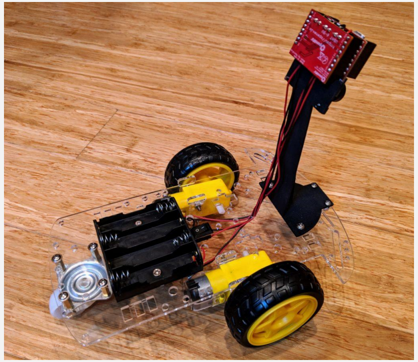


DIY Robocars
Day 2: The Embedded World

INTREPID
CONTROL SYSTEMS

The Integrated Development Environment (IDE)

- Compile - Before your program “code” can be sent to the board, it needs to be converted into instructions that the board understands. This process is called compiling.

- Stop - This stops the compilation process. (I have never used this button and you probably won’t have a need to either.)

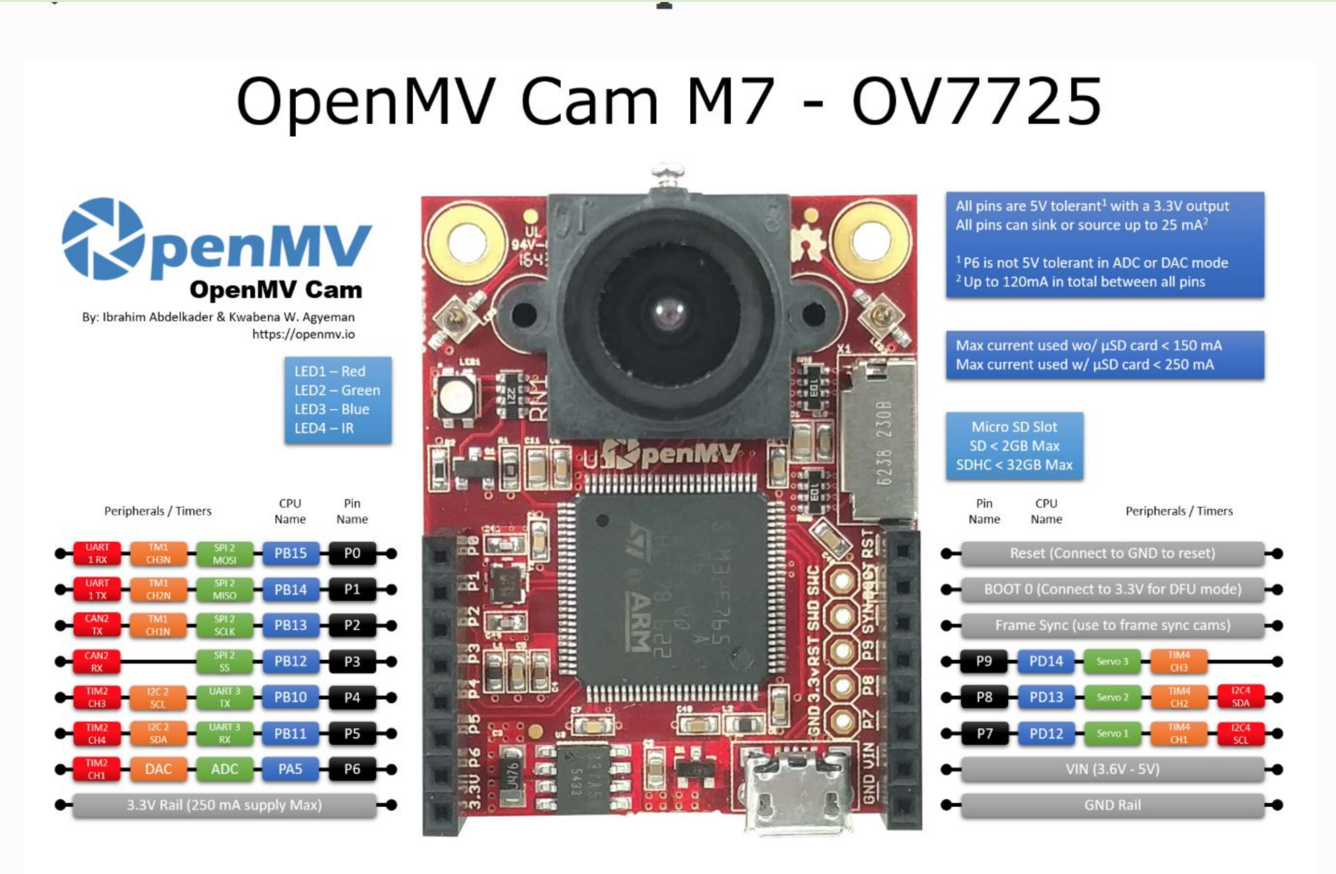
- Text Console - This shows you what the IDE is currently doing and is also where error messages display if you make a mistake in typing your program. (often called a syntax error)

- Line Number - This shows you what line number your cursor is on. It is useful since the compiler gives error messages with a line number

LED

- An RGB LED is really three small LEDs next to each other. A Red one, a Green one, and a Blue one. (Hence, why it is called RGB). It turns out that you can make any color by mixing these three light colors together.

General board control



Delay and timing

- It accepts a single integer as an argument. This number represents the time in milliseconds the program has to wait until moving on to the next line of code.

- time.sleep(1) your OpenMV stops on that line for 1 second.

- Blocking functions prevent a program from doing anything else until that particular task has completed. If you need multiple tasks to occur at the same time, you simply cannot use delay().

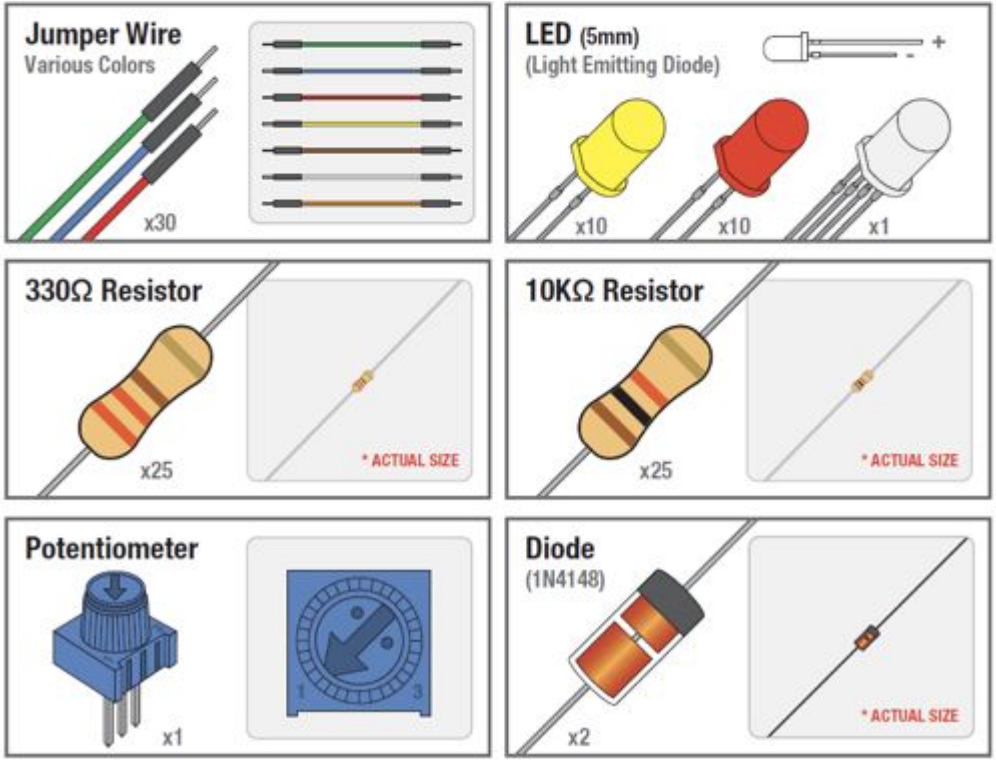
Why

- Because by using some math, you can easily verify how much time has passed without blocking your code.

SIK Components

Name	Image	Type	Function	Notes
Push Button		Digital Input	Switch - Closes or opens circuit	Polarized, needs resistor
Trim potentiometer		Analog Input	Variable resistor	Also called a Trimpot.
Photoresistor		Analog Input	Light Dependent Resistor (LDR)	Resistance varies with light.
Relay		Digital Output	Switch driven by a small signal	Used to control larger voltages
Temp Sensor		Analog Input	Temp Dependent Resistor	
Flex Sensor		Analog Input	Variable resistor	
Soft Trimpot		Analog Input	Variable resistor	Careful of shorts
RGB LED		Dig & Analog	16,777,216	Ooh... So pretty.

Components



Jumper Wire
Various Colors
x30
LED (5mm)
(Light Emitting Diode)
+
-
x10
x10
x1
330Ω Resistor
x25
* ACTUAL SIZE
10KΩ Resistor
x25
* ACTUAL SIZE
Potentiometer
x1
1
3
Diode
(1N4148)
x2
* ACTUAL SIZE

Components

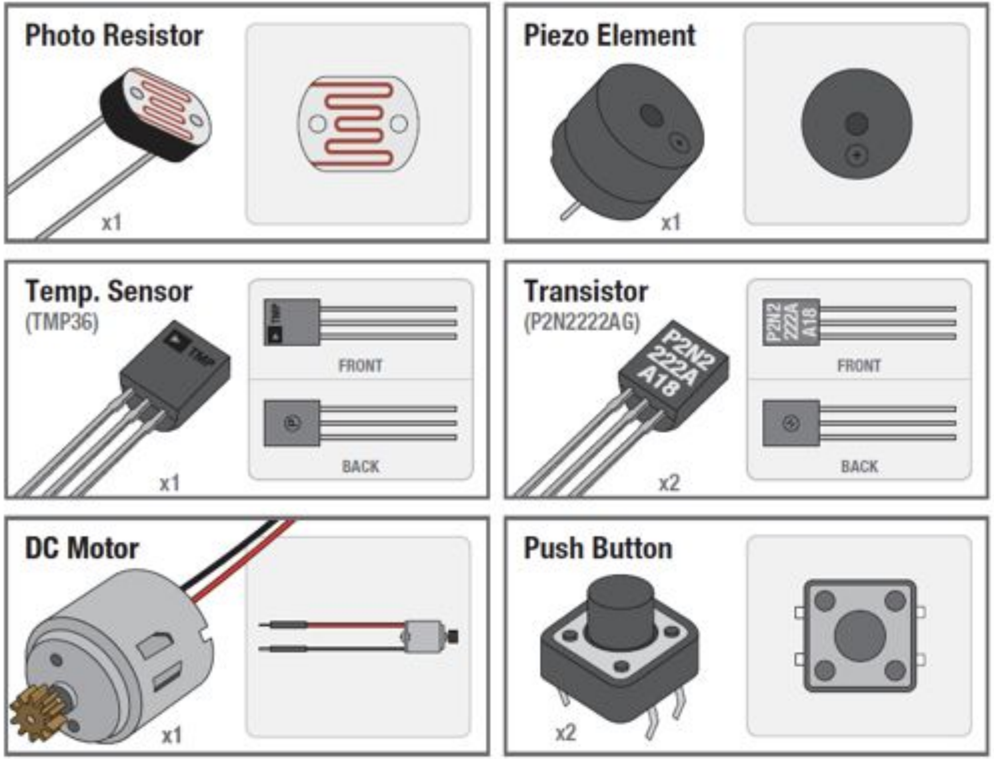


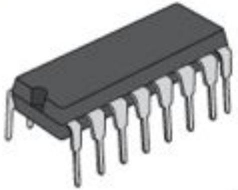
Photo Resistor
x1
Piezo Element
x1
Temp. Sensor
(TMP36)
x1
FRONT
BACK
Transistor
(P2N2222AG)
P2N2
222A
A18
x2
FRONT
BACK
DC Motor
x1
Push Button
x2

Relay

x1

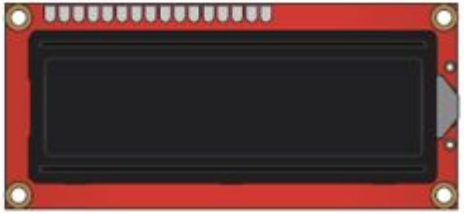
Integrated Circuit

(IC)



x1

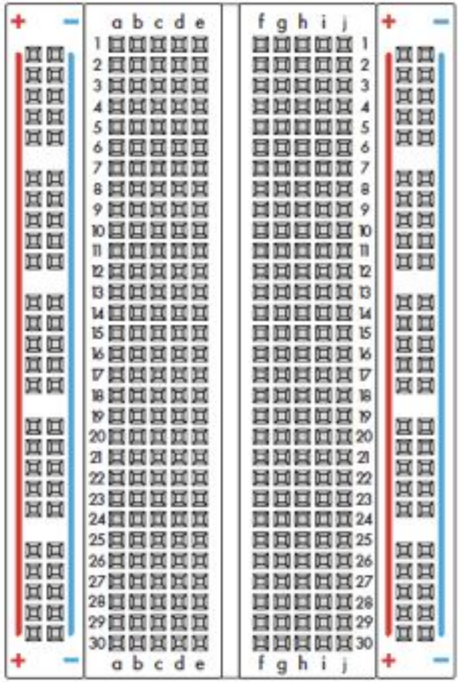
LCD



x1

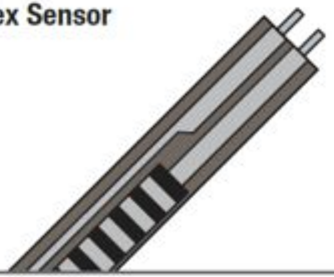
Breadboard

Standard Solderless (Color may vary)



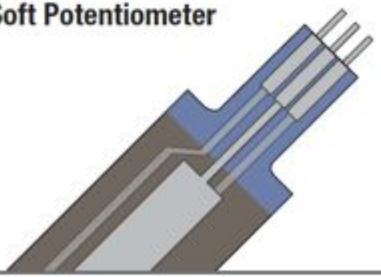
x1

Flex Sensor



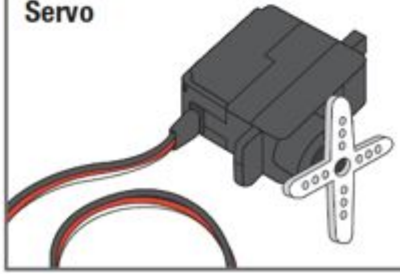
x1

Soft Potentiometer



x1

Servo



x1

Electricity \ Electronics Basic Concept Review

- Ohms Law
- Voltage
- Current
- Resistance
- Using a Multi-meter

Ohm’s Law

- Ohms Law
- Voltage
- Current
- Resistance
- Using a Multi-meter

Ohm's Law describes the direct relationship between the Voltage (V), Current (I), and Resistance (R) of a circuit.

The three different forms of Ohm's Law are as follows:

$$\boldsymbol{V = I \cdot R} \quad \boldsymbol{I = \frac{V}{R}} \quad \boldsymbol{R = \frac{V}{I}}$$

Voltage V	Current I	Resistance R
• Defined as the amount of potential energy in a circuit. • <u>Units</u>: Volts (V)	• The rate of charge flow in a circuit. • <u>Units</u>: Amperes (A)	• Opposition to charge flow. • <u>Units</u>: Ohms (Ω)

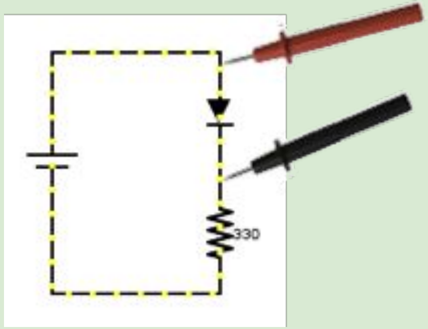
Continuity – Is it a Circuit?

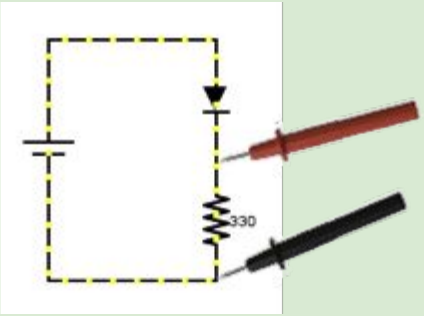
- The word “circuit” is derived from the circle. An Electrical Circuit must have a continuous LOOP from Power (Vcc) to Ground (GND).

- Continuity is important to make portions of circuits are connect. Continuity is the simplest and possibly the most important setting on your multi-meter. Sometimes we call this “ringing out” a circuit.

Measuring Electricity – Voltage

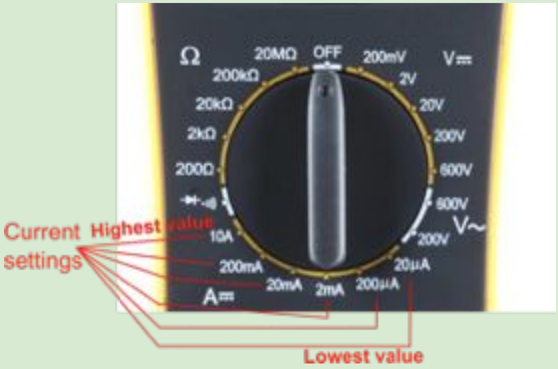
- Voltage is a measure of potential electrical energy. A voltage is also called a potential difference – it is measured between two points in a circuit – across a device.

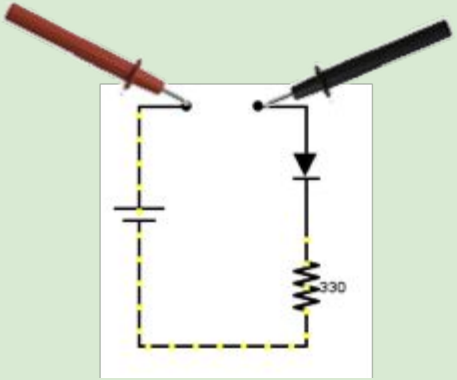




Measuring Electricity -- Current

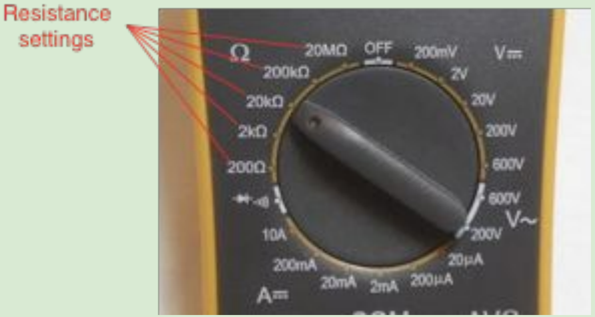
- Current is the measure of the rate of charge flow. For Electrical Engineers – we consider this to be the movement of electrons.
- In order to measure this – you must break the circuit or insert the meter in-line (series).





Measuring Electricity -- Resistance

- Resistance is the measure of how much opposition to current flow is in a circuit.
- Components should be removed entirely from the circuit to measure resistance. Note the settings on the multi-meter. Make sure that you are set for the appropriate range.



Concepts: INPUT vs. OUTPUT

- Inputs is a signal / information going into the board.

- Output is any signal exiting the board.

- Almost all systems that use physical computing will have some form of output

- What are some examples of Outputs?

Concepts: INPUT vs. OUTPUT

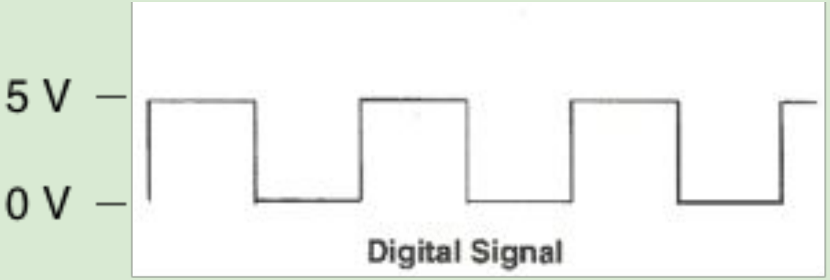
- Inputs is a signal / information going into the board.
  - Examples: Buttons Switches, Light Sensors, Flex Sensors, Humidity Sensors, Temperature Sensors…

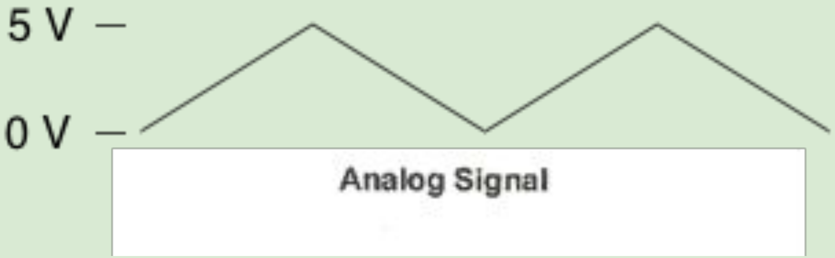
- Output is any signal exiting the board.
  - Examples: LEDs, DC motor, servo motor, a piezo buzzer, relay, an RGB LED

Concepts: Analog vs. Digital

- Microcontrollers are digital devices – ON or OFF. Also called – discrete.

- **Analog** signals are anything that can be a full range of values. What are some examples? More on this later…





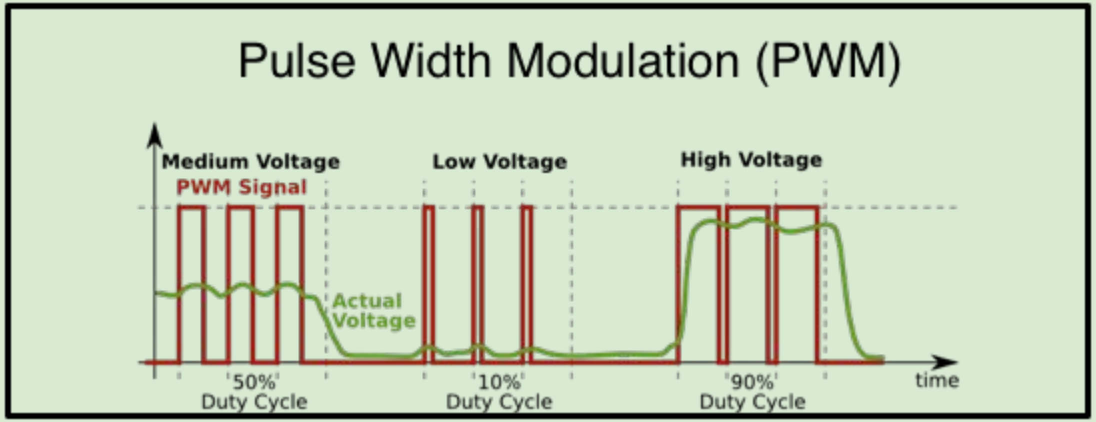
Analog

- Analog signals are made up of continuously streaming data

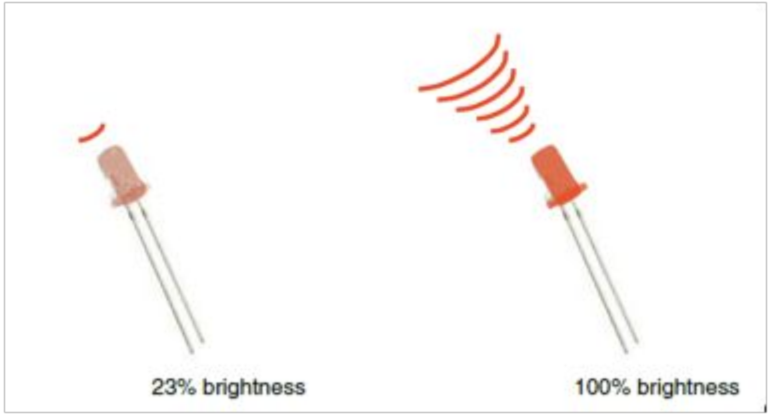
- Analog clocks provide continuous data (as shown by the constant movement of the second and minute hands) just like analog signals provide continuous data.

Concepts: Analog vs. Digital

To create an analog signal, the microcontroller uses a technique called PWM. By varying the duty cycle, we can mimic an “average” analog voltage.



- Can a digital devise produce analog output?



- Analog output can be simulated using pulse width modulation (PWM)

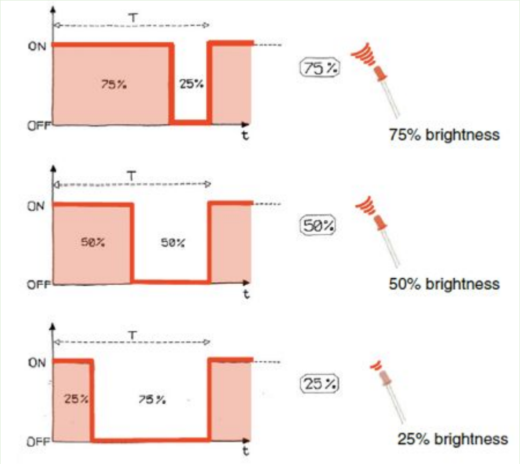
Concepts: Analog vs. Digital

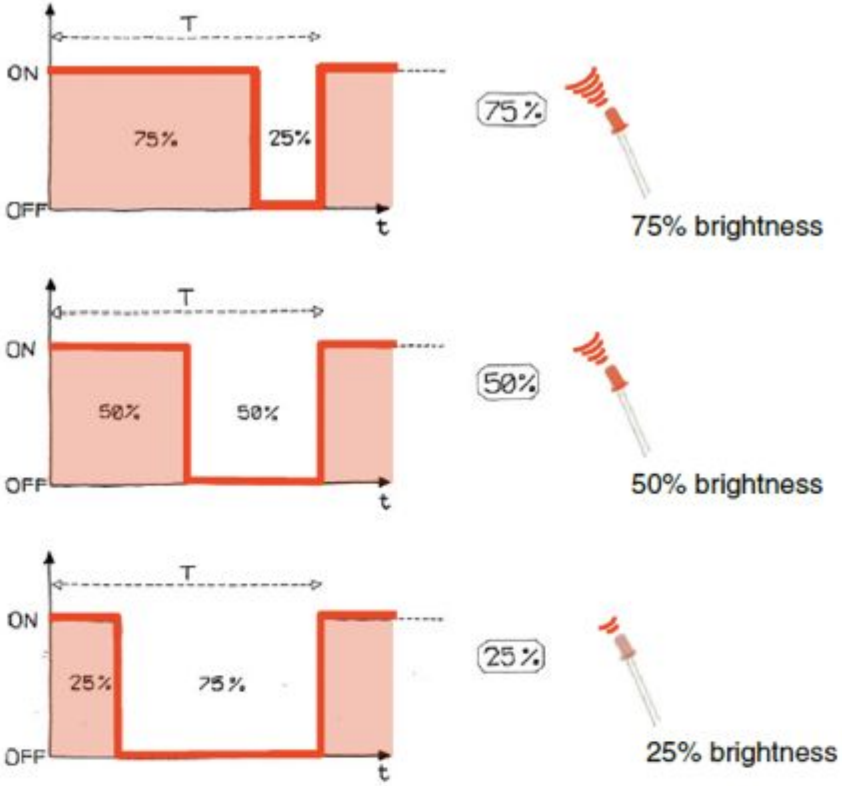
- Digital sensors are more straight forward than Analog
- No matter what the sensor there are only two settings: On and Off
- Signal is always either HIGH (On) or LOW (Off)

Pulse Width Modulation

- Can't use digital pins to directly supply say 2.5V, but can pulse the output on and off really fast to produce the same effect

- The on-off pulsing happens so quickly, the connected output device "sees" the result as a reduction in the voltage




T
ON
OFF
t
75%
25%
75%
75% brightness
T
ON
OFF
t
50%
50%
50%
50% brightness
T
ON
OFF
t
25%
75%
25%
25% brightness

Autonomous

For the last few years each game has started out with an autonomous period, in which the robots move based only on pre-programmed commands and sensor input.

Timer Based

- Robot follows pre-programmed commands that each last for a given length of time.

- Time can be judged by either using the processors timers or by counting the number of cycles that have passed.

Common Sensors

- Range Finders

Common Sensors

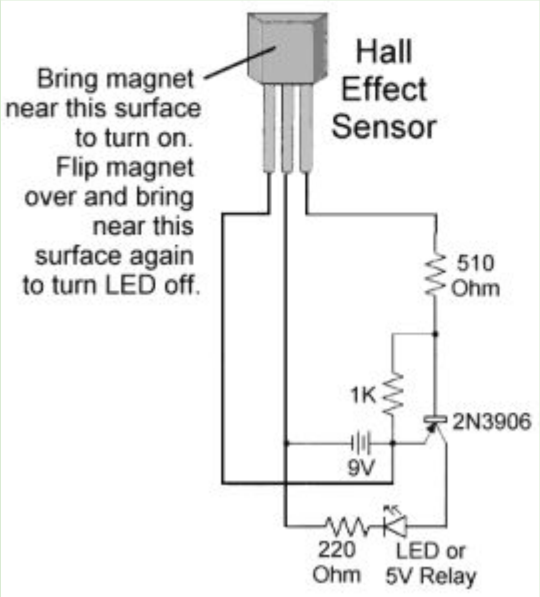
- Range Finders
- Light Sensors

Common Sensors

- Range Finders
- Light Sensors
- Limit Switches

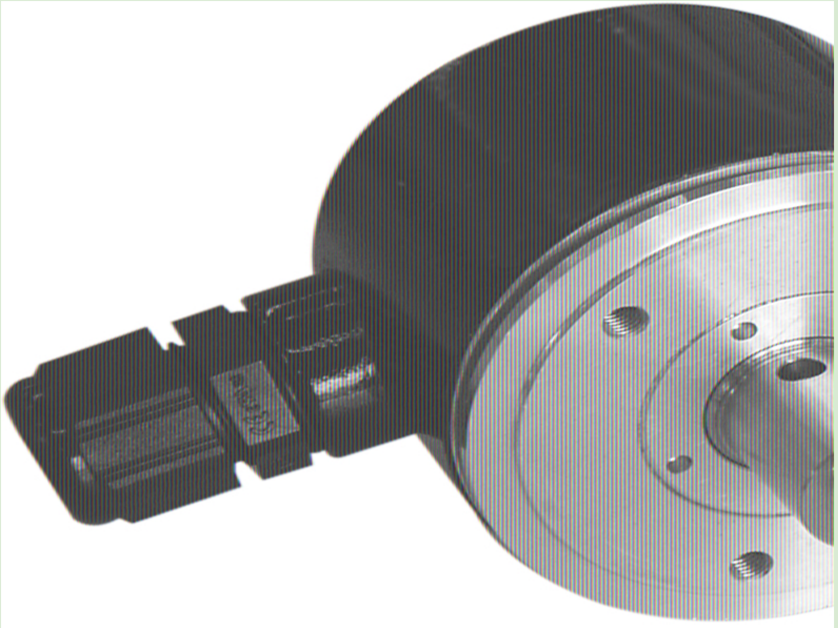
Common Sensors

- Range Finders
- Light Sensors
- Limit Switches
- Hall Effect



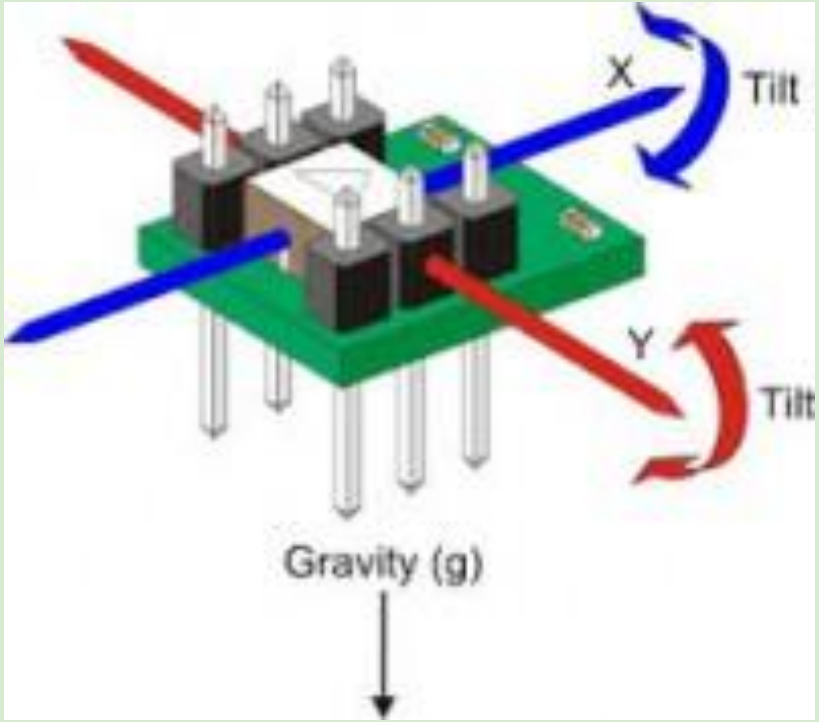
Common Sensors

- Range Finders
- Light Sensors
- Limit Switches
- Hall Effect
- Encoders



Common Sensors

- Range Finders
- Light Sensors
- Limit Switches
- Hall Effect
- Encoders
- Gyros



Common Sensors

- Range Finders
- Light Sensors
- Limit Switches
- Hall Effect
- Encoders
- Gyros
- CMU Camera
- etc

Basic Sensor Use

- There is always some value you are trying to “get” to.

- Go until you get to that value, then stop

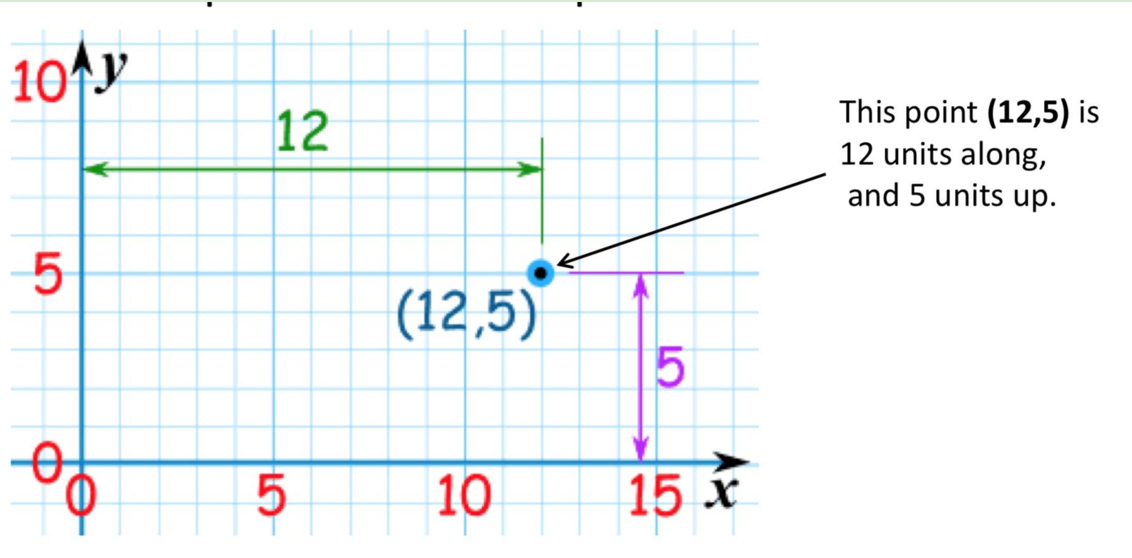
- Problems: tends to overshoot the goal

PID Control - Some Math Vocabulary

- In order to understand the math to come, it’s important to review and/or learn a few math terms and math symbols:
  - Point
  - Infinity
  - Line
  - Tangent Line
  - Variable
  - Function
  - Change
  - Slope
  - Summation

What's the Point?

- A point is an exact location. It is not a thing, but a place. It has no size or any dimensions, we just use a dot to represent where a point is.

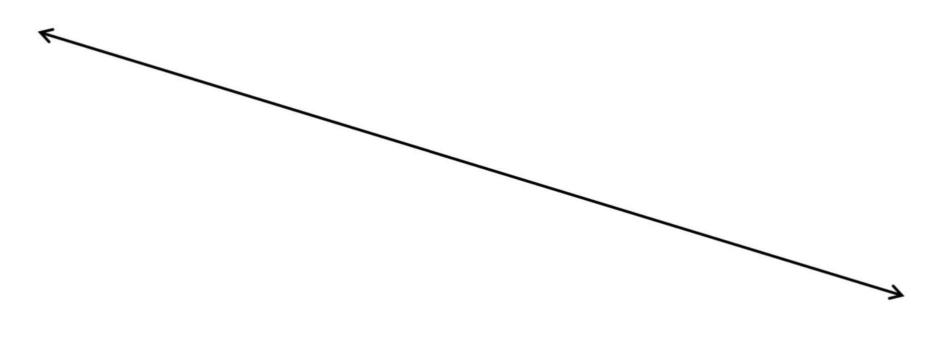


Infinite Wisdom

- Infinity is the idea that something has no end. If someone has read every single book about pyramids, you might say she has infinite knowledge about pyramids (that of course would be an exaggeration of “infinite”). She will sure stop talking about them at some point, right?

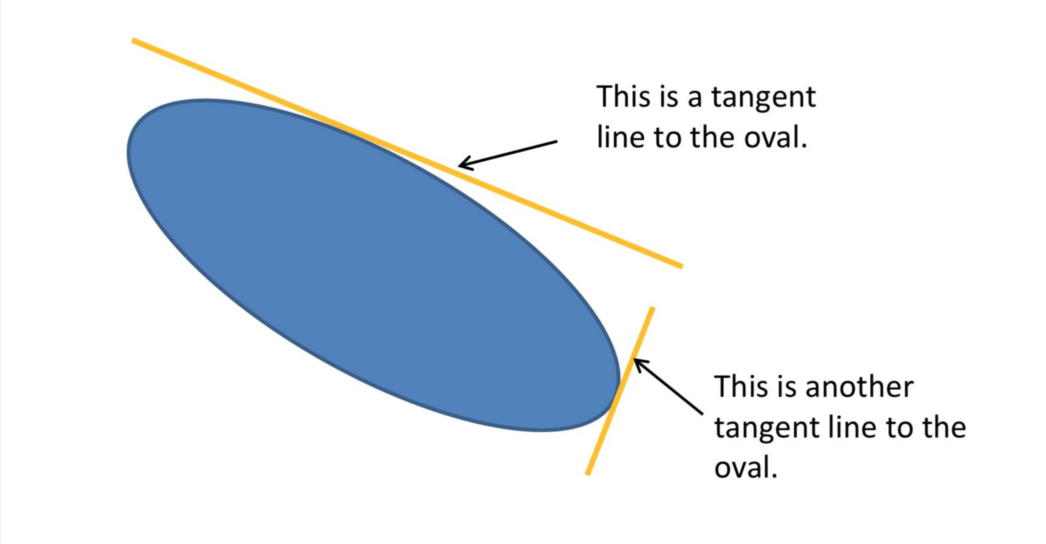
Walk the Line?

- A line is a straight, one-dimensional string of infinite points. If you draw a line with a pencil, it just represents where the line is, because if you look at the line under a microscope it would show a line with a large width!



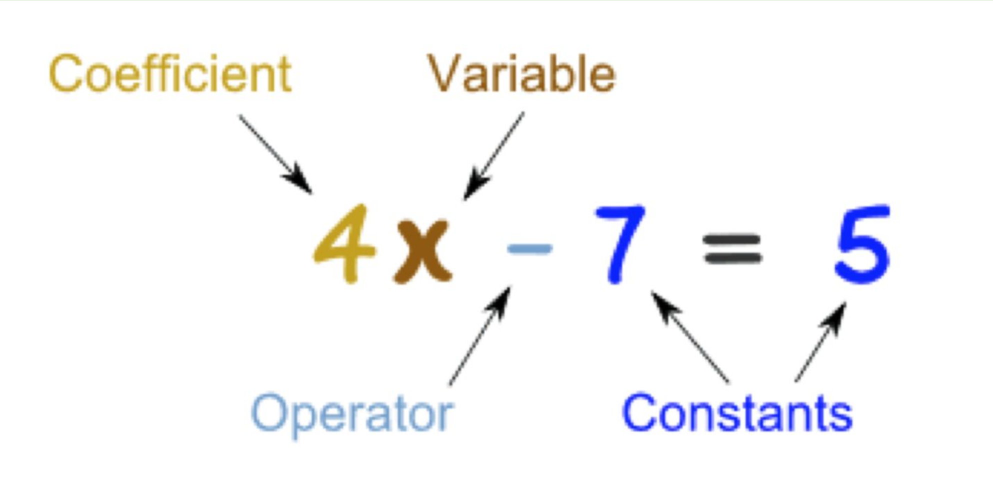
Tangential Thinking

- A tangent line touches a curve at just one point (location). The “radius” of that curve is always perpendicular (right angle, 90 degrees) to the tangent line.



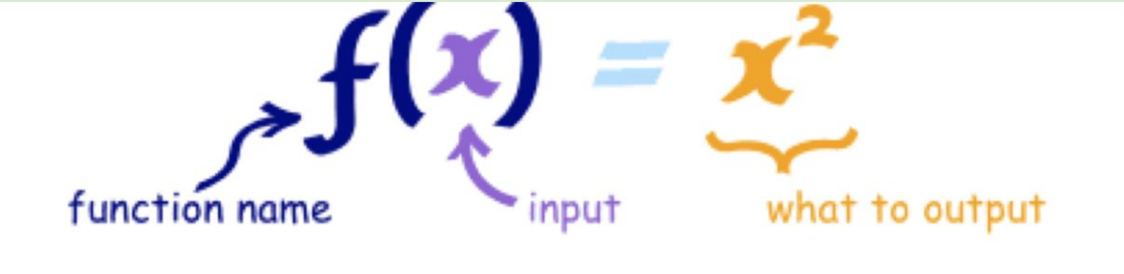
Variables hold value

- A variable is simply a symbol, or letter, or word, or even a phrase (ThisIsALongVariableName) that can contain a value (an integer, real number, or even a color!).



Functions: Math Machines!

- A function relates some input or inputs to some output. A function is like a machine that cranks on the input and generates an output.



- A function gets a name, like “f”, or “g”, or “PIG”, or “AREA”. A function simply does some action on the input to make an output.

The Change will do you good!

- A useful symbol used in math and engineering is the

Delta symbol: Δ

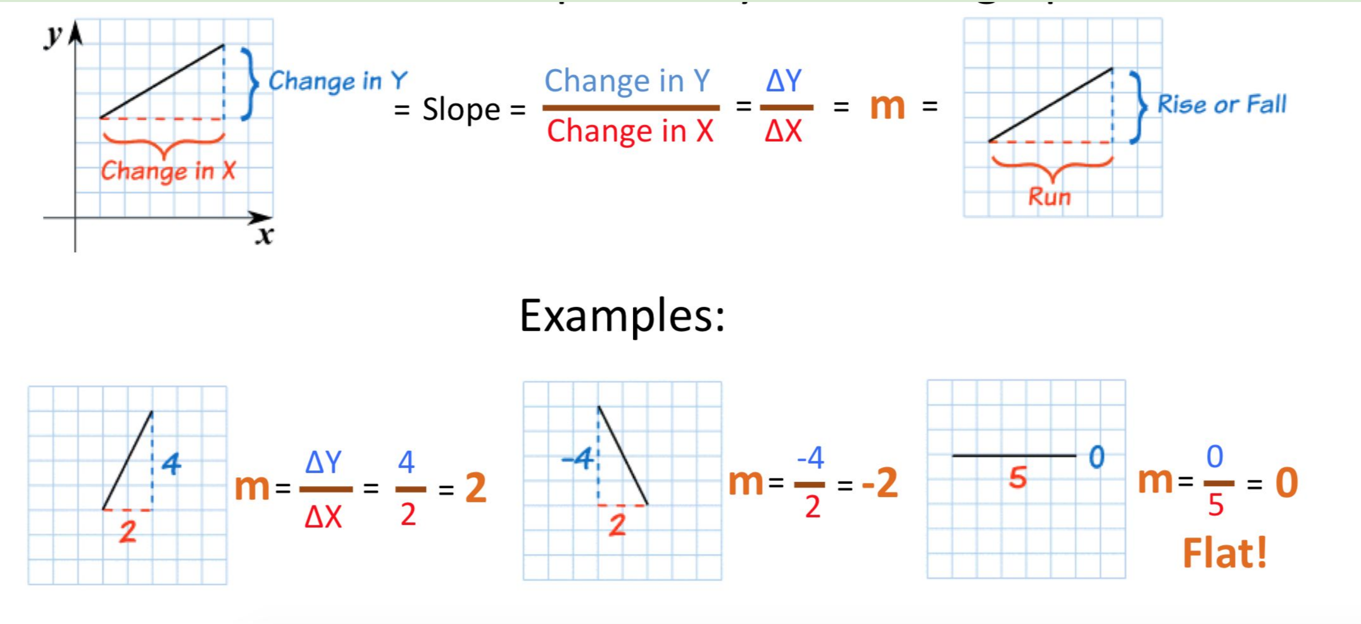
- It represents change in value (or the difference in value).

Δ = change in value

- If a value of a light sensor changes from 75 to 50 from one time we sample it to the next, we say it has a “Delta”, or Δ of -25.

Slope (or Gradient)

- Slope of a Line: We can find the slope of any line on a graph:



In Summation...

- Another useful symbol used in math and engineering is the Summation symbol: $\sum$
- It represents summing up, or adding together a bunch of values.

$\sum$ = summation of values

- For example, the $\sum$ of all integers from 1 to 9 is $1+2+3+4+5+6+7+8+9 = 45$.

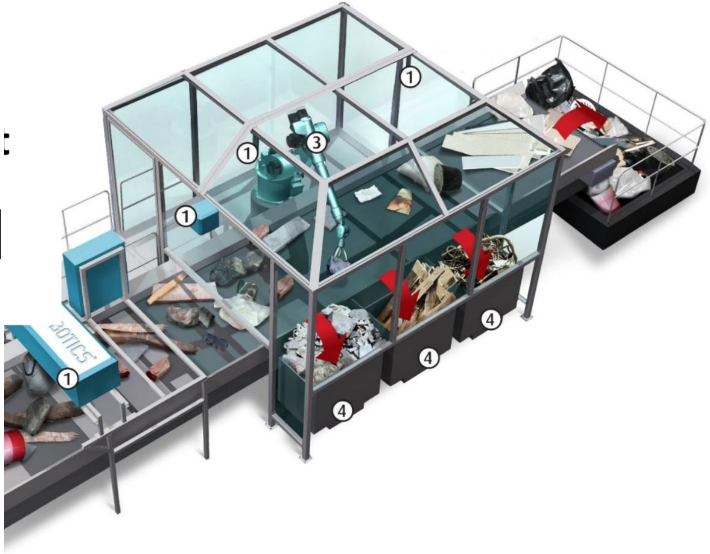
What is PID?

- PID stands for Proportional, Integral, and Differential (P-I-D). It is the most common closed loop control method used to control real-world things like temperature and cruise control speed of a car.

- For example, when your home is too warm, you turn down the thermostat, and a PID controller turns on and off your air conditioner to keep your home at the new temperature you set.

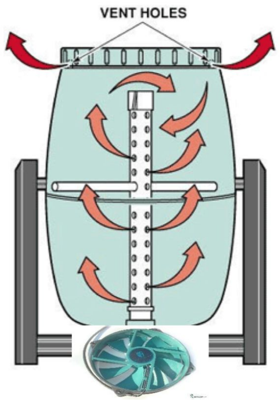
Example of PID Control

A Trash Sorter Conveyor Belt Speed



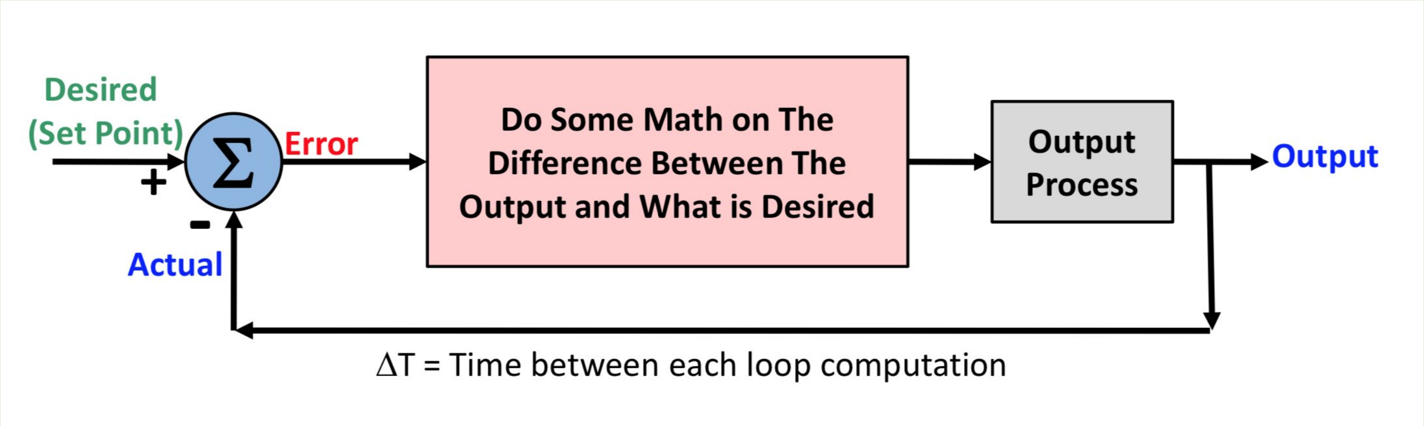
- Output: Missed Sorted Trash
- SetPoint:Allowed Missed Trash Rate
- Error: Missed Sorted Trash Rate above Set Point
- Output Process:Belt Speed

An Aerator fan for composting



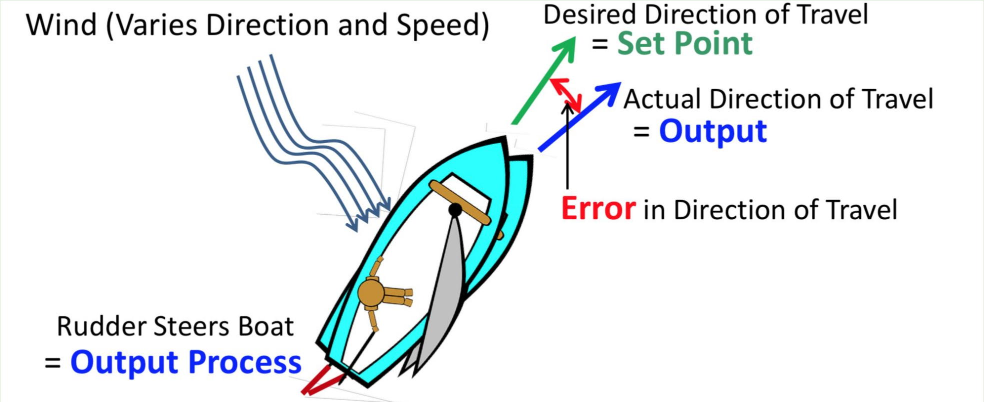
- Output: Temperature of Compost
- SetPoint:Ideal Compost Temperature
- Error: Difference in Temperature of Compost
- Output Process: Aerator Fan Speed

A Feedback Control Loop



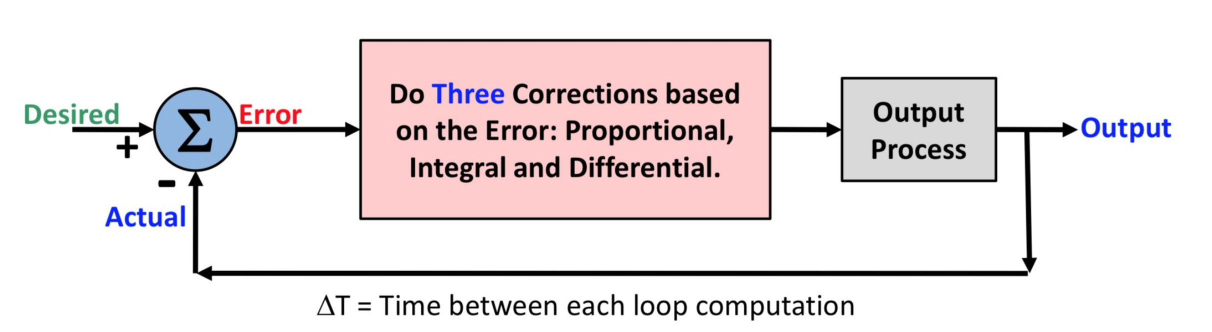
- A Feedback Control Loop takes the actual output and compares it to the desired output (finding the error) at some loop rate and tries to make the actual output match the desired.

Consider A Sailboat



- In a sailboat, the human controlling the rudder is the feedback control loop process for steering the boat in the right direction. As the wind changes direction and speed the human compensates by changing the rudder position based on the error in the direction the boat is going.

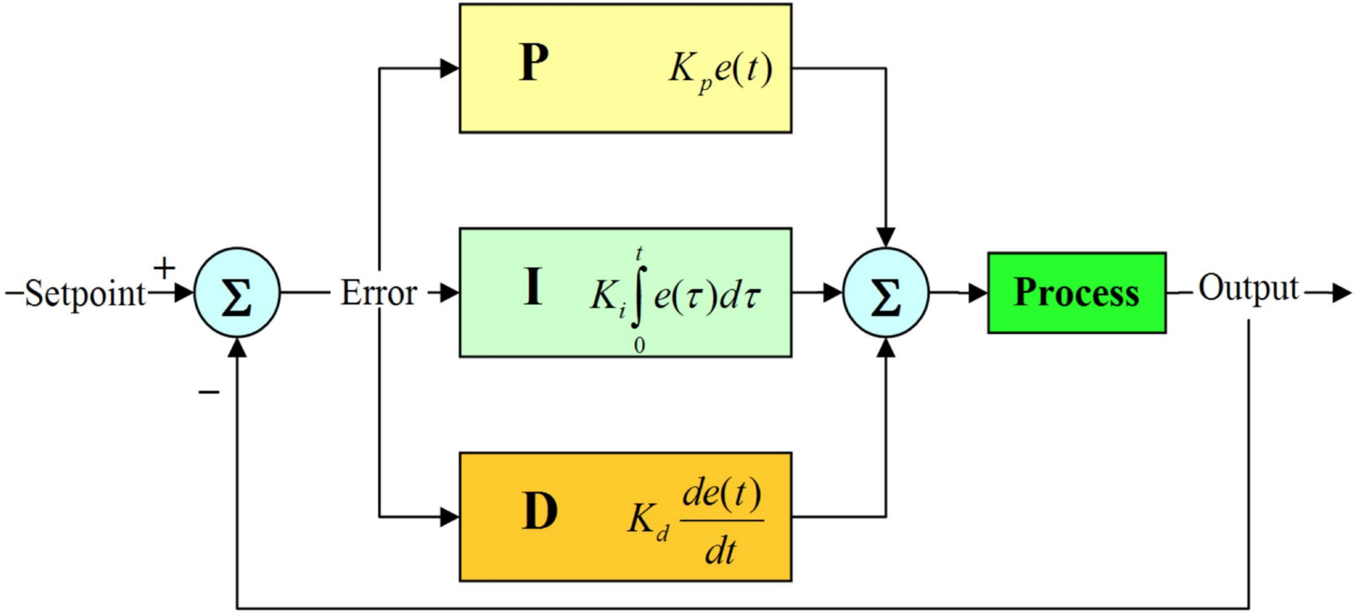
PID Feedback Control Loop



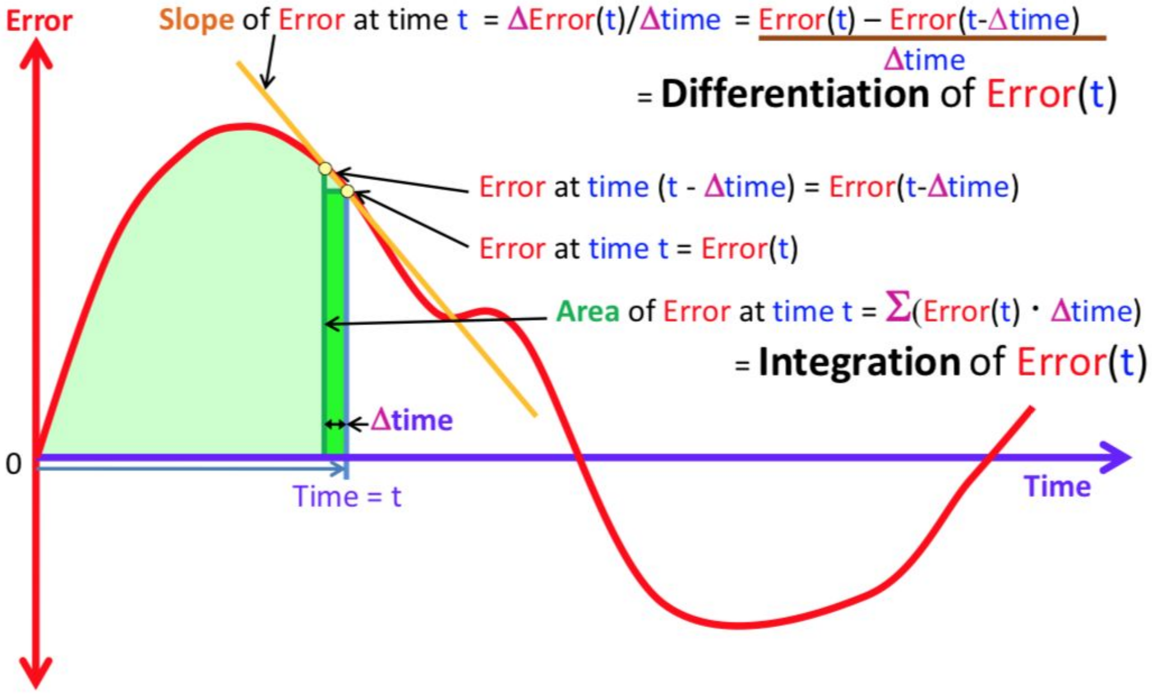
A PID Feedback Control Loop uses Error term three different ways:

- Proportional term changes output proportional to error
- Integral term sums all previous error (error area) and compensates output
- Differential term affects output based on last and current error (slope of error)

Yikes! What’s all of this nonsense? Calculus!



Graphic Representation of Error



The Proportional Term

Proportional

$= K_p \cdot Error$

Kp is the Proportional Gain Constant. It defines how much this component of the PID controller affects the output.

Error is the error in the output from what is desired. It is

computed as the difference of the desired set point and the current actual point.

The Differential Term

Differential

$$= K_d \cdot (\Delta Error / \Delta T)$$

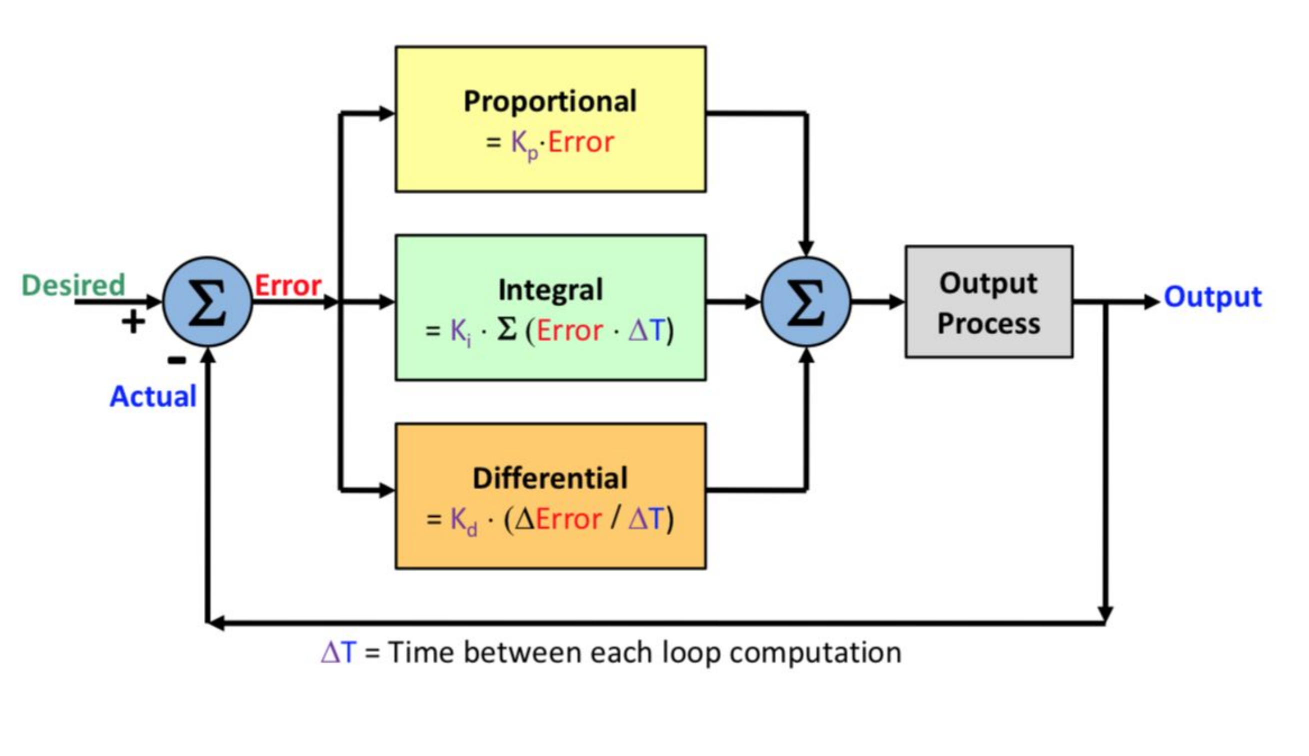
Kd is the Differential Gain. It defines how much the differential component of the PID controller affects the output.

Δ is a Greek symbol delta that we use to mean change. Here, we want to compute the change since the last time we did this calculation.

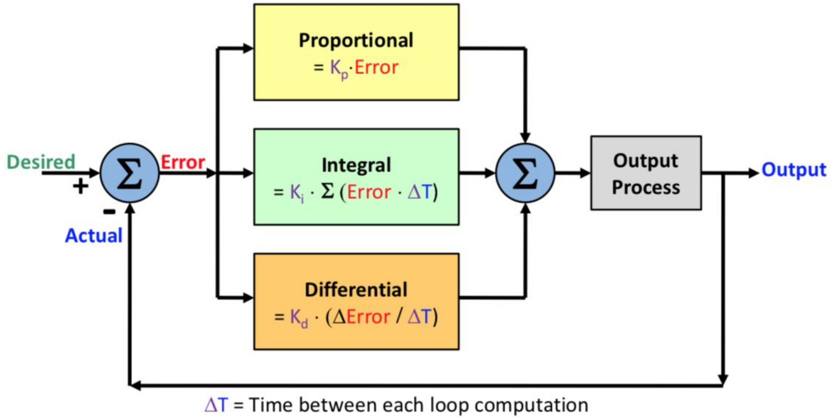
ΔError is the change in error of the output from what is desired. It is computed as the difference between the current error and the error last time we did this calculation.

ΔT is the amount of time since the last time we performed this calculation. T is the letter we use to mean Time. So T is the change in time since we did this computation last. It is our sampling time of the error.

Putting the PID together



PID Pseudo Code

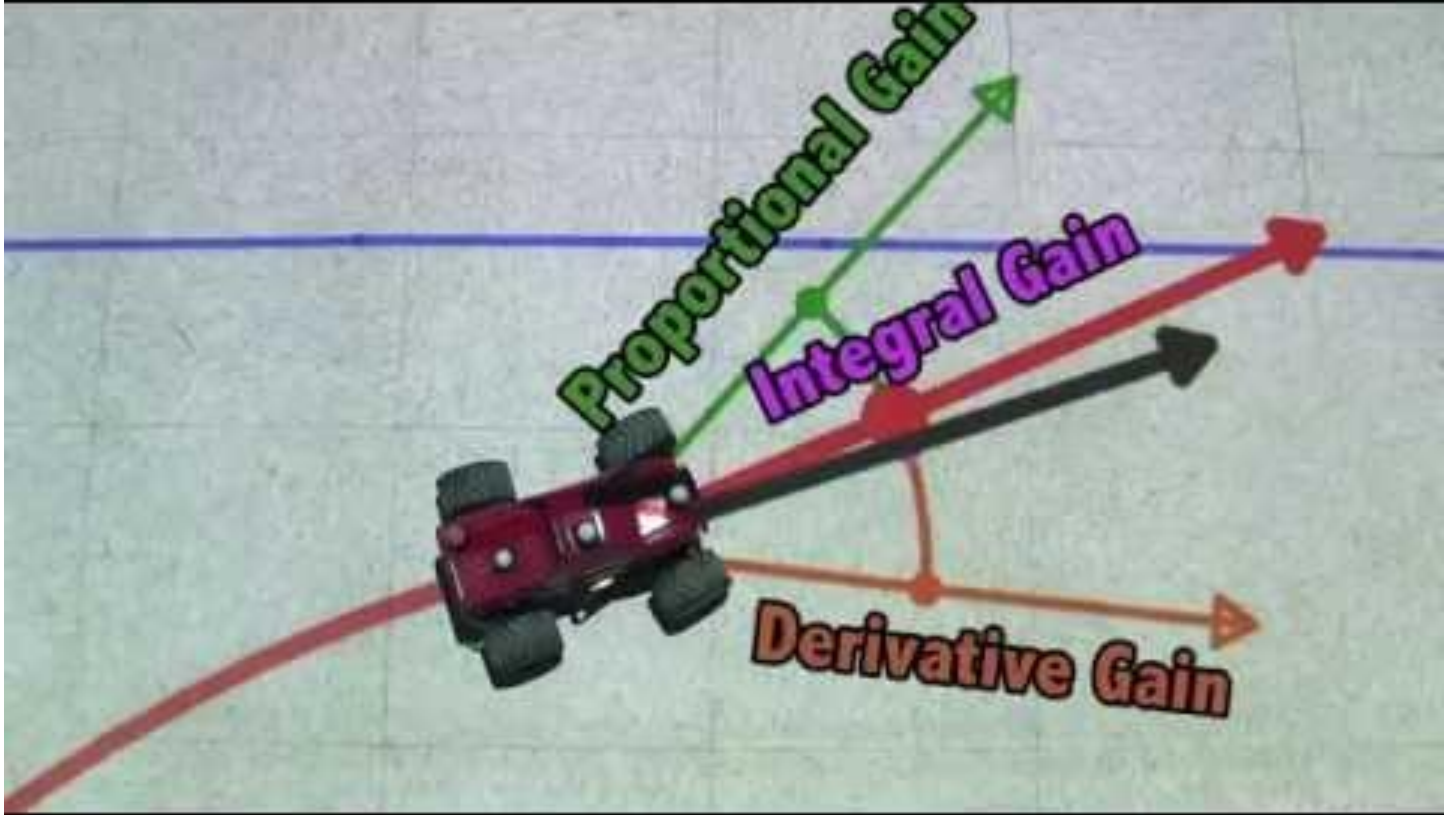



```
Previous_Error = 0
Integral       = 0
DeltaT         = 1/100

Loop:
    Error = Desired - Actual
    Integral = Integral + (Error * DeltaT)
    Derivative = (Error - Previous_Error)/DeltaT
    Previous_Error = Error
    Output =((Kp*Error) + (Ki*Integral) + (Kd*Derivative))
Wait(DeltaT)
Repeat Loop
```

Having PID code in your hand is like having a freshly factory built, un-tuned Piano -

- It is completely useless as an instrument
- - until it is tuned!


Proportional Gain
Integral Gain
Derivative Gain

Finding the Gains K_p, K_i, and K_d

Ziegler-Nichols Method

The hard part of using a PID controller is finding the gains (Kp, Ki, and Kd) for each of the three contributing components for a specific controller use. Finding these gains requires iterative (repeating) experiments with thecontroller. This is called “tuning”. One common tuningmethod is called the Zeigler-Nichols method named after the gentlemen that came up with it, John G. Zeigler and Nathaniel B. Nichols.

Ziegler-Nichols Method

Step 1: Set Gains Ki and Kd to zero. Thus, the controller becomes a P-type controller only.

Step 2: Start with Kp as a “small” value and increase Kp until the output of the controller starts to oscillate (doesn’t settle to an output value). This gain value is called the critical gain, Kc.

Step 3: Measure the time period of the oscillation when Kp = Kc. This is called the critical oscillation period, Pc.

Step 4: Use the values Kc and Pc to determine Kp, Ki, and Kd using the formulas that Zeigler and Nichols determined for a P, PI, or PID controller (shown on next slide).

Ziegler-Nichols Method Gain Values

Gain	Rise Time	Overshoot	Settling Time	Error at equilibrium
K_p	Decrease	Increase	Little change	Decrease
K_i	Decrease	Increase	Increase	Eliminate
K_d	Indefinite	Decrease	Increase	None

Example

- We want to drive our robot forward 10 metres, and then stop

- The concepts we explore do not always have to involve distance. (ie your goal may be to maintain a certain speed or certain rate of acceleration)

Proportional (P)

- A measure of how far you are away from your goal.

- The larger the P value, the larger the motor output should be, since you have farther to go.

- In our example this would be the distance remaining between our current position and our goal of 10 metres.

Derivative (D)

- How quickly you are moving towards your goal.

- Unlike P, when D is higher you want to “pull back” your motor output. This keeps you from heading towards you target too quickly. If P is still large (ie you are far from your target) then it will “overpower” the D.

- In our example this would be the speed we are traveling towards the goal.

Integral (I)

- The integral is how long you have been away from your target.

- The idea is that the longer the robot has not made it to the target, the more power should be applied to get it there.

- The most difficult part to incorporate. PD control alone can be very effective.

Calculating the Motor Output

- The basic formula is: output = P * CP + I * CI - D * CD

- Where CP, CI, and CD are values tweaked to get the proper result.

- There are some methods to calculate the values of CP, CI, and CD, but it is generally more effective to find them by trial in error in robotics.

Method for Setting Values

- Start with CP small and CI, CD both zero.

- Raise CP until the robot is oscillating consistently around the target.

- Once this is accomplished, start increasing CD until the robot stops oscillating.

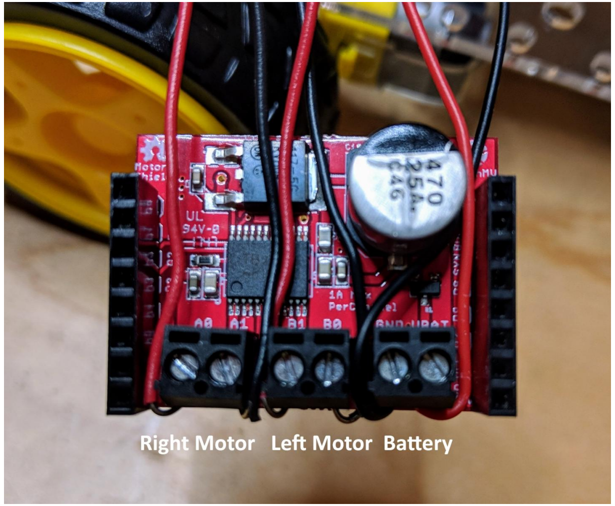
- Then add CI until the robot stops within a desired range of the target.

```python
import sensor, image, pyb, math, time
from pyb import LED
from pyb import Pin, Timer
```

Python code in one module gains access to the code in another module by the process of importing it.

The pyb module contains specific functions related to the board. Example: Delay

```
# these are motor driver pins, which set the direction of each motor
pinADir0 = pyb.Pin('P0', pyb.Pin.OUT_PP, pyb.Pin.PULL_NONE)
pinADir1 = pyb.Pin('P1', pyb.Pin.OUT_PP, pyb.Pin.PULL_NONE)
pinBDir0 = pyb.Pin('P2', pyb.Pin.OUT_PP, pyb.Pin.PULL_NONE)
pinBDir1 = pyb.Pin('P3', pyb.Pin.OUT_PP, pyb.Pin.PULL_NONE)
```



```
# Dir0/1 must be not equal to each other for forward or backwards
# operation. If they are equal then that's a brake operation.
# If they are not equal then the motor will spin one way other the
# other depending on its hookup and the value of channel 0.

pinADir0.value(1)
pinADir1.value(0)


# Dir0/1 must be not equal to each other for forward or backwards
# operation. If they are equal then that's a brake operation.
# If they are not equal then the motor will spin one way other the
# other depending on its hookup and the value of channel 0.

pinBDir0.value(0)
pinBDir1.value(1)
```

```
tim = Timer(4, freq=1000) # Frequency in Hz

cruise_speed = 20 # how fast should the car drive, range from 0 to 100
steering_direction = -1   # use this to revers the steering if your car goes in the wrong direction
steering_gain = 1.65  # calibration for your car's steering sensitivity
steering_center = 37  # set to your car servo's center point
kp = 0.8   # P term of the PID
ki = 0.0     # I term of the PID
kd = 0.4    # D term of the PID
```

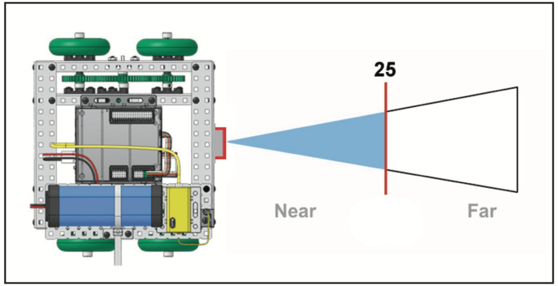
```
# Color Tracking Thresholds (L Min, L Max, A Min, A Max, B Min, B Max)
# The below thresholds track in general red/green things. You may wish to tune them...
# old thresholds = [(30, 100, 15, 127, 15, 127), # generic_red_thresholds
#              (30, 100, -64, -8, -32, 32), # generic_green_thresholds
#              (0, 15, 0, 40, -80, -20)] # generic_blue_thresholds

threshold_index = 0
# 0 for red, 1 for green, 2 for blue

thresholds = [(18, 89, 20, 97, 69, -10), # generic_red_thresholds
            (0, 100, -87, 18, -128, 33), # generic_green_thresholds
            (0, 100, -128, -10, -128, 51)] # generic_blue_thresholds
# You may pass up to 16 thresholds above. However, it's not really possible to segment any
# scene with 16 thresholds before color thresholds start to overlap heavily.
```

Thresholds are values that set a cutoff in a range of values, so that even if there are many possibilities, the value eventually falls above the threshold, or below the threshold.

Using thresholds allows you to perform certain behaviors depending on where a certain value (usually a sensor value) falls in relation to the threshold.



If you look at this image, it shows an VEX using an Ultrasonic Sensor. The threshold in this case is 25 inches. We can create behaviors that tell the robot to go forward until Ultrasonic Sensor detects something closer than 25 inches.

The threshold is just used to determine at which point the robot should be peforming a different behavior.

Region of Interest

```
# Each roi is (x, y, w, h). The line detection algorithm will try to find the
# centroid of the largest blob in each roi. The x position of the centroids
# will then be averaged with different weights where the most weight is assigned
# to the roi near the bottom of the image and less to the next roi and so on.
ROIS = [ # [ROI, weight]
        (38,1,90,38, 0.4),
        (35,40,109,43,0.2),
        (0,79,160,41,0.6)
       ]

blue_led  = LED(3)
```

```
old_error = 0
measured_angle = 0
set_angle = 90 # this is the desired steering angle (straight ahead)
p_term = 0
i_term = 0
d_term = 0
old_time = pyb.millis()
radians_degrees = 57.3 # constant to convert from radians to degrees
```