Applying LIDAR for Autonomous Wall Following

Presented by RSS Team 13
Alazar Lemma, Benjamin Soria, Caden Moore,
Henry Heiberger, and Vittal Thirumalai
3/10/2023

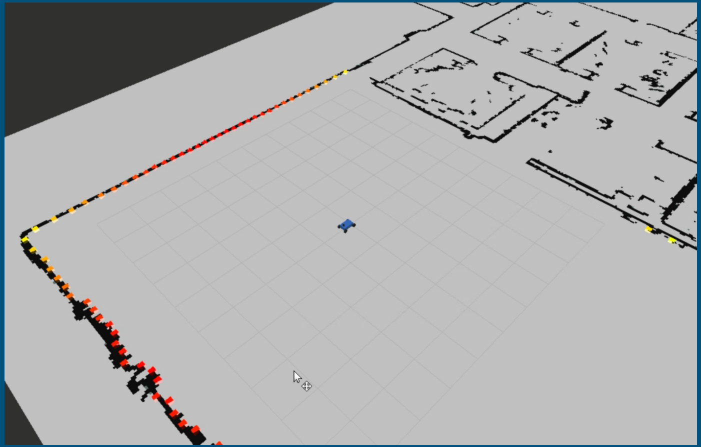
Team 23

Henry Heiberger   Alazar Lemma   Ben Soria   Caden Moore   Vittal Thirumalai

THE NEET TEAM!!!!!

Overview

The purpose of this lab is to implement a wall following algorithm tested in simulation onto a physical racecar.



Goals

Follow the wall

And

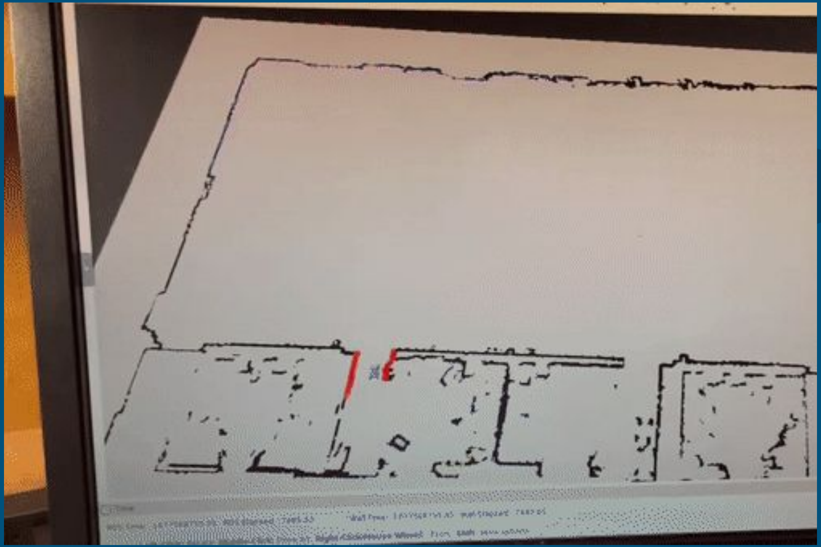
Don’t hit the wall!

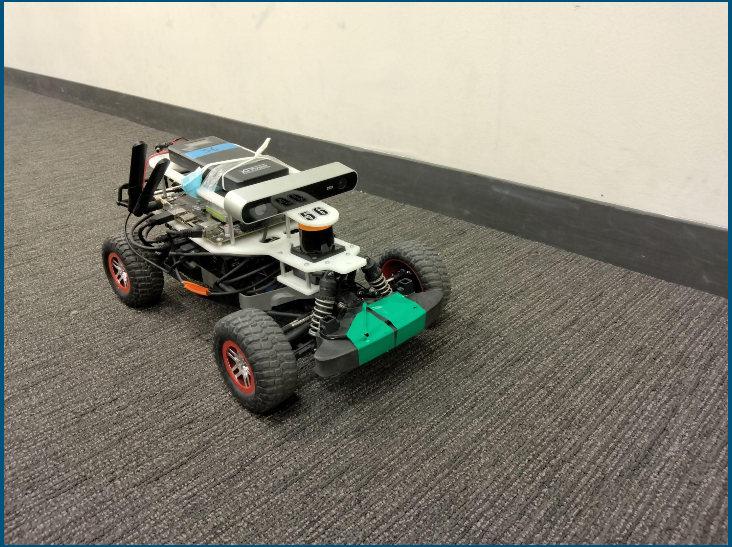
The main problem for Lab 3...

Going from this...

>>>>

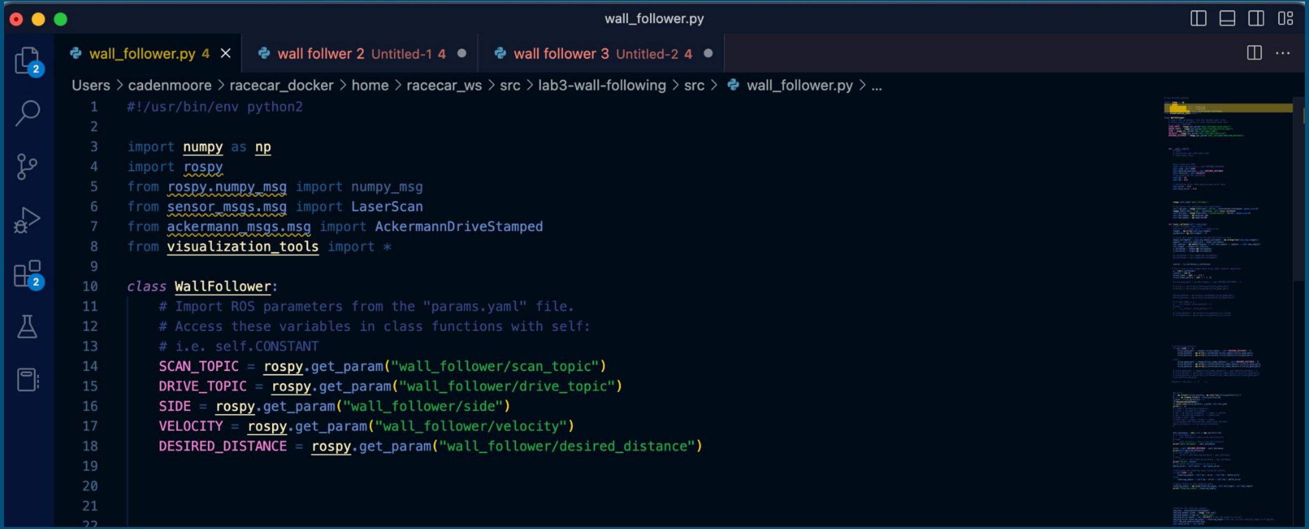
to this!





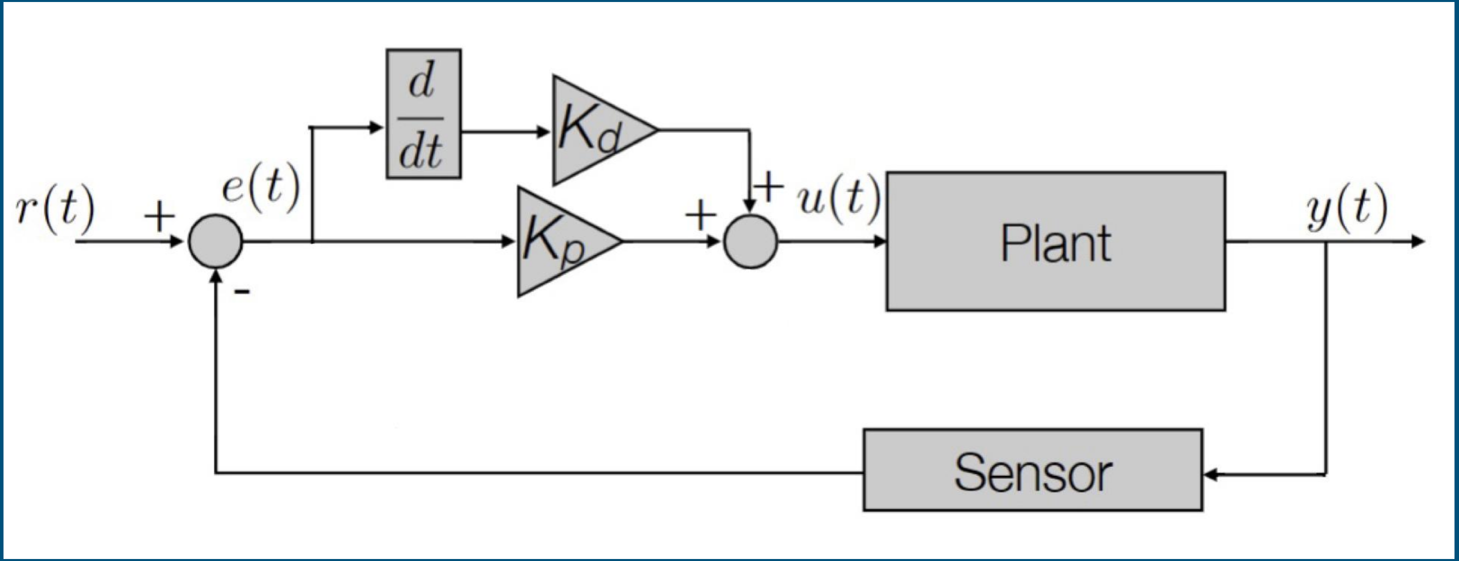
How we started

Find which parts of our codes worked best, then combine

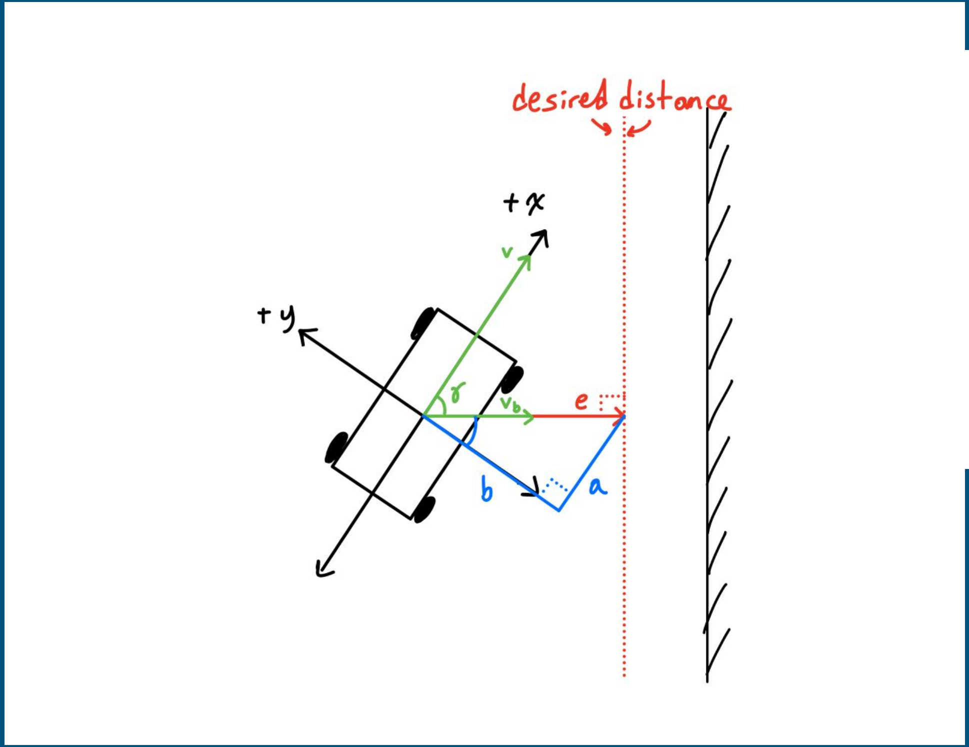

wall_follower.py
wall_follower.py 4
wall follwer 2 Untitled-1 4
wall follower 3 Untitled-2 4
Users > cadenmoore > racecar_docker > home > racecar_ws > src > lab3-wall-following > src > wall_follower.py > ...

```python
#!/usr/bin/env python2

import numpy as np
import rospy
from rospy.numpy_msg import numpy_msg
from sensor_msgs.msg import LaserScan
from ackermann_msgs.msg import AckermannDriveStamped
from visualization_tools import *

class WallFollower:
    # Import ROS parameters from the "params.yaml" file.
    # Access these variables in class functions with self:
    # i.e. self.CONSTANT
    SCAN_TOPIC = rospy.get_param("wall_follower/scan_topic")
    DRIVE_TOPIC = rospy.get_param("wall_follower/drive_topic")
    SIDE = rospy.get_param("wall_follower/side")
    VELOCITY = rospy.get_param("wall_follower/velocity")
    DESIRED_DISTANCE = rospy.get_param("wall_follower/desired_distance")
```

Implementing PID control



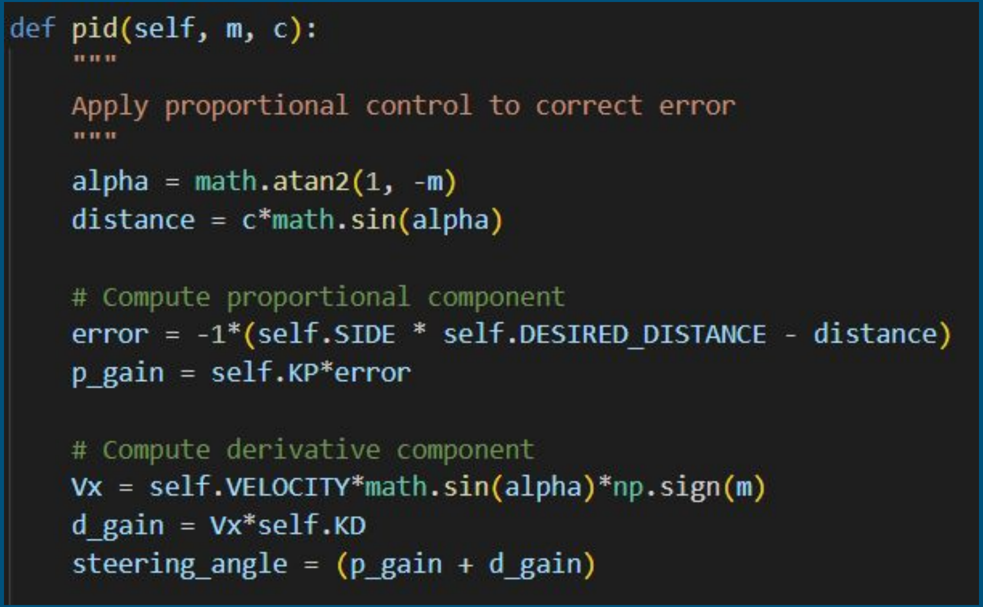
Implementing PID control



$$\frac{d}{dt}e = v_b = v\sin\alpha$$

$$\tan(\alpha) = \frac{\Delta x}{\Delta y} = \frac{1}{m}$$

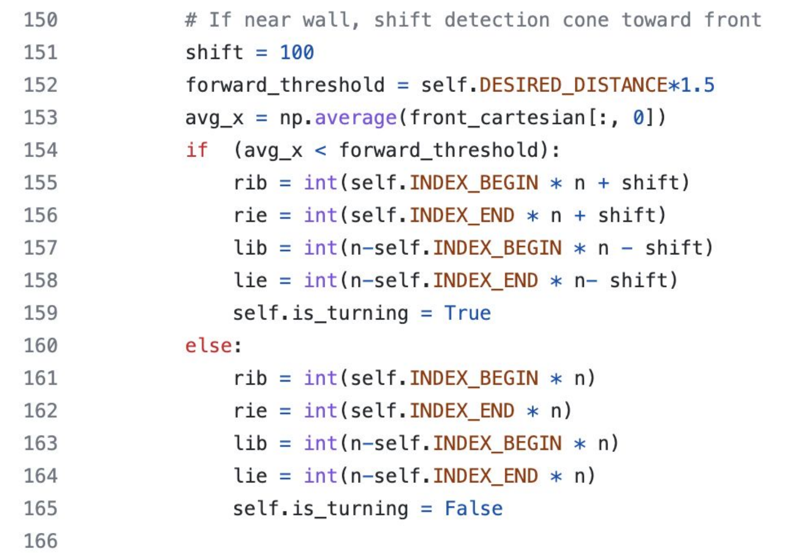
Implementing PID control



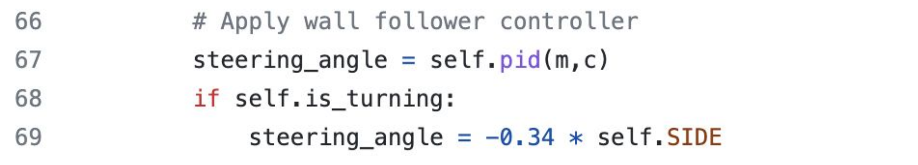
```python
def pid(self, m, c):
    """
    Apply proportional control to correct error
    """
    alpha = math.atan2(1, -m)
    distance = c*math.sin(alpha)

    # Compute proportional component
    error = -1*(self.SIDE * self.DESIRED_DISTANCE - distance)
    p_gain = self.KP*error

    # Compute derivative component
    Vx = self.VELOCITY*math.sin(alpha)*np.sign(m)
    d_gain = Vx*self.KD
    steering_angle = (p_gain + d_gain)
```

Corner Overriding

Override PID to max turn angle when close to a corner



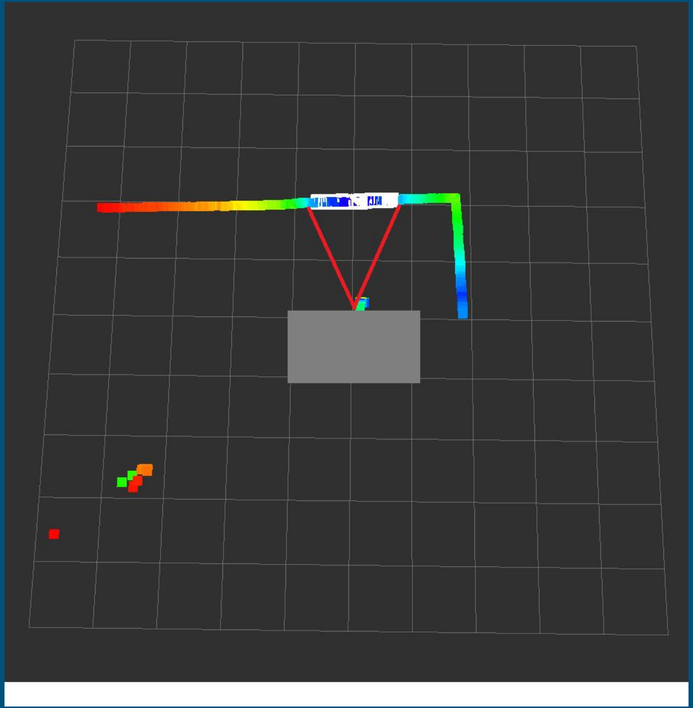
```python
# If near wall, shift detection cone toward front
shift = 100
forward_threshold = self.DESIRED_DISTANCE*1.5
avg_x = np.average(front_cartesian[:, 0])
if  (avg_x < forward_threshold):
    rib = int(self.INDEX_BEGIN * n + shift)
    rie = int(self.INDEX_END * n + shift)
    lib = int(n-self.INDEX_BEGIN * n - shift)
    lie = int(n-self.INDEX_END * n- shift)
    self.is_turning = True
else:
    rib = int(self.INDEX_BEGIN * n)
    rie = int(self.INDEX_END * n)
    lib = int(n-self.INDEX_BEGIN * n)
    lie = int(n-self.INDEX_END * n)
    self.is_turning = False

```

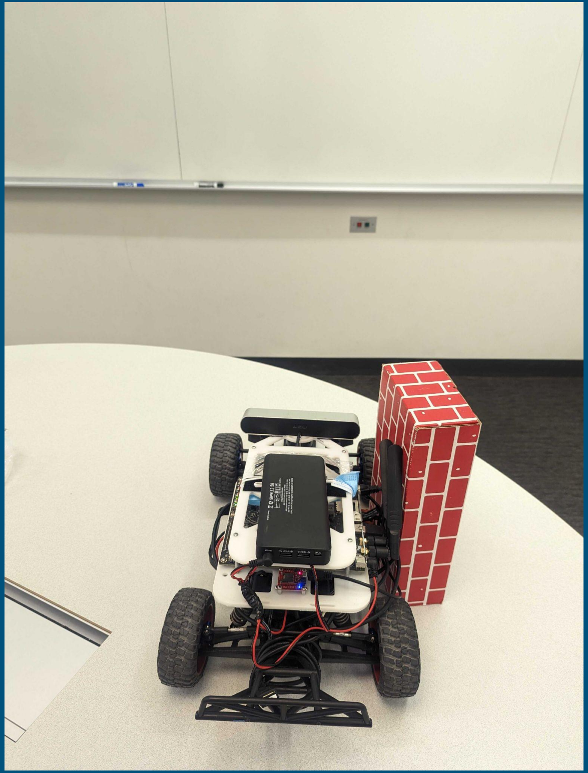


```python
# Apply wall follower controller
steering_angle = self.pid(m,c)
if self.is_turning:
    steering_angle = -0.34 * self.SIDE
```

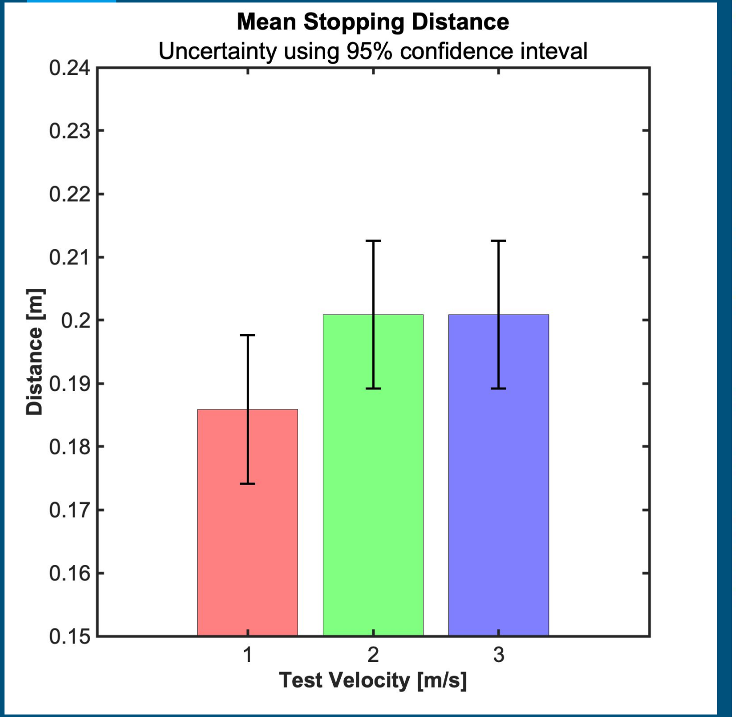
Wall Follower

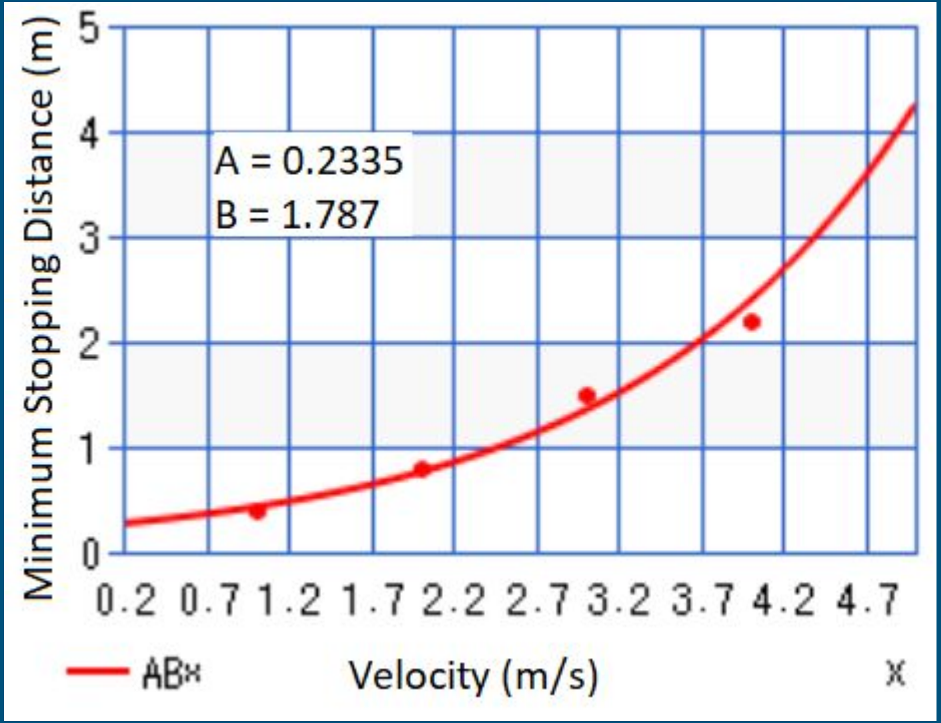
Implementing Safety Controller



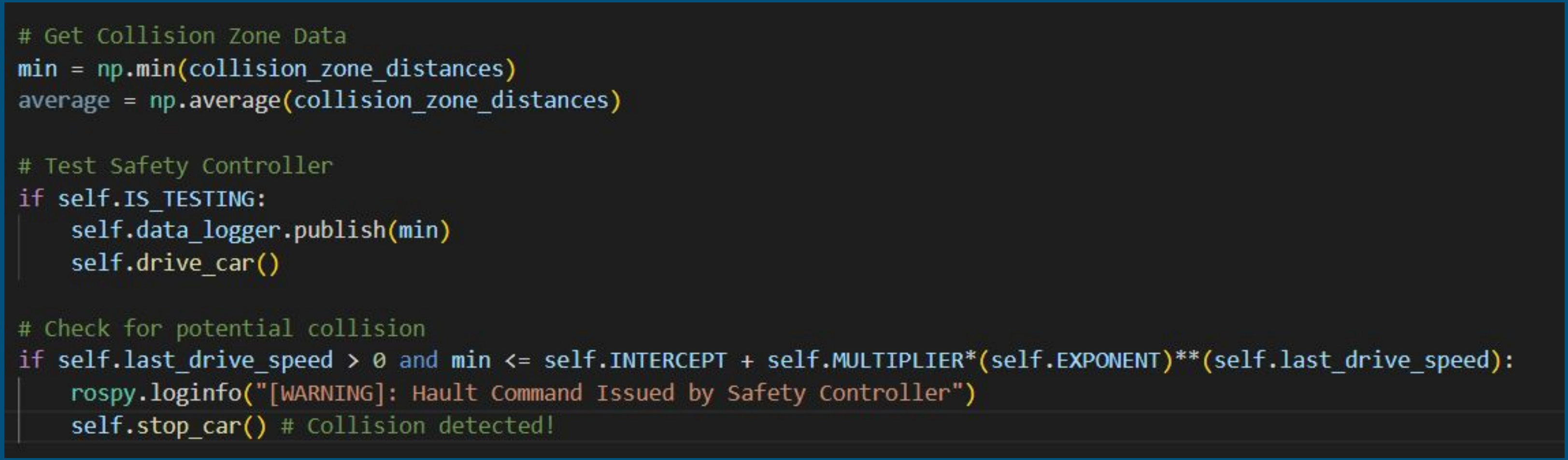


Implementing Safety Controller





Implementing Safety Controller



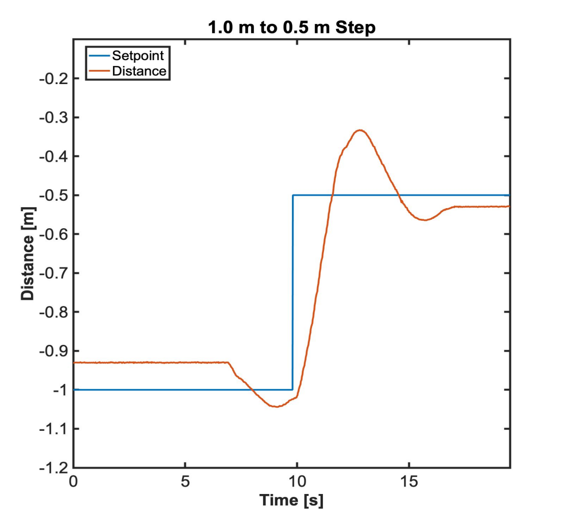
```
# Get Collision Zone Data
min = np.min(collision_zone_distances)
average = np.average(collision_zone_distances)

# Test Safety Controller
if self.IS_TESTING:
    self.data_logger.publish(min)
    self.drive_car()

# Check for potential collision
if self.last_drive_speed > 0 and min <= self.INTERCEPT + self.MULTIPLIER*(self.EXPONENT)**(self.last_drive_speed):
    rospy.loginfo("[WARNING]: Hault Command Issued by Safety Controller")
    self.stop_car() # Collision detected!
```

Safety Controller

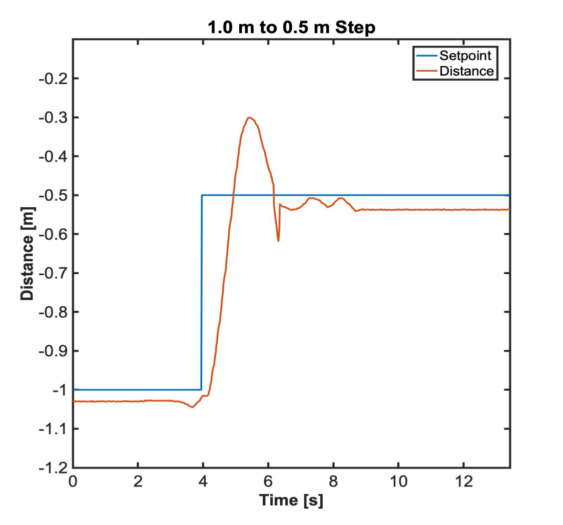
Experimental Evaluation

- Experimental tuning for slice of laserscan data used and PID parameters
- Safety controller: 10 tests each of speeds 1 - 3
- Autonomous wall follower: Rosbag analysis

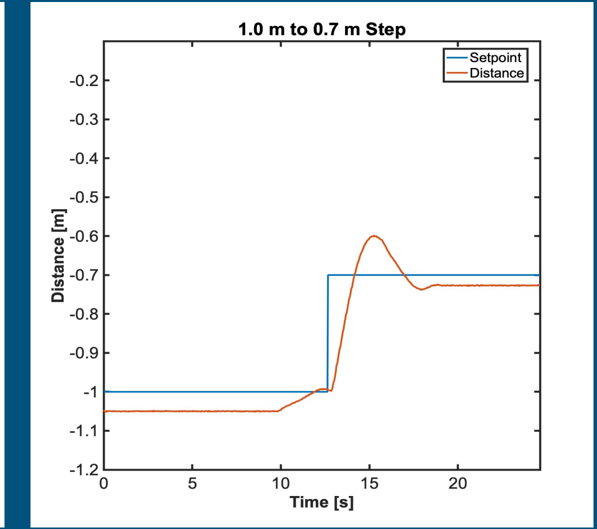
Results at 0.5 m/s



1.0 m to 0.5 m Step

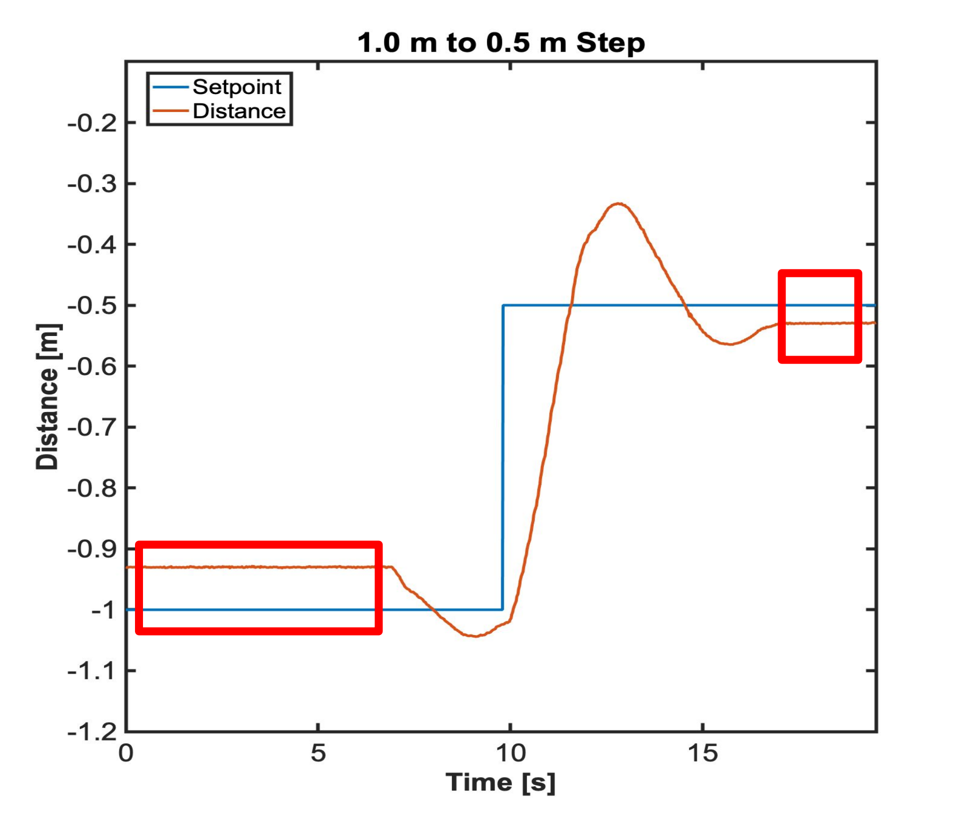


1.0 m to 0.5 m Step



1.0 m to 0.7 m Step

Issue?: Lingering steady-state error exists



Lessons Learned

- Translating control theory to a real-world implementation raises many issues
- How ROS is used on a physical robot, better understanding of middleware
- Comfort in working in a Linux environment
- Physical sensors have measurement errors that must be accounted for
- Using the IEEE style guidelines in a technical paper

Summary

- We achieved a high performing, accurate wall follower with built-in safety features
- Many lessons were learned and bugs fixed during the process
- Good teamwork overall with delegation and collaboration
- Flexible and portable for use in future labs that build on wall following

Questions?