Carnegie Mellon ROBOTICS CLUB

Control Lab

Spring 2026

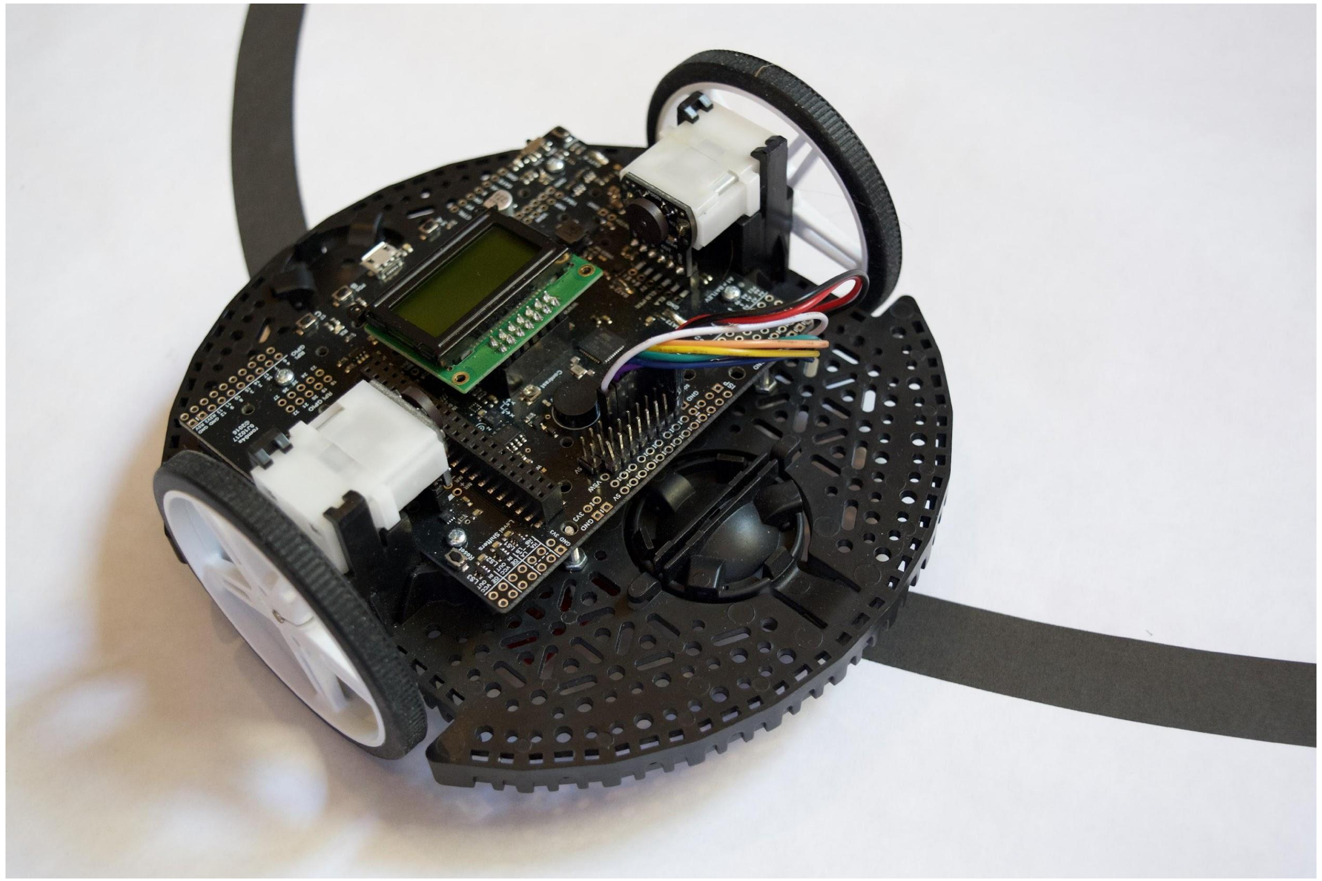
Overview

- Lab Goals
- Reflectance Sensors
- Control Intro
- Proportional Control
- Proportional/Derivative Control
- Advanced Control

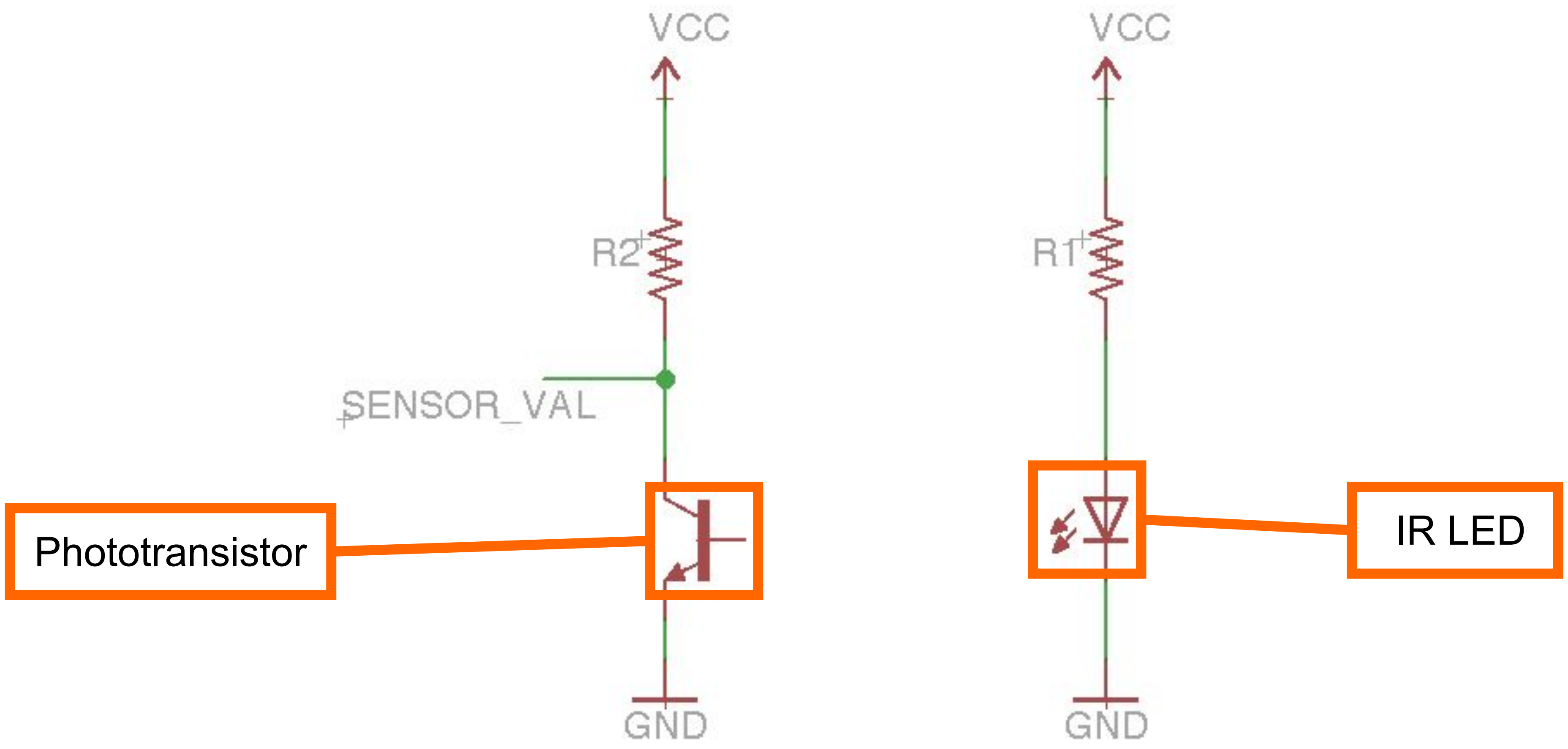
Carnegie Mellon ROBOTICS CLUB

Lab Challenge

- You need to program your robot to follow our line
- The robot needs to complete 2 laps without leaving the line
- 45 sec for two laps



Reflectance Sensors



Carnegie Mellon ROBOTICS CLUB

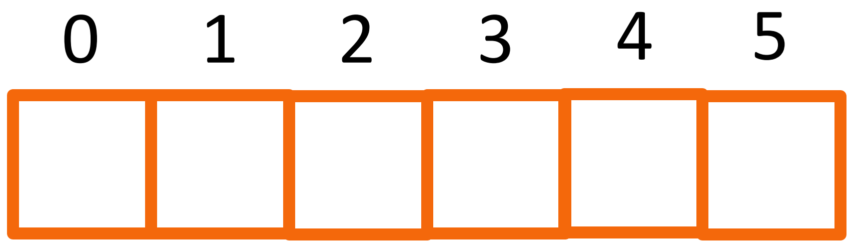
Reflectance Sensors

readLine function

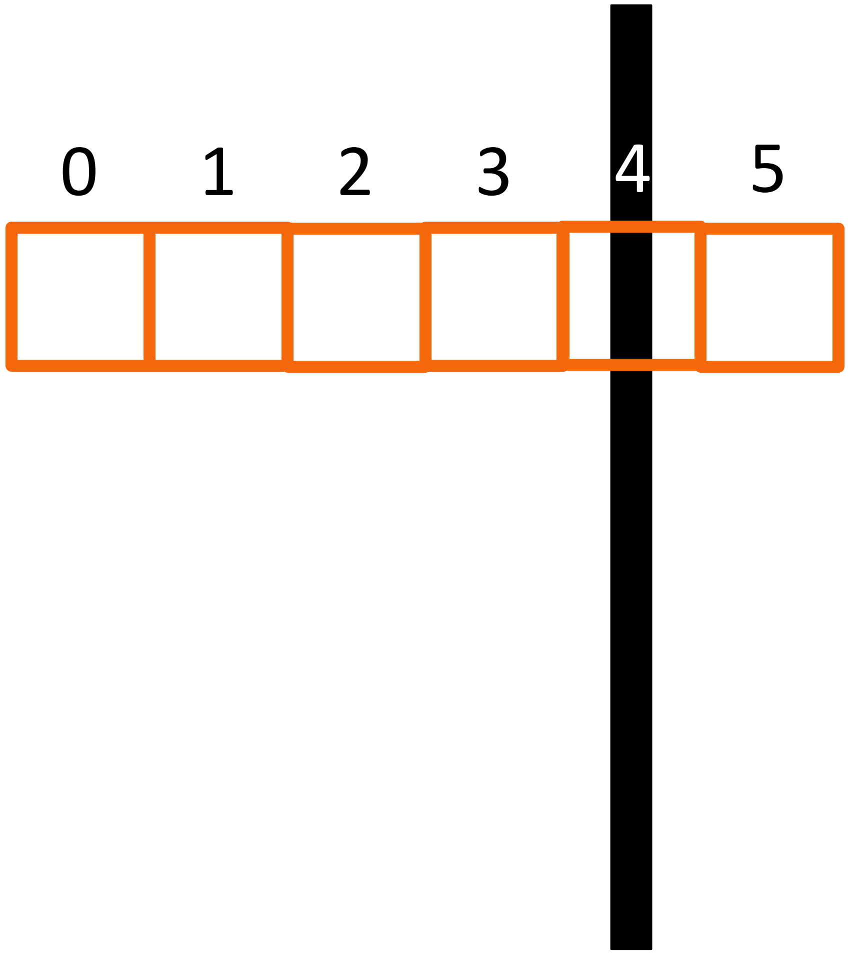
- To figure out where the line is, use:
  `qtrrc.`**`readLine`**`(sensors);`
- This will read all the sensors and figure out where the line is

Carnegie Mellon ROBOTICS CLUB

How readLine Works



How readLine Works

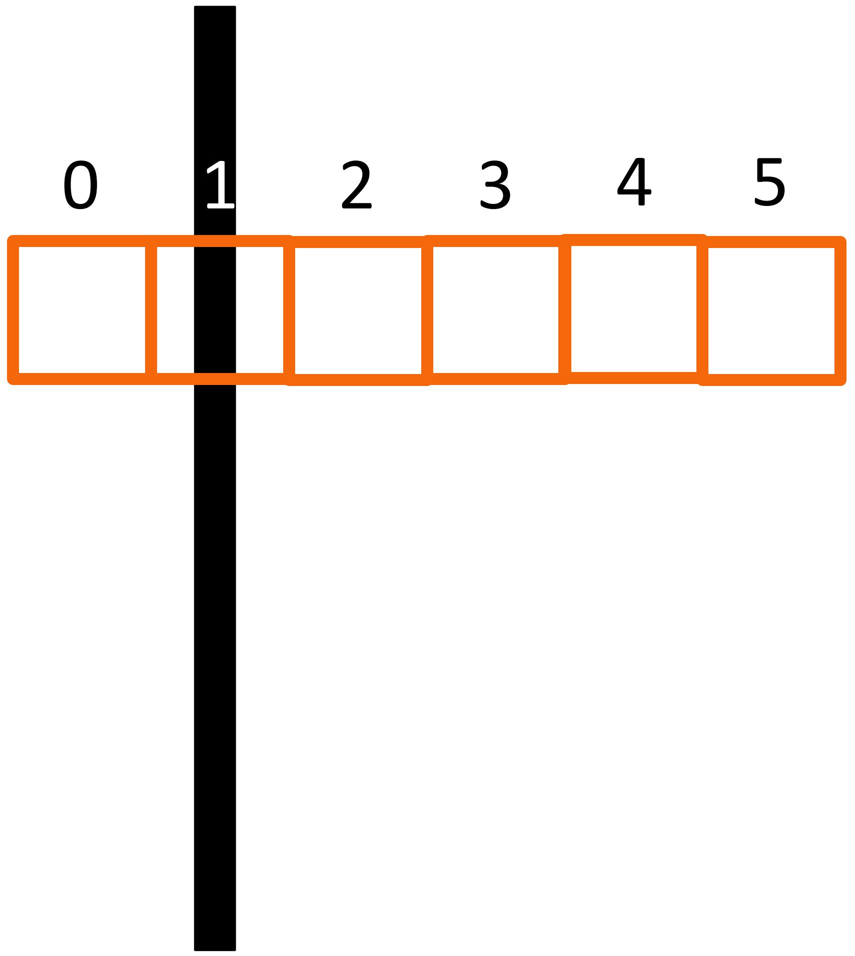


Output:

~4000

Carnegie Mellon ROBOTICS CLUB

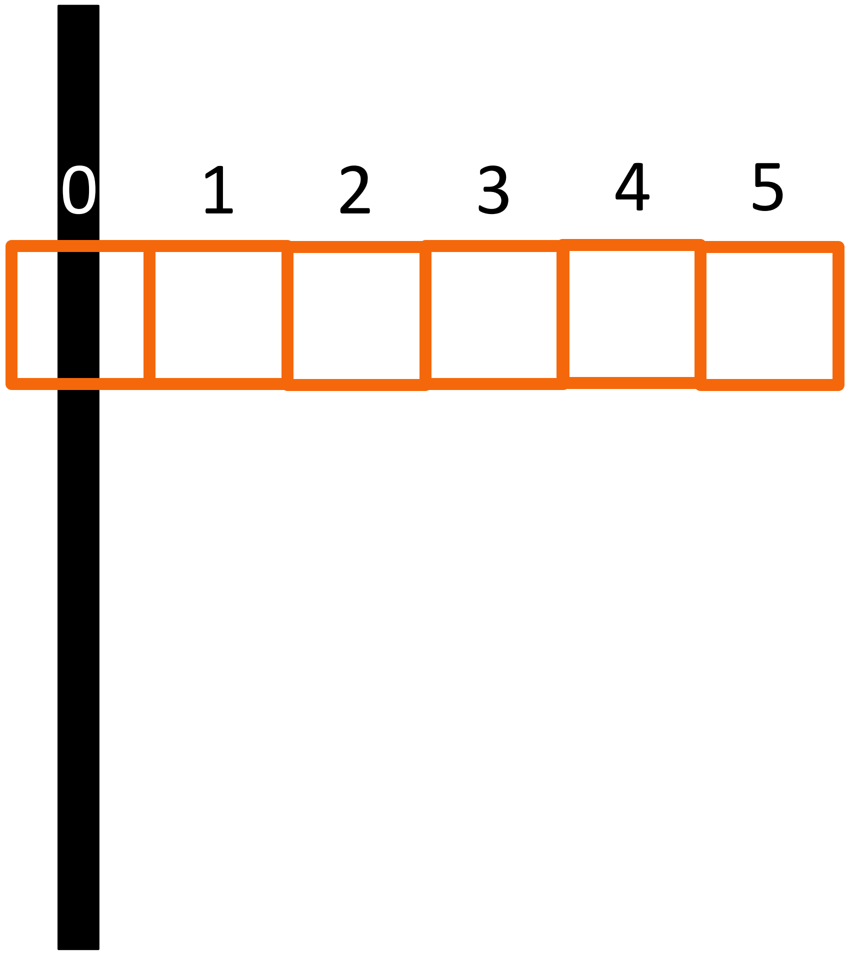
How readLine Works



Output:

~1000

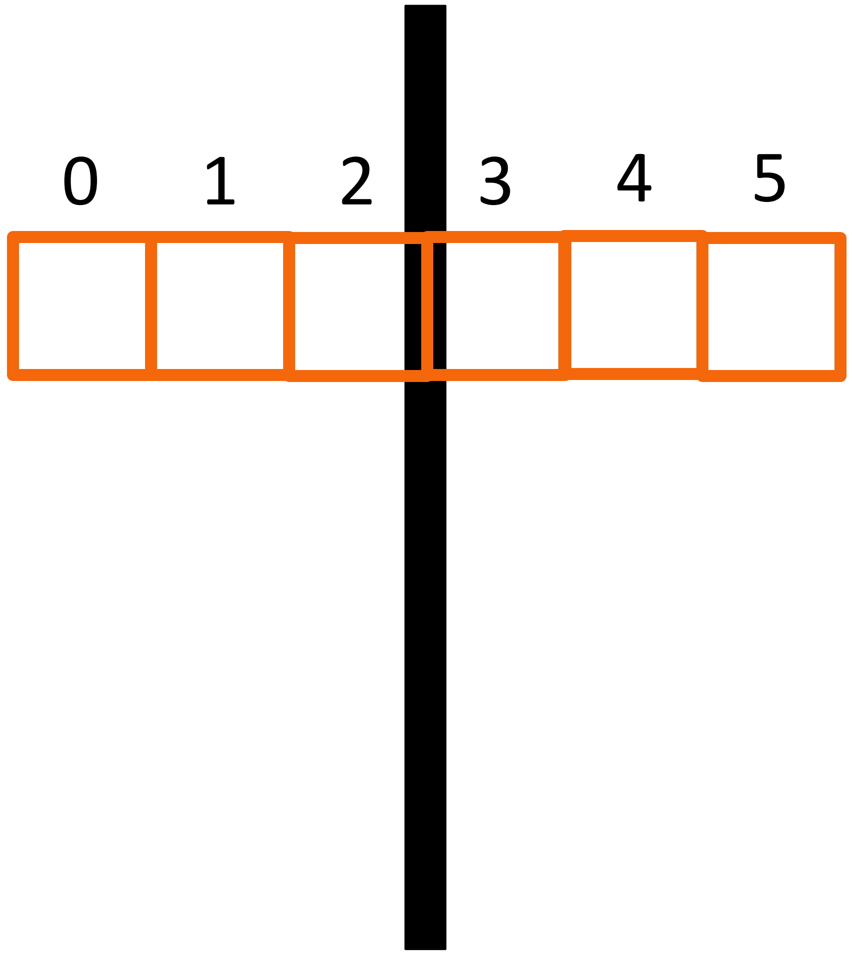
How readLine Works



Output: ~0

Carnegie Mellon ROBOTICS CLUB

How readLine Works

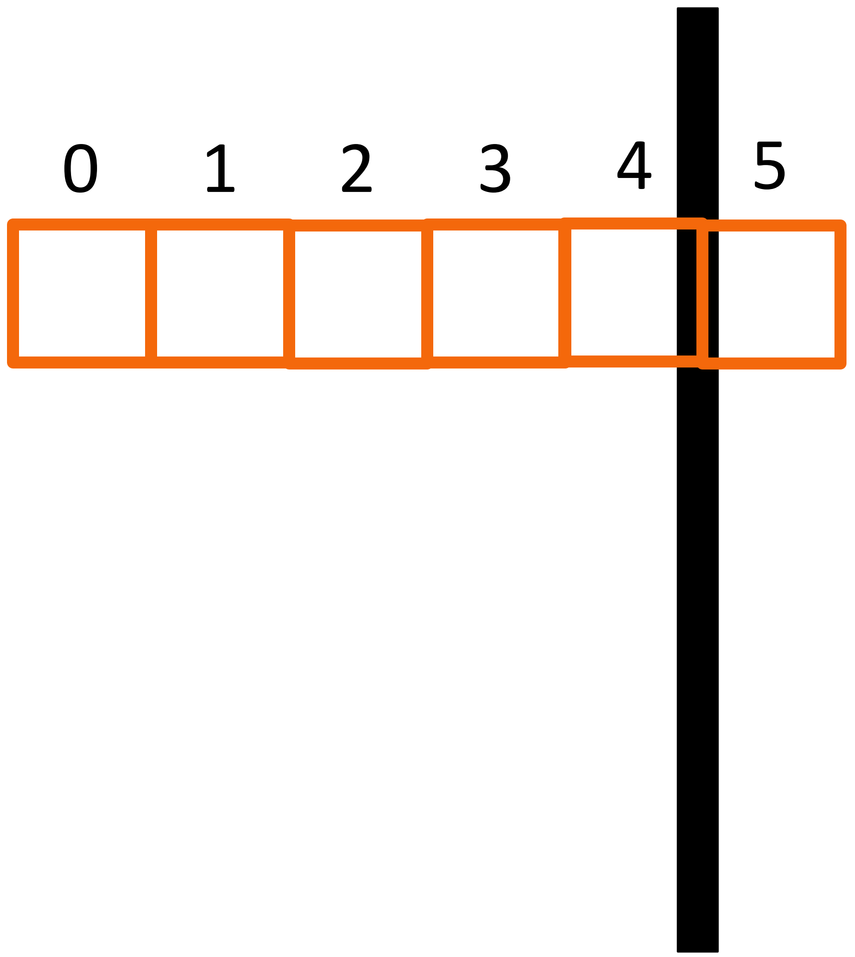


Output:

~2500

Carnegie Mellon ROBOTICS CLUB

How readLine Works

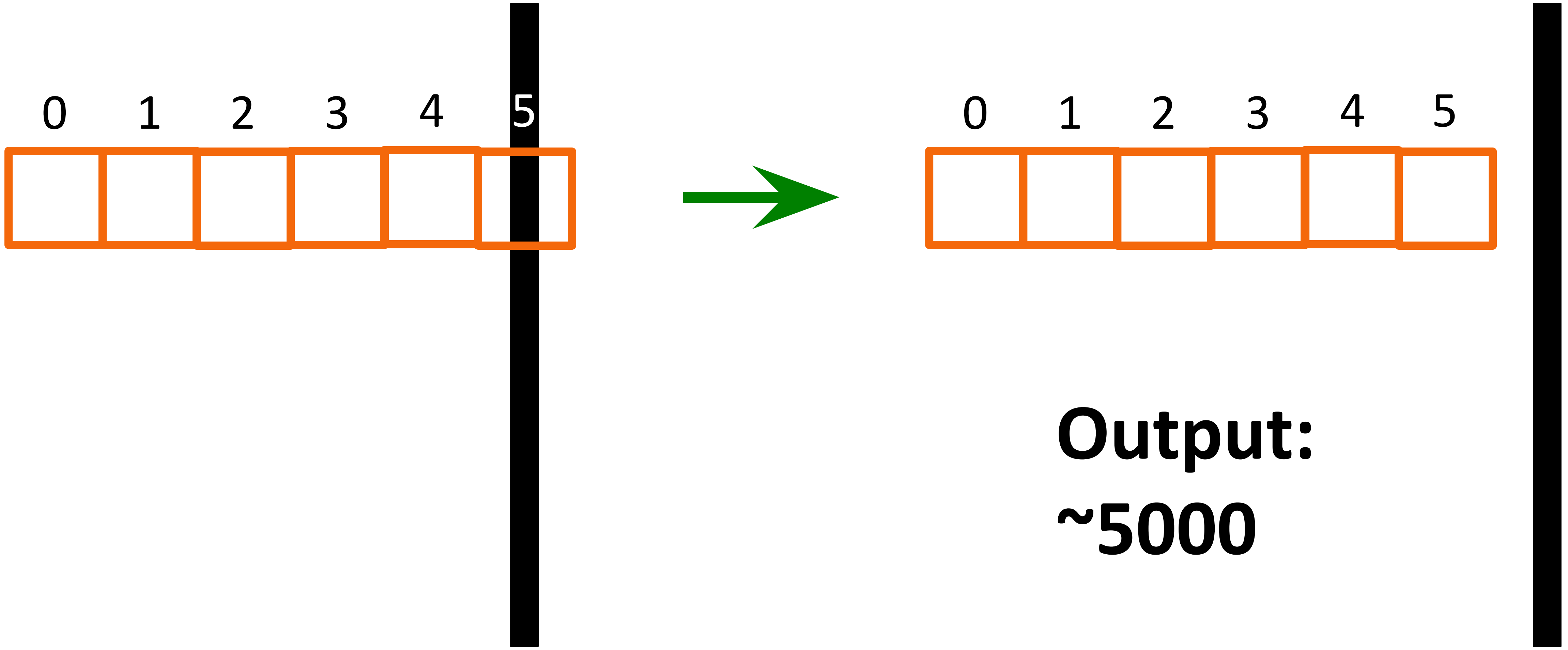


Output:

~4500

Carnegie Mellon ROBOTICS CLUB

How readLine Works



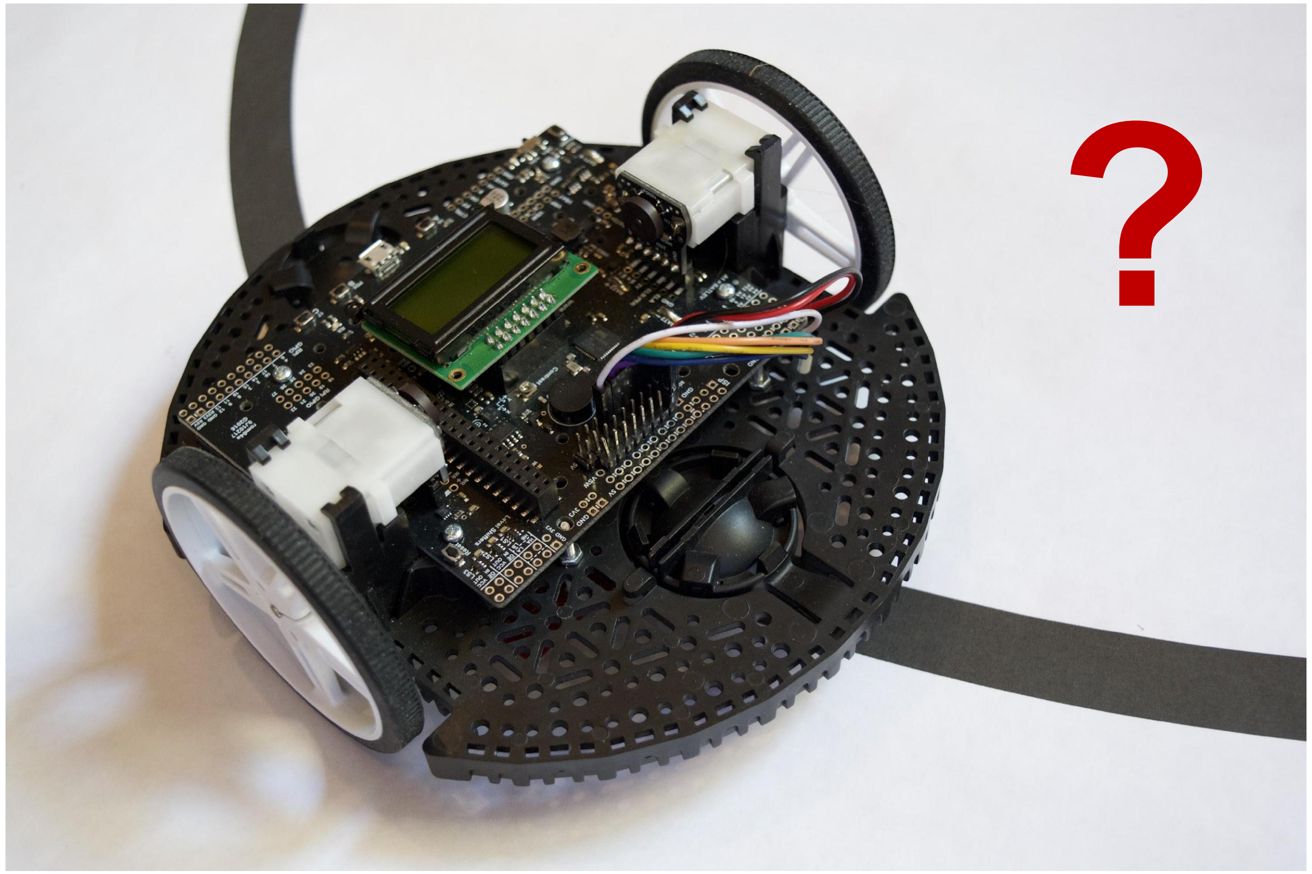
Output:

~5000

Carnegie Mellon ROBOTICS CLUB

Robot Control

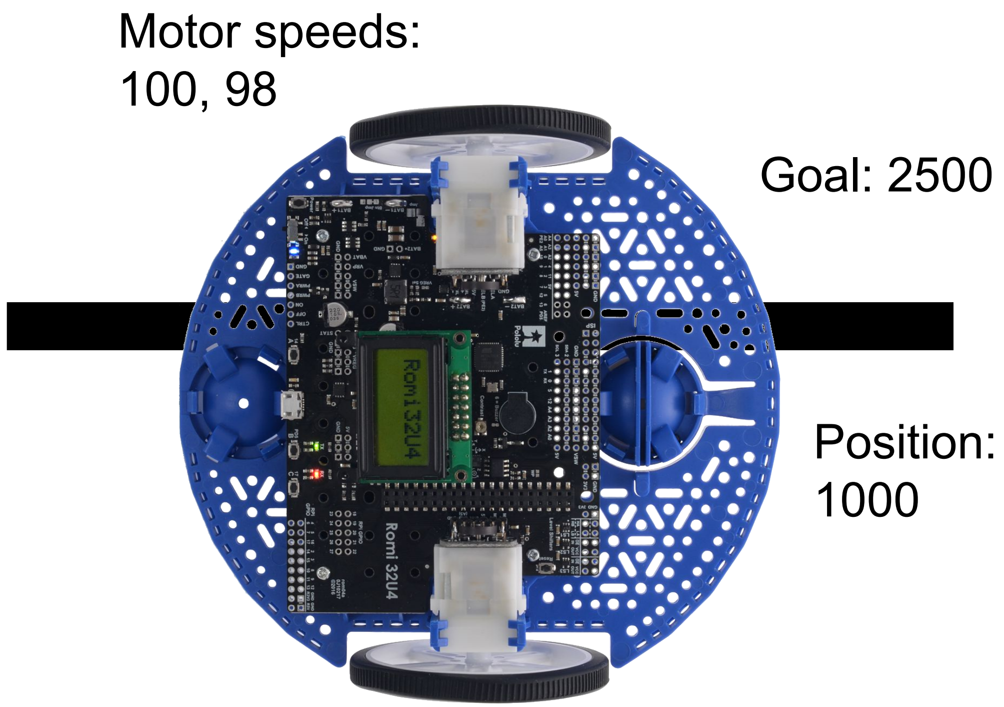
- We know where the line is, but how do we follow it?



Carnegie Mellon ROBOTICS CLUB

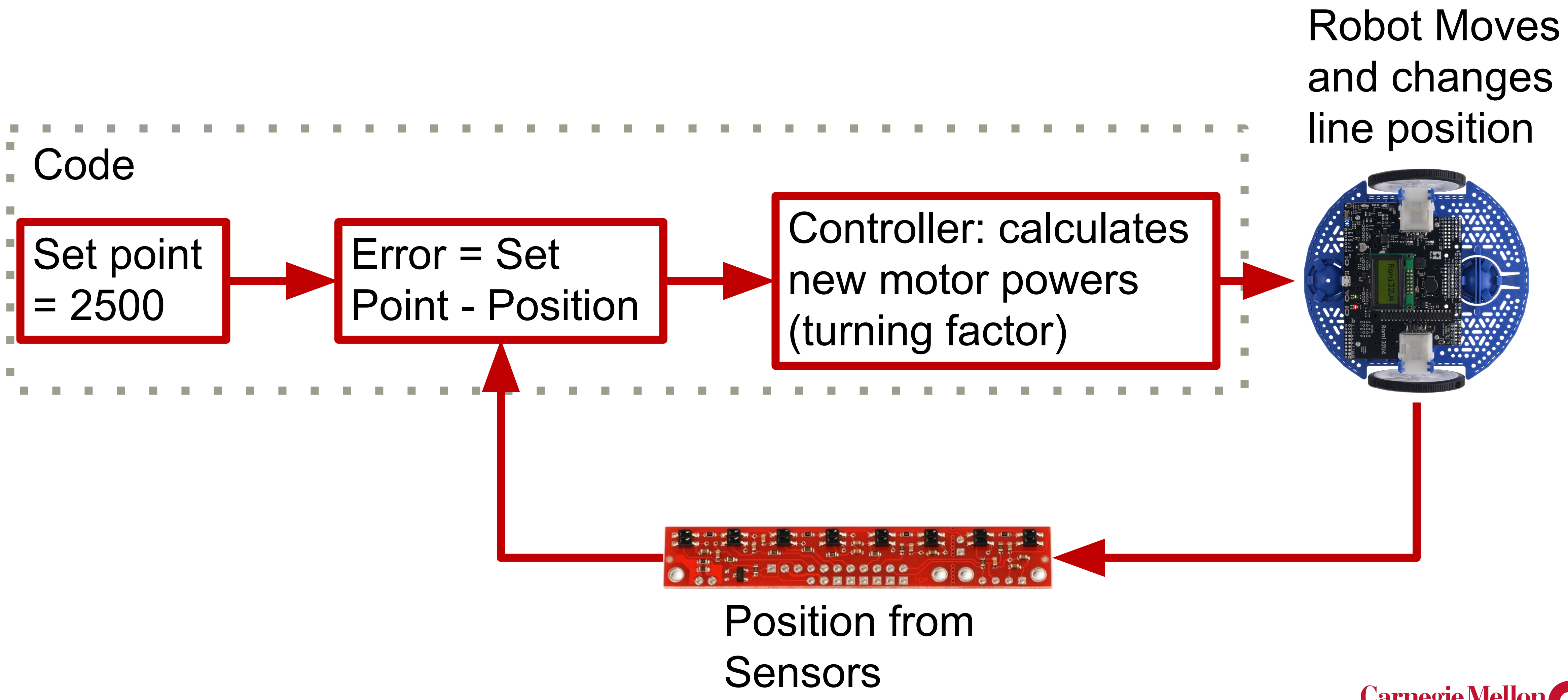
The Control Problem

- Knowns
  - Current position of line
  - Where we want the line to be
- Unknown
  - What motor values will get us to our goal



Carnegie Mellon ROBOTICS CLUB

The Control System


Robot Moves and changes line position
Code
Set point = 2500
Error = Set Point - Position
Controller: calculates new motor powers (turning factor)
Position from Sensors

Candy Question!

- What are some real-life problems that would require control theory and sensors to control autonomously?
    - Example: Balancing on a beam

Carnegie Mellon ROBOTICS CLUB

Control Systems in the Wild

- What are similar problems in real life:
  - Maintaining temperature in a room
  - Move a robotic arm to a specific angle (and keep it there)
  - Keeping a car in its lane (Very similar to line following)
  - Landing a rocket
- Real World Controllers
  - Bang-Bang controller
  - Flyball Governor
  - PID controller

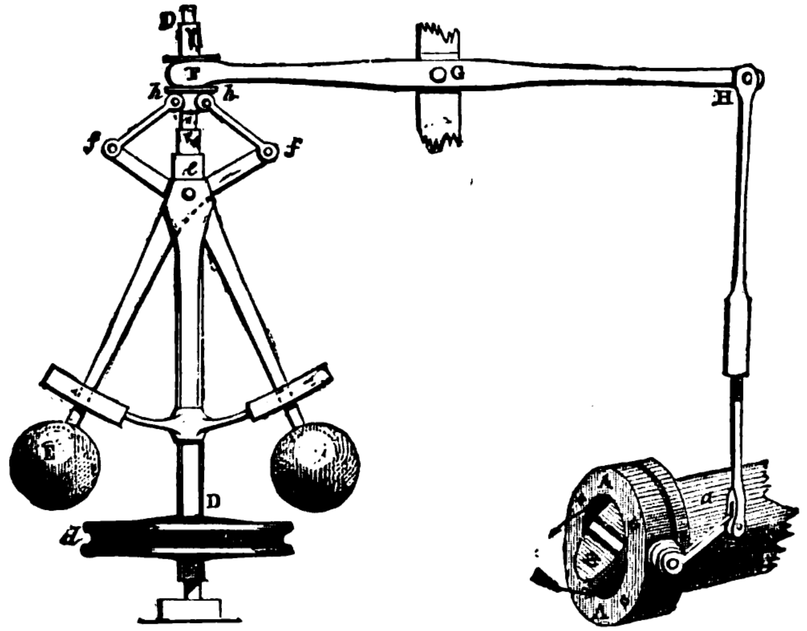
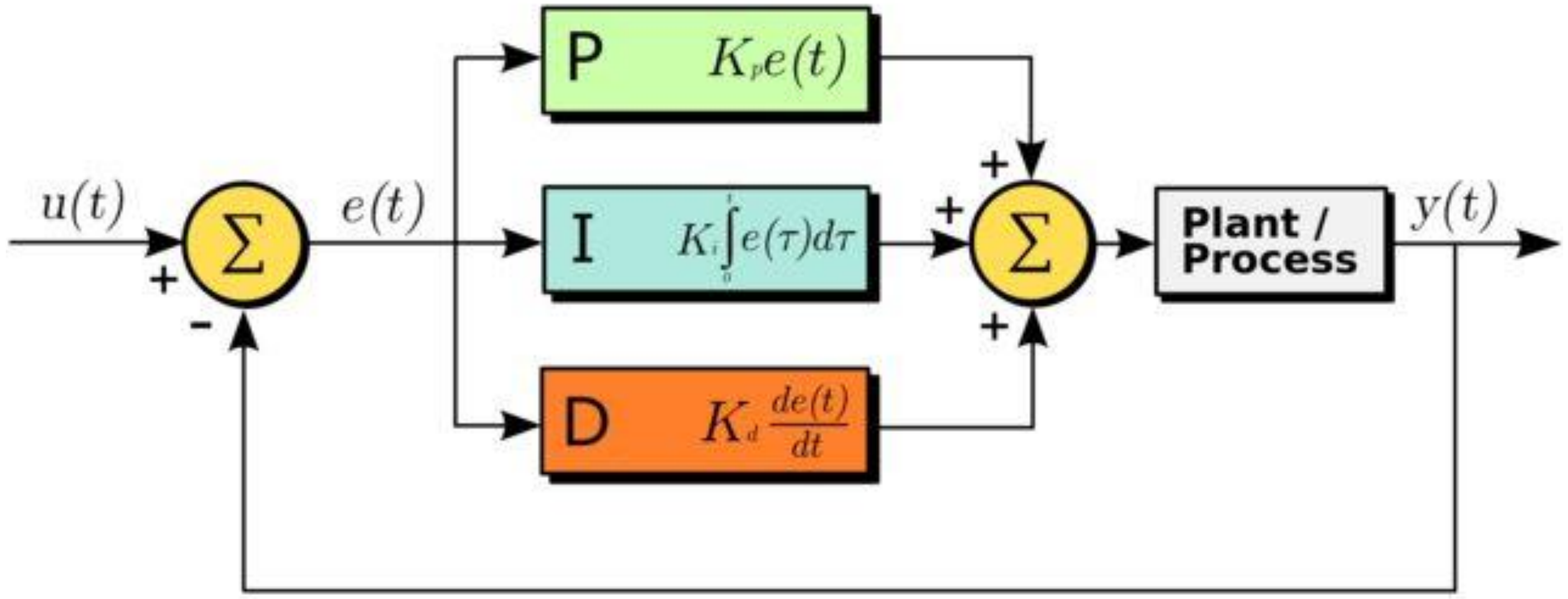


FIG. 4.—*Governor and Throttle-Valve.*

PID Control

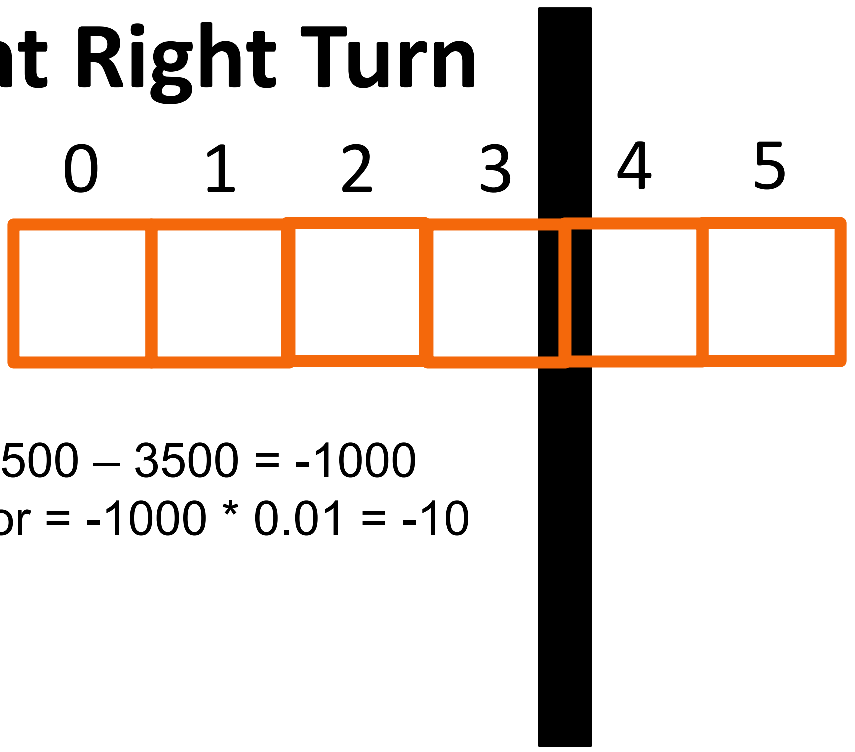
- Output = Proportional * Kp + Integral * Ki + Derivative * Kd



Proportional Control

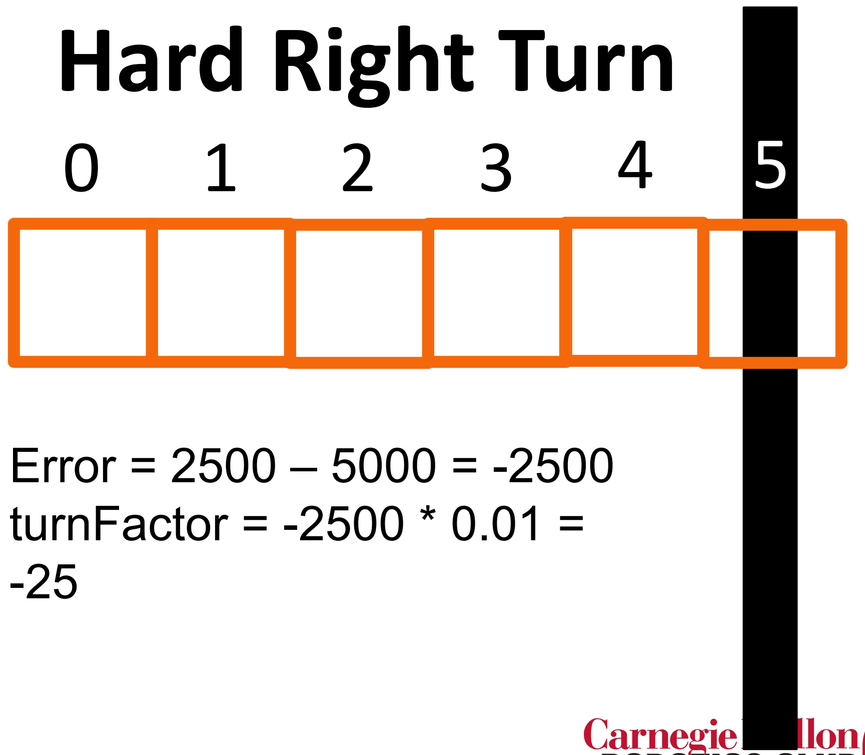
- Change how strongly you turn by how far off you are from your goal

Slight Right Turn



Error = 2500 – 3500 = -1000
turnFactor = -1000 * 0.01 = -10

Hard Right Turn



Error = 2500 – 5000 = -2500
turnFactor = -2500 * 0.01 =
-25

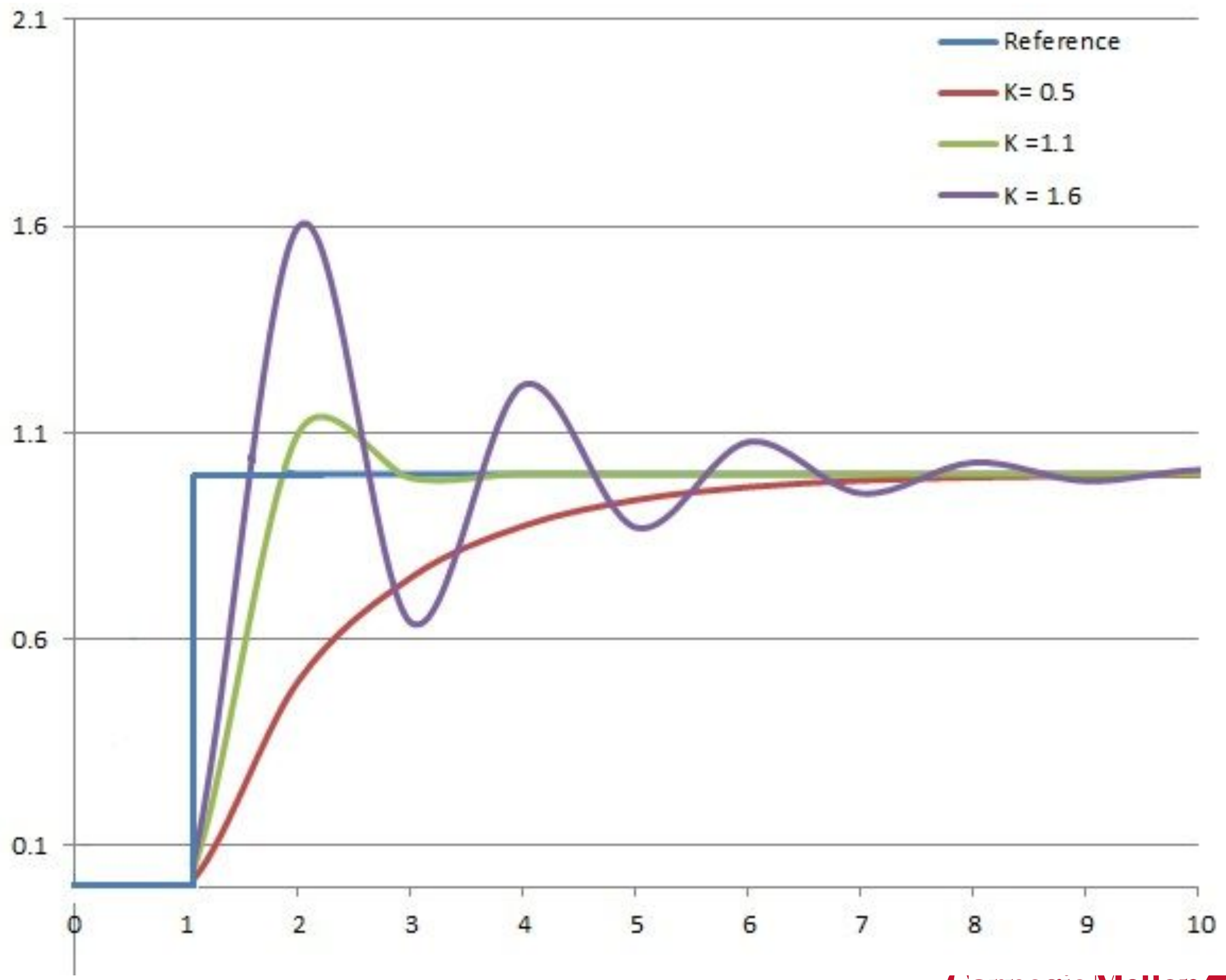
Carnegie Mellon ROBOTICS CLUB

Proportional Control

- The constant factor - Kp
- Different systems need different Kp's based on the system's sensors and actuators as well as the environment
- Goal: Minimize time to correct an error and overshoot
  - Too small and you never reach the goal
  - Too big and you end up overshooting and then oscillate
- Tuning proportional control

Candy Question

What is the best Kp on the graph?



Carnegie Mellon ROBOTICS CLUB

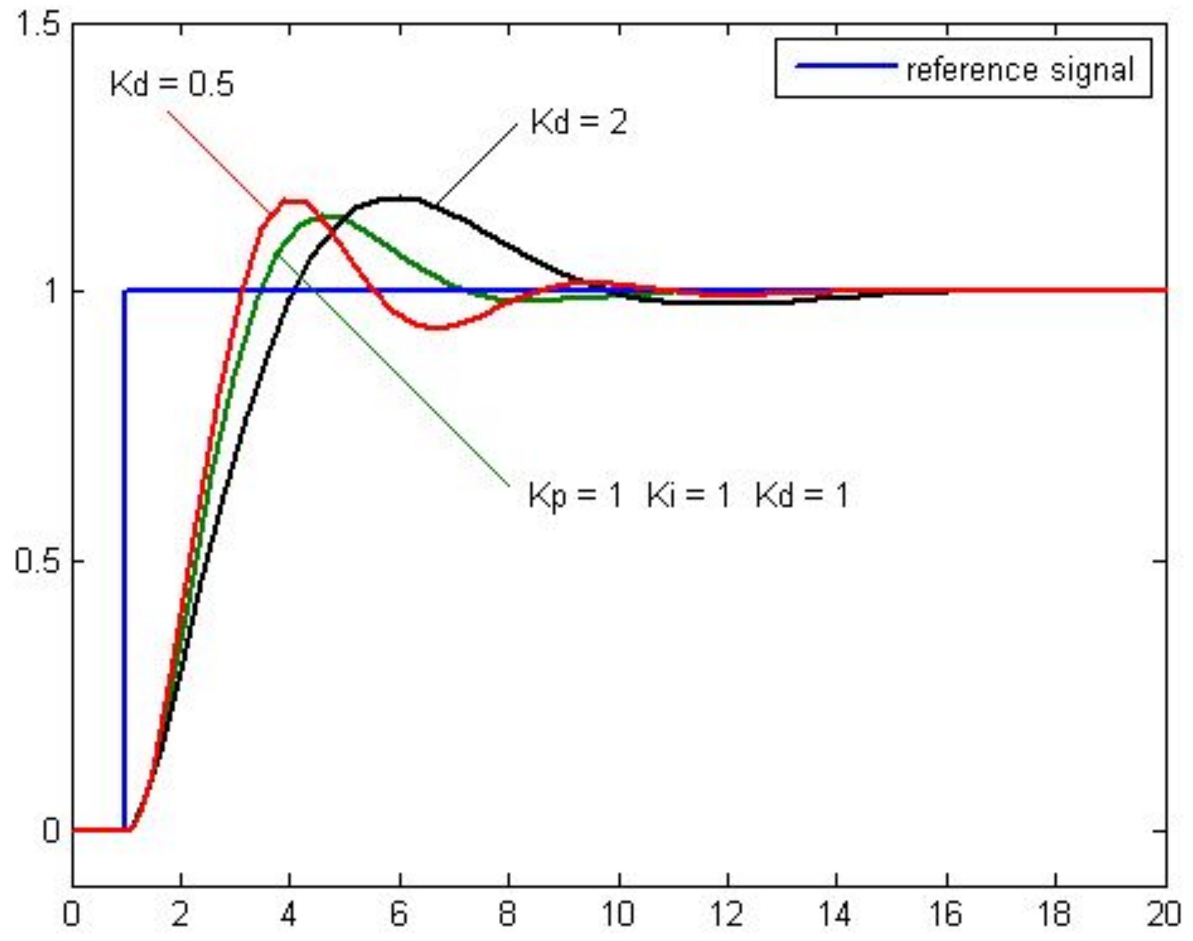
Derivative Control

- Derivative is the D in PID control
- Proportional on how fast the error is changing (the error's derivative)
- Slow down extra if approaching set point really quickly
  - This helps prevent oscillation on tight turns
- Can amplify noise, so may need to filter error
  - More on this next week

Derivative Control

- Derivative is needed to smooth out the effects of the proportional movement: it's also called a damping term.
- When approaching an end state, we want to smoothly stop
- Too big of a kD causes the robot to slow down too much when reaching a goal.
- Too small of a kD causes the robot to still overshoot
- Optimally start tuning kD after kP is tuned (increase kP by a bit when implementing kD)

Graph!



1.5
1
0.5
0
0
2
4
6
8
10
12
14
16
18
20
reference signal
Kd = 0.5
Kd = 2
Kp = 1 Ki = 1 Kd = 1

Derivative Control

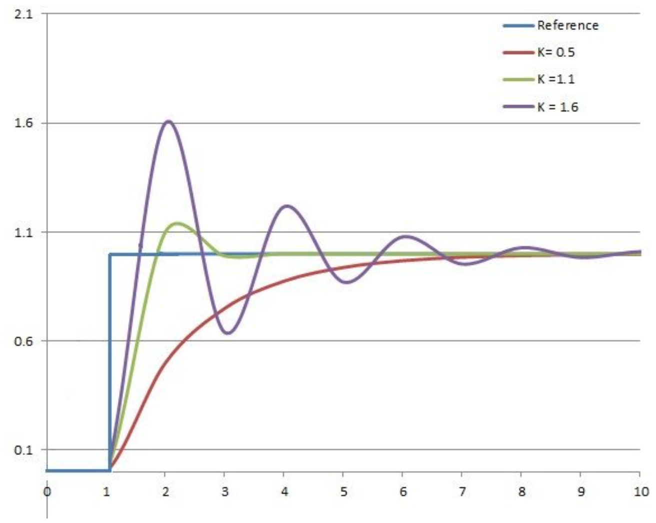
- Derivative of the error is multiplied by a constant, Kd
- Kd must be tuned
- To calculate the Derivative of the error:

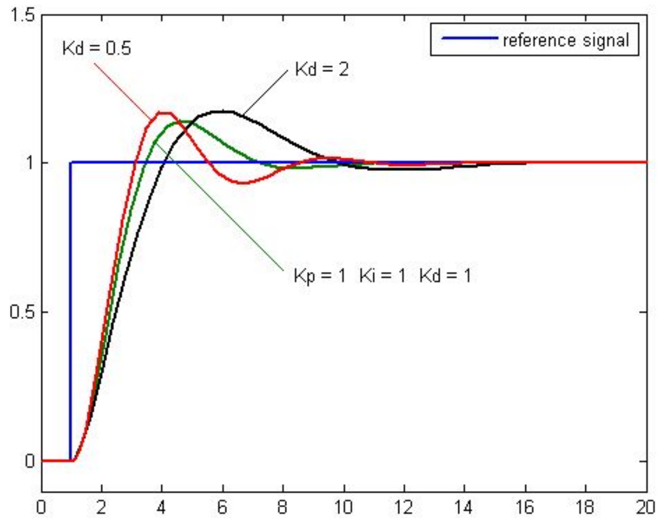
```
int old_err = 0;
void loop() {
    // calculate position and error
    int derivative = error - old_err;
    // calculations
    old_err = error;
}
```

Carnegie Mellon ROBOTICS CLUB

Candy Question

- Why do we need to increase kP after tuning it before introducing kD?
- If kD is too big, then the robot reaches its goal too slowly. What kP error would cause the same thing?





Carnegie Mellon ROBOTICS CLUB

Answer

As we mentioned before, kD will slow down the movement of the robot near the goal, which is going to influence the strength of kP, therefore we increase kP by a bit to compensate for this change.

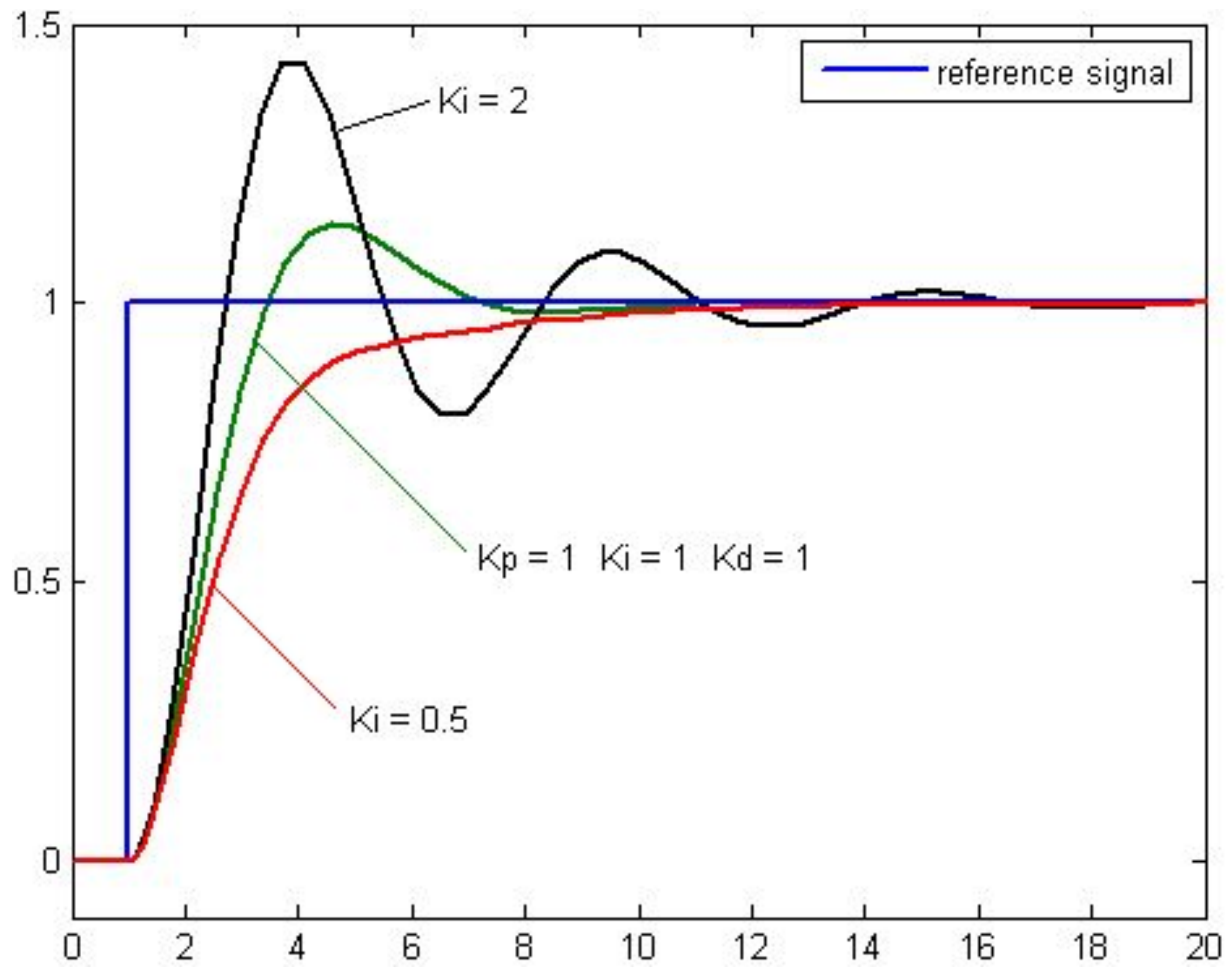
If kD is too big, the robot acts as if kP is too small. In other words, if kD is too big, kP is damped too much and becomes too small.

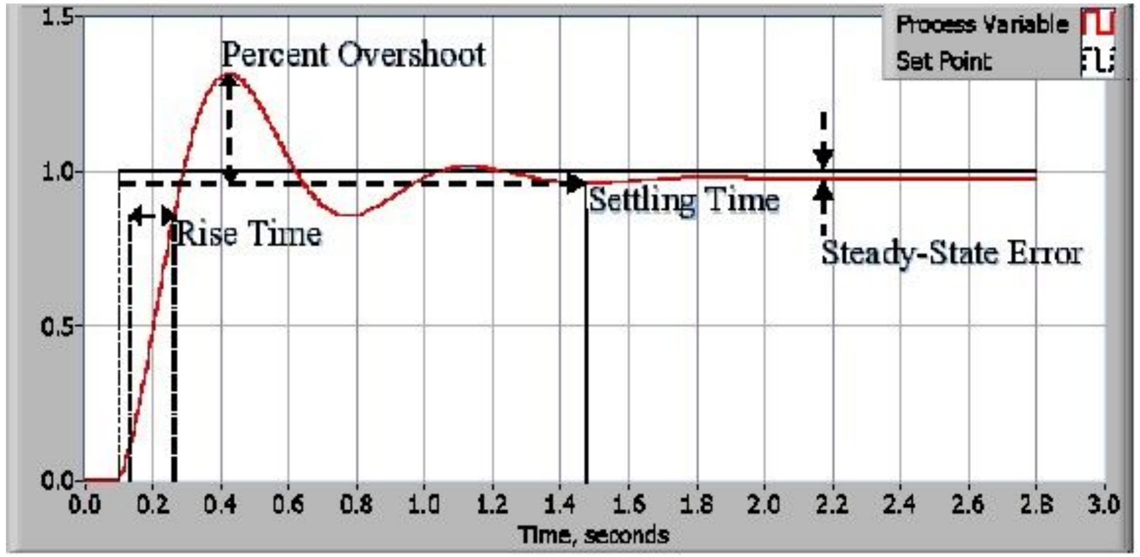
Carnegie Mellon ROBOTICS CLUB

Integral Control

- The I in PID
- The sum of the error over time (multiplied by Ki)
- Gets rid of “steady-state” error
    - <u>We don’t have this in line following</u>
- Can be tough to implement well
    - Can cause crazy oscillations when too high, often needs other terms that go along with it (such as integral delay or integral window)
- Needed when a physical obstacle occurs that is not related to the error

Graph!





Carnegie Mellon ROBOTICS CLUB

Candy Question!

- Why **<u>don't</u>** we need an integral term for line following?

Answer

- The integral term is mostly used when the robot (say, a robotic arm) is supposed to be at an angle of X degrees but can’t quite make it there every time due to factors like gravity acting on it.
- In line following, there is no final goal we are trying to reach and no steady state error. The line is constantly shifting so it would not really be useful to have an integral term.

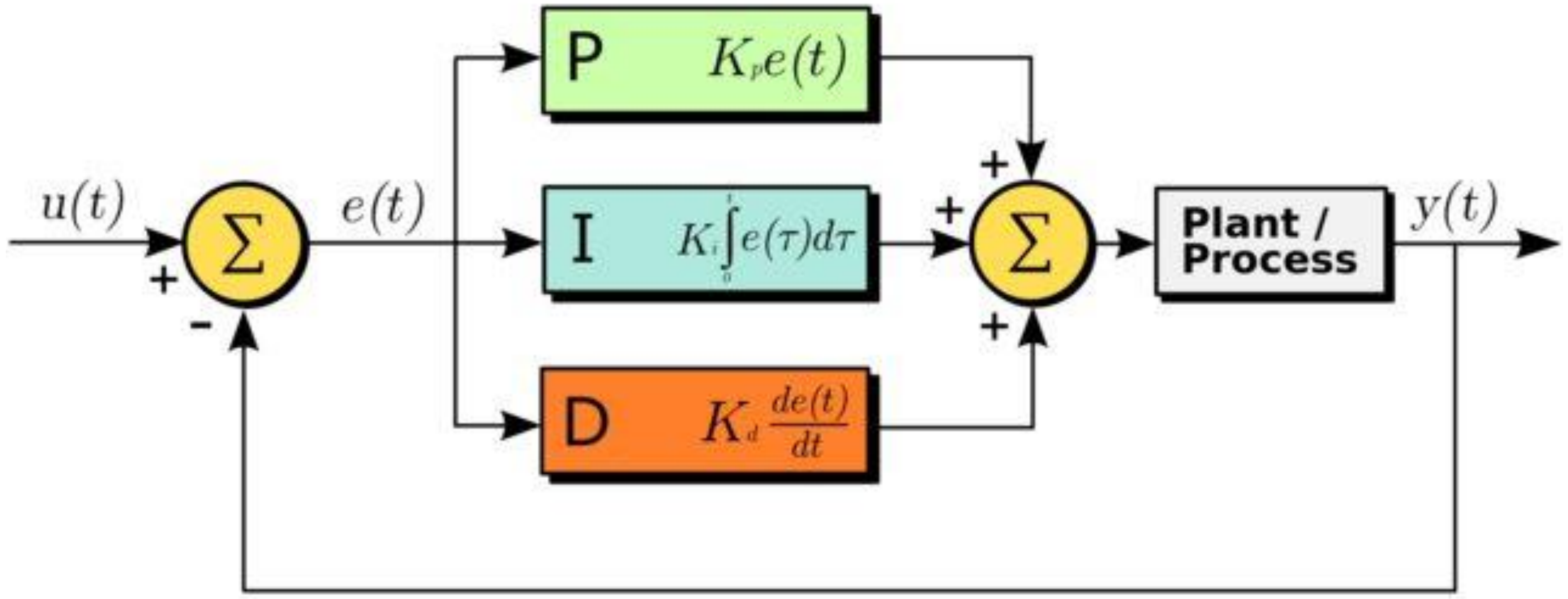
Carnegie Mellon ROBOTICS CLUB

Feed Forward

- Usually denoted as kB
- Interchangeable with integral control
    - Easier to use, less likely to cause crazy errors.
- Constant value on the robot unrelated to the error
- In terms of a moving robot it can be considered as its base speed
- It can be arbitrary or based on the relation between input and output of an algorithm.

PID Control

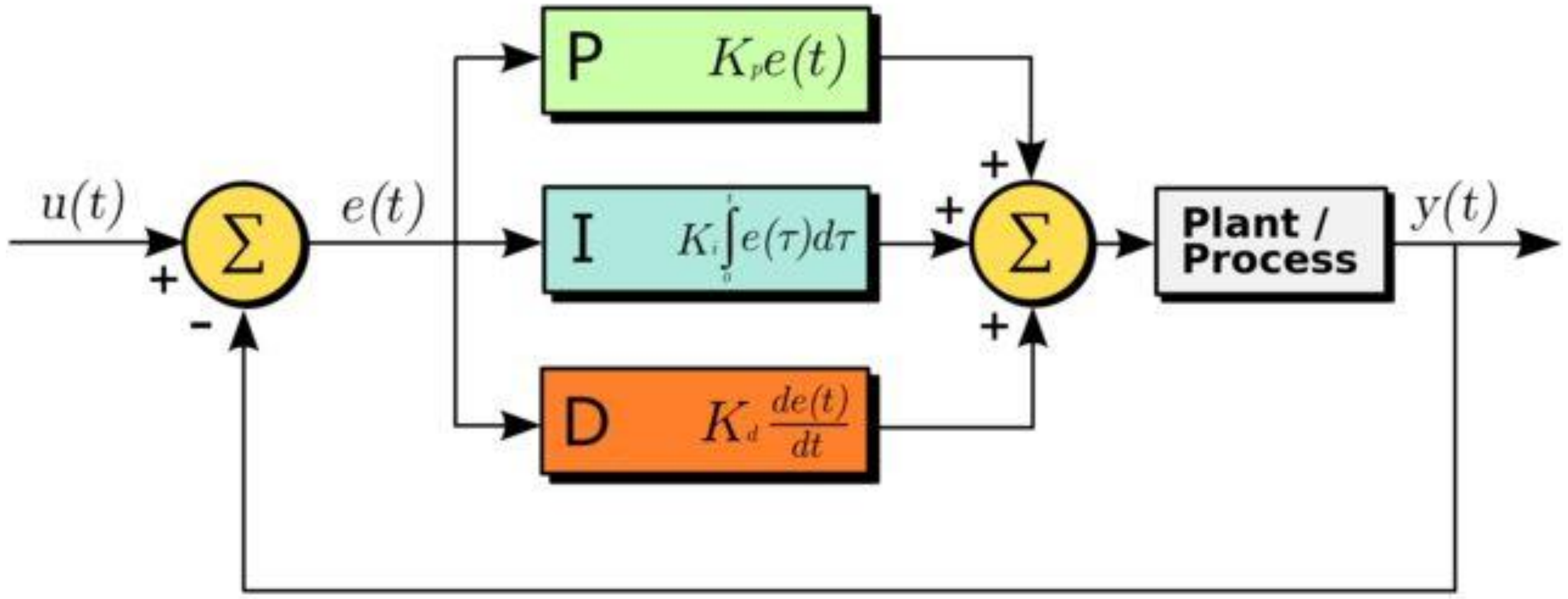
- Output = Proportional * Kp + Integral * Ki + Derivative * Kd



Carnegie Mellon ROBOTICS CLUB

PID Control

- Output = Feed Forward
  +/- Proportional * Kp + Integral * Ki + Derivative * Kd



Carnegie Mellon ROBOTICS CLUB

Note on Delays

- Control loops rely on repeated updating
- New outputs are calculated each loop
- You want to do this as often as possible
- Long delays = slow reaction time

Carnegie Mellon ROBOTICS CLUB

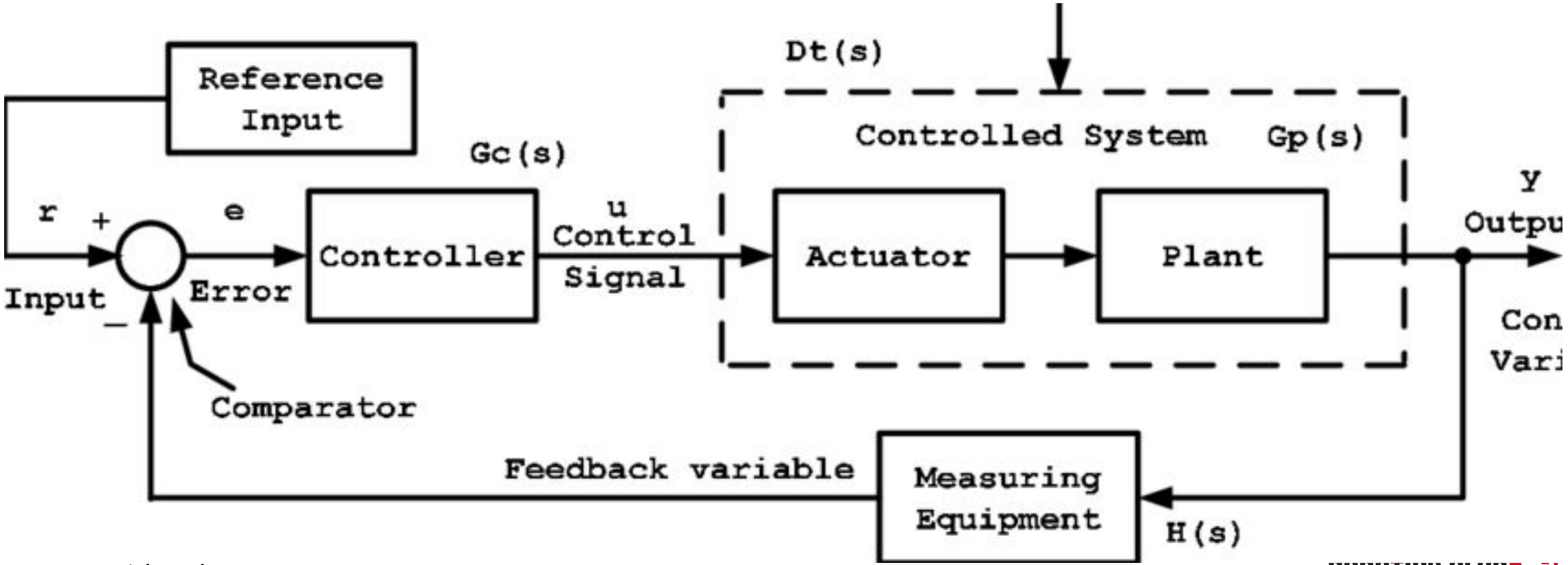
Starter Code Tour

Installing the QTR Library

- Installation Instructions

Bonus Candy Question

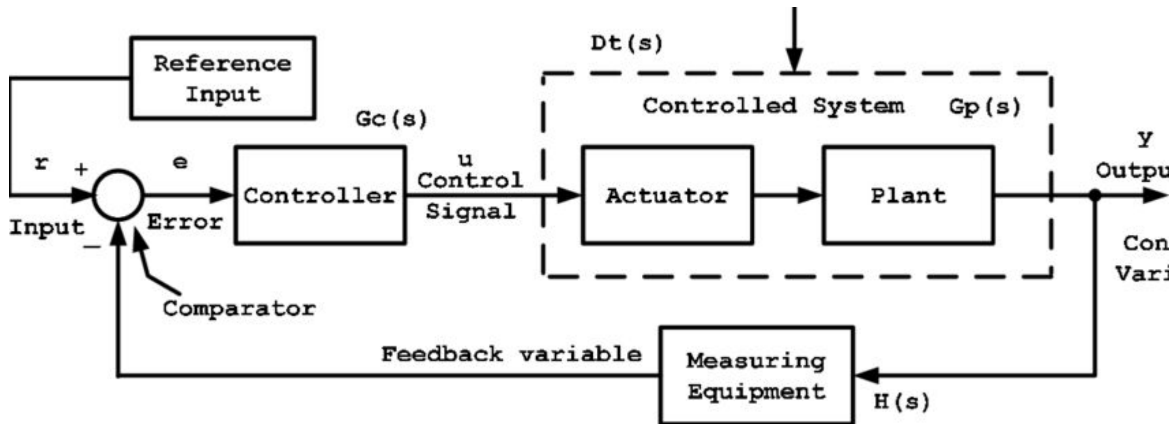
What is the reference input, controller, and output in our system? What is the measuring equipment?



ROBOTICS CLUB

Answer:

- **Input**: Desired position for robot
- **Controller**: PID
- **Actuator**: Robot's motors
- **Plant**: Robot
- **Output:** Robot's current position
- **Feedback Variable**: Robot's current position, sensed



Carnegie Mellon ROBOTICS CLUB