F1TENTH Boot Camp

Day 2 : Reactive Methods

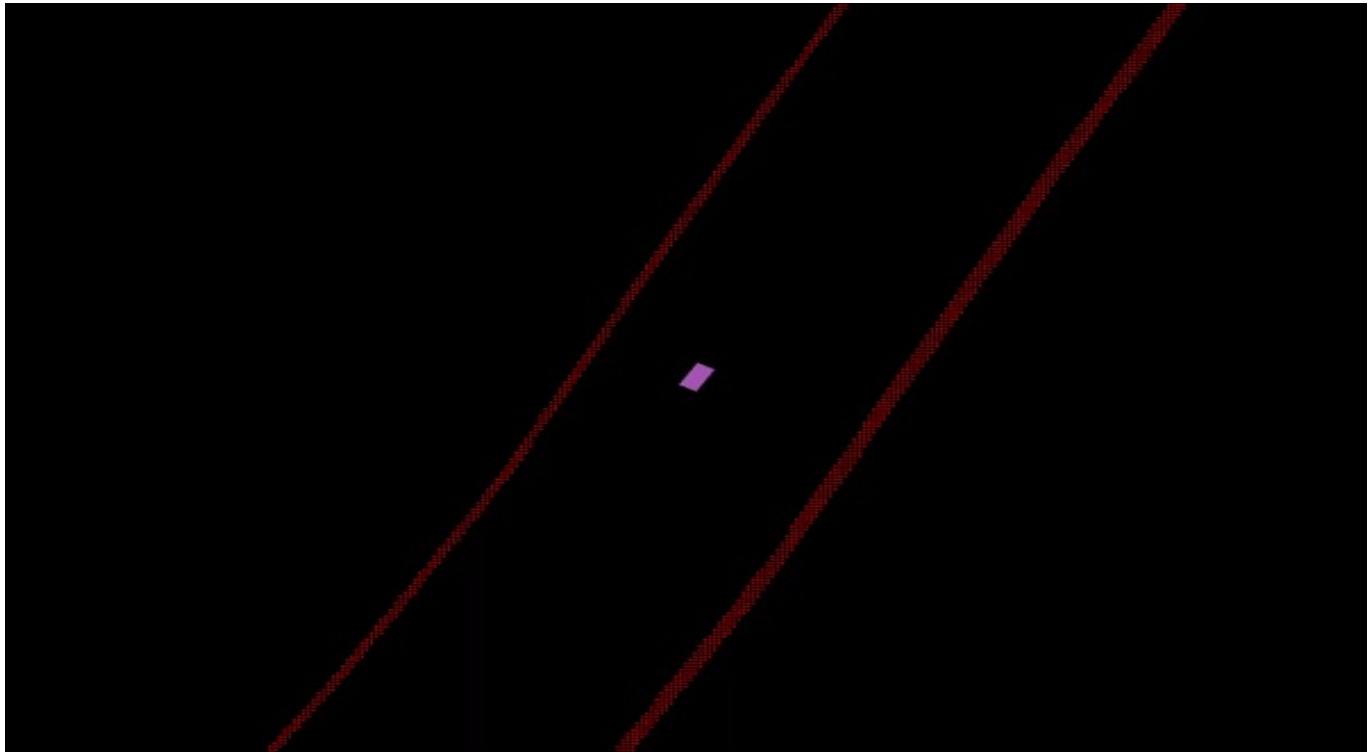
Week6-1

Reactive Strategy

2023.04.04

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

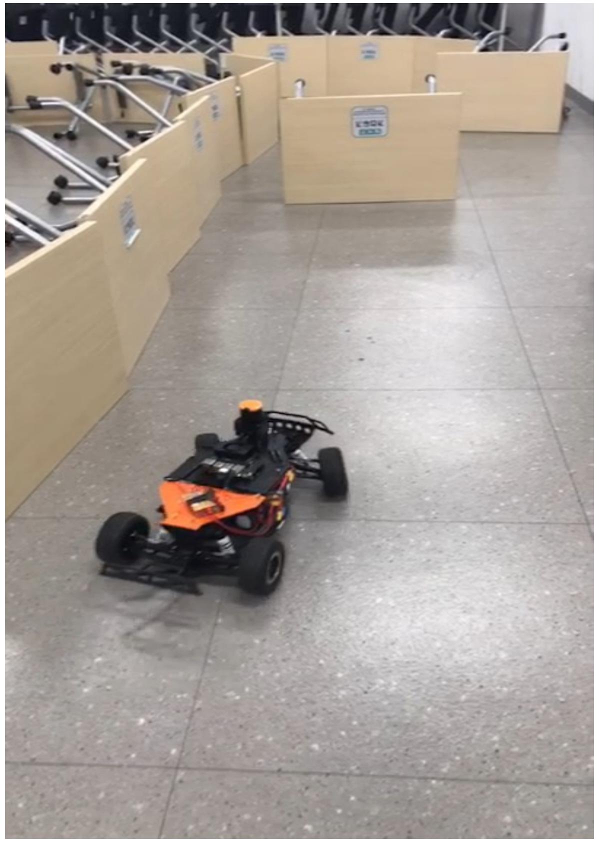
Reactive Methods



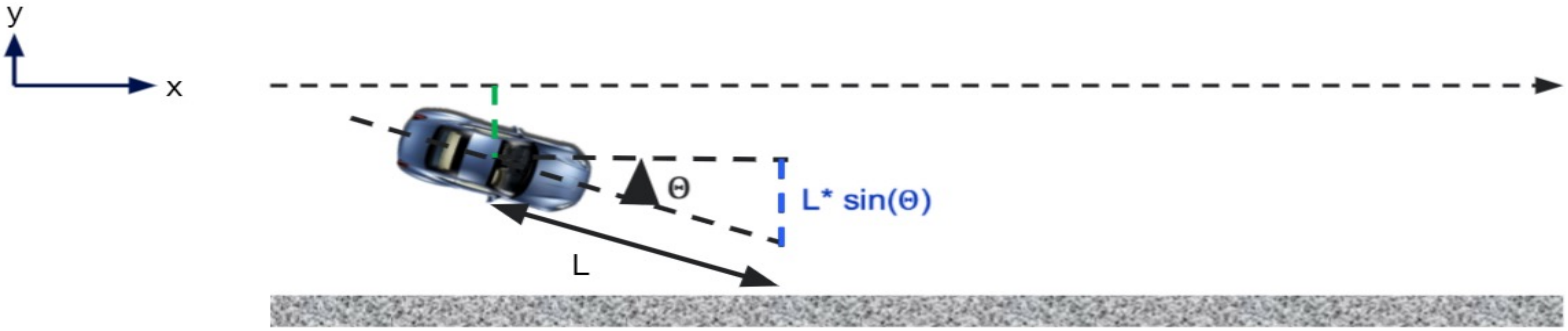
How do we get the decent performance without a map?

Reactive Methods: Wall-Following

- **Problem:** Drive the car around the track rather than emergent braking
- **Understand:** Basic of PID, how to compute error, failure modes
- **Implement:** Wall following in simulation and on the vehicle



Review: PID Controller Design

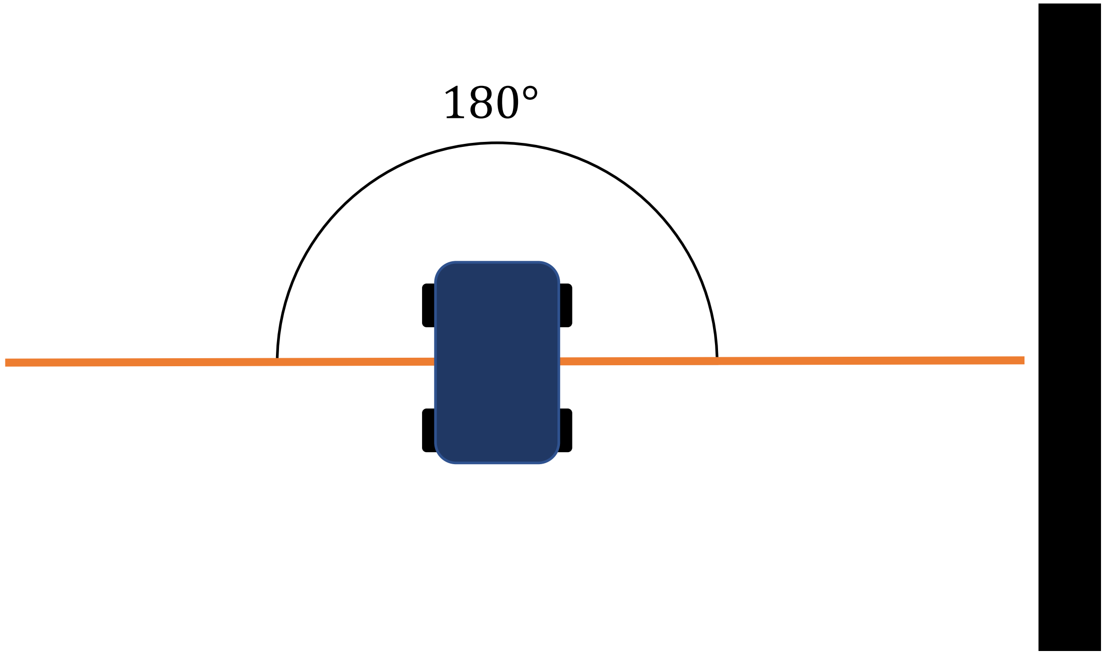


- Control Input : Steering Angle θ
- Velocity will be held constant
- Goal :
  - Keep the car driving along the Centerline : y = 0
  - Drive Parallel to the walls : θ = 0 → Better : $\mathrm{L} * \sin(\theta) = 0$

  → **Can do more elaborate geometric design using vehicle kinematics model!**

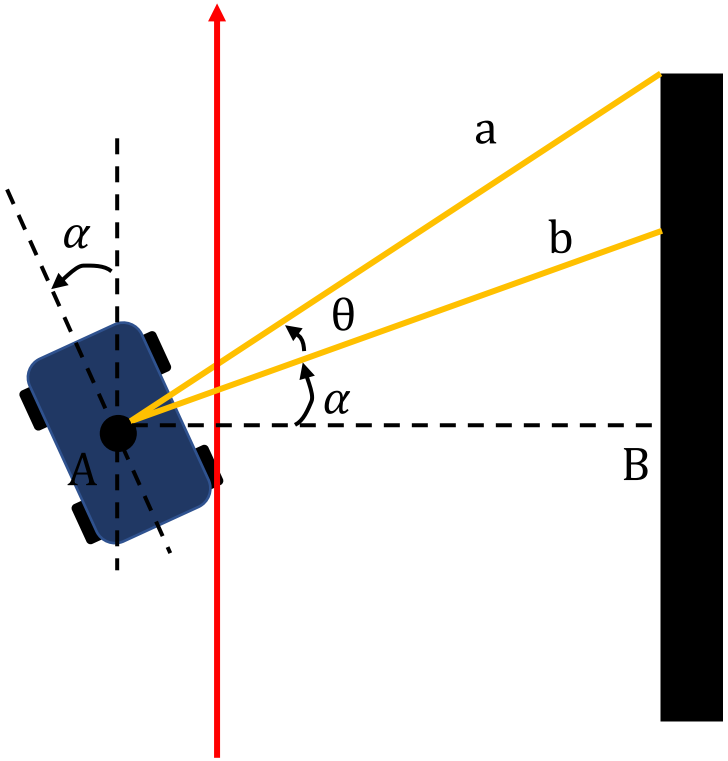
Wall Following

- Lidar scan Angles:



Wall Following

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

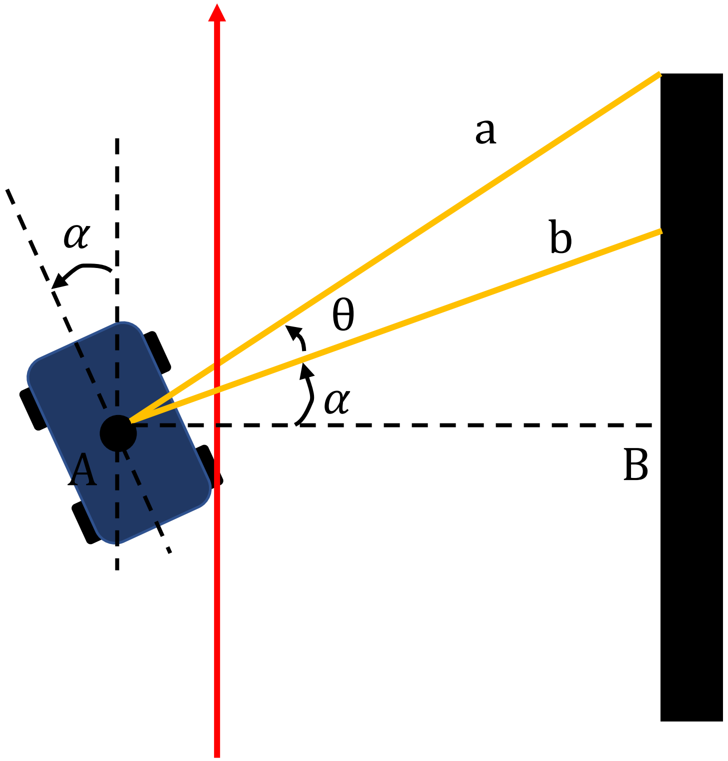


$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right)$$
$$\overline{AB} = b\cos(\alpha)$$

Error = desired trajectory - $\overline{AB}$?

Wall Following

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



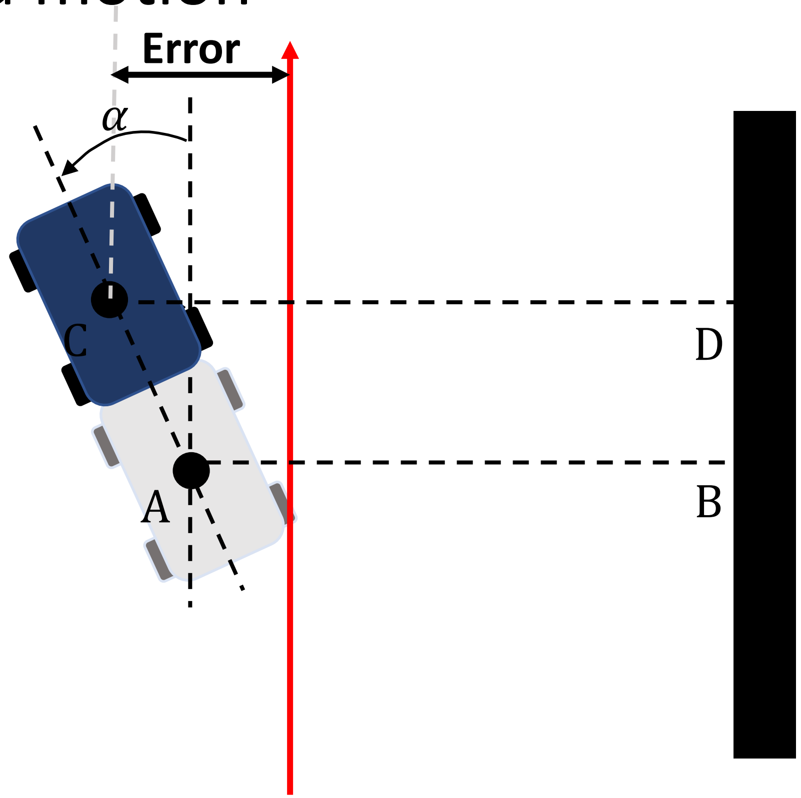
$$\alpha = \tan^{-1}\left(\frac{a\cos(\theta) - b}{a\sin(\theta)}\right)$$

$$\overline{AB} = b\cos(\alpha)$$

Error = desired trajectory - $\overline{AB}$? Not quite

Wall Following

- Account for the forward motion of the car



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

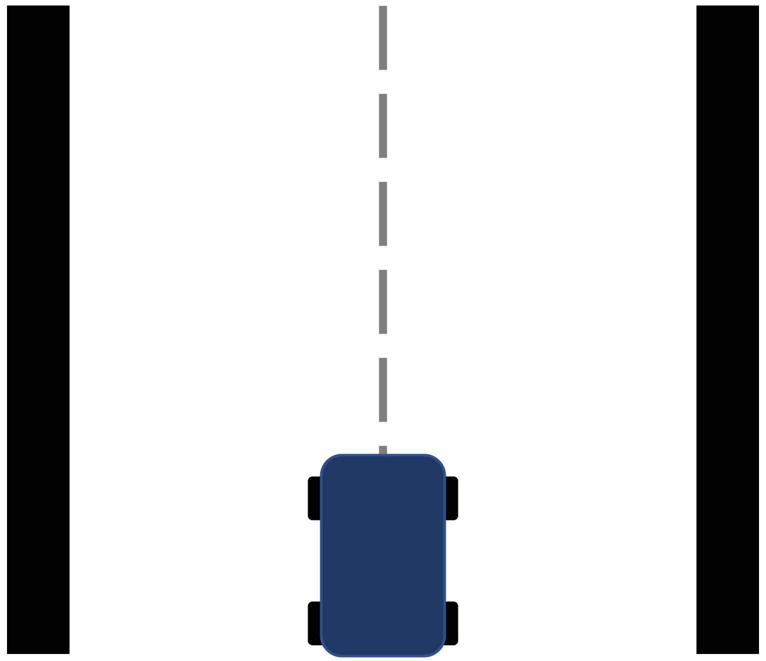
$$\overline{AB} = b\cos(\alpha)$$

$$\overline{CD} = \overline{AB} + \overline{AC}\sin(\alpha)$$

Error = desired trajectory - $\overline{CD}$

PID Control for Wall Following

- Control objectives:
  - Keep the car driving along the centerline
  - Parallel to the walls



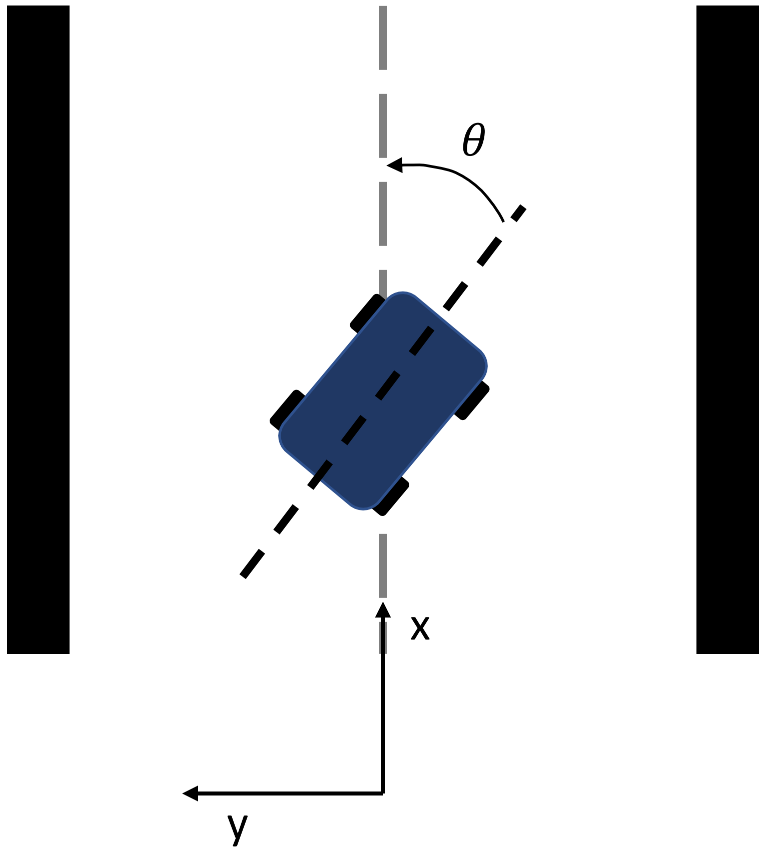
PID Control for Wall Following

- Control objectives:
  - Keep the car driving along the centerline

    $$y = 0$$

  - Parallel to the walls

    $$\theta = 0$$



PID Control for Wall Following

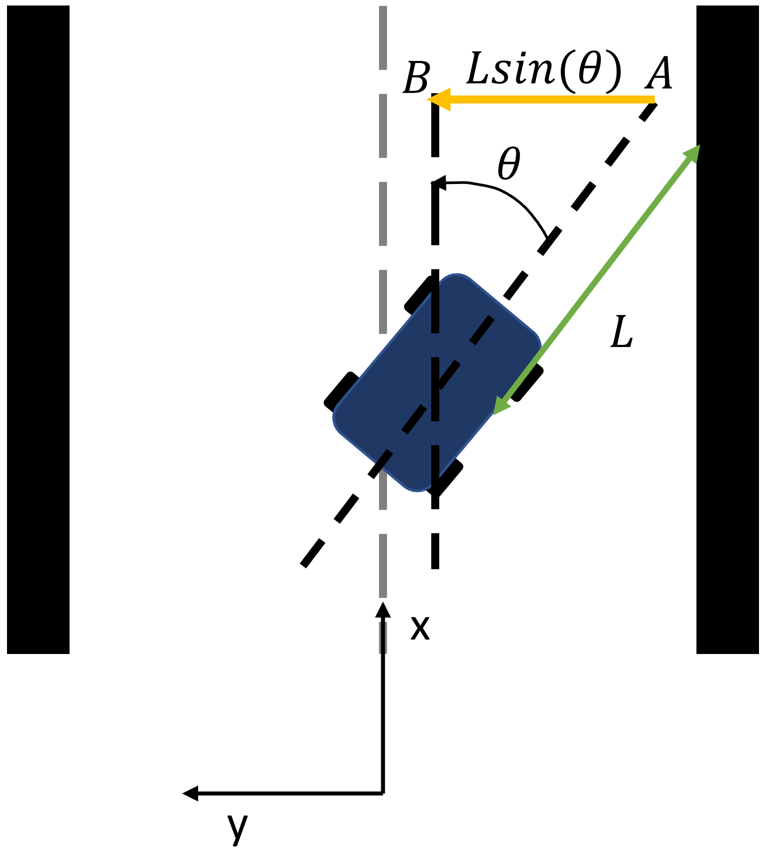
- Control objectives:
  - Keep the car driving along the centerline

$$y = 0$$

  - After driving L meters, it is still on the centerline:

Horizontal distance after driving L meters

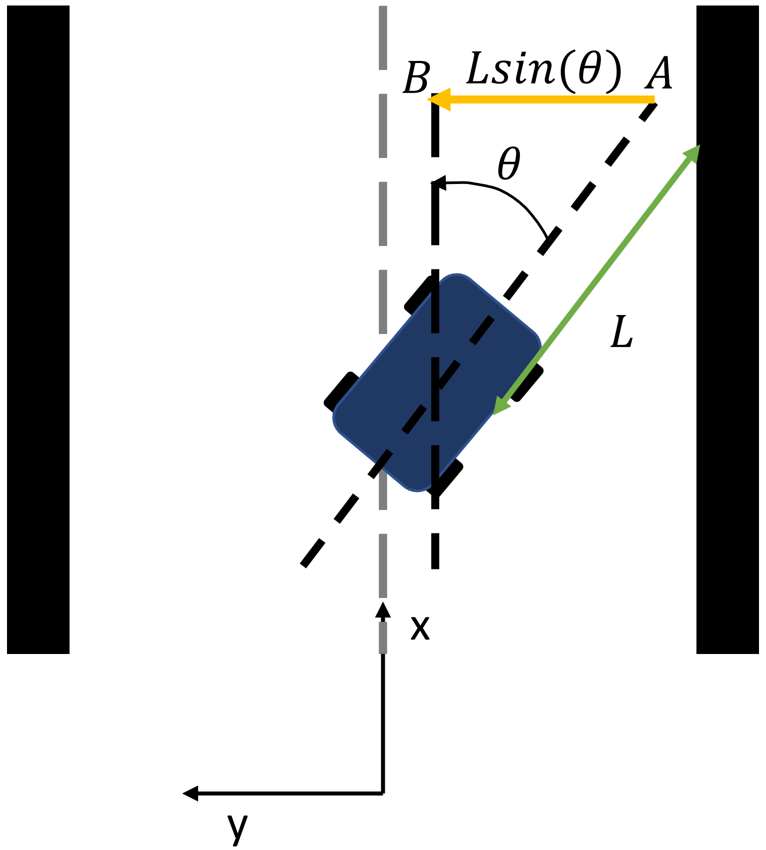
$$Lsin(\theta) = 0$$



PID Control for Wall Following

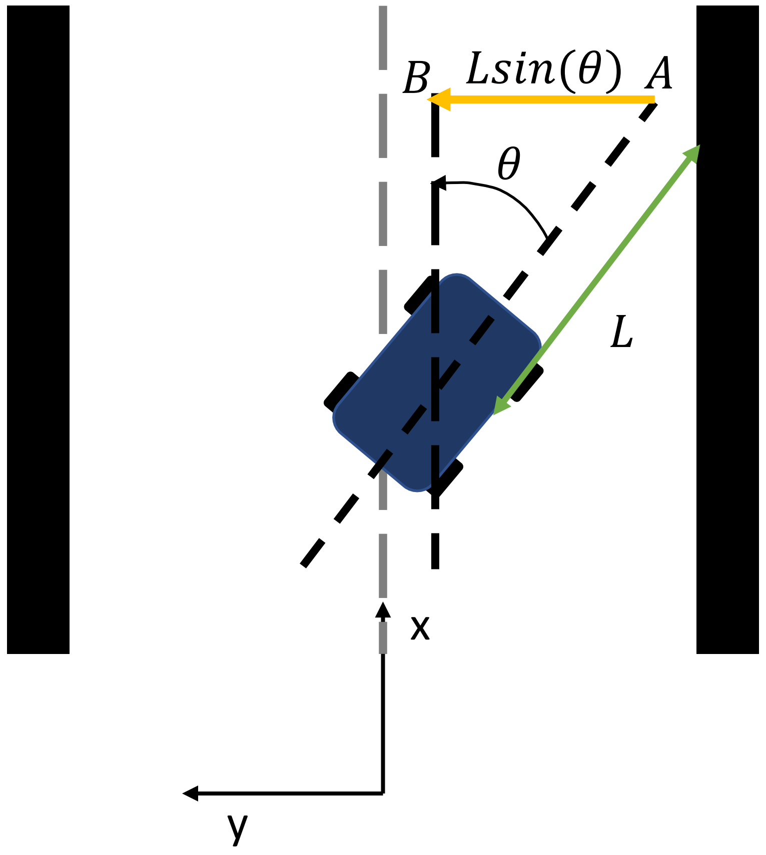
- Control input
  - Steering angle θ
  - We will hold the velocity constant

  - How do we control the steering angle to keep
$$y = 0,\ Lsin(\theta) = 0$$
    as much as possible?



PID Control for Wall Following

- Error term
  - Want both y and $Lsin(\theta)$ to be zero
    → Error term $e(t) = -(y + Lsin(\theta))$



PID Control for Wall Following

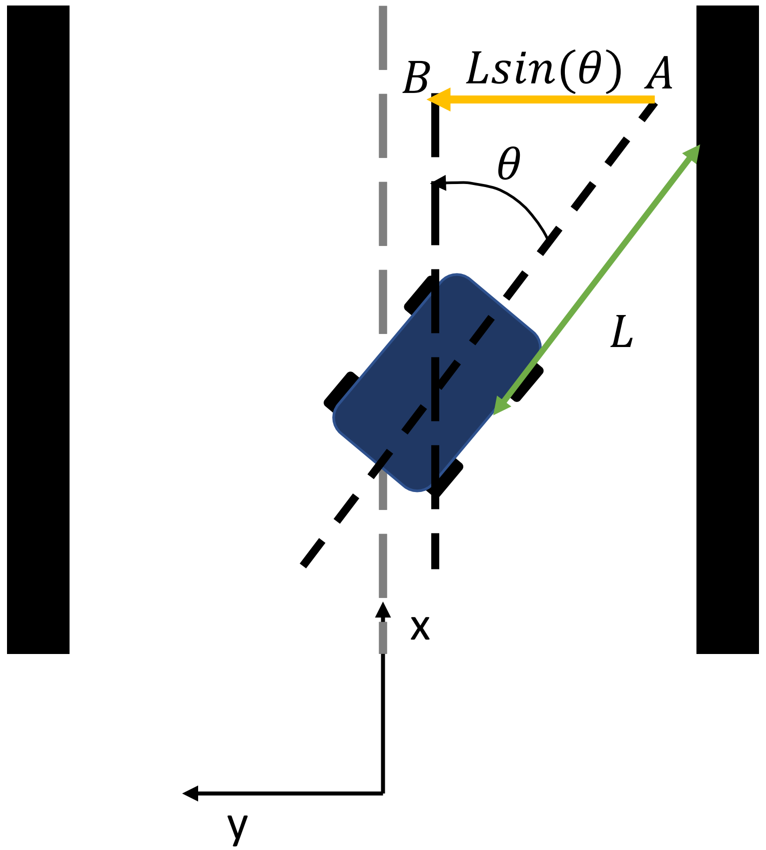
- Computing input
  - When $y > 0$, car is to the left of centerline

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

    → Want to steer right: $\theta < 0$

  - Set desired angle to be

$$\theta_d = K_p(-y - Lsin(\theta))$$

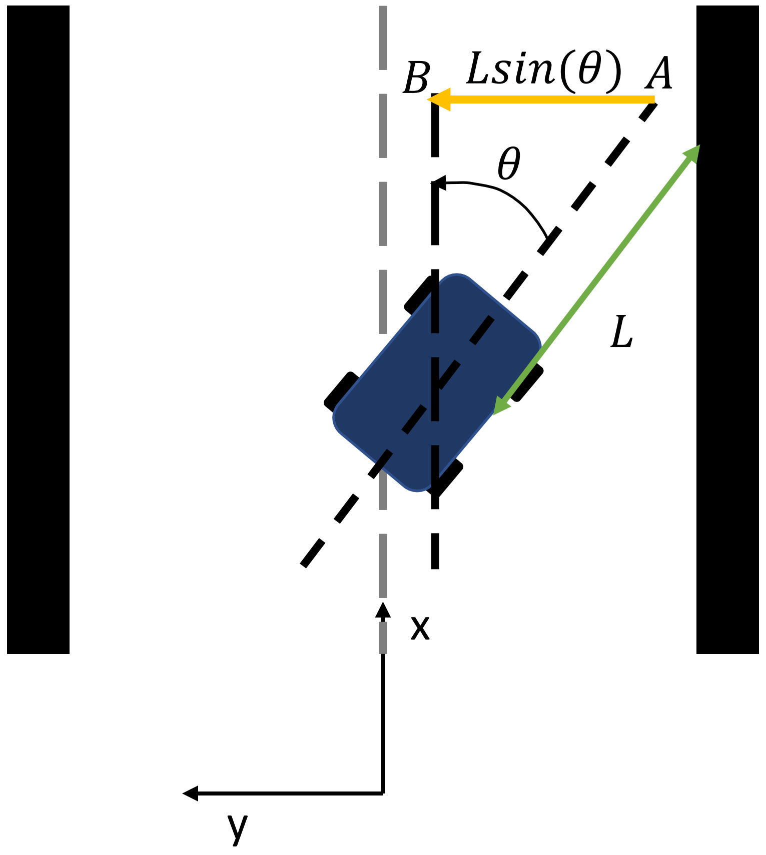


PID Control for Wall Following

- Computing input
  - When $\mathrm{y} < 0$, car is to the right of centerline
    → Want to steer left: $\theta > 0$
  - When $Lsin(\theta) < 0$, we will be to the right of centerline in L meters
    → Want to steer right: $\theta > 0$

  - Set desired angle to be

$$\theta_d = K_p(-y - Lsin(\theta))$$



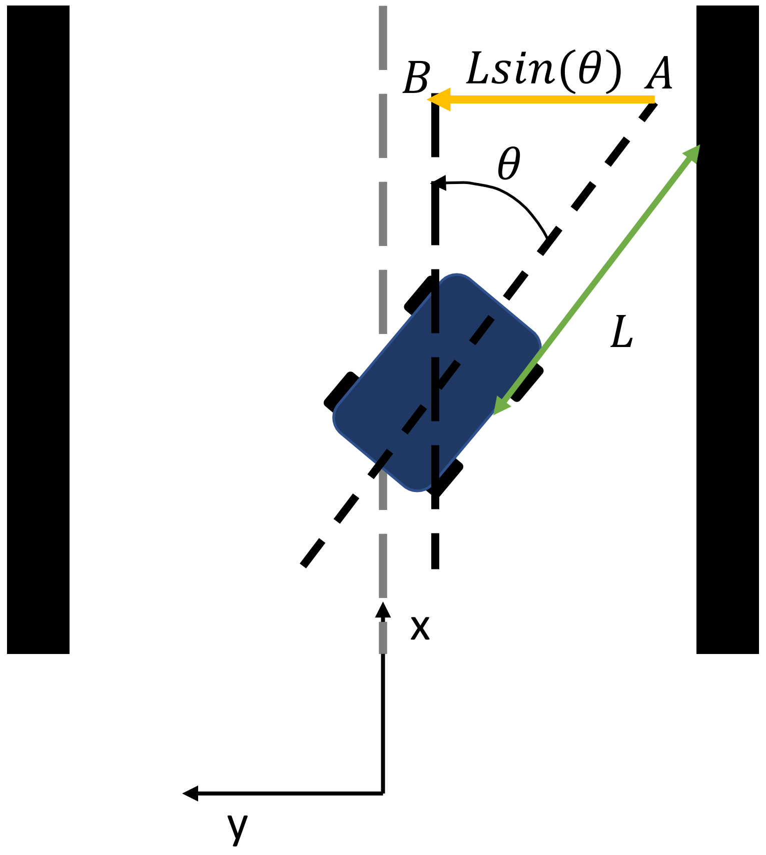
HMCL
High-assurance Mobility Control Lab

PID Control for Wall Following

- **Proportional control**

$$\theta_d = K_p\left(-y - Lsin(\theta)\right) = CK_p e(t)$$

  - This is **P**roportional control
  - The extra C constant is for scaling distances to angles

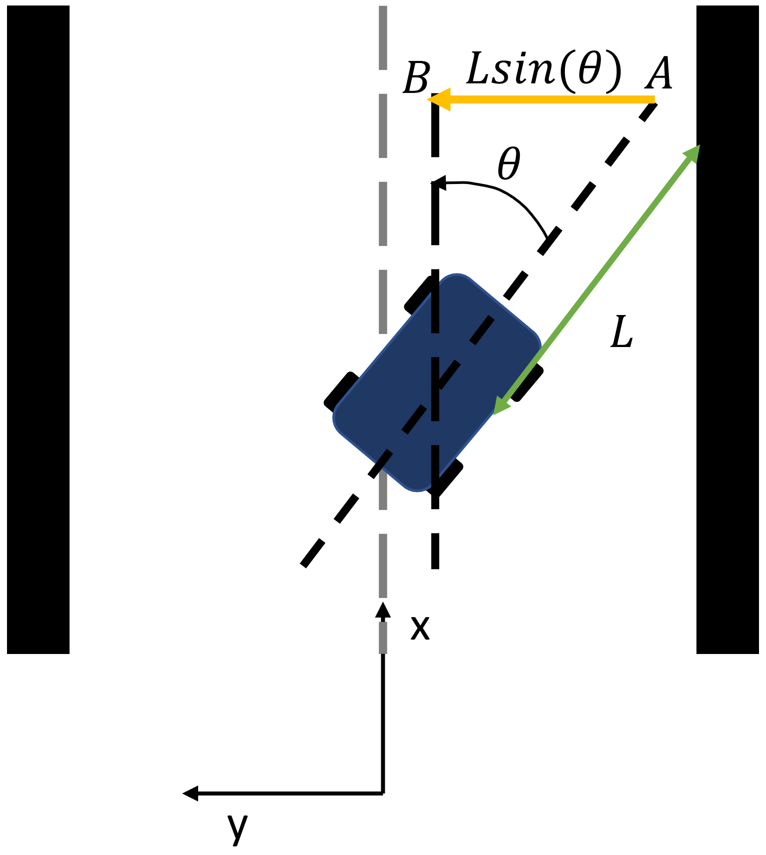


PID Control for Wall Following

- **Derivative control**
  - If error term is increasing quickly, we might want the controller to react quickly

    → Apply a **D**erivative gain:

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

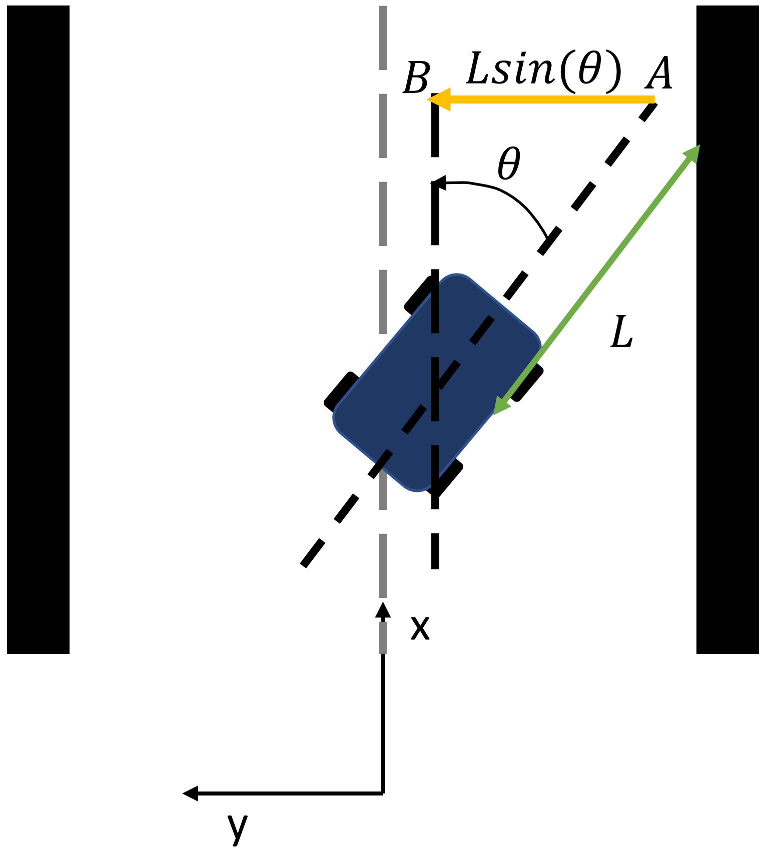


PID Control for Wall Following

- **Integral control**
  - Integral control is proportional to the cumulative error

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

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

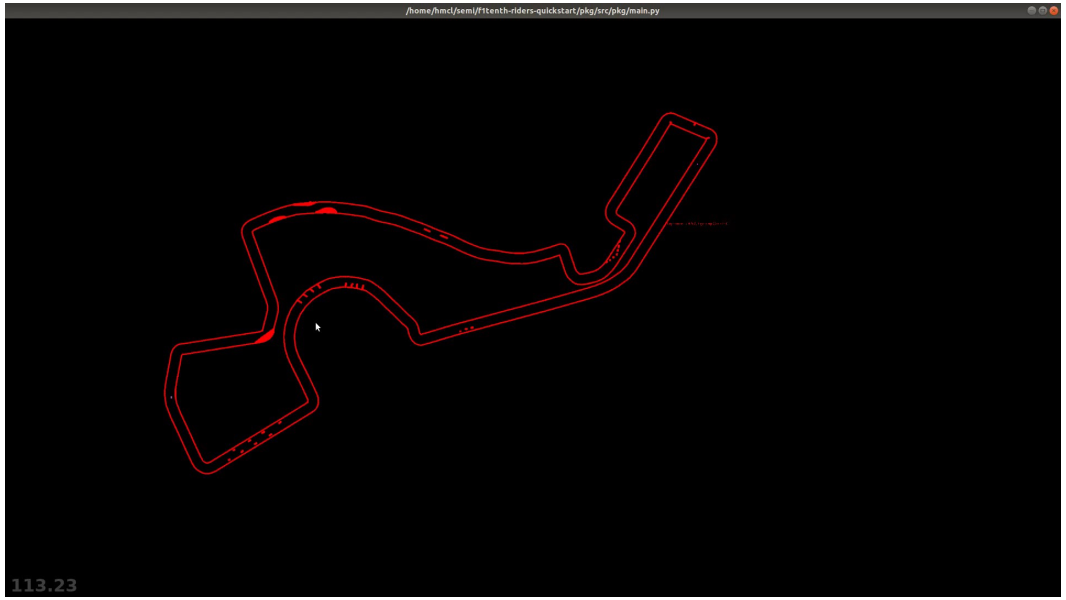


Obstacle Avoidance without a Map

- Can we Wall Following to navigate the complex racing track?

- If we build a map we can plan an obstacle free path to follow (=map-based method)

- **Challenge:** Locally avoid obstacles (=Reactive method) without any map information



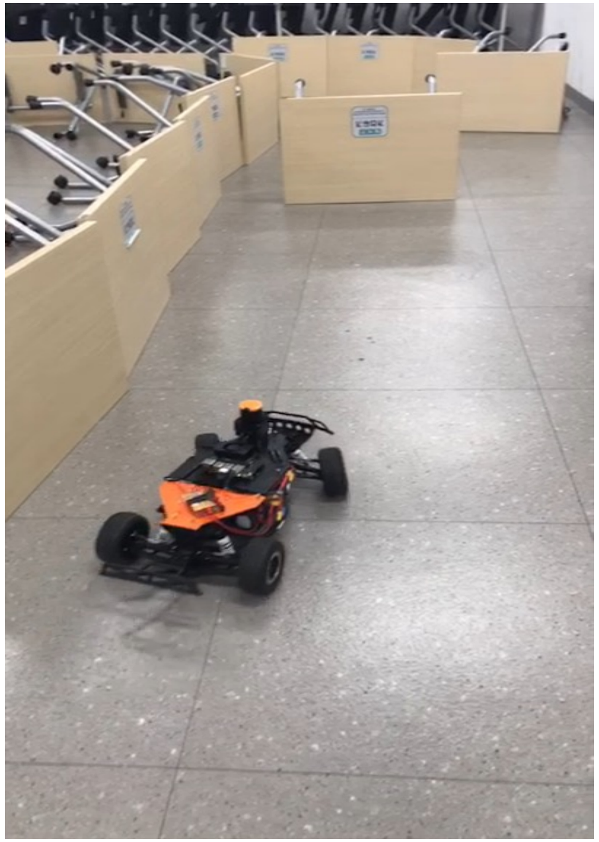
How do we get that level of performance without a map?

Reactive Methods: Gap-Following

- **Problem:** How can avoid obstacles

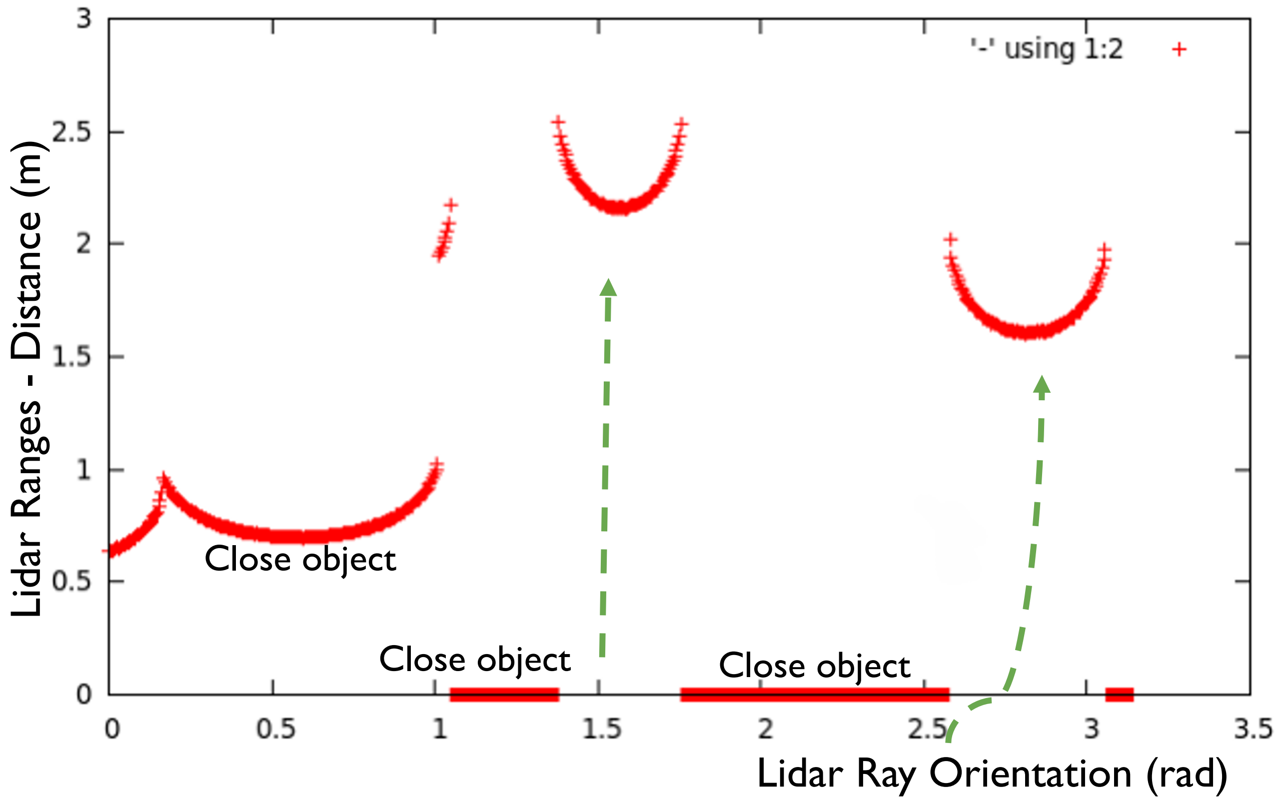
- **Understand:** Basics of reactive navigation, avoidance on both static and dynamic obstacles

- **Implement:** Gap following in simulation and on the vehicle



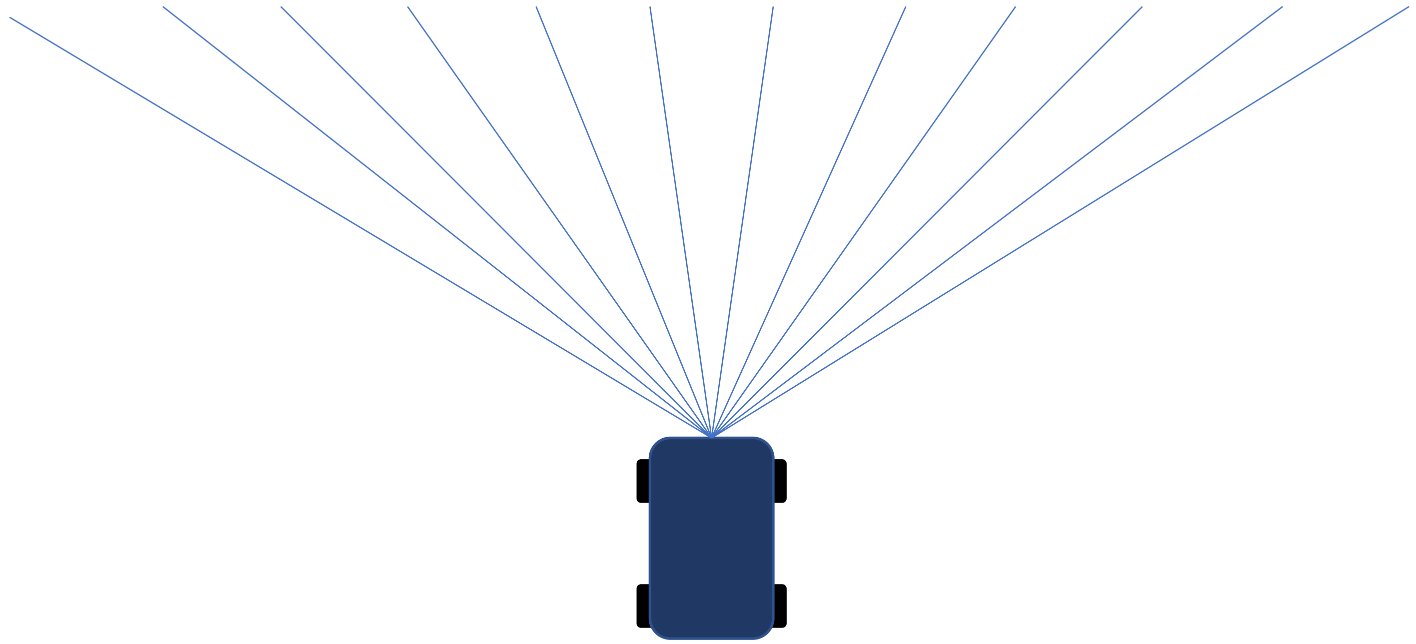
Sensing Gaps between Obstacles

- Sensing Obstacles from Lidar Measurements



Follow the Gap

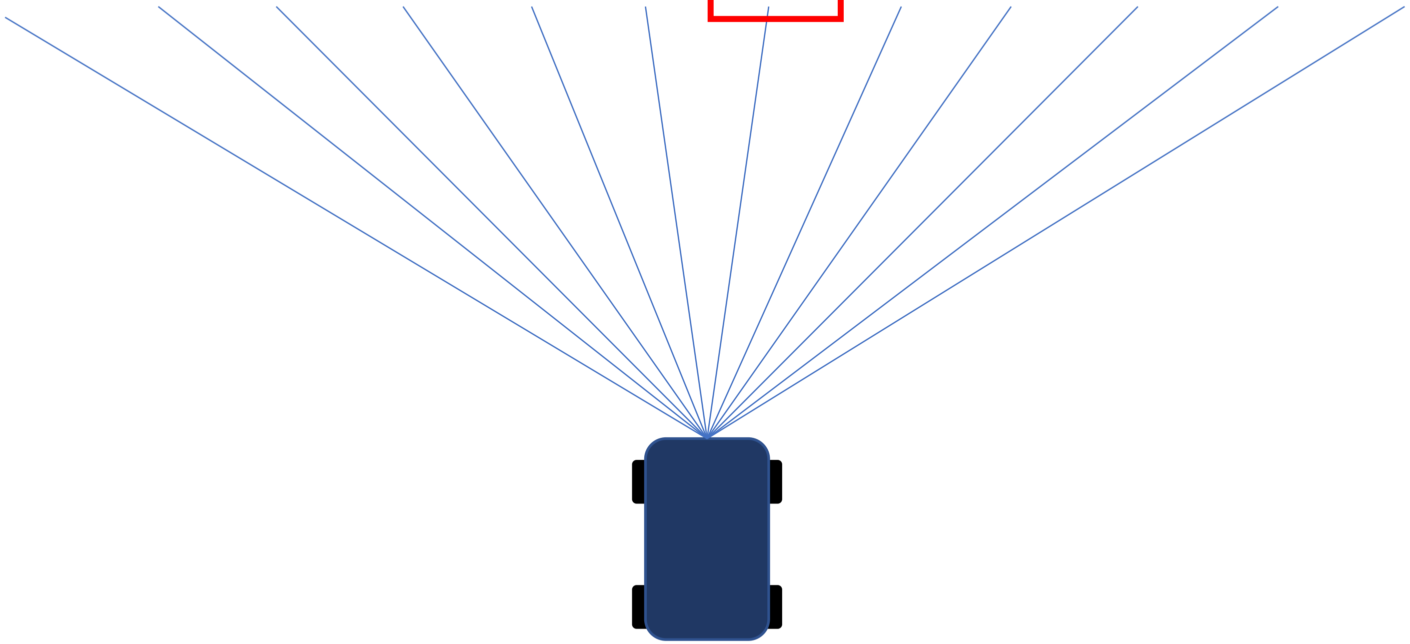
[0.5, 5.1, 6.0, 7.0, inf, 3.0, inf, 3.0, inf, 8.0, 5.0, 3.0]



Where should car go?

Follow the Gap

[0.5, 5.1, 6.0, 7.0, inf, 3.0, $\boxed{\text{inf,}}$ 3.0, inf, 8.0, 5.0, 3.0]

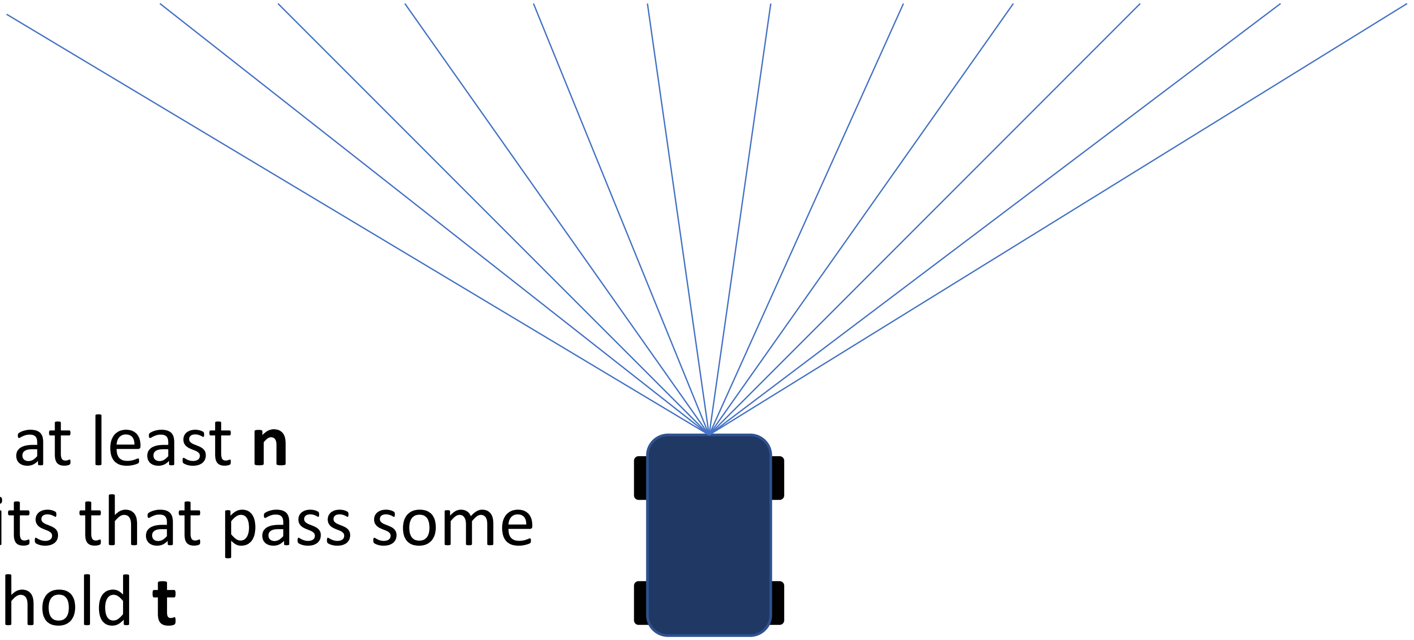


Furthest distance?

Why might this be wrong?

Follow the gap

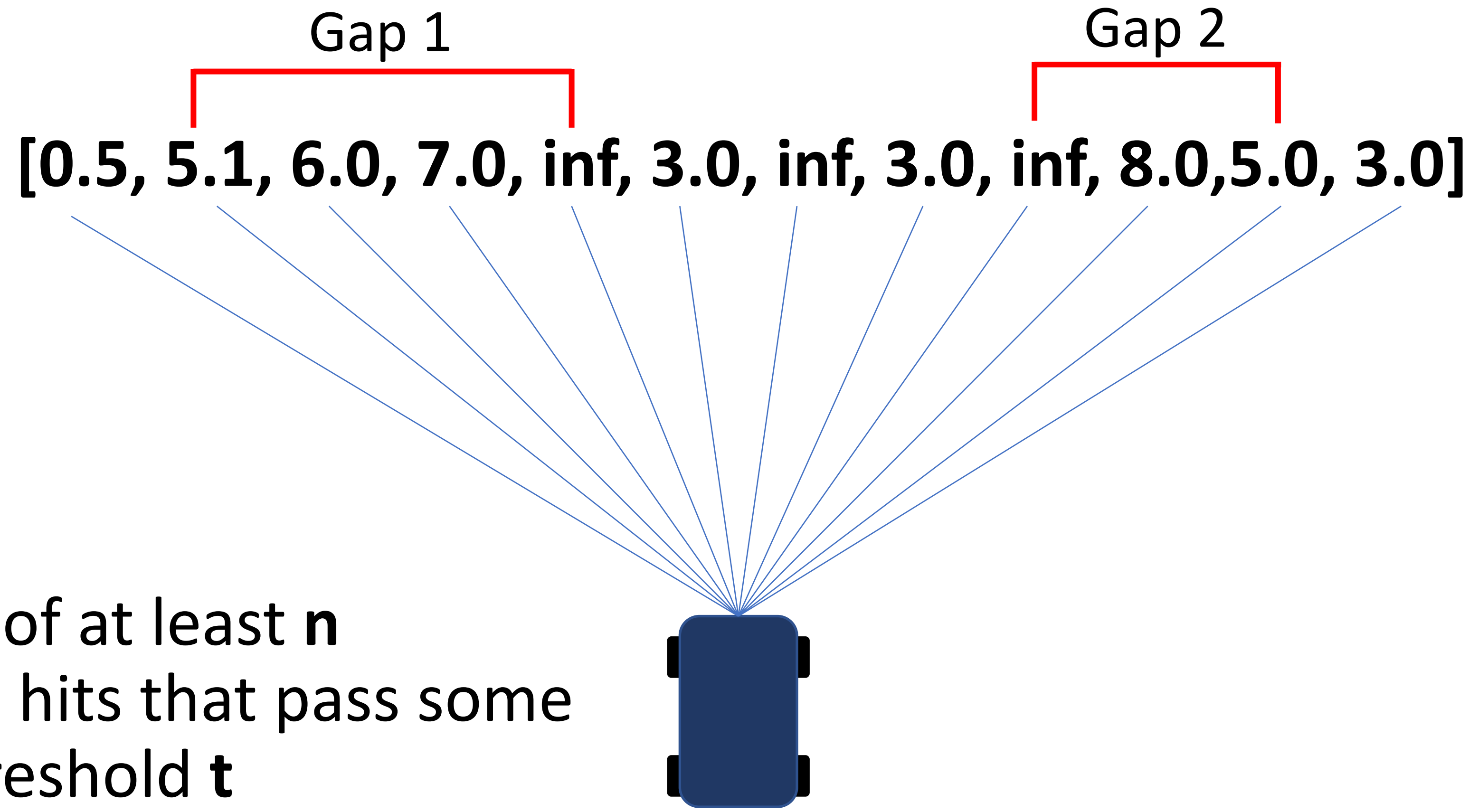
[0.5, 5.1, 6.0, 7.0, inf, 3.0, inf, 3.0, inf, 8.0,5.0, 3.0]



Gap: Series of at least **n** consecutive hits that pass some distance threshold **t**

n=3, t = 5.0

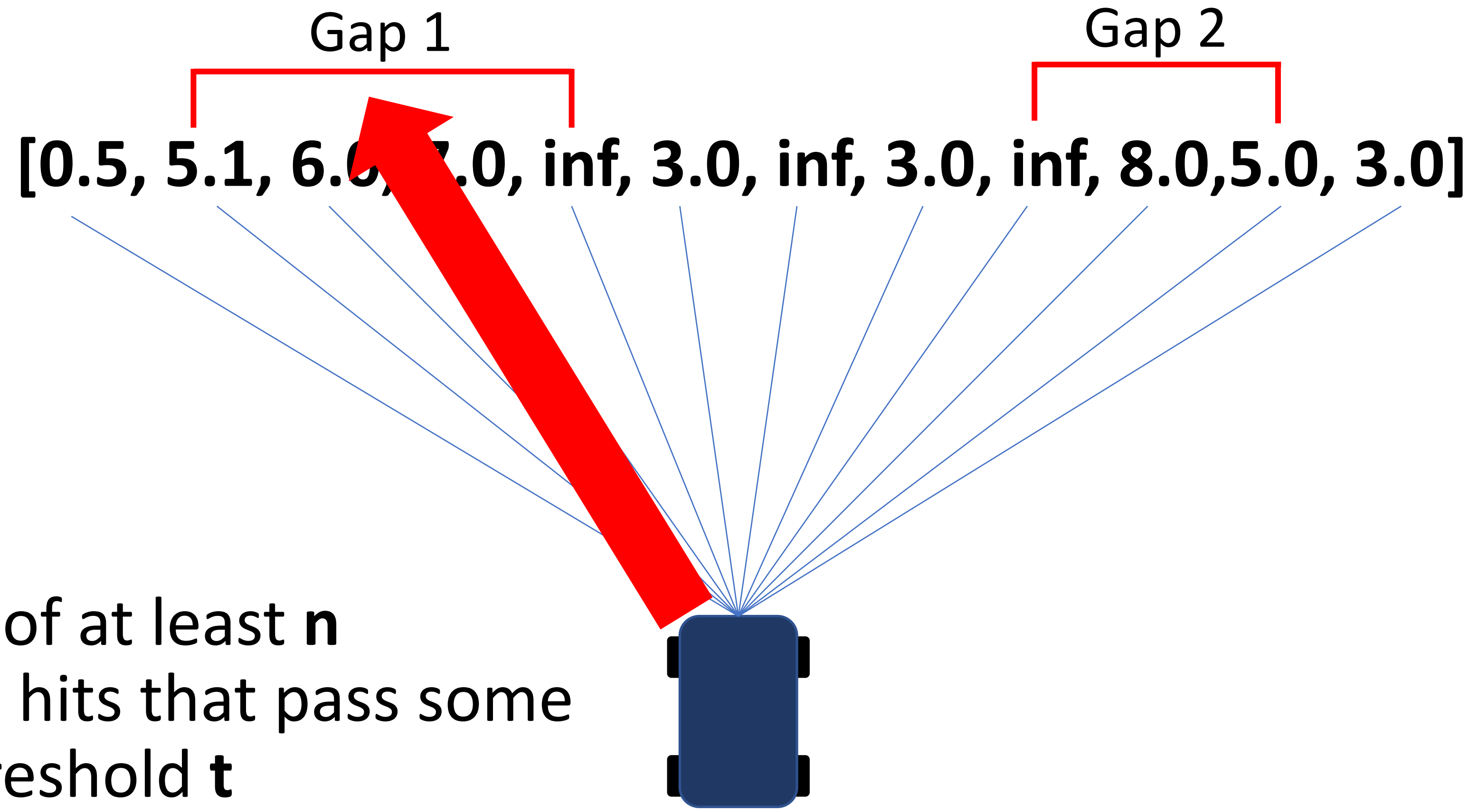
Follow the Gap



Gap: Series of at least **n** consecutive hits that pass some distance threshold **t**

n=3, t = 5.0

Follow the Gap

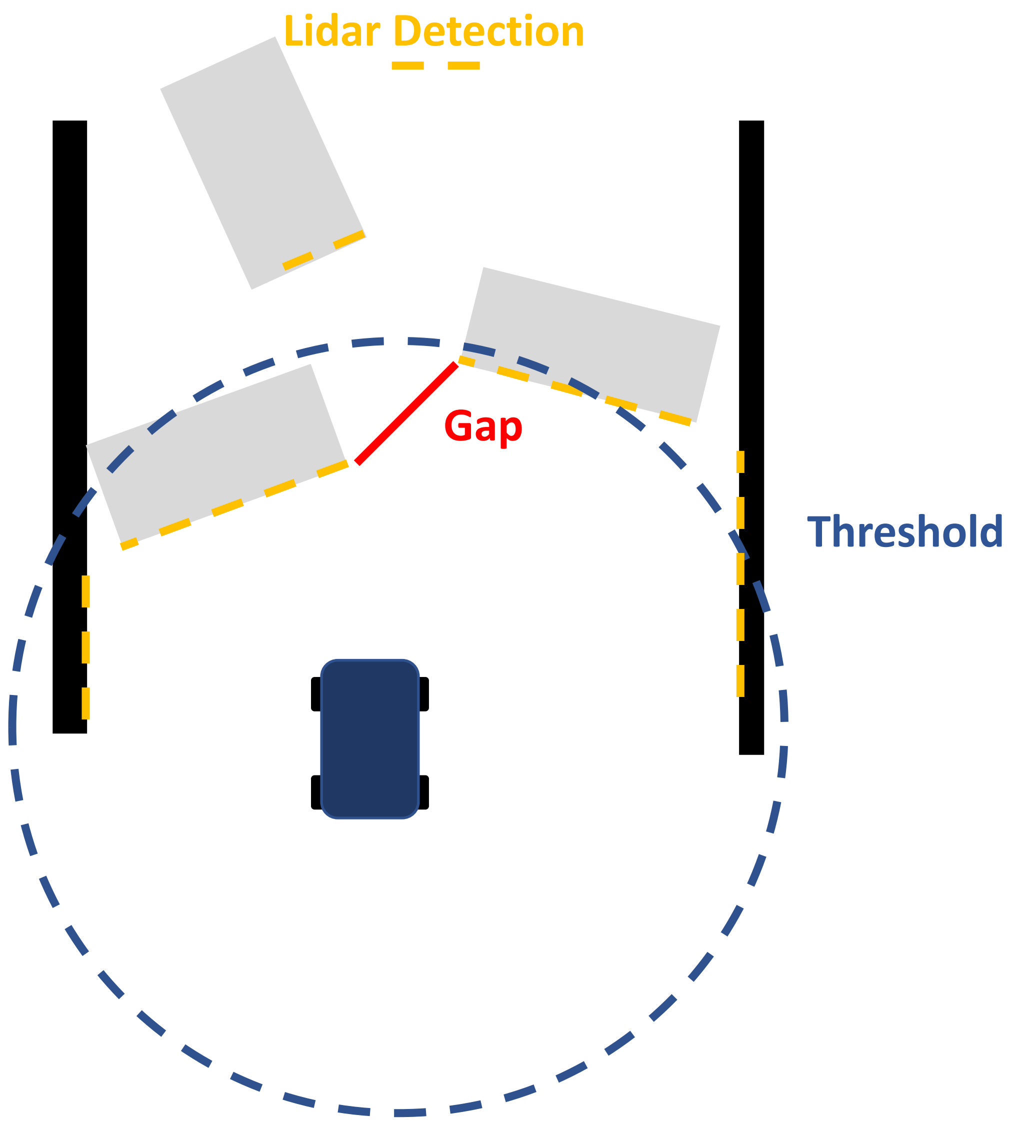


Gap: Series of at least **n** consecutive hits that pass some distance threshold **t**

n=3, t = 5.0

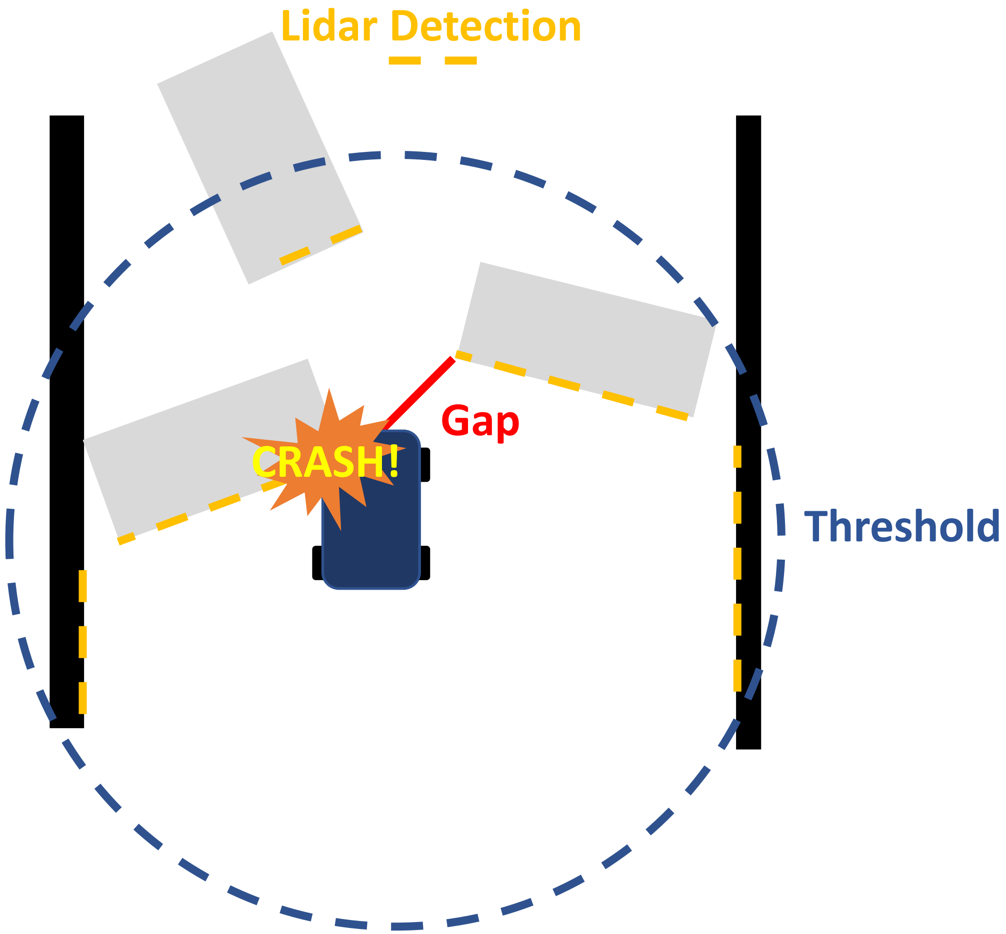
Follow the Gap

- Why Naïve “Follow the Gap” doesn’t work



Follow the Gap

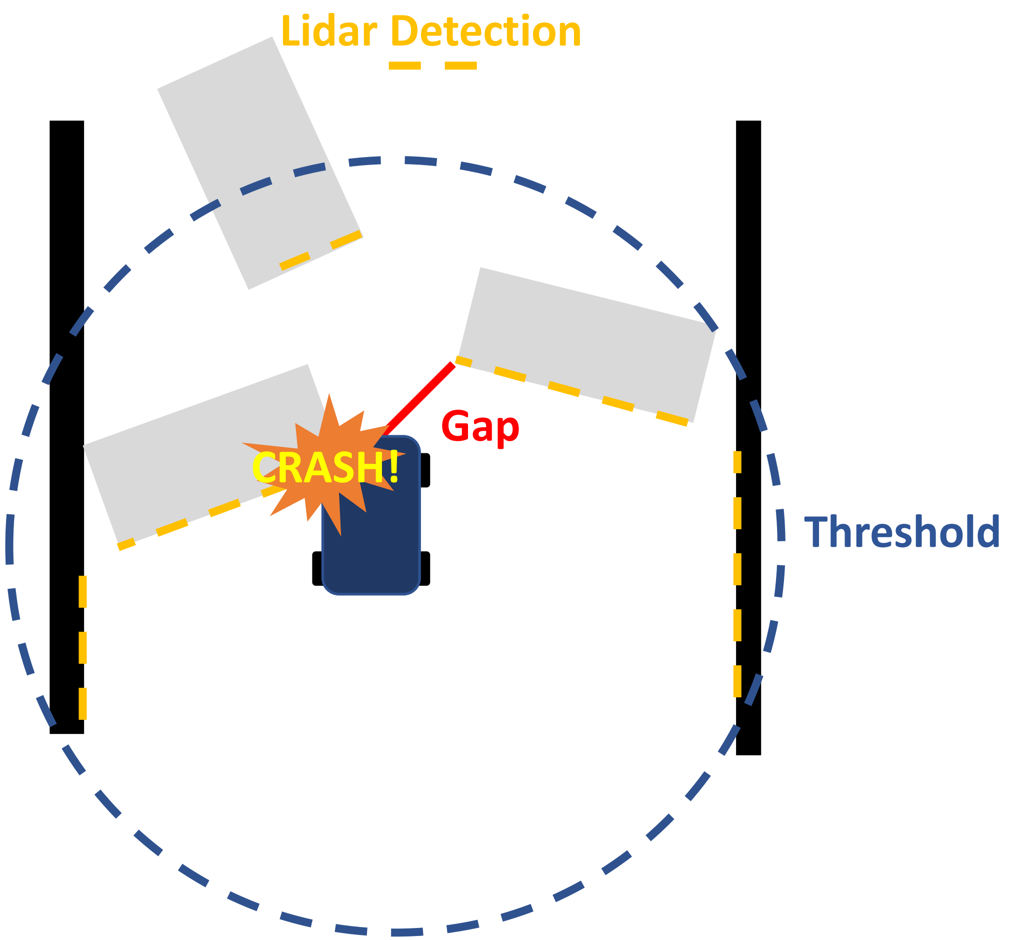
- Why “Find the Largest Gap” doesn’t work



Follow the Gap

- The idea: “Seek out the **largest gap**”

  - Works fine for holonomic robots(e.g. turtlebots)
  - Works fine for non-holonomic robots in environment with sparse obstacles

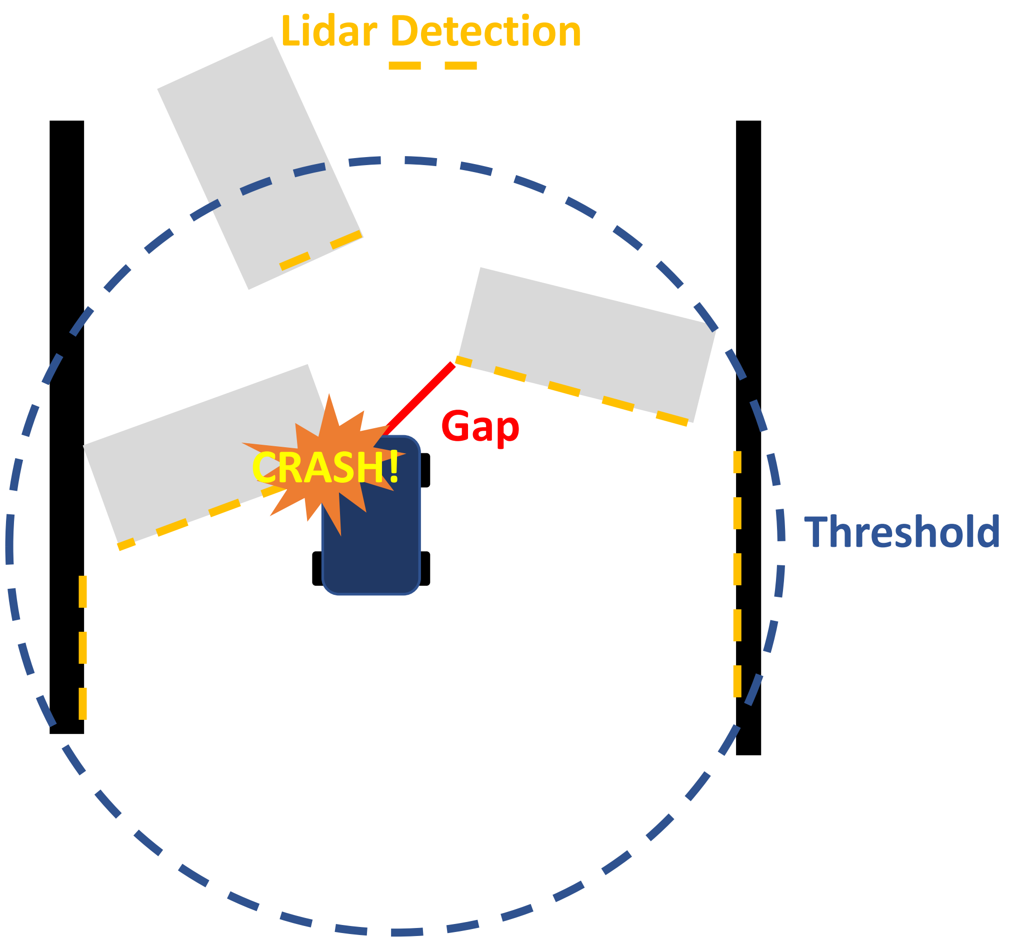


Follow the Gap

- The idea: “Seek out the **largest gap**”

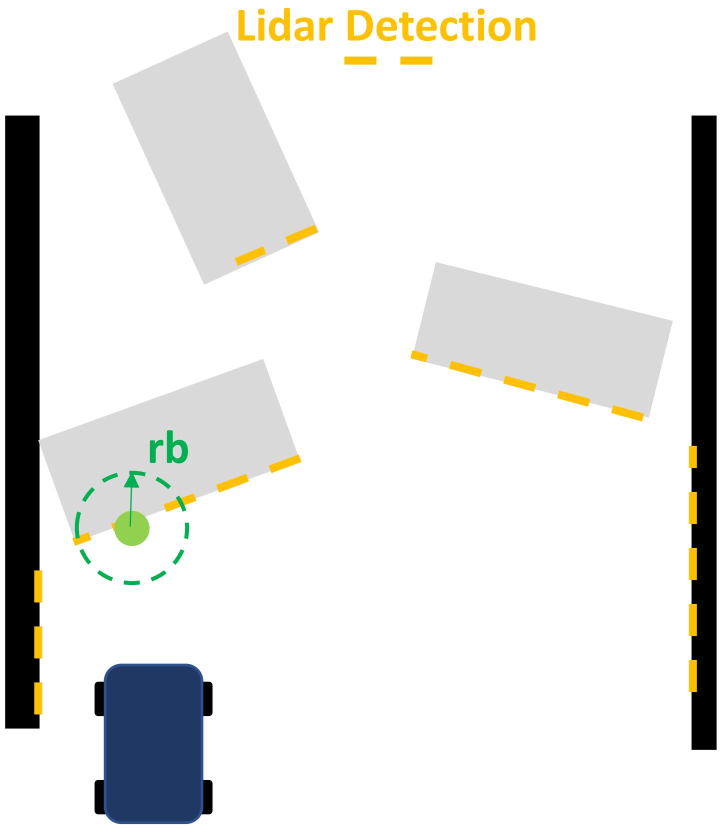
  - Works fine for holonomic robots(e.g. turtlebots)
  - Works fine for non-holonomic robots in environment with sparse obstacles

→ **Doesn’t optimize for safety**
→ **Doesn’t consider car’s dimensions**
→ **Hard to decide threshold t**



Follow the Gap: Better Method

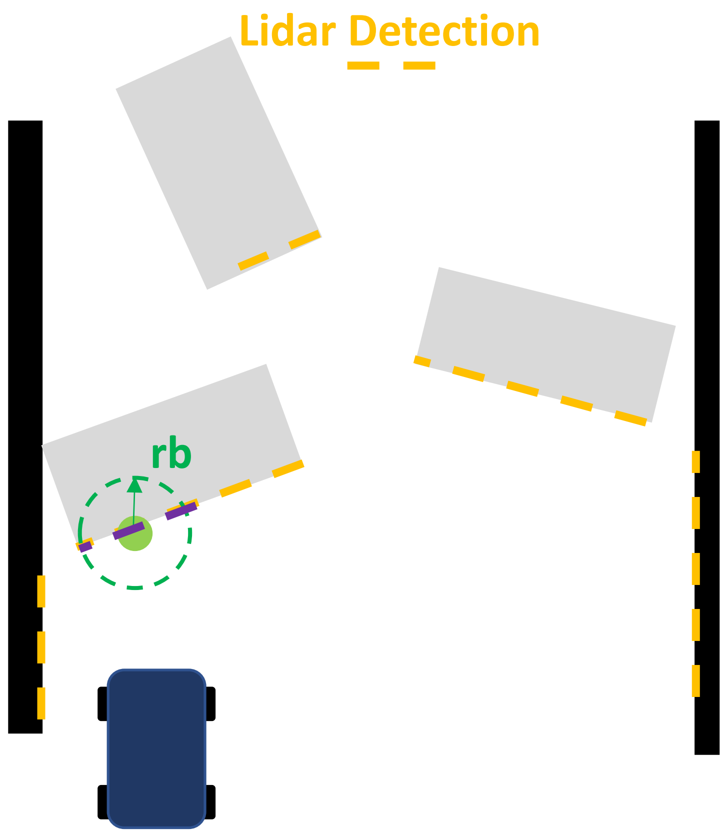
- Step 1
  - Find nearest LIDAR point and put a “safety bubble” around it of radius rb



[4.8, **3.1**, 4.3, 4.5, …, 10.1, 8.3]

Follow the Gap: Better Method

- Step 2
  - Set all points inside bubble to distance 0. All nonzero points are considered 'free space'

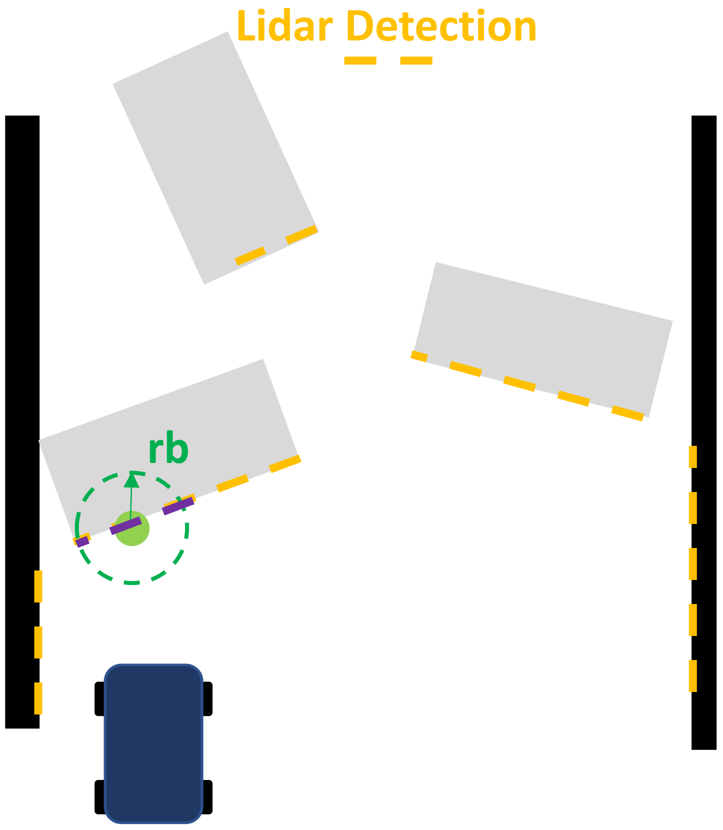


[4.8, **3.1**, 4.3, 4.5, ..., 10.1, 8.3]

[4.8, **0.0**, **0.0**, **0.0**, ..., 10.1, 8.3]

Follow the Gap: Better Method

- Step 3
  - Find maximum length sequence of consecutive non-zeros among the ‘free space’ points. (The **max gap**)

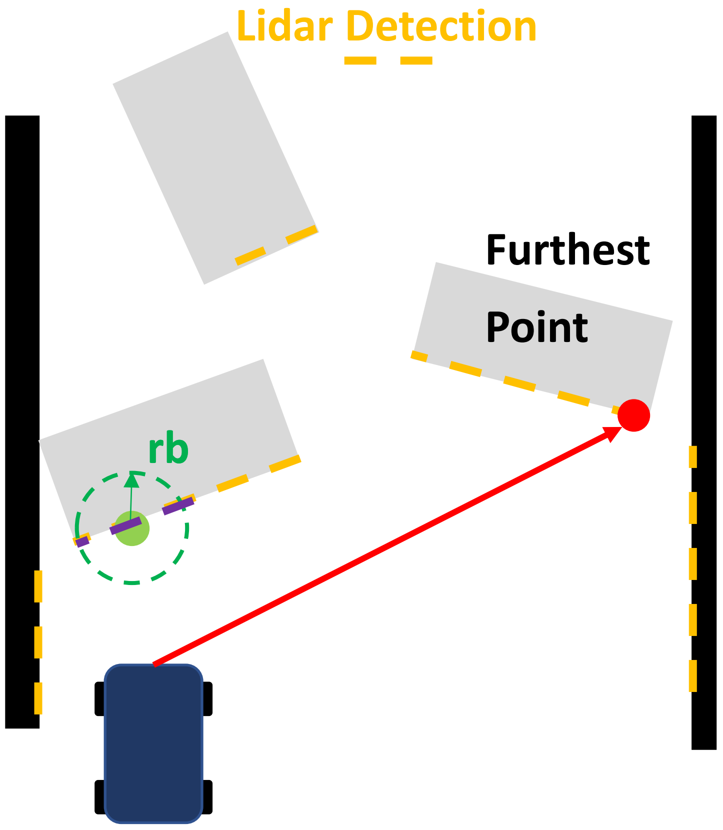


[4.8, **3.1**, 4.3, 4.5, …, 10.1, 8.3]

[4.8, **0.0**, **0.0**, **0.0**, …, 10.1, 8.3]

Follow the Gap: Better Method

- Step 4
  - Find the ‘best’ point among this maximum length sequence
  - **Naïve**: Choose the furthest point in free space, and set your steering angle towards it

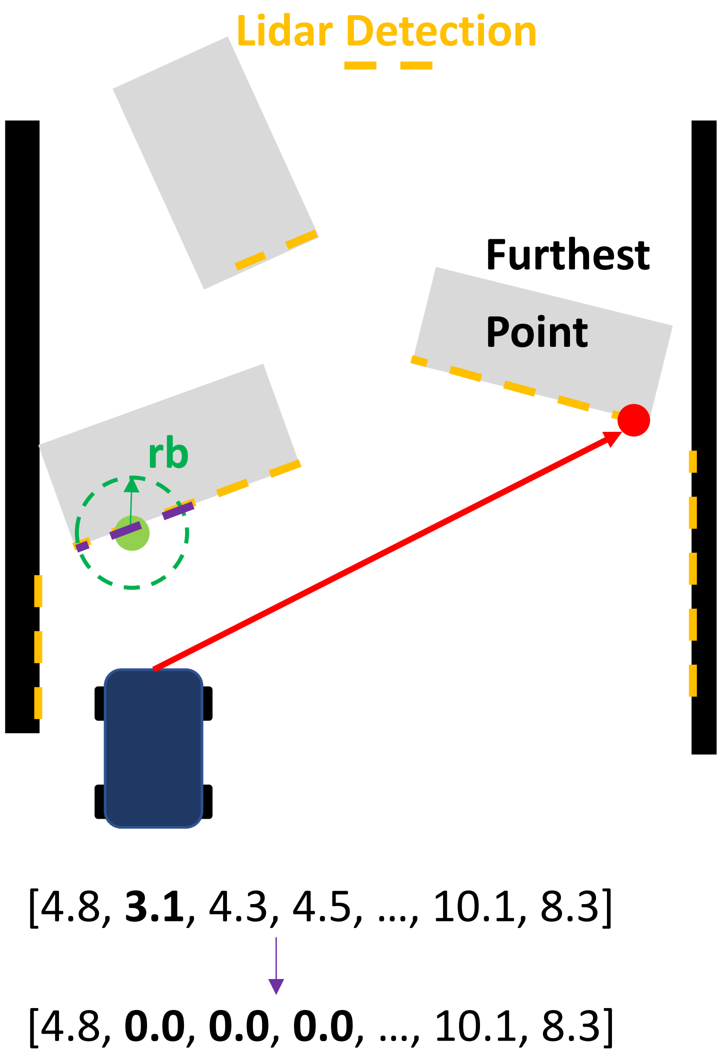


[4.8, **3.1**, 4.3, 4.5, …, 10.1, 8.3]

[4.8, **0.0**, **0.0**, **0.0**, …, 10.1, 8.3]

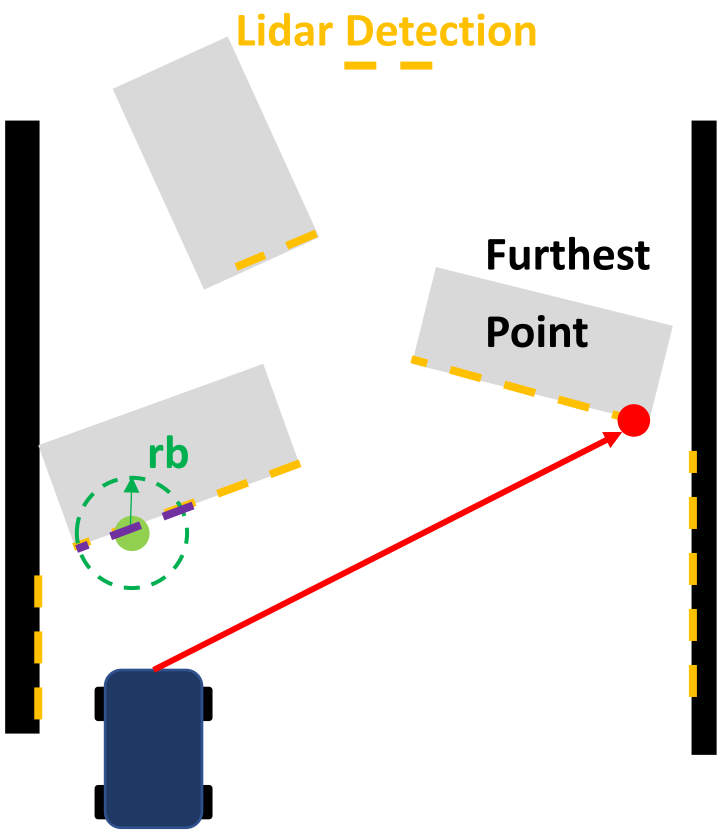
Follow the Gap: Better Method

- Step 4
  - Find the **'best' point** among this maximum length sequence
  - **Naïve**: Choose the furthest point in free space, and set your steering angle towards it
  - **Better Idea Intuition:** If you're 3-4m away from your closest obstacle, should you immediately make a sharp turn to avoid it?



Follow the Gap: Better Method

- Step 1
  - Find nearest LIDAR point and put a "safety bubble" around it of radius rb
- Step 2
  - Set all points inside bubble to distance 0. All nonzero points are considered 'free space'
- Step 3
  - Find maximum length sequence of consecutive non-zeros among the 'free space' points. (The **max gap**)
- Step 4
  - Choose the appropriate (furthest) point in free space, and set your steering angle towards it



Reactive Methods Demonstration

Wall Following Implementation

```
#!/usr/bin/env python
from __future__ import print_function
import sys
import math
import numpy as np

#ROS Imports
import rospy
from sensor_msgs.msg import LaserScan
from ackermann_msgs.msg import AckermannDriveStamped



#TO DO: Tune parameters
#PID CONTROL PARAMS
kp = 0.8
kd = 0.5
ki = 2
servo_offset = 0.0
prev_error = 0.0
error = 0.0
integral = 0.0

#WALL FOLLOW PARAMS
ANGLE_RANGE = 270 # Hokuyo 10LX has 270 degrees scan
DESIRED_DISTANCE_RIGHT = 1.0 # meters
DESIRED_DISTANCE_LEFT = 1.0
VELOCITY = 1 # meters per second
CAR_LENGTH = 0.50 # TfollowLeftraxxas Rally is 20 inches or 0.5 meters
```

Parameters for PID control

Parameters for wall following

Wall Following Implementation

```python
class WallFollow:
    """ Implement Wall Following on the car
    """
    def __init__(self):
        #Topics & Subs, Pubs
        self.lidar_sub = rospy.Subscriber("/scan", LaserScan, self.lidar_callback)#: Subscribe to LIDAR
        self.drive_pub = rospy.Publisher('/vesc/low_level/ackermann_cmd_mux/output', AckermannDriveStamped, queue_size=10)#: Publish to drive


    def getRange(self, data):
        # data: single message from topic /scan
        # angle: between -45 to 225 degrees, where 0 degrees is directly to the right
        # Outputs length in meters to object with angle in lidar scan field of view
        #make sure to take care of nans etc.
        #TODO: implement
        laser_ranges = data.ranges
        laser_ranges = laser_ranges[600:710]
        return 0.0


    def pid_control(self, error, velocity):

        #TODO: Use kp, ki & kd to implement a PID controller
        #       Example:
        #               drive_msg = AckermannDriveStamped()
        #               drive_msg.header.stamp = rospy.Time.now()
        #               drive_msg.header.frame_id = "laser"
        #               drive_msg.drive.speed = 0
        #               self.drive_pub.publish(drive_msg)

    def lidar_callback(self, data):

        #TODO:
        #       1. Get LiDAR message
        #       2. Calculate length to object with angle in lidar scan field of view
        #          and make sure to take care of nans etc.
        #       3. Based on length to object, callback 'pid_control'
```

Gap Following Implementation

- For this lab, you will make a '**Gap Following Node**' that should make the car maintain a constant distant from the wall while moving forward
- You will need to first subscribe to the scan topic and find the max length "gap" with the ***LaserScan*** messages
- We are sending ***AckermannDriveStamped*** messages with desired velocity and steering angle

```python
#!/usr/bin/env python
from __future__ import print_function
import sys
import math
import numpy as np

#ROS Imports
import rospy
from sensor_msgs.msg import LaserScan
from ackermann_msgs.msg import AckermannDriveStamped
```