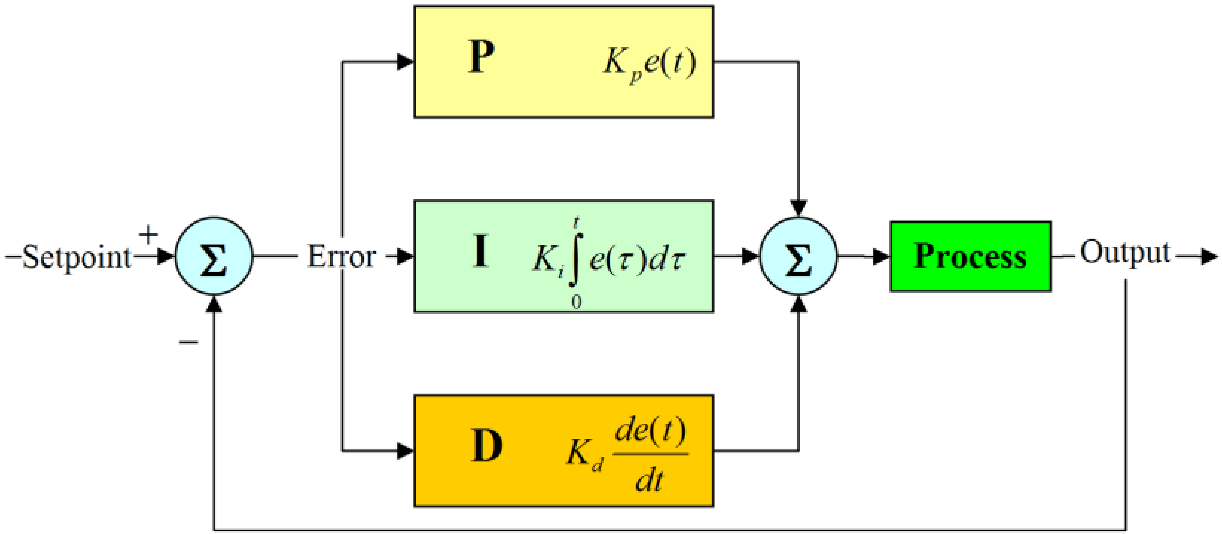
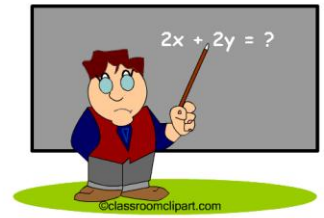


PID Closed Loop Feedback Control Theory for Fun and Profit

An Advanced Topic Workshop

Instructor: Scott Gray

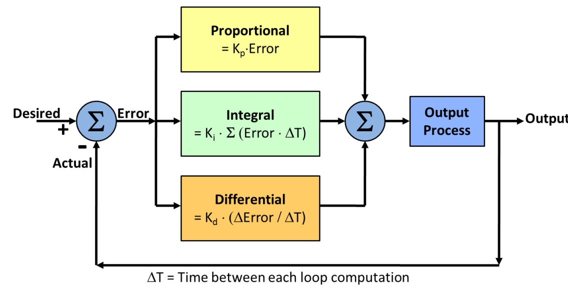


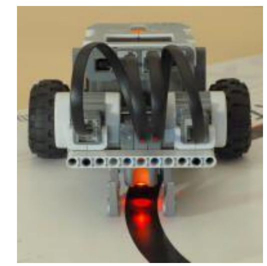
e-mail: AZSquib@GrayChalet.com

Qualifications:

- FLL Coach for 6 Years
    - Teams did very well
    - Successfully taught PID control theory to team
- Head FLL Referee for 3 Years
- Love Robotics
    - Designed Electronics and Building R2-D2 droids for years

Three Sessions for Today



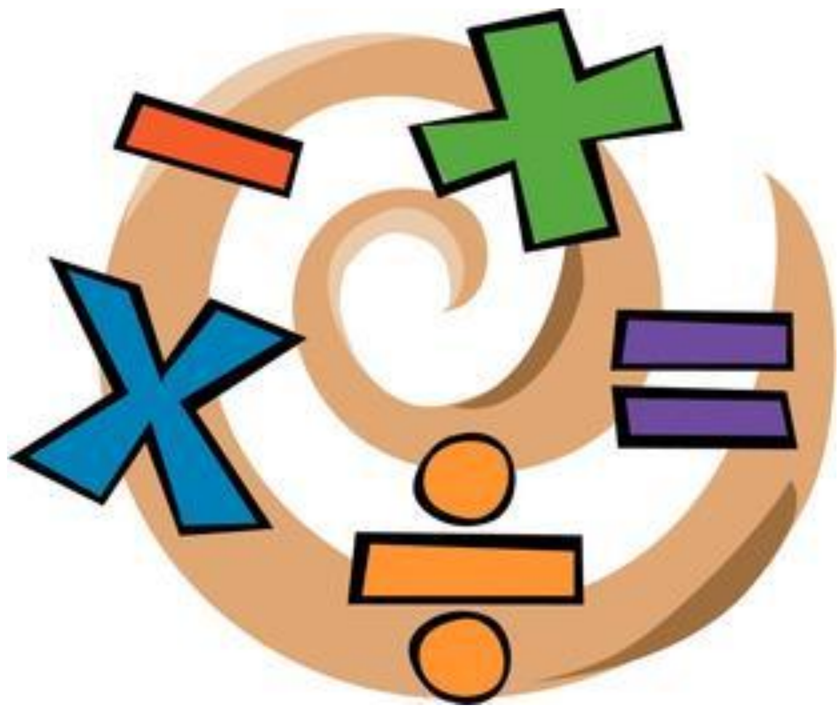


1. Math Vocabulary, Symbols, and Some Math Tools for Our Knowledge Toolbox
2. Theory and Implementation of PID Closed Feedback Loops for Our Skills Toolbox
3. Application of PID Control Loop for Line Following Robots

Some Math Vocabulary

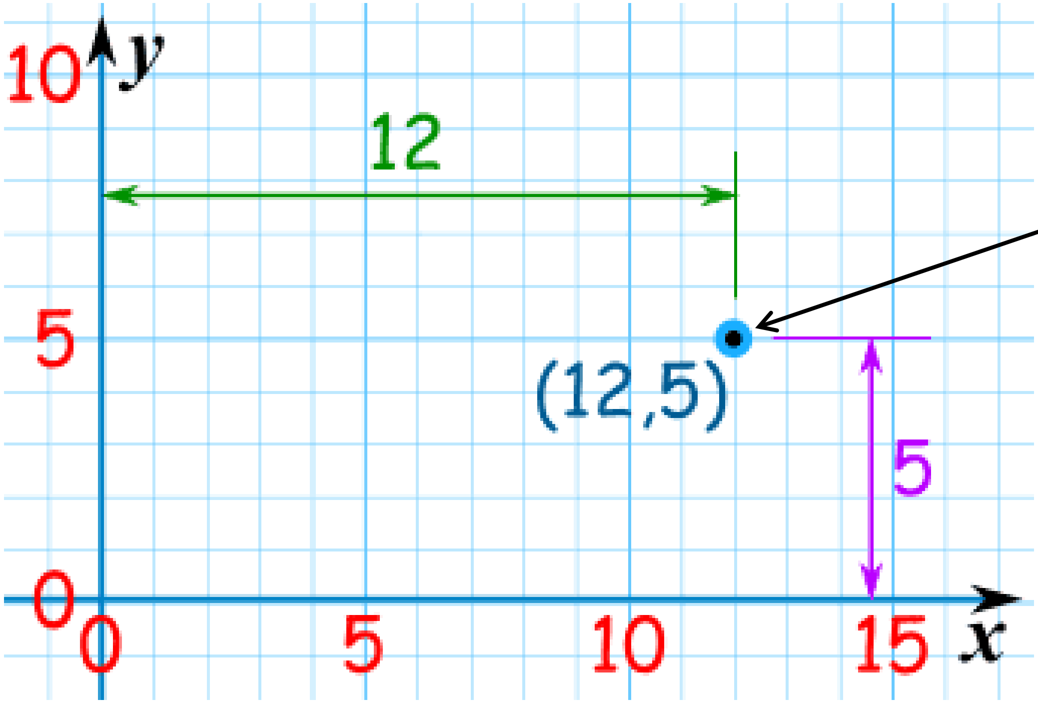
In order to understand the math to come, it's important to review and/or learn a few math terms and math symbols:

1. Point
2. Infinity - ∞
3. Line
4. Tangent Line
5. Variable
6. Function
7. Change - Δ
8. Slope (Gradient)
9. Summation- $\sum$



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.



This point **(12,5)** is
12 units along,
and 5 units up.

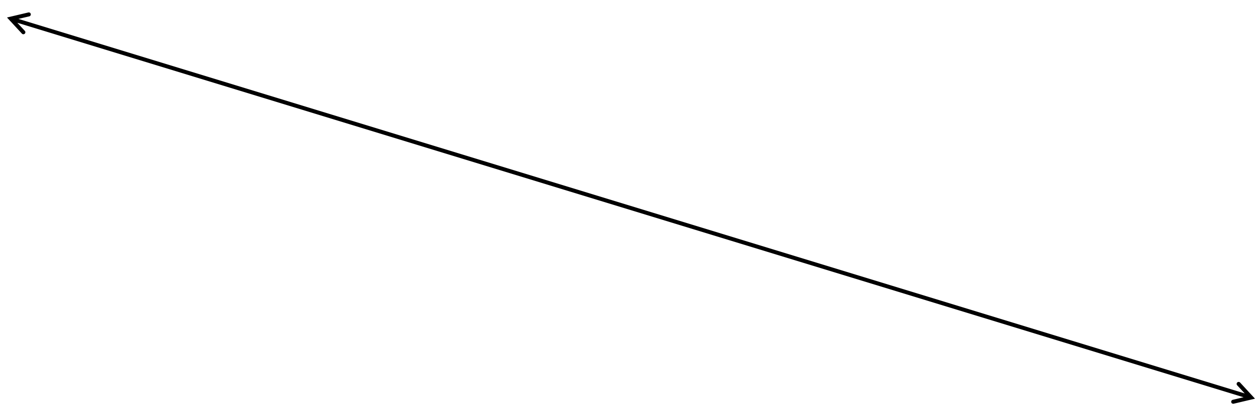
Infinite Wisdom

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?

Infinity Symbol: ∞

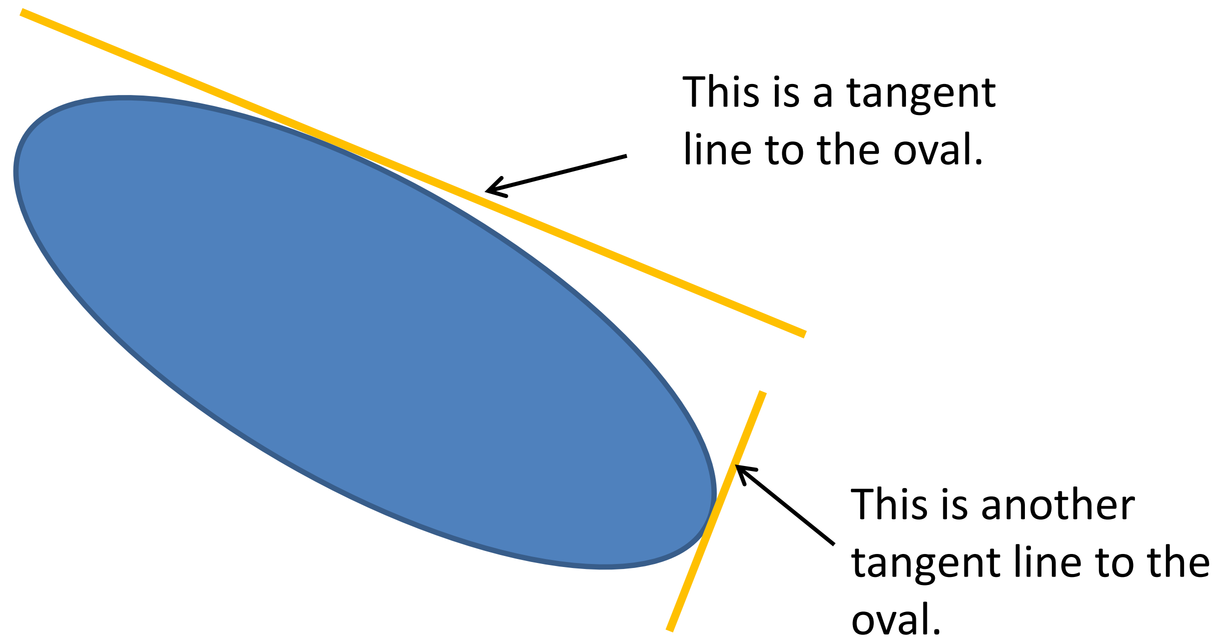
Walk the Line?

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!



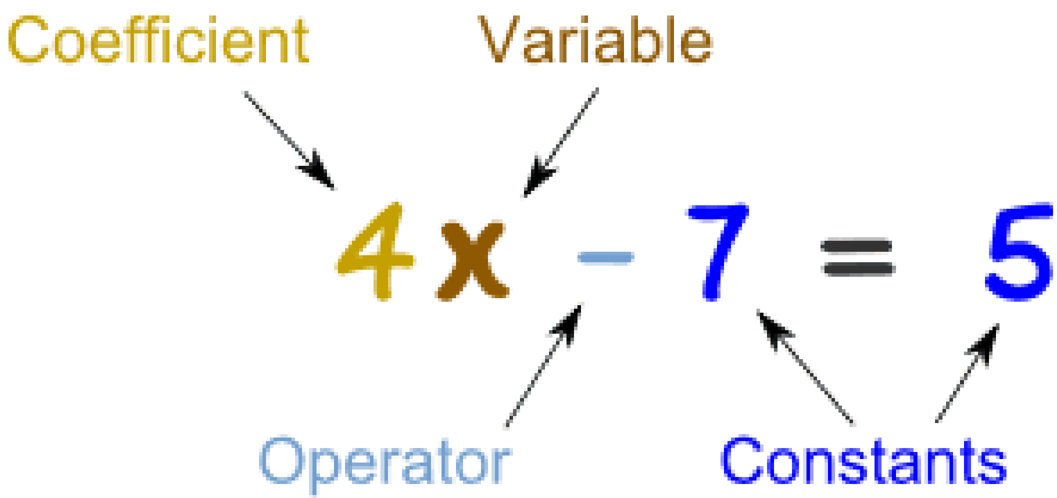
Tangential Thinking

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.



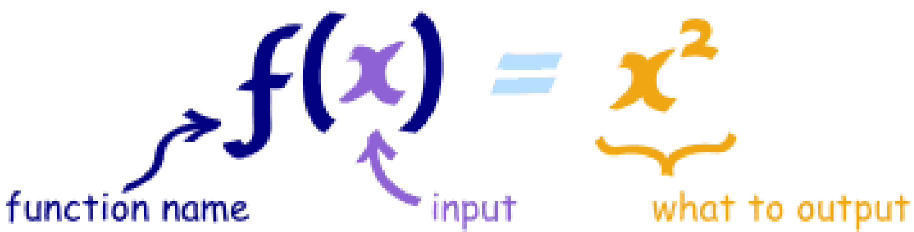
Variables hold value

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).

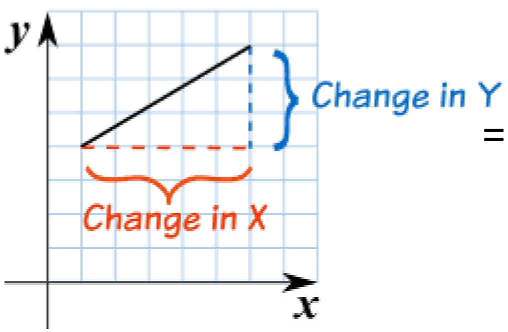
Δ = change in value

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.

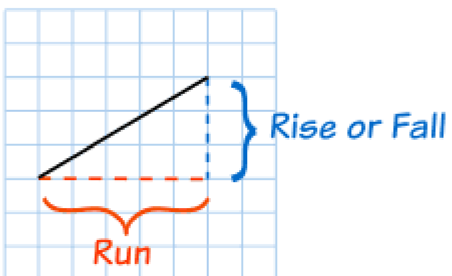
Slope (or Gradient)

Slope of a Line:

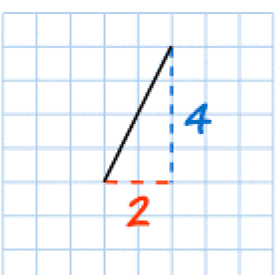
We can find the slope of any line on a graph:



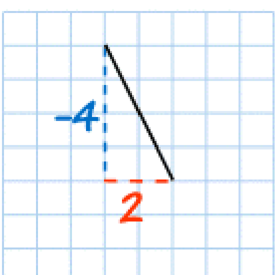
$$= \text{Slope} = \frac{\text{Change in Y}}{\text{Change in X}} = \frac{\Delta Y}{\Delta X} = \mathbf{m} =$$



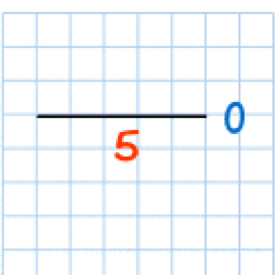
Examples:



$$\mathbf{m} = \frac{\Delta Y}{\Delta X} = \frac{4}{2} = \mathbf{2}$$



$$\mathbf{m} = \frac{-4}{2} = \mathbf{-2}$$



$$\mathbf{m} = \frac{0}{5} = \mathbf{0}$$

Flat!

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.

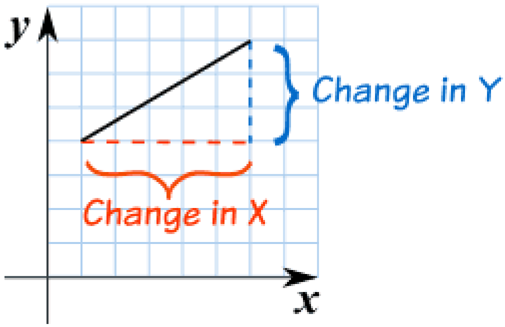
$\sum$ = summation of values

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

Algebra We Care About

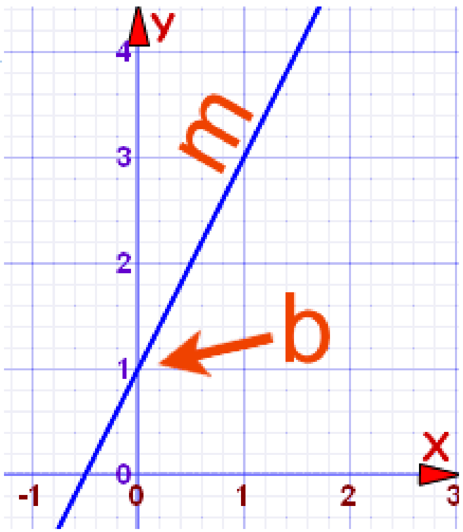
Equation of a Line:

Using Variables, we can describe any line on a graph:



Slope = $\frac{\text{Change in Y}}{\text{Change in X}}$

$= \frac{\Delta Y}{\Delta X} =$ **m**



y = mx + b

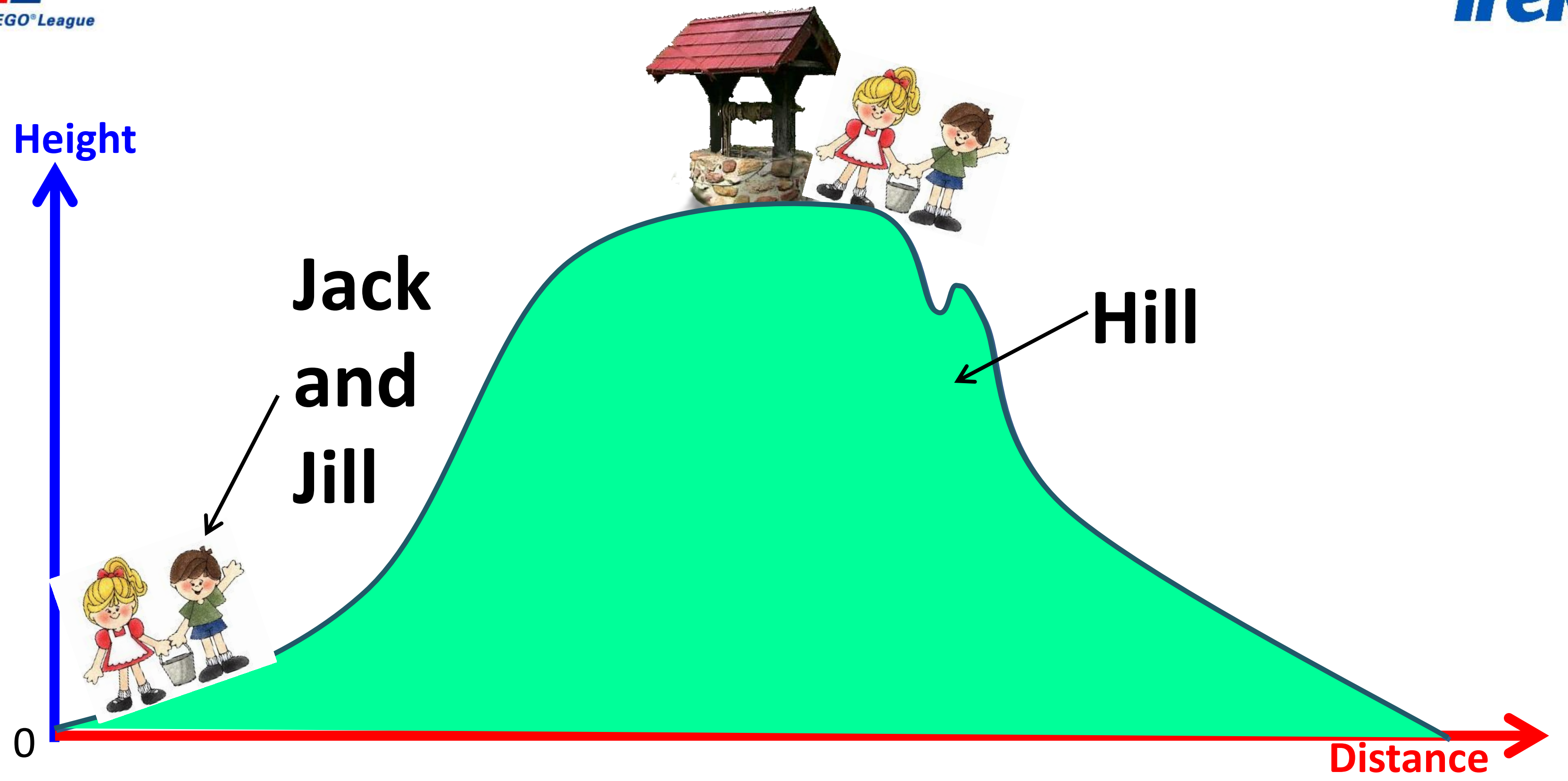
y = how far up

x = how far over

m = Slope (how steep the line is)

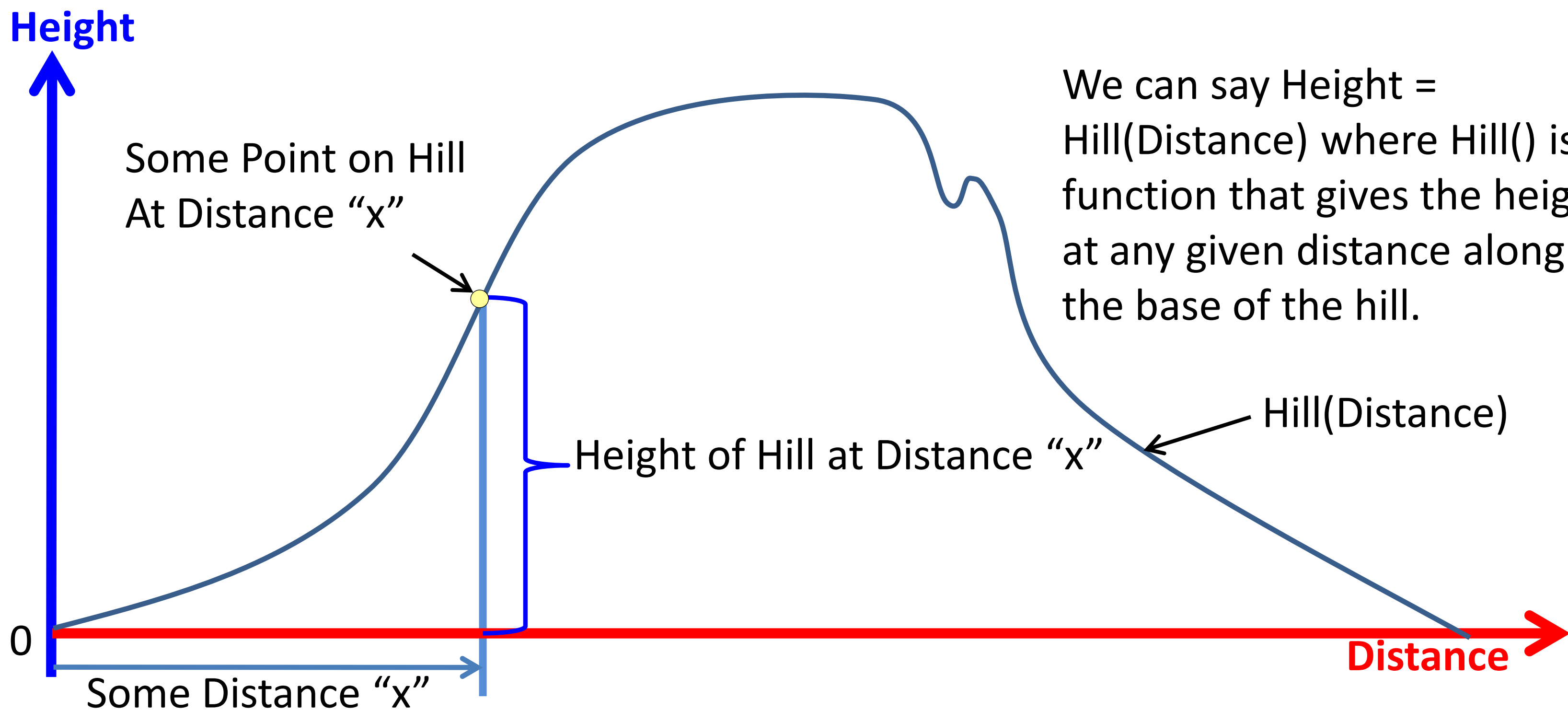
b = the Y Intercept (where the line crosses the Y axis)

Calculus with Jack and Jill



Graph of Jack and Jill's Hill with Height vs. Distance

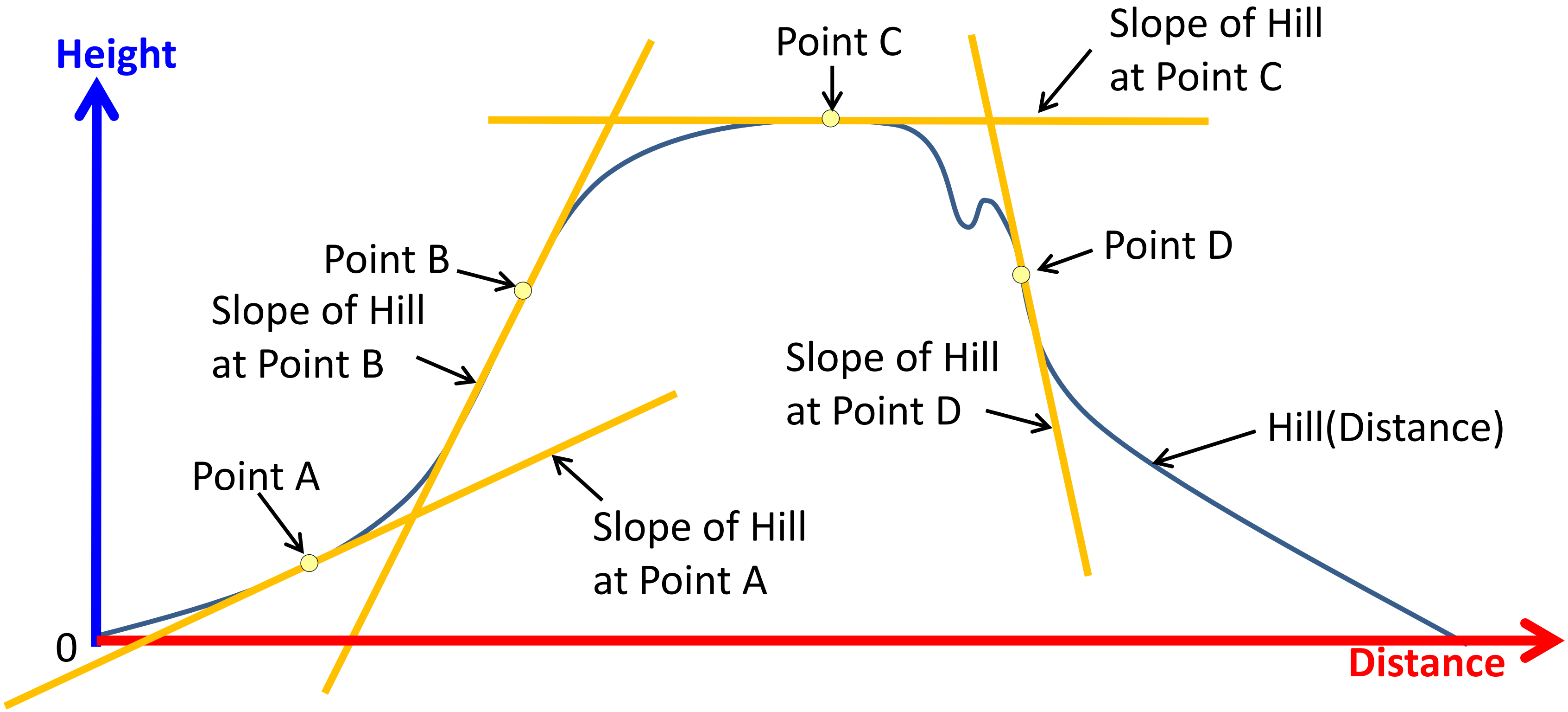
Calculus with Jack and Jill



We can say Height = Hill(Distance) where Hill() is a function that gives the height at any given distance along the base of the hill.

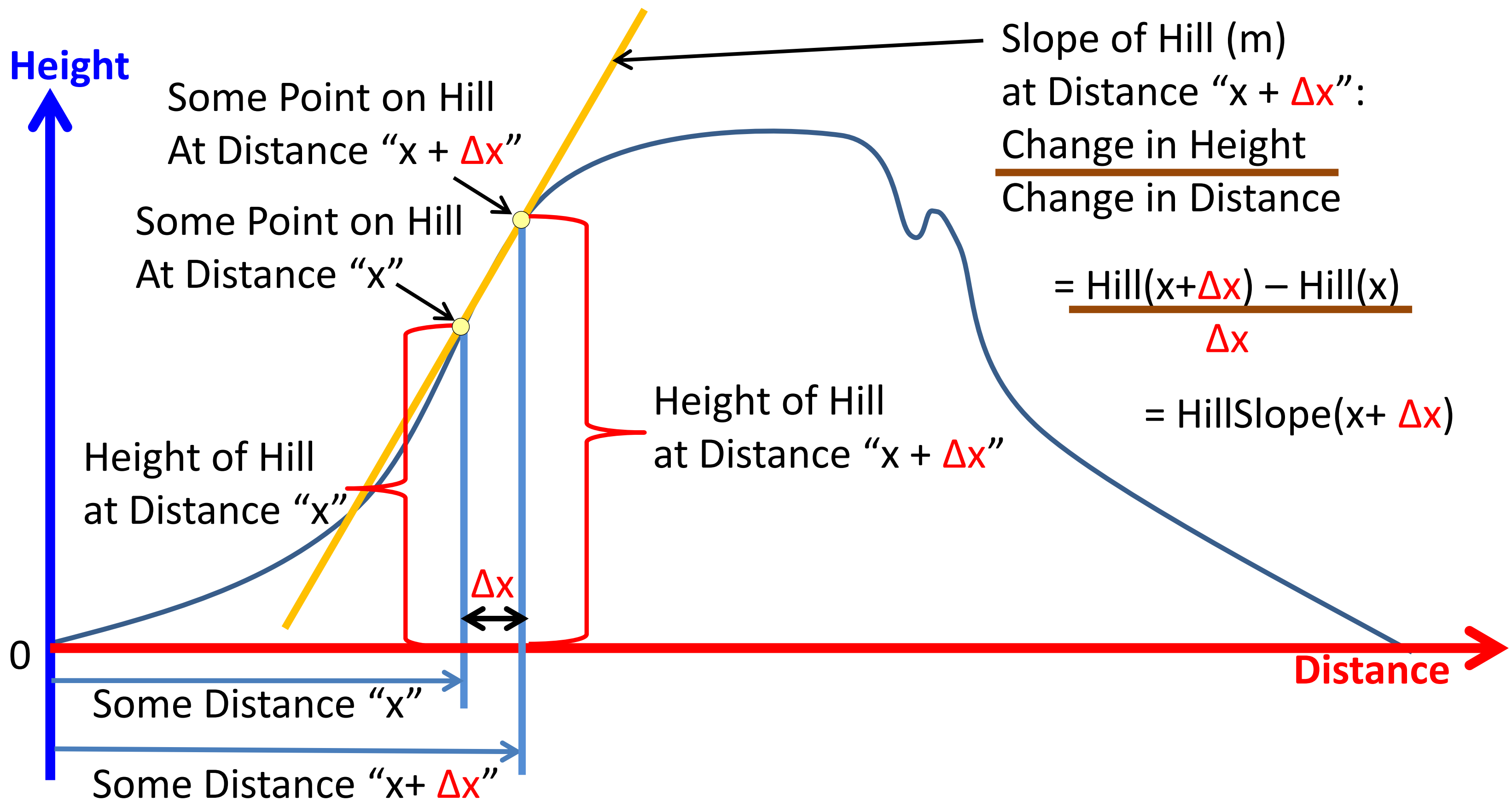
We can represent Height vs. Distance using a Function

Calculus with Jack and Jill



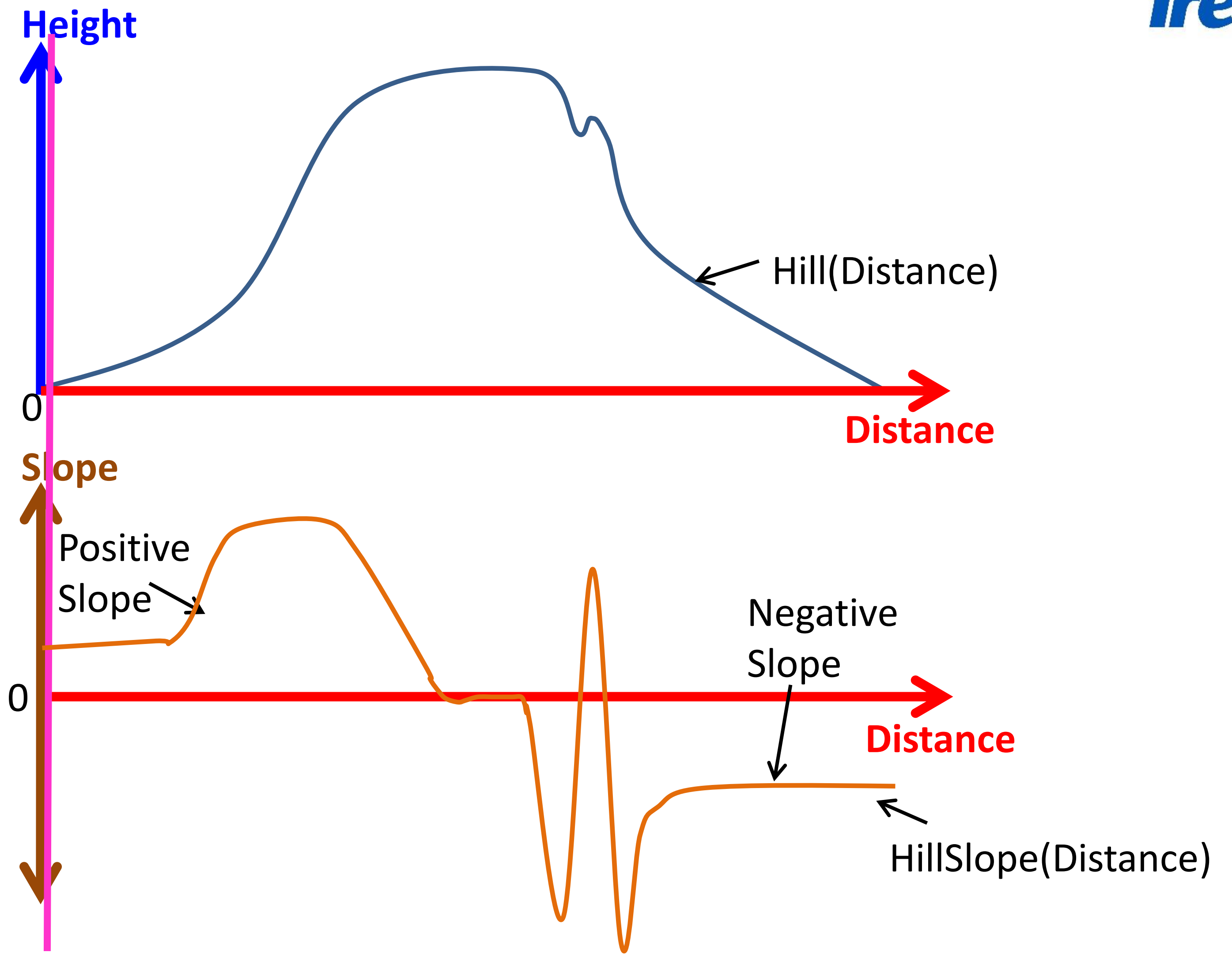
The Hill has a Slope (m) at every point along the Hill.

Calculus with Jack and Jill



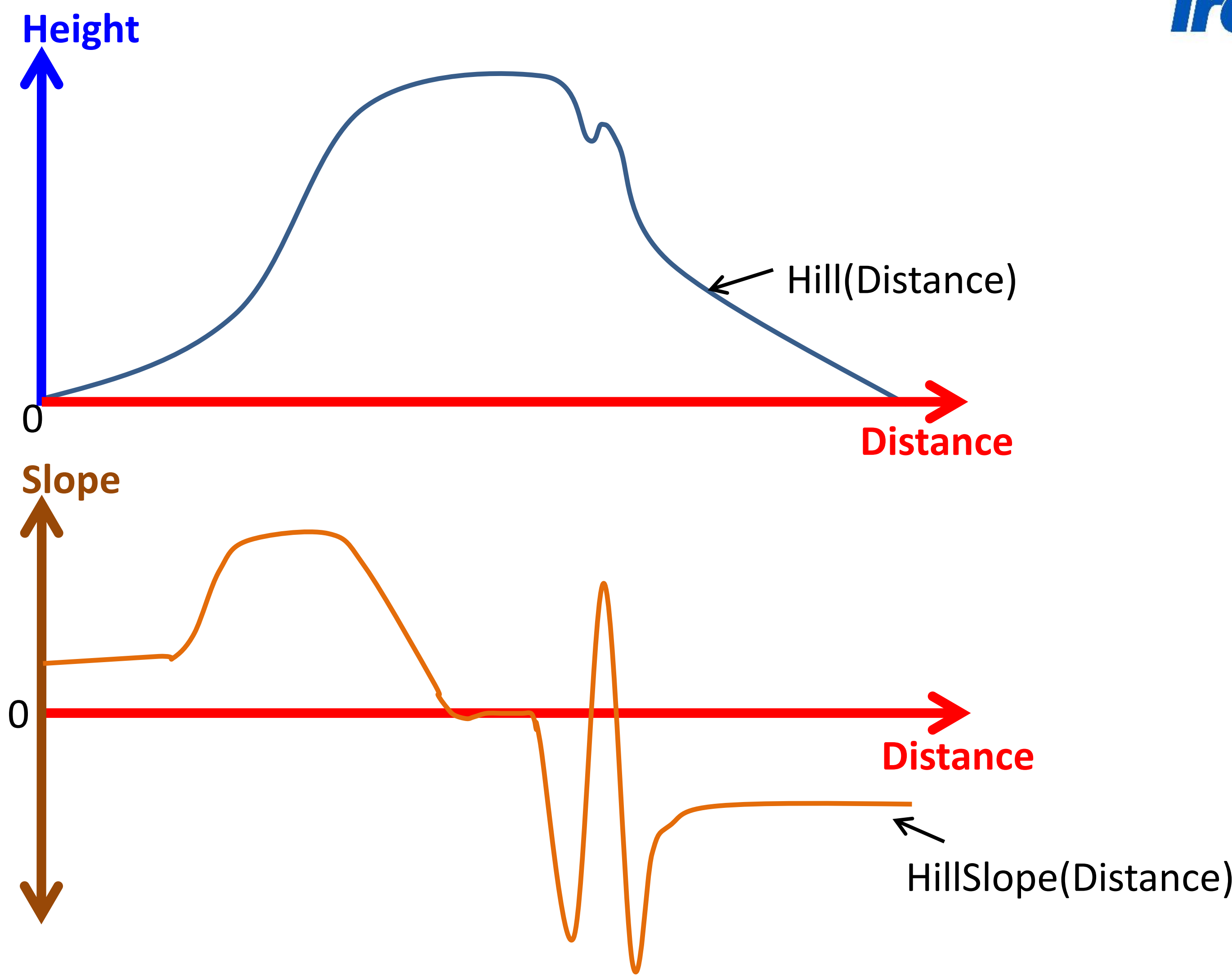
We can Approximate Slope (m) using two points.

Calculus with Jack and Jill



We can plot the slope of hill on another graph!

Calculus with Jack and Jill

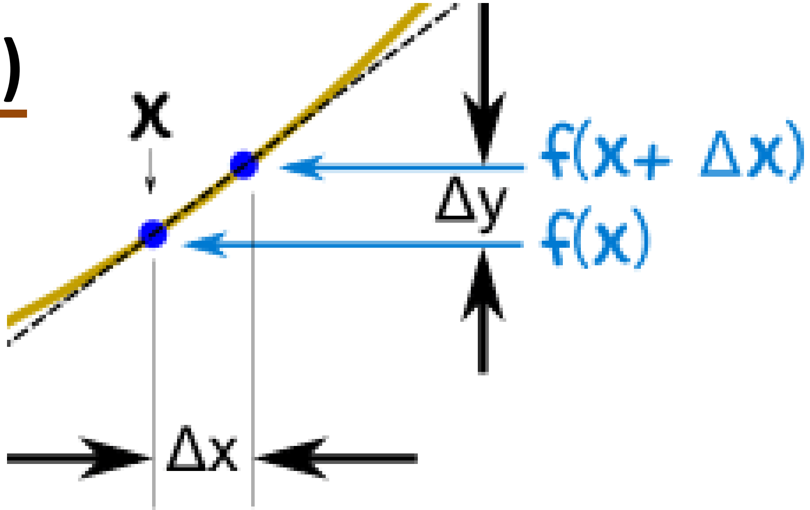


HillSlope() is the Derivative Function of Hill()!

Calculus with Jack and Jill

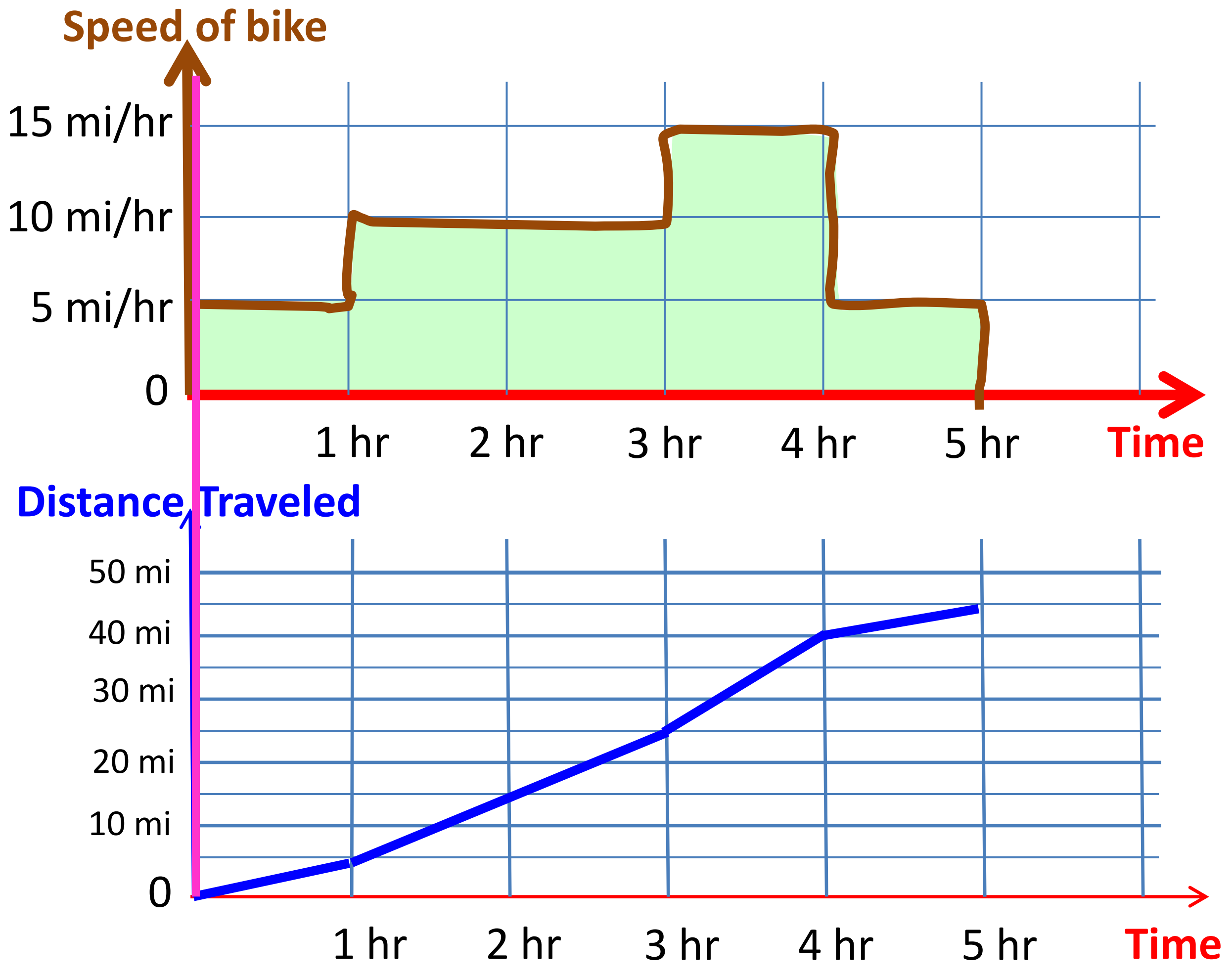
So, to recap, the derivative function of any function, say, f(x), is simply the slope of the function at every point, which we can approximate as:

Derivative of f(x) = $\dfrac{f(x+\Delta x) - f(x)}{\Delta x}$



Calculus with Jack and Jill

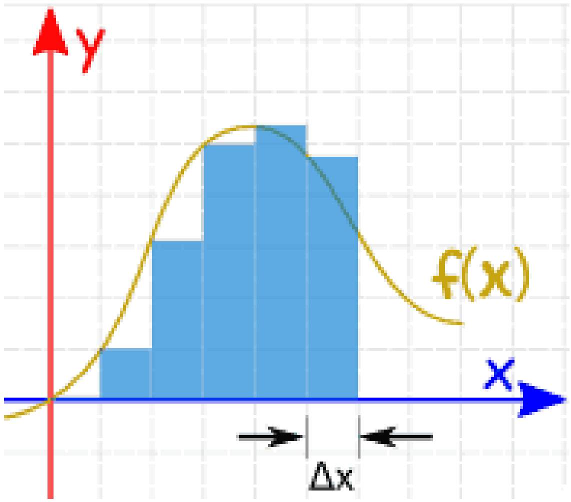
Integration is the reverse of the derivative. If we have a derivative function, we can integrate to find its integral function.



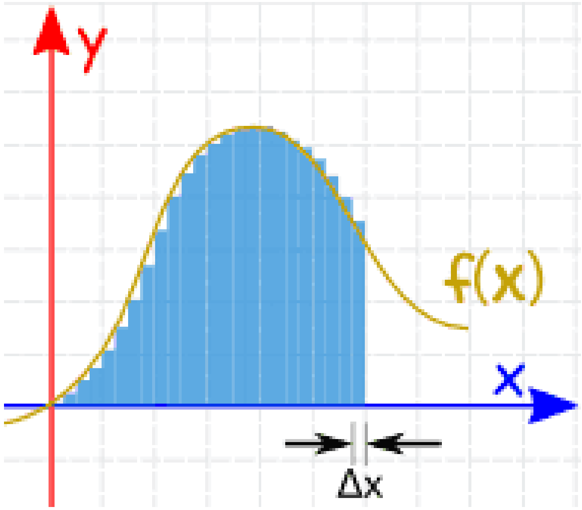
Calculus with Jack and Jill

So, integration is simply finding the area under any function, say f(x), which we can approximate as:

Integral of f(x) = ∑ f(x) X Δx

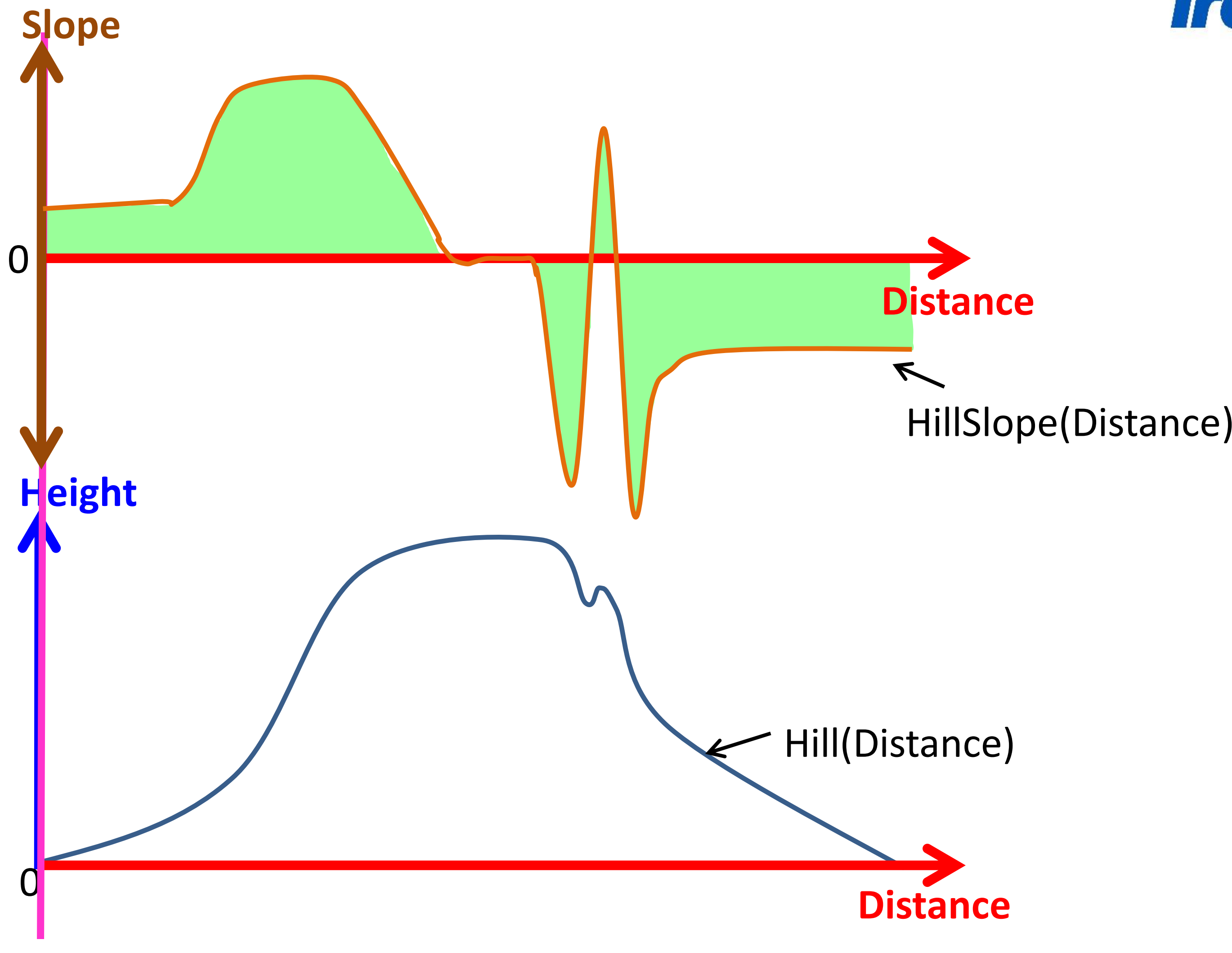


Adding up "slices" of a function F(x) by stepping every **Δx.**



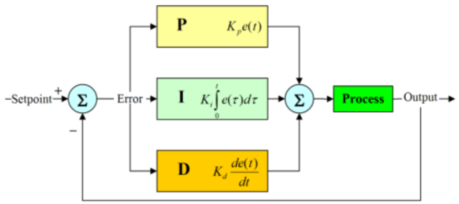
Adding up more "slices" of a function f(x) by taking smaller **Δx** steps gives better answer.

Calculus with Jack and Jill



Hill() is the Integral Function of HillSlope()!

What is PID?



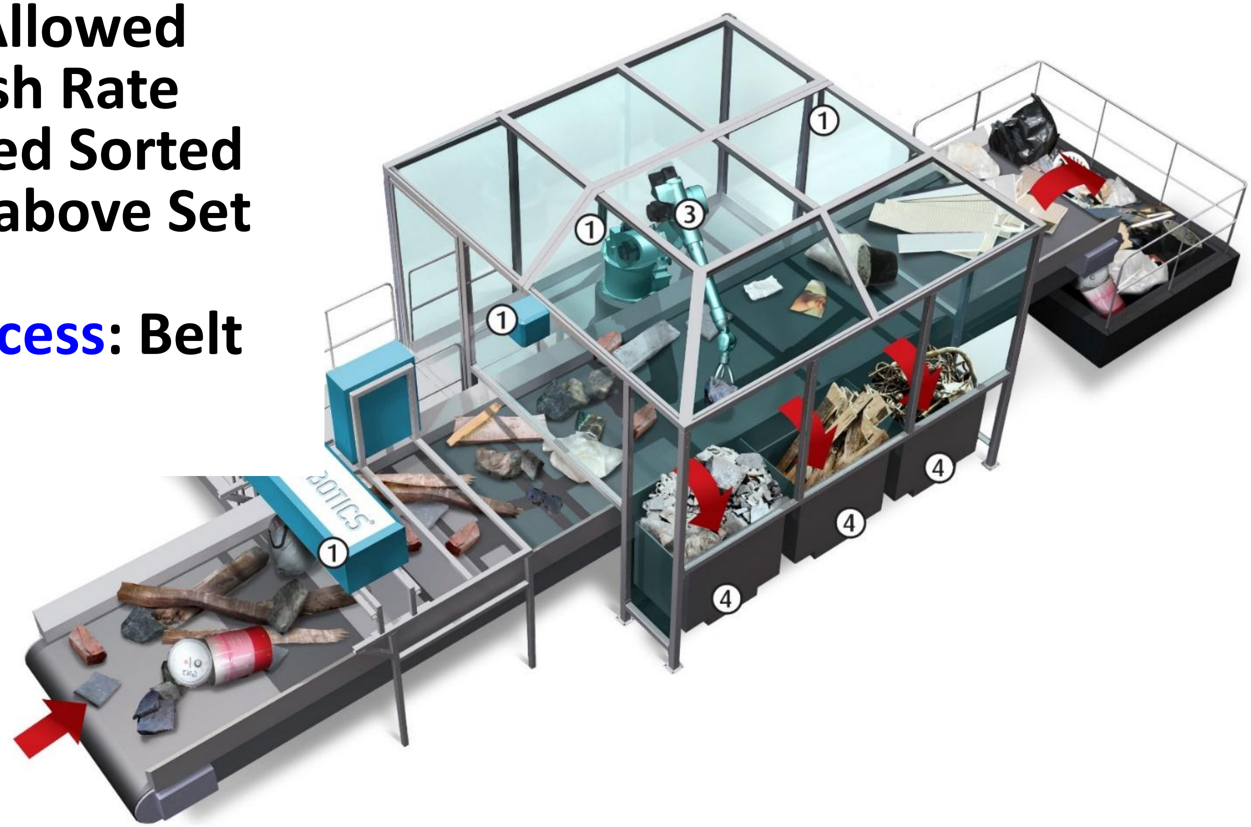
PID stands for **Proportional**, **Integral**, and **Differential** (P-I-D). It is the most common closed loop control method used to control real-world things like temperature and cruise control speed of a car.

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

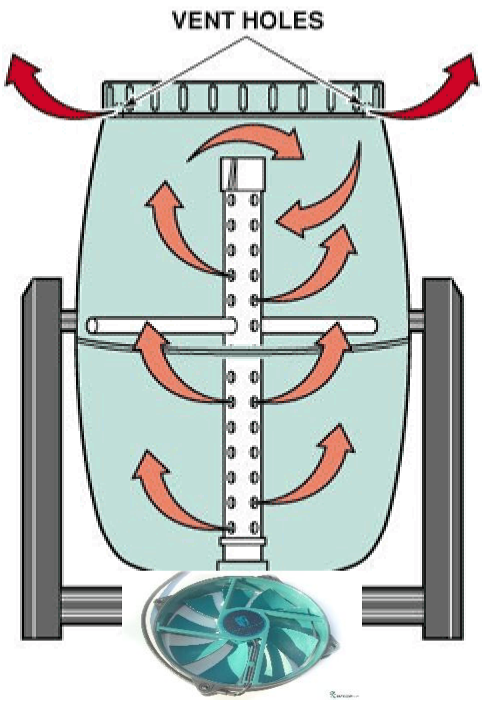
A Trash Sorter Conveyor Belt Speed

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



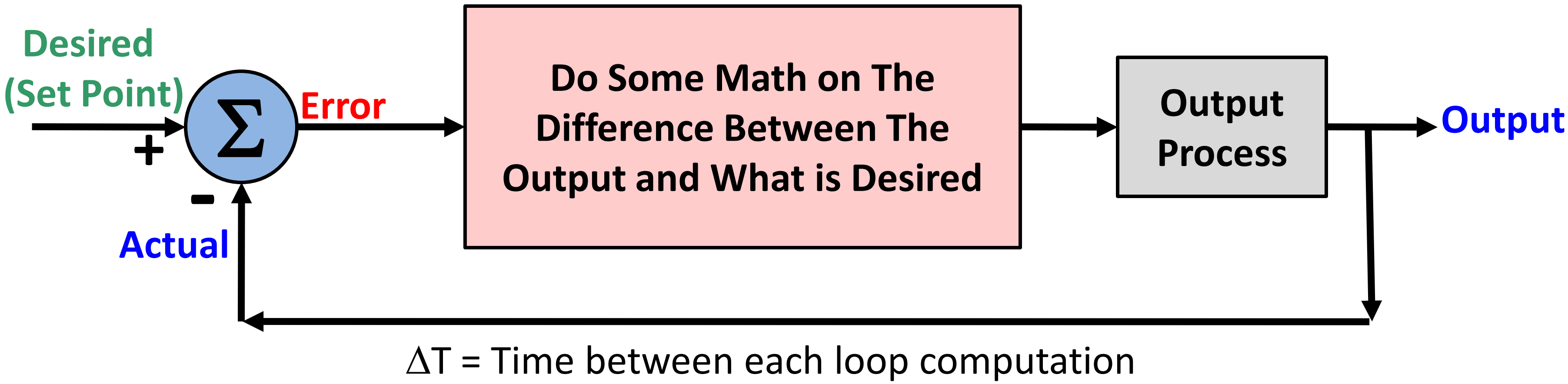
Another Example of PID Control

An Aerator fan for composting



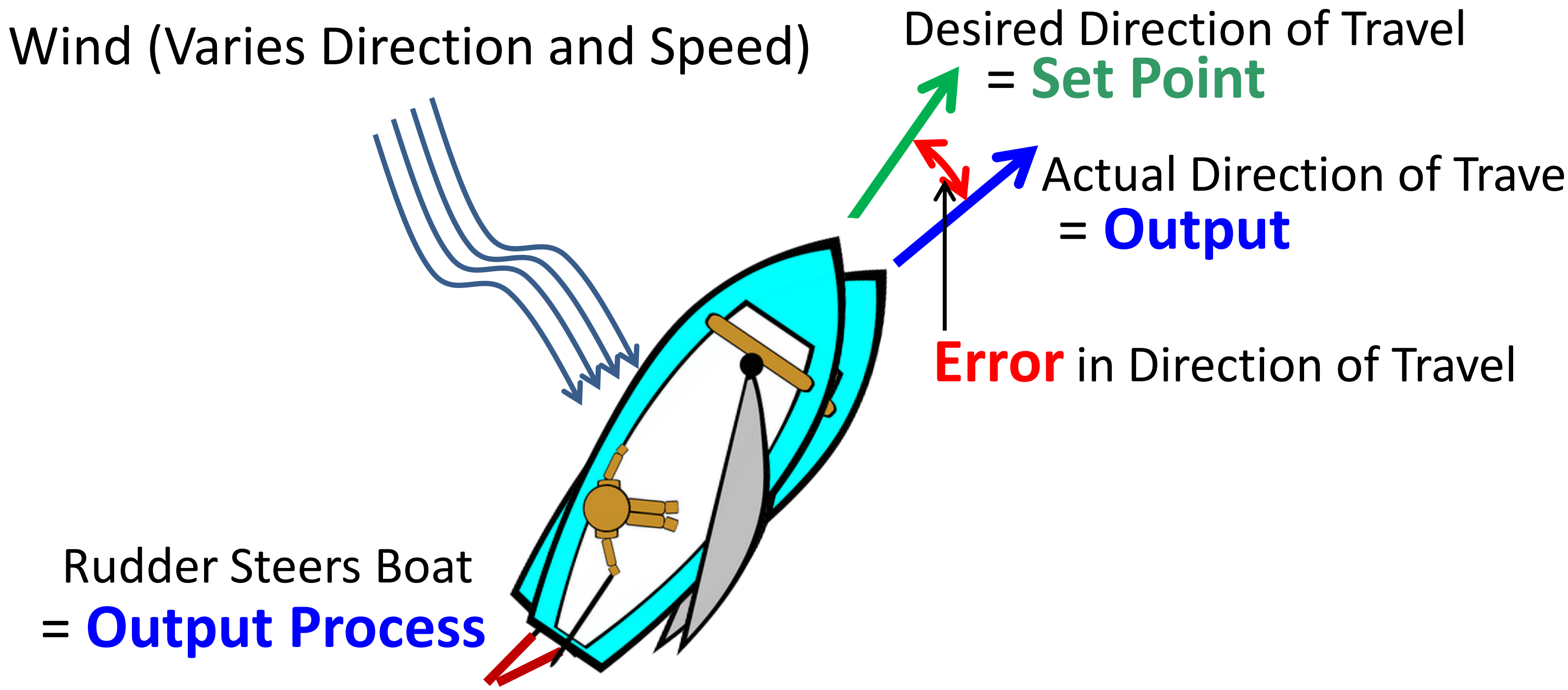
- **Output: Temperature of Compost**
- **Set Point: Ideal Compost Temperature**
- **Error: Difference in Temperature of Compost**
- **Output Process: Aerator Fan Speed**

A Feedback Control Loop



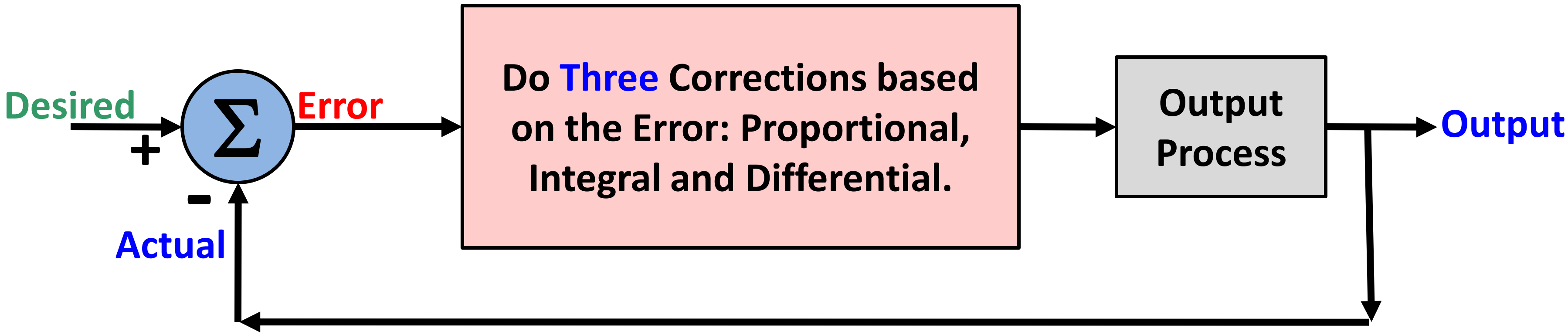
A Feedback Control Loop takes the actual output and compares it to the desired output (finding the error) at some loop rate and tries to make the actual output match the desired.

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

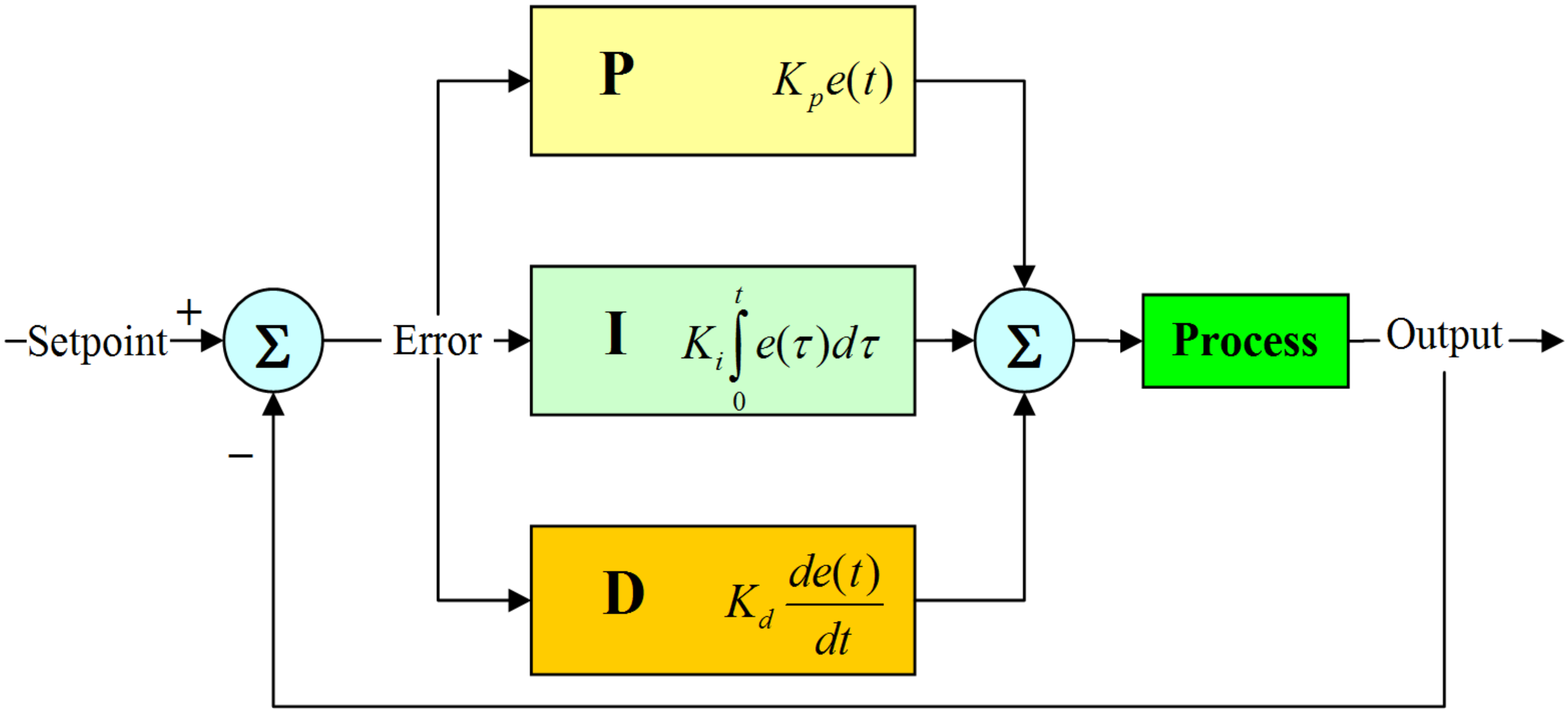


ΔT = Time between each loop computation

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

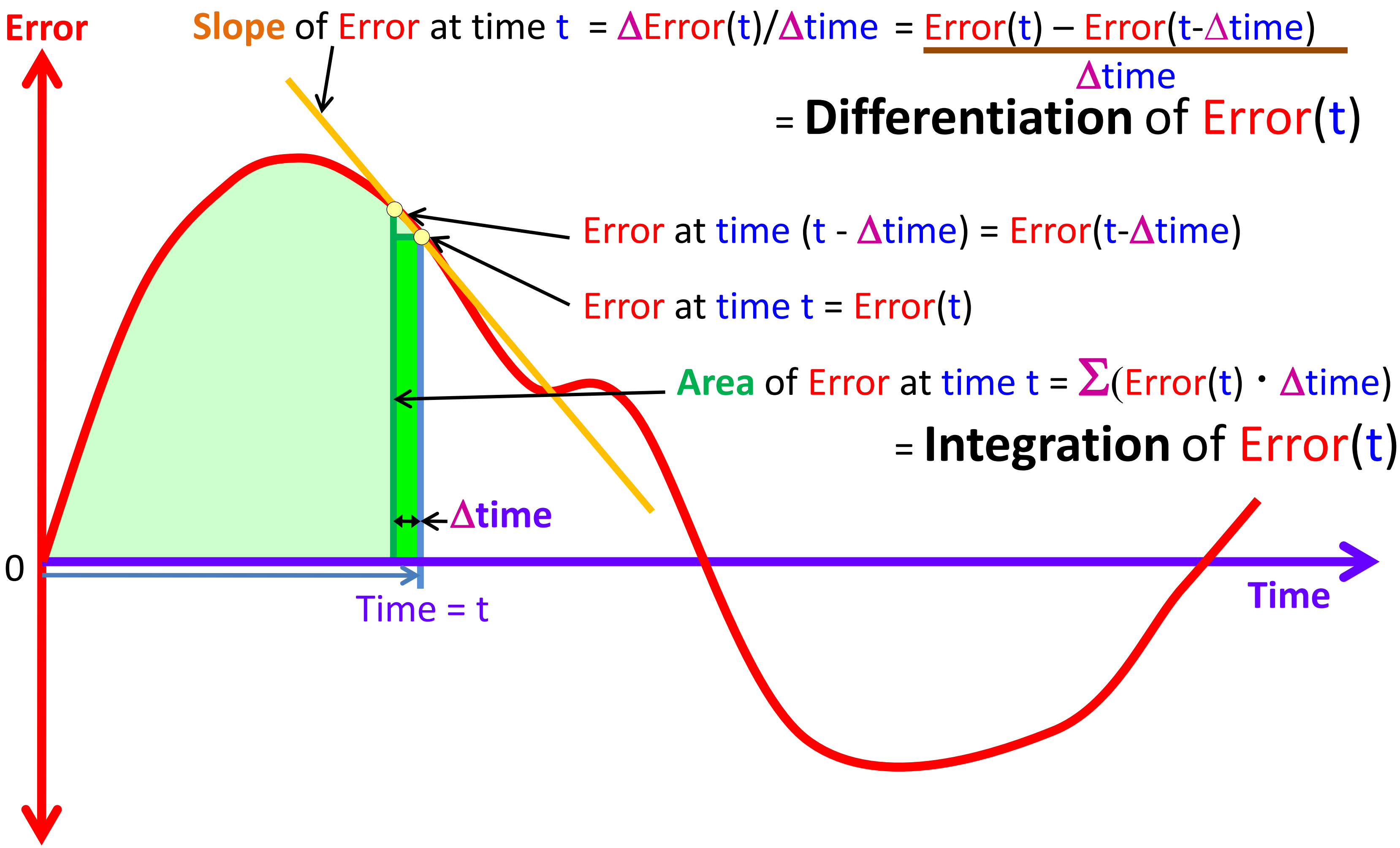
- **Proportional** term changes output proportional to error
- **Integral** term sums all previous error (error **area**) and compensates output
- **Differential** term affects output based on last and current error (**slope** of error)

The Generic PID Block Diagram



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

Graphic Representation of Error



The Proportional Term

Proportional

$= K_p \cdot Error$

K_p 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 Integral Term

Integral

$$= K_i \cdot \Sigma\,(Error \cdot \Delta T)$$

K_i is the Integral Gain constant. It defines how much the integral component of the PID controller affects the output.

Σ is a Greek symbol we use to mean summation. In this case we want to sum (add) all the errors since we started the PID Controller.

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 point.

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

The Differential Term

Differential

$$= K_d \cdot (\Delta \text{Error} / \Delta T)$$

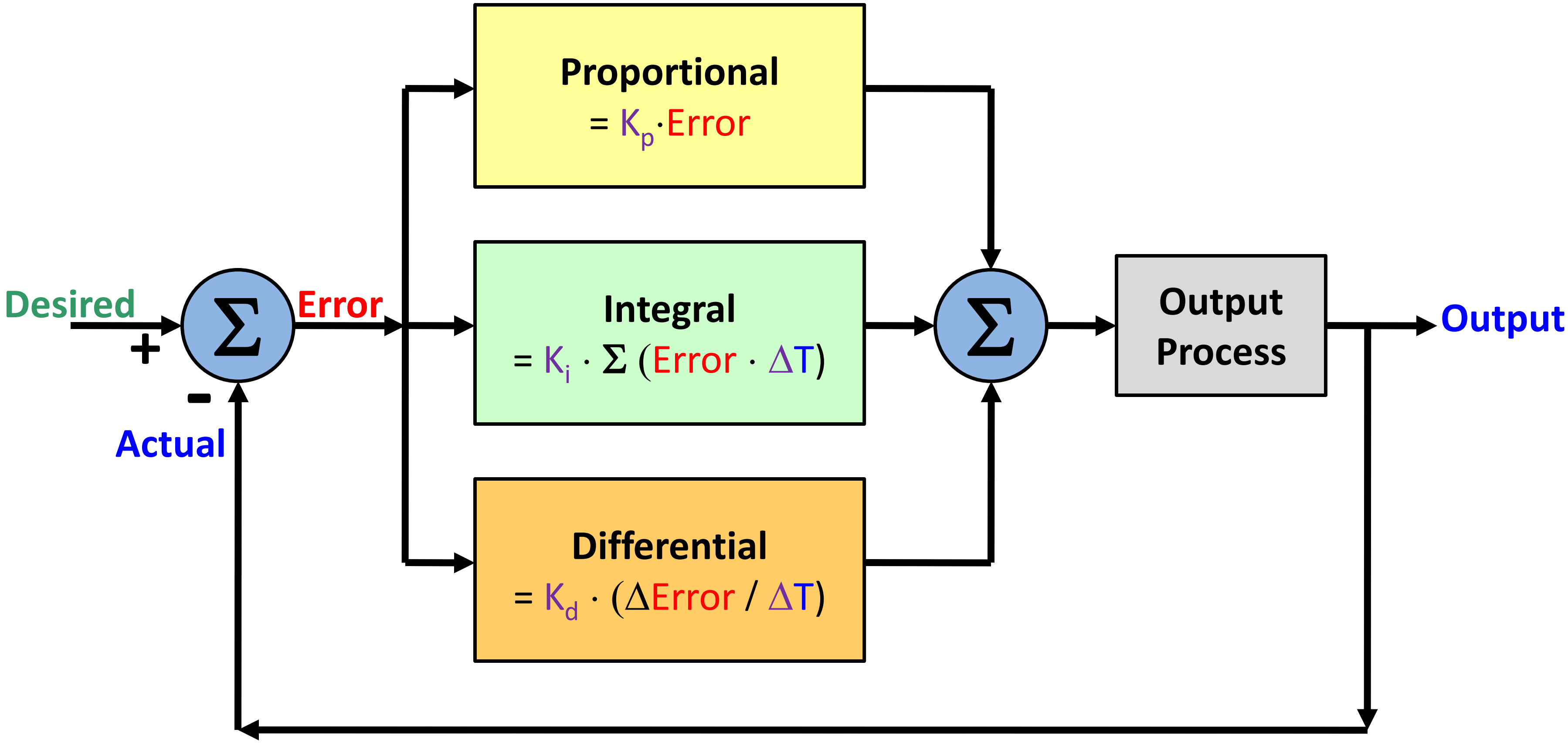
K_d is the Differential Gain. It defines how much the differential component of the PID controller affects the output.

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.

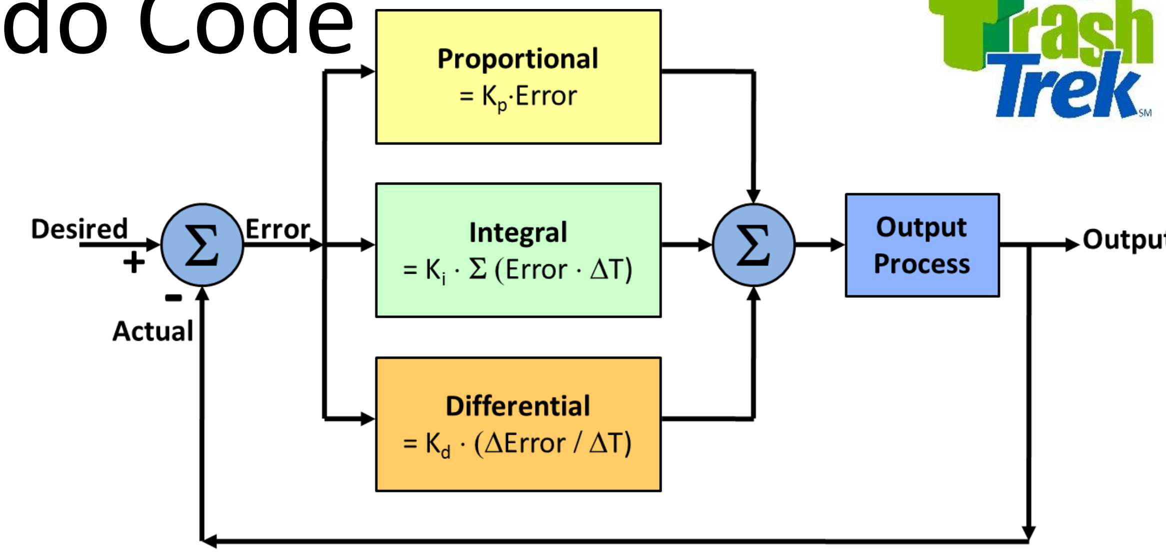
ΔT is the amount of time since the last time we performed this calculation. T is the letter we use to mean Time. So ΔT is the change in time since we did this computation last. It is our sampling time of the error.

Putting the PID together



ΔT = Time between each loop computation

PID Pseudo Code



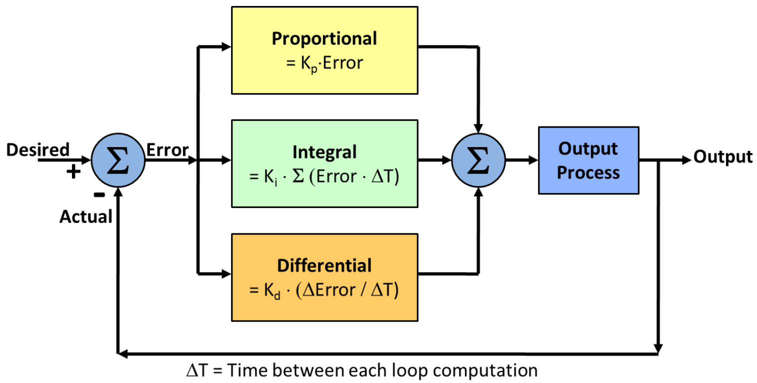
ΔT = Time between each loop computation

```
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
```

You Can't Tuna Fish…

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!**

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 (K_p, K_i, and K_d) for each of the three contributing components for a specific controller use. Finding these gains requires iterative (repeating) experiments with the controller. This is called "tuning". One common tuning method 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 K_i and K_d to zero. Thus, the controller becomes a P-type controller only.

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

Step 3: Measure the time period of the oscillation when $K_p = K_c$. This is called the critical oscillation period, P_c.

Step 4: Use the values K_c and P_c to determine K_p, K_i, and K_d using the formulas that Zeigler and Nichols determined for a P, PI, or PID controller (shown on next slide).

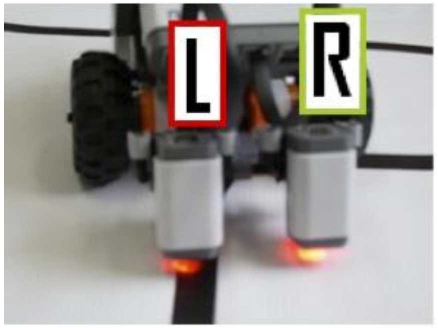
Ziegler-Nichols Method Gain Values

Control Type	K_p	K_i	K_d
P	$0.50K_c$	-	-
PI	$0.45K_c$	$1.2K_p / P_c$	-
PID	$0.60K_c$	$2.0K_p / P_c$	$K_pP_c / 8$

Effects of Increasing 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

Two Light Sensor Line Follower



For a two light sensor line following robot consider:

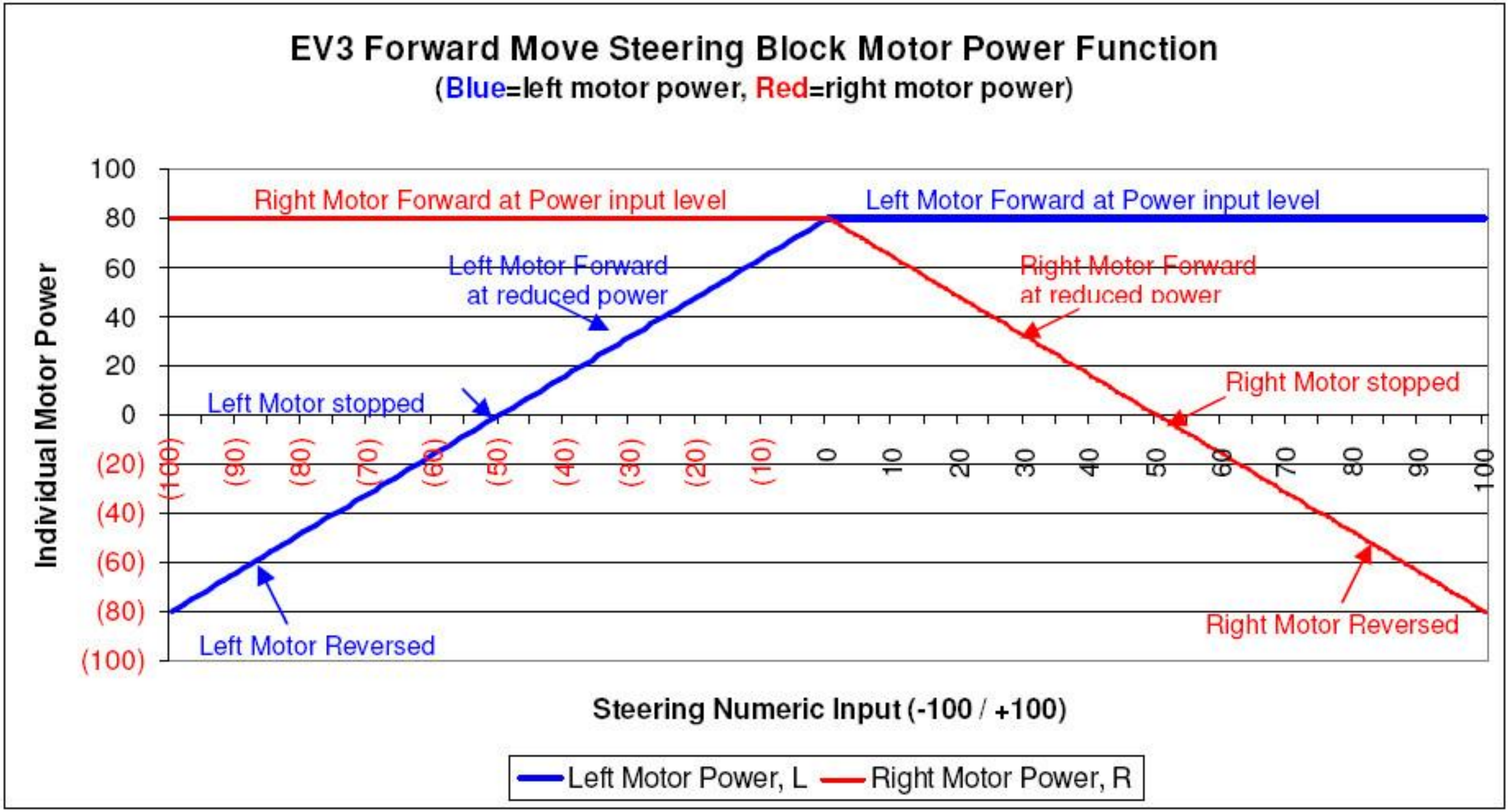
- What is **Desired** (Set Point)?
- What is the **Error**? The difference between the light values seen? What about differences in the sensors?
- How to **Steer** the robot? Use Steering Motor Block or separate Motor Power Blocks?
- What is your Sample Time (ΔT, i.e. you loop time)?
- What are your variables? What are your constants?

PID 2 Light Sensor Line Follower Pseudo Code

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

Loop:
  Get Left_Light_Value
  Get Right_Light_Value
  Error = Left_Light_Value – Right_Light_Value – Light_Sensor_Correction
  Integral = Integral + (Error * DeltaT)
  Derivative = (Error - Previous_Error)/DeltaT
  Previous_Error = Error
  Steering = (Kp*Error) + (Ki*Integral) + (Kd*Derivative)
  Wait(deltaT)
Repeat Loop
```

EV3 Move Steering Block Power



References

- http://www.mathisfun.com/
- http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html
- http://thetechnicgear.com/2014/03/howto-create-line-following-robot-using-mindstorms/
- https://en.wikipedia.org/wiki/PID_controller
- http://controlguru.com/table-of-contents/
- http://www.wired.co.uk/magazine/archive/2013/06/start/a-smarter-way-to-sort-your-recycling
- http://www.urban-composting.com/