Autonomous Racing

1/10 the scale. 10 times the fun!

Wall Following – Autonomous Racing

Madhur Behl
(University of Virginia)

Reminder

- Assignment 3: TeleOp Control due on Wednesday, Apr 3 @ 2:00pm
- Demo in the lab session on Wednesday
- Share your code in a private github repo with the TA and the instructor and upload to Collab (one submission per team)
- Share a link to your blog/website.

Today

Wall following
PID control

Perception – Wall following

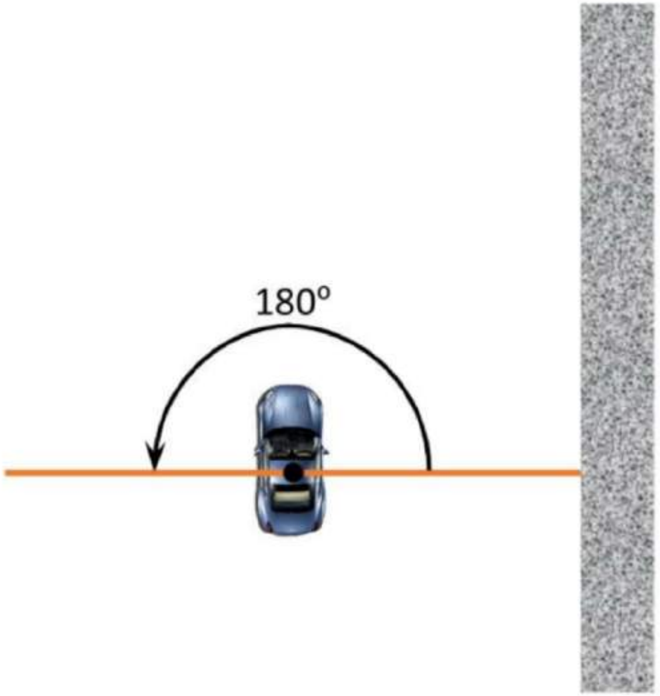
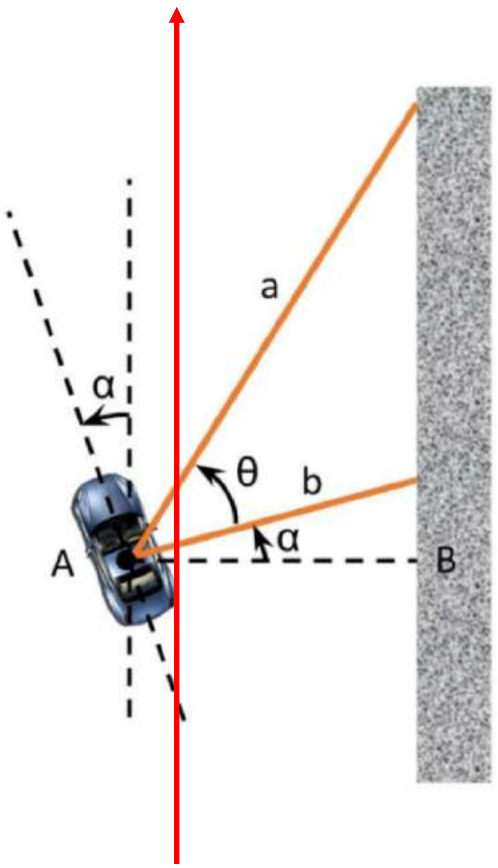


Figure 1: Lidar scan angles

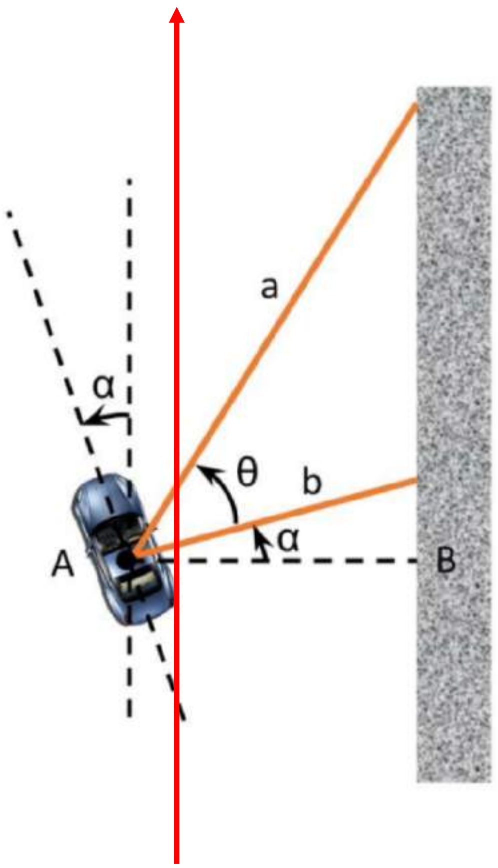
Pick two LIDAR rays facing right – One at 0° and one at θ°



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

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

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

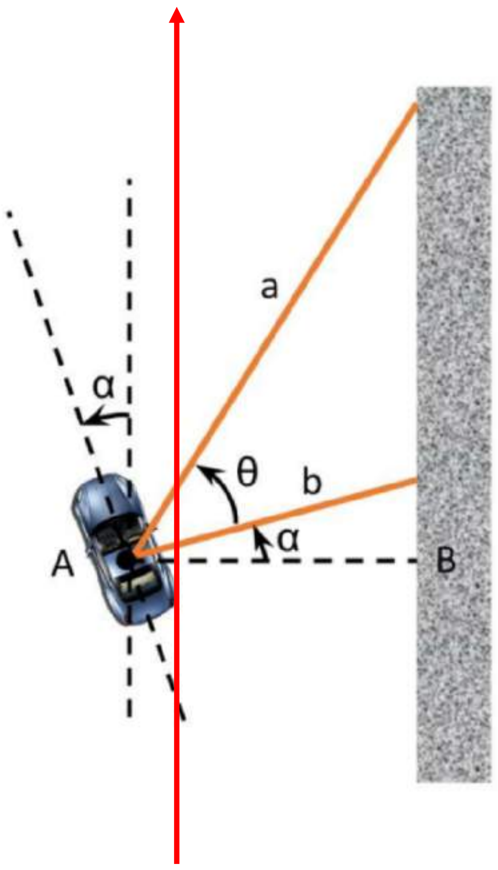


$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right)$$

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

Error = desired trajectory – AB ?

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



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

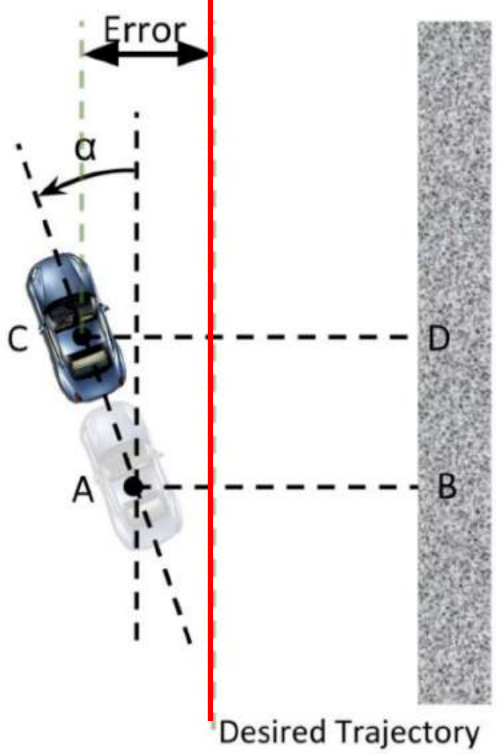
$$AB = b\cos(\alpha)$$

Error = desired trajectory – AB ?

Not quite

Account for the forward motion of the car

Figure 3: Projecting the car future in time



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

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

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

Error = desired trajectory – CD

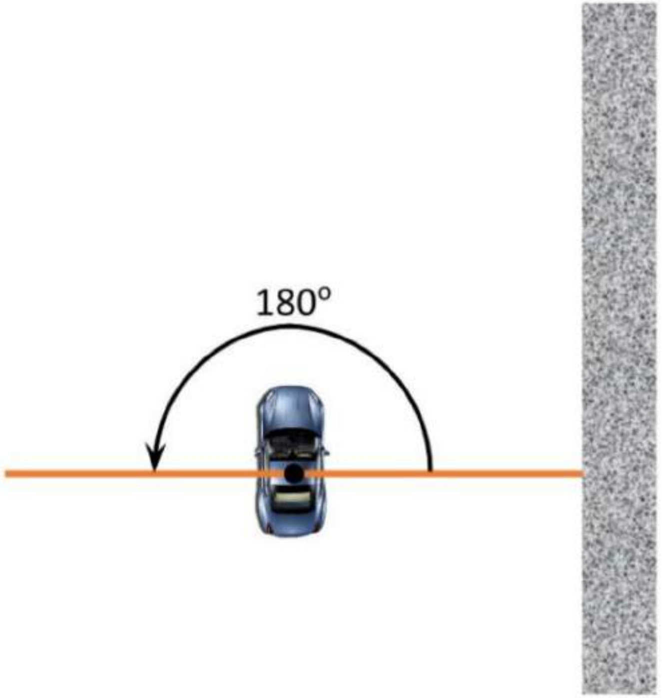
PID Steering Control

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

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

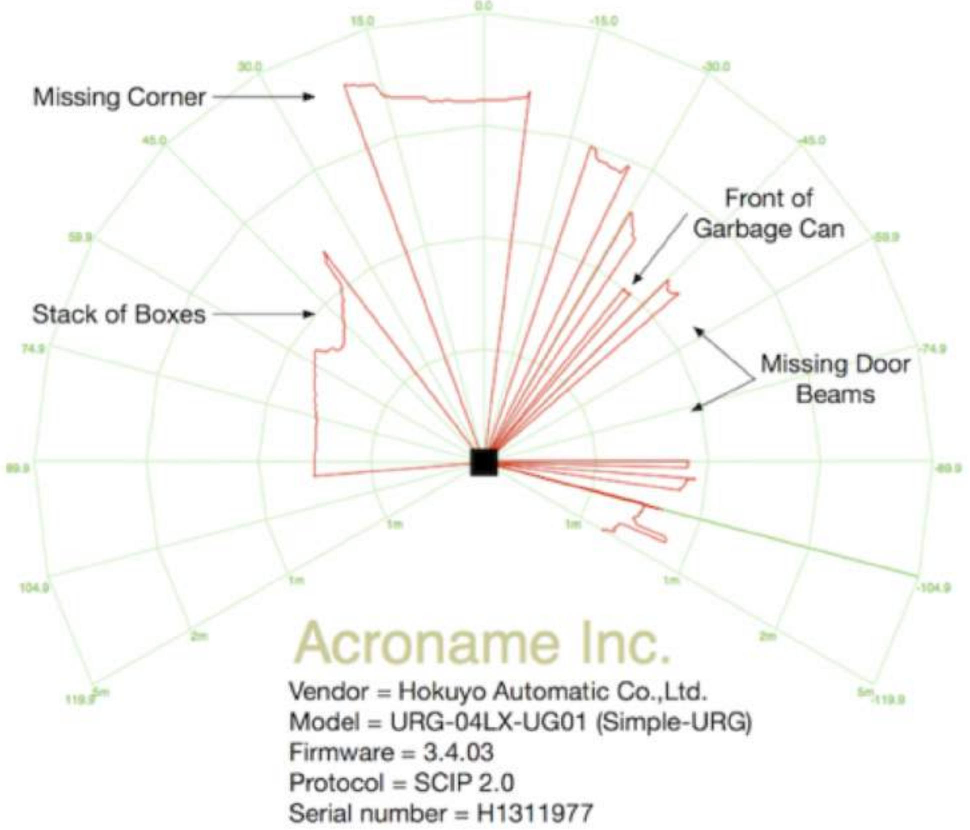
$$steering\ angle = steering\ angle\ - V_\theta$$

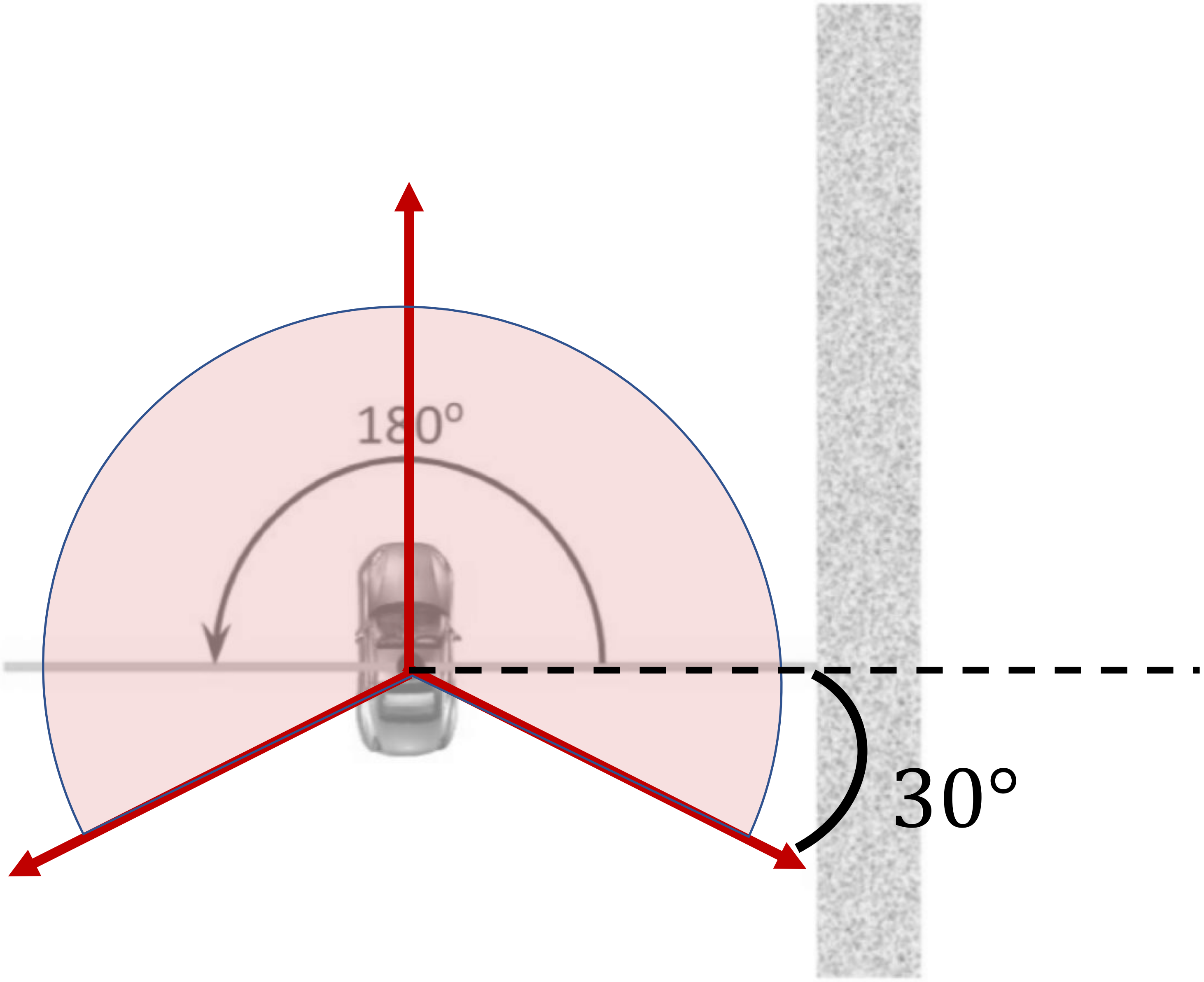
Field of View



← **Assumption**

Reality →




180°
30°

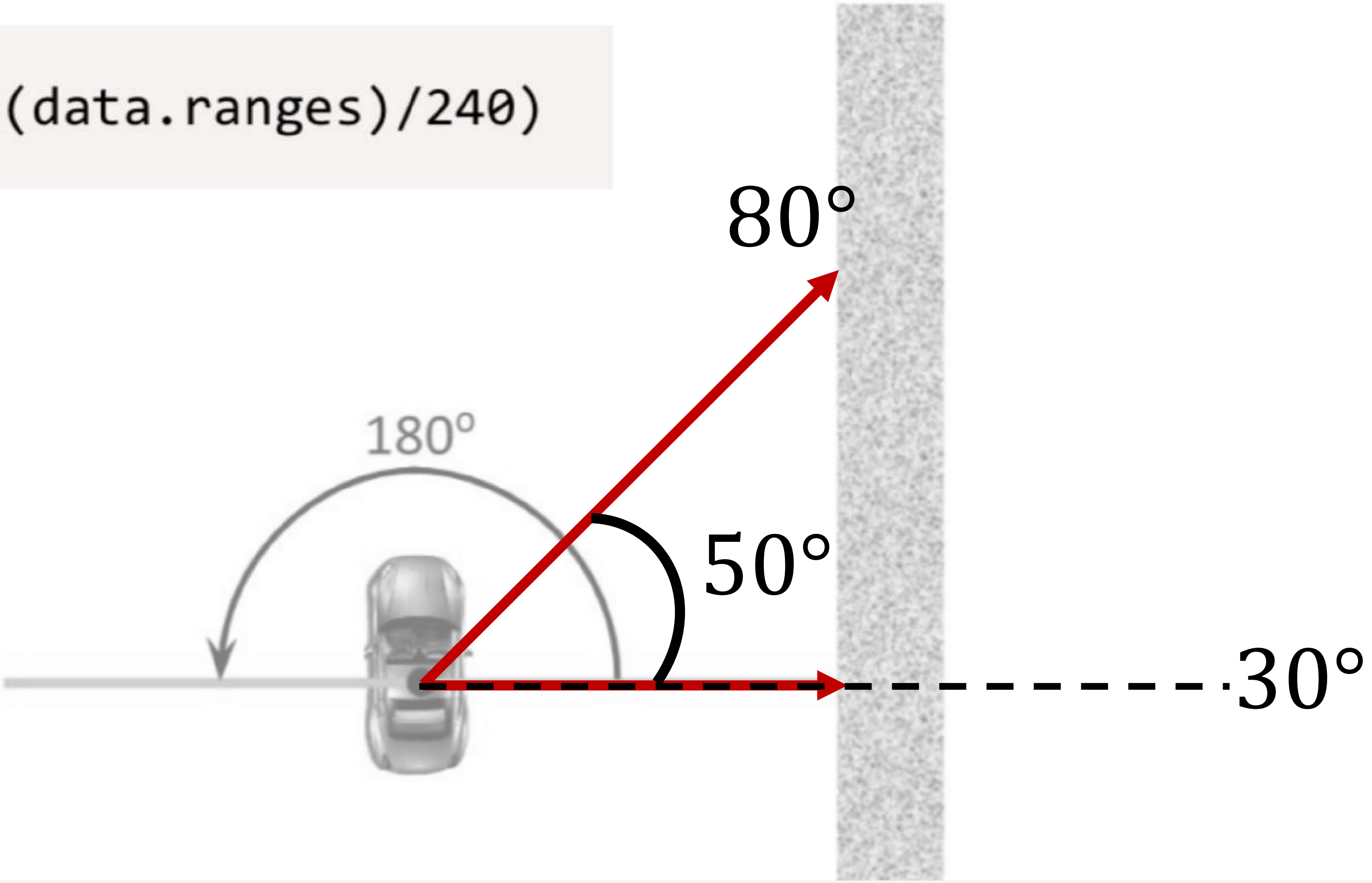
http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

You can set these in the LIDAR configuration

You can verify/output these values from **Laserscan** messages

```
index = theta * (len(data.ranges)/240)
```

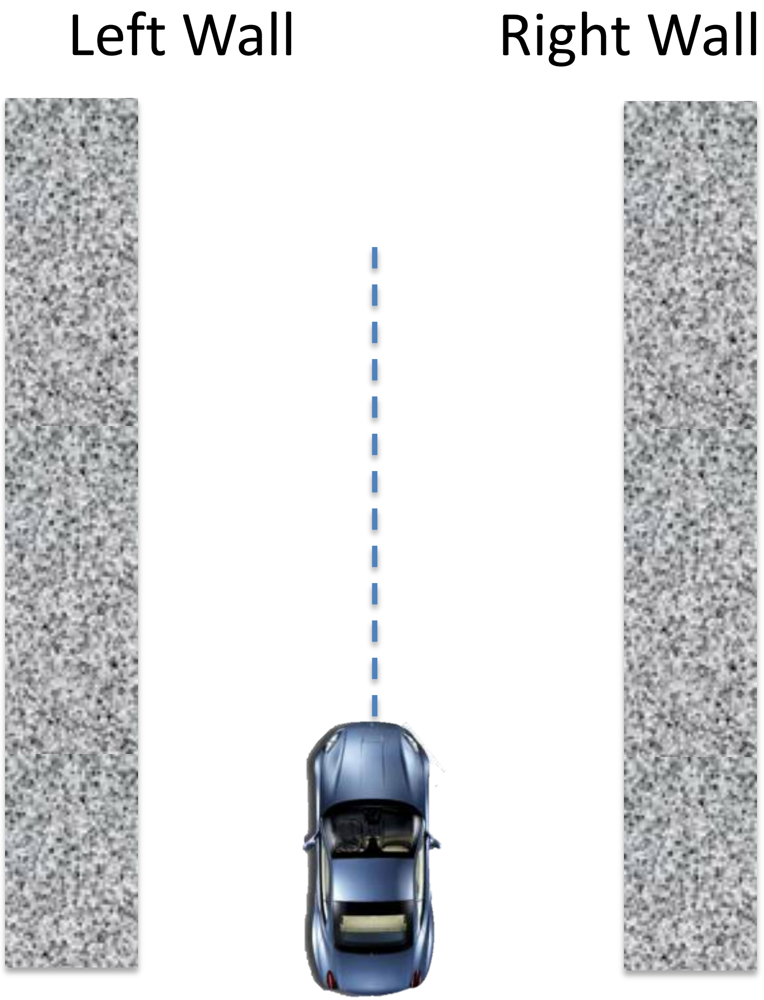



```
zero_ray_index = 30 * (len(data.ranges)/240) = 0.125 * (len(data.ranges)    )
```

Control

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

PID control: objectives

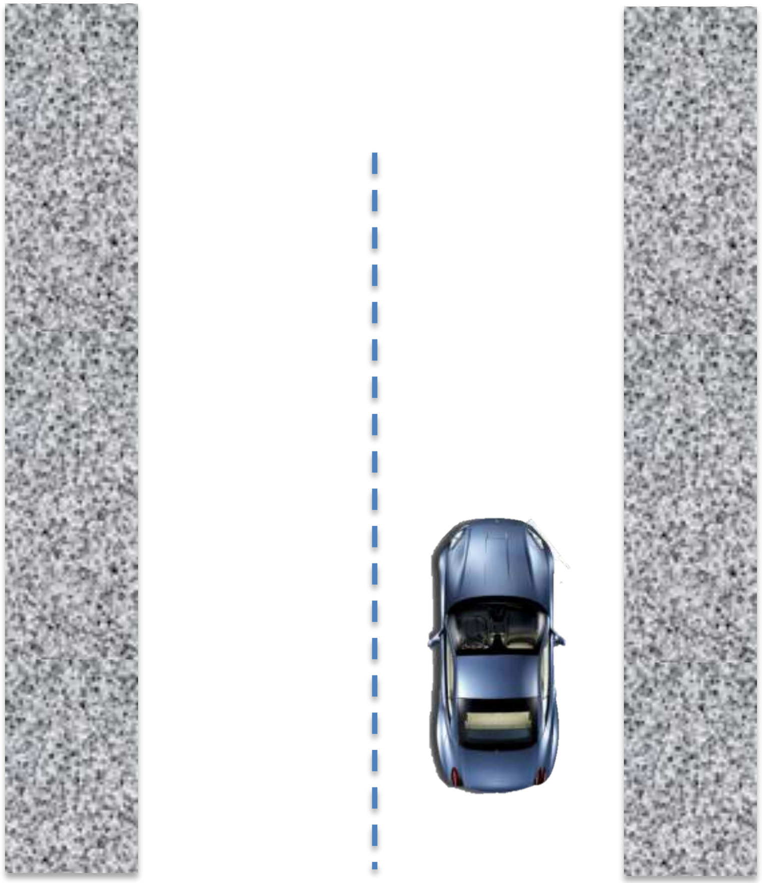


Control objective:

1) keep the car driving along the centerline,

2) parallel to the walls.

PID control: objectives

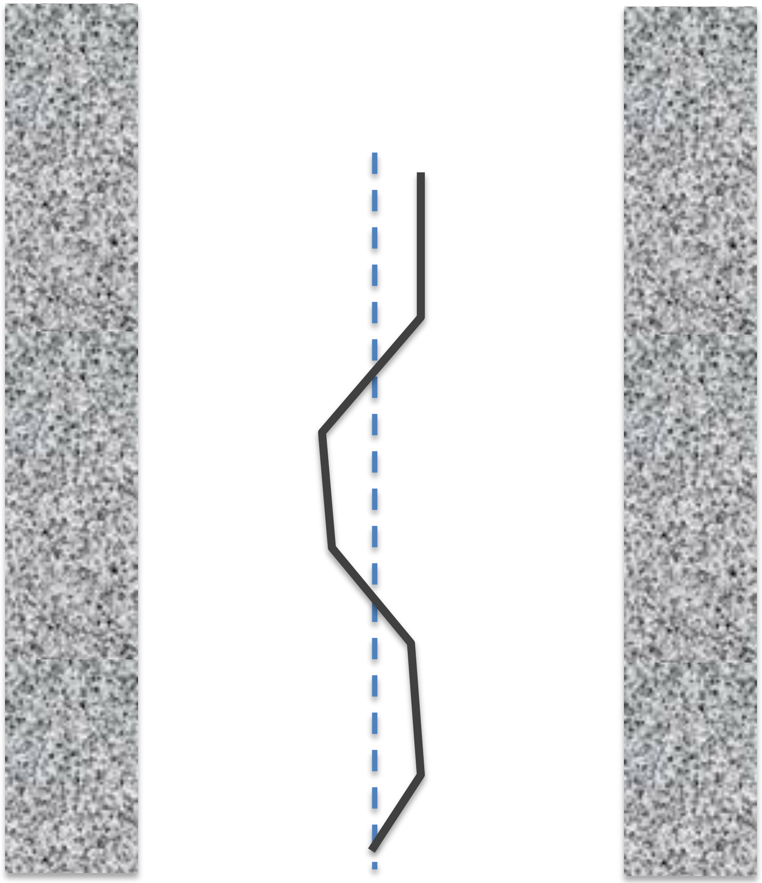


Control objective:

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

2) parallel to the walls.

PID control: objectives

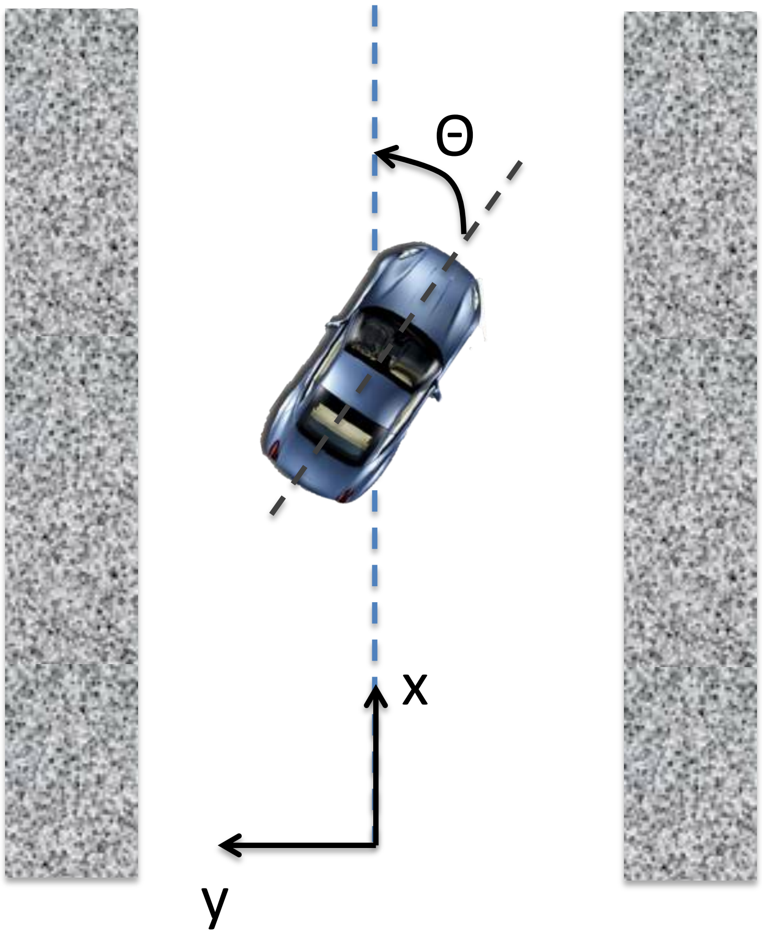


Control objective:

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

~~2) parallel to the walls.~~

PID control: control objectives



Control objective:

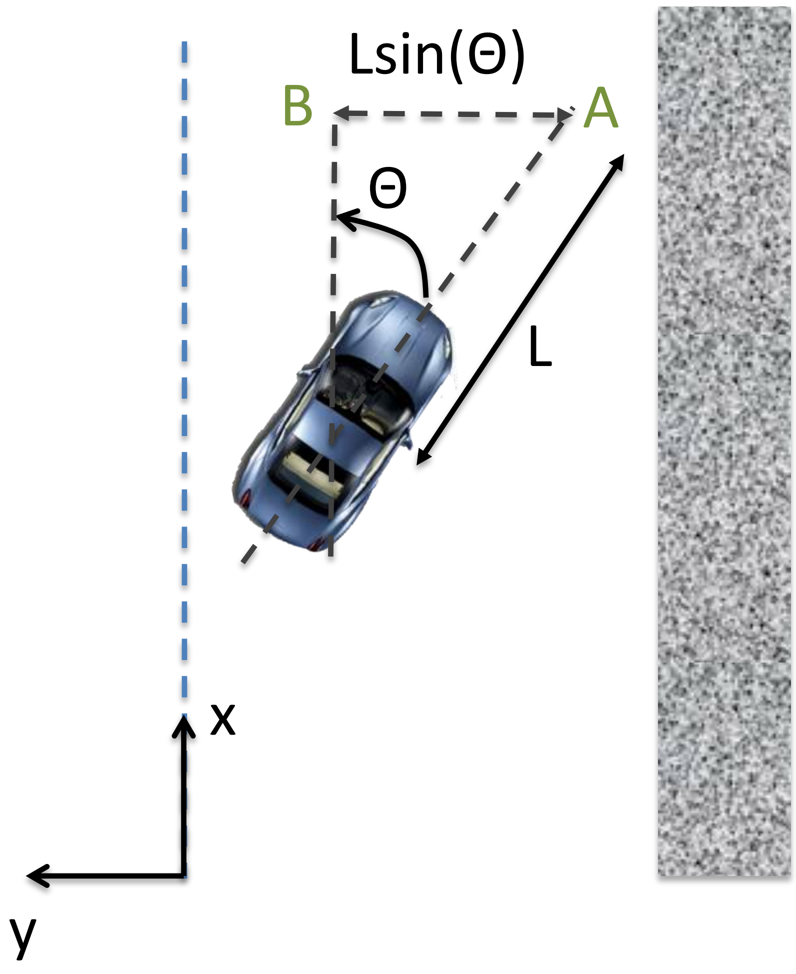
1) keep the car driving along the centerline,

$$y = 0$$

2) parallel to the walls.

$$\Theta = 0$$

PID control: control objectives



Control objective:

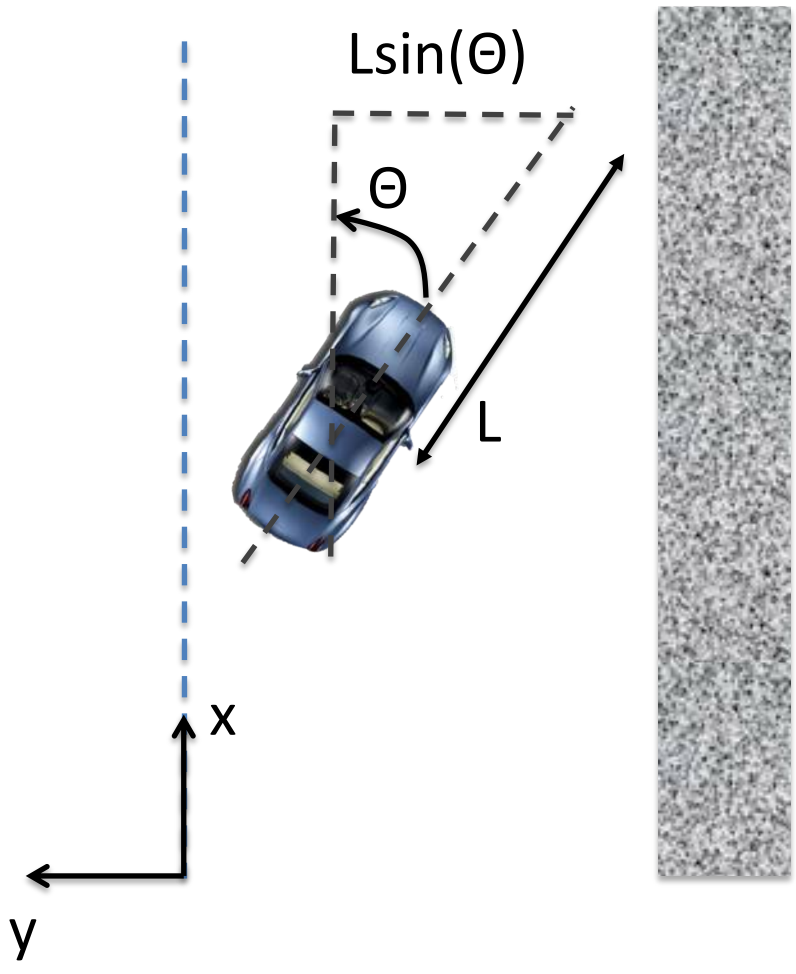
1) keep the car driving along the centerline,

$$y = 0$$

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

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

PID control: control inputs



Control input:
Steering angle Θ

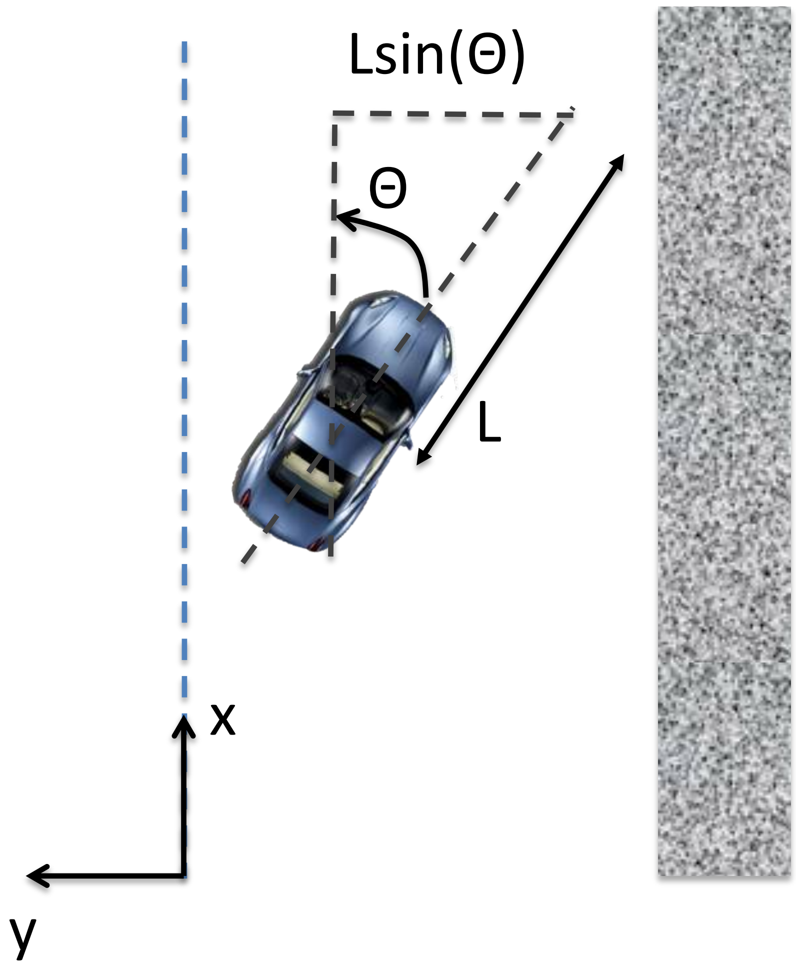
We will hold the velocity constant.

How do we control the steering angle to keep

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

as much as possible?

PID control: error term

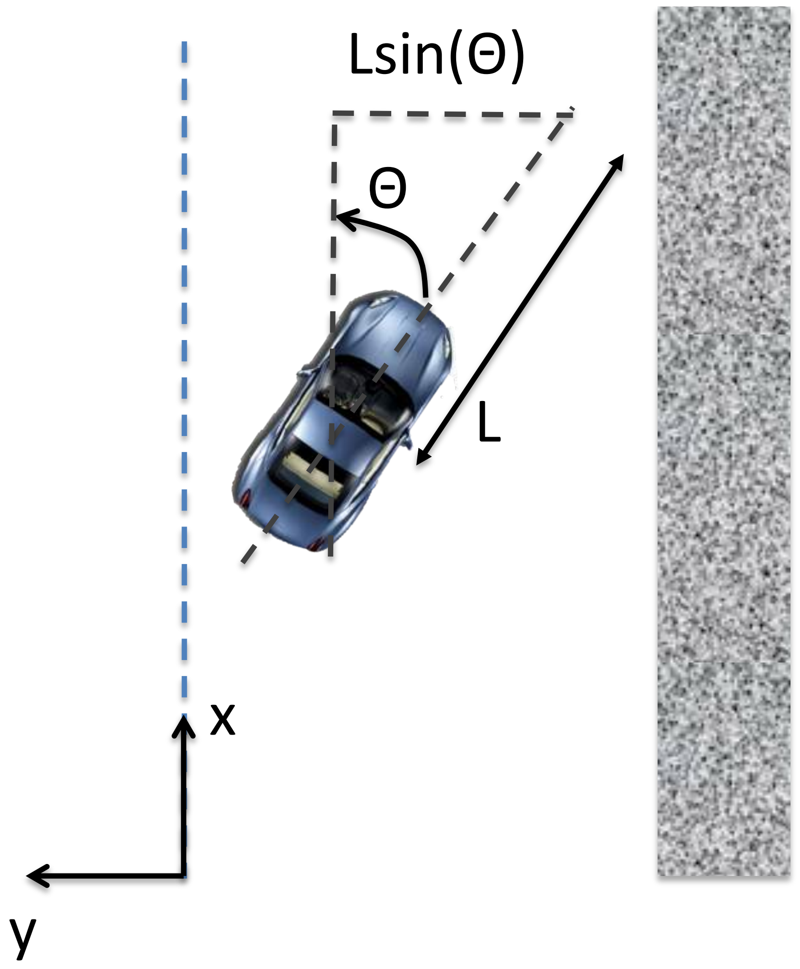


Want both y and Lsin(Θ) to be zero

→ Error term $e(t) = -(y + L\sin(\Theta))$

We'll see why we added a minus sign

PID control: computing input

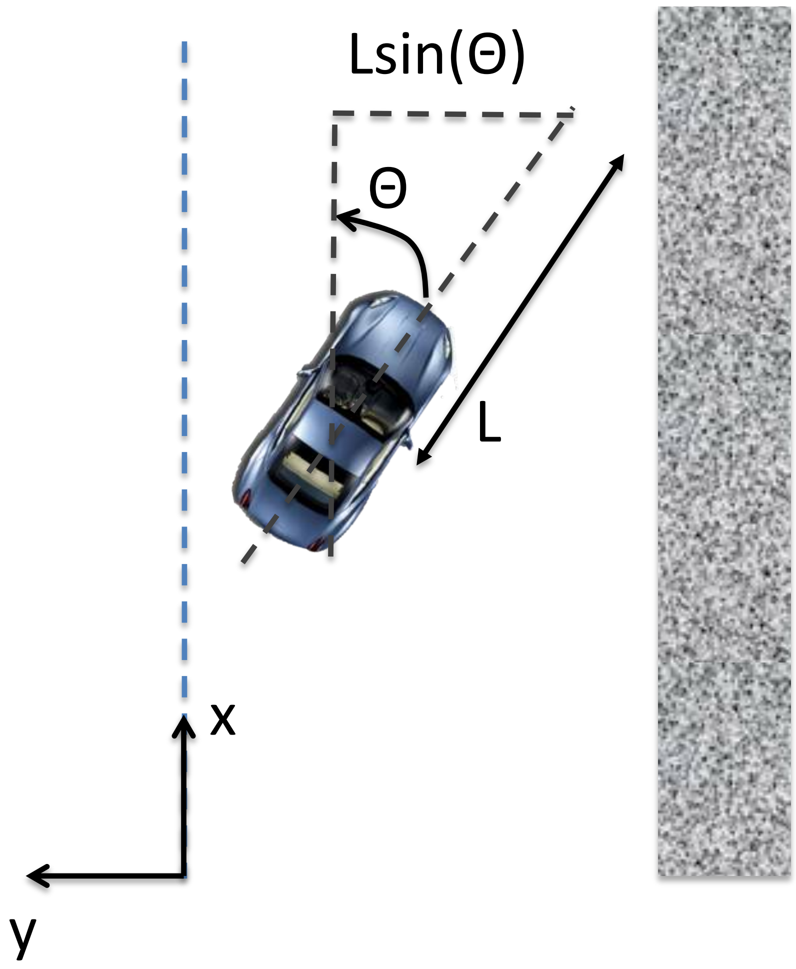


When y > 0, car is to the left of centerline
→ Want to steer right: $\Theta < 0$

When $L\sin(\Theta) > 0$, we will be to the left of centerline in L meters
→ so want to steer right: $\Theta < 0$

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 $\Theta > 0$

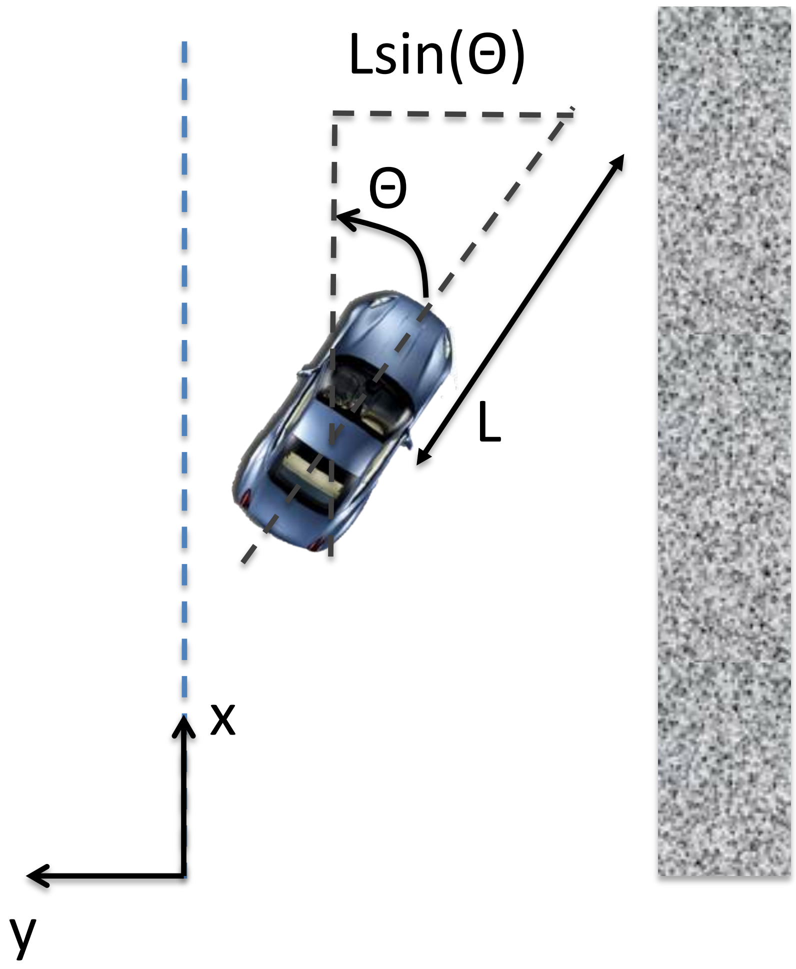
When $L\sin(\Theta) < 0$, we will be to the right of centerline in L meters, so want to steer left

→Want $\Theta > 0$

Consistent with previous requirement:

$\Theta_d = Kp\,(-y - L\sin(\Theta))$

PID control: Proportional control

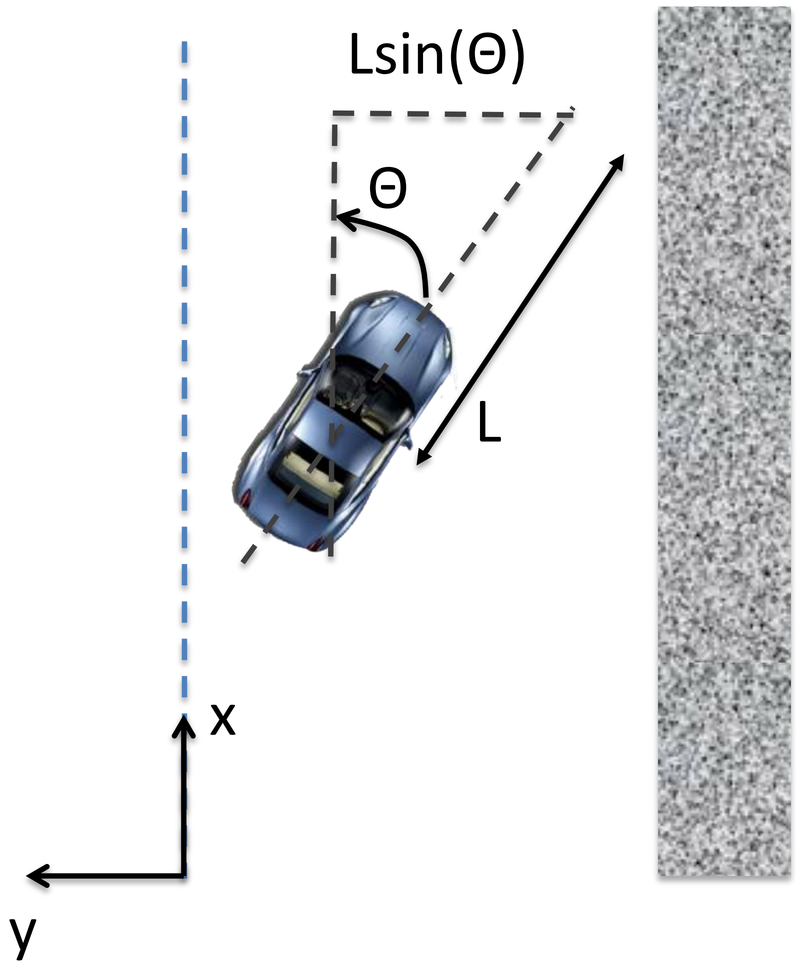


$\Theta_d = C\,K_p\,(-y - L\sin(\Theta)) = C\,K_p\,e(t)$

This is **P**roportional control.

The extra C constant is for scaling distances to angles.

PID control: Derivative control

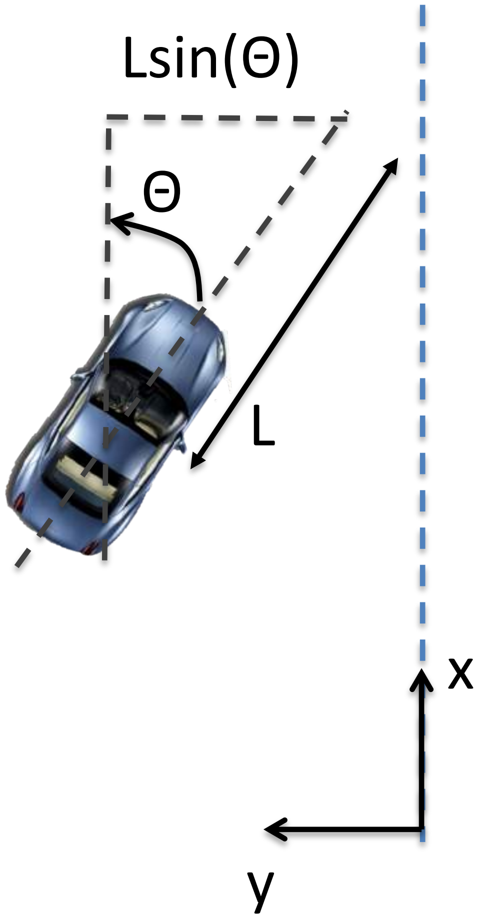


If error term is increasing quickly, we might want the controller to react quickly

→ Apply a *derivative gain*:

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

PID control: Derivative control



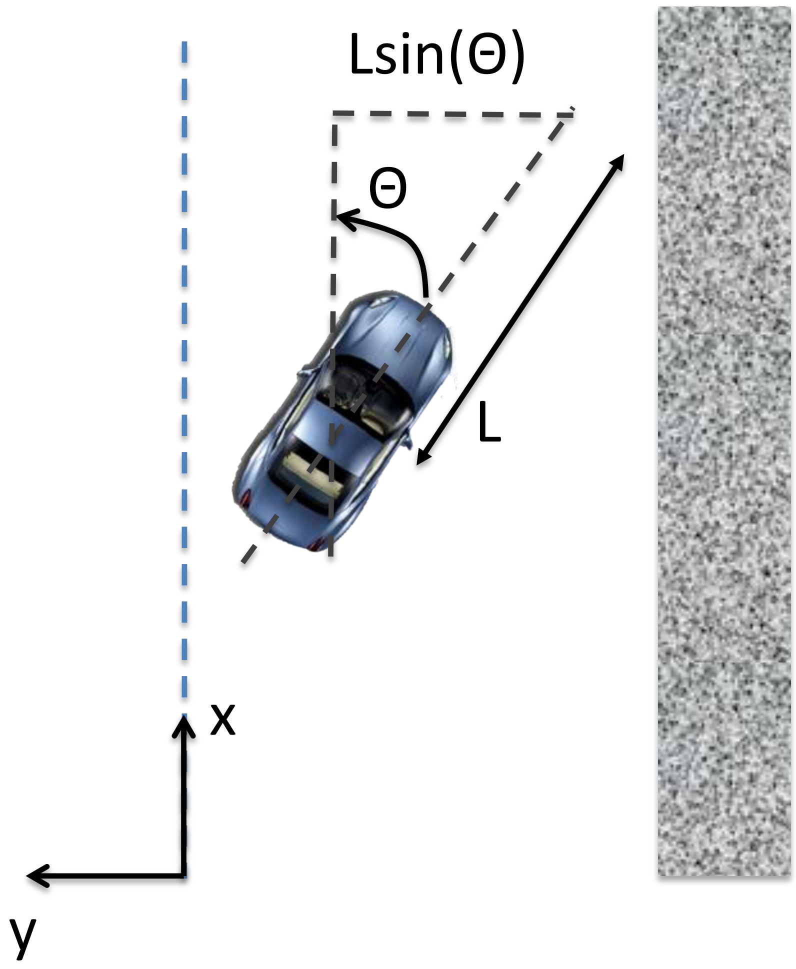
In shown scenario, controller tries to nullify y term.

→ Steer right so y decreases → y' < 0

→ Error derivative
$-y' - L\cos(\Theta)\,\Theta' > 0$

→ Trajectory correction takes longer (smoothing out)

PID control: Integral control



Integral control is proportional to the *cumulative* error

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

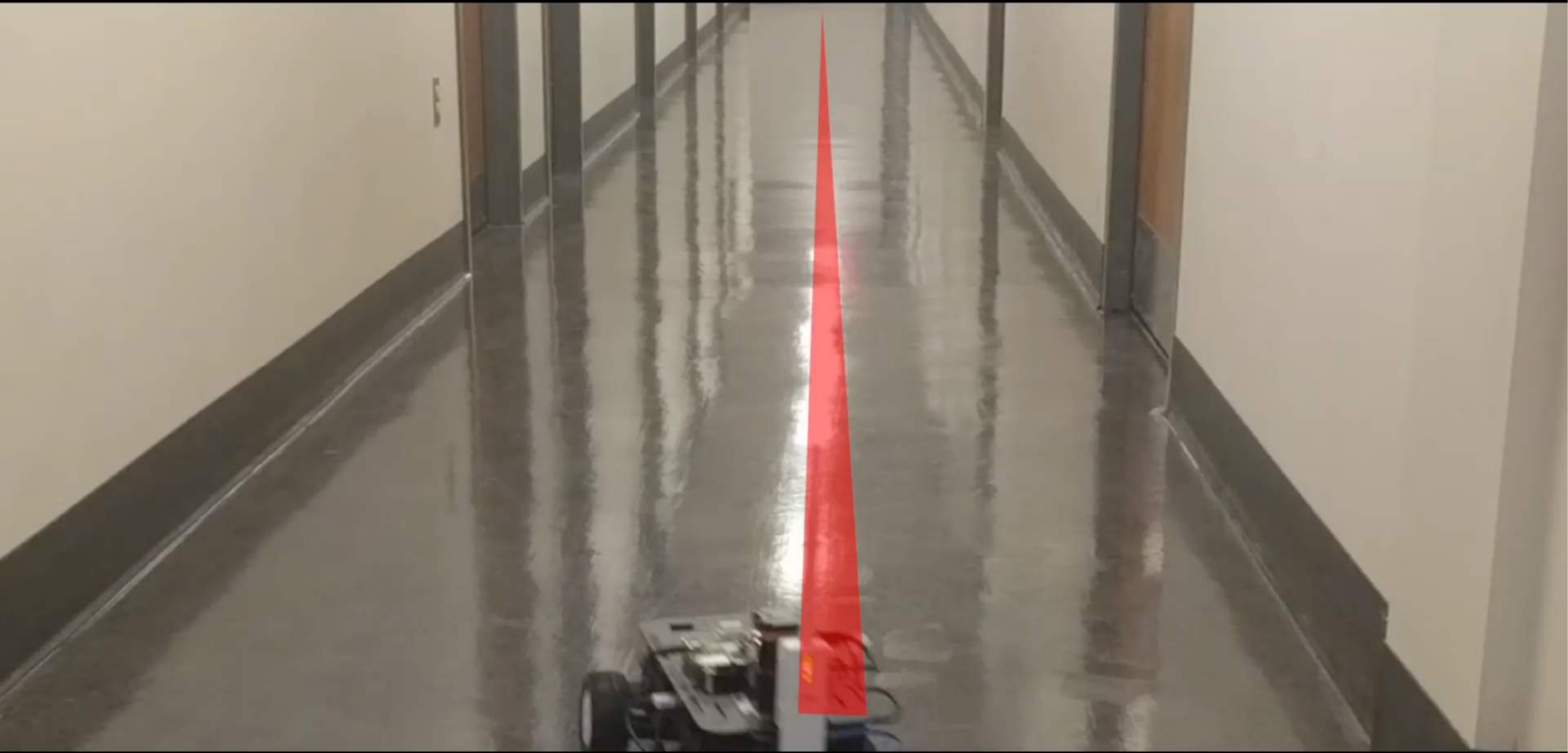
Where E(t) is the integral of the error up to time t (from a chosen reference time)

PID control: tuning the gains

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

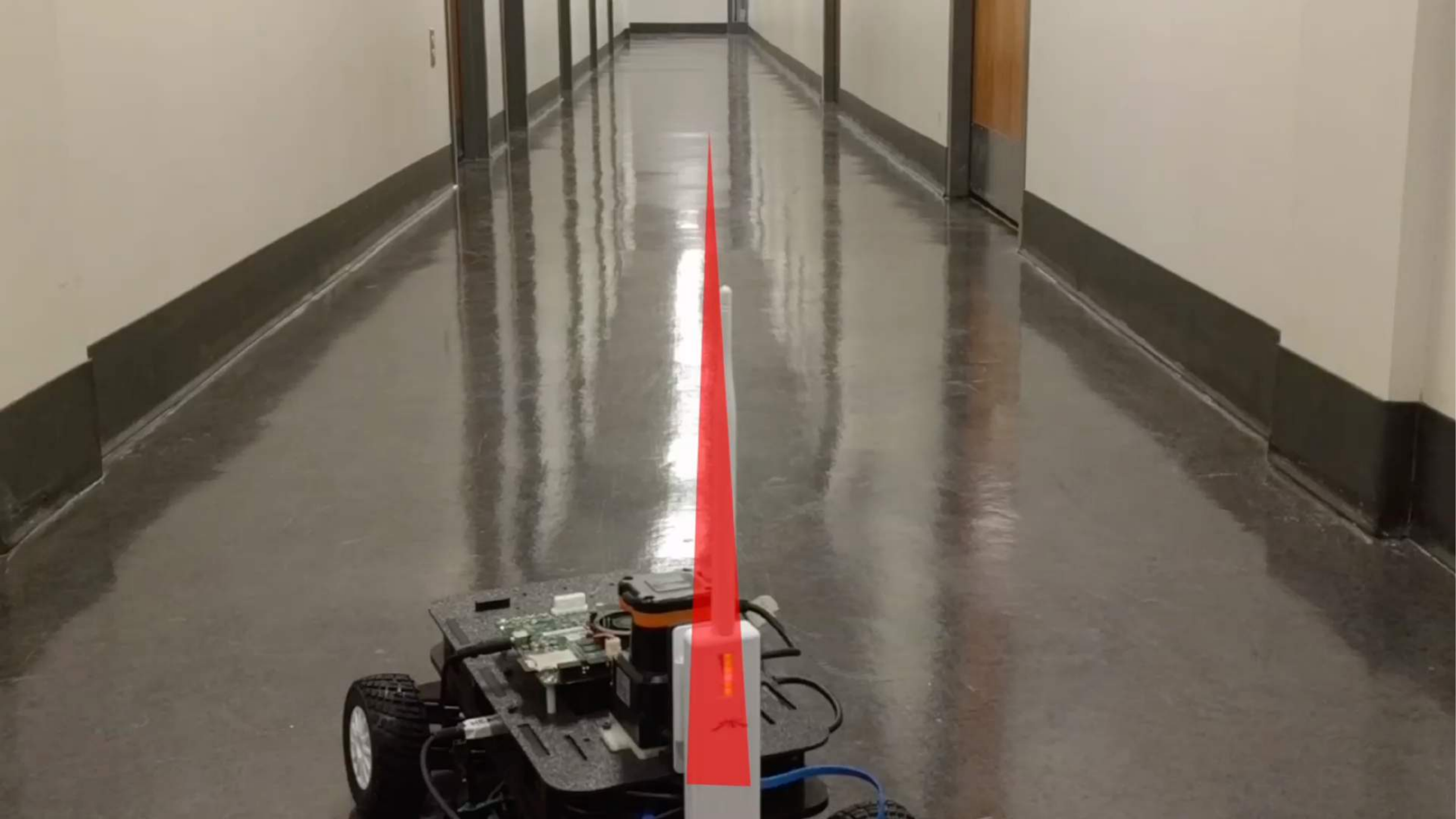
PID control: tuning the gains

- Reduce K_p → less responsive to error magnitude
  - $K_p = 5$
  - $K_i = 0$
  - $K_d = 0.09$

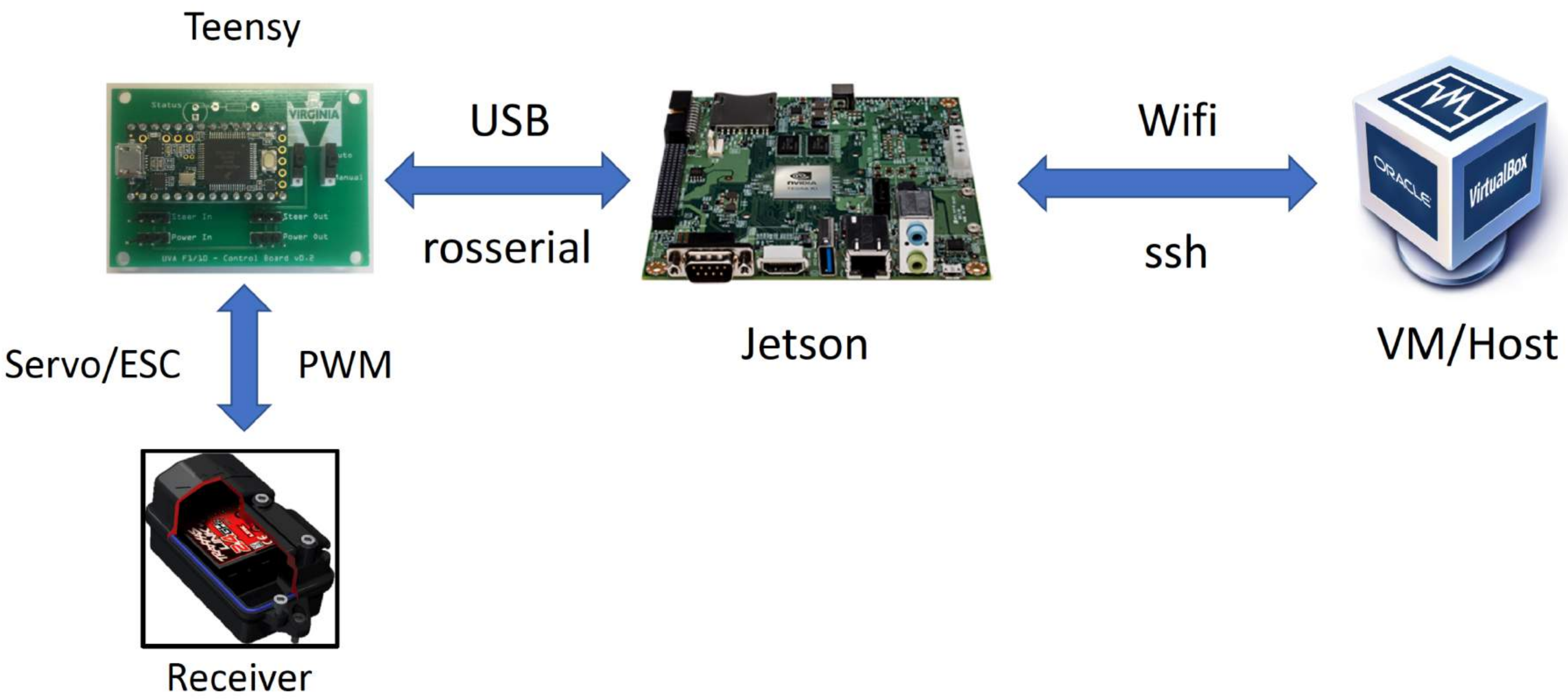


PID control: tuning the gains

- Include K_i → overly sensitive to accumulating error → over-correction
  - $K_p = 14$
  - $K_i = 2$
  - $K_d = 0.09$



So far...

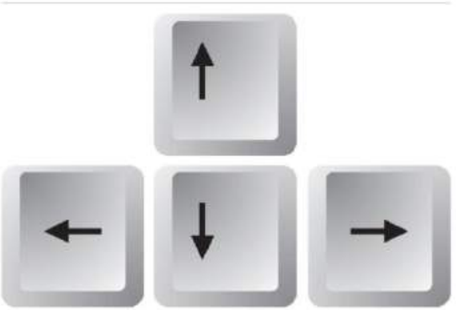

Teensy
USB
rosserial
Wifi
ssh
Jetson
VM/Host
Servo/ESC
PWM
Receiver

keyboard.py

Publishes topic – **drive_parameters**

Custom message type: **drive_param**

Subscribes to– **drive_pwm**



Remote connection
SSH

talker.py

Subscribes to– **drive_parameters**

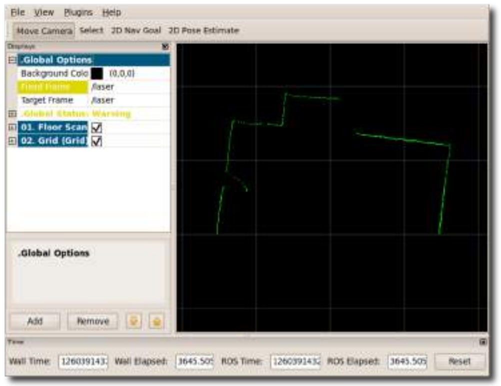
Publishes topic – **drive_pwm**

control.py

Publishes to topic – **drive_parameters**

Subscribes to : **error**

ROS over Network



Subscribes to– **drive_pwm**

Remote connection

SSH >> rviz visualization

dist_finder.py

Publishes topic – **error** | message type pid_input.msg

Subscribes to topic – **scan**