F1TENTH Autonomous Racing

Reactive Methods for Navigation

PID for Wall-following

Rahul Mangharam
University of Pennsylvania
rahulm@seas.upenn.edu

F1 TENTH

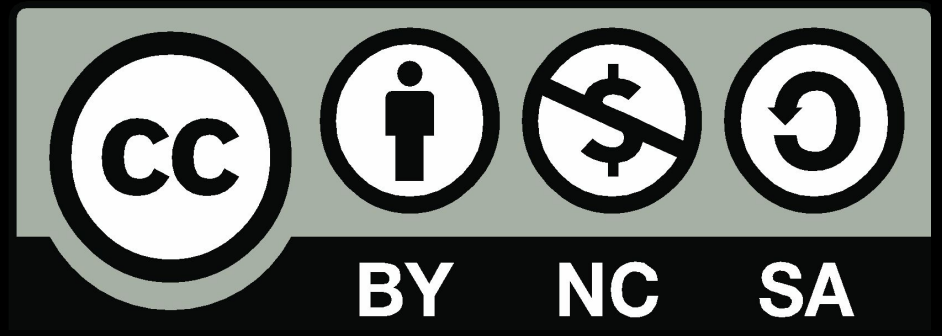
Safe Autonomy Lab
University of Pennsylvania

Penn Engineering

Acknowledgements

This course is a collaborative development with significant contributions from:
Hongrui Zheng (lead), Matthew O'Kelly (lead), Johannes Betz (lead), Madhur Behl (lead), Joseph Auckley, Luca Carlone, Jack Harkins, Paril Jain, Kuk Jang, Sertac Karaman, Dhruv Karthik, Nischal KN, Thejas Kesari, Venkat Krovi, Matthew Lebermann, Kim Luong, Yash Pant, Varundev Shukla, Nitesh Singh, Siddharth Singh, Rosa Zheng and many others.

We are grateful for learning from each other



Lesson Plan

Part I: Tracking a reference signal

Part II: PID control for wall following

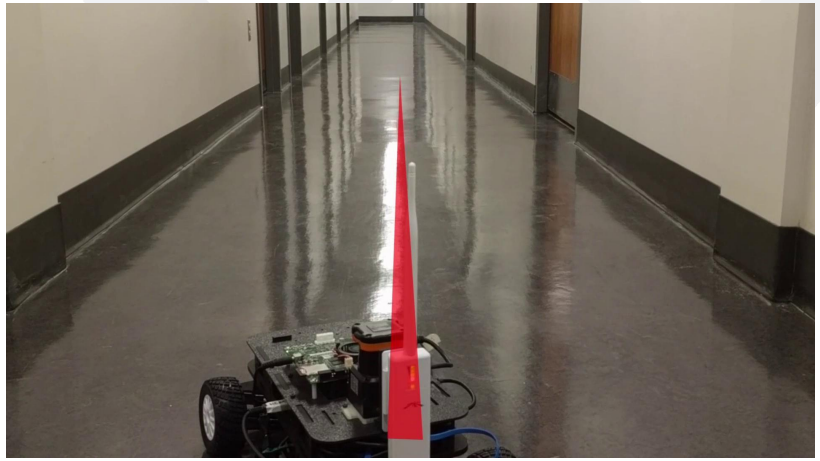
Part III: Application to the F1Tenth race car

Learn to Drive: Tracking a Reference Signal

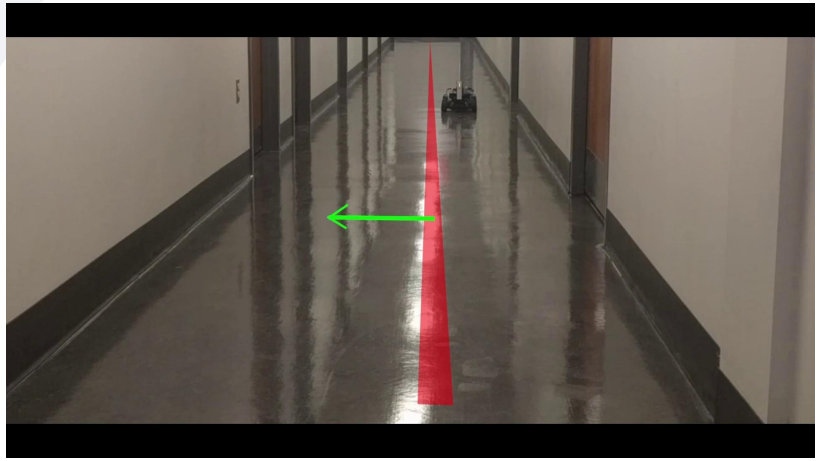
Learn to drive straight along a centerline

Effective tuning of the steering and speed controllers is crucial for good performance.

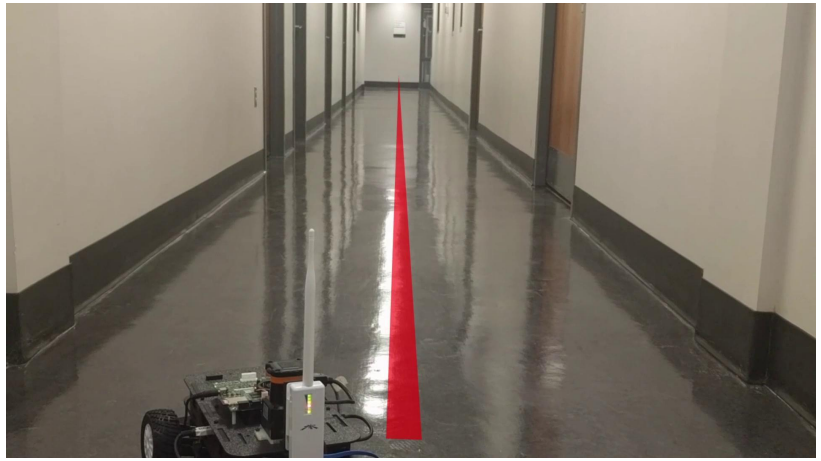
$K_P=14$, $K_i=0.01$, $K_D=0$



$K_P=5$, $K_i=0$, $K_D=0.09$



$K_P=14$, $K_i=0$, $K_D=0.09$



Underdamped Performance

Learn to Race: Wall-following

Challenge:
How can we drive the car around the track

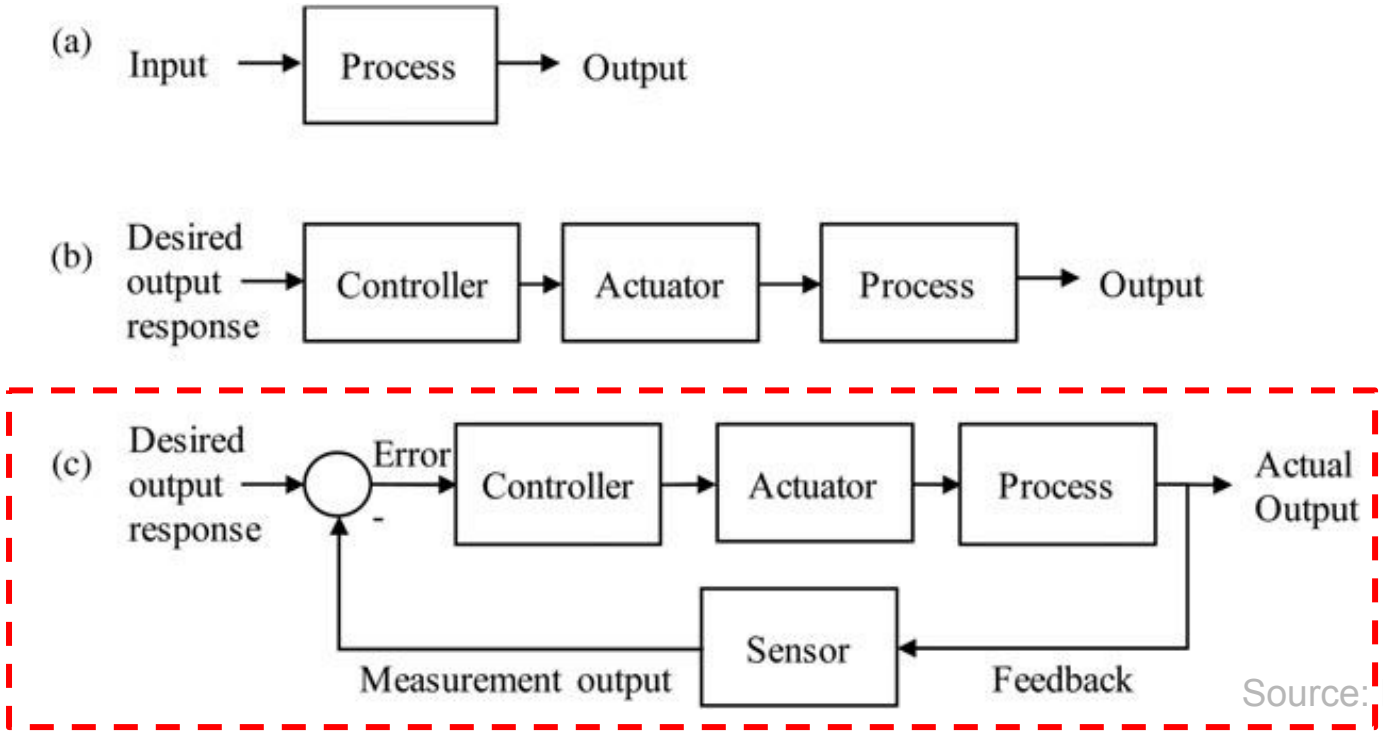
Learning Outcome:
Basics of PID, how to compute error, failure modes.

Assignment:
Wall following in simulation and on the vehicle.

Tracking a reference signal

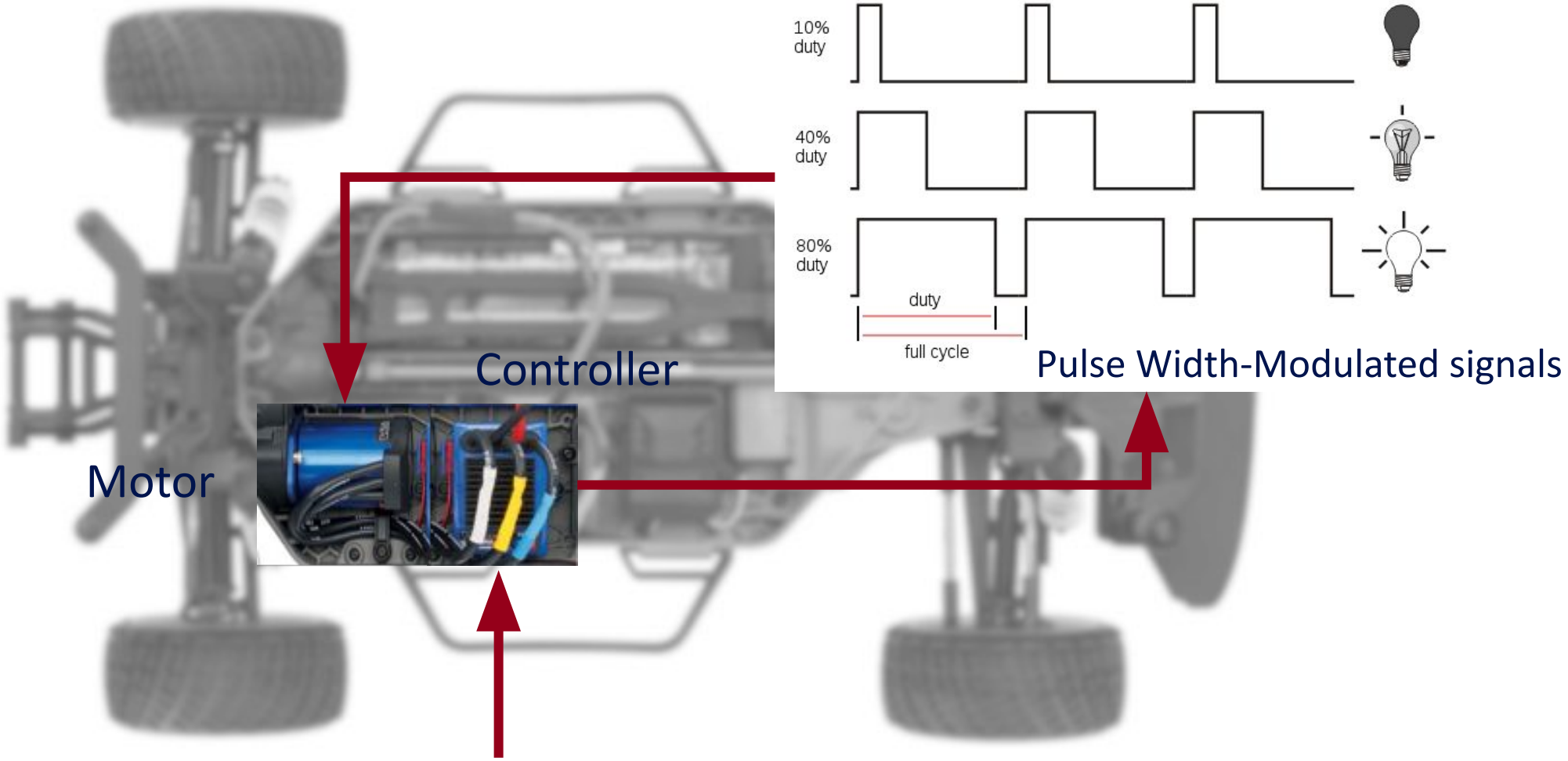
Control Engineering

- **Control Engineering:** We focus on modeling a diverse range of dynamic systems (e.g. mechanical system, electrical systems)
- We will design **controllers** that will cause these systems to behave in the desired manner



Source: Researchgate

Tracking a Reference Signal: Engine Speed



Control System: Engine Speed

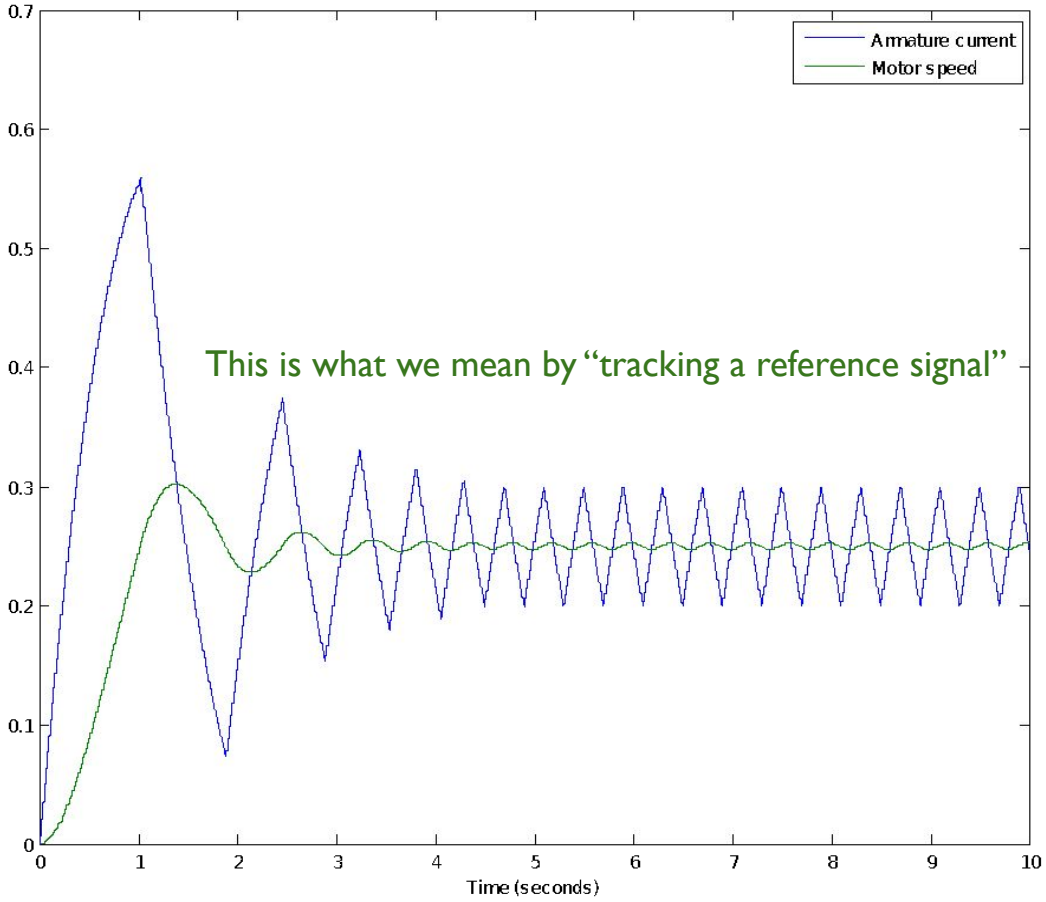
Reference
r(t)
("60rpm")

CONTROLLER

PLANT

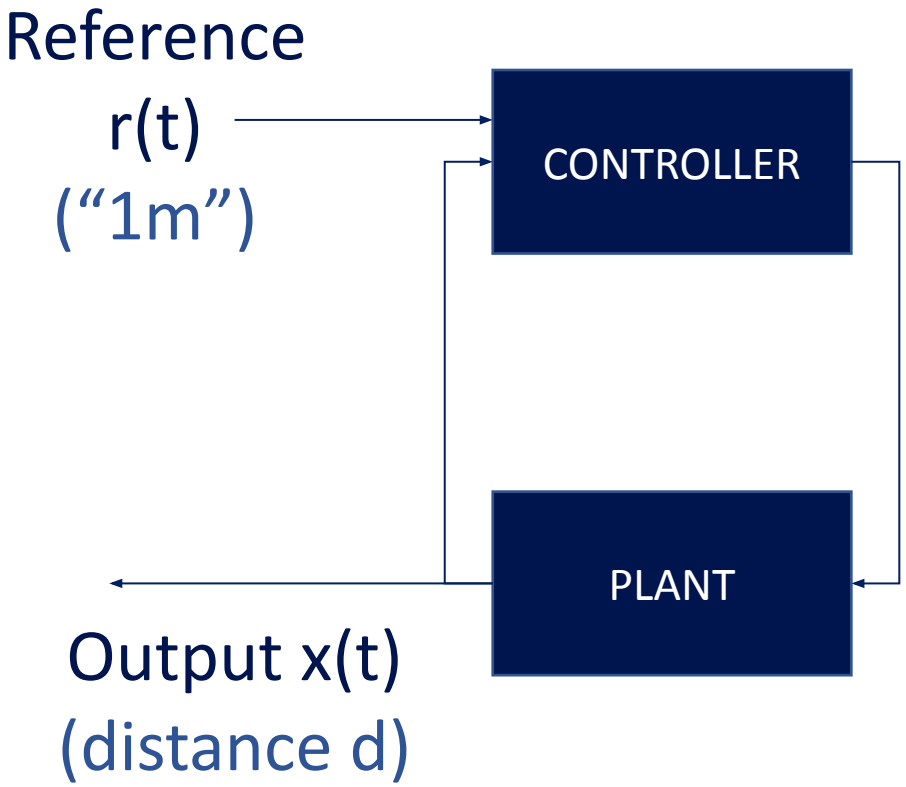
Output x(t)
(angular velocity)

Control input
u(t)
(electric current)

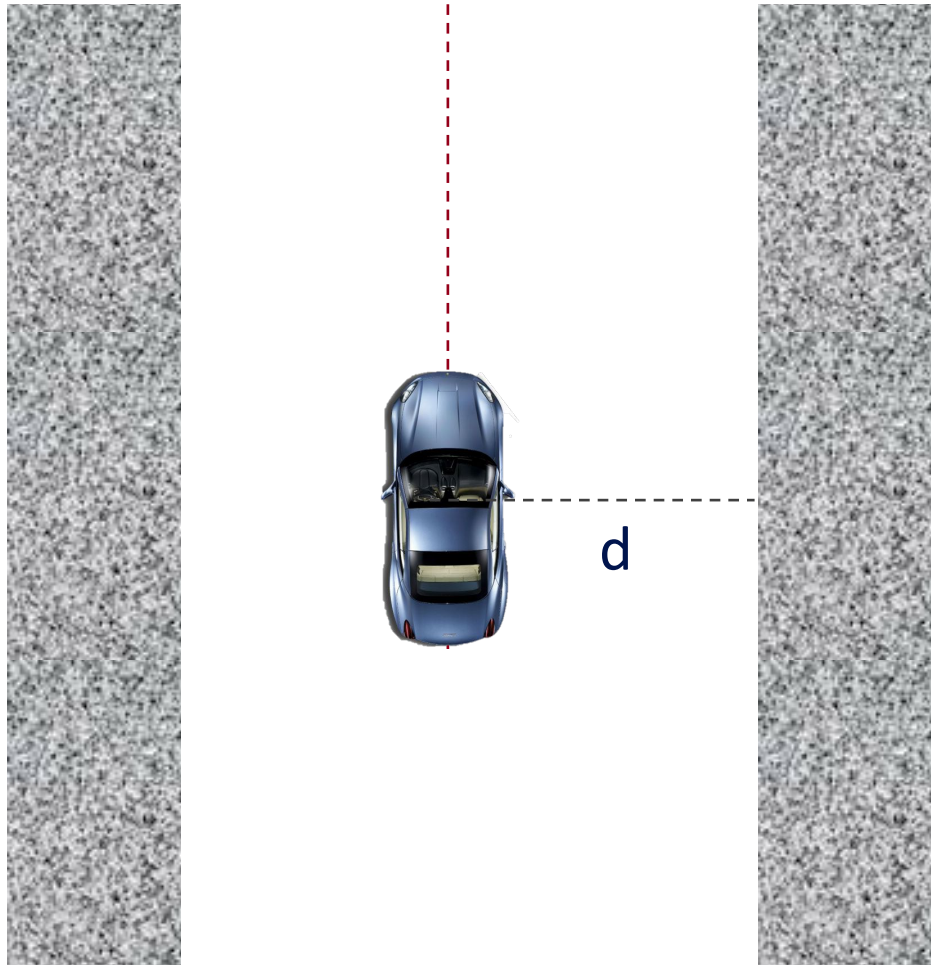


Penn Engineering

Control System: Path Tracking

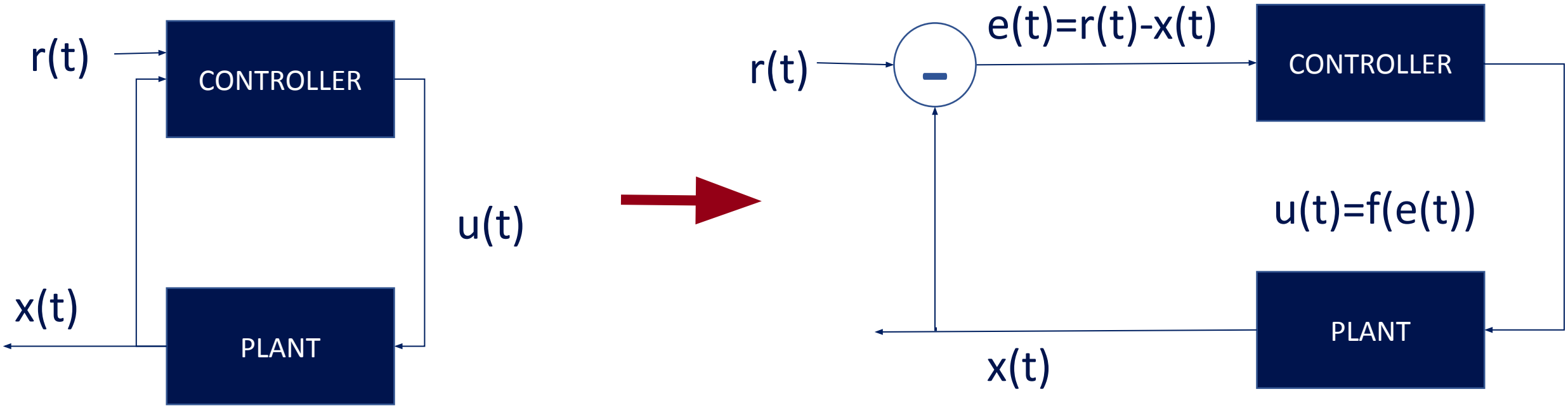


Control input
u(t)
(steering rate)



Penn Engineering

Control System: Path Tracking



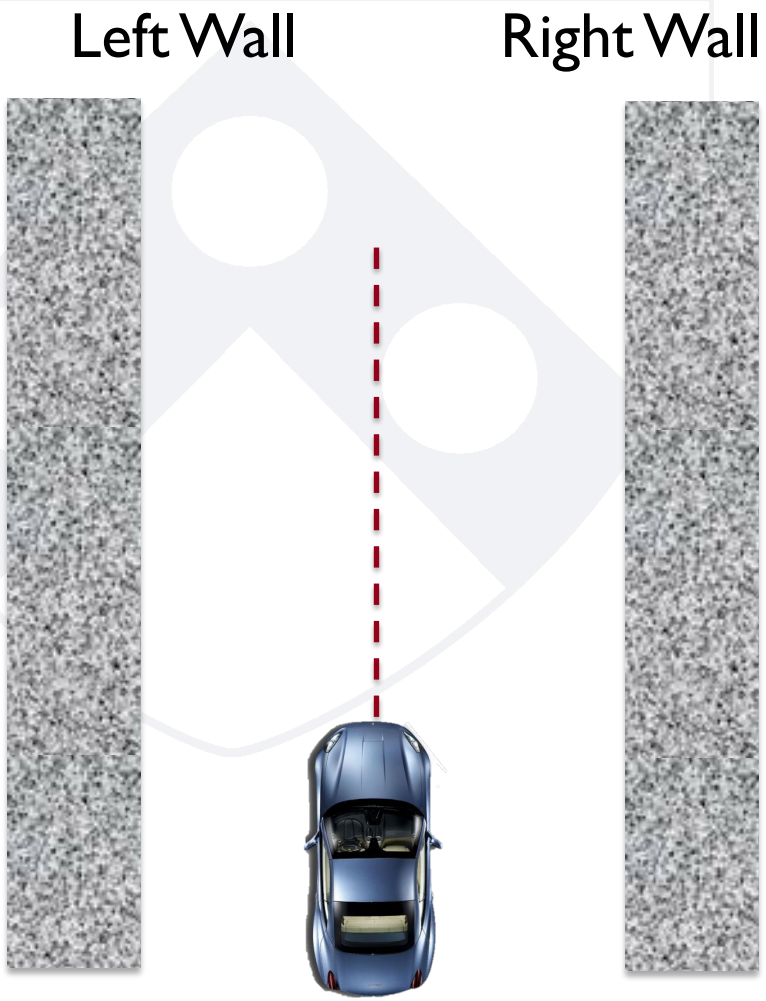
Enhancing of the control system with an feedback error e(t)

PID control for Wall Following

Proportional, **I**ntegral and **D**erivative control

PID control: objectives

For advancements we can use MPC(Model Predictive Controller), which is an advance controller than PID

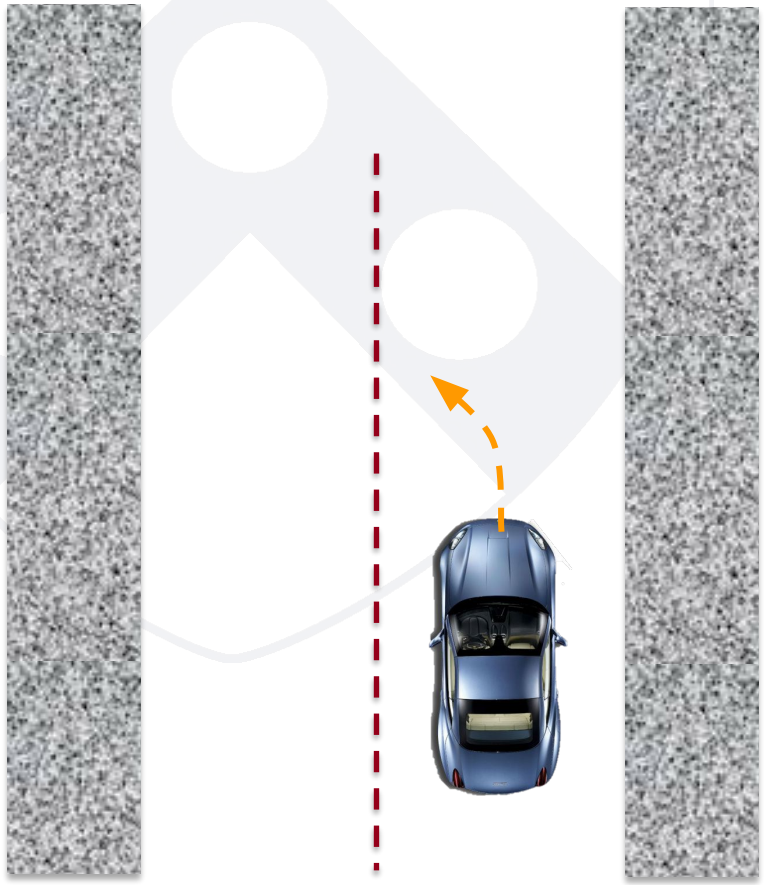


Control objective:

1) keep the car driving along the centerline

2) parallel to the walls.

PID control: objectives

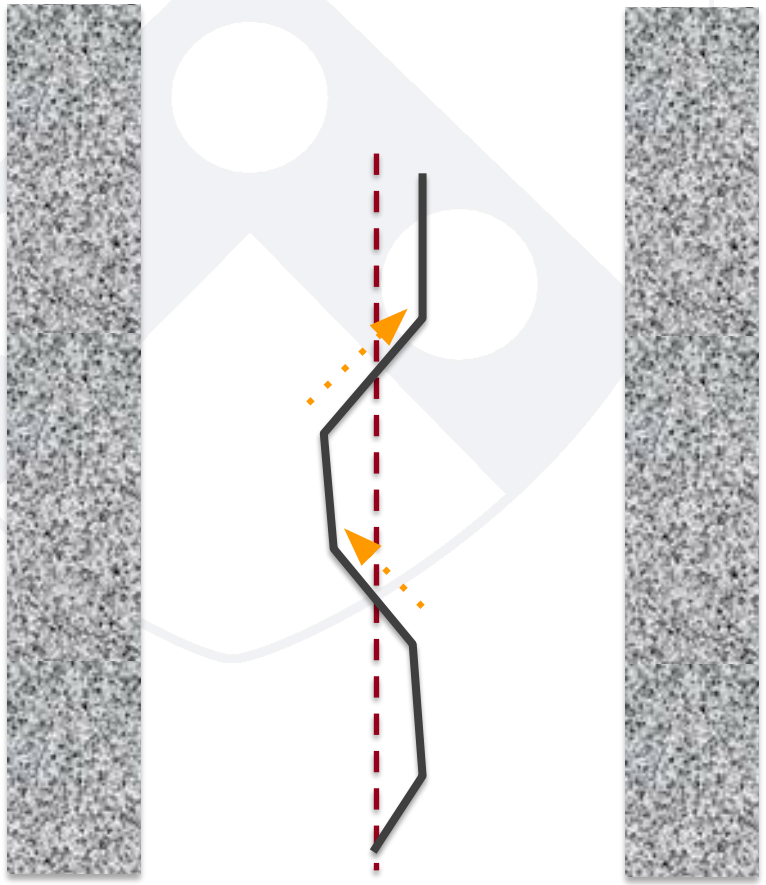


Control objective:

~~1) keep the car driving along the centerline,~~

2) parallel to the walls.

PID control: objectives

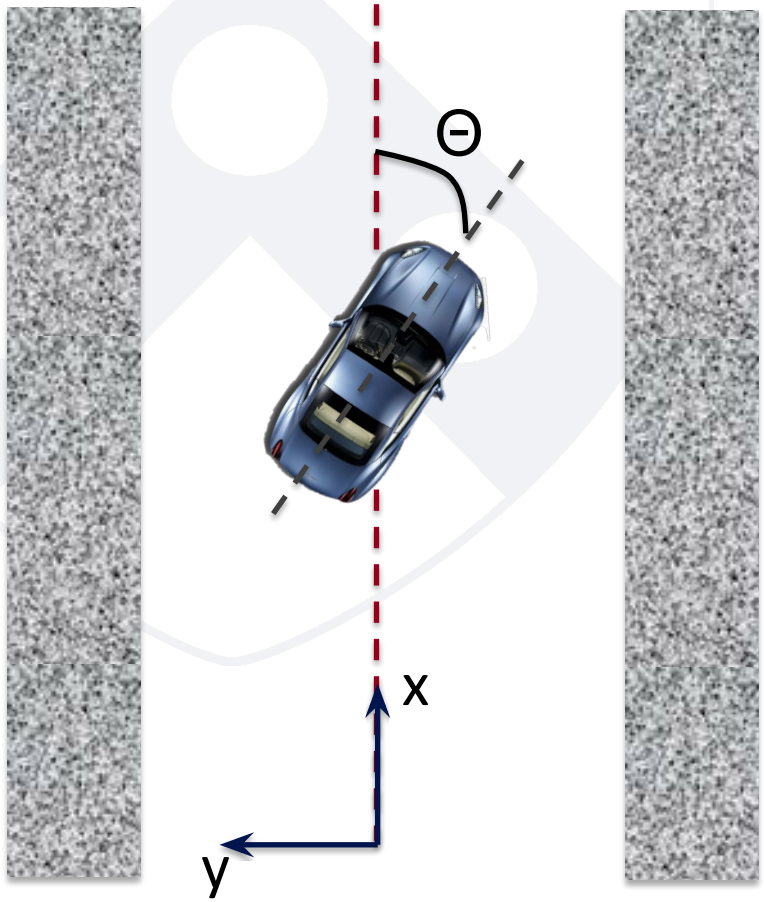


Control objective:

1) keep the car driving (roughly) along the centerline,

~~2) parallel to the walls.~~

PID control: control objectives



Control objective:

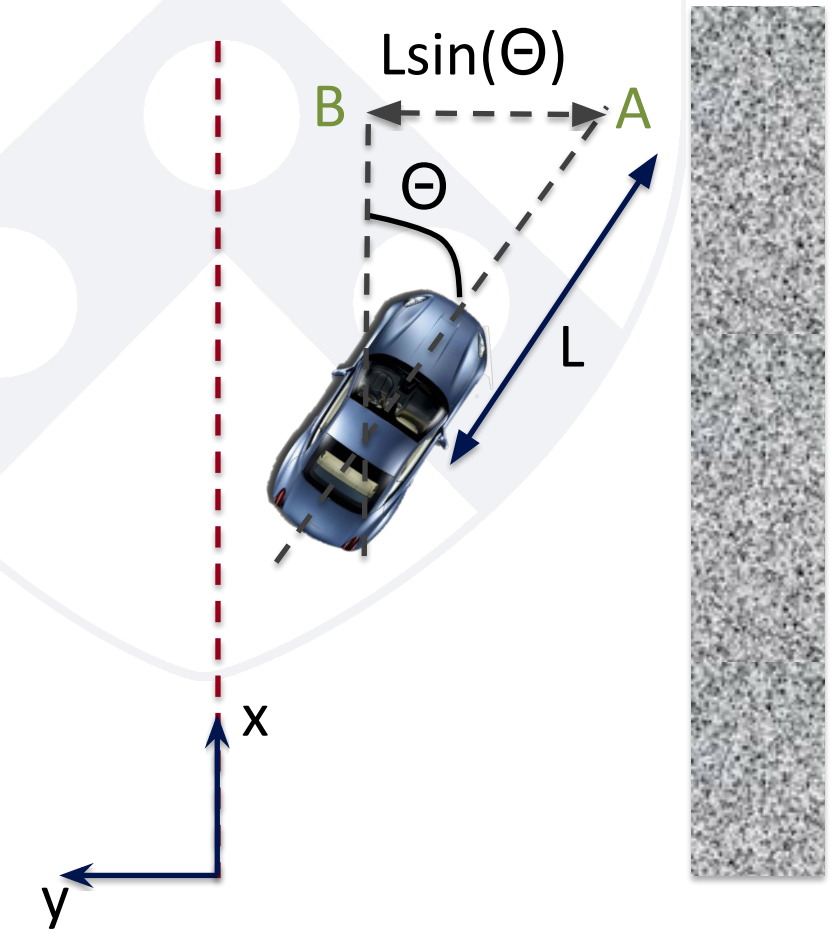
1) keep the car driving along the centerline,

$$y = 0$$

2) parallel to the walls.

$$\Theta = 0$$

PID control: control objectives



Control objective:

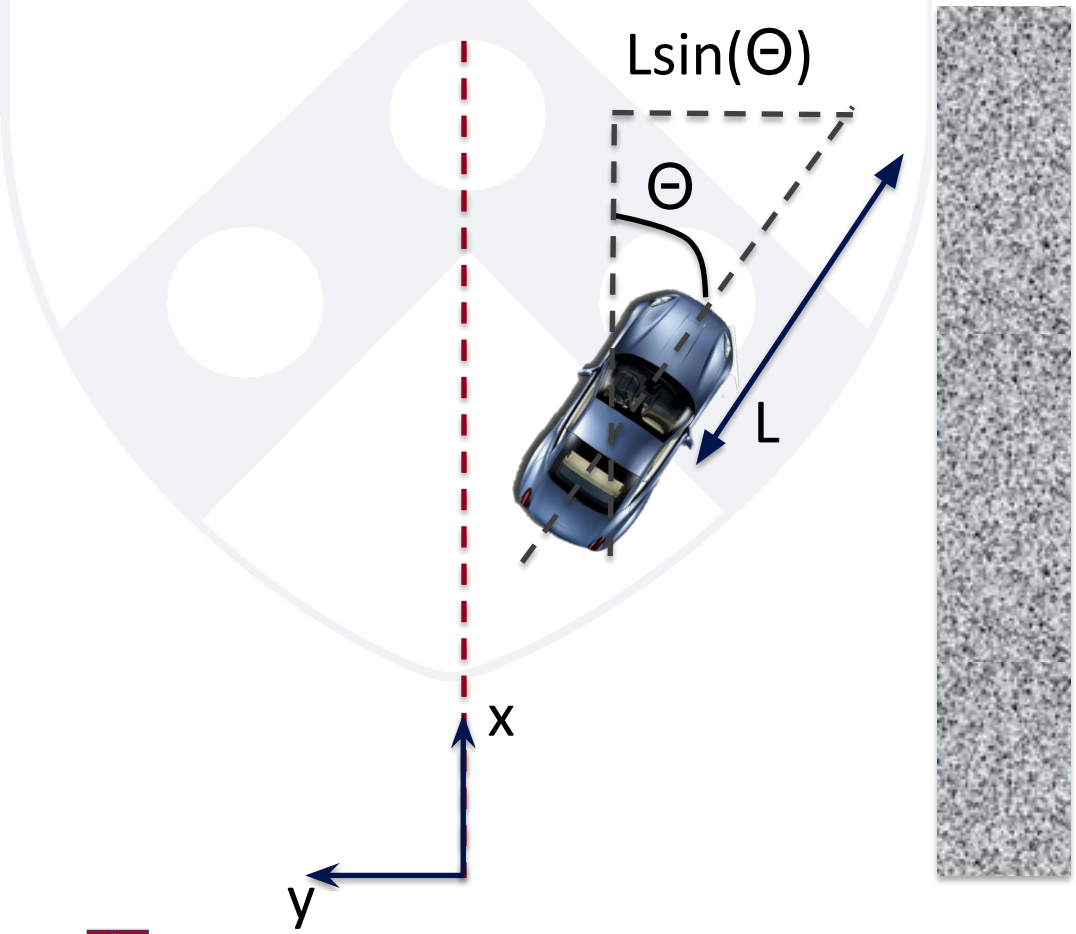
1) keep the car driving along the centerline,

$$y = 0$$

2) After driving L meters, still on the centerline:
Horizontal distance after driving L meters

$$L\sin(\Theta) = 0$$

PID control: control inputs



Control input:
Steering angle Θ

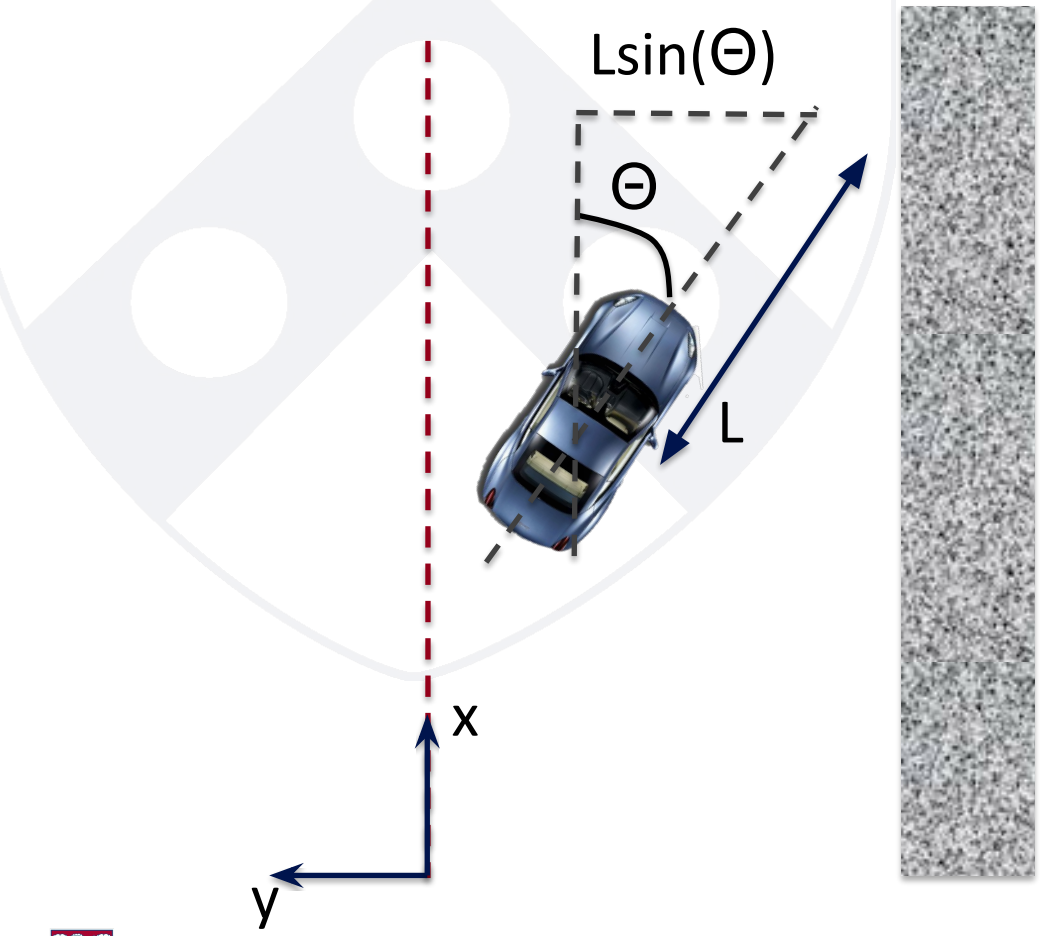
We will hold the velocity constant.

How do we control the steering angle to keep

$$y = 0, L\sin(\Theta) = 0$$

as much as possible?

PID control: error term

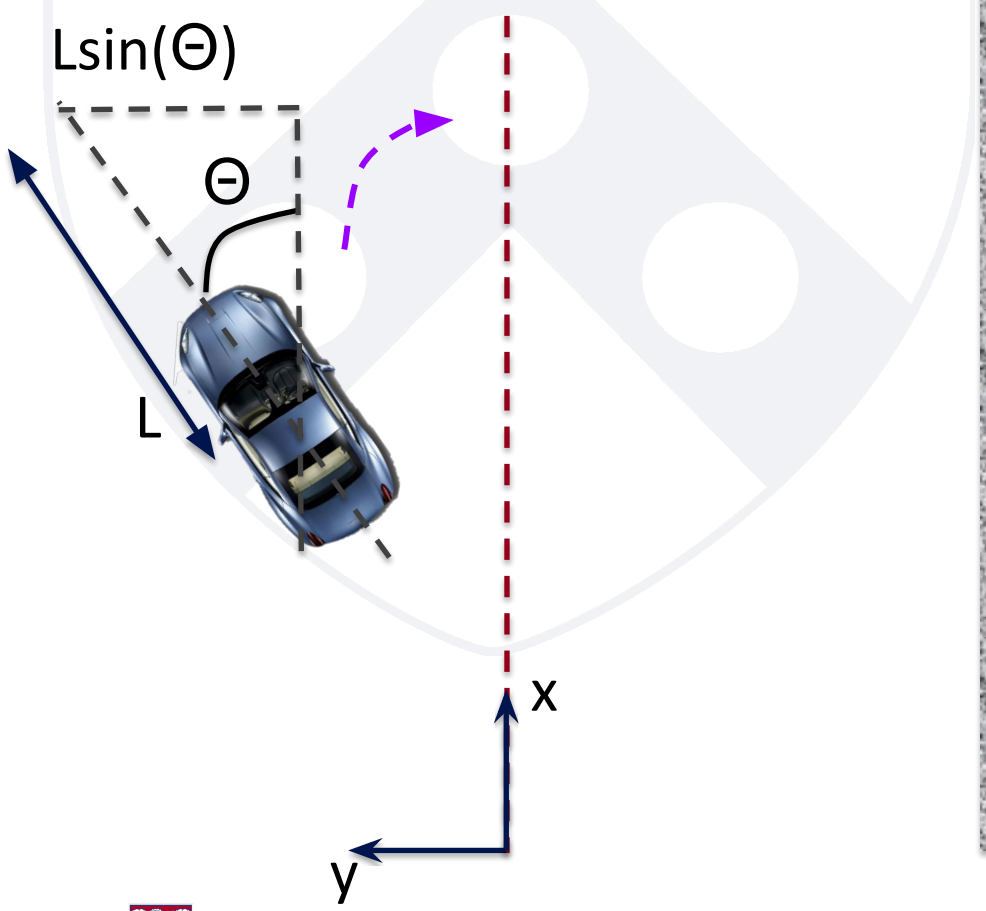


Want both y and Lsin(Θ) to be zero

- Error term e(t) = -(y + Lsin(Θ))

We'll see why we added a minus sign

PID control: computing input

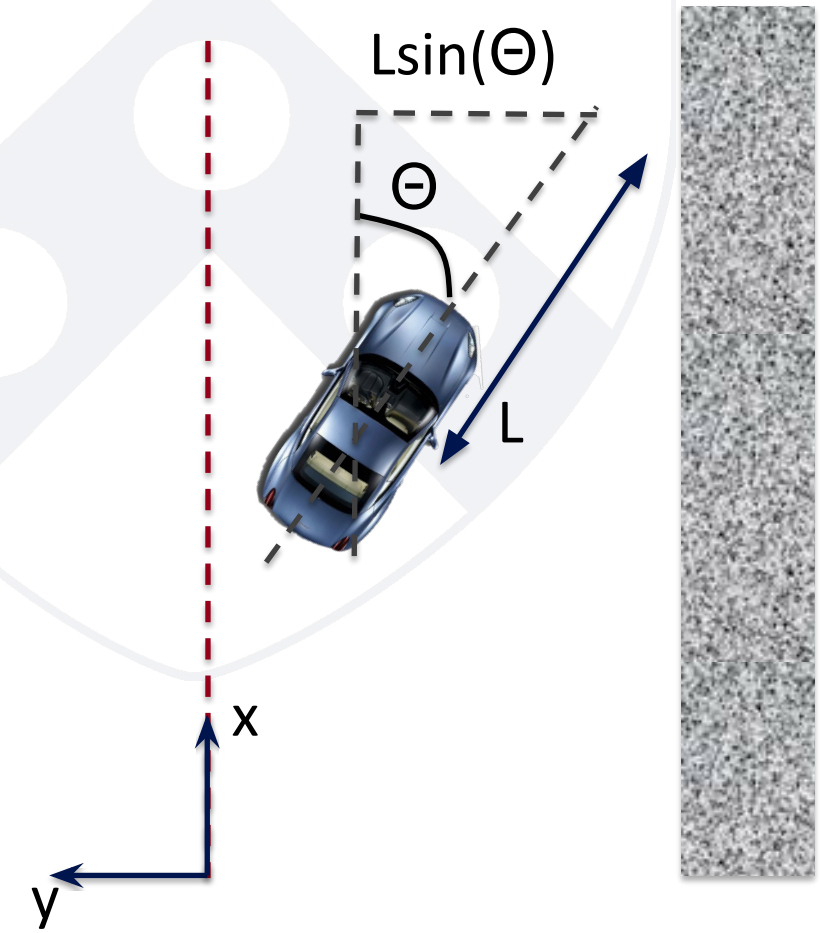


When y > 0, car is to the left of centerline
→ Want to steer right: Θ < 0

When Lsin(Θ) > 0, we will be to the left of centerline in L meters
→ Want to steer right: Θ < 0

Set *desired* angle to be
$\Theta_d = K_p\ (-y - L\sin(\Theta))$

PID control: computing input

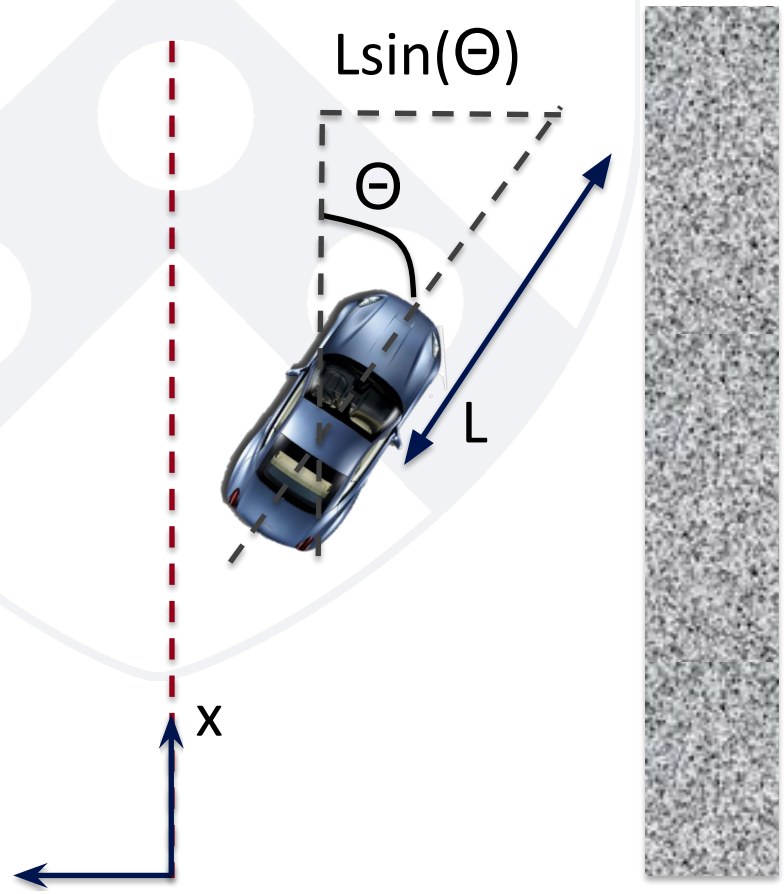


When y < 0, car is to the right of centerline
→ Want to steer left
 Want Θ > 0

When Lsin(Θ) < 0, we will be to the right of centerline in L meters, so want to steer left
→ Want Θ > 0

Consistent with previous requirement:
$\Theta_d = K_p (-y - L\sin(\Theta))$

PID control: Proportional control



$$\Theta_d = C\,K_p\,(-y - L\sin(\Theta)) = C\,K_p\,e(t)$$

This is **P**roportional control.

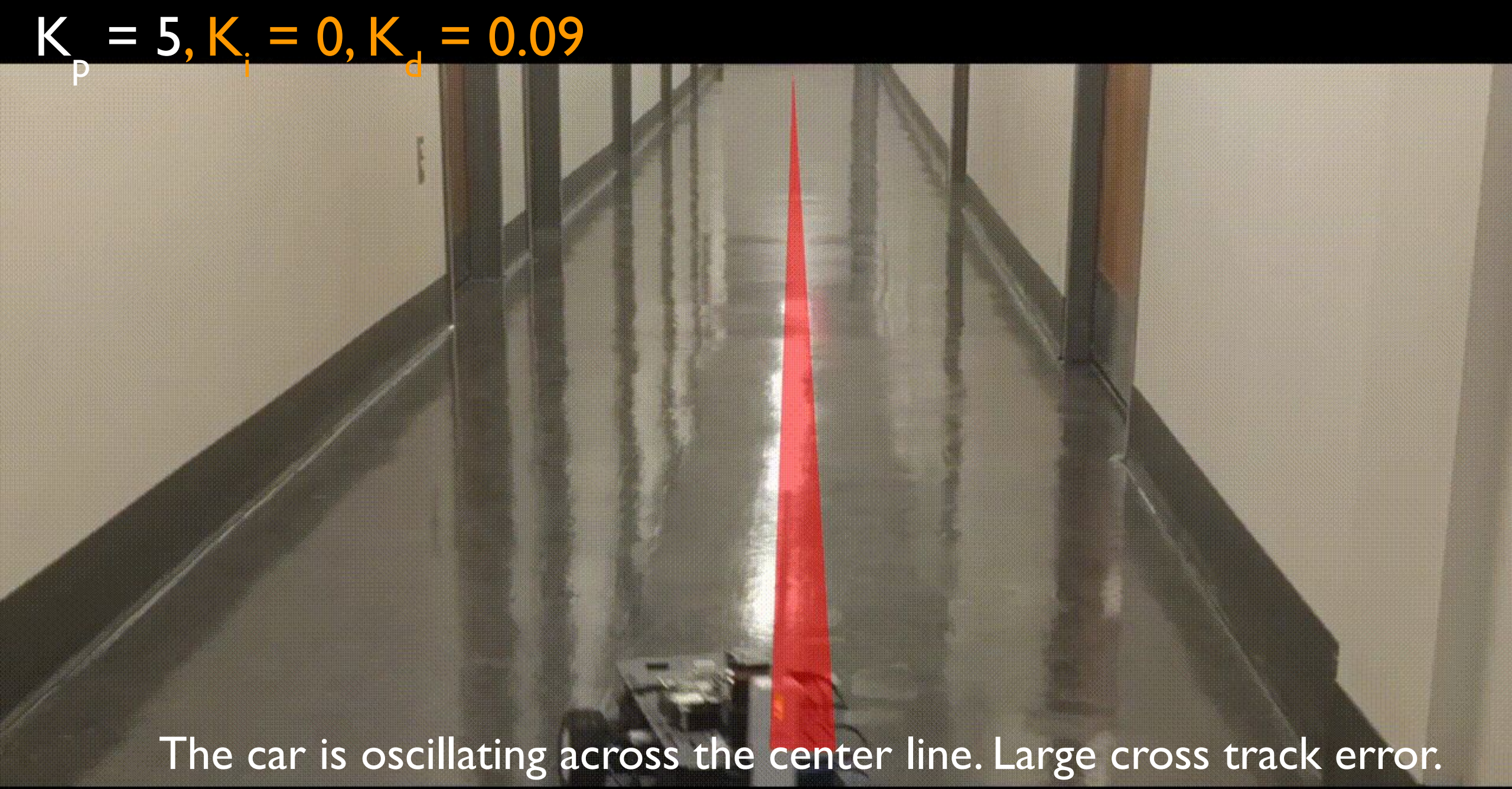
The extra C constant is for scaling distances to angles.

PID control: tuning the gains

Let's try these gains to straighten and align the car to the center line

- $K_p = 5$
- $K_i = 0$
- $K_d = 0.09$

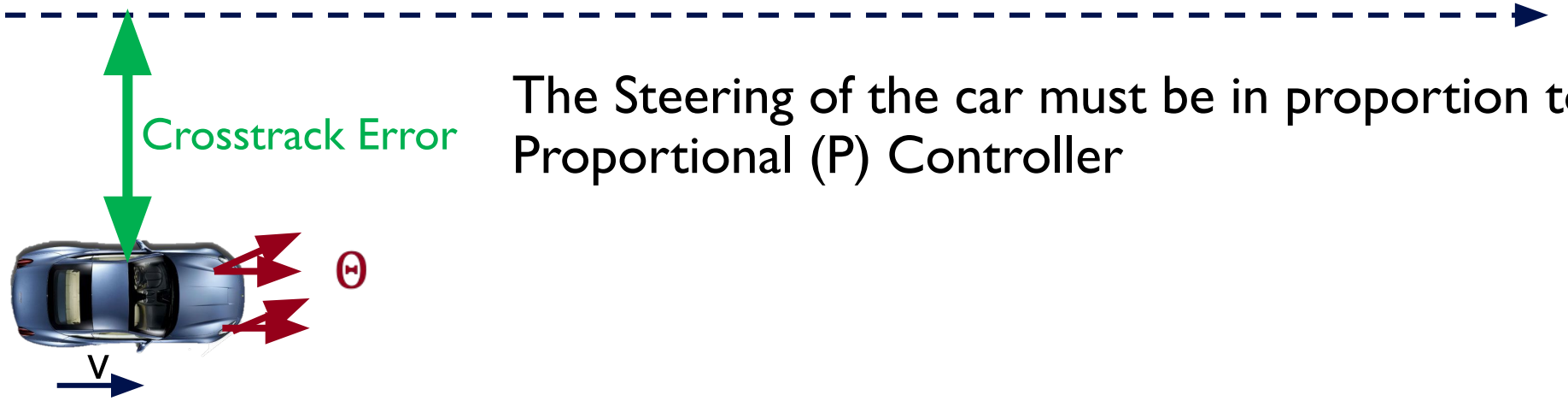
$K_p = 5, K_i = 0, K_d = 0.09$



The car is oscillating across the center line. Large cross track error.

$K_p = 5$ is less responsive to error magnitude. So let's increase K_p

Proportional Control: Summary

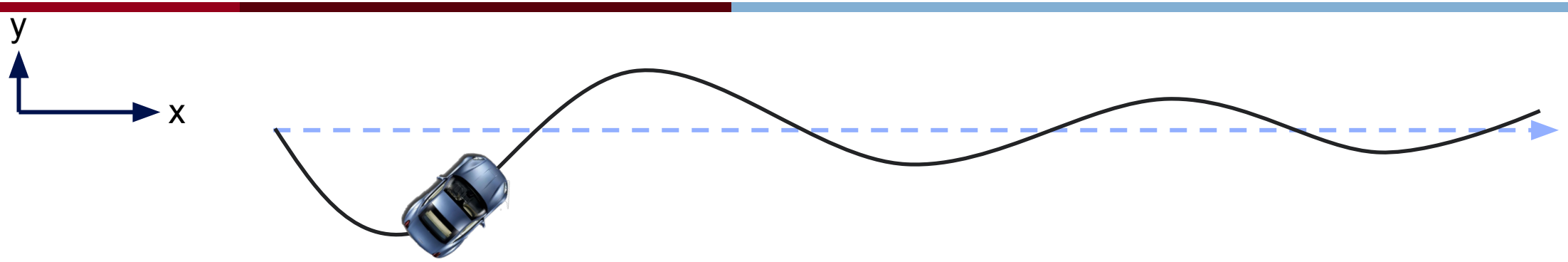


The Steering of the car must be in proportion to Cross Track Error: Proportional (P) Controller

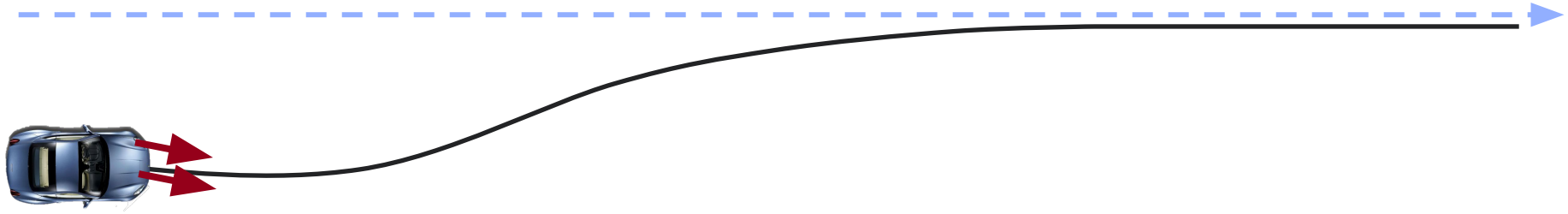
- Idea: the applied control input is proportional to the cross track error
  $\mathbf{e(t) = ref(t) - x_{actual}(t)}$
- The larger the current error e(t), the larger the applied input to correct it.

  Control Input: $\mathbf{u(t) = K_p * e(t)}$

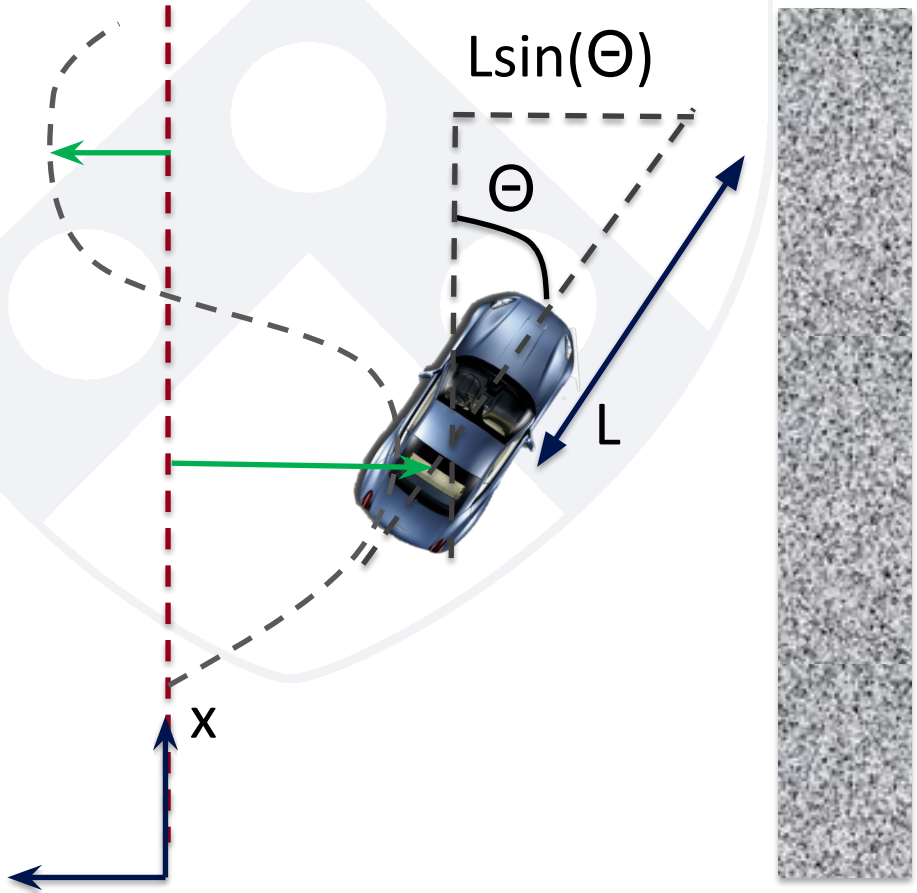
Derivative Control



- By modifying the control parameter, the car might oscillate around the centerline
- We need to avoid the overshoot of the car and we might to react quickly
- We enhance the P-Controller with a Derivate (D) part: PD-Controller



PID control: Derivative control



If error term is increasing quickly, we might want the controller to react quickly

→ Apply a *derivative gain*:

$$\Theta = K_p\, e(t) + K_d\, de(t)/dt$$

Derivative Control

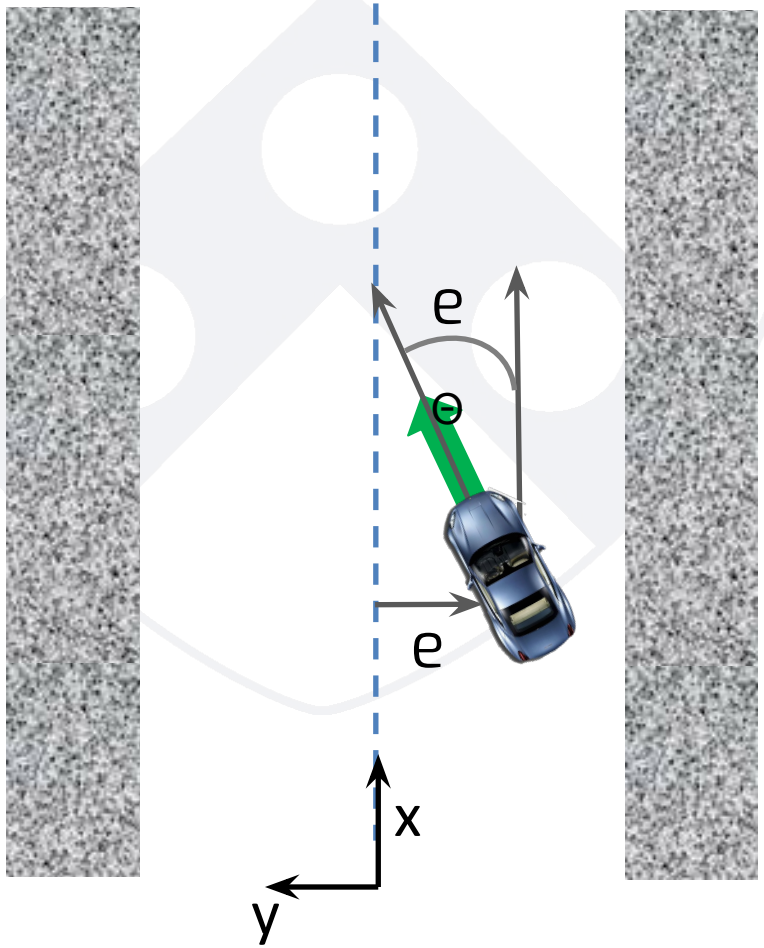
- Idea: We apply and early corrective action that will predict, which way the error is going and therefore if it is decreasing or increasing
- A simple prediction is to apply a derivative gain that is based on the error e(t):

$$K_d \frac{d}{dt} e(t)$$

- We add this new term to our steering angle calculation to setup a PD-Controller:

Desired angle to be: $\Theta_d = K_p * (-y - L*\sin(\Theta)) + K_d \frac{d}{dt} e(t)$

PID control: Derivative Gain



Derivative control doesn't come for free.
In shown scenario, position controller tries to nullify position error e_x.

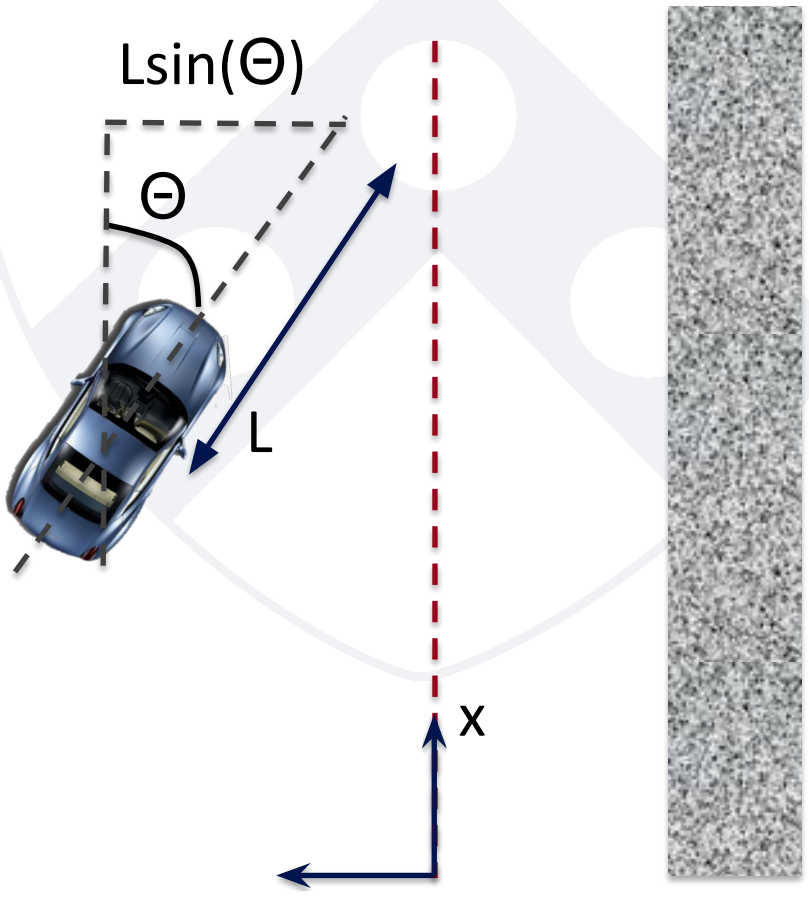
→ Increase angle Θ beyond 0

→ Angle error $e_\Theta(t) = (0 - \Theta)$ decreases (becomes "more negative") so $e_\Theta' < 0$

→ Derivative angle control $K_{d,\omega}\, e_\Theta'$ will seek to decrease the angle back to 0.

→ Trajectory correction takes longer (smoothing out)

PID control: Derivative control



Controller tries to nullify y term

→ Steer right, Θ < 0 and
y decreases → y' < 0

Steer RIGHT

→ Error derivative -y' -Lcos(Θ)>0

Steer LEFT

→ Trajectory correction takes longer (smoothing out)

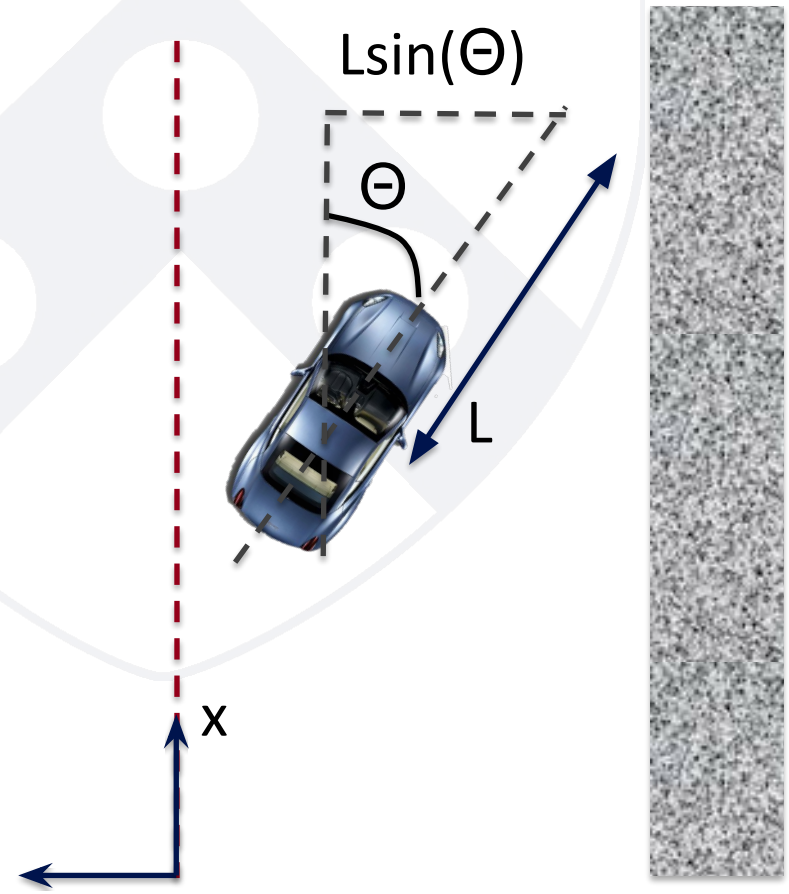
We can reduce the overshoot by adjusting the Kd term

PID control: tuning the gains

- Default set of gains, determined empirically to work well for this car.
  - $K_p = 14$
  - $K_i = 0$
  - $K_d = 0.09$


$K_p = 14, K_i = 2, K_d = 0.09$

PID control: Integral control



Integral control is proportional to the *cumulative* error

$$\Theta = K_p\, e(t) + K_I\, E(t) + K_d\, de(t)/dt$$

Where E(t) is the integral of the error up to time t (from a chosen reference time)

Integral control

- Idea: The steady-state error accumulates over time. We can apply an additional correcting input that is sensitive to the total error so far.
- We create the sum/integral of all Cross Track error e(t) we ever observed:

$$K_i \int_o^t e(t)\ dt$$

- We add this new term to our steering angle calculation to setup a PID-Controller:

$$\text{Desired angle to be: } \Theta_d = K_p * (-y - L*\sin(\Theta)) + K_d \frac{d}{dt} e(t) + K_i \int_o^t e(t)\ dt$$

PID control: tuning the gains

Include K_i - overly sensitive to accumulating error & over-correction

- $K_p = 14$
- $K_i = 2$
- $K_d = 0.09$

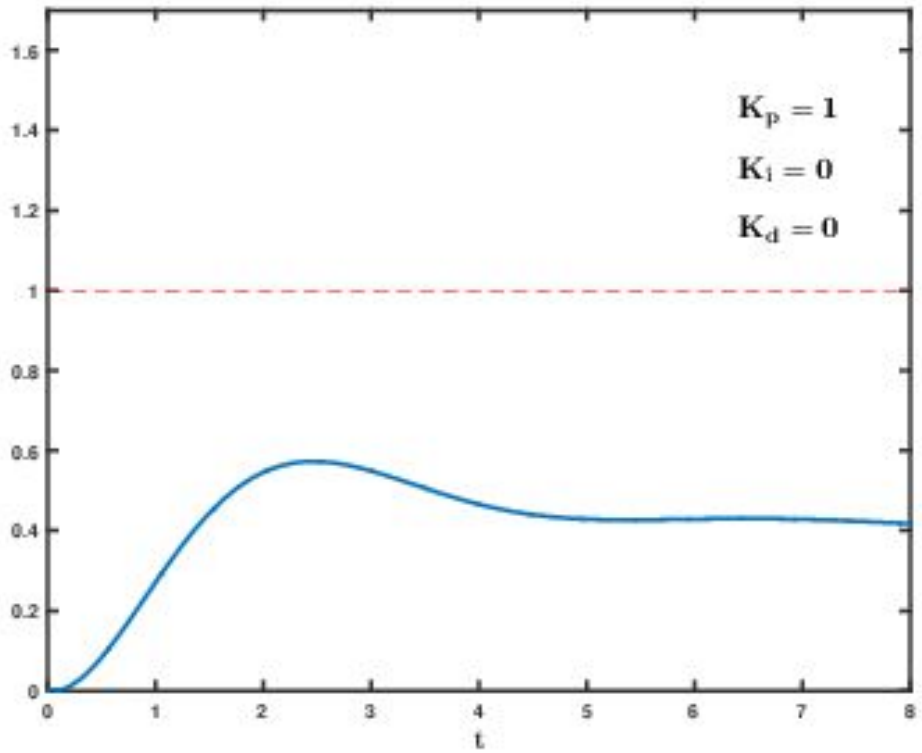

$K_p = 14, K_i = 2, K_d = 0.09$

PID Controller

$$\Theta_d = \underbrace{K_p * e(t)}_{\text{P: Proportional}} + \underbrace{K_d \frac{d}{dt} e(t)}_{\text{D: Differential}} + \underbrace{K_i \int_o^t e(t)\, dt}_{\text{I: Integral}}$$

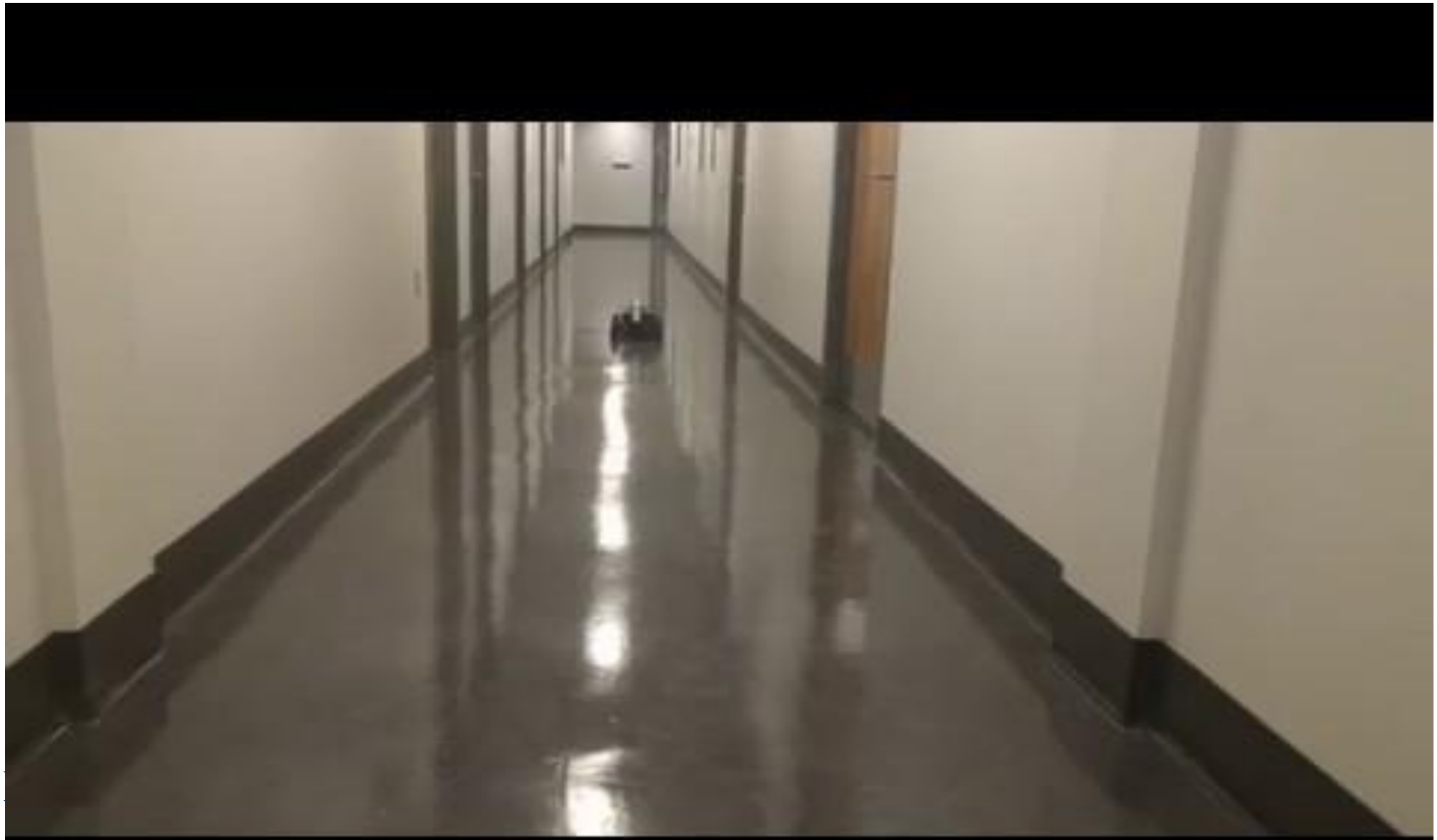
With Error Term: $e(t) = -(y + L * \sin(\Theta))$

Ki must be properly tuned otherwise overshoots will happen.
Since the impact of the integral will be large at that moment,
P and D controller can't compensate for that control input and
the system will become unstable.



Penn Engineering

Eventually, the racecar is tuned!



Penn

PID Tuning

- PID Tuning: How do we find good control gains K_p, K_i, K_d ?
- PID Tuning needs experience and hands on tuning with experiments to evaluate the tuning
- Depends on the plant: what it can tolerate, what's dangerous and what's ok
- Depends on desired performance characteristics: low steady-state error, quick settling time
- Heuristics: Some heuristics more popular than others (e.g., Ziegler-Nichols)
- Use Optimization Algorithm

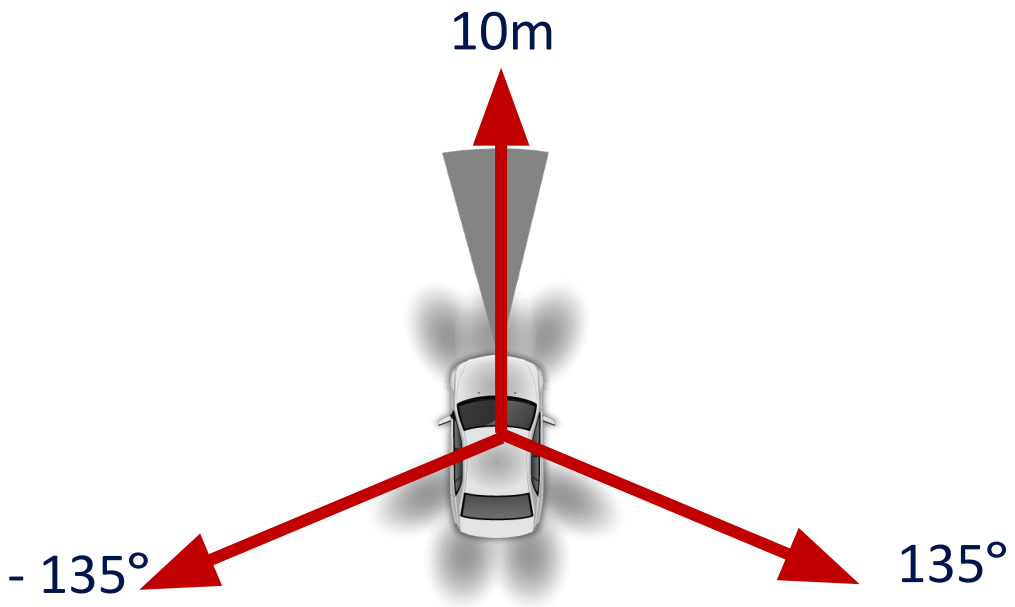
Distance Finder and PID

Wall Following - Computing the actual errors

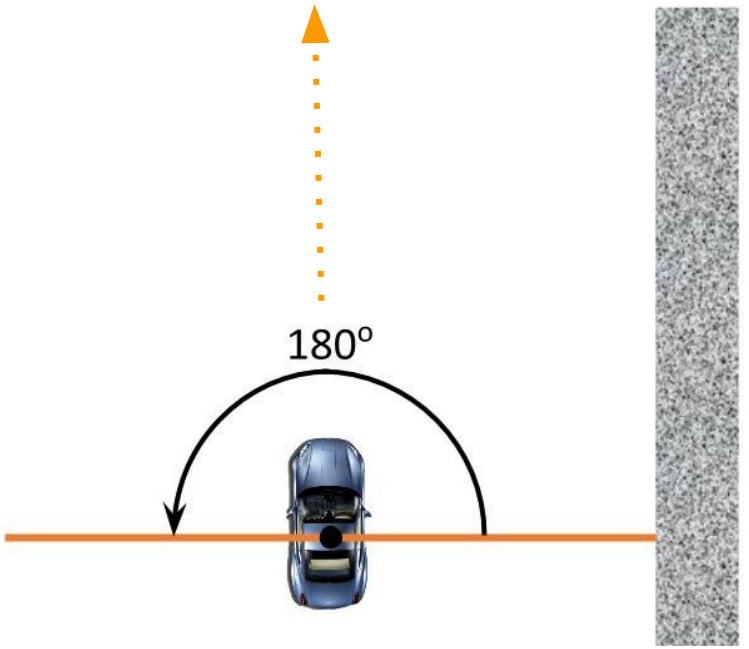
Objective:

Use the lidar to measure the distance from the wall.

Continuously align the car to drive at a fixed distance from the wall.



270° Lidar angle range

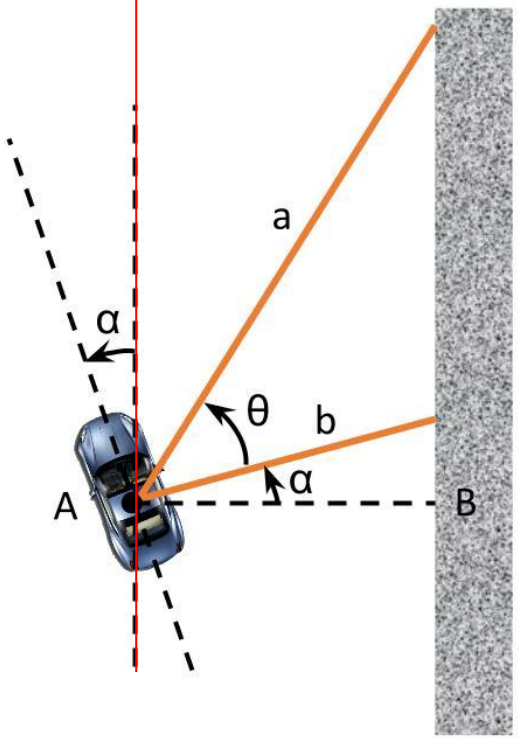


Lidar angles along 0° to 180° in the vehicle frame

How can we steer the car to get back on track?

Initially our car is not just at an offset but misaligned too

Pick two LIDAR rays facing right - One at 0° and one at θ°



$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right)$$

$$AB = b\cos(\alpha)$$

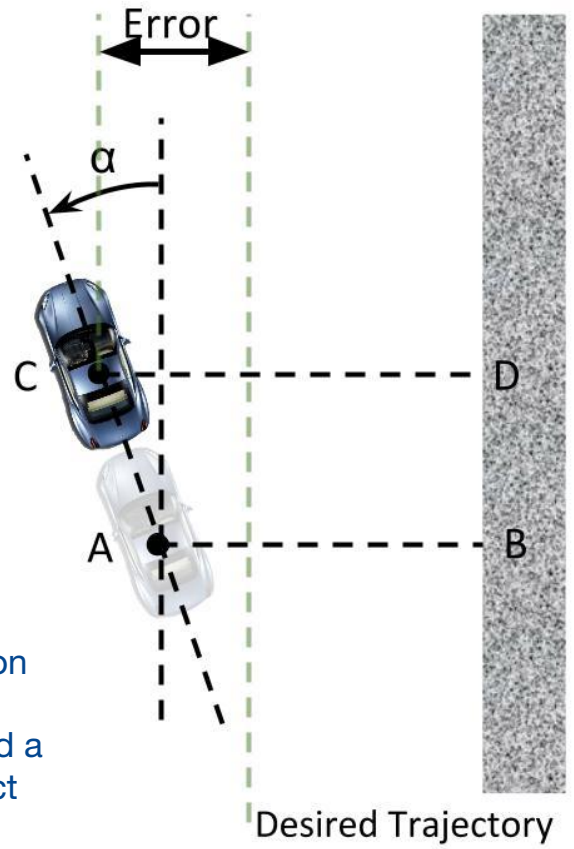
Error = desired trajectory – AB ?

Not quite

Account for the forward motion of the car

Future position with the same heading

Current position and heading



$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta)-b}{a\sin(\theta)}\right)$$

$$AB = b\cos(\alpha)$$

$$CD = AB + AC\sin(\alpha)$$

It is not recommended to get the error at the current position and correct it since the system has finite delays and at the moment of correction control signal the car will have moved a certain distance. So, to compensate that we need to predict the error at a future instance. That's why CD is chosen.

Error = desired trajectory – CD

Penn Engineering

PID Steering Control

$$V_\theta = K_p \times e(t) + K_d \frac{de(t)}{dt}$$

$$V_\theta = K_p \times error + K_d \times previous\ error - current\ error$$

Current steering angle = previous steering angle - V_θ

Learn to Drive Straight *in ROS*

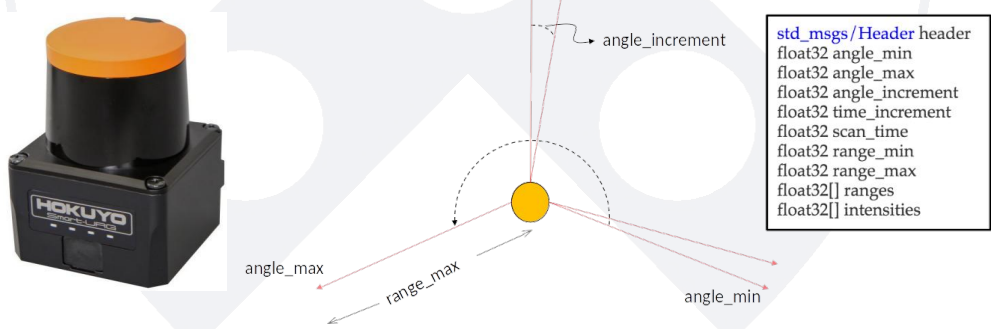
Penn Engineering × F1 TENTH

New package: ‘race’

- > race
  - >> msg
    - >>> drive_param.msg
    - >>> drive_values.msg
    - >>> pid_input.msg
  - >> src
    - >>> dist_finder.py
    - >>> control.py
    - >>> scan_test.py
  - >> CMakeLists.txt
  - >> package.xml

Skeleton code provided – You complete

Basic setup for wall following



dist_finder node → `pid_input` → Drive Command

Get sensor data → Calculate current and projected error → Compute the PID → Send drive command

You need to capture this in ROS nodes, topics, publishers and subscribers

~/Projects/arc_tutorials/Week 2/dist_finder.py – Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
4  import math
5  from sensor_msgs.msg import LaserScan
6  from race.msg import pid_input
7
8  desired_trajectory = 1
9  vel = 30
10
11 pub = rospy.Publisher('error', pid_input, queue_size=10)
12
13 ##  Input:  data: Lidar scan data
14 ##          theta: The angle at which the distance is requried
15 ##  OUTPUT: distance of scan at angle theta
16 def getRange(data,theta):
17 # Find the index of the arary that corresponds to angle theta.
18 # Return the lidar scan value at that index
19 # Do some error checking for NaN and ubsurd values
20 ## Your code goes here
21
22     return
23
24 def callback(data):
25     theta = 50;
26     a = getRange(data,theta)
27     b = getRange(data,0)
28     swing = math.radians(theta)
29
```

Line 24, Column 5 | Tab Size: 4 | Python

~/Projects/arc_tutorials/Week 2/dist_finder.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
1  #!/usr/bin/env python
2
3  import rospy
4  import math
5  from sensor_msgs.msg import LaserScan
6  from race.msg import pid_input
7
8  desired_trajectory = 1
9  vel = 30
10
11 pub = rospy.Publisher('error', pid_input, queue_size=10)
12
13 ##  Input:  data: Lidar scan data
14 ##          theta: The angle at which the distance is requried
15 ##  OUTPUT: distance of scan at angle theta
16 def getRange(data,theta):
17 # Find the index of the arary that corresponds to angle theta.
18 # Return the lidar scan value at that index
19 # Do some error checking for NaN and ubsurd values
20 ## Your code goes here
21
22     return
23
24 def callback(data):
25     theta = 50;
26     a = getRange(data,theta)
```

9 characters selected | Tab Size: 4 | Python

File: sensor_msgs/LaserScan.msg

Raw Message Definition

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

Compact Message Definition

~/Projects/arc_tutorials/Week 2/dist_finder.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
22      return
23
24  def callback(data):
25      theta = 50;
26      a = getRange(data,theta)
27      b = getRange(data,0)
28      swing = math.radians(theta)
29
30      ## Your code goes here
31
32
33
34
35      ## END
36
37      msg = pid_input()
38      msg.pid_error = error
39      msg.pid_vel = vel
40      pub.publish(msg)
41
42
43  if __name__ == '__main__':
44      print("Laser node started")
45      rospy.init_node('dist_finder',anonymous = True)
46      rospy.Subscriber("scan",LaserScan,callback)
47      rospy.spin()
```

9 characters selected | Tab Size: 4 | Python

~/Projects/arc_tutorials/Week 2/dist_finder.py – Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
13  ##  Input:  data: Lidar scan data
14  ##          theta: The angle at which the distance is requried
15  ##  OUTPUT: distance of scan at angle theta
16  def getRange(data,theta):
17  # Find the index of the arary that corresponds to angle theta.
18  # Return the lidar scan value at that index
19  # Do some error checking for NaN and ubsurd values
20  ## Your code goes here
21
22      return
23
24  def callback(data):
25      theta = 50;
26      a = getRange(data,theta)
27      b = getRange(data,0)
28      swing = math.radians(theta)
29
30      ## Your code goes here
31
32
33
34
35      ## END
36
37      msg = pid_input()
38      msg.pid_error = error
```

8 characters selected

Callback:

1. Pick two rays on the right side.
   1. Specify the correct index for the ray in the range 0 to len(data.ranges)-1
   2. Pick rays at $0°$ and $\theta°$
2. Obtain distances ‘a’ and ‘b’ at the two angles by calling **getRanges.**
3. Calculate alpha, AB, and, CD using a, b, and theta
4. Calclate error = desired_trajectory – future_distance (CD)
5. Constrcut pid_input message
   1. msg.pid_error = error
   2. msg.pid_vel = vel
   3. pub.publish(msg)

~/Projects/arc_tutorials/Week 2/dist_finder.py – Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
13  ##  Input:  data: Lidar scan data
14  ##          theta: The angle at which the distance is requried
15  ##  OUTPUT: distance of scan at angle theta
16  def getRange(data,theta):
17  # Find the index of the arary that corresponds to angle theta.
18  # Return the lidar scan value at that index
19  # Do some error checking for NaN and ubsurd values
20  ## Your code goes here
21
22      return
23
24  def callback(data):
25      theta = 50;
26      a = getRange(data,theta)
27      b = getRange(data,0)
28      swing = math.radians(theta)
29
30      ## Your code goes here
31
32
33
34
35      ## END
36
37      msg = pid_input()
38      msg.pid_error = error
```

8 characters selected

getRange(data,angle)

1. Takes input the received scan data, and an angle (0° or θ°)
2. Outputs the distance reported by the LIDAR at that angle. (a or b)
3. Accounts for any NaN or inf data fields.

~/Projects/arc_tutorials/Week 2/control.py – Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Subscribes to the topic error of the message type pid_input

dist_finder.py × control.py ×

```
1  #!/usr/bin/env python
2
3  import rospy
4  from race.msg import drive_param
5  from race.msg import pid_input
6
7  kp = 14.0
8  kd = 0.09
9  servo_offset = 18.5
10 prev_error = 0.0
11 vel_input = 25.0
12
13 pub = rospy.Publisher('drive_parameters', drive_param, queue_size=1)
14
15 def control(data):
16     global prev_error
17     global vel_input
18     global kp
19     global kd
20
21     ## Your code goes here
22     # 1. Scale the error
23     # 2. Apply the PID equation on error
24     # 3. Make sure the error is within bounds
25
26
```

Line 1, Column 1 Tab Size: 4 Python

~/Projects/arc_tutorials/W

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
28
29      msg = drive_param();
30      msg.velocity = vel_input
31      msg.angle = angle
32      pub.publish(msg)
33
34  if __name__ == '__main__':
35      global kp
36      global kd
37      global vel_input
38      print("Listening to error for PID")
39      kp = input("Enter Kp Value: ")
40      kd = input("Enter Kd Value: ")
41      vel_input = input("Enter Velocity: ")
42      rospy.init_node('pid_controller', anonymous=True)
43      rospy.Subscriber("error", pid_input, control)
44      rospy.spin()
```

9 characters selected Tab Size: 4 Python

ackermann_msgs/AckermannDrive Message

File: ackermann_msgs/AckermannDrive.msg

Raw Message Definition

```
float32 steering_angle [-100, 100]
float32 steering_angle_velocity
float32 speed  [-100, 100]
float32 acceleration
float32 jerk
```

~/Projects/arc_tutorials/Week 2/control.py - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
1  #!/usr/bin/env python
2
3  import rospy
4  from race.msg import drive_param
5  from race.msg import pid_input
6
7  kp = 14.0
8  kd = 0.09
9  servo_offset = 18.5
10 prev_error = 0.0
11 vel_input = 25.0
12
13 pub = rospy.Publisher('drive_parameters', drive_param, queue_size=1)
14
15 def control(data):
16     global prev_error
17     global vel_input
18     global kp
19     global kd
20
21     ## Your code goes here
22     # 1. Scale the error
23     # 2. Apply the PID equation on error
24     # 3. Make sure the error is within bounds
25
26
```

11 characters selected | Tab Size: 4 | Python

~/Projects/arc_tutorials/Week 2/control.py – Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

dist_finder.py × control.py ×

```
13 pub = rospy.Publisher('drive_parameters', drive_param, queue_size=1)
14
15 def control(data):
16     global prev_error
17     global vel_input
18     global kp
19     global kd
20
21     ## Your code goes here
22     # 1. Scale the error
23     # 2. Apply the PID equation on error
24     # 3. Make sure the error is within bounds
25
26
27     ## END
28
29     msg = drive_param();
30     msg.velocity = vel_input
31     msg.angle = angle
32     pub.publish(msg)
33
34 if __name__ == '__main__':
35     global kp
36     global kd
37     global vel_input
38     print("Listening to error for PID")
```

5 characters selected | Tab Size: 4 | Python

roscore

rosrun hokuyo_node hokuyo_node

A node to obtain laser data

rosrun race control.py

The PID node

rosrun race dist_finder.py

The node to process the sensor data

Launch file

Create a launch file `autonomous.launch` in the race package which launches **all** the above demo nodes in the correct order and accepts user inputs for kp, kd, and velocity.

Velocity PID

You will notice that in the assignment we are correcting the steering based on the error, but what about velocity ? It is held constant. Modify the `control.py` file so that velocity of the car also changes with error, i.e. on the straight parts of the track when the car is parallel to the wall, and error is low (or zero), the car drives at a higher velocity, but during the turns when the error is high, the velocity of the car reduces appropriately.

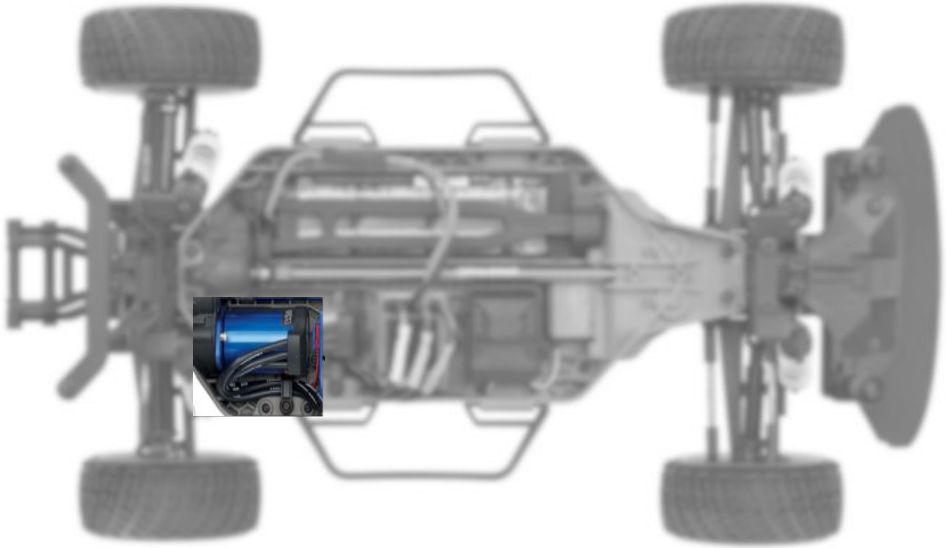
Electronic Speed Control

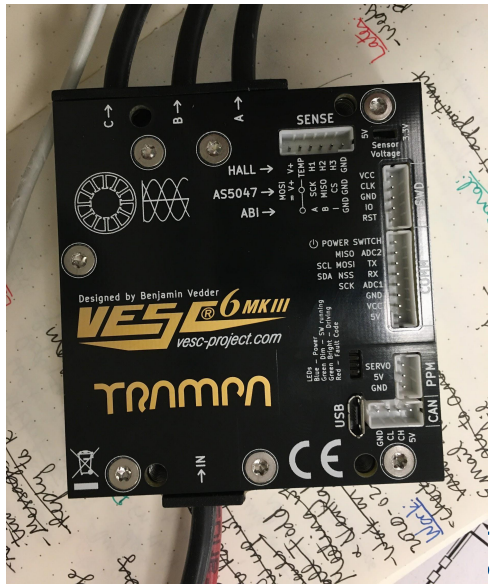
Challenge:
Actuate vehicle via commands in physics units

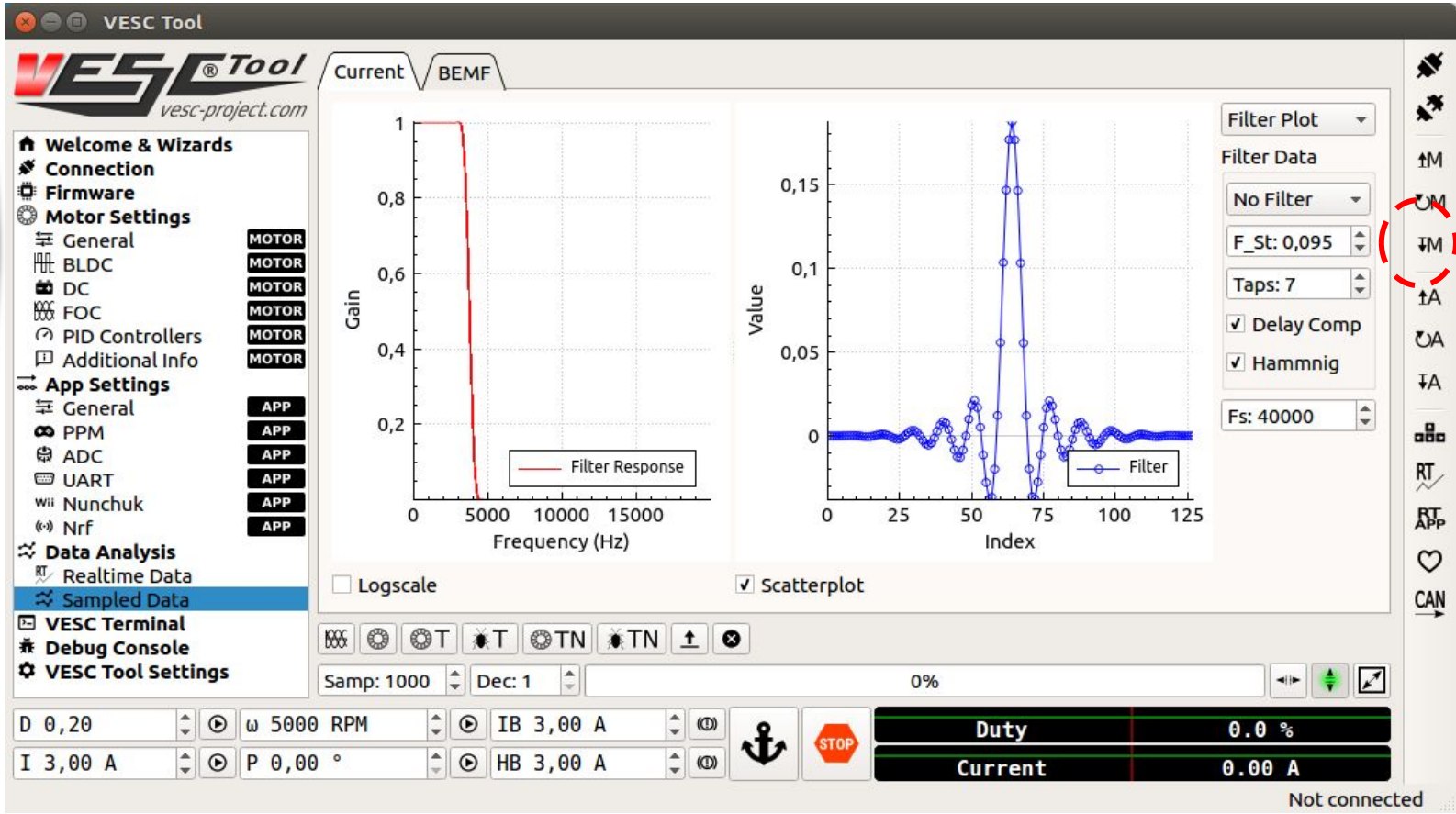
Learning Outcome:
PID, motor control, etc.

Assignment:
Motor controller tuning and related parameters

PID-Tuning: VESC Controller

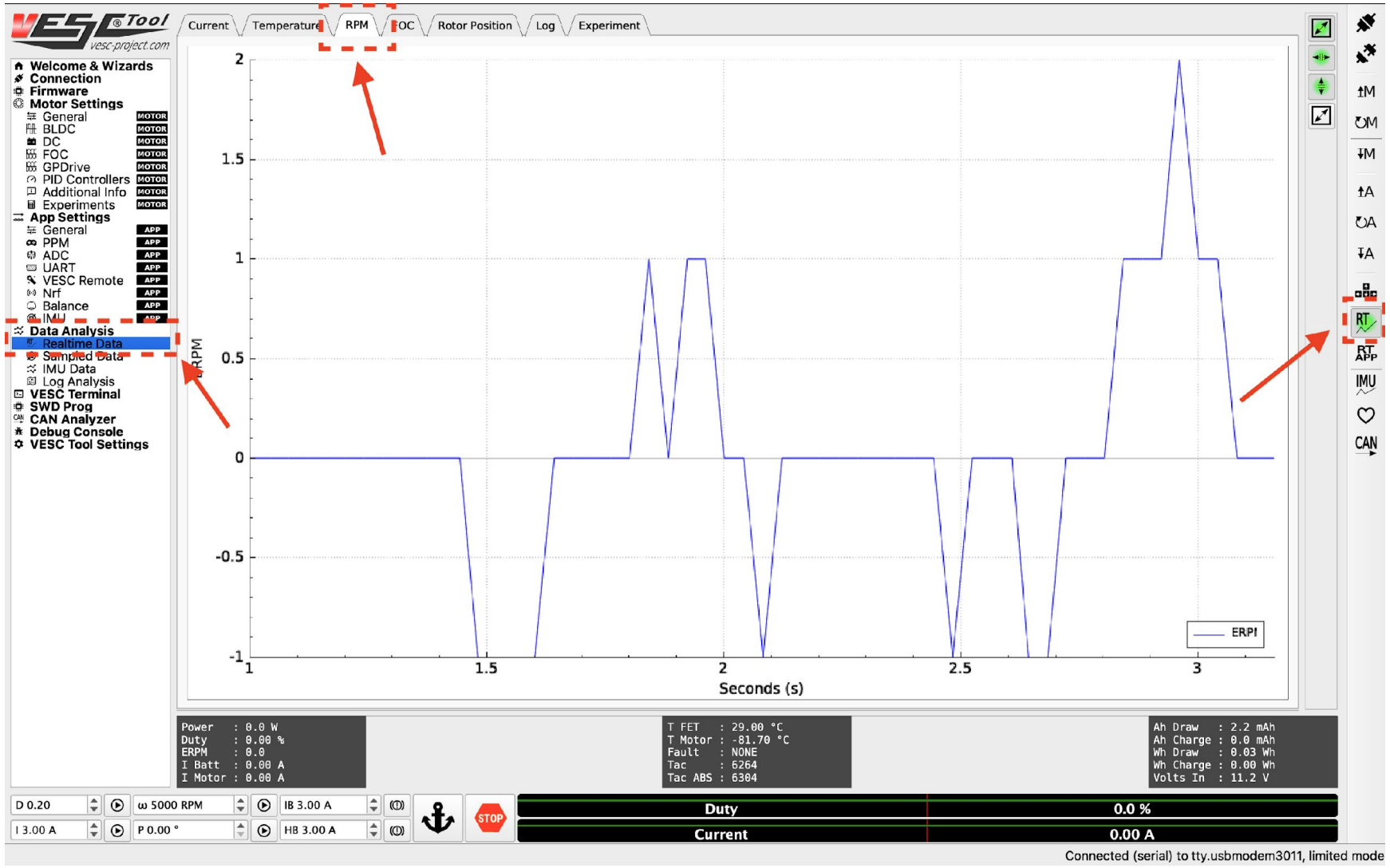




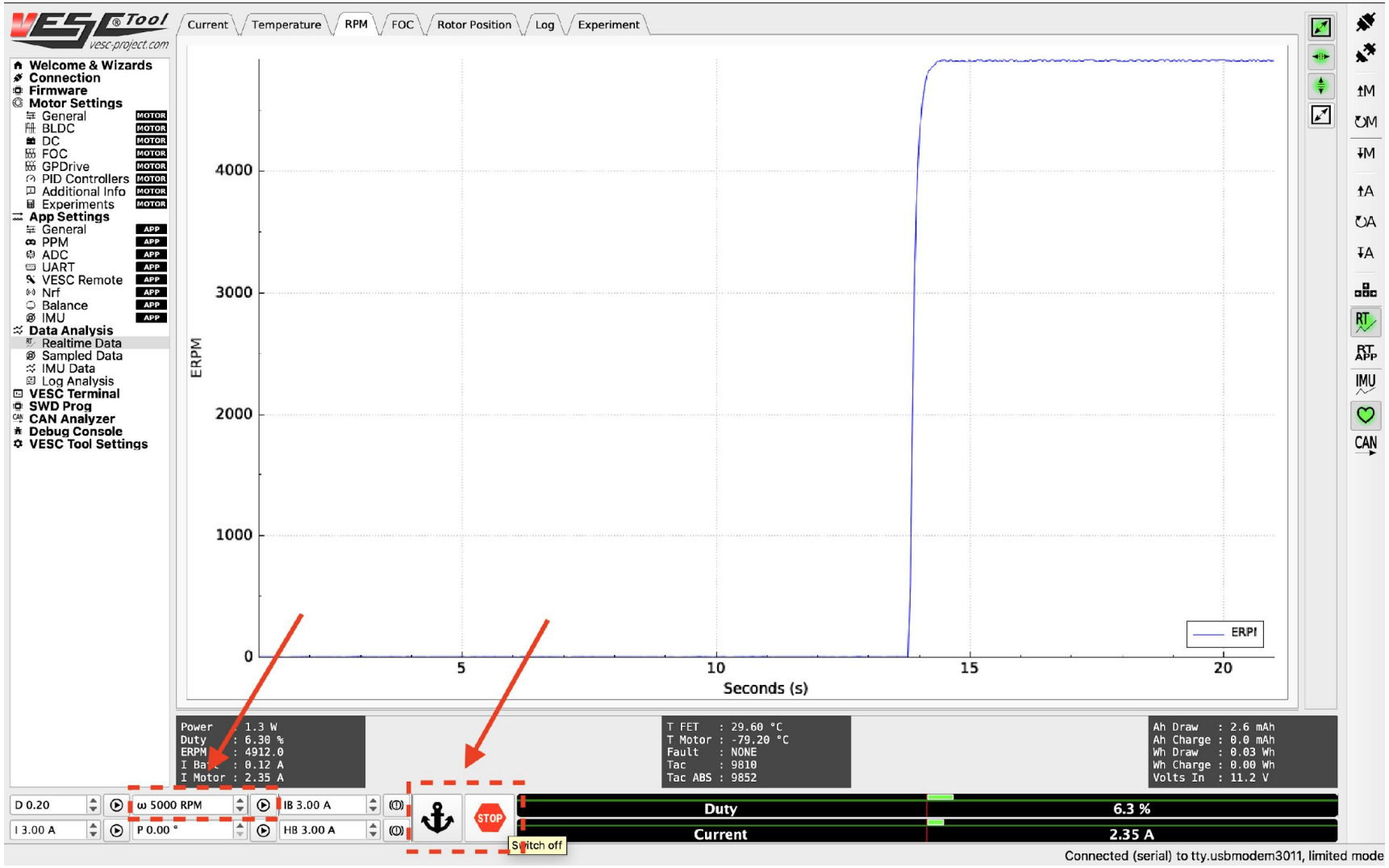


VESC Tool https://vesc-project.com/vesc_tool

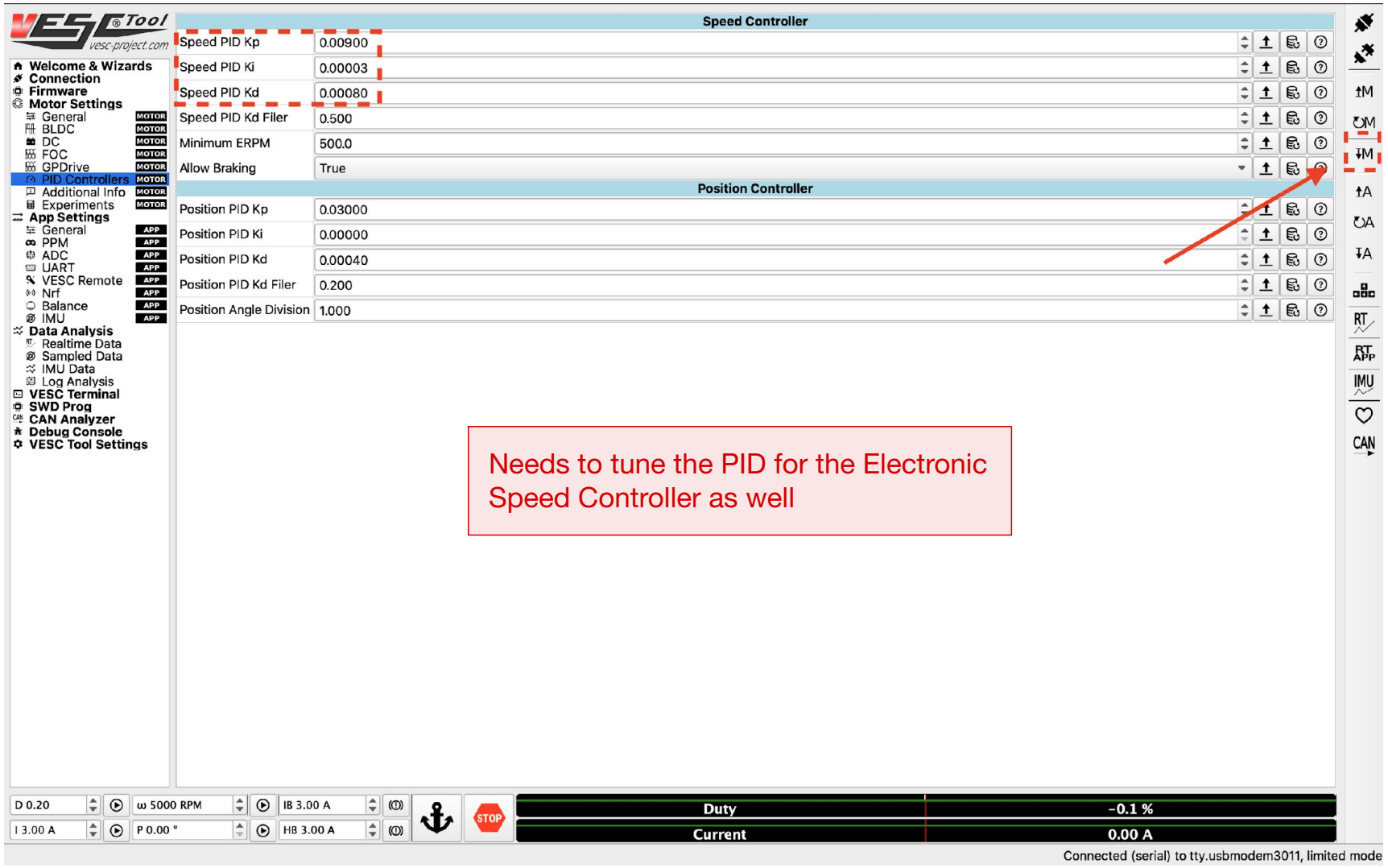
PID-Tuning: VESC Controller



PID-Tuning: VESC Controller

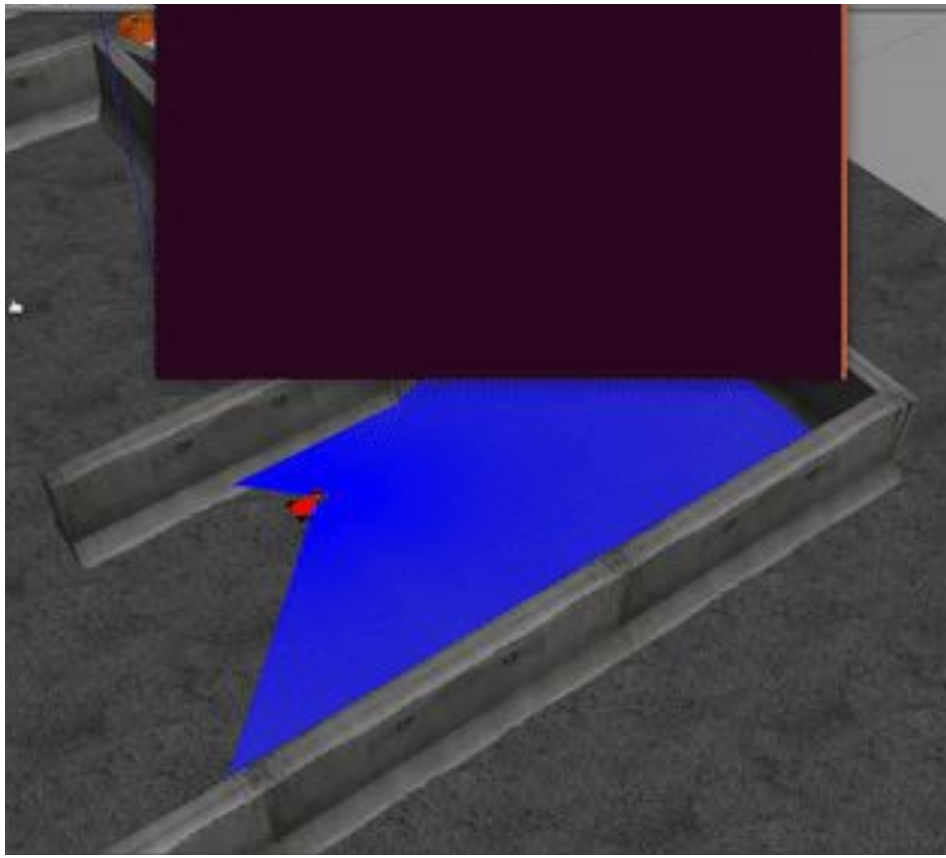


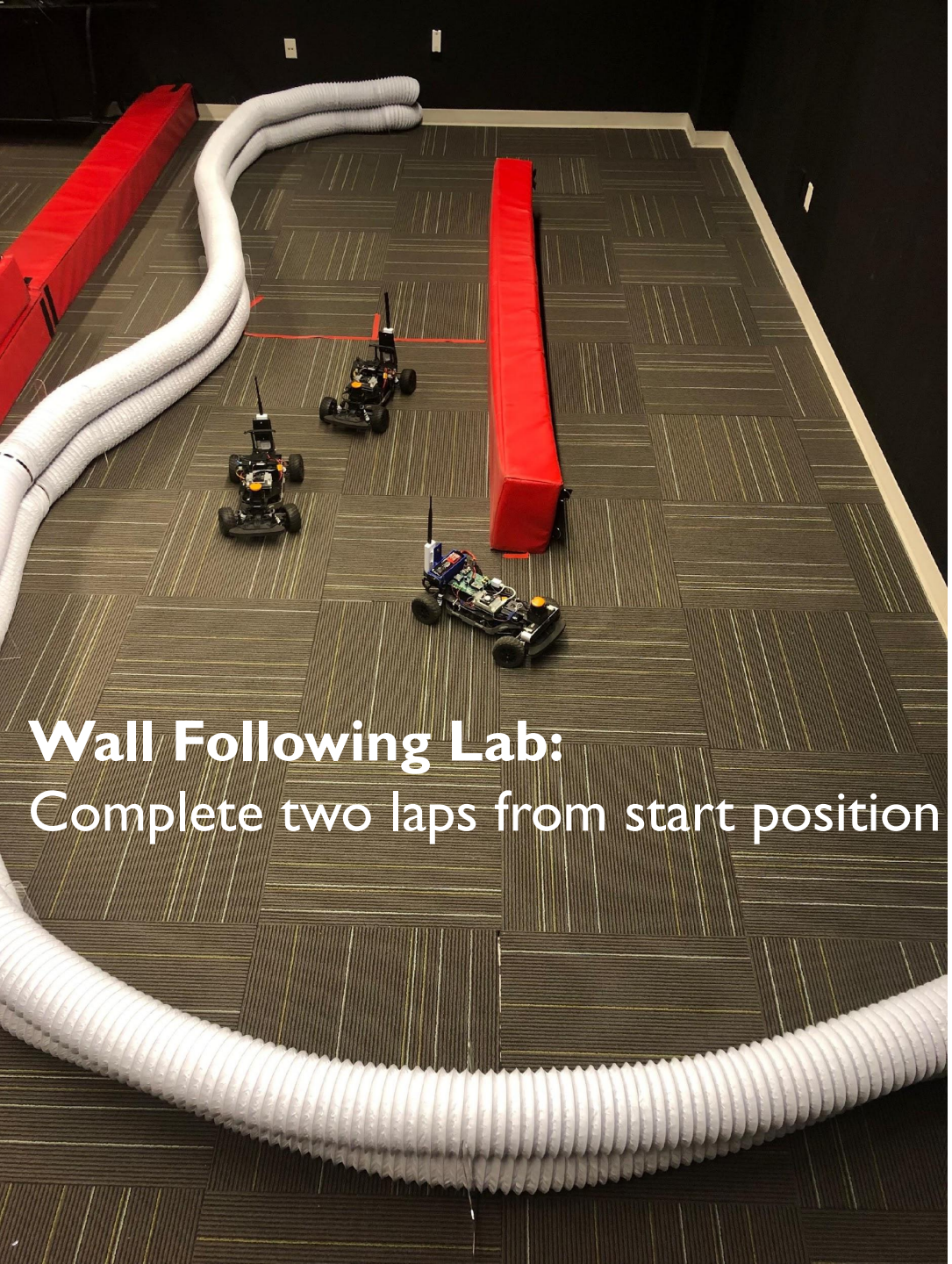
PID-Tuning: VESC Controller



Key Takeaways

1. What it means to track a reference signal
2. PD control for wall following
3. ROS makes it easy in sim and race car
4. VESC makes it easy to tune the motor controller





Wall Following Lab:
Complete two laps from start position

Questions?

Proportional Control

$K_p = 11, K_i = 0, K_d = 0$

Penn