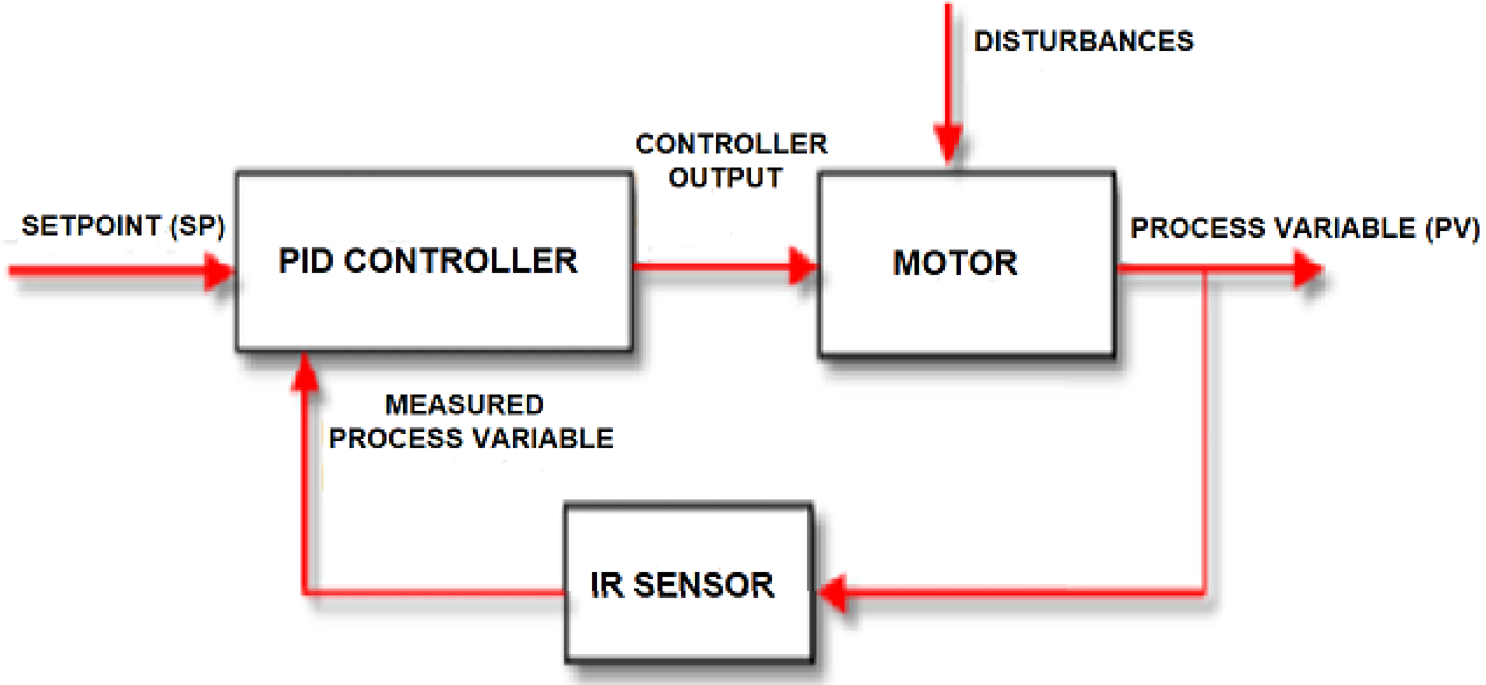
Left Servo Speed : 50 - error
Right Servo Speed : 50 + error

http://www.instructables.com/id/Line-Follower-Robot-PID-Control-Android-Setup/

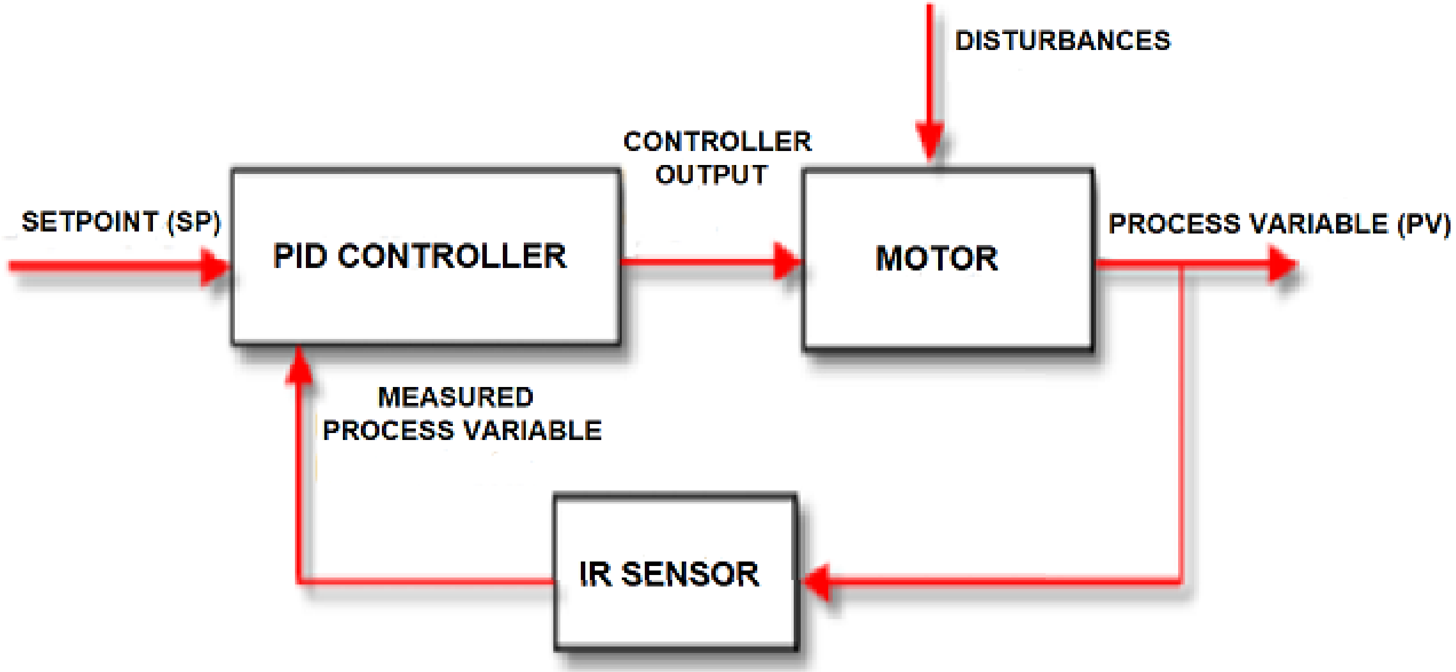
PID Algorithm

Definition

A robust closed-loop control algorithm particularly well-suited for rapid prototyping robotics



Block Diagram

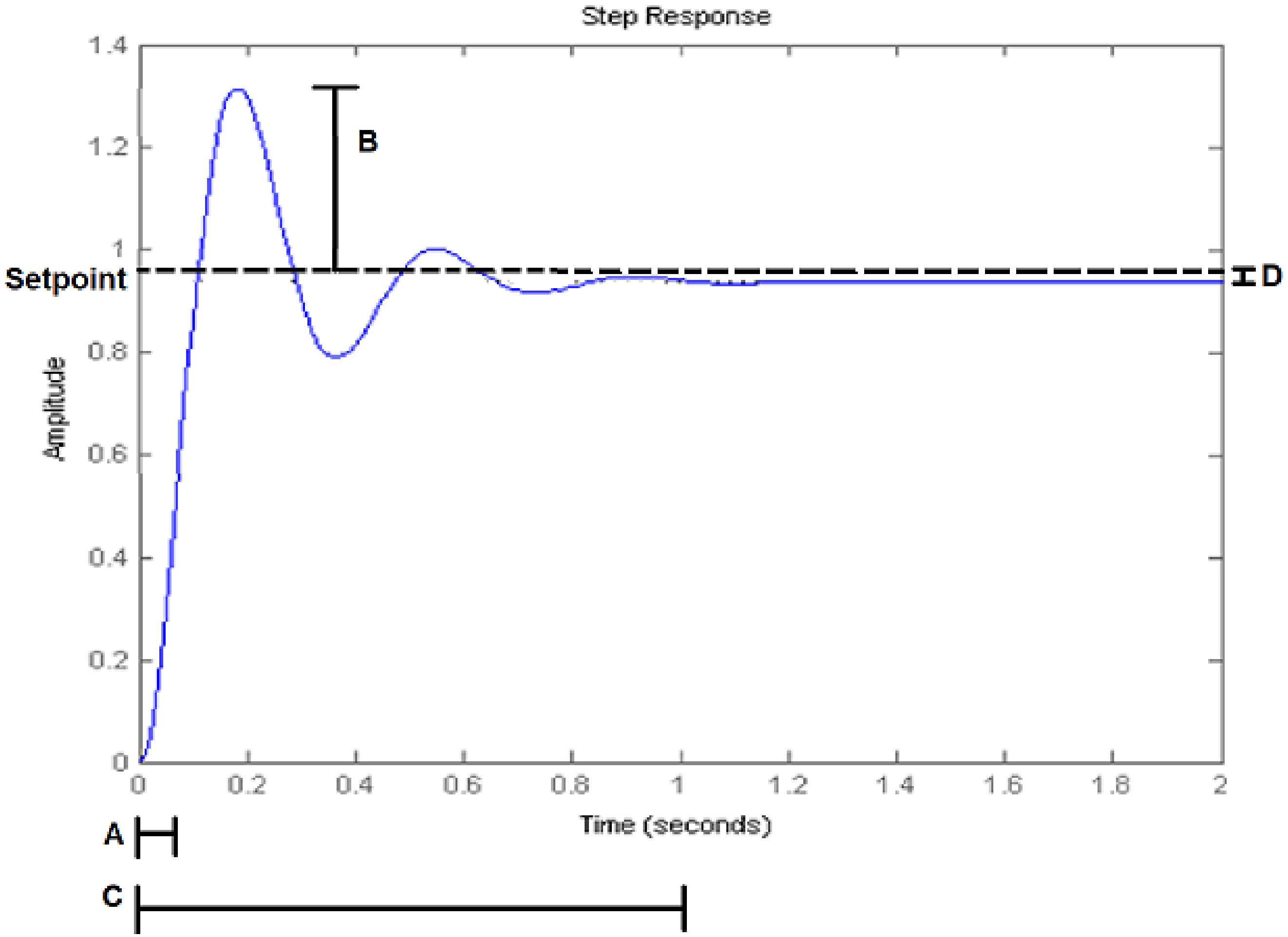


Set-point: Distance from line we want

Controller Output: Motor speed we want

Process Variable: Distance from line we get

Transfer Function



A - Rise Time

B - Percent Overshoot

C - Settling Time

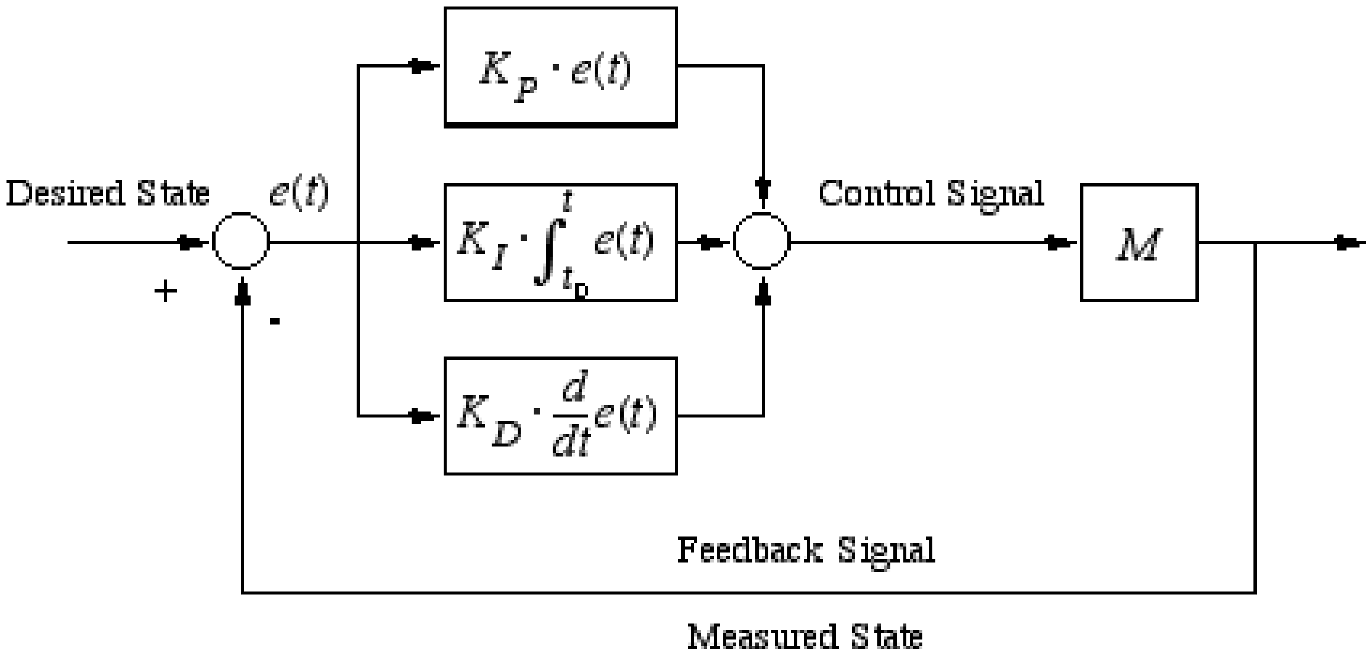
D - Steady State Error

https://en.wikipedia.org/wiki/PID_controller

PID Algorithm

- Proportional: Difference between set-point and measured process variable (error)
- Integral: Sum of errors over time
- Derivative: Rate of change of error over time

Fundamental Block Diagram



$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

https://qph.ec.quoracdn.net/main-qimg-f80e2346eca7905cdc90e38450483be6

Proportional Pseudocode

```
int error = 0; // Value given to servo if off line
int Kp = X; // Tuned coefficient for Proportional term

void ComputeError()
{
    if(leftSensorOnLine && rightSensorOffLine)
    {
        error = Y; // Turn to the left
    }
    .
    .
    .
    motorSpeed = Kp*error + originalspeed; // Sets new motor speed
}
```

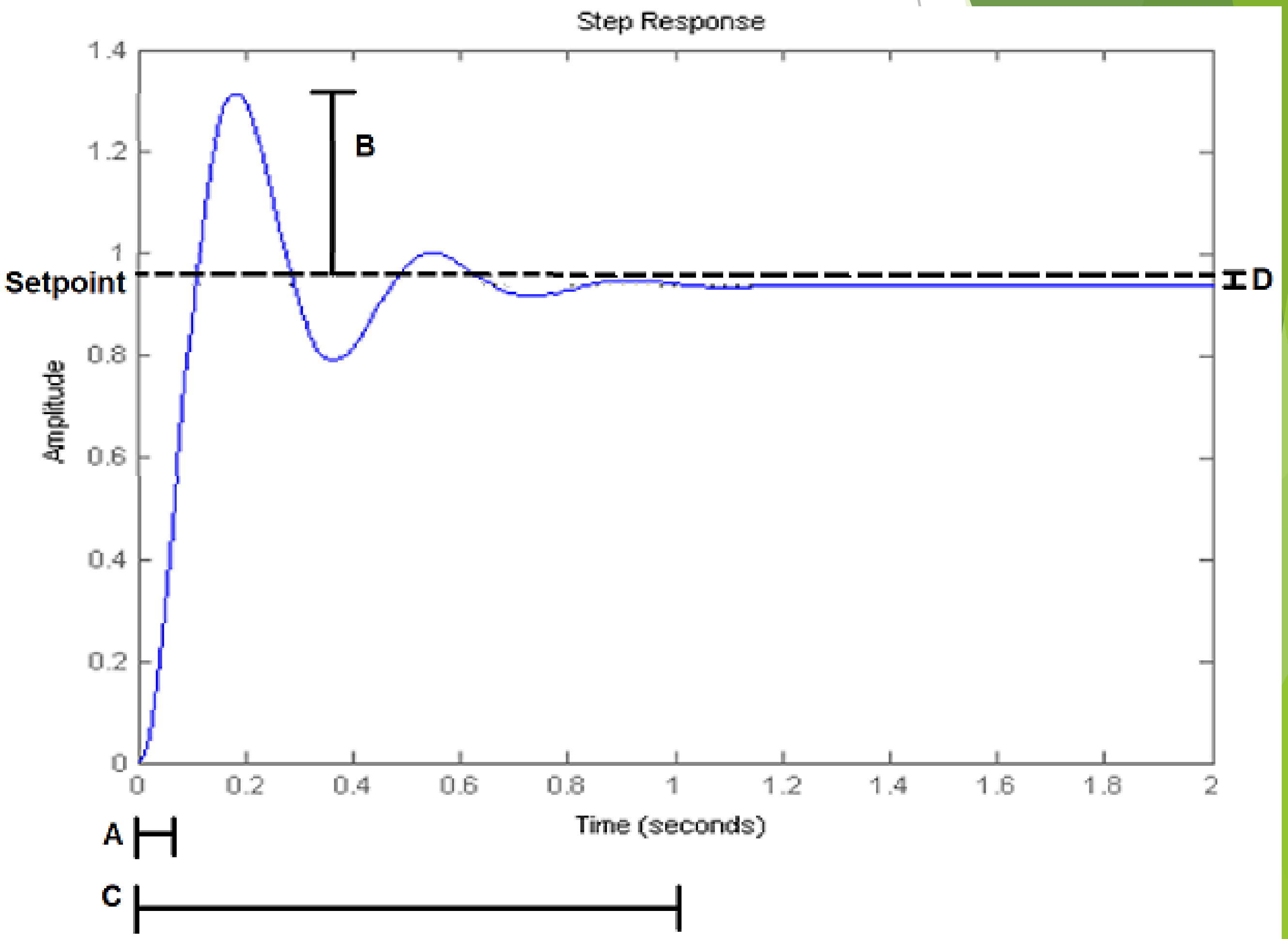
$$u(t) = K_p e(t)$$

Why isn’t Proportional Enough?

Why isn’t Proportional enough?

- High Steady State Error

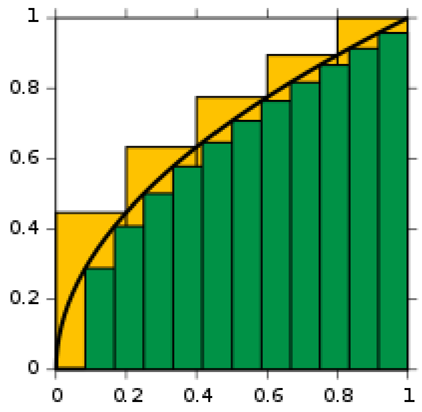
- Long Settling Time



https://en.wikipedia.org/wiki/PID_controller

PI Pseudocode

```
int error = 0; // Value given to servo if off line
int Kp = X; // Tuned coefficient for Proportional term

int Ki = X; // Tuned coefficient for Integral term
int errorSum = 0; // Sum of errors for Integral term
long currentTime; // Current time used for Integral Term
long lastTime; // Last time used for Integral Term

void ComputeError()
{
    currentTime = millis(); // returns current time
    timeChange = currentTime - lastTime; // returns change in time

    if(leftSensorOnLine && rightSensorOffLine)
    {
        error = Y; // Turn to the left
    }
    .
    .
    .
    errorSum = errorSum + error*timeChange;

    motorSpeed = Kp*error + Ki*errorSum + originalSpeed;

    lastTime = currentTime; // Sets time variable for next iteration
}
```

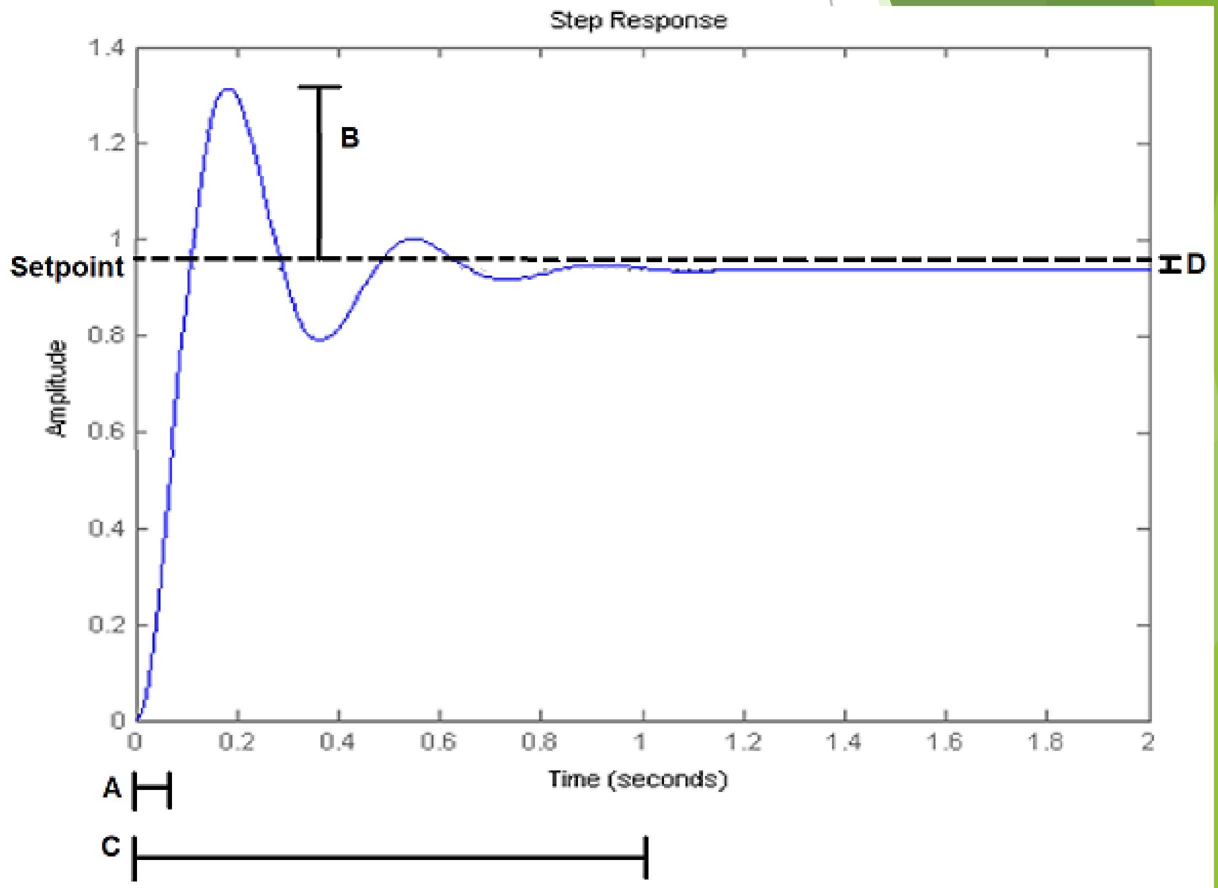


Why isn’t PI Enough?

Why isn’t PI enough?

- High Percentage Overshoot

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau$$

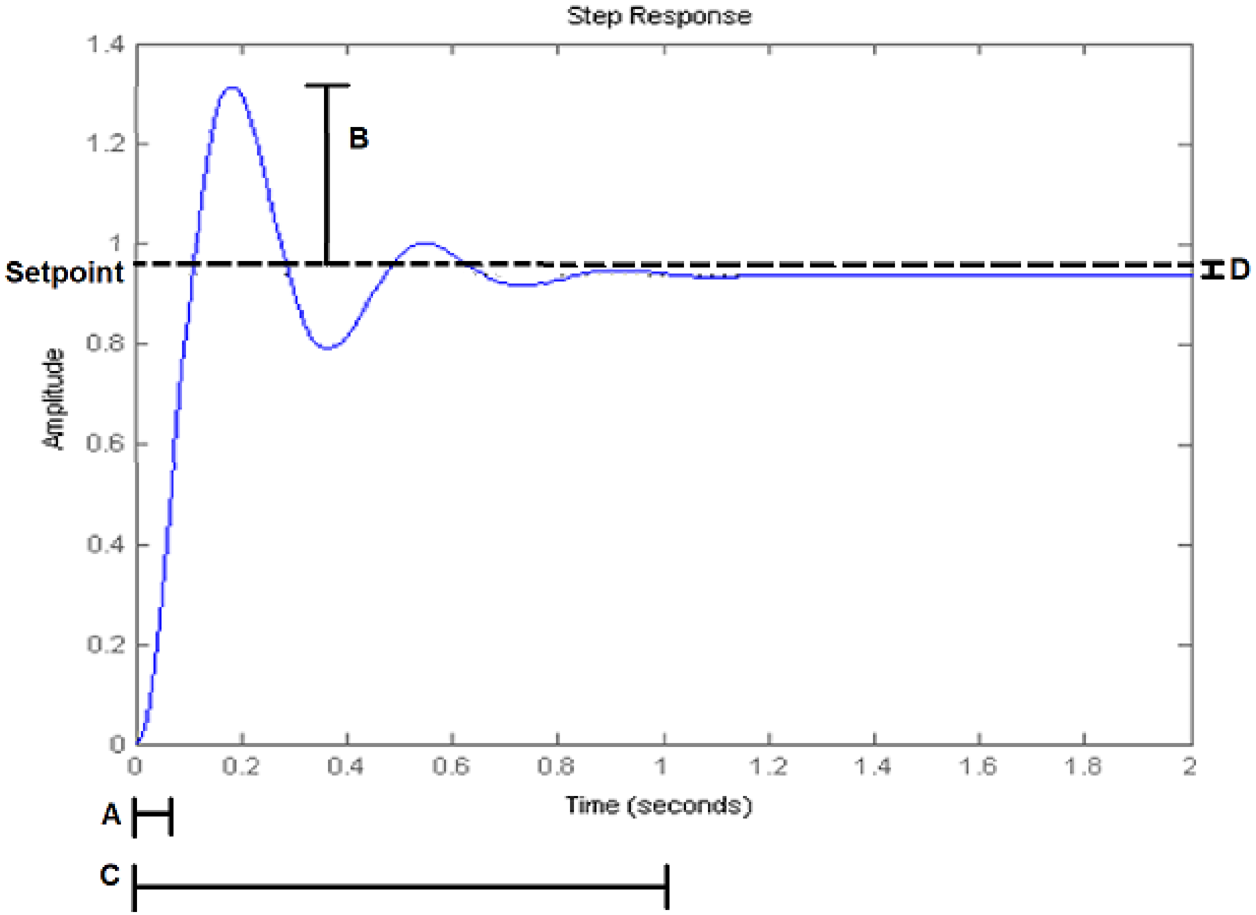


https://en.wikipedia.org/wiki/PID_controller

PID Pseudocode

```
int Kp = X; // Tuned coefficient for Proportional term
int error = 0; // Error value given to servo if off line

int Ki = X; // Tuned coefficient for Integral term
int errorSum = 0; // Sum of all errors
long currentTime; // Shows current time of error computation
long lastTime; // Shows time of last error computation

int Kd = X; // Tuned coefficient for Derivative term
int lastError; // Error from last iteration
int errorDiff = 0; // Rate of change of errors over time

void ComputeError()
{
    currentTime = millis(); // returns current time
    timeChange = currentTime - lastTime; // returns change in time

    if(leftSensorOnLine && rightSensorOffLine)
    {
        error = Y; // Turn to the left
    }
    .
    .
    .
    errorSum = errorSum + error*timeChange;
    errorDiff = (error - lastError)/timeChange;

    motorSpeed = Kp*error + Ki*errorSum + Kd*errorDiff + originalSpeed;

    lastTime = currentTime; // Sets time variable for next iteration
    lastError = error;
}
```

Effects on Transfer Function

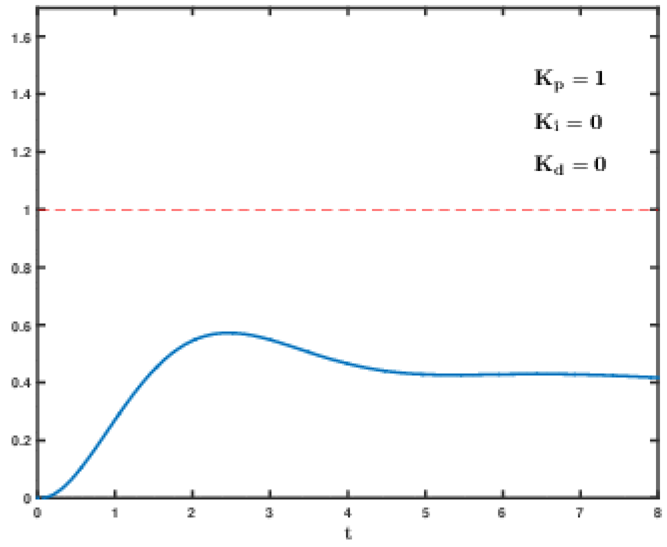


Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

https://en.wikipedia.org/wiki/PID_controller

Finding Coefficients

- Set all coefficients to zero
- Increase Kp until system oscillates
- Increase Ki until steady state error corrected
- Increase Kd until overshoot decreased

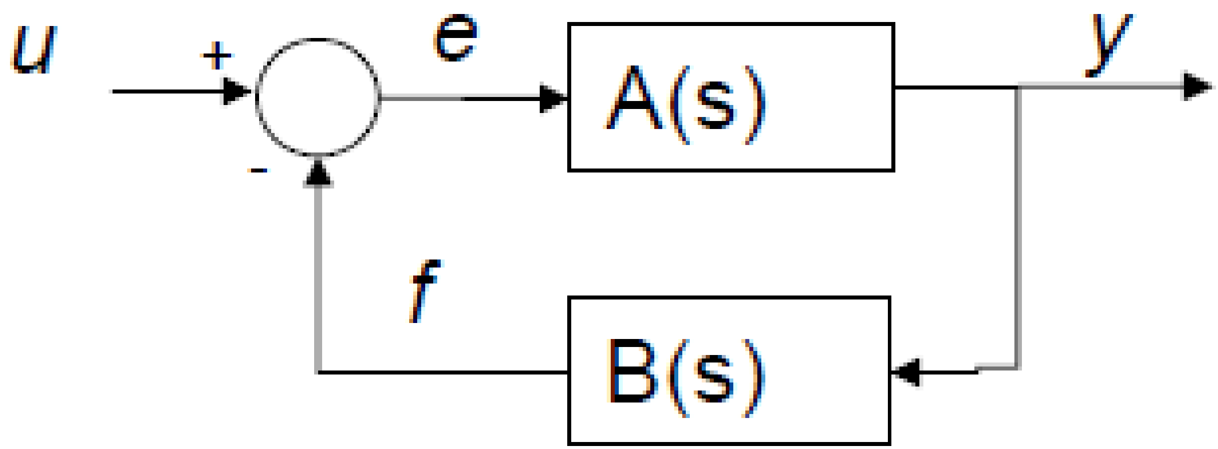


$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

https://en.wikipedia.org/wiki/PID_controller

Recap

Feedback Control



$$H(s) = \frac{y}{u} = \frac{A(s)}{1 + A(s)B(s)}$$

Analog Application

$$u = V_{in}$$
$$y = V_{out}$$
$$A(s) = A_v$$
$$B(s) = \frac{R_1}{R_1 + R_f}$$

$$\frac{V_{out}}{V_{in}} = \frac{A(s)}{1 + A(s)B(s)}$$

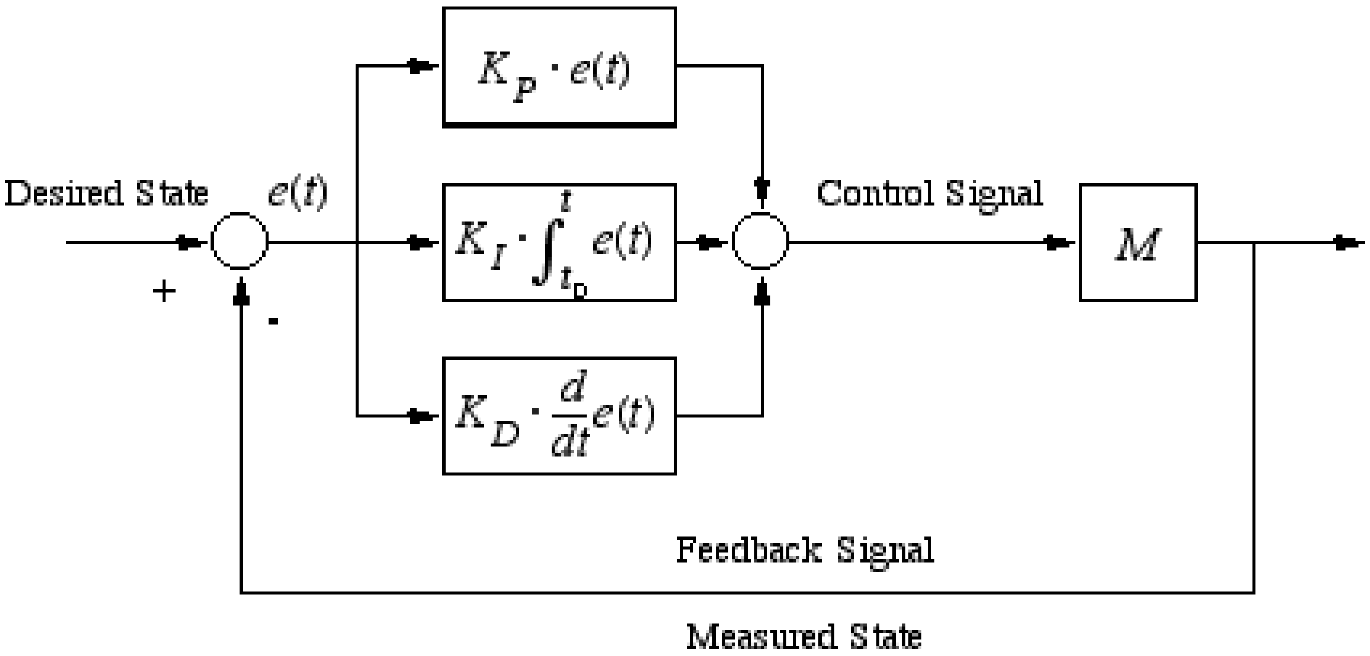
$$= \frac{A_v\,(R_1 + R_f)}{(R_1 + R_f + A_v R_1)}$$

➡

$$u = V_{in}$$
$$y = V_{out}$$
$$A(s) = A_v$$
$$B(s) = \frac{R_1}{R_1 + R_f}$$

$$H(s) = \frac{R_1 + R_f}{R_1}$$

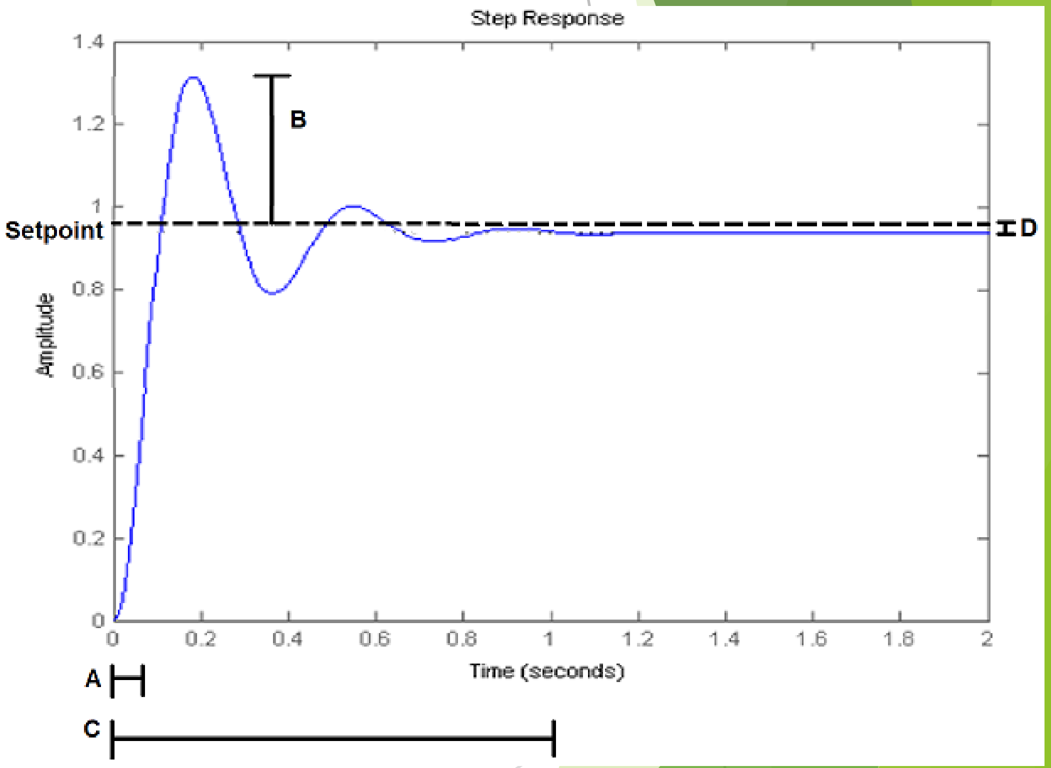
Software Application



$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

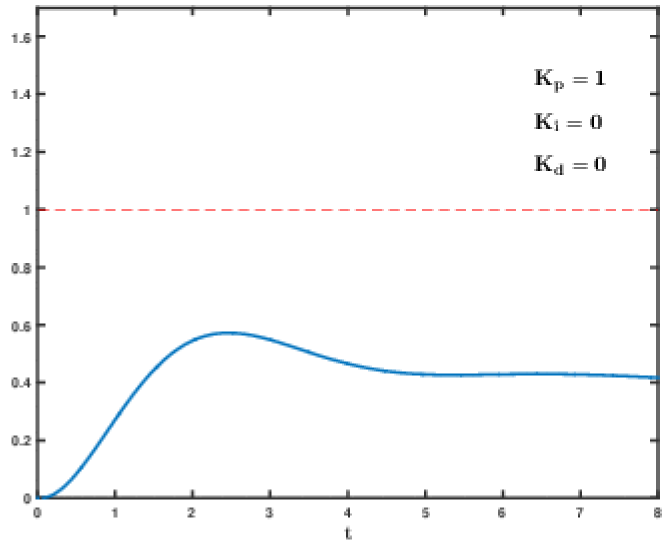
Implementation of PID

- Kp term adds large settling time and steady-state error
- Ki term adds large percentage overshoot
- Kd term adds dampening term, can cause oscillation



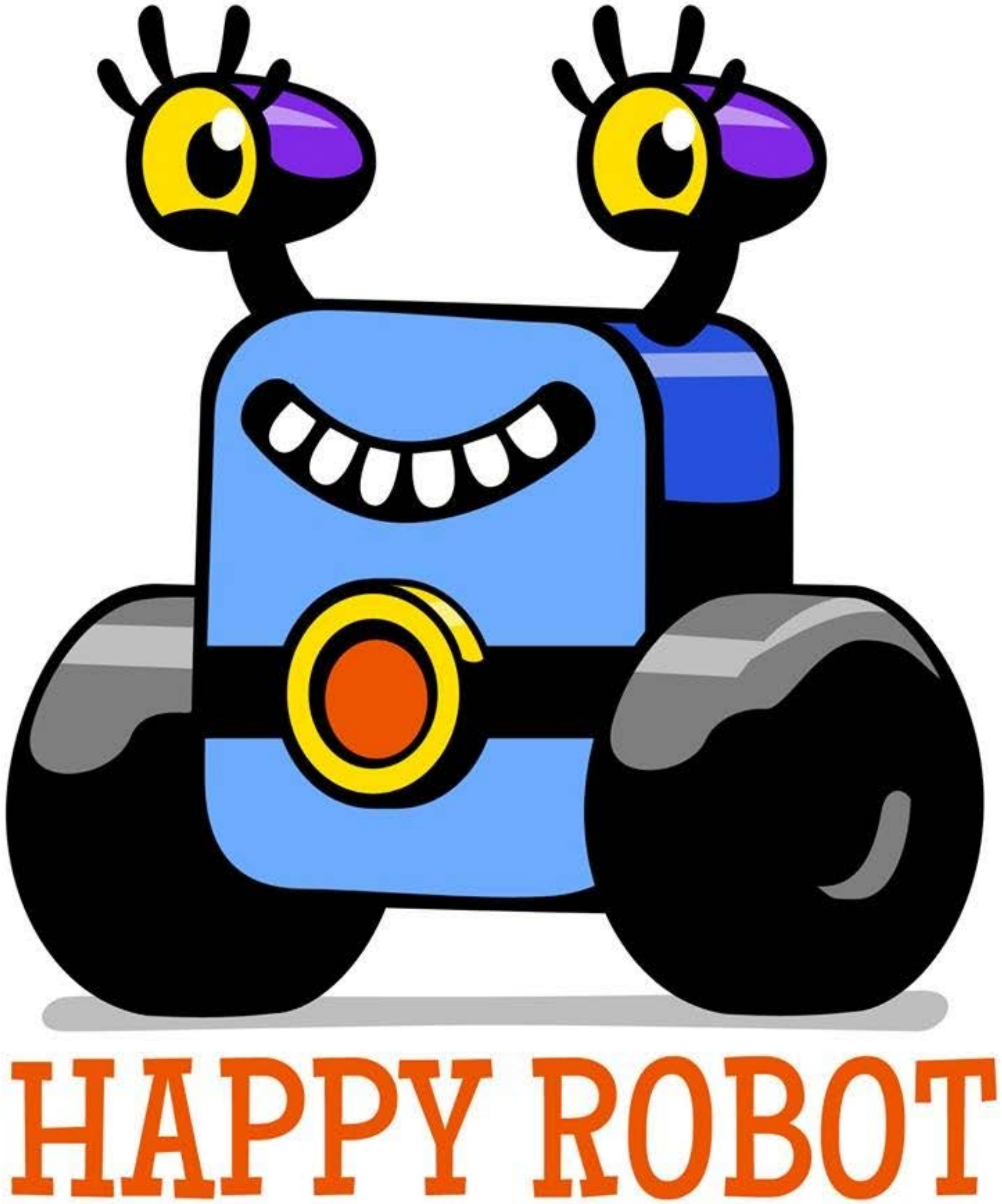
Finding Coefficients

- Set all coefficients to zero
- Increase Kp until system oscillates
- Increase Ki until steady state error corrected
- Increase Kd until overshoot decreased



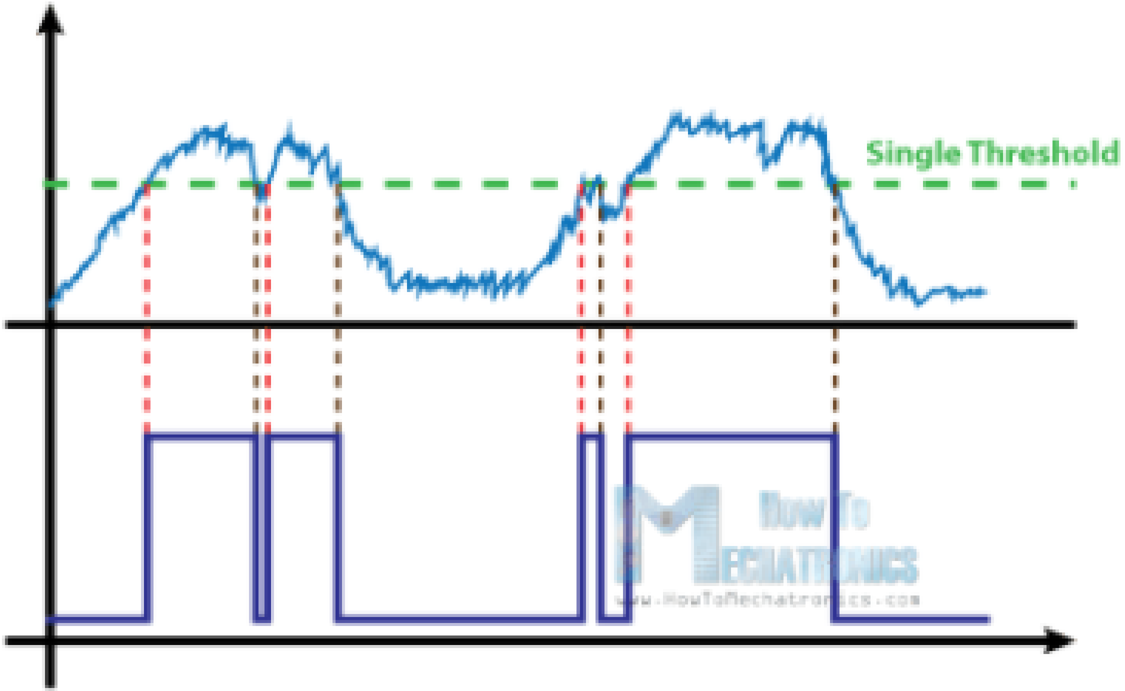
$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

https://en.wikipedia.org/wiki/PID_controller

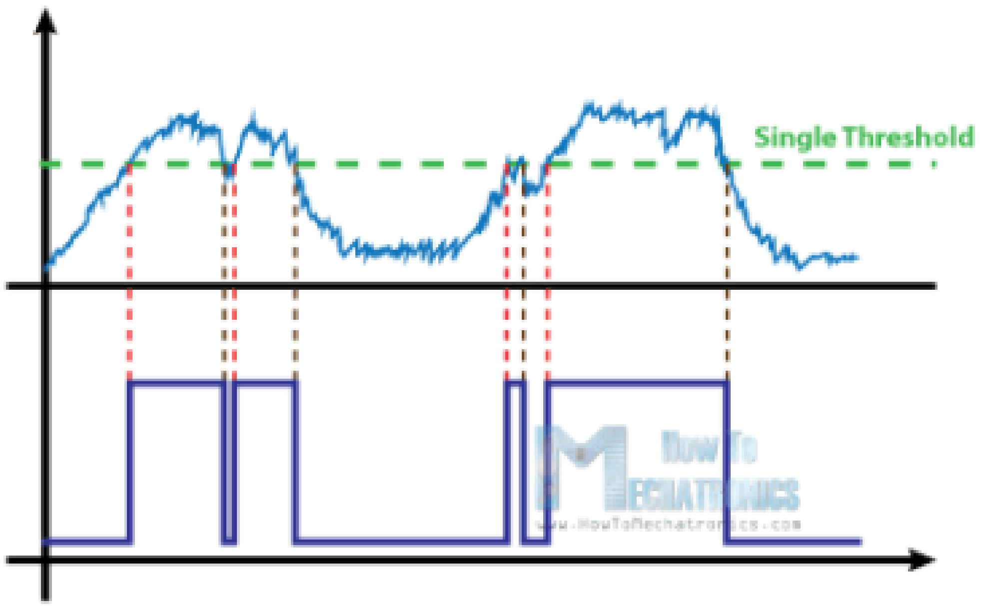

HAPPY ROBOT

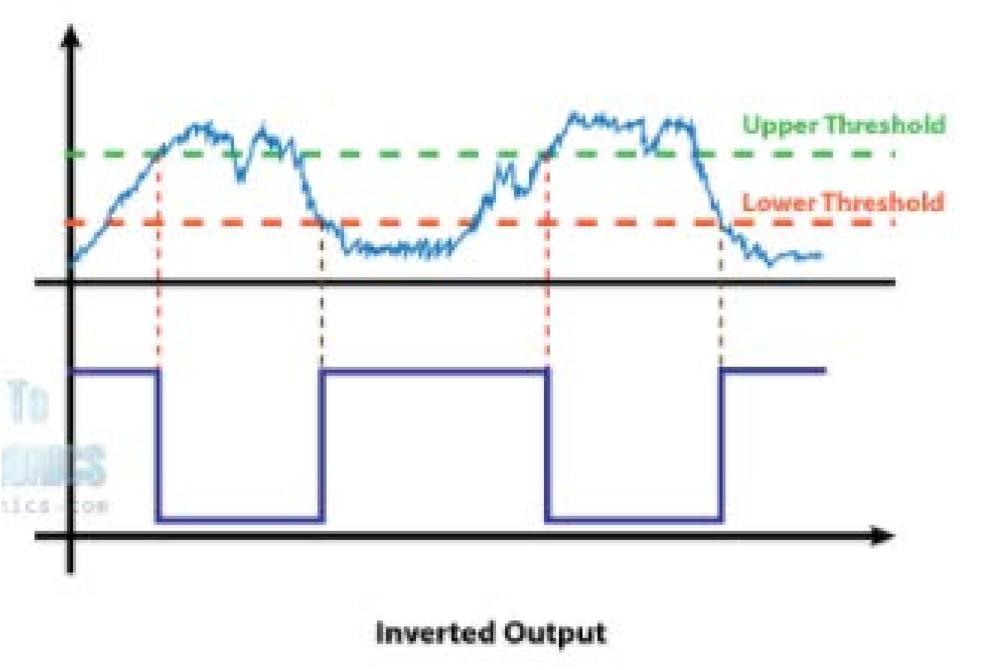
Extra Slides

Problem Statement

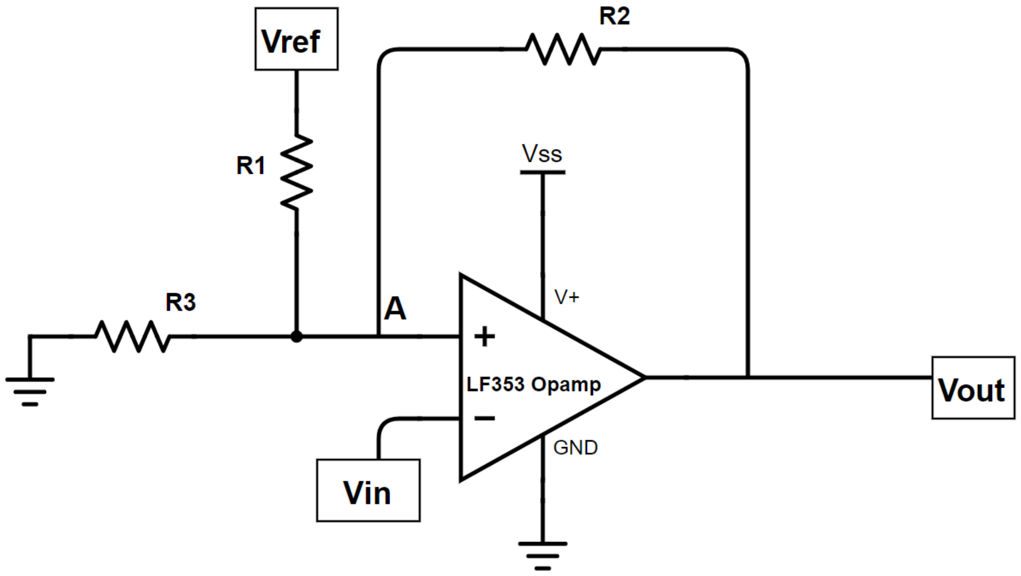


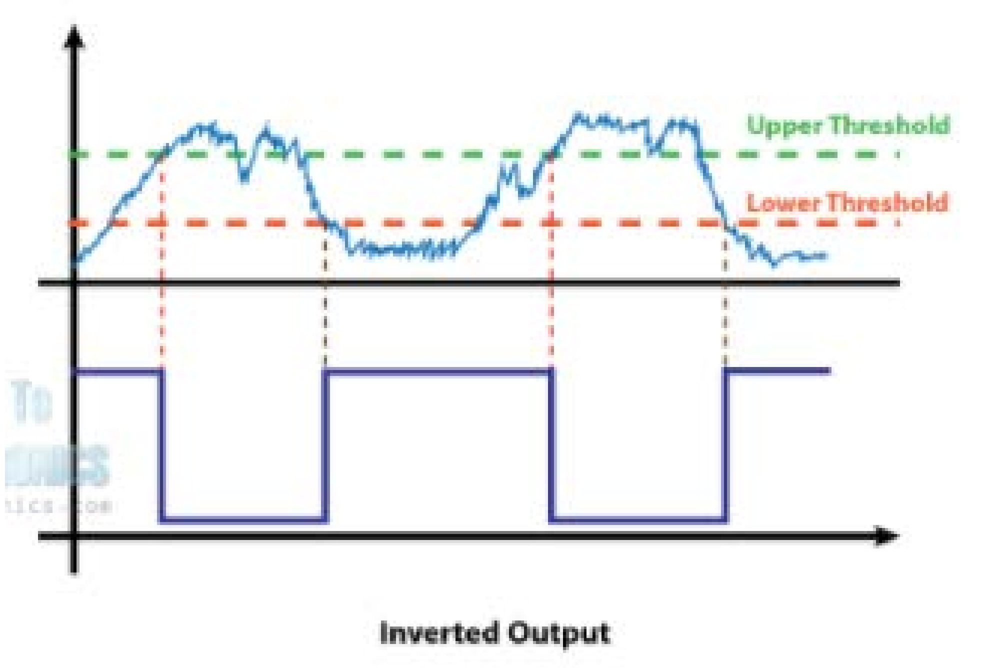
Jitter


Single Threshold


Upper Threshold
Lower Threshold
Inverted Output

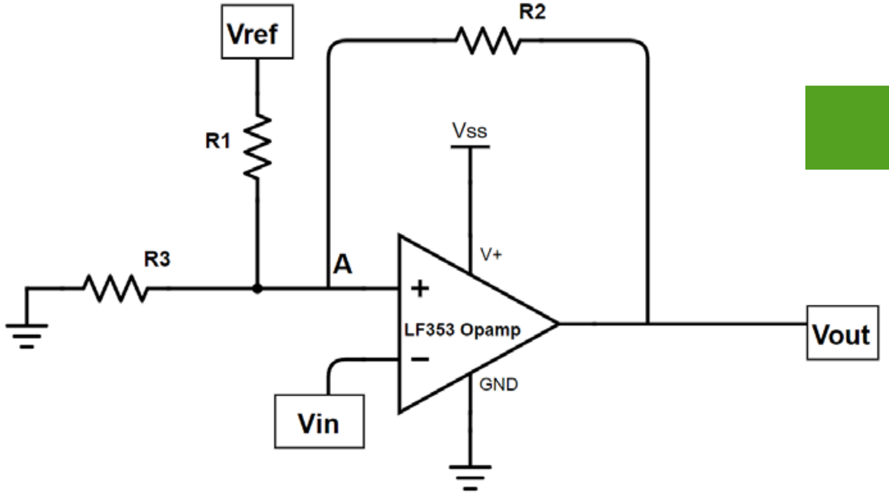
Schmitt Trigger

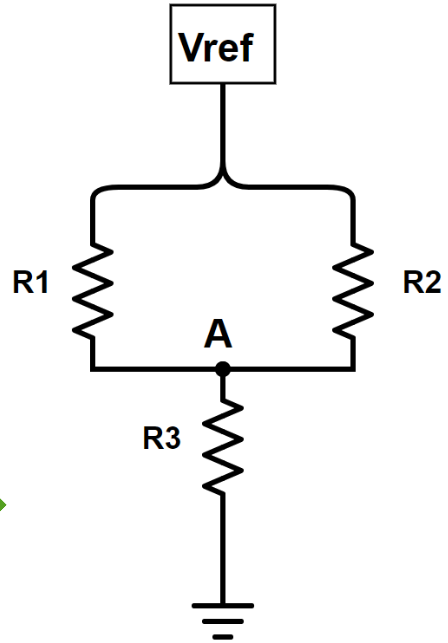




Vupper = Voltage that needs to be crossed to register a digital low
Vlower = Voltage that needs to be crossed to register a digital high

Operation

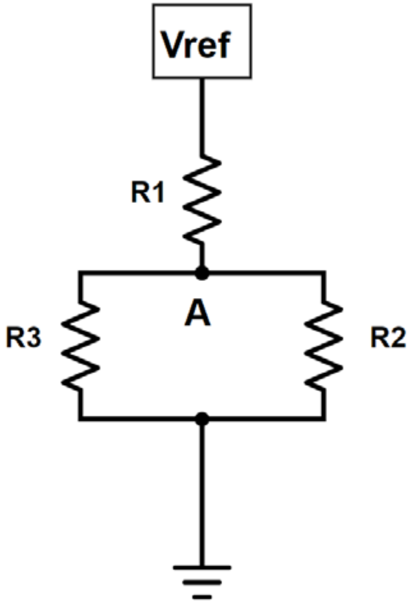




$$V_{out} = 5\,Volts$$

$$V_a = \frac{R_2}{R_2 + (R_1||R_3)}$$

$$V_a = Vupper$$

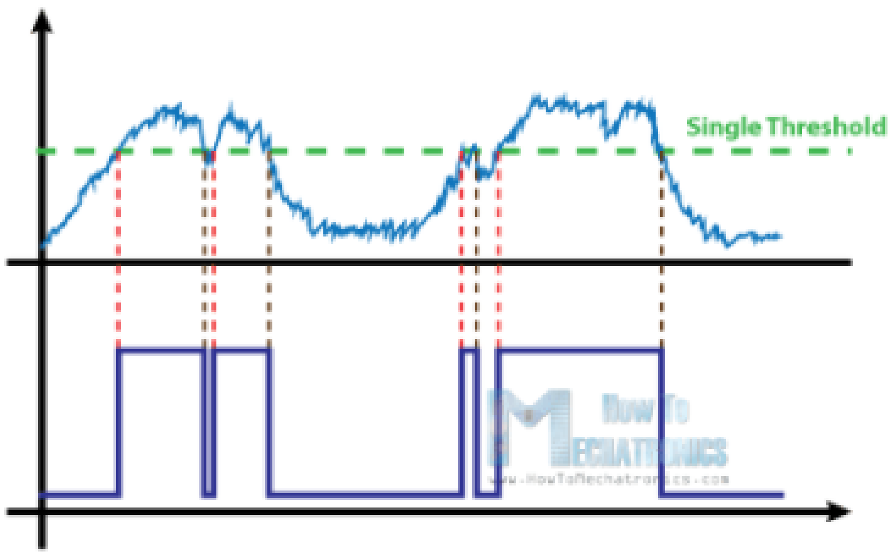


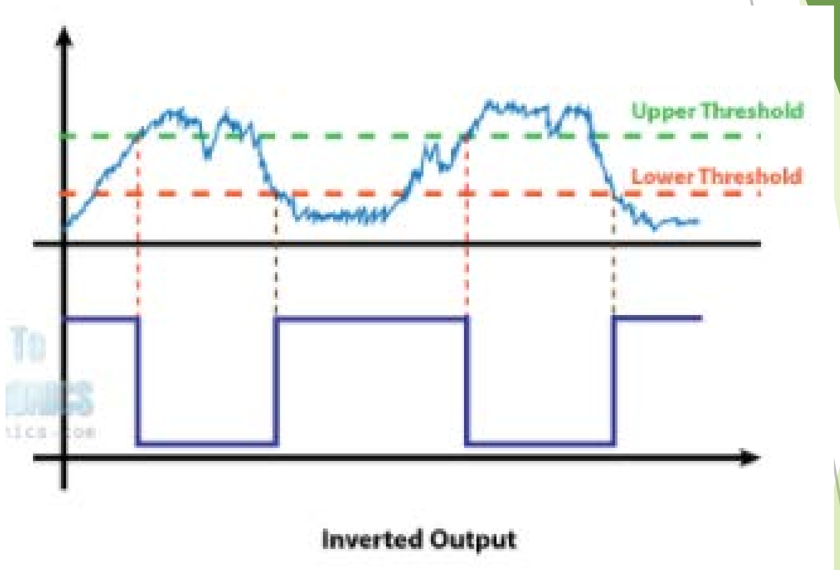
$$V_{out} = 0\,Volts$$

$$V_a = \frac{R_3||R_2}{R_1 + (R_3||R_2)}$$

$$V_a = Vlower$$

Example of Operation





If resistors all set to 10 KilaOhms

$$V_{upper} = \frac{R_2}{R_2+(R_1||R_3)} = 3.3 \text{ V}$$

$$V_{lower} = \frac{R_3||R_2}{R_1+(R_3||R_2)} = 1.66 \text{ V}$$

Copyright Slide

- Written and Presented by Adarsh Jayakumar

- Acknowledgements to Professors Kirstin Petersen and Alyosha Molnar for their guidance