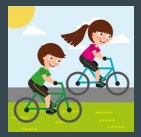
CS4501
Robotics for Soft Eng

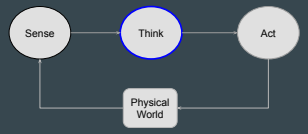
•••

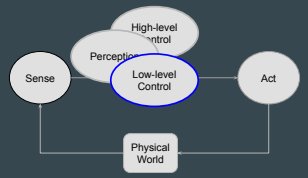
Control

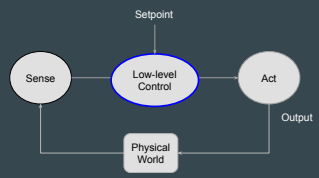
Problem: Ride over straight line



- Sensors are noisy
  - Eyes, ears-balance, ...
- Actuators are noisy
  - Muscles, bike gears, breaks, ...
- Environment changes
  - Street, Grass, Rock, Mud, …

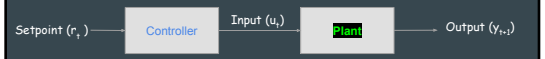


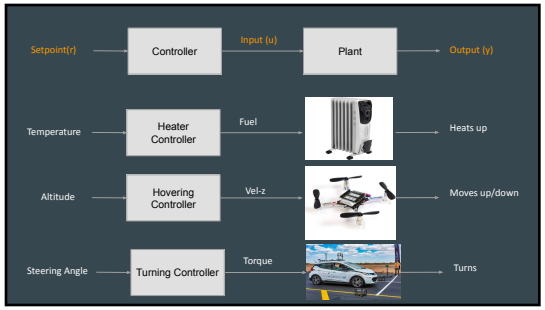


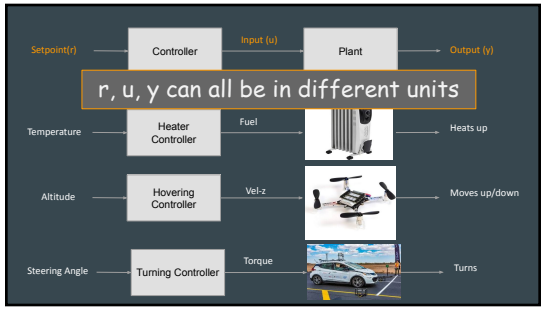


Goal of Low-Level Controller
Sensed Output = Setpoint

- Goal
  - Controller aims to make Sensed Output = Setpoint
- Terms
  - Plant (system) with Inputs (u) and Outputs (y)
  - Setpoint (r)

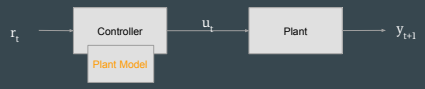






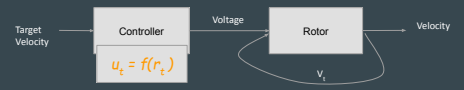
Families of controllers

Open-loop controller



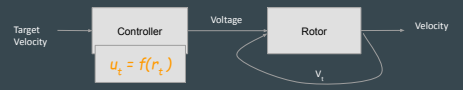
- Assumes we have a Good Model of the Plant
- Computes u_t input to plant given r_t: $u_t = F(r_t)$

Open-Loop Controller: Drone Example



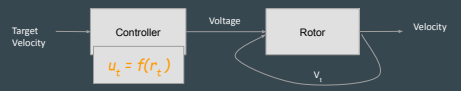
- Assumes we have a good model of Drone Rotor
  $Velocity_{t+1} = Velocity_t * 0.2 + Voltage * 0.5$

Open-Loop Controller Example



- Assumes we have a good model of Drone Rotor
  $Velocity_{t+1} = Velocity_t * 0.2 + Voltage * 0.5$
- Objective: compute input (u_t) to plant based to reach r_t
  $Voltage_t = f(TargetVel_t)$

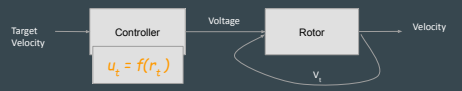
Open-Loop Controller Example



- Assumes we have a Good Model of Drone Rotor
  $Velocity_{t+1} = Velocity_t * .2 + Voltage * 0.5$
- Computes input to plant based: $Voltage_t = f(TargetVel_t)$

What should f be in the controller?

Open-Loop Controller Example

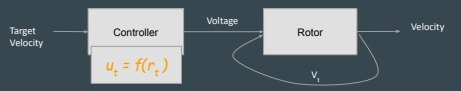


- Assumes we have a Good Model of Drone Rotor
  $Velocity_{t+1} = Velocity_t * .2 + Voltage * 0.5$
- Computes input to plant based: $Voltage_t = f(TargetVel_t)$

What should f be in the controller?

$u_t = f(r_t) = P * r_t$

Open-Loop Controller Example



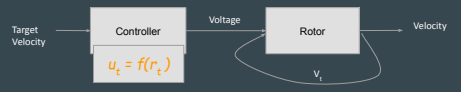
- Assumes we have a Good Model of Drone Rotor
  $Velocity_{t+1} = Velocity_t * .2 + Voltage * 0.5$
- Computes input to plant based: $Voltage_t = f(TargetVel_t)$

What should f be in the controller?

$u_t = f(r_t) = P * r_t$

$Voltage = P\ TargetVel$

Open-Loop Controller Example



- Assumes we have a Good Model of Drone Rotor
  $Velocity_{t+1} = Velocity_t * 0.2 + Voltage * 0.5$
- Computes input to plant based
  $Voltage_t = f(TargetVel_t)$

What should f be in the controller?

$u_t = f(r_t) = P * r_t$

$Voltage = P * TargetVel$

What should P be in the controller?

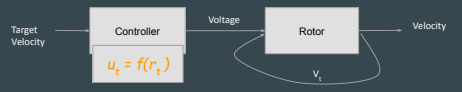
Steady state: $Velocity_{t+1} = Velocity_t$

$Velocity = Velocity * 0.2 + Voltage * 0.5$

$Velocity * 0.8 = Voltage * 0.5$

$Velocity * 1.6 = Voltage$

Open-Loop Controller Example



- Assumes we have a Good Model of Drone Rotor
  $Velocity_{t+1} = Velocity_t * 0.2 + Voltage * 0.5$
- Computes input to plant based
  $Voltage_t = f(TargetVel_t)$

What should f be in the controller?

$u_t = f(r_t) = P * r_t$

$Voltage = P * TargetVel = 1.6 * Target\ Velocity$

What should P be in the controller?

Steady state: $Velocity_{t+1} = Velocity_t$

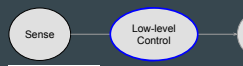
$Velocity = Velocity * 0.2 + Voltage * 0.5$

$Velocity * 0.8 = Voltage * 0.5$

$Velocity * 1.6 = Voltage$

Open-Loop Controller

- Good enough to keep temperature steady with expected air volume/flow
- Not as good if there is air flow or volume varies

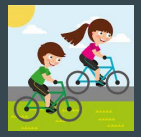


- Good enough for rpm on motors, drone on the ground, no propellers
- Not as good with propellers due to their differences
- Pretty bad when flying due to angle variations

Open-Loop Controller - Self test

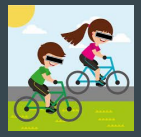
- Eyes closed
- Rotate 5 times in place
- Iterate
  - Walk 3 steps, rotate 90

Open-Loop Controller Example

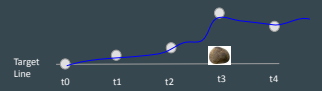


- Drive over a straight line

Open-Loop Controller (less ideal) Example

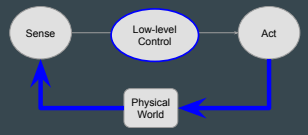


- Drive over straight line
- Open-loop ≈ close your eyes (no feedback)
  - Small errors will accumulate over time
  - Wheel may be a bit crooked
  - Disturbances (hitting a rock) may cause drastic changes



Limitations of Open-Loop Controller

- Performance depends on model/s
  - Fidelity in capturing relationships between input and output
  - Robustness to environment variations
  - Generalizability to other plants
- Good-enough Models may be difficult or impossible to derive



Close-Loop Controller

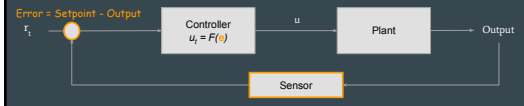
- Incorporates feedback to the Controller
  - Knows impacts of actions
  - Diffs setpoint and sensed output
  - Aims to make that difference zero

Close-Loop Controller

- Incorporates feedback to the Controller
  - Knows impacts of actions
  - Diffs setpoint and output
  - Aims to make that difference zero

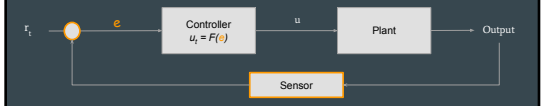
Close-Loop Controller

- Incorporates feedback to the Controller
  - Knows impacts of actions
  - Diffs setpoint and sensed output
  - Aims to make that difference zero



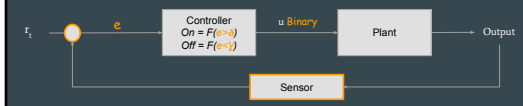
Close-Loop Controller

- Incorporates feedback to the Controller
  - Knows impacts of actions
  - Diffs setpoint and sensed output
  - Aims to make that difference zero

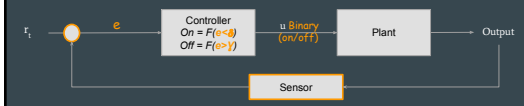


Close-Loop Controller: Bang-Bang

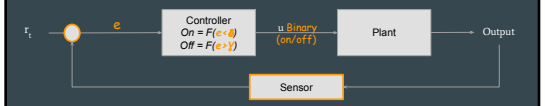
- Incorporates feedback to the Controller
  - Knows impacts of actions
  - Diffs setpoint and sensed output
  - Aims to make that difference zero



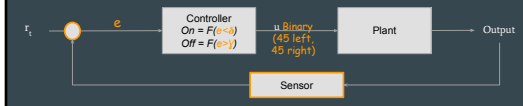
Close-Loop Controller: Bang-Bang



Close-Loop Controller: Bang-Bang



Close-Loop Controller: Bang-Bang



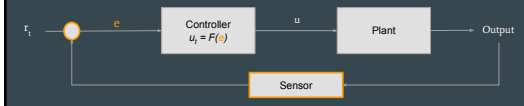
Close-Loop Controller: Proportional

- Objective: adjust based on magnitude of error

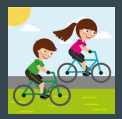
  $F(e) = K_p (e_t)$

  $= K_p (r_t - o_t)$
- Example

  $= 0.5$ (Setpoint - V_t)

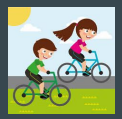


Close-Loop Controller: Proportional Example

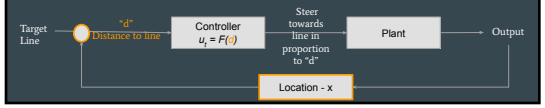


- Drive over straight line
- Process
  - Observe line
  - Compute wheel misalignment
  - Change steering angle proportional to misalignment

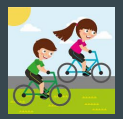
Close-Loop Controller: Proportional Example

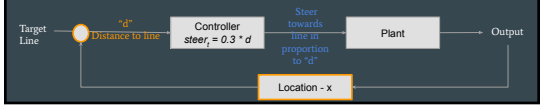


- Drive over straight line
- Process
  - Observe line
  - Compute wheel misalignment
  - Change steering angle proportional to misalignment

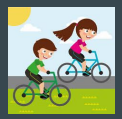


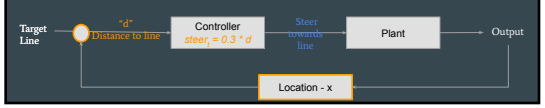
Close-Loop Controller: Proportional Example

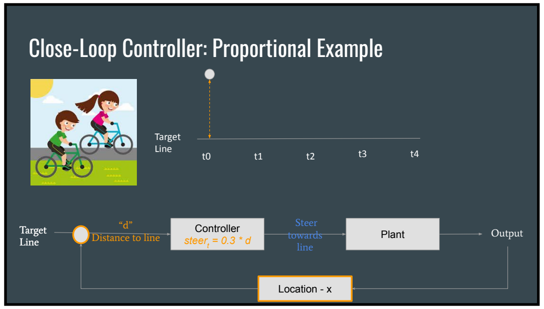


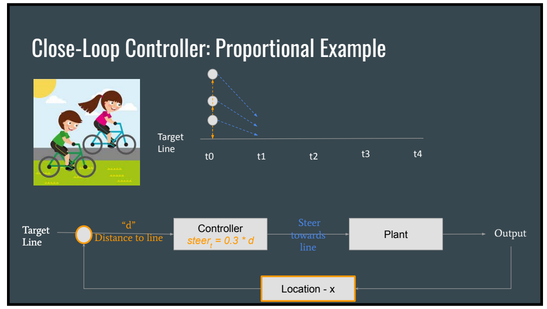


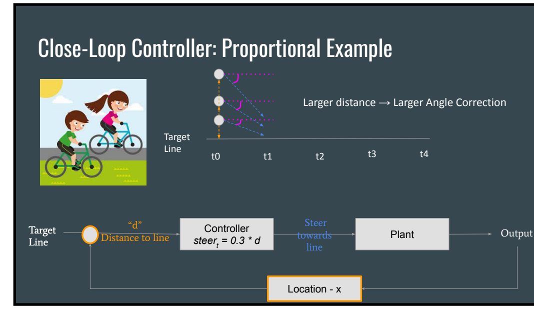
Close-Loop Controller: Proportional Example

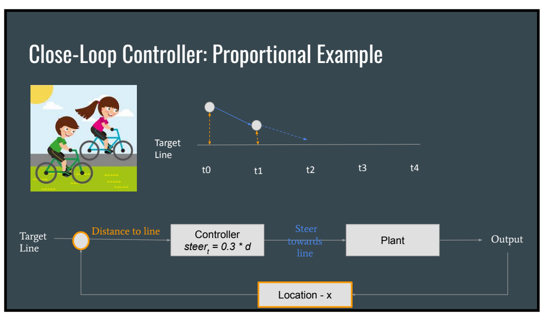


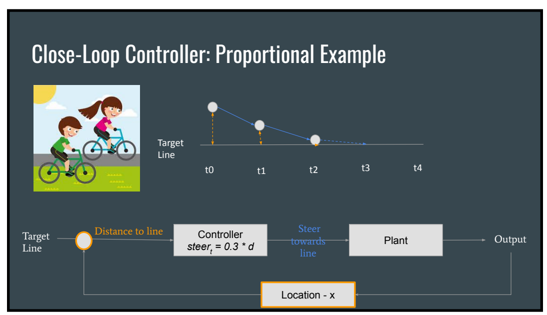


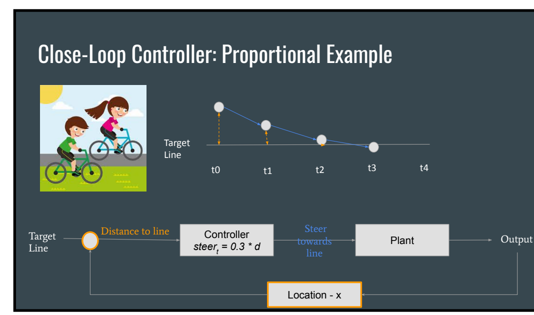

Close-Loop Controller: Proportional Example
Target Line
t0 t1 t2 t3 t4
Target Line
"d"
Distance to line
Controller
$steer_t = 0.3 * d$
Steer towards line
Plant
Output
Location - x


Close-Loop Controller: Proportional Example
Target Line
t0 t1 t2 t3 t4
Target Line
"d"
Distance to line
Controller
$steer_t = 0.3 * d$
Steer towards line
Plant
Output
Location - x

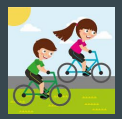

Close-Loop Controller: Proportional Example
Larger distance → Larger Angle Correction
Target Line
t0 t1 t2 t3 t4
Target Line
"d"
Distance to line
Controller
$steer_t = 0.3 * d$
Steer towards line
Plant
Output
Location - x

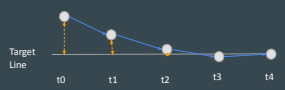

Close-Loop Controller: Proportional Example
Target Line
t0 t1 t2 t3 t4
Target Line
Distance to line
Controller
$steer_t = 0.3 * d$
Steer towards line
Plant
Output
Location - x

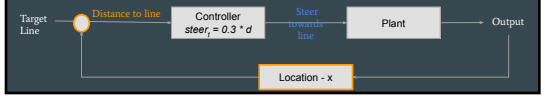

Close-Loop Controller: Proportional Example
Target Line
t0 t1 t2 t3 t4
Target Line
Distance to line
Controller
$steer_t = 0.3 * d$
Steer towards line
Plant
Output
Location - x


Close-Loop Controller: Proportional Example
Target Line
t0 t1 t2 t3 t4
Target Line
Distance to line
Controller
$steer_t = 0.3 * d$
Steer towards line
Plant
Output
Location - x

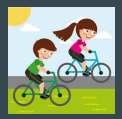
Close-Loop Controller: Proportional Example

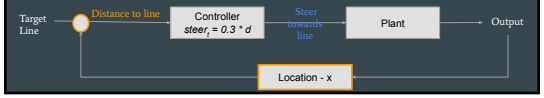






Close-Loop Controller: Proportional Example

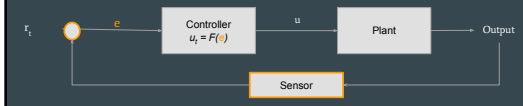




Exercise: Develop Proportional Controller for Car Cruise Control

Plant:
Set point (rt):
Input to Plant (u):
Output of Plant (y):
Sensor:

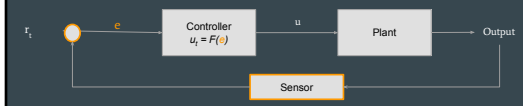
Expected Disturbances:



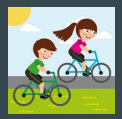
Exercise: Develop Proportional Controller for Car Cruise Control

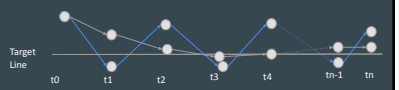
Plant: *Engine*
Set point (rt): *target speed*
Input to Plant (u): *torque*
Output of Plant (y): *vel/acc*
Sensor: *velocimeter*

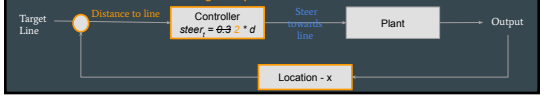
Expected Disturbances: *hills, turns, traffic*



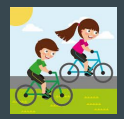
Close-Loop Controller: Proportional Example

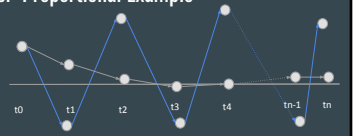


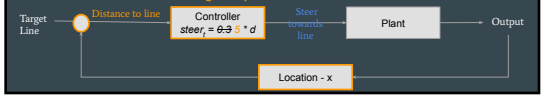




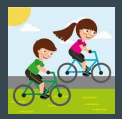
Close-Loop Controller: Proportional Example

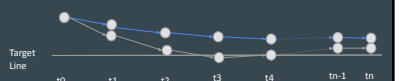


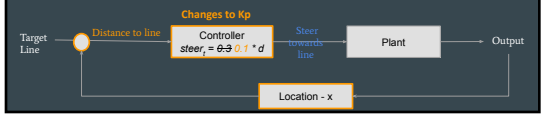




Close-Loop Controller: Proportional Example



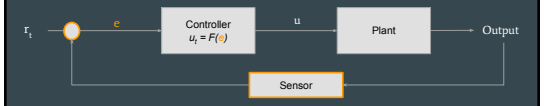




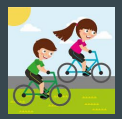
Close-Loop Controller: Proportional Derivative

- Objective: reduce oscillation
- Adjust input based on rate of output change
  - If too slow, increase input
  - If too fast, decrease input

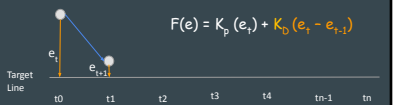
$F(e) = K_p (e_t) + K_D (e_t - e_{t-1})$



Close-Loop Controller: Proportional Derivative Example

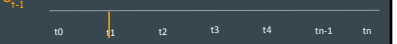


$F(e) = K_p (e_t) + K_D (e_t - e_{t-1})$

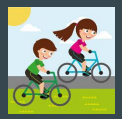


- Error is reducing from t0 to t1
- Derivative term is negative to counter Proportional term

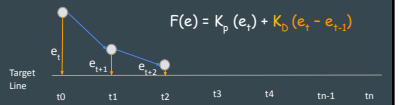
$e_t - e_{t-1}$



Close-Loop Controller: Proportional Derivative Example

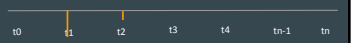


$F(e) = K_p (e_t) + K_D (e_t - e_{t-1})$

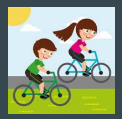


- Error is reducing from t0 to t1 to t2
- Derivative term is still negative
- Derivative term becomes smaller as amount of error decreases

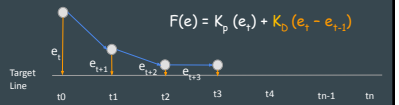
$e_t - e_{t-1}$



Close-Loop Controller: Proportional Derivative Example



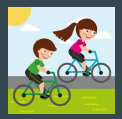
$F(e) = K_p (e_t) + K_D (e_t - e_{t-1})$

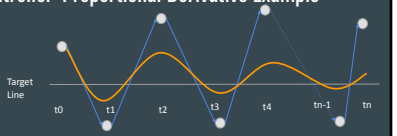


- Error is constant
- Derivative term is zero
- Only proportional term correction

$e_t - e_{t-1}$

Close-Loop Controller: Proportional Derivative Example





D term damps the aggressiveness of P

Proportional to error growth

Exercise: Develop PD Controller for Altitude Controller

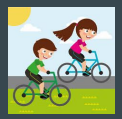
Plant:
Set point (rt):
Input to Plant (u):
Output of Plant (y):
Sensor:

Close-Loop Controller: Proportional + Derivative + Integral

- Objective: reduce steady state error
- Sum total error over time (potential for overcompensation)

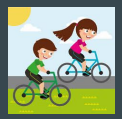
$F(e) = K_p (e_t) + K_D (e_t - e_{t-1}) + K_I (e_0 + e_1 + e_2 + \dots + e_{t-1})$

Close-Loop Controller: Proportional + Integral Example



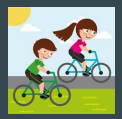
- Steady-State error is the final difference with setpoint
  - P gets to stable point that is deemed too far from setpoint
- Caused by disturbances
  - Gravity
  - More friction turning right than left
  - Leaning certain way

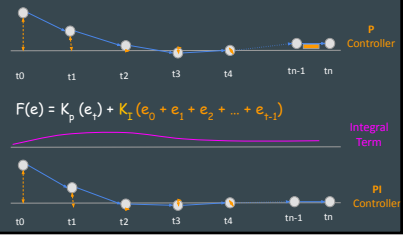
Close-Loop Controller: Proportional + Integral Example



$F(e) = K_p (e_t) + K_I (e_0 + e_1 + e_2 + \dots + e_{t-1})$

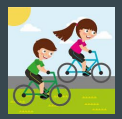
Close-Loop Controller: Proportional + Integral Example





$F(e) = K_p (e_t) + K_I (e_0 + e_1 + e_2 + \dots + e_{t-1})$

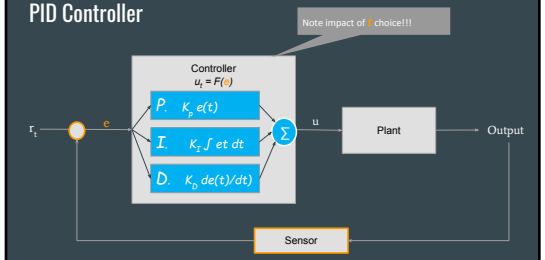
Close-Loop Controller: Proportional + Integral Example



$F(e) = K_p (e_t) + K_I (e_0 + e_1 + e_2 + \dots + e_{t-1})$

Integral Windup curse
- Integral term increases while output is ramping up
- This can cause overshoot and oscillation
- Solution is to limit integral term

PID Controller



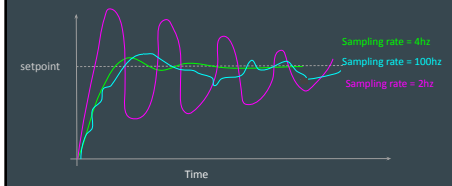
In Code

```
float setpoint = read()
float lasterr = 0;
float integral = 0;

float  PIDcontroller (float measure) {
       err = setpoint - measure;
       dt = currentTime - lastTime;
      integral += err * dt;
       float deriv = (err - lasterr ) / dt;
       float output = Kp*err + Ki*integral + Kd*deriv;
       lasterr = err;
       lastTime = currentTime
       return output;
}
```

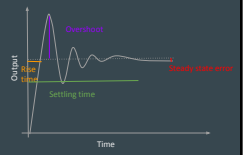
Missing
- Definition of K coefficients
- Bounds on output
- Bounds/reset integral term

Caveat: Tuning depends on Sampling Rate



Controller Performance

- Stability
  - Error should converge to within threshold
  - No oscillation
- Performance
  - Rise time - within threshold of steady state
  - Overshoot - over final value
  - Settling time - time before output within threshold
- Robustness
  - Stability and performance variations in the presence of plant changes

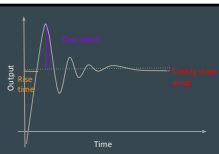


Tuning PID

- Many heuristics, my favorite
  - Initialize Kd = Ki = 0
  - Iterate
    - Increase Kp until oscillation
    - Decrease Kp by 2
    - Increase Ki until just before loss of stability
    - Increase Kd to reduce oscillation

Tuning PID

Debugging / Trade-offs present through subtle interactions



Effect / Increasing Term	Rise Time	Overshoot	SS error
Proportional	decrease	increase	decrease
Integral	decrease	increase	eliminate
Derivative		decrease	

Takeaways

- Controllers can
  - Make your robot respond faster
  - Abstracts physics away from desired response
- Close-loop
  - Feedback helps to adjust/tolerate unexpected world
- PID Controllers
  - Most controllers in the world, simple, effective
  - Setting K constants and sampling time are the keys!