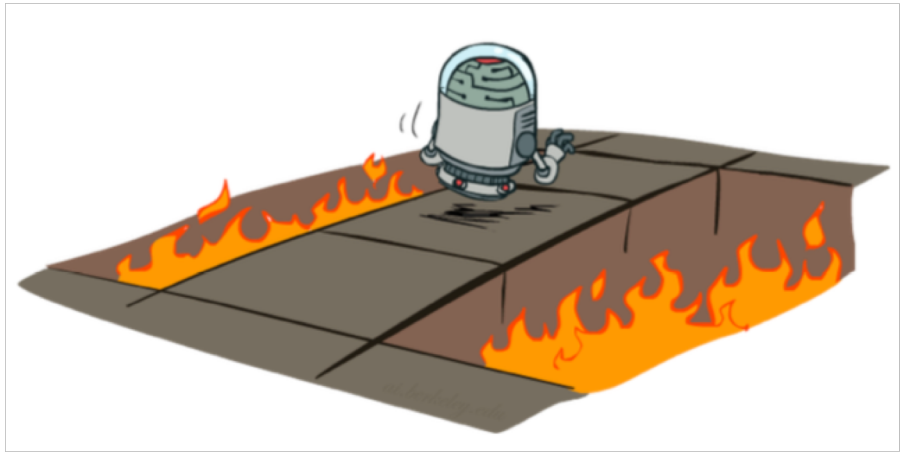
CS 1501: Intro to Robotics

Autonomy, AI, and Applications



Motion Control

Rohan Raval

Monday 1-1:50pm, MEC 213

Reading Responses and Anon. Feedback

Title of Reading *

Autonomous Mobile Robots

Reading Response *

Please also include a link if this is not one of the readings on the website!

I liked the idea, just a little reading heavy.

Any topics you would like to go over more?

5 responses

the coding behind basic robotics

I know very little about these subjects, so I'll leave it up to you to c
come into class.

More complicated and current algorithms

More coding topic styles.

none

What c[illegible] to make the class better?

8 responses

more class involvement

I would like it if you could incorporate m[illegible] programming into the lectures, as in showing us algorithms for how data collection works or explaining to us how [illegible] about writing code to collect data.

An overview of the topic at the start of class may be h[illegible] If possible, incorporating videos showing algorithms working might also be helpful.

More in class activities like the algorithm practice from 3/18/19

More hands on projects if you can :-)

algorithm worksheets?

As much as I dislike extra work, I feel like there should be something to at least have us sort of demonstrate our knowledge. Right now I feel like it's easy to forget what I learn in this class because I don't really have to interact with the material at all outside of lecture.

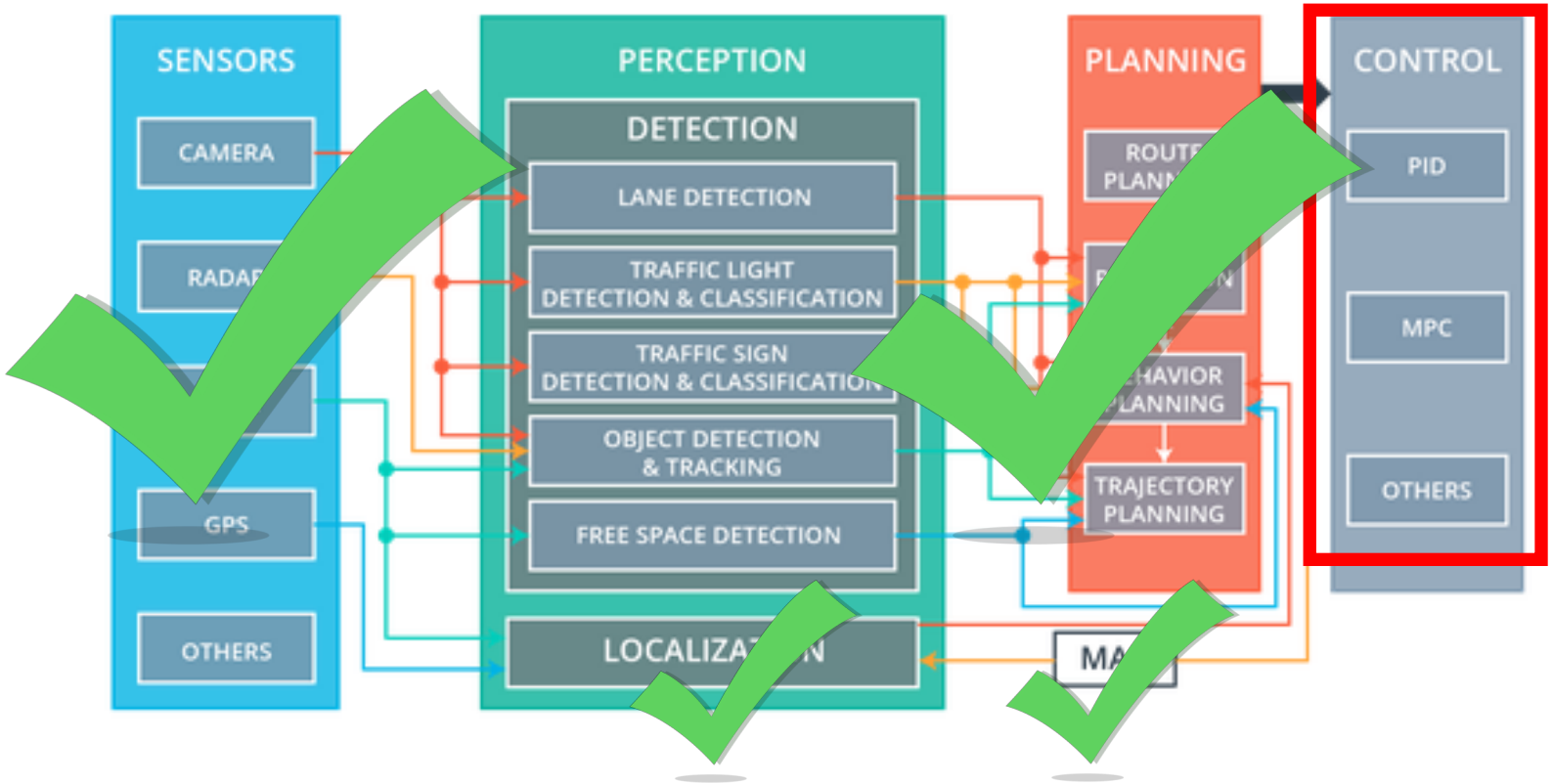
maybe more activities

mOrE cOdEiNG???

In the News...

https://www.youtube.com/watch?v=5iV_hB08Uns

Roadmap: See-Think-Act



Roadmap: Today's Lecture...

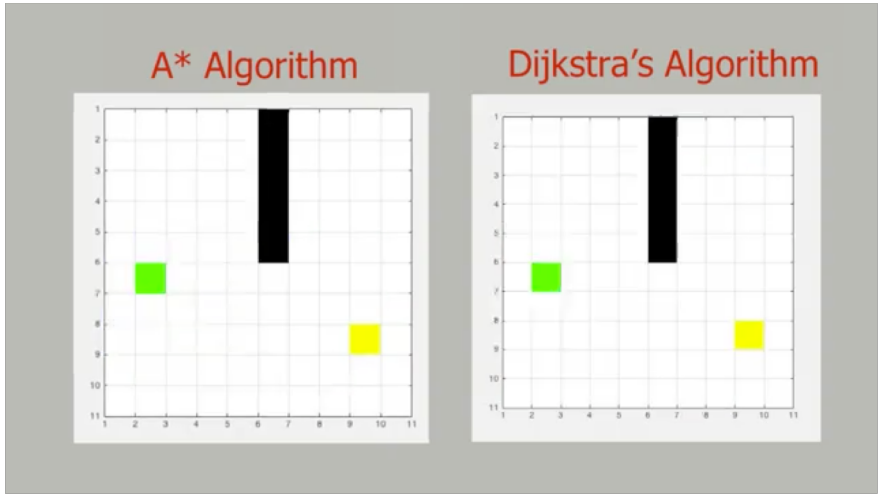
Goal: How does a robot *robustly* **execute** the motion that we've planned?

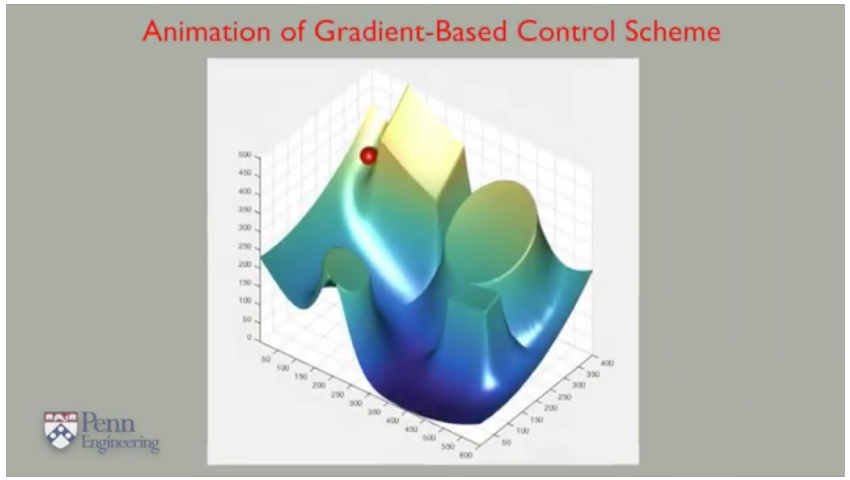
Topics:

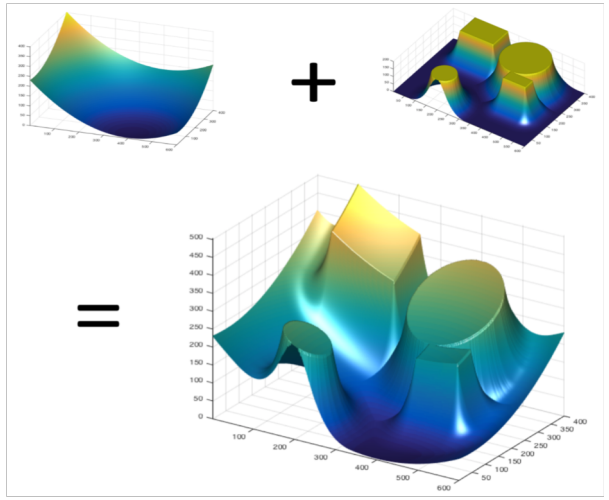
- Recap of Motion Planning
- Motivation for Control Theory
- Feedback Loop
- What is a Controller
- P-controller
- PD-controller and PI-controller
- PID Controller
- Parameter Optimization
- Twiddle

Recap: Motion Planning

- Basics: BFS and DFS
- Dijkstra's Shortest Path Algorithm
- Heuristics
- A* Algorithm
  - Is it optimal?
- Artificial Potential Fields!
  - Gradient Descent
  - Local Minima
  - Is it optimal?







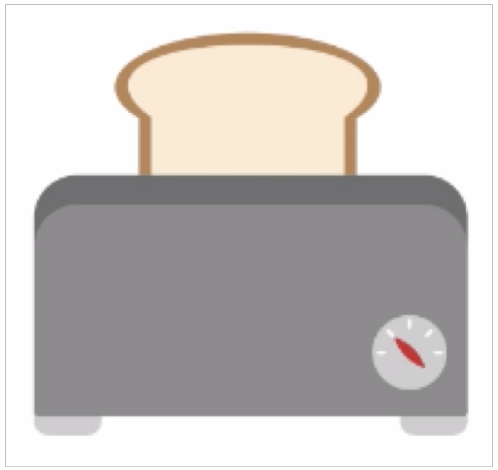
Motivation

- **Goal**: Autonomously move from Point A to Point B on a map
  - **What** should the robot do (path) = *Motion Planning*
  - **How** should the robot do it (actions) = *Motion Control*

- *How* do we make robots move in **effective**, **safe** and **predictable** ways?
  - Avoid obstacles (on the fly)
  - Smooth (non-oscillatory) paths
  - Resilient to disturbances
  - Variations in the system

Motivation

- Robots
- Thermostat
- Cruise Control
- Stock Markets
- Circuits
- Epidemics
- Toasters...



Motivation

https://www.youtube.com/watch?v=4PaTWufUqqU

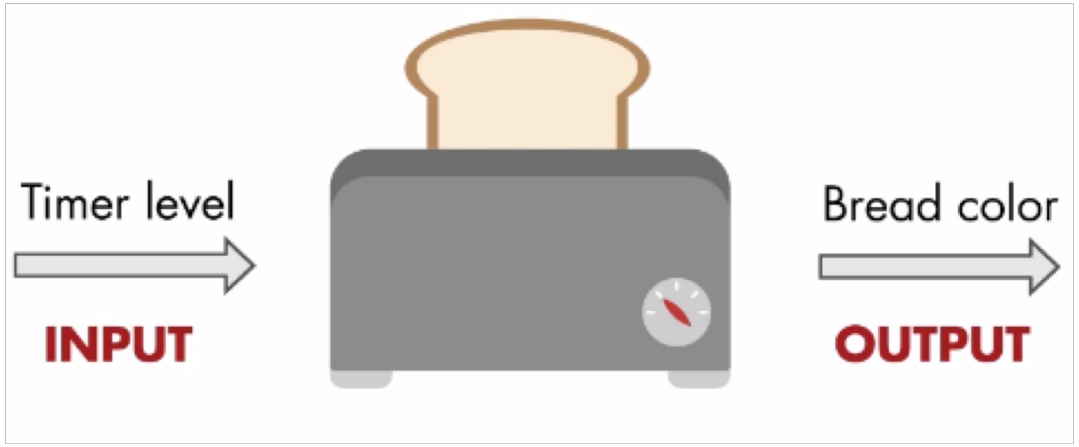
https://www.youtube.com/watch?v=gstcLQ_LC1g

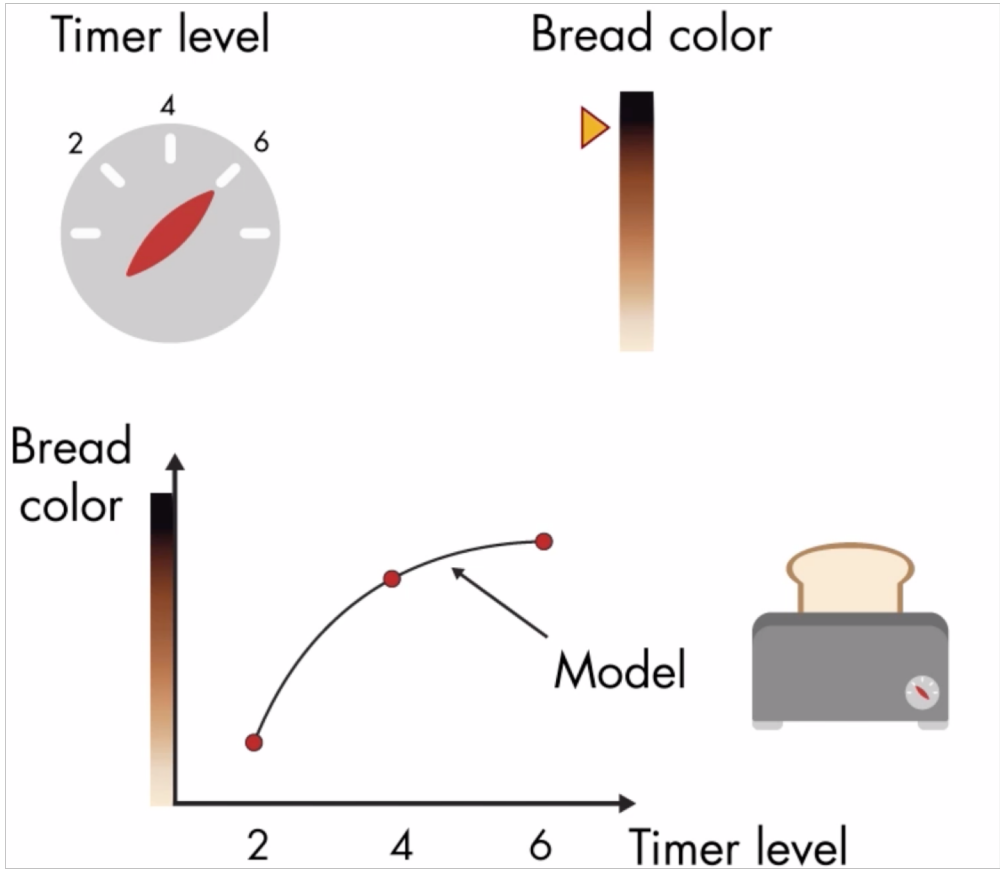
https://www.youtube.com/watch?v=se318w2LXD0

Open-Loop Control

Example: Toaster Controller

- Want to make toast of a desired "color"
- Need to set timer accordingly
- Find "model" of system



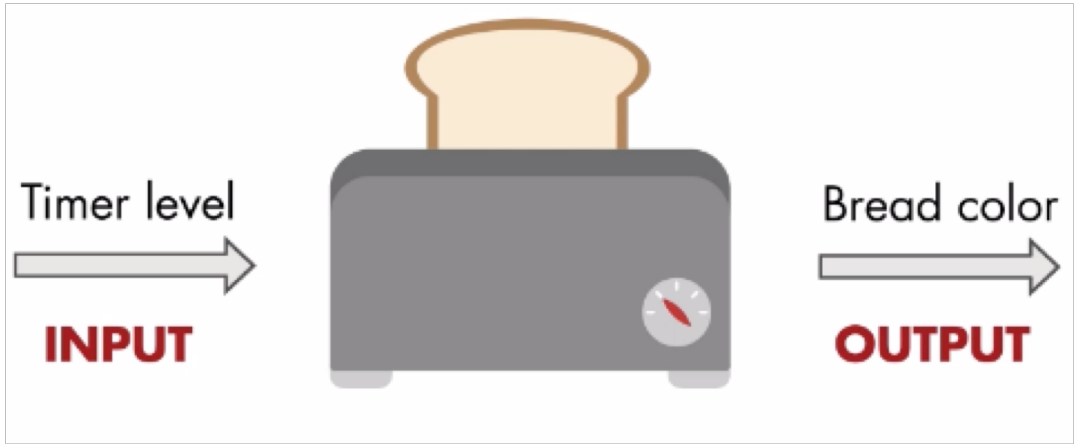


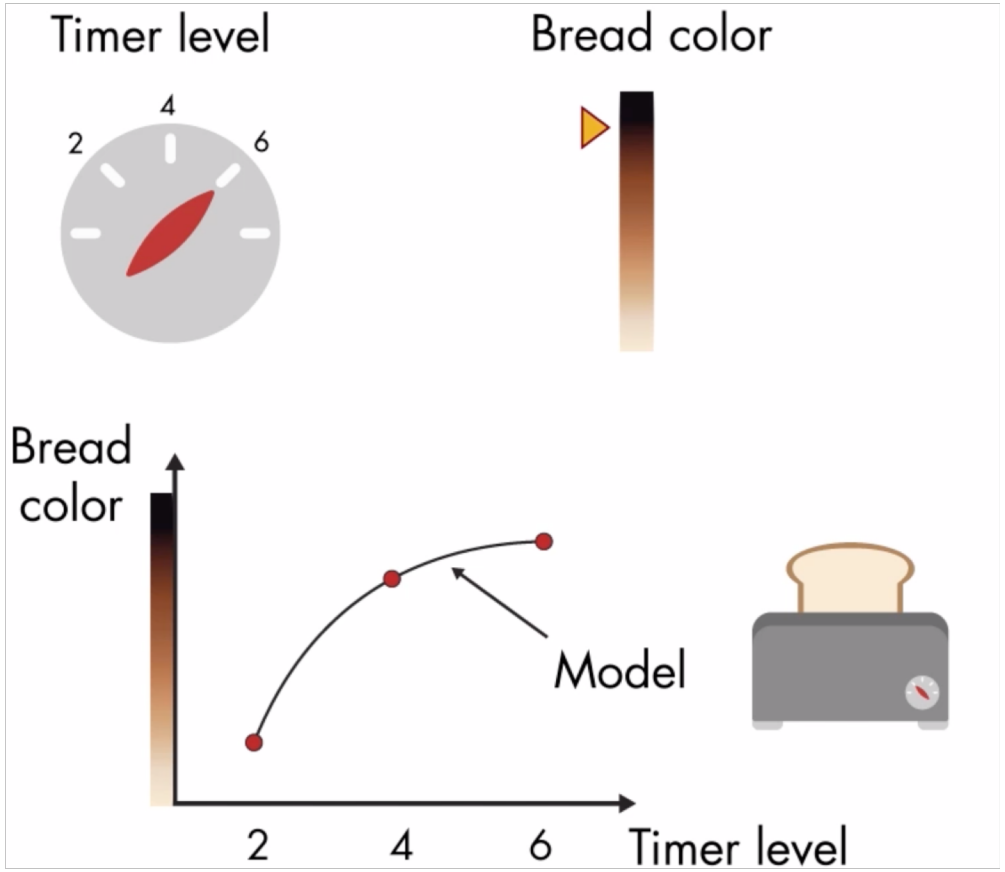
Pictures adopted from MathWorks Tutorial

Open-Loop Control

Problems?

- What if we want to make a bagel?
- What if there is unexpected disturbances?

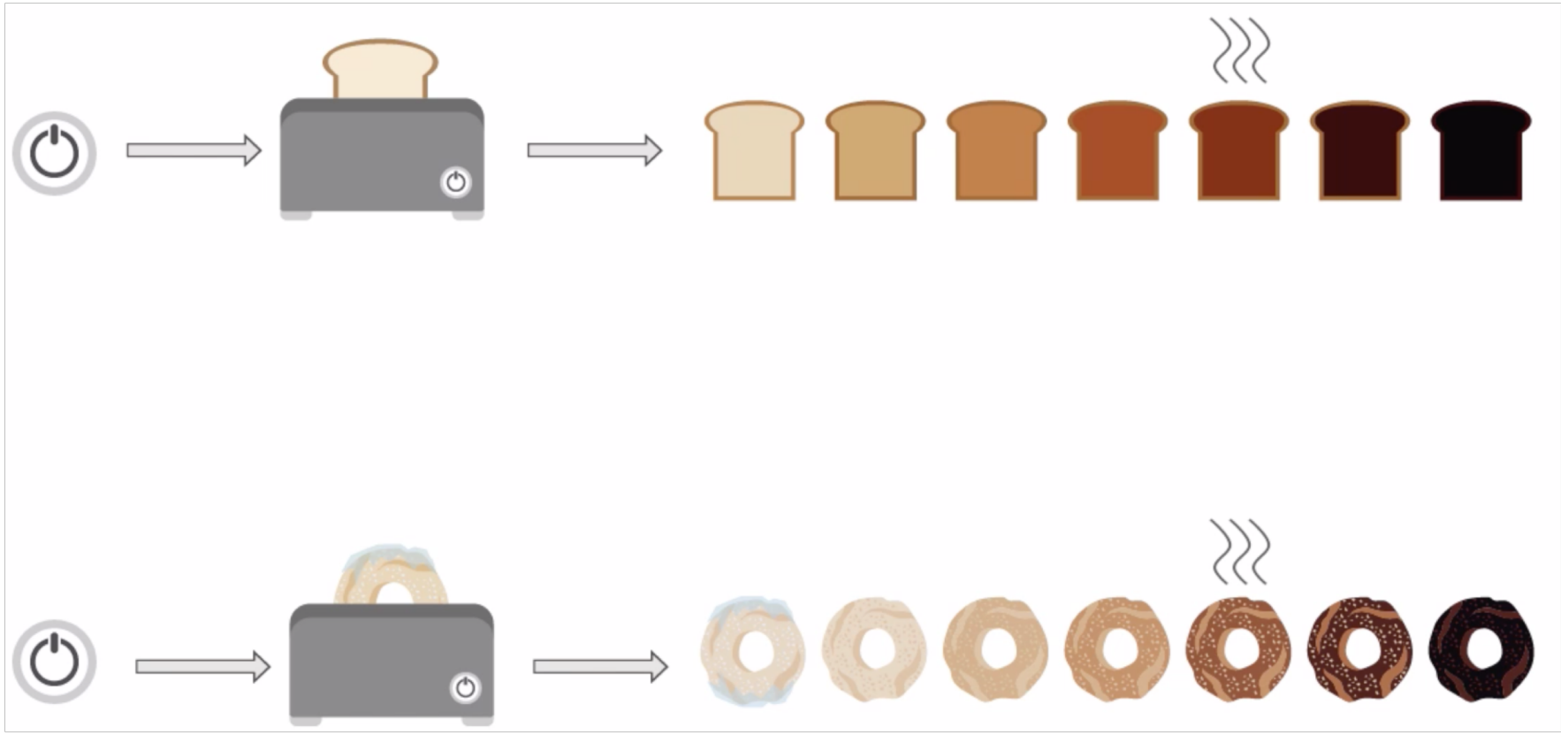




Pictures adopted from MathWorks Tutorial

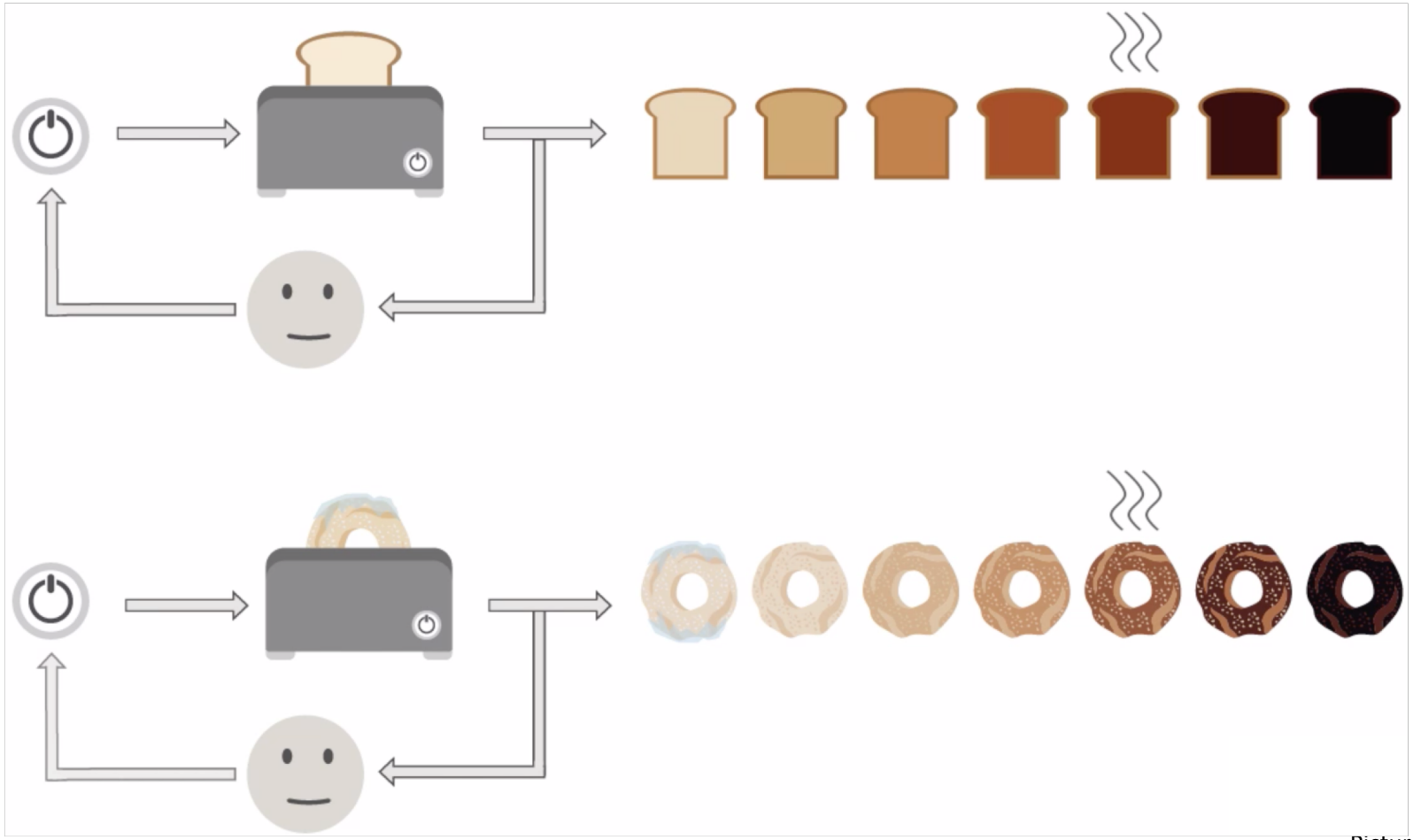
Feedback (Closed-Loop) Control

Instead of toasting based on timer, what if we toast based directly on color?



Feedback (Closed-Loop) Control

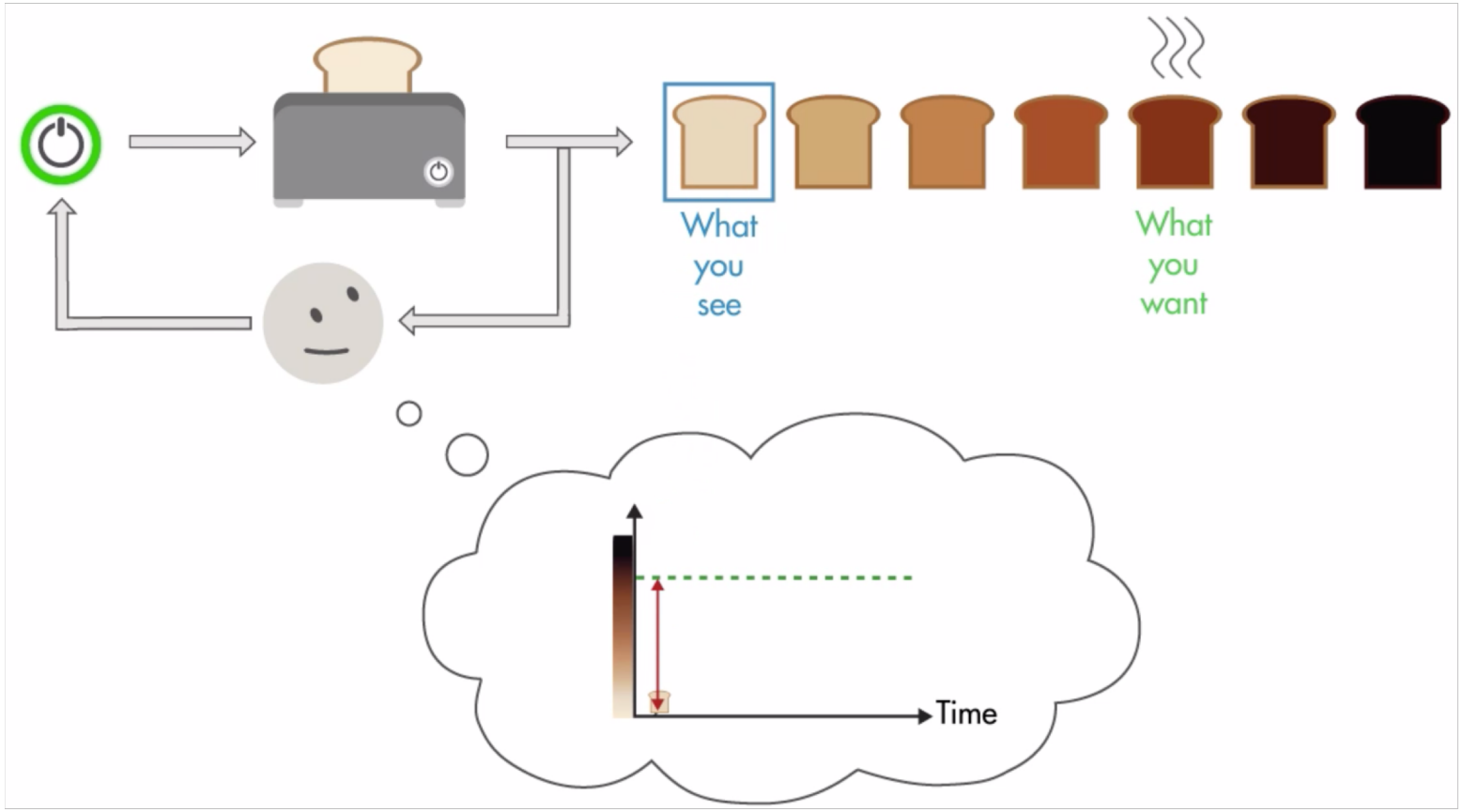
Introduce a **sensor** (you) to monitor the system!



Pictures adopted from MathWorks Tutorial

Feedback (Closed-Loop) Control

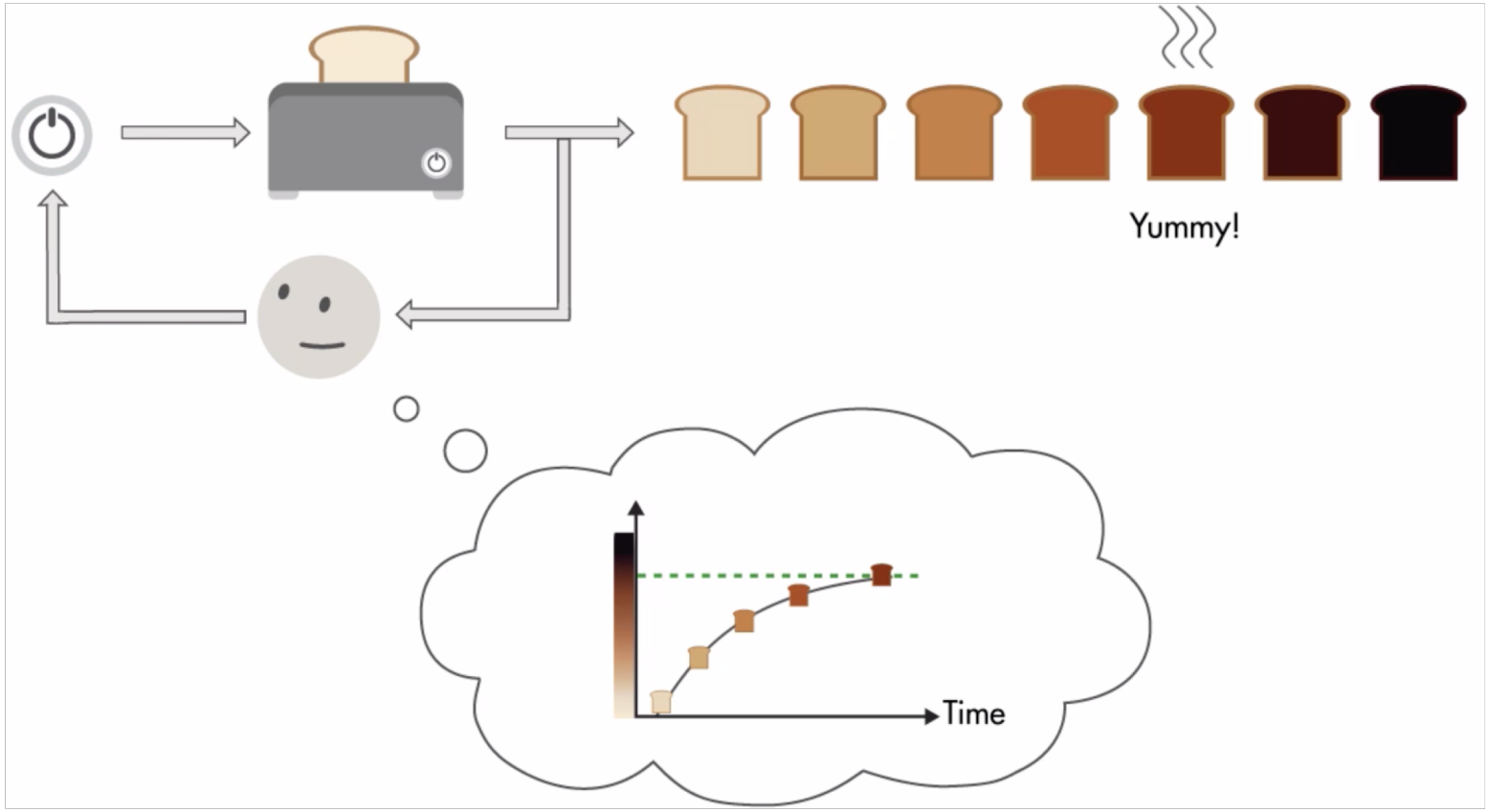
Sensor calculates **error** between "what you want" and "what you see"



Pictures adopted from MathWorks Tutorial

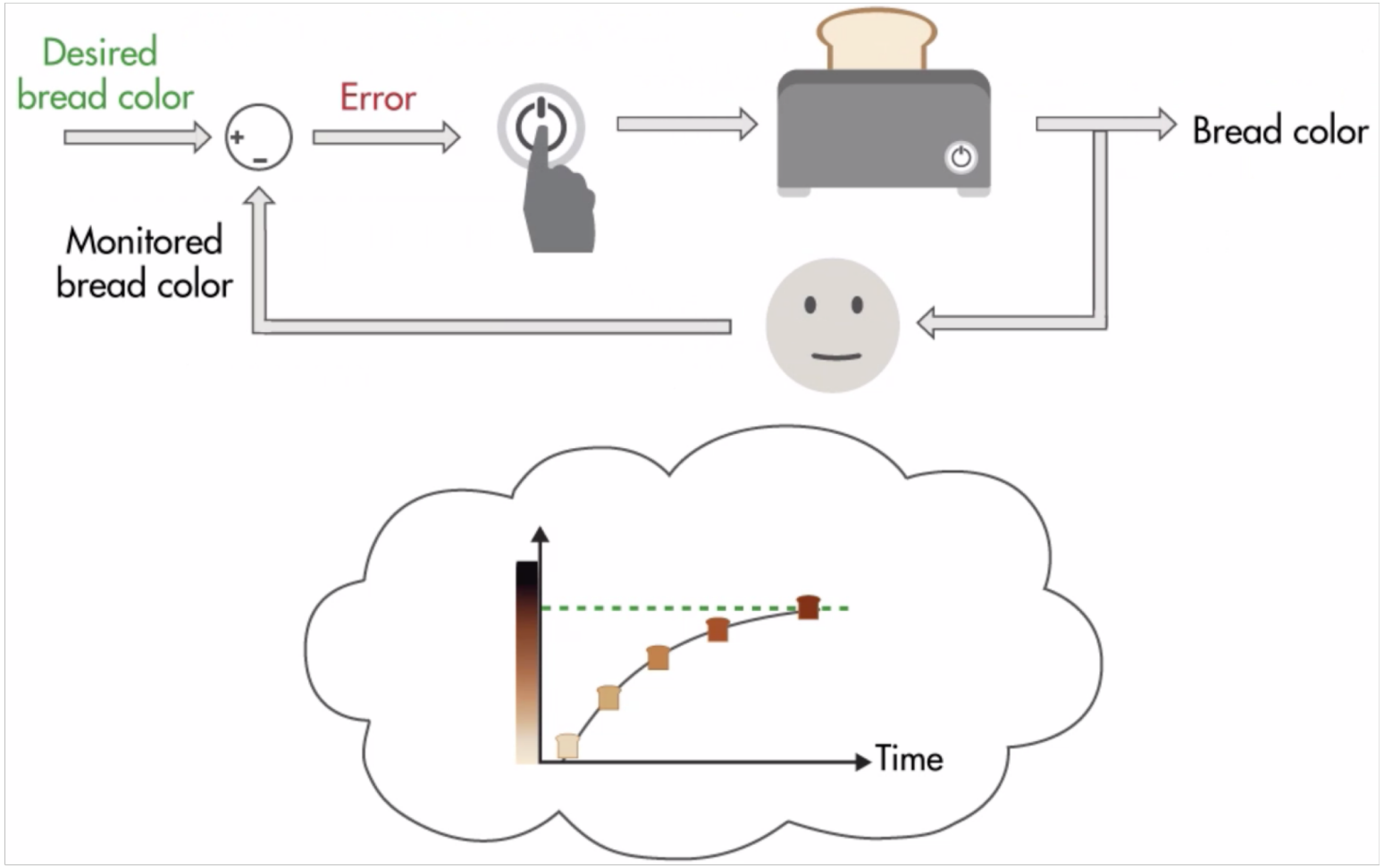
Feedback (Closed-Loop) Control

Sensor calculates **error** between "what you want" and "what you see"



Pictures adopted from MathWorks Tutorial

Feedback (Closed-Loop) Control



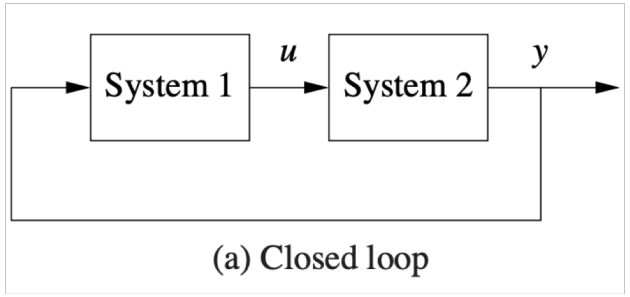
Pictures adopted from MathWorks Tutorial

Control Theory: Definitions

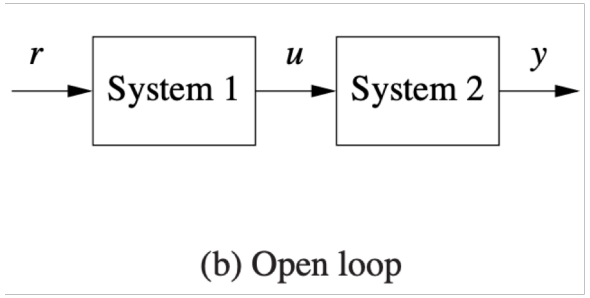
Dynamic Systems: Systems of correlated variables that *change* (evolve) over time

Control: How we *influence* that change

Feedback: two or more dynamical systems are connected together such that each system influences the other



(a) Closed loop



(b) Open loop

FEEDBACK CONTROL: To make corrective actions based on the difference between the desired and the actual values of a quantity

Control Theory: Building Blocks

State (x): Representation of what the system is currently doing

- E.g. Position/Velocity/Steering-angle of Robot

Dynamics: Description of how the state changes

- E.g. Dynamics of Robot Motion, like 4-wheel drive, and associated physics

Reference (r): What we want the system to do

- E.g. "desired velocity" for cruise control

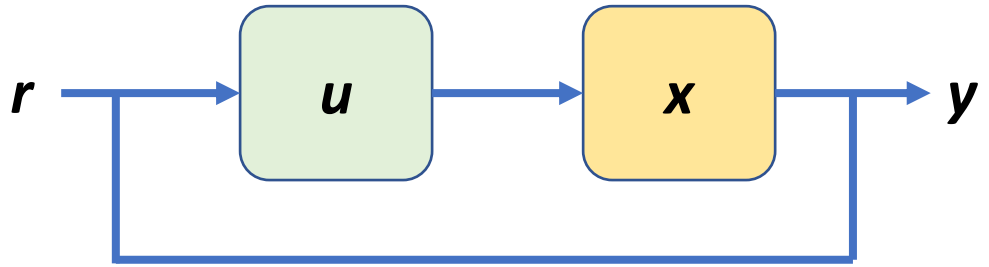
Output (y): Measurement of relevant aspect of the system

- E.g. odometer measurement for velocity

Input (u): Control Signal

- E.g. "accelerate at 30% throttle for 5 seconds"

Feedback: Mapping from outputs to inputs

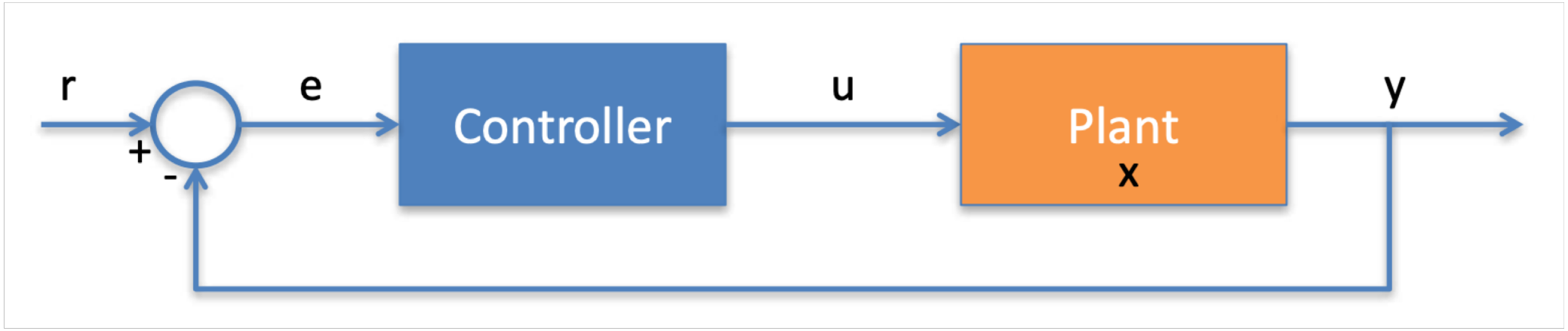


Control Theory: Building Blocks

Control Theory = how to pick the input signal u?

Objectives:

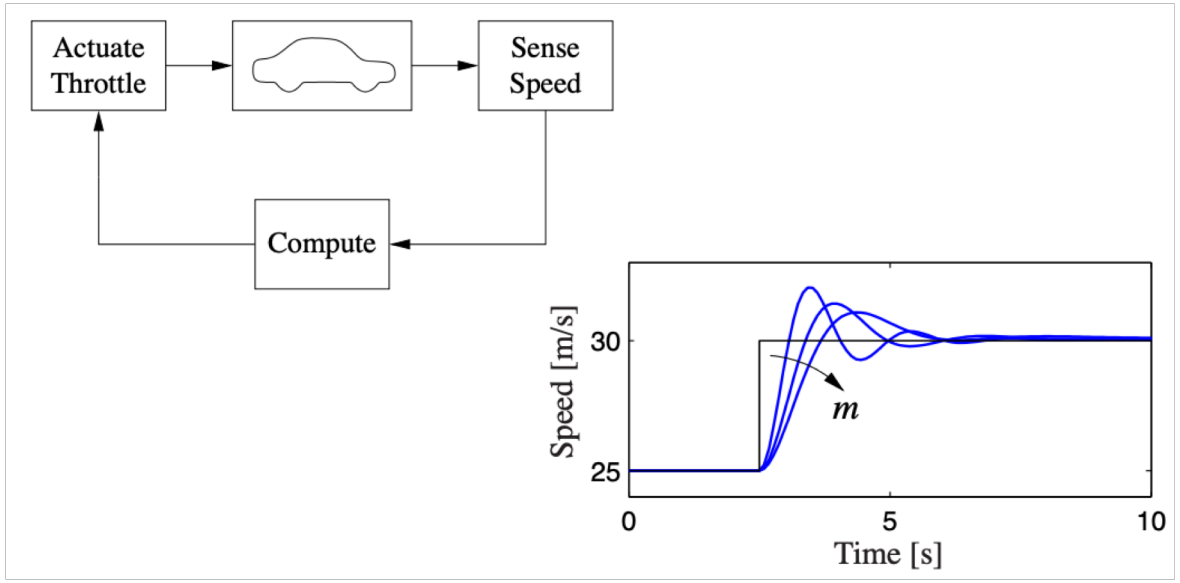
- Stability, Tracking, Robustness, Disturbance Rejection, Optimality...



Cruise Controller

What properties should the control signal *u* have?

- Should be a function of *error* (difference between reference *r* and output *y*)
  - If small error, then small *u*... no overreacting
- *u* should not be "jerky"
- *u* should not depend on us knowing exactly all the parameters of the system (e.g., mass, inertia)



Attempt 1: Bang-Bang Controller

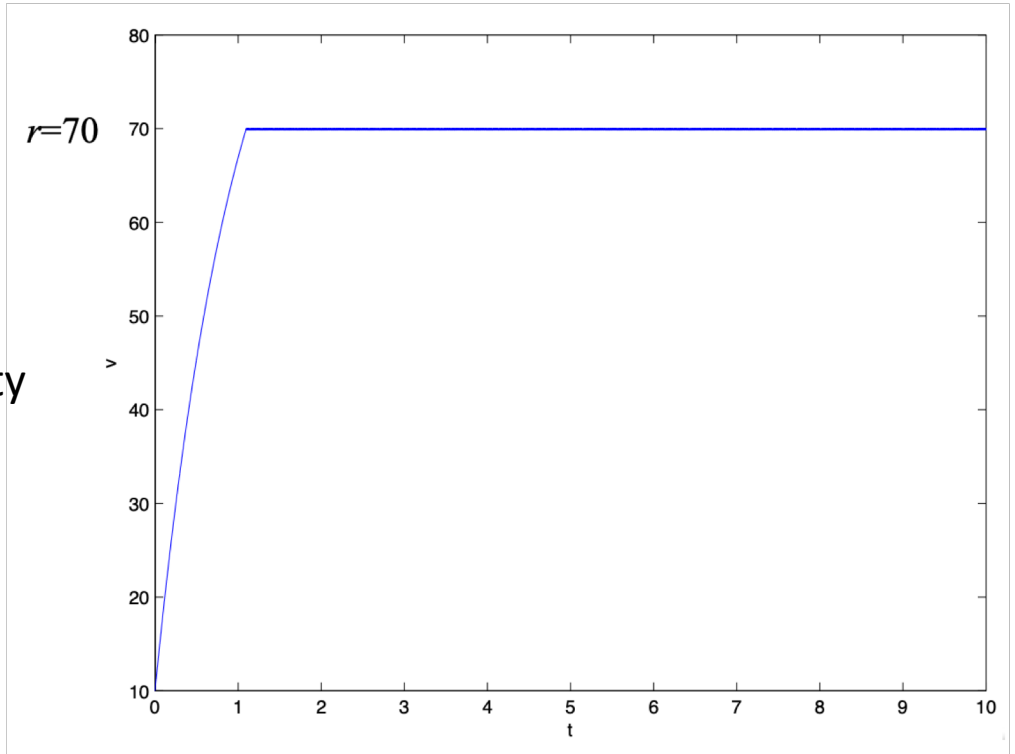
$$u = \begin{cases} u_{\max} & \text{if } e > 0 \\ -u_{\max} & \text{if } e < 0 \\ 0 & \text{if } e = 0 \end{cases}$$

"FLOOR IT"

"SLAM THE BRAKES"

"Do nothing"

velocity



Goal:

- Cruise Control at 70mph

This reaches the goal pretty well!

What's the problem...?

Attempt 1: Bang-Bang Controller

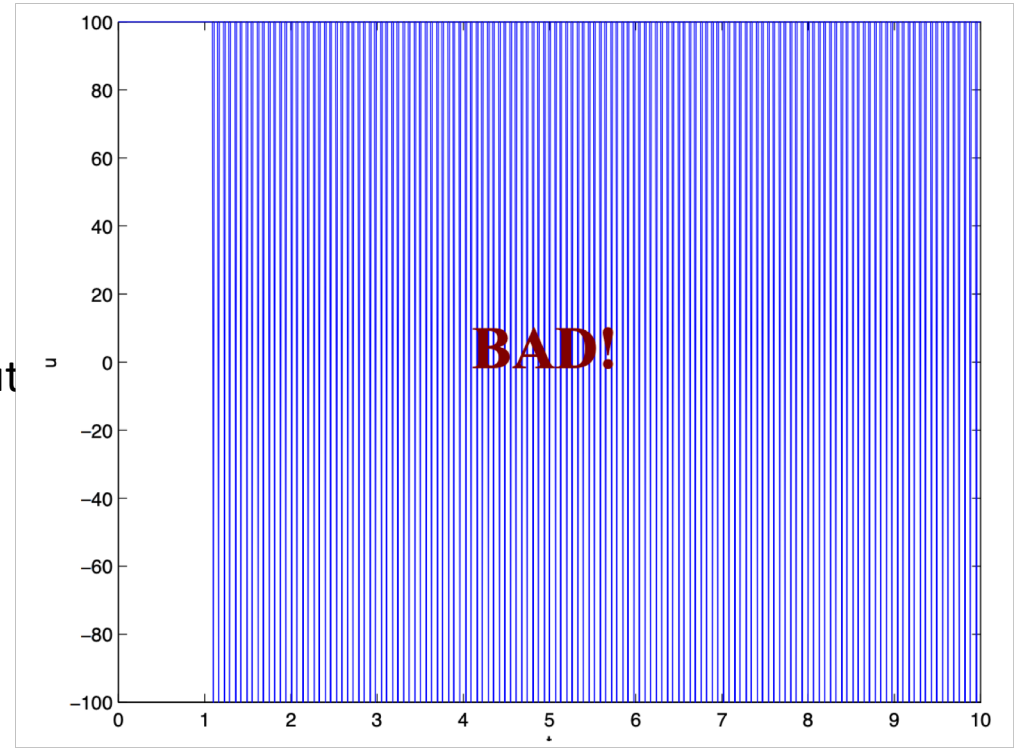
$$u = \begin{cases} u_{\max} & \text{if } e > 0 \\ -u_{\max} & \text{if } e < 0 \\ 0 & \text{if } e = 0 \end{cases}$$

"FLOOR IT"

"SLAM THE BRAKES"

"Do nothing"

Control input

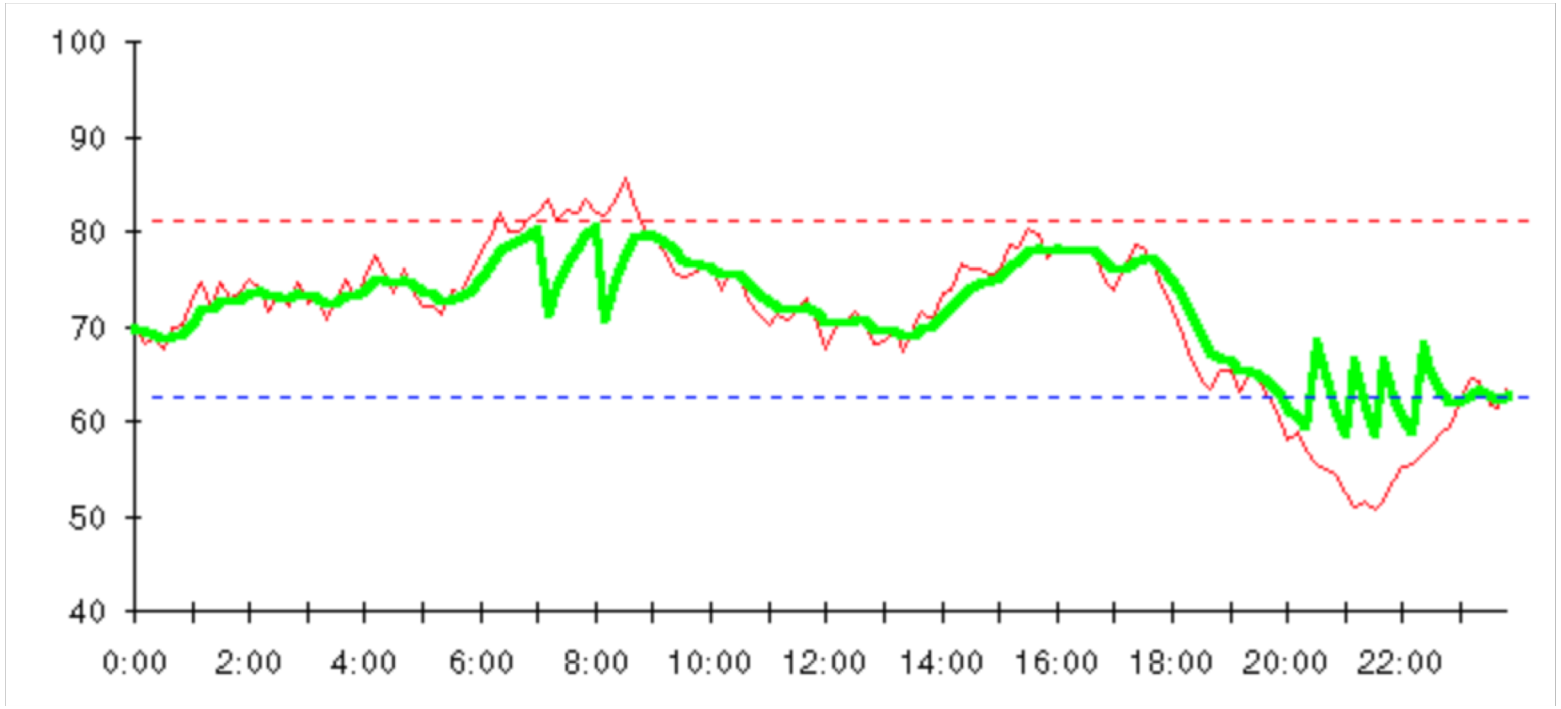


Why bad?

- Bumpy ride
- Burns out actuators!

Attempt 1: Bang-Bang Controller

Example: Thermostat Regulation

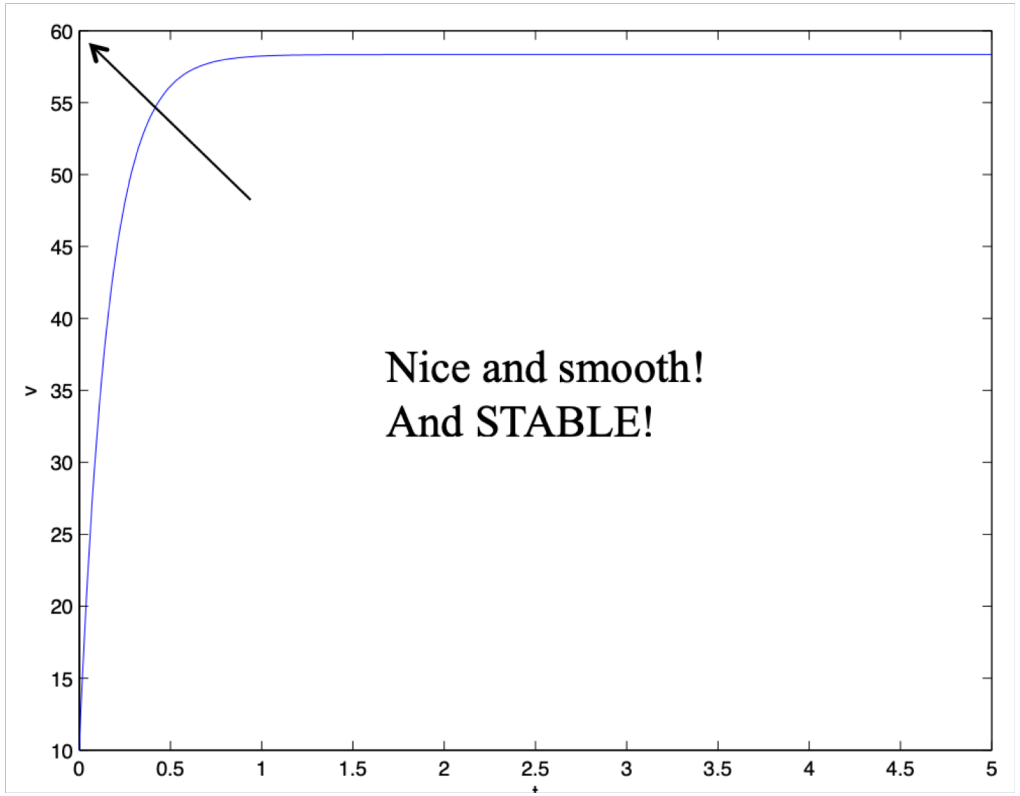


Attempt 2: P-Controller

$$u = ke$$

control signal *u* **proportional** to error *e,* by *k*

velocity



This also looks pretty good!

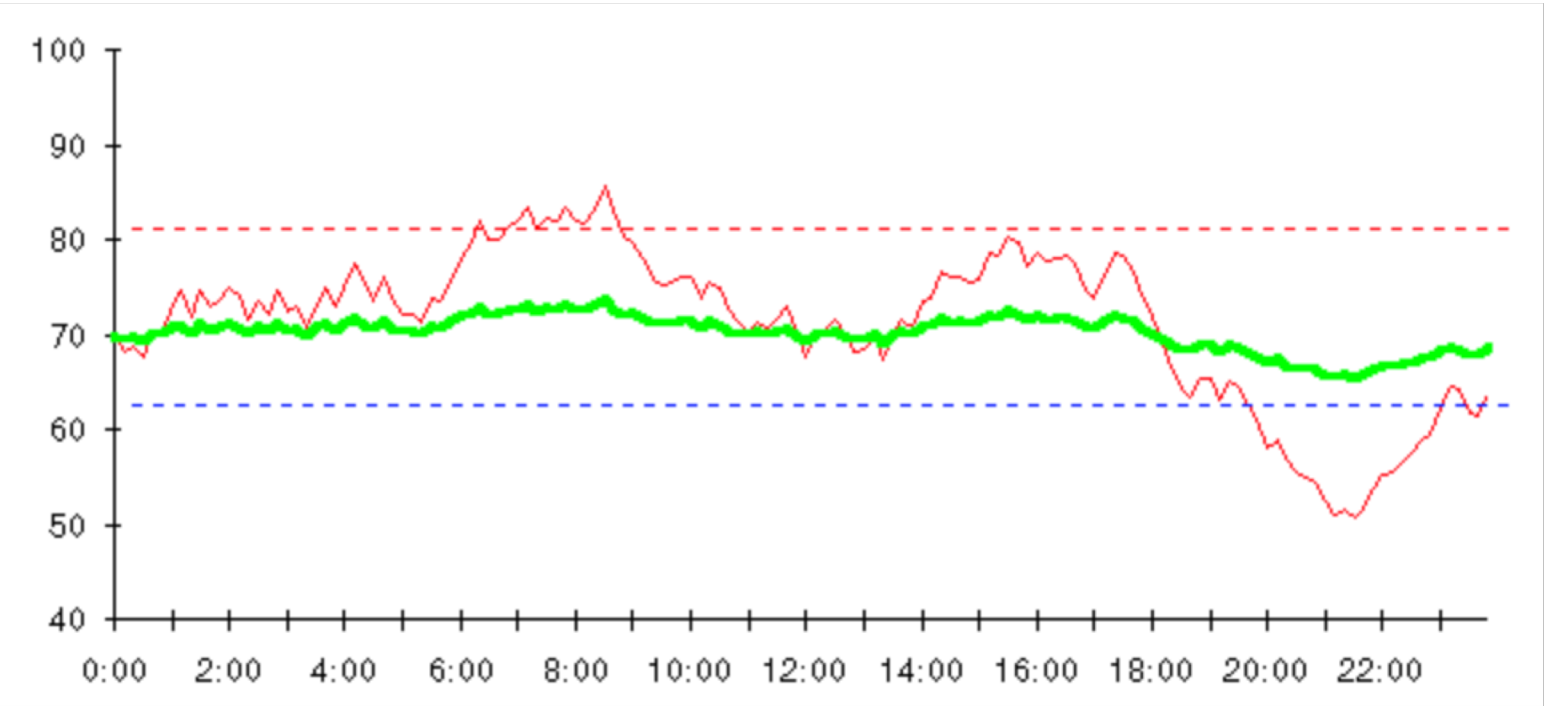
What's the Problem?

- Doesn't actually get to 70mph goal
- Vulnerable to sustained sources of large error
  - Disturbances (wind resistance)
  - Systematic bias

Need to *accumulate* error over time...

Attempt 2: P-Controller

Example: Thermostat Regulation

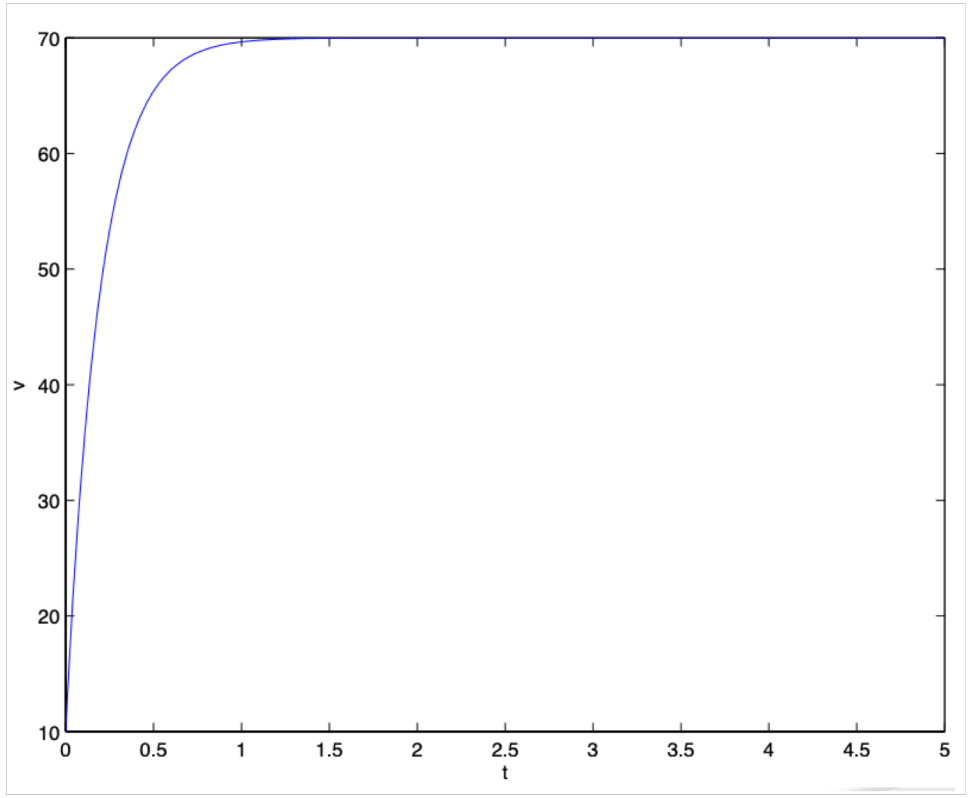


Attempt 3: PI-Controller

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)d\tau$$

control signal *u* **proportional** to error *e,* by k_p and proportional to **integral** of error by k_i

velocity



This looks great! Are we done yet??

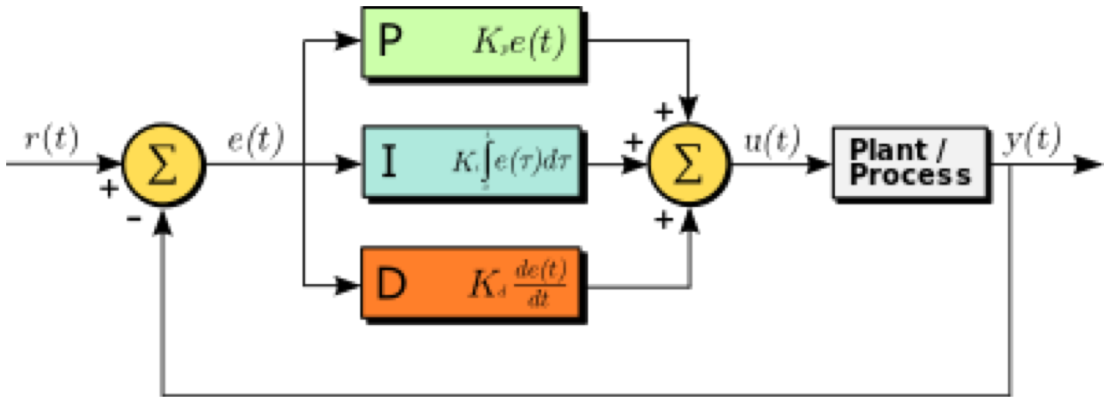
No… What's the problem?

- P causes oscillations (you will usually overshoot) and become unsteady ("marginally stable")
- Integral does *not* solve these oscillations

PID Controller

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)d\tau + k_D \frac{de(t)}{dt}$$

- **Proportional (k_P)**: Contributes to stability, medium-rate responsiveness
- **Integral (k_I)**: Tracking and disturbance/bias rejection, slow-rate responsiveness
- **Derivative (k_D)**: Fast-rate responsiveness, sensitive to noise



PID Controller

$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)d\tau + k_D \frac{de(t)}{dt}$$

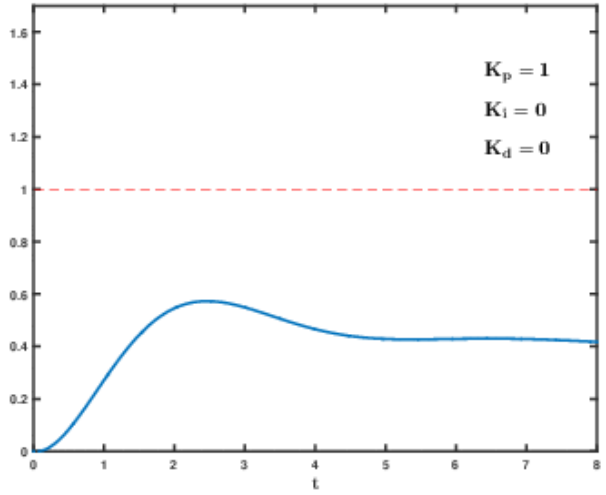
- Pseudo-code:

```
previous_error = 0
integral = 0
start:
  error = setpoint - measured_value
  integral = integral + error*dt
  derivative = (error - previous_error)/dt
  output = Kp*error + Ki*integral + Kd*derivative
  previous_error = error
  wait(dt)
  goto start
```

Parameter Optimization: Tuning a PID

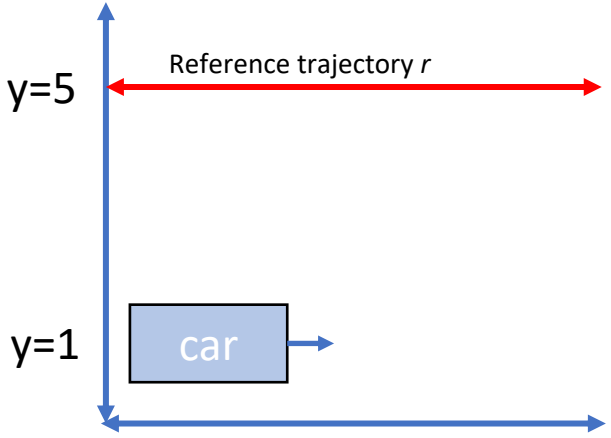
$$u(t) = k_P e(t) + k_I \int_0^t e(\tau)d\tau + k_D \frac{de(t)}{dt}$$

- **Steps to tune your k_P, k_I and k_D paramters:**
  - Start with k_P and increase until oscillations appear
  - Then move to k_I to reduce error
  - Finally move k_D to reduce oscillations
  - Go back to k_P and change until almost no oscillations (just a little overshoot)

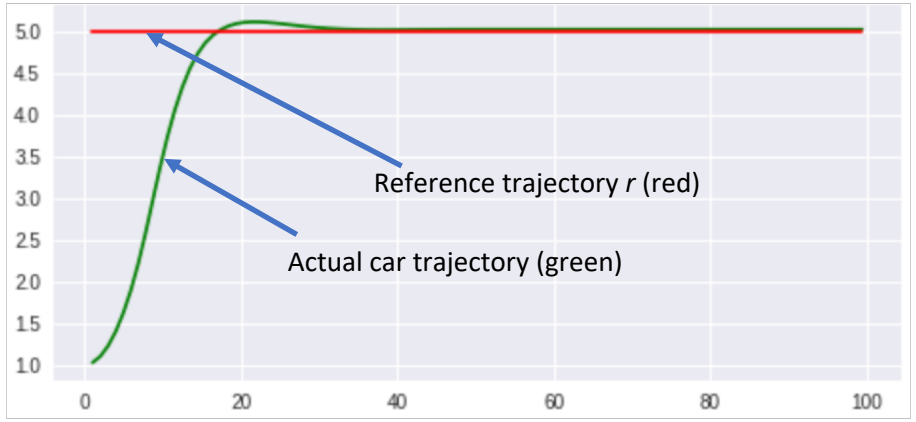


PID Controller: Exercise

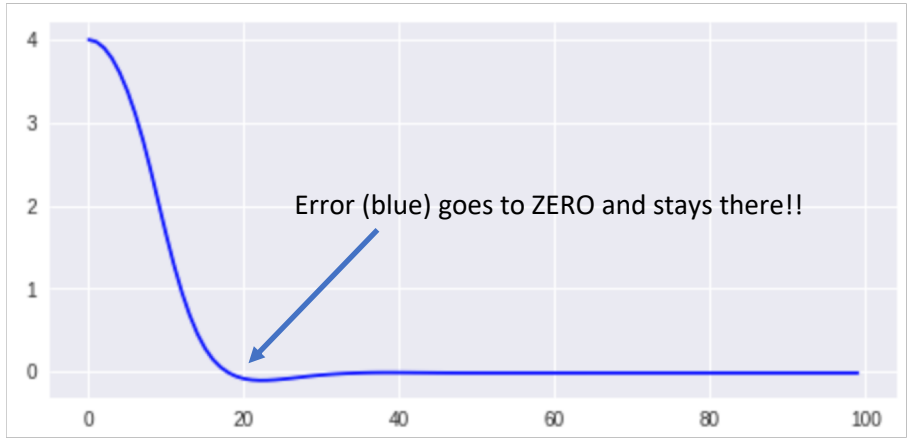
- **Task:**
  - Car starts at y=1
  - We want to follow a trajectory of y=5 (for all x)
  - So we need to steer car from y=1 to y=5, using a PID controller, & keep it there
  - Controller is on the *steering angle* (not velocity, like the previous slides!)
  - Fill in the code here: [bit.ly/cs1501-pid](bit.ly/cs1501-pid)



Depiction of task



What we want to do



*What we want the **error** graph to look like*