Autonomous Racing

Mapping · Optimization · Control

CPEN 391: F1TENTH Autonomous Racing Project

Group A1: Can, Ian, Miguel, Pete, Wit

Mapping → Raceline Optimization → Pure Pursuit → MPC

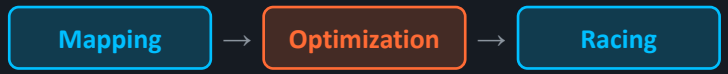
PROBLEM STATEMENT & GOALS

What Are We Building?

🎯 Goal

Build a fully autonomous pipeline that maps an unknown track, computes an optimal raceline, and races it as fast as safely possible.

Three-Phase Pipeline



⚙️ Constraints

Framework

ROS 2 nodes: mapping, optimizer, racing, safety

Libraries

NumPy + SciPy

Safety

AEB safety node runs in parallel at all times

SYSTEM PIPELINE

How the car goes from a blank map to racing autonomously

Safety Node (AEB) runs in parallel — monitors TTC and distance, overrides drive commands when needed

TRACK MAPPING

How It Works

1 LiDAR beams at ±90°

Sample the closest valid point directly left and right of the car

2 World-frame transform

Both hits converted to global coords using current odometry pose

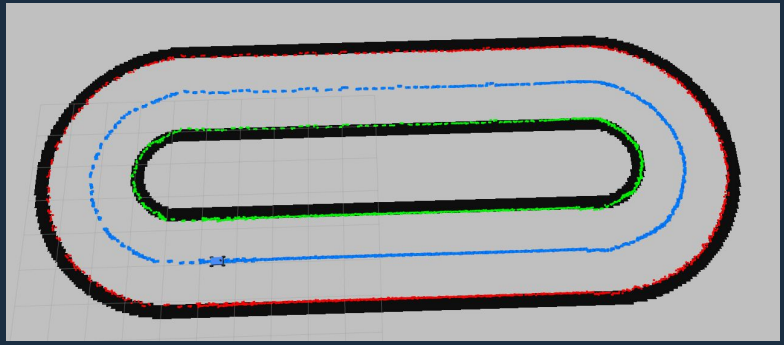
3 Centerline midpoint

Average of left & right wall hits

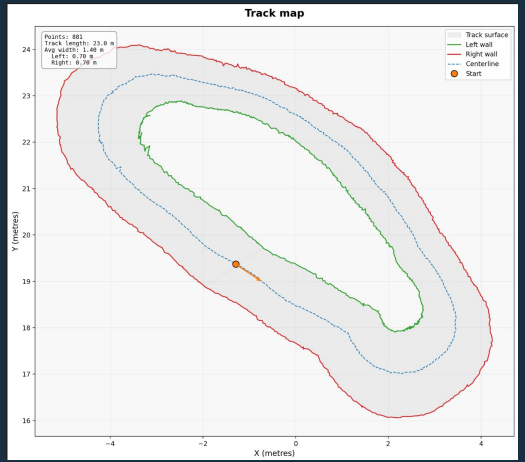
4 Width storage

Distance from center to each wall stored as width_right, width_left

```
Output: track.npy  –  (N, 4)
[x_center, y_center, width_right, width_left]
```



Live RViz view — simulator mapping



Resulting track map — physical demo

RACELINE OPTIMIZATION

Minimum-Curvature QP

Core Idea

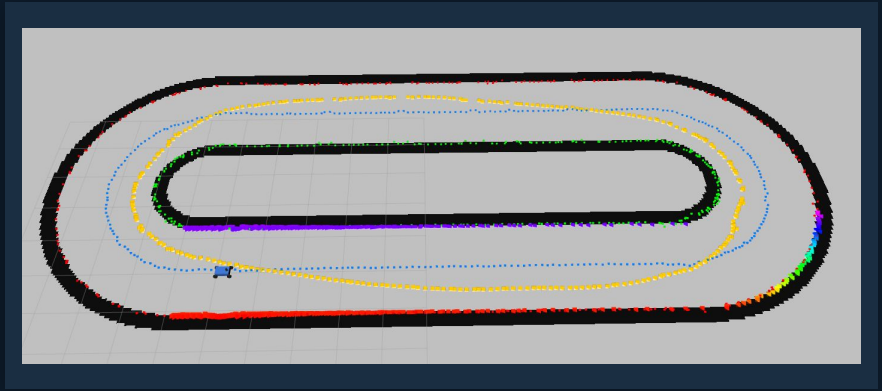
Lower curvature = smoother path = higher sustainable speed.

The optimizer shifts each of the 300 waypoints left or right within the track corridor, finding the combination with the minimum total curvature — subject to staying **15 cm away** from each wall.

Corridor Constraints

Each waypoint is constrained to stay within [right wall + margin, left wall − margin] along the local normal direction. X and Y are solved jointly.

Solver: scipy trust-constr · 300 waypoints · ~500 iterations to convergence



Optimizer output — raceline in RViz (simulator)

1. **Reconstruct Boundaries**
   Center ± widths × local normal ⟶ left/right walls
2. **Build Curvature Matrix**
   Second-difference operator over all N waypoints (periodic)
3. **Solve QP Jointly**
   X and Y optimized together with corridor constraints
4. **Assign Speed Profile**
   Flat 2.0 m/s; racing.py derives curvature-based speeds

CONTROL STACK

Two controllers — selectable via USE_MPC flag in racing.py

Pure Pursuit

› Lookahead goal: first waypoint ≥ 1.5 m ahead

› Reads current position & heading from odometry

› Computes steering angle to arc toward the target point

› Speed set per-waypoint: slows for curves, faster on straights

› Tracks the nearest raceline waypoint as the car advances

› Detects raceline direction on startup to avoid driving backwards

Kinematic MPC

› Plans steering + acceleration 10 steps (0.5 s) into the future

› Uses position, heading & speed from odometry as current state

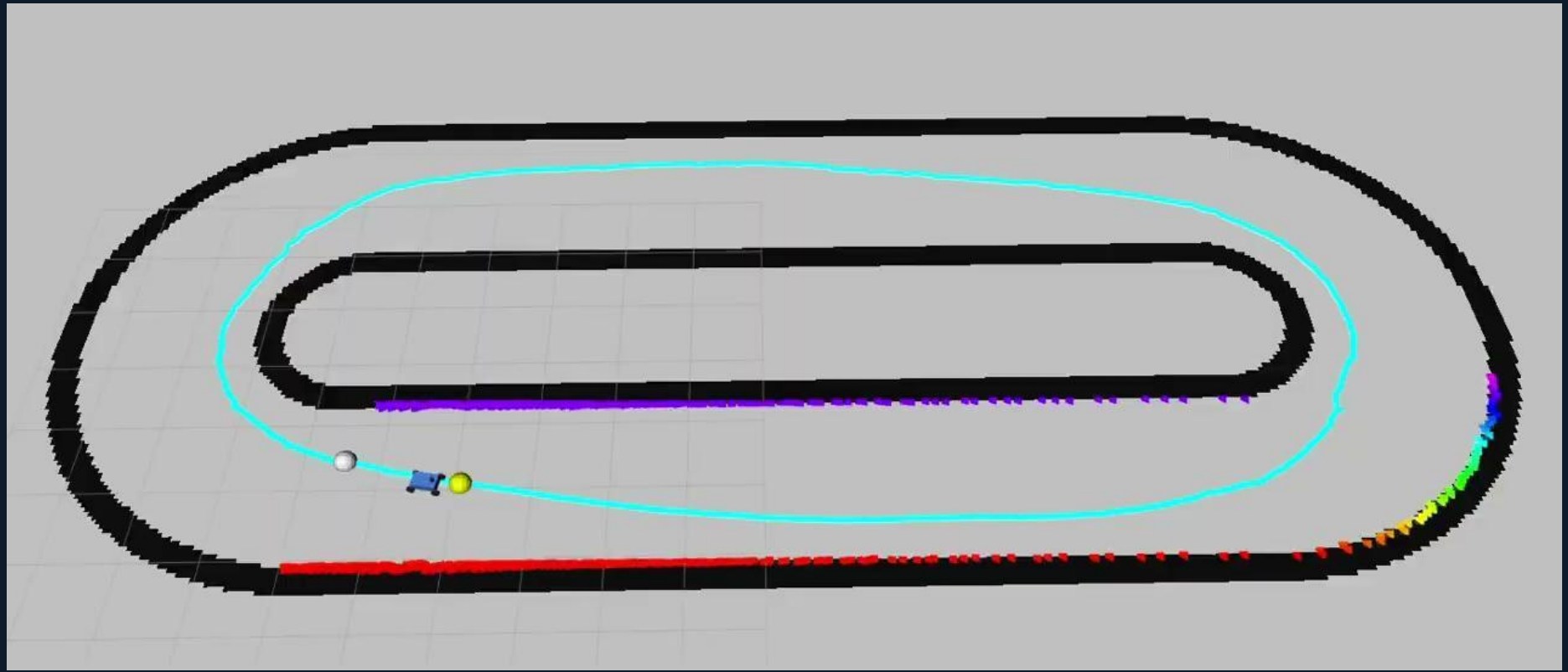
› Minimizes track-following error while penalizing jerky inputs

› Replans at 20 Hz — always working from the latest odometry

› Reuses the previous plan as a starting point for faster solving

› Speed target adjusts automatically for straights vs. corners

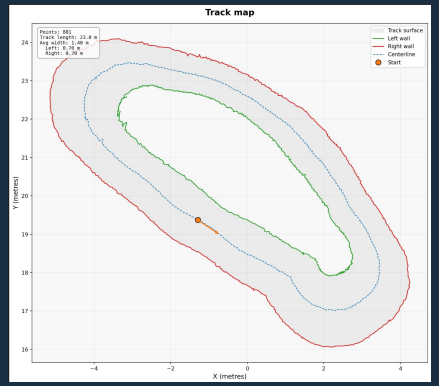
Short Demo (https://youtu.be/9zyfzv9-PDA)



DEMO RESULTS

Track Mapping

✓ WORKED

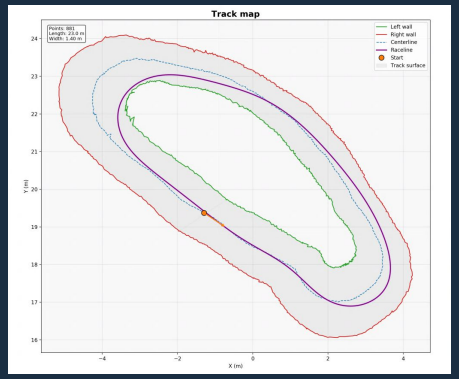


Clean oval map from demo

LiDAR-based mapper produced relatively clean readings with accurate centerline and width estimates for the oval track.

Raceline Optimizer

✓ CONVERGED



Raceline in purple

Raceline takes a wider arc on one oval turn:
- mathematically correct for min-curvature, but is not necessarily globally optimal for lap time

Pure Pursuit

✓ SIMULATOR

Navigated raceline cleanly in simulation. Minimally tested on physical car — limited track time before demo.

MPC

⚠ NEEDS FIXES

Simulator issues remain — cost tuning and constraint handling. Not tested on hardware.

KEY TAKEAWAYS

What We Learned

Centerline + width representation works

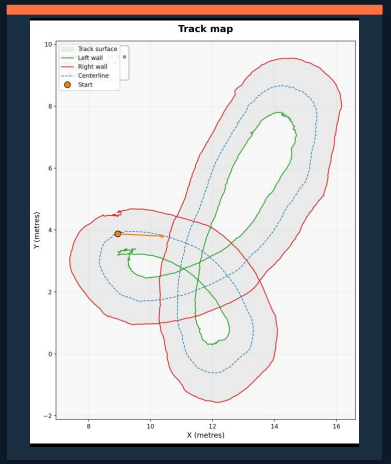
Compact (N×4) and directly usable by the optimizer

Min-curvature ≠ apex cutting

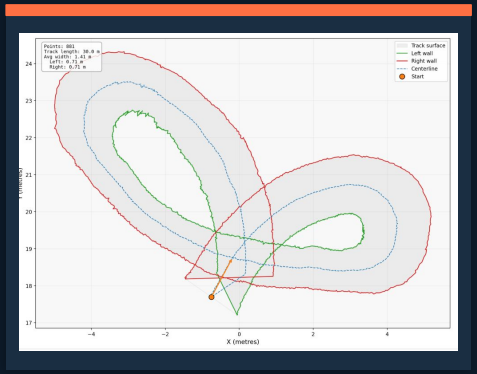
Wide arcs are mathematically optimal for curvature; physical cars may prefer apex

Drift correction matters

Linear loop-closure fix essential for closed-track raceline quality



Early physical run — multiple laps of drift accumulation (corrected by one lap mapping with loop-closure fix)



Early physical run — one lap mapping (less drift, no loop-closure)

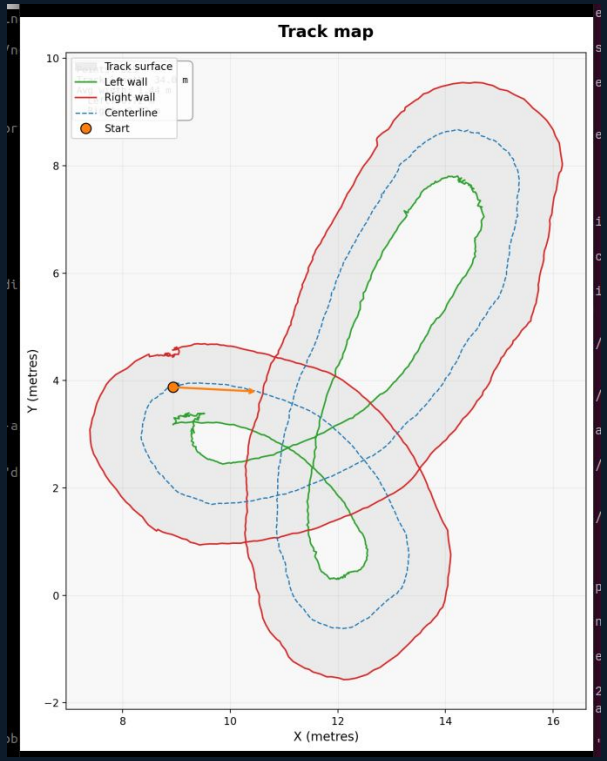
Thank you for listening!

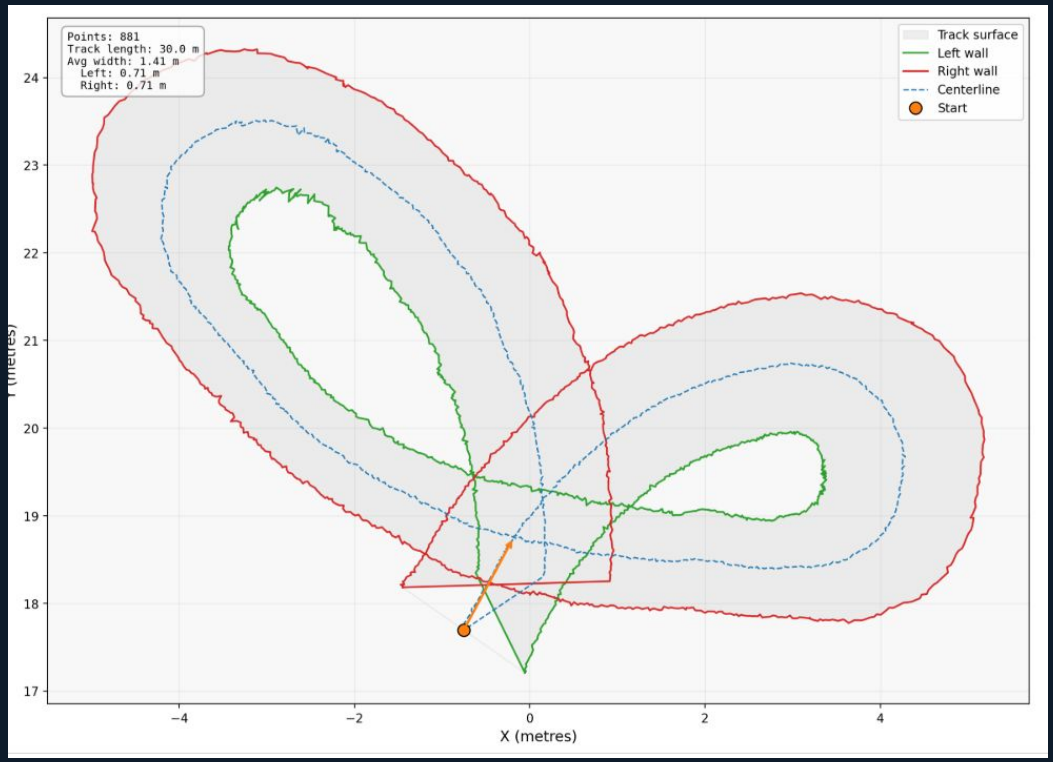
Questions?

Resources

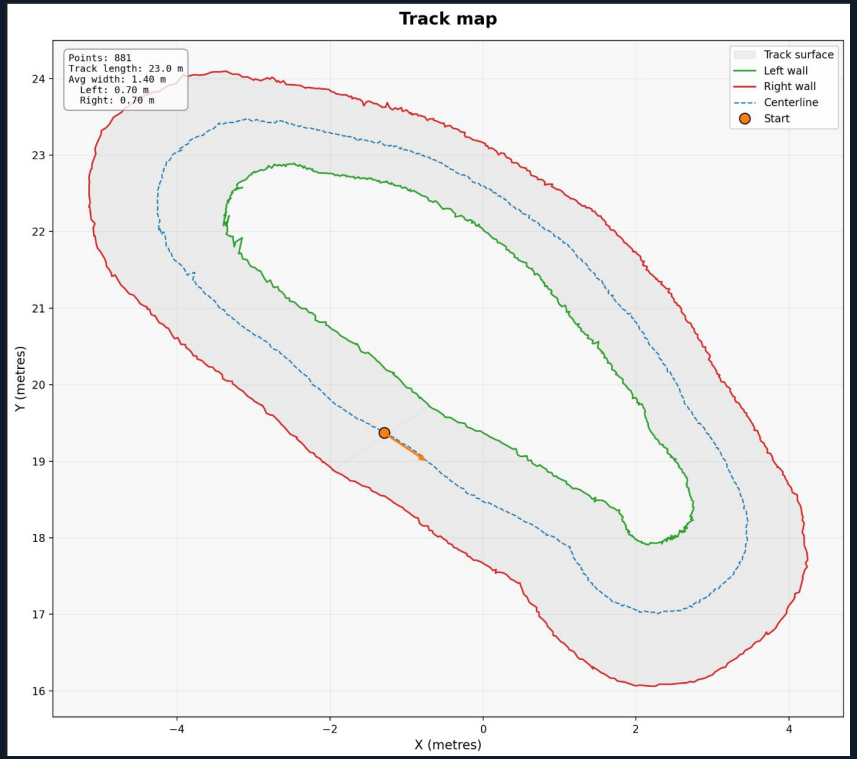
- https://docs.google.com/presentation/d/1rudShBR5f63UaBfhXmK2RvD5Za5fkx4i5Pv8px9o-2E/edit
  - UPenn Raceline Optimization
- https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf
  - Pure Pursuit
- https://github.com/TUMFTM/global_racetrajectory_optimization
  - TUMFTM global_racetrajectory_optimization

Appendix (Failed mapping attempts on the real car)





Appendix



- Optimization problem:

$$\text{minimize } [\alpha_i \dots \alpha_N] \sum_{i=1}^{N} \kappa_i^2(t)$$

$$\text{subject to} \quad \alpha_i \in [\alpha_{i,\min}, \alpha_{i,\max}] \quad \forall\, 1 \le i \le N$$

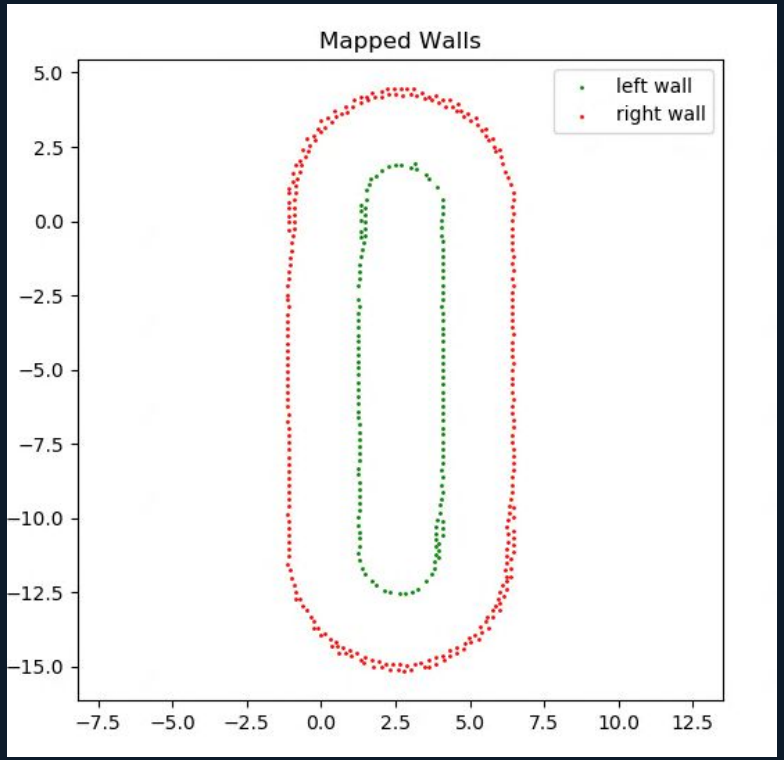
- Evaluation solely at the discretization points: $t = 0$
- Calculation of the squared curvature values

$$\kappa_i = \frac{x_i' y_i'' - y_i' x_i''}{\left(x_i'^2 + y_i'^2\right)^{\frac{3}{2}}} \quad \Rightarrow \quad \kappa_i^2 = \frac{x_i'^2 y_i''^2 - 2x_i' x_i'' y_i' y_i'' + y_i'^2 x_i''^2}{\left(x_i'^2 + y_i'^2\right)^3}$$

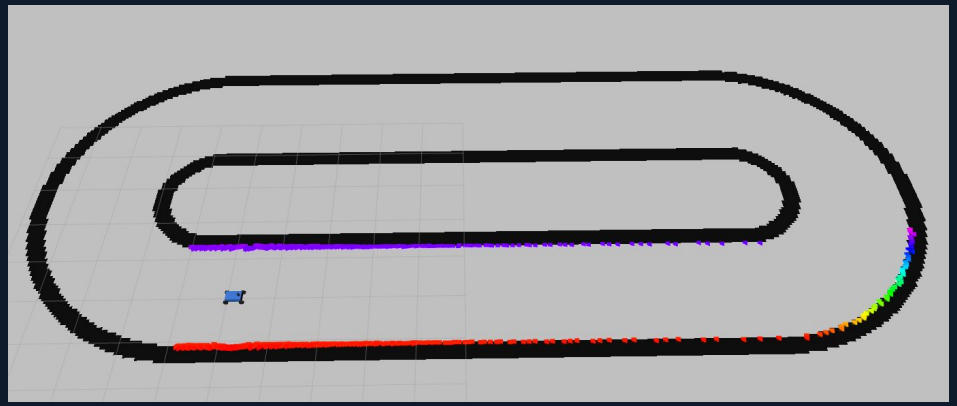
- Several matrix reshaping operations lead to an implementable formulation of the problem (see code on GitHub)

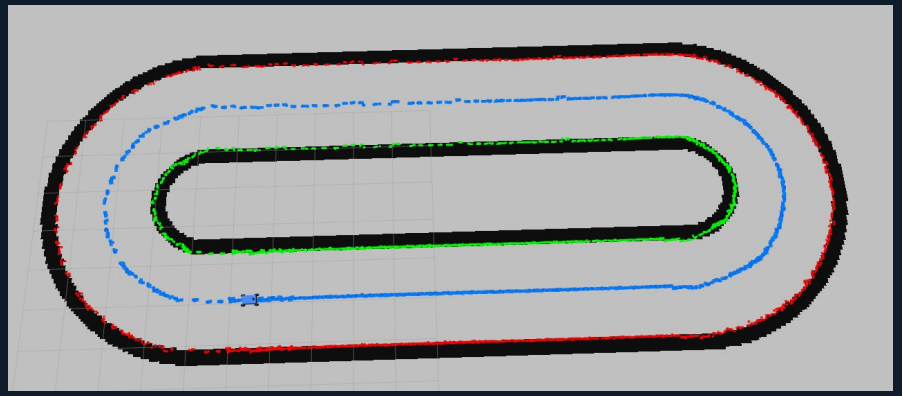
Penn Engineering

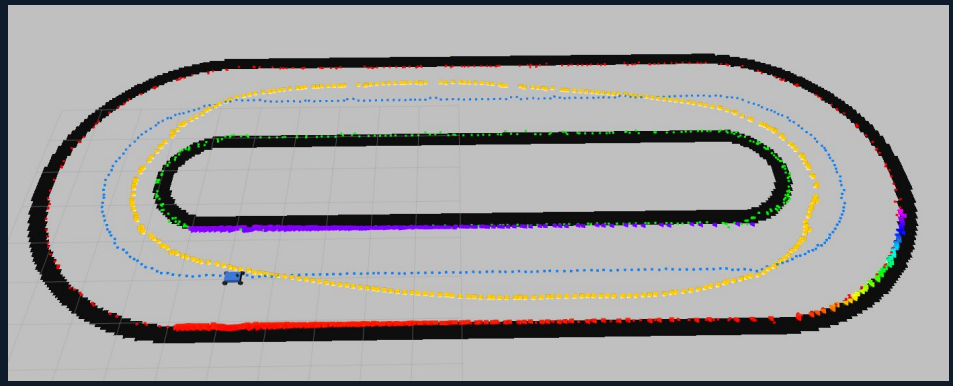
Appendix


Mapped Walls
left wall
right wall
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5

Appendix







Appendix (https://youtu.be/9zyfzv9-PDA)