

FITENTH Autonomous Racing

Reactive Methods for Navigation

Object Avoidance with Follow-the-Gap



Rahul Mangharam

University of Pennsylvania

rahulm@seas.upenn.edu



Safe Autonomy Lab
University of Pennsylvania



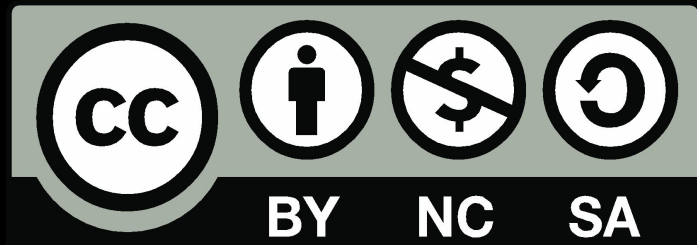
Penn
Engineering

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, Xiyuan Zhu and many others.

We are grateful for learning from each other



Except where otherwise noted, this work is licensed under

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

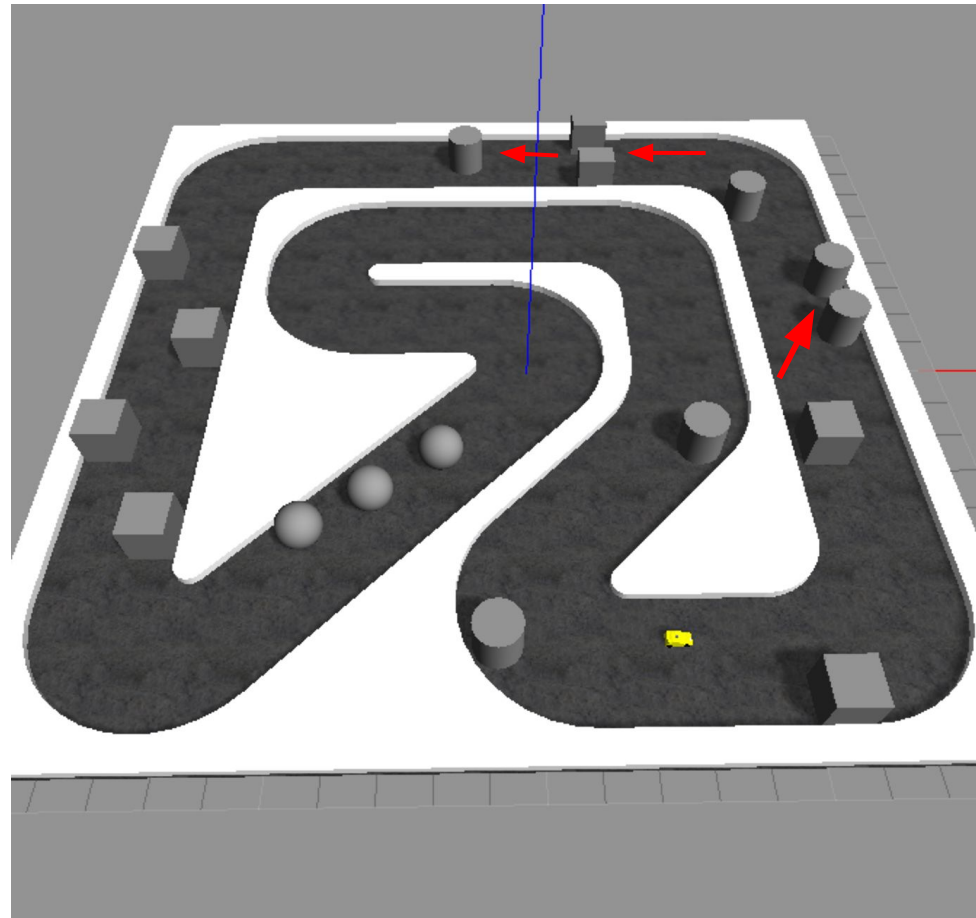
Obstacle Avoidance without a Map

Can we use Wall Following to navigate this race track?

If we build a map we can plan an obstacle free path to follow.

Challenge:

Locally avoid obstacles (reactive approach) without any map.



Obstacle Avoidance: Follow the Gap

Challenge:

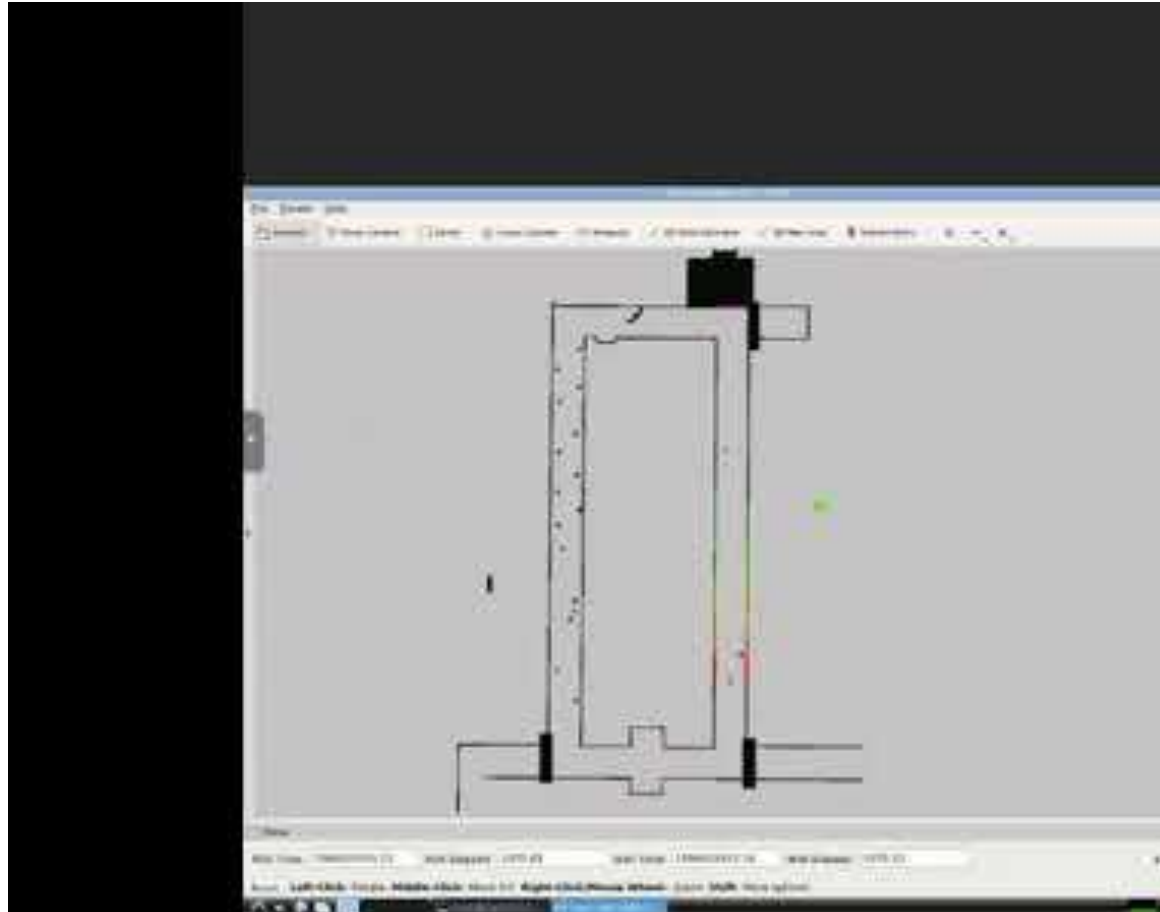
How can we avoid obstacles

Learning Outcome:

Basics of reactive navigation, avoidance on both static and dynamic obstacles

Assignment:

Follow the gap in simulation and on the vehicle.



Obstacle Avoidance: Follow the Gap

Challenge:

How can we avoid obstacles

Learning Outcome:

Basics of reactive navigation, avoidance on both static and dynamic obstacles

Assignment:

Follow the gap in simulation and on the vehicle.



Race I: Reactive Methods

Race Format:

Time attack, single car

Penalties:

Crashing

Baseline:

Complete 5 laps without crashing

Example Video:

Follow the Gap in 2017 FITenth
Grand Prix

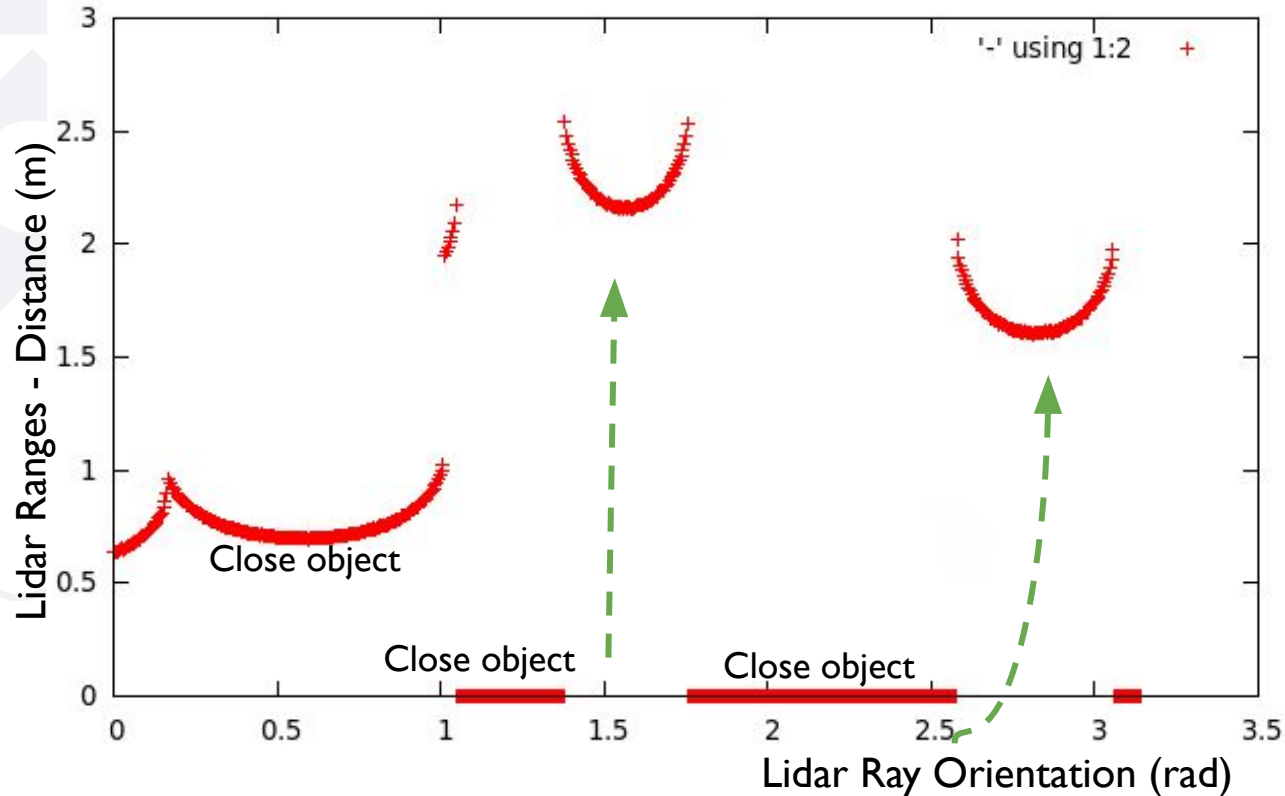


How do we get to that level of performance without a map?

Lecture Outline

1. Follow the Gap
2. Disparity Extender
3. Other Reactive Approaches
4. ROS Lab

Where should the car go?



Follow the Gap: Which gap?

[0.5, 5.1, 6.0, 7.0, inf, 3.0, inf, 3.0, inf, 8.0, 1.0, 3.0]

Where should
the car go?



Follow the Gap: Which gap?

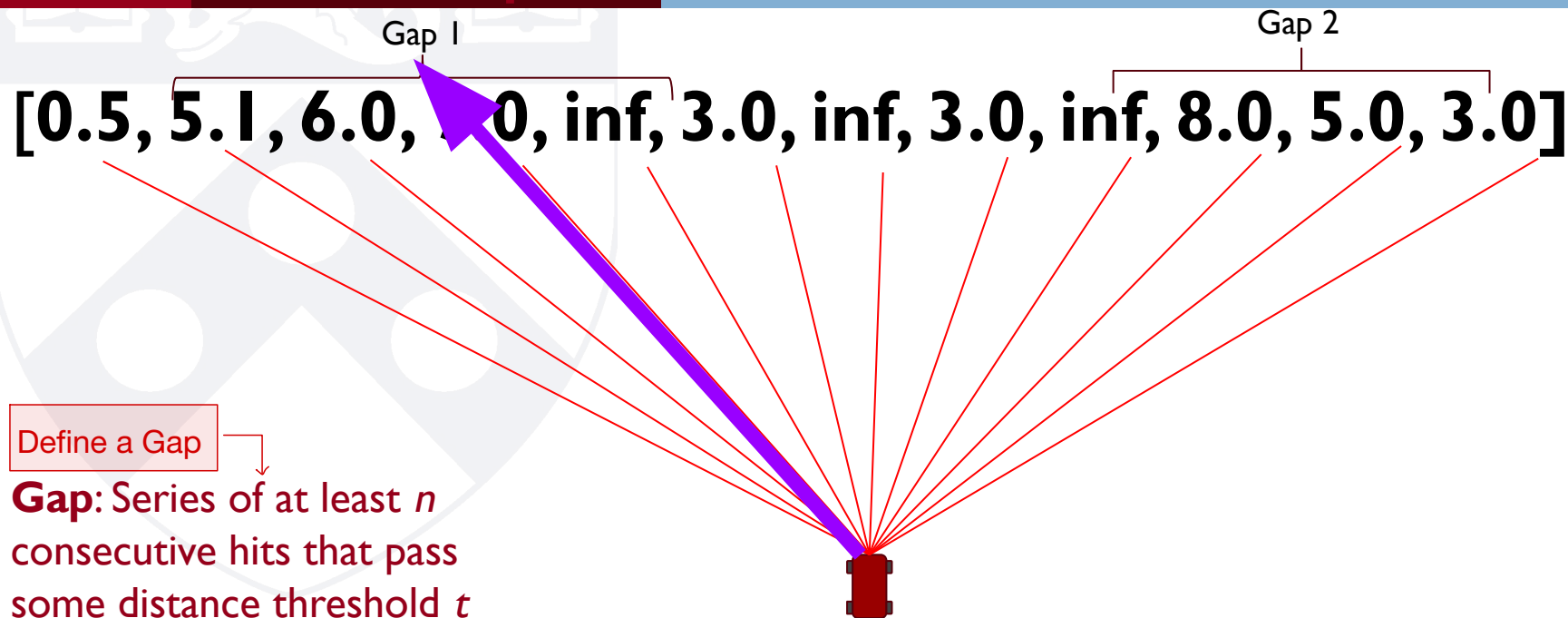
[0.5, 5.1, 6.0, 7.0, inf, 3.0, inf, 3.0, inf, 8.0, 1.0, 3.0]

Furthest distance?
Why might this be wrong?

The ray passes through but
the car won't



Follow the Gap



Define a Gap

Gap: Series of at least n consecutive hits that pass some distance threshold t
 $n = 3, t = 5.0$

Follow the Gap

Gap 1

Gap 2

[0.5, 5.1, 6.0, 4.0, inf, 3.0, inf, 3.0, inf, 8.0, 5.0, 3.0]

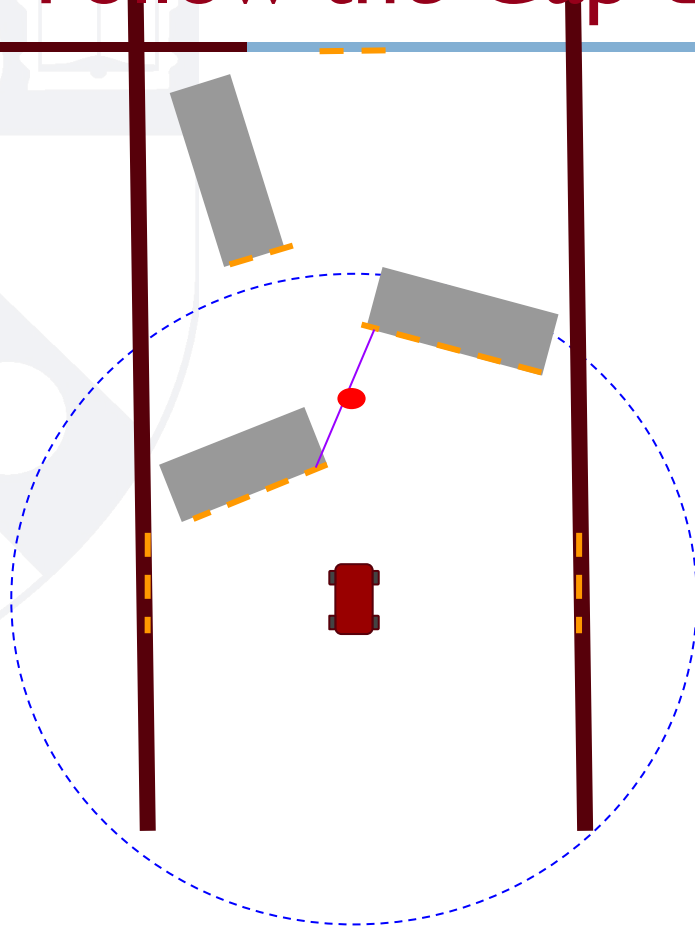


Gap: Series of at least n consecutive hits that pass some distance threshold t
 $n = 3, t = 5.0$

1. Find the gaps
2. Calculate the width of each gap
3. Determine the widest gap
4. Optional: Determine the “deepest” gap

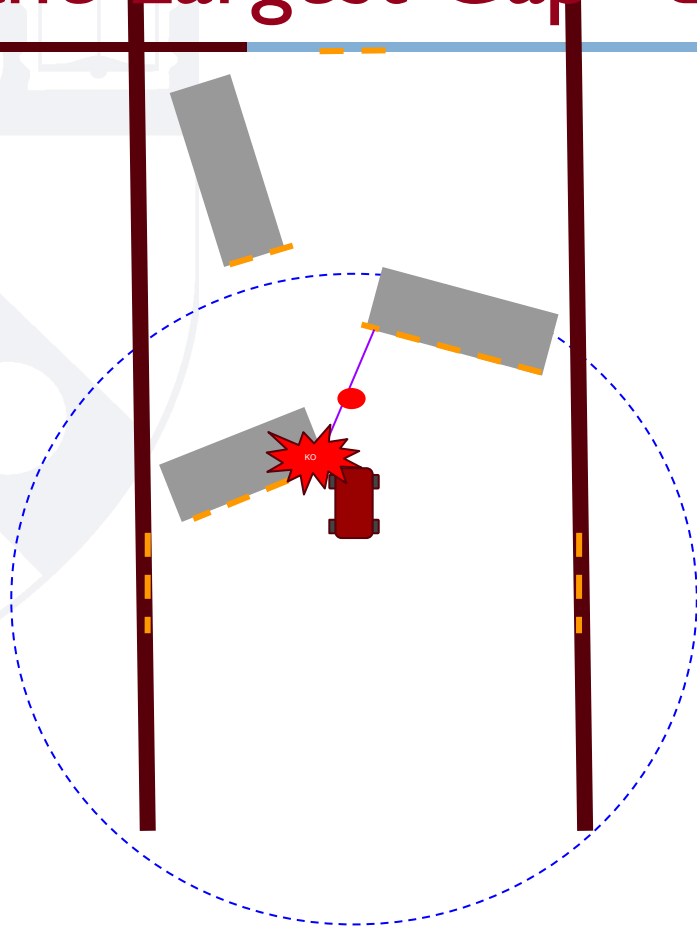
Why 'Naive' Follow the Gap doesn't work

Distance
threshold



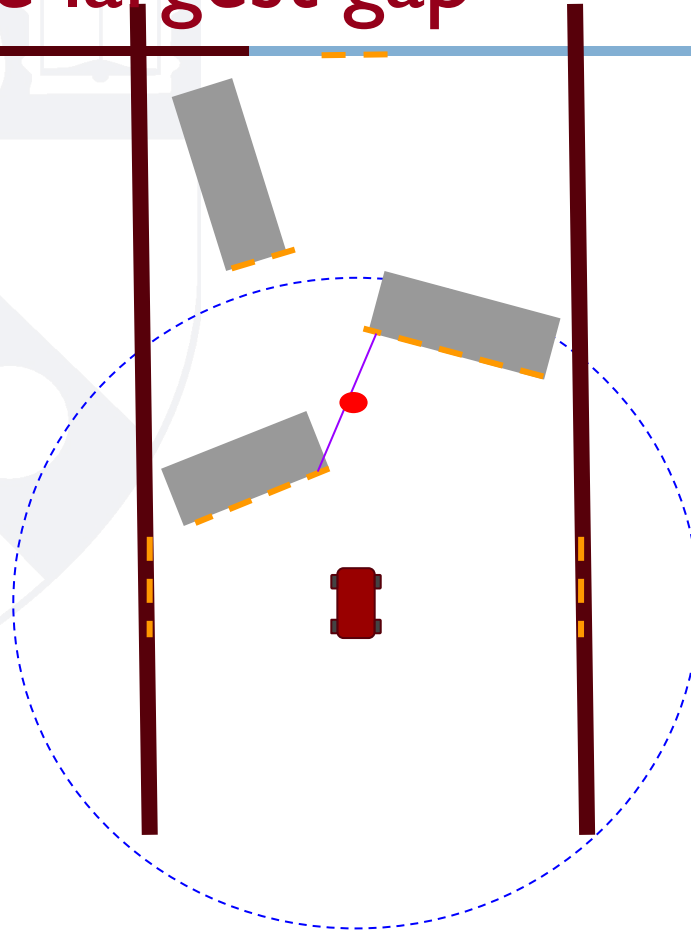
Why “Find the Largest Gap” doesn’t work

Threshold



Seek out the largest gap

Threshold



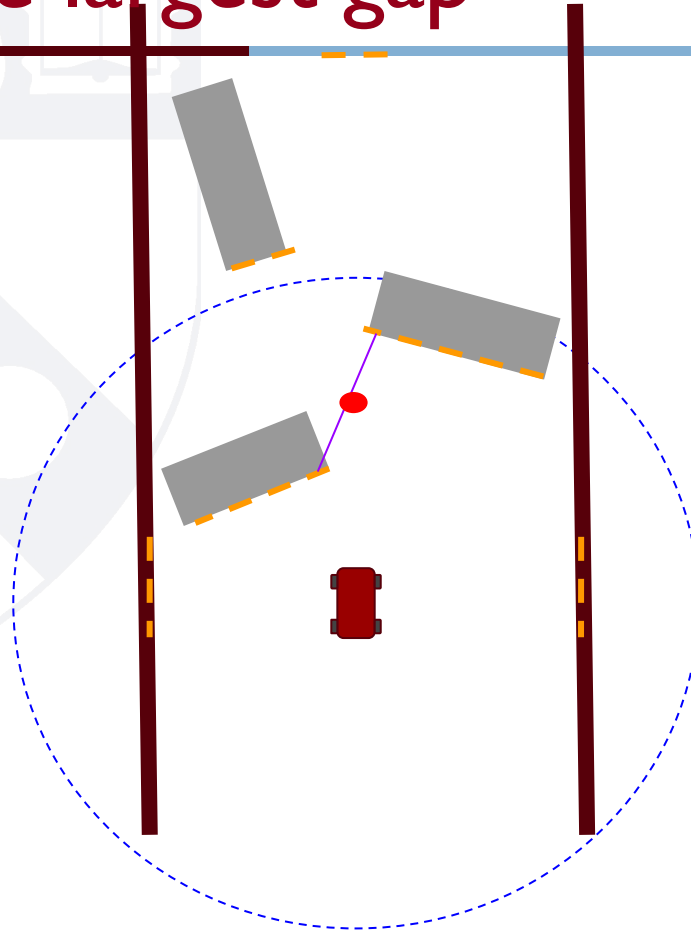
Works fine for holonomic robots (eg. turtlebots)

Works fine for non-holonomic robots in environments with sparse obstacles

Robots have to reposition them self and go to the certain point for non-holonomic

Seek out the largest gap

Threshold



Works fine for holonomic robots (eg. turtlebots)

Works fine for non-holonomic robots in environments with sparse obstacles

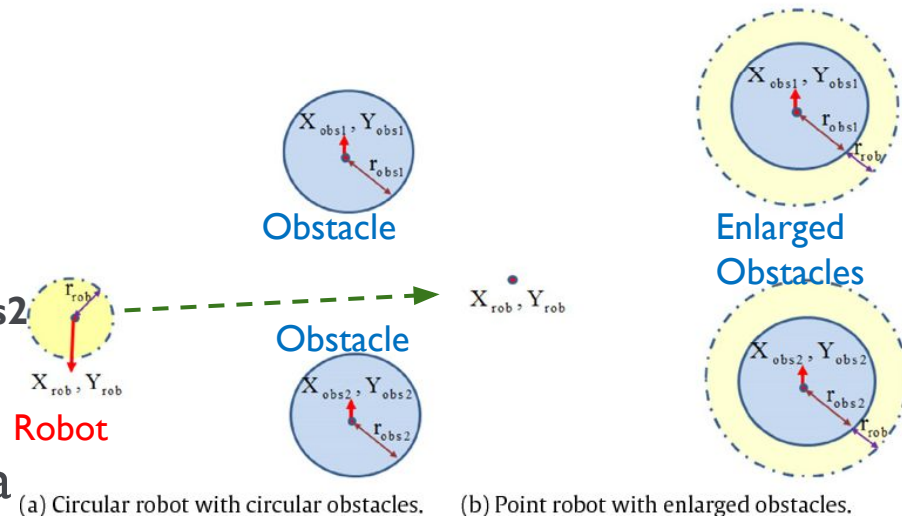
Doesn't account for safety

Doesn't consider car's dimensions

Hard to decide threshold t

Incorporating Affordances in Navigation

- Represent all bots and obstacles as circles
- Our robot has radius r_{rob}
- Obstacles have radius r_{obs1} and r_{obs2}
- Inflate objects by r_{rob}
- Robot can now be represented as a point without any size



FTG Tweak 1:

At every timestep, cleverly avoid the nearest obstacle

Step 1

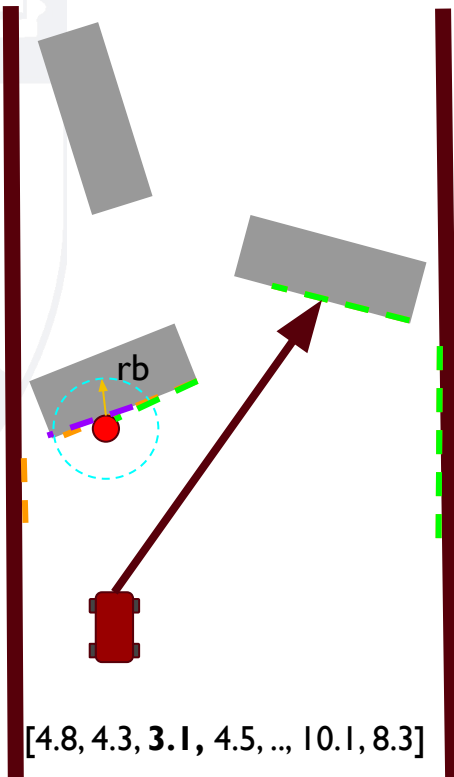
Find nearest LIDAR point and put a “safety bubble” around it of radius rb

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

Find the ‘best’ point among this maximum length sequence

Naive: Choose the furthest point in free space, and set your steering angle towards it

FTG Tweak 1:

At every timestep, cleverly avoid the nearest obstacle

Step 1

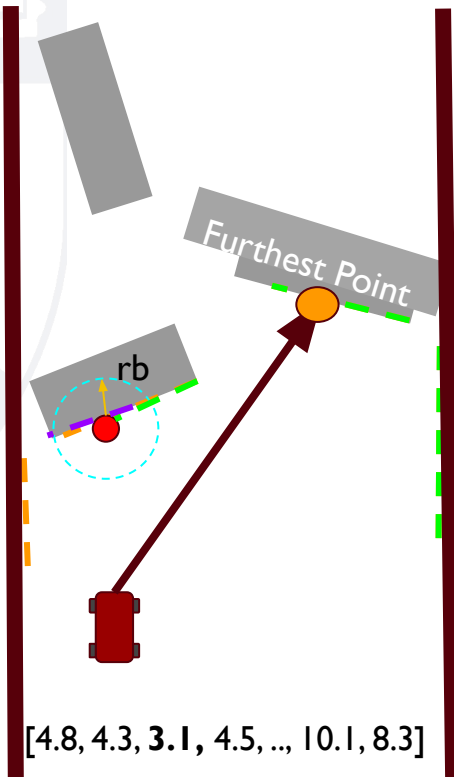
Find nearest LIDAR point and put a “safety bubble” around it of radius rb

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**



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

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

Step 4

Find the ‘best’ point among this maximum length sequence

Naive: Choose the furthest point in free space, and set your steering angle towards it

At every timestep, cleverly avoid the nearest obstacle

Step 1

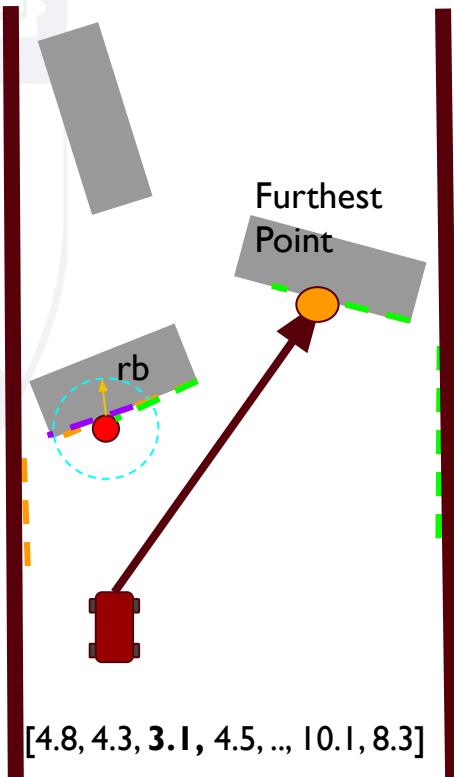
Find nearest LIDAR point and put a “safety bubble” around it of radius rb

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

Find the ‘best’ point among this maximum length sequence

Naive: Choose the furthest point in free space, and set your steering angle towards it

Changing speed results in you losing velocity

Rapid changing in direction will cause losing a great amount of speed which will result in losing the performance

At every timestep, cleverly avoid the nearest obstacle

Step 1

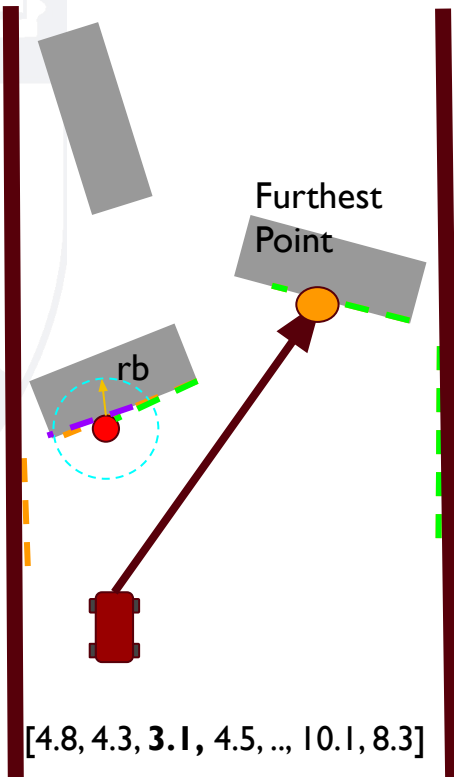
Find nearest LIDAR point and put a “safety bubble” around it of radius rb

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**



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

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

Step 4

Find the ‘best’ point among this maximum length sequence

Naive: Choose the furthest point in free space, and set your steering angle towards it

Changing speed results in you losing velocity

Better Idea Intuition

If you’re 3-4m away from your closest obstacle, should you immediately make a sharp turn to avoid it?

FTG Tweak 1:

At every timestep, cleverly avoid the nearest obstacle

Step 1

Find nearest LIDAR point and put a “safety bubble” around it of radius r

Step 2

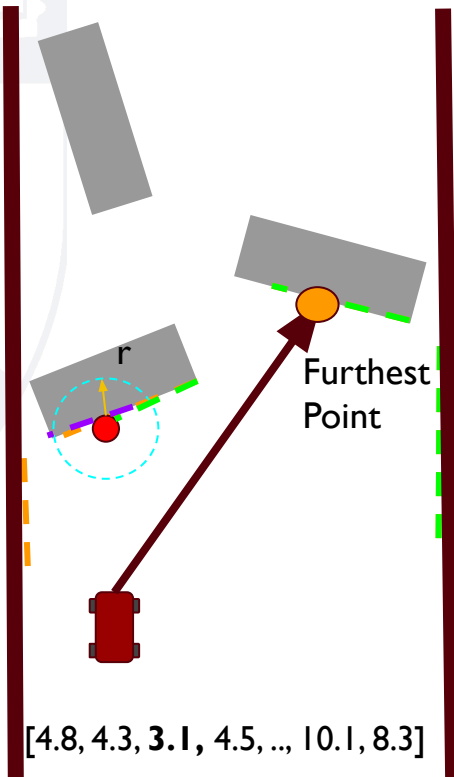
Set all points inside bubble to distance 0. All nonzero points are considered ‘free space’

Step 3

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

Step 4

Choose the furthest point in free space, and set your steering angle towards it



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

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

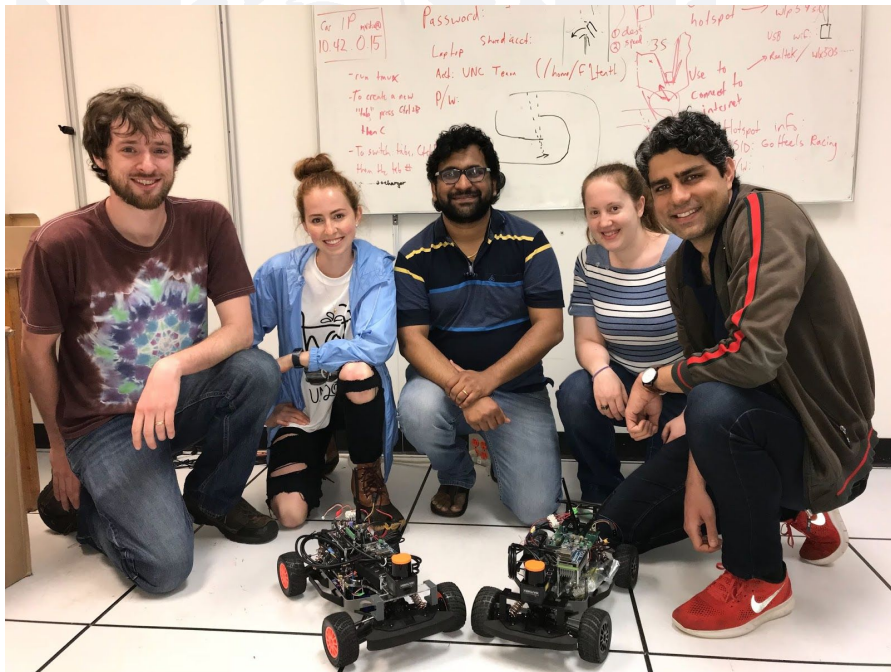
The drive is still slow and wobbly

This is because of the changing in the direction frequently



FTG Tweak 2:

Use the Disparity Extender approach

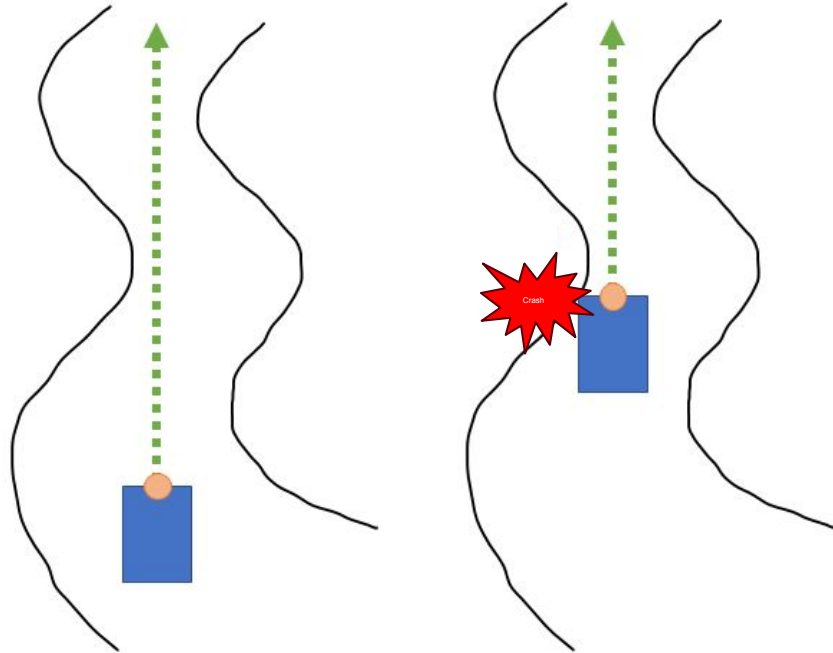


This tweak was proposed by the UNC Team: Nathan Otterness, Charlotte, Prof. Sridhar Duggirala, Tanya, Abel
Winner of 2019 FITenth Autonomous Racing Grand Prix in Montreal, Canada

Use the Disparity Extender approach

At every instant:

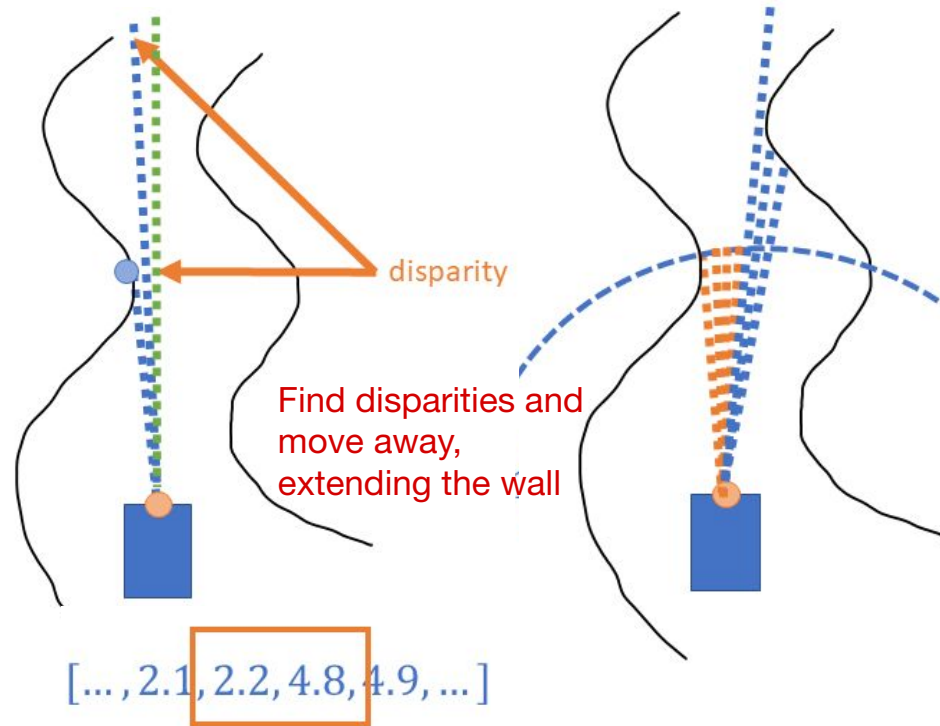
1. Head towards the farthest distance, not behind the car, that the car can safely reach by driving in a straight line.
2. Go in that direction.



Use the Disparity Extender approach

Head towards the farthest distance,
extending disparities

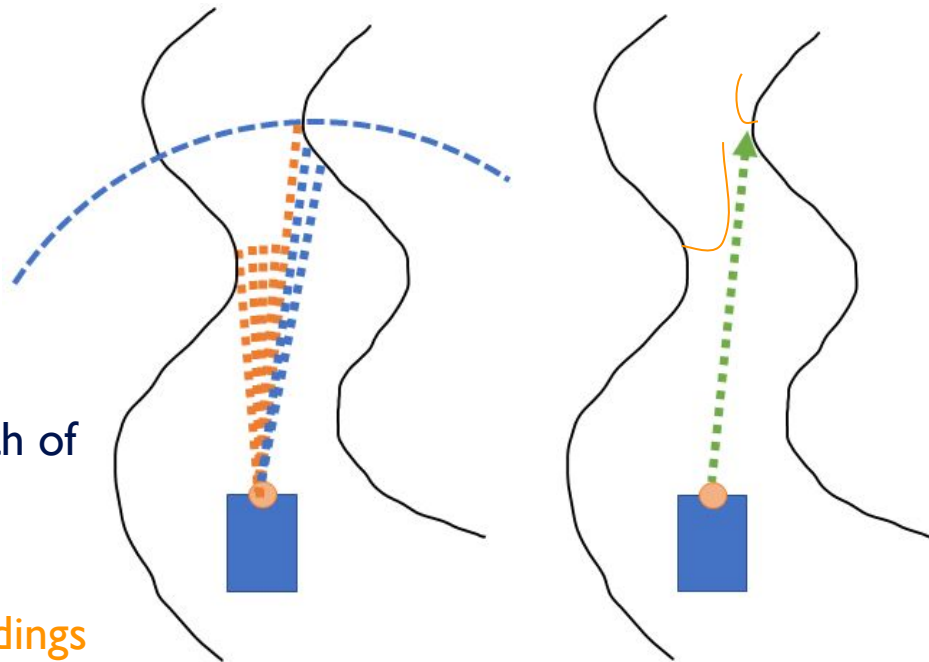
1. Find **disparities** in the lidar readings



Use the Disparity Extender approach

Head towards the farthest distance,
extending disparities

1. Find **disparities** in the lidar readings
2. For each disparity, **extend** it half the width of the car
3. Choose a path based on **virtual lidar readings**



FTG Tweak 2:

Use the **Disparity Extender** approach

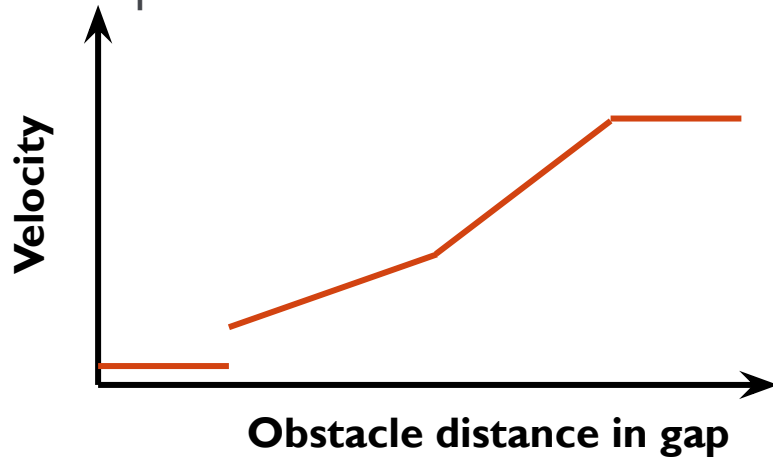


Use the Disparity Extender approach

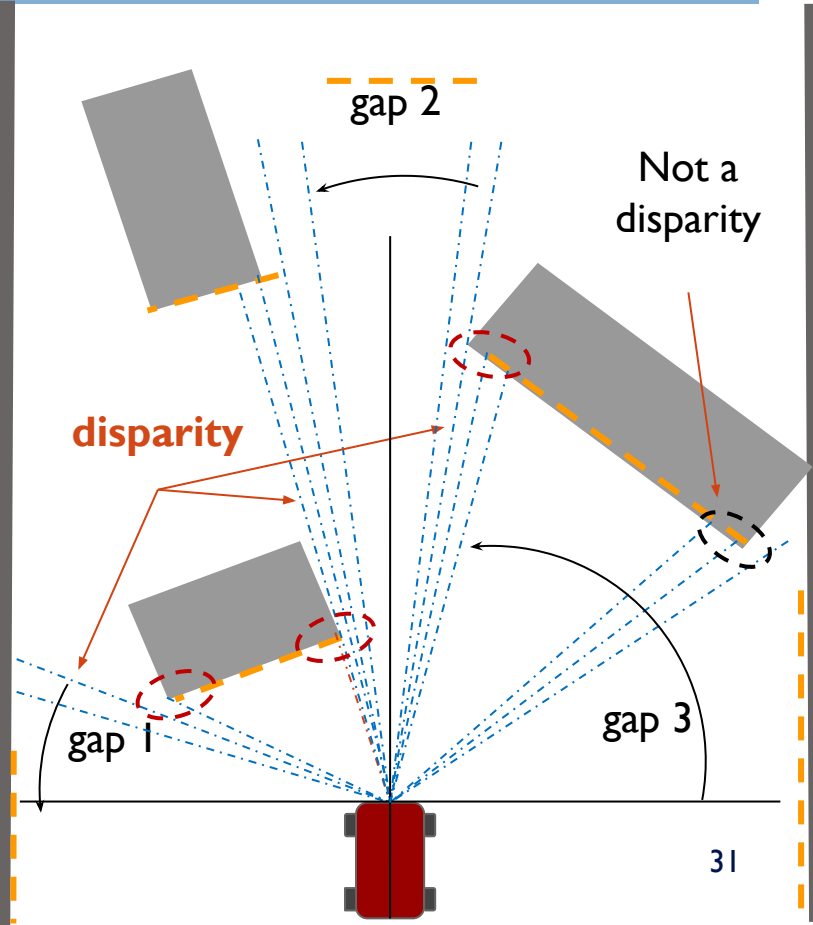


Use the Disparity Extender approach

- This is a greedy approach. An optimal reactive approach could use dynamic programming
- Choose speed based on distance

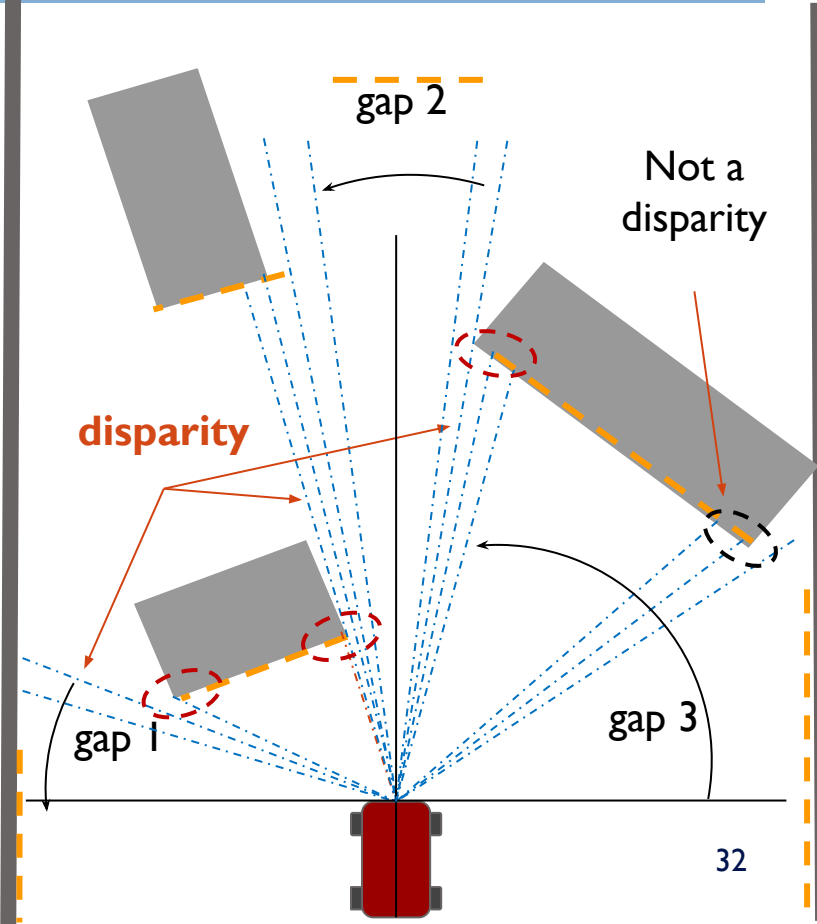


- Process each lidar reading quickly - update direction/speed at each step
- 40Hz Lidar update \rightarrow 25ms. So keep your processing $< 10\text{ms}$



Use the Disparity Extender approach

1. Set a threshold to detect disparities in Lidar scan ranges
2. Set a bubble at each disparity and all points in the bubble are set to zero. All non-zero points are considered as gaps;
3. Decide which gap to follow and at what speed
4. Use piecewise linear speed settings



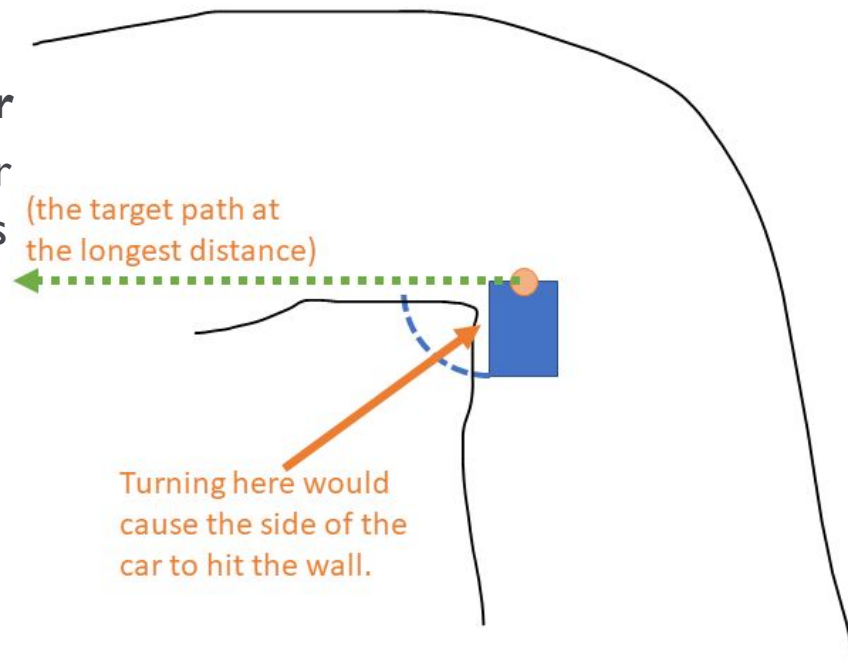




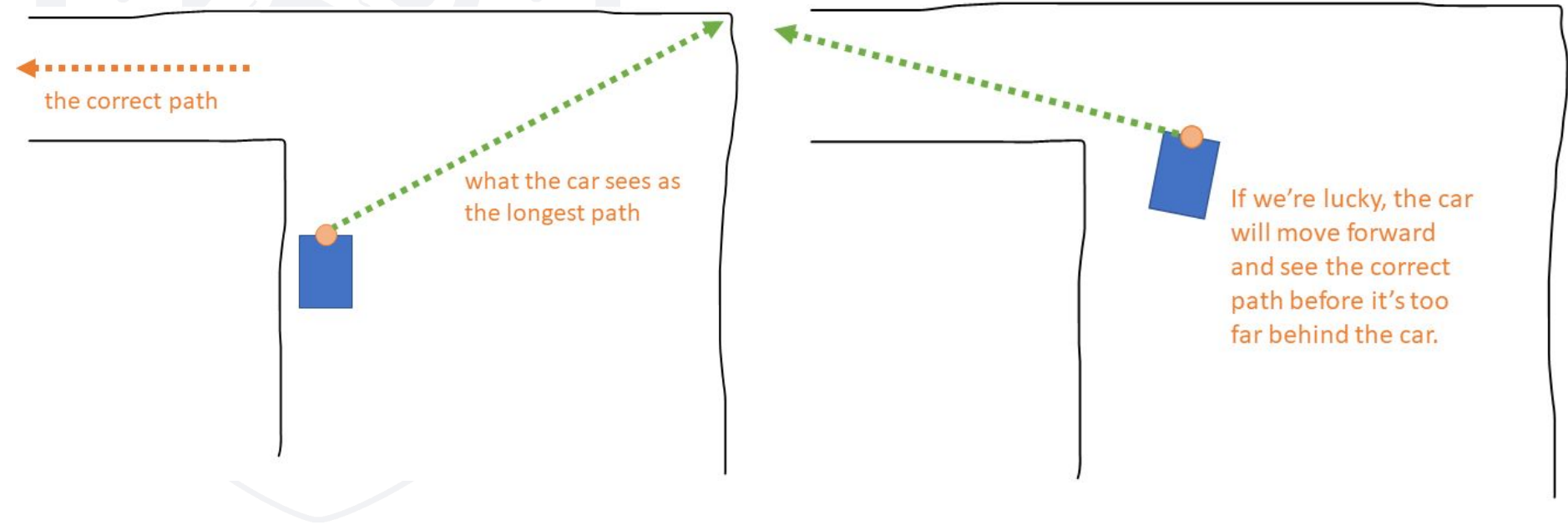
Disparity Extender - Cornering

To make sure the car doesn't strike a corner

1. Scan all available LIDAR samples below -90 or above $+90$ degrees (these will cover the sides of the car to the back).
2. If any point is below a safe distance on the side of the car in the direction the car is turning, stop turning and just keep going straight

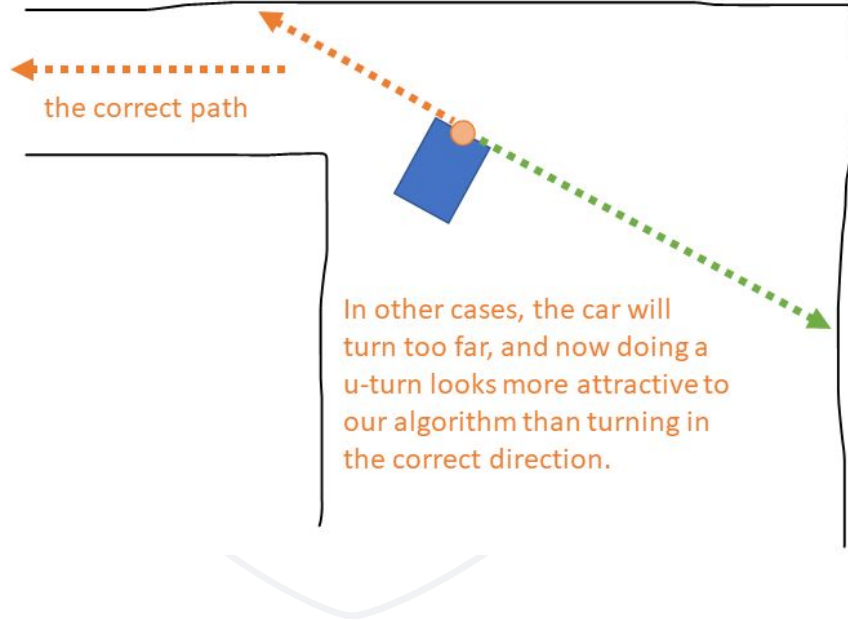


Disparity Extender - Where it fails



FTG Tweak 2:

Use the Disparity Extender approach



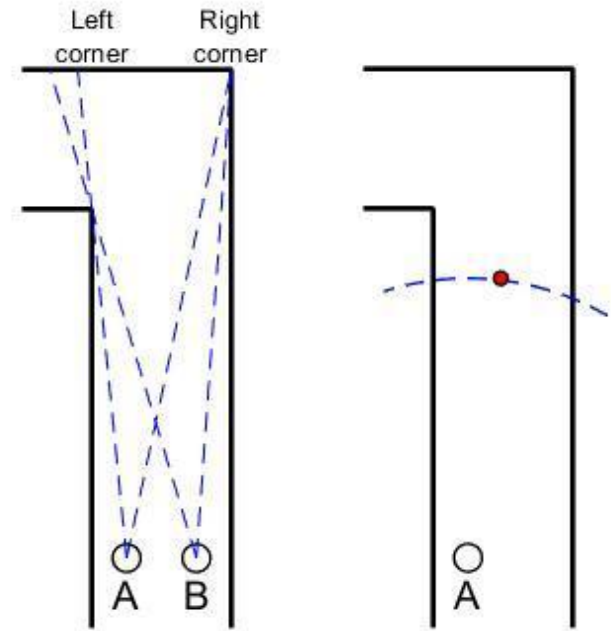
FTG Tweak 4: Minimize the 'wiggling problem'

Set a max distance threshold and drive at max speed

The wiggling problem: the robot keeps turning left and right, running in a 'S' shape

Those robots have this problem because they set their direction to the deepest range

→ Set a limit threshold say Threshold = 2m or 3m, then follow the center of the deepest gap instead of the deepest direction point



Race I: Reactive Methods

- **Race Format:** Time attack, single car
- **Penalties:** Crashing
- **Baseline:** Complete 5 laps without crashing
- **Example Video:** Follow the Gap in CPSWeek Grand Prix'17



Later on, as part of the course projects:

Use RL for Obstacle Avoidance with camera only

Imitation Learning:

Knowledge of follow the gap can help you train agents on Lidar and camera.

Then you can drive with camera only, without the Lidar



Key Takeaways

1. Follow the Gap is simple and effective for single vehicle driving
2. Once you setup the basic framework, a few tweaks will improve the robustness and speed
3. Reactive methods can race well!



Initialize your driver

Reactive Methods Lab

Instructor: Rahul Mangharam

Name: Student name(s), *PennId:* PennId(s)

Course Policy: Read all the instructions below carefully before you start working on the assignment, and before you make a submission.

- All sources of material must be cited. The University Academic Code of Conduct will be strictly enforced.

Goals and Learning Outcomes: The following fundamentals should be understood by the students upon completion of this lab:

- Reactive methods for obstacle avoidance

1 Overview

In this lab, you will implement a reactive algorithm for obstacle avoidance. While the base starter code defines an implementation of the F1/10 Follow the Gap Algorithm, you are allowed to submit in C++, and encouraged to try different reactive algorithms or a combination of several. In total, the python code for the algorithm is only about 120 lines.

2 Review of F1/10 Follow the Gap

The lecture slides on F1/10 Follow the gap is the best visual resource for understanding every step of the algorithm. However, the steps are outlined over here:

1. Obtain laser scans and preprocess them
2. Find the closest point in the LiDAR ranges array
3. Draw a safety bubble around this closest point and set all points inside this bubble to 0. All other non-zero points are now considered 'gaps' or 'free space'
4. Find the max length 'gap', in other words, the largest number of consecutive non-zero elements in your ranges array
5. Find the best goal point in this gap. Naively, this could be the furthest point away in your gap, but you can probably go faster if you follow the 'Better Idea' method as described in lecture.
6. Actuate the car to move towards this goal point by publishing an AckermannDriveStamped to the '/nav' topic

3 Instructions

Implement a reactive algorithm to make the car drive autonomously around the Levine Hall map. You are free to implement any reactive algorithm you want, but the skeleton code is for the F1/10 follow the gap algorithm in lecture. You can implement this node in either C++ or Python but the skeleton code is only in Python. You can download it from <https://github.com/mlab-upenn/f110-fall2019-skeletons>. You will only have to edit `reactive_gap_follow.py`. We will also be releasing a test map with obstacles for you to evaluate it on.

This is just a guide
Your actual lab is different

```

4 import concurrent.futures
5 import os
6 import sys
7 import yaml
8 from argparse import Namespace
9
10 # Get ./src/ folder & add it to path
11 current_dir = os.path.abspath(os.path.dirname(__file__))
12 sys.path.append(current_dir)
13
14 # Import your drivers here
15
16 from drivers import GapFollower
17
18 # Choose your drivers for each of the cars you have on the track here.
19 drivers = [GapFollower()]
20
21 # Choose your racetrack here
22 RACETRACK = 'Spielberg'
23
24
25 class GymRunner(object):
26
27     def __init__(self, racetrack, drivers):
28         self.racetrack = racetrack
29         self.drivers = drivers
30
31     def run(self):

```

Load map, start gym env and start positions

We initialize the environment with the map and drivers

We can have multiple drivers and need to specify the starting pose for each driver

This is just a guide
Your actual lab is different

```

28     self.racetrack = racetrack
29     self.drivers = drivers
30
31     def run(self):
32
33         # load map and the specific information for the map
34         with open('config_Spielberg_obs_map.yaml') as file:
35             conf_dict = yaml.load(file, Loader=yaml.FullLoader)
36             conf = Namespace(**conf_dict)
37
38         # Create the F1TENTH GYM environment
39         env = gym.make('f1tenth_gym:f1tenth-v0', map=conf.map_path, map_ext=conf.map_ext, num_agents=1)
40
41         # Specify starting positions of each agent
42         driver_count = len(drivers)
43         if driver_count == 1:
44             poses = np.array([[0.8007017, -0.2753365, 4.1421595]])
45         elif driver_count == 2:
46             poses = np.array([
47                 [0.8007017, -0.2753365, 4.1421595],
48                 [0.8162458, 1.1614572, 4.1446321],
49             ])
50         else:
51             raise ValueError("Max 2 drivers are allowed")
52
53         # Initially parametrize the environment
54         obs, step_reward, done, info = env.reset(poses=poses)
55
56     GymRunner > run()

```

Load map, start gym env and start positions

This is our main run loop which will drive the car for 2 laps

At each step it updates the steering and speed

```

47         [0.8007017, -0.2753365, 4.1421595],
48         [0.8162458, 1.1614572, 4.1446321],
49     ])
50     else:
51         raise ValueError("Max 2 drivers are allowed")
52
53     # Initially parametrize the environment
54     obs, step_reward, done, info = env.reset(poses=poses)
55     env.render()
56     laptime = 0.0
57     start = time.time()
58
59     # Start the simulation
60     while not done:
61         actions = []
62         futures = []
63         with concurrent.futures.ThreadPoolExecutor() as executor:
64             for i, driver in enumerate(drivers):
65                 futures.append(executor.submit(driver.process_lidar, obs['scans'][i]))
66         for future in futures:
67             # Get the actions based on the Follow the Gap Algorithm
68             speed, steer = future.result()
69             actions.append([steer, speed])
70         actions = np.array(actions)
71
72         # Send the actions from the Follow the Gap to the simulation environment
73         obs, step_reward, done, info = env.step(actions)

```

This is just a guide
Your actual lab is different

GymRunner > run() > while not done

Process Lidar to find max gap

Your task is to implement the `process_lidar` function to find the max gap and find the direction in which you can go the longest and fastest

This is just a guide
Your actual lab is different

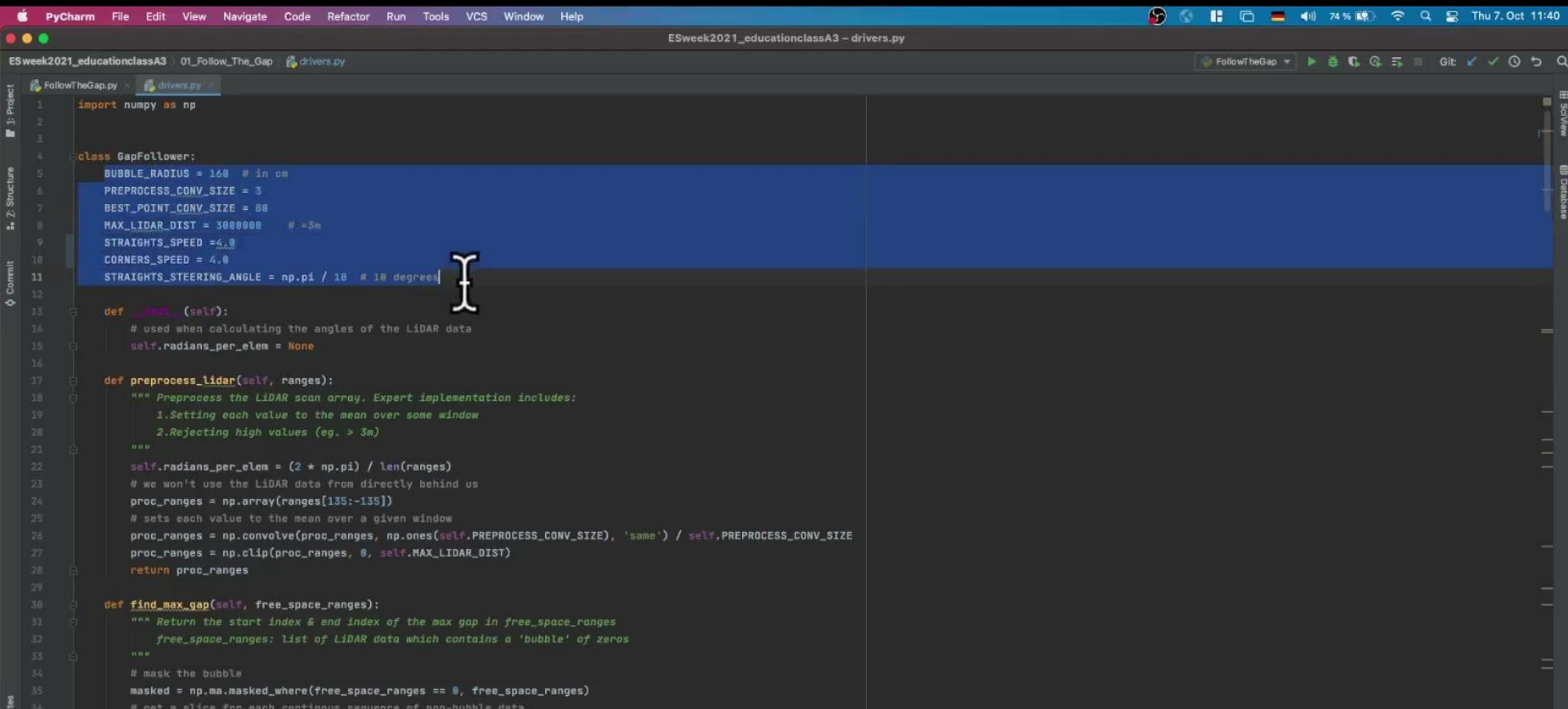
```

47         [0.8007017, -0.2753365, 4.1421595],
48         [0.8162458, 1.1614572, 4.1446321],
49     ))
50     else:
51         raise ValueError("Max 2 drivers are allowed")
52
53     # Initially parametrize the environment
54     obs, step_reward, done, info = env.reset(poses=poses)
55     env.render()
56     laptime = 0.0
57     start = time.time()
58
59     # Start the simulation
60     while not done:
61         actions = []
62         futures = []
63         with concurrent.futures.ThreadPoolExecutor() as executor:
64             for i, driver in enumerate(drivers):
65                 futures.append(executor.submit(driver.process_lidar, obs['scans'][i]))
66         for future in futures:
67             # Get the actions based on the Follow the Gap Algorithm
68             speed, steer = future.result()
69             actions.append([steer, speed])
70         actions = np.array(actions)
71
72     # Send the actions from the Follow the Gap to the simulation environment
73     obs, step_reward, done, info = env.step(actions)

```

GymRunner > run() > while not done > with concurrent.futures.ThreadPoolExecutor() as executor: > for i, driver in enumerate(drivers):

Example lap



```
PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help
ESWeek2021_educationclassA3 - drivers.py
FollowTheGap.py drivers.py
1 import numpy as np
2
3
4 class GapFollower:
5     BUBBLE_RADIUS = 160 # in cm
6     PREPROCESS_CONV_SIZE = 3
7     BEST_POINT_CONV_SIZE = 80
8     MAX_LIDAR_DIST = 3000000 # ~3m
9     STRAIGHTS_SPEED = 4.0
10    CORNERS_SPEED = 4.0
11    STRAIGHTS_STEERING_ANGLE = np.pi / 18 # 18 degrees
12
13    def __init__(self):
14        # used when calculating the angles of the LiDAR data
15        self.radians_per_elem = None
16
17    def preprocess_lidar(self, ranges):
18        """ Preprocess the LiDAR scan array. Expert implementation includes:
19            1. Setting each value to the mean over some window
20            2. Rejecting high values (eg. > 3m)
21        """
22        self.radians_per_elem = (2 * np.pi) / len(ranges)
23        # we won't use the LiDAR data from directly behind us
24        proc_ranges = np.array(ranges[135:-135])
25        # sets each value to the mean over a given window
26        proc_ranges = np.convolve(proc_ranges, np.ones(self.PREPROCESS_CONV_SIZE, 'same') / self.PREPROCESS_CONV_SIZE
27        proc_ranges = np.clip(proc_ranges, 0, self.MAX_LIDAR_DIST)
28        return proc_ranges
29
30    def find_max_gap(self, free_space_ranges):
31        """ Return the start index & end index of the max gap in free_space_ranges
32            free_space_ranges: list of LiDAR data which contains a 'bubble' of zeros
33        """
34        # mask the bubble
35        masked = np.ma.masked_where(free_space_ranges == 0, free_space_ranges)
36        # get a slice for each continuous sequence of non-bubble data
```

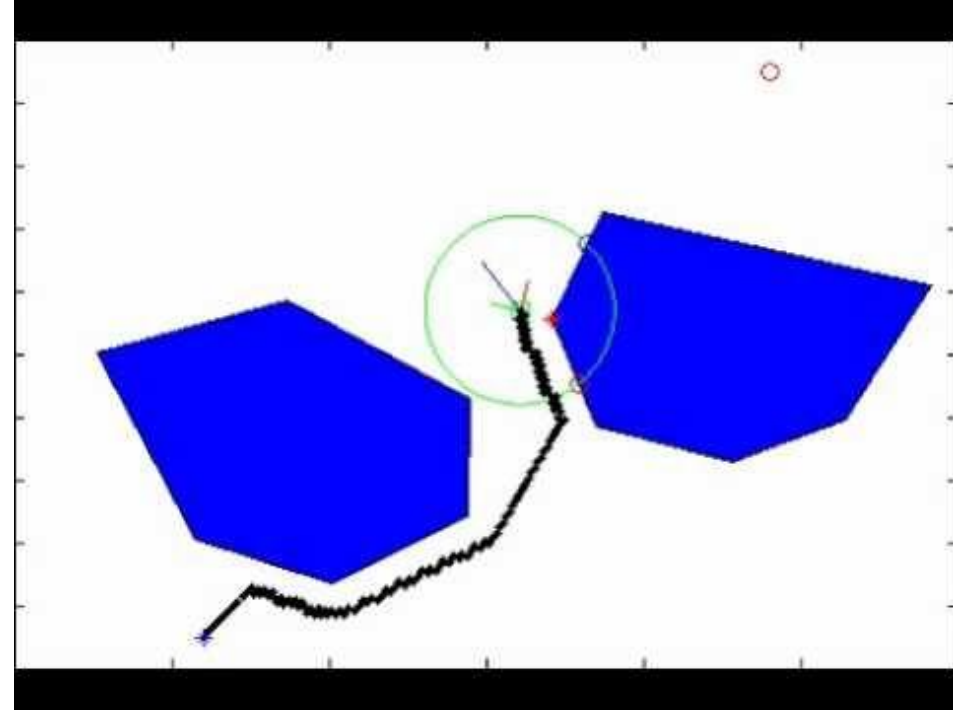
References for Gap-based Navigation Methods

1. First proposed as GND method: J. Minguez, L. Montano, T. Simeon, and R. Alami, “Global nearness diagram navigation (GND),” *IEEE ICRA* 2001.
2. Later renamed as Follow the Gap Method (FGM): V. Sezer and M. Gokasan, “A novel obstacle avoidance algorithm: follow the gap method,” *Robotics and Autonomous Systems*, Elsevier, vol. 60, no. 9, pp. 1123–1134, 2012.
3. Variants: CGF and TGF: M. Mujahed, D. Fischer, B. Mertsching, and H. Jaddu, “Closest gap based (CG) reactive obstacle avoidance navigation for highly cluttered environments,” in *IROS*, pp. 1805 – 1812, 2010. -- “Tangential gap flow (tgf) navigation: A new reactive obstacle avoidance approach for highly cluttered environments,” *ACM Rob. Auto. Sys.*, vol. 84, pp. 15–30, 2016.
4. A detailed review of reactive methods including FGM: J.A. Tobaruela and A.O. Rodriguez, “Reactive navigation in extremely dense and highly intricate environments,” *PLOS One*, <https://doi.org/10.1371/journal.pone.0189008>, pp1-51, Feb. 2017

Other reactive methods

Bug Algorithms (1996)

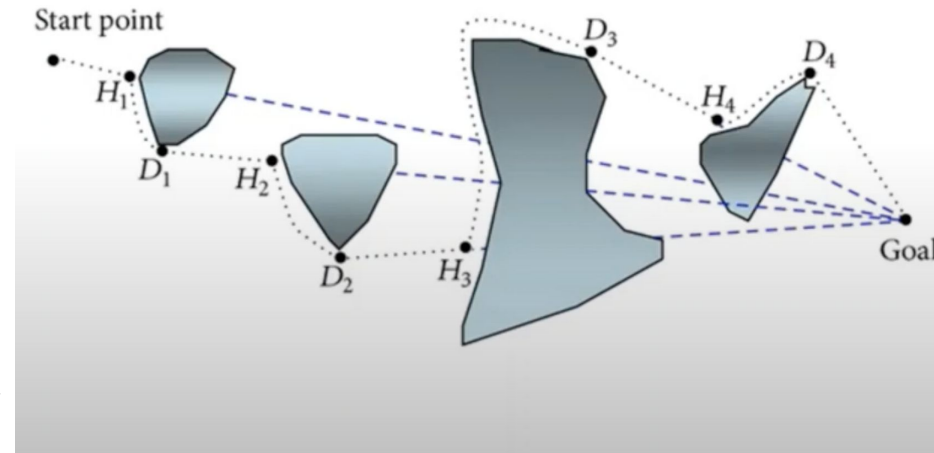
- No global model of the world, *i.e* all obstacles are unknown prior
- Only assumes *local* knowledge of environment & a **global goal** (specified by heading, distance)
- Original bug algorithm variations (*bug0*, *bug1*) dealt with planning based on tactile sensing (short range ultrasonic, bump sensors) published around 1986
- We will deal with **TangentBug**, a variation that uses range measurements from LiDAR



Distance Functions for Range Sensors

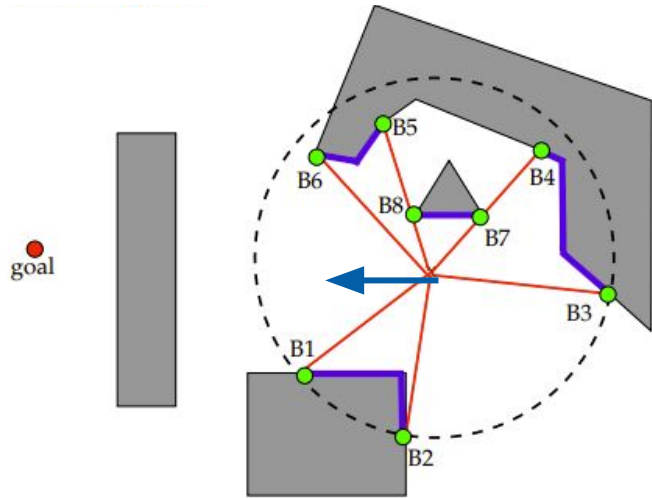
- $\rho(x, \theta)$ is the distance to the closest obstacle along the ray emanating from point $x \in \mathbb{R}^2$ at an angle $\theta \in [0, 2\pi]$
- Saturated Distance Function for

$$\rho_R(x, \theta) = \begin{cases} \rho(x, \theta), & \text{if } \rho(x, \theta) < R \\ \infty, & \text{otherwise} \end{cases}$$



TangentBug Algorithm - Idea

Use range sensor measurements to compute endpoints of continuous segments on obstacle boundaries

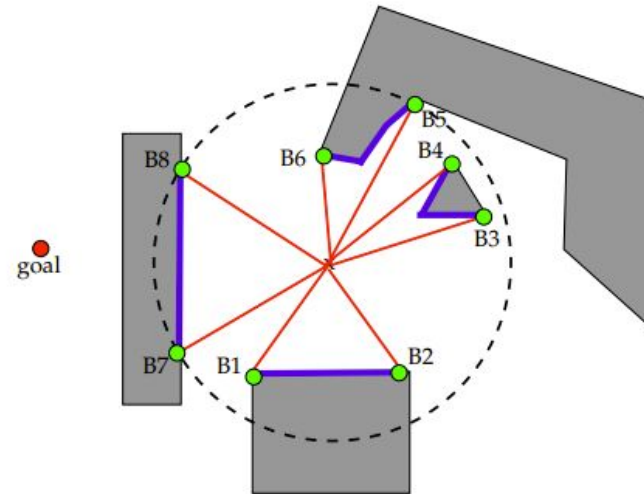
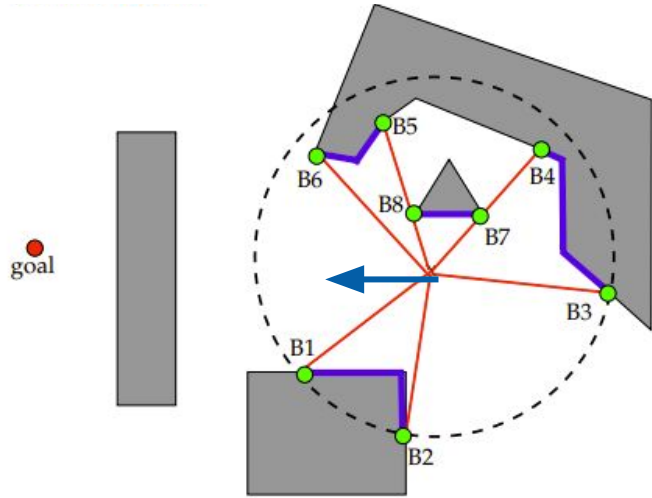


Algorithm currently thinks it has unobstructed way to goal

TangentBug Algorithm - Idea

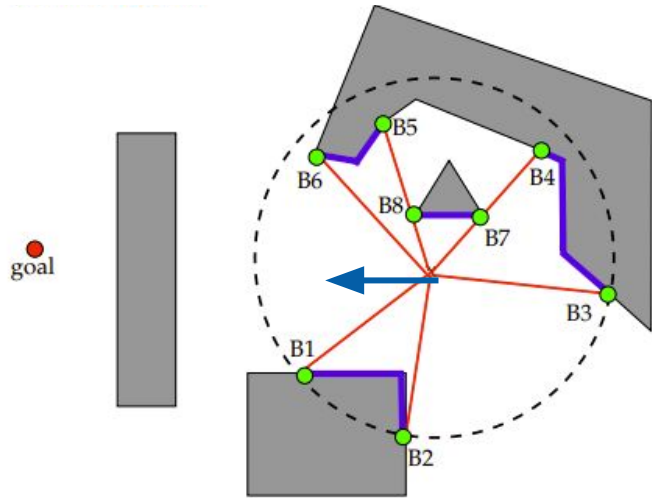
Use range sensor measurements to compute endpoints of continuous segments on obstacle boundaries

- Algorithm now sees that it can't go straight to the goal. What can it do?

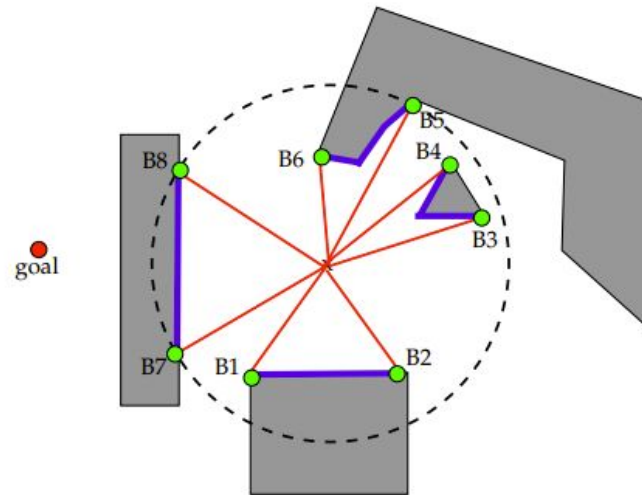


TangentBug Algorithm - Idea

Use range sensor measurements to compute endpoints of continuous segments on obstacle boundaries

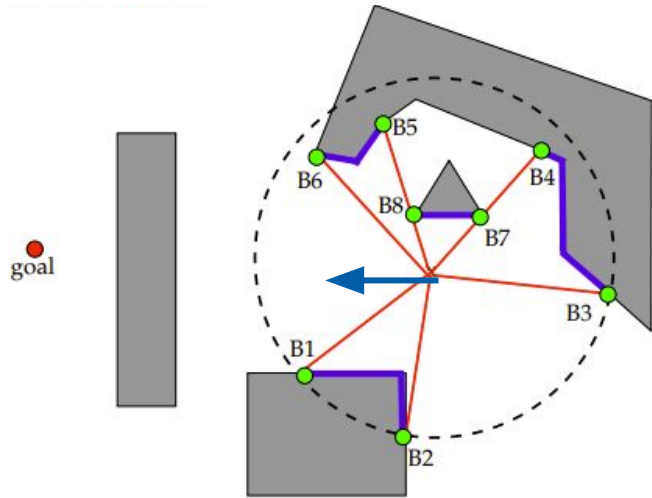


- Algorithm now sees that it can't go straight to the goal. What can it do?
- Choose point B_i that minimizes heuristic $d(x, B_i) + d(B_i, Goal)$

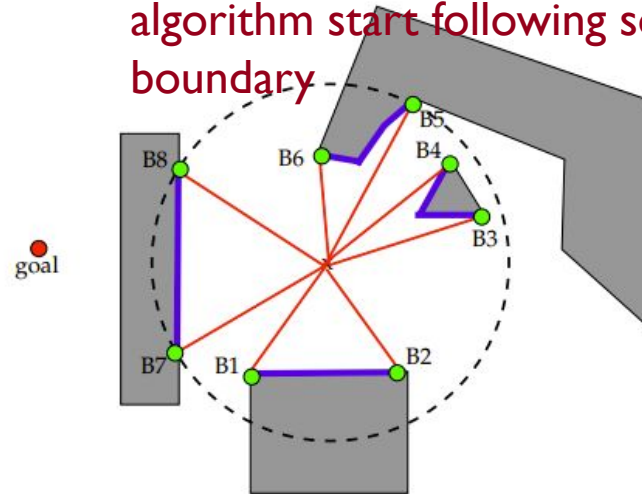


TangentBug Algorithm - Idea

Use range sensor measurements to compute endpoints of continuous segments on obstacle boundaries

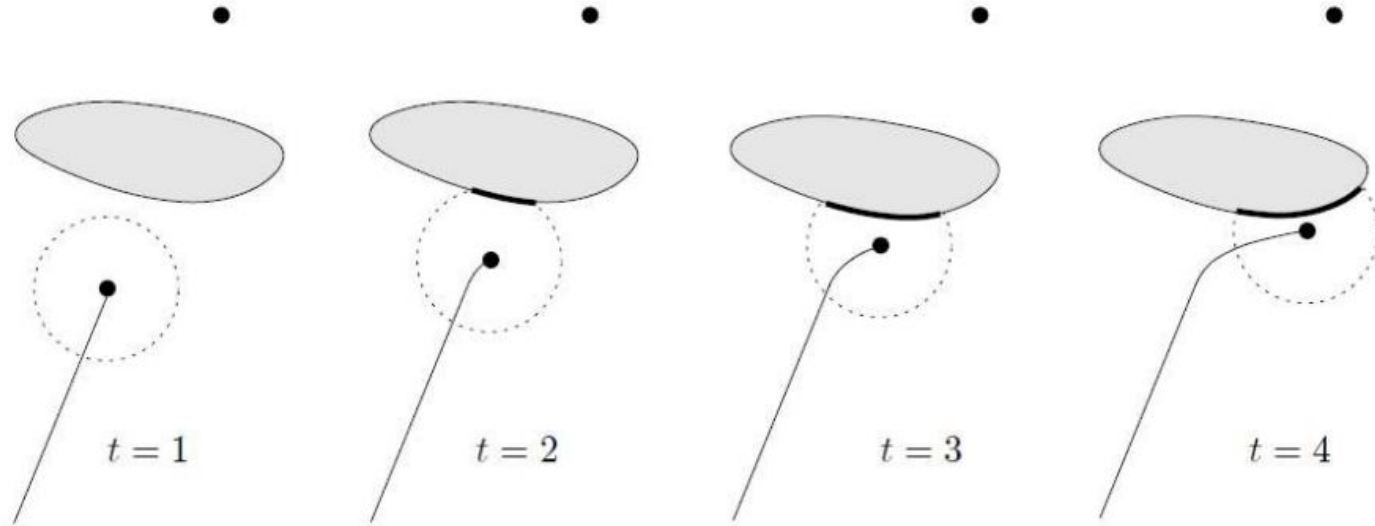


- Algorithm now see's that it can't go straight to the goal. What can it do?
- Choose point B_i that minimizes heuristic $d(x, B_i) + d(B_i, \text{Goal})$
- If this distance starts increasing, algorithm start following some boundary



TangentBug Algorithm - Example

Automatically appears to follow boundary



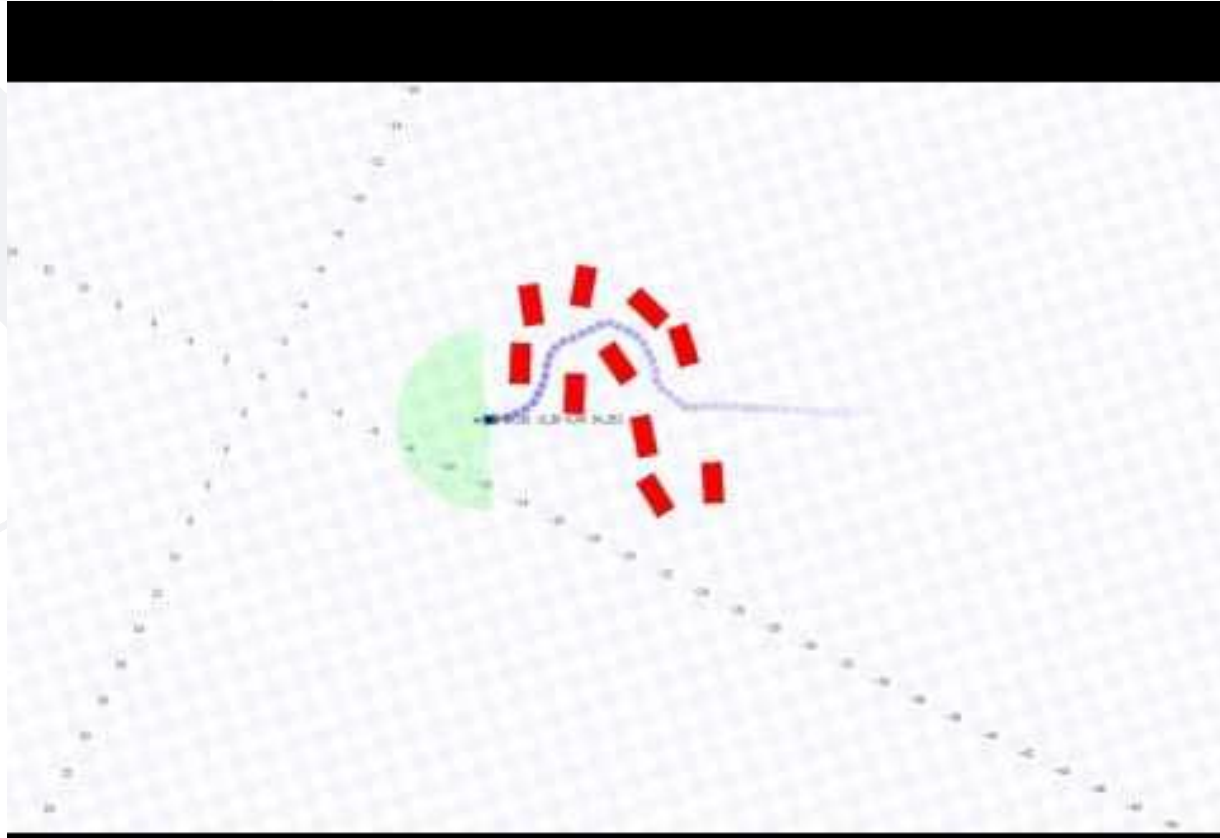
Disadvantages

- Prone to taking long trajectories towards goal, although it occasionally gets close
- Distance heuristic $d(x, B_i) + d(B_i, Goal)$ requires knowledge about distance to goal, which isn't necessarily easy to get (likely requires some beacon setup)
- Even if we let the goal be some local point in the LiDAR scan, we still need another heuristic to figure out which goal point this should be

Additional Reading:

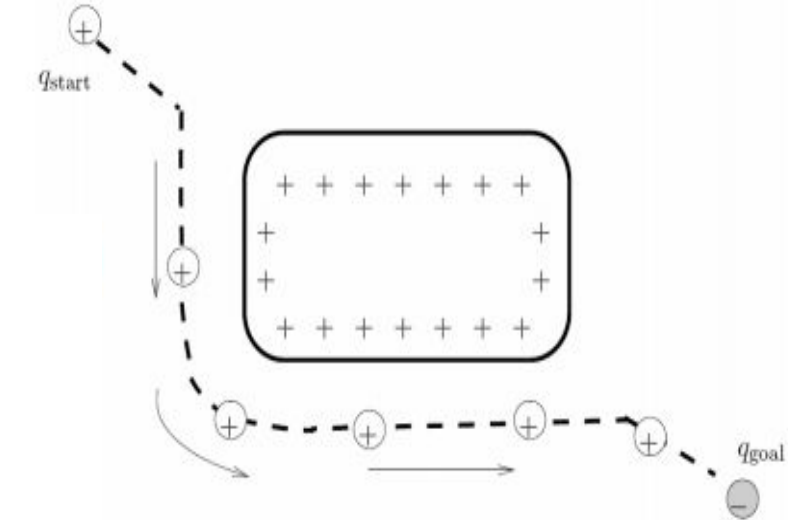
1. <http://www.robotmotionplanning.org/teaching/LecRoboBugAlgorithms.pdf> (very nice diagrams & algorithm overview)
2. https://www.cs.cmu.edu/~motionplanning/lecture/Chap2-Bug-Alg_howie.pdf (well presented guarantees on the different bug algo's)

Artificial Potential Fields



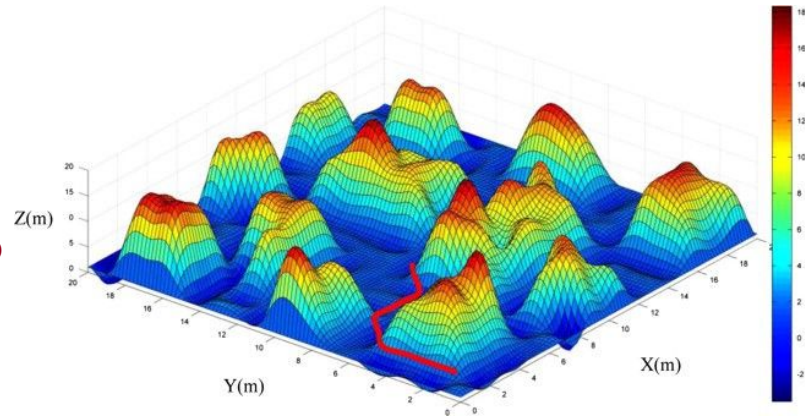
Artificial Potential Fields: Basic Idea

- Think of placing an electric/magnetic field over the environment
- Robot is at q_{start} , making its way to q_{goal}
- Attach positive charges to obstacle boundaries
- Attach a negative charge to your goal
- This gives you a potential field that can be expressed as $U(q) = U_{att}(q) + U_{rep}(q)$



Reactive Planning With Gradient Descent

- In 3D, this potential function gives us a local 'landscape'
- From a reactive standpoint, we will have to create a new 'landscape' for every timestep using the local LiDAR scan
- Can move towards goal via **gradient descent**:
 - $q(0)=q_{\text{start}}$
 - $i = 0$
 - while $\nabla U(q(i)) \neq 0$ do
 - $q(i+1) = q(i) - \alpha(i) \nabla U(q(i))$
 - $i=i+1$

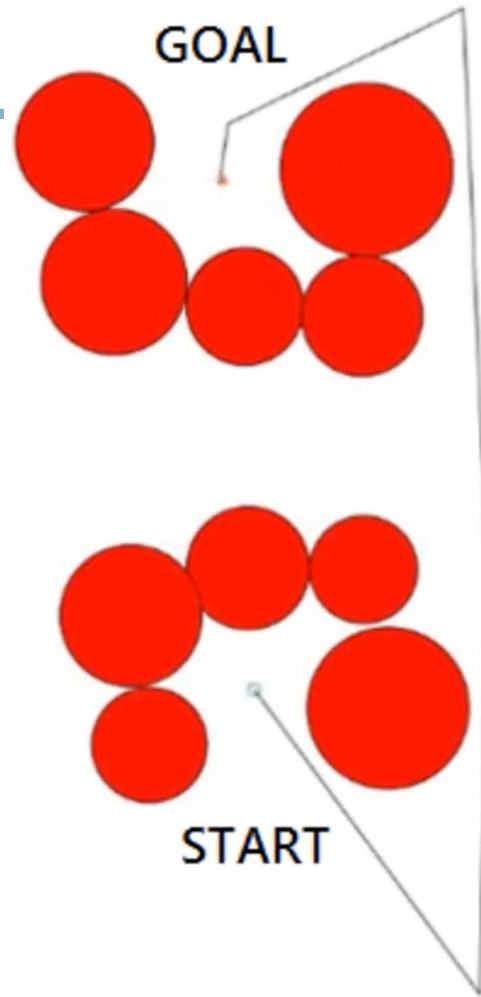


Disadvantages

- Local Minima with Gradient Descent!
 - Several variations of the original algorithm have been proposed to reduce this problem, some of these require a global map (*harmonic potential fields*)
- In reactive implementation, have to repeatedly do gradient descent at each timestep
 - For small local scans, can also use ***brushfire algorithm***
- Once again, how do I choose the goal point?

Additional Reading:

1. <https://ieeexplore.ieee.org/abstract/document/4587222> (harmonic potential fields)
2. https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf (APF's in greater depth, brushfire)
3. <https://www.coursera.org/learn/robotics-motion-planning> (great deep dive by Prof. CJ Taylor on this method & variations)



Takeaway on Alternate algorithms

- Most of them require some notion of a global goal, which we don't actually have in a race track
- FITenth FGM implicitly encodes the global goal, which may be a good or bad thing
- **Try your own thing for RACE I!** You have base code outlining the algorithm