Custom Mapping and Raceline Optimization

CAN KIRLI, University of British Columbia, Canada
IAN LEE, University of British Columbia, Canada
MIGUEL ALESANDRO TAMPUBOLON, University of British Columbia, Canada
PETE THARANAMAI, University of British Columbia, Canada
WIT LIN, University of British Columbia, Canada

This report presents the design and implementation of an end-to-end autonomous racing pipeline for the F1TENTH platform. The original proposal targeted a full Raceline Optimization with Model Predictive Control (MPC) system; the final scope concentrates on the two most technically demanding and novel contributions: (1) a custom LiDAR-based track mapping node that operates without a third-party SLAM package, and (2) a minimum-curvature raceline optimizer that produces smooth, boundary-respecting racing lines from raw map data.

The mapping node fuses LiDAR perpendicular-wall hits with odometry to build a track representation stored as centerline coordinates plus per-point wall widths. The optimizer resamples this representation, reconstructs track boundaries via spline fitting, and solves a quadratic program with corridor constraints to yield a minimum-curvature raceline with a curvature-based speed profile. Pure Pursuit tracking of the generated raceline was validated in simulation. MPC was implemented and partially tested but is excluded from the demo due to insufficient hardware test time. The system demonstrated robust mapping on the physical track with limited odometry drift, distinguishing it from approaches observed in peer teams.

Additional Key Words and Phrases: autonomous racing, LiDAR, MPC, raceline optimization, F1TENTH, pure pursuit, kinematic bicycle model, mapping, SLAM

1 Literature Review

1.1 Mapping and Simultaneous Localisation and Mapping (SLAM)

Classical SLAM approaches such as Extended Kalman Filter SLAM (EKF-SLAM) and Graph-SLAM provide well-understood probabilistic frameworks for simultaneous mapping and localisation. However, these methods are computationally expensive and were not designed with racing throughput in mind [6]. SLAM Toolbox by Macenski et al. offers a robust, production-grade ROS 2 mapping solution that many F1TENTH teams deploy out of the box [5]. While SLAM Toolbox produces high-quality occupancy grid maps, it does not extract a structured track representation (centerline + widths) directly usable by a raceline optimizer, requiring an additional post-processing step. Our approach bypasses this by building a compact track model directly from LiDAR geometry, avoiding the computational overhead of full occupancy-grid SLAM and the dependency on a separate extraction pipeline.

1.2 Trajectory Optimization

Heilmeier et al. formulated minimum-curvature trajectory planning as a quadratic program (QP), demonstrating that minimizing the integral of squared curvature produces smooth trajectories that exploit available track width [1]. Their method assumes full prior knowledge of the track. The University of Waterloo F1TENTH team extended offline trajectory planning concepts to the small-scale racing domain, achieving time-optimal lap times through iterative numerical optimization, though their pipeline also relied on an offline, pre-surveyed track model [2]. Our work integrates a

Authors' Contact Information: Can Kirli, cankrl05@gmail.com, University of British Columbia, Vancouver, British Columbia, Canada; Ian Lee, ian.ant.lee@gmail.com, University of British Columbia, Vancouver, British Columbia, Canada; Miguel Alesandro Tampubolon, miguelalesandro.tamp@gmail.com, University of British Columbia, Vancouver, British Columbia, Canada; Pete Tharanamai, warotpete@gmail.com, University of British Columbia, Vancouver, British Columbia, Canada; Wit Lin, witlin55@gmail.com, University of British Columbia, Vancouver, British Columbia, Canada.

similar minimum-curvature QP formulation directly downstream of a live mapping step, enabling the system to operate on a freshly driven map without manual track surveys.

1.3 Control Architectures

Pure Pursuit is a classical path-tracking controller that steers a vehicle toward a lookahead point on a reference trajectory. Its simplicity and robustness to model mismatch make it a common baseline in F1TENTH competitions [4]. Model Predictive Control (MPC) generalizes this by optimizing a sequence of control inputs over a receding horizon, explicitly accounting for vehicle dynamics, actuator limits, and multi-objective cost functions. Kinematic bicycle models are frequently used for MPC on F1TENTH hardware due to their low computational cost relative to dynamic models [3]. Our implementation employs a kinematic bicycle MPC with cross-track error, heading error, and speed error penalization, alongside steering smoothness terms.

2 Solution and Implementation

The system comprises four sequential stages executed as separate ROS 2 nodes: (i) wall-following for data collection, (ii) LiDAR-based mapping, (iii) raceline optimization, and (iv) racing control. Figure 1 illustrates the overall data flow.

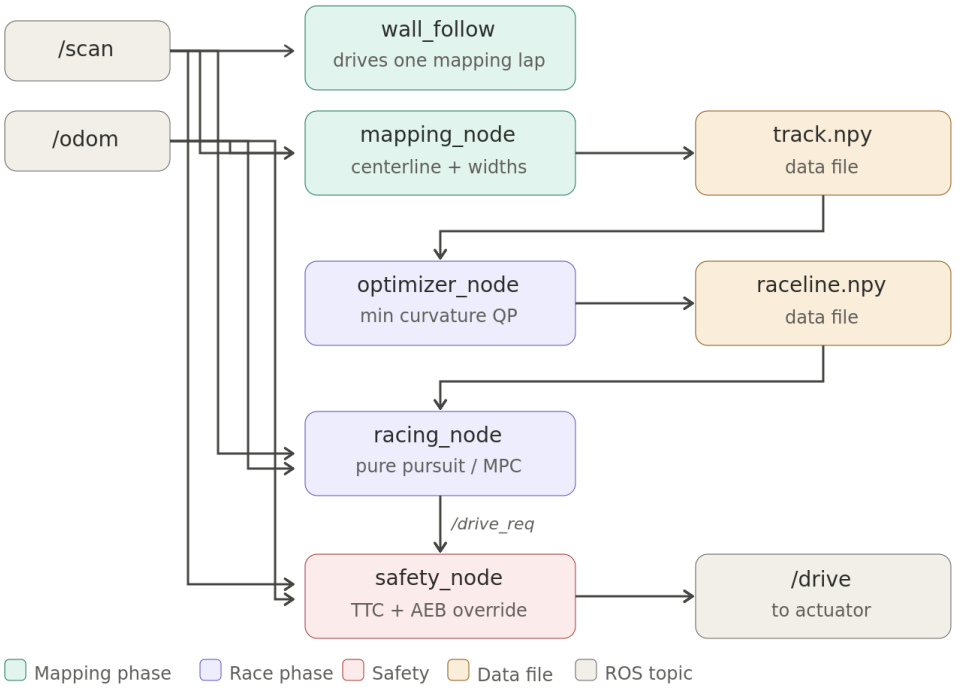

Fig. 1. End-to-end system architecture. Arrows denote ROS 2 topic or file dependencies.

2.1 Stage 1: Wall-Following for Track Traversal

During the mapping phase, the vehicle drives around the track using a PID-based wall-following controller (`wall_follow.py`). The controller subscribes to `/scan` and publishes Ackermann drive commands to `/drive_req`, which are passed through a safety node before reaching the actuators.

The wall follower extracts two LiDAR beams—one perpendicular to the chosen wall and one at a 45-degree offset—and applies the geometric relationship:

$$\alpha = \text{atan2}(a\cos\theta - b,\ a\sin\theta), \tag{1}$$

$$D_{\text{current}} = b\cos\alpha, \tag{2}$$

$$D_{\text{future}} = D_{\text{current}} + L\sin\alpha, \tag{3}$$

where a and b are the two range measurements, $\theta = 45^\circ$ is the beam separation angle, L is a lookahead projection distance, and D_{future} is the predicted wall distance after advancing L metres. The PID error is $e = D_{\text{desired}} - D_{\text{future}}$, with gains $K_p = 1.0$, $K_i = 0.0001$, $K_d = 0.1$. Velocity is reduced proportionally to steering magnitude to stabilize sharp corners.

2.2 Stage 2: LiDAR Mapping Node

The mapping node (`mapping.py`) builds a compact track representation from live sensor data. It subscribes to `/scan` and `/ego_racecar/odom` (simulator) or `/odom` (hardware), and saves the result to `track.npy` on Ctrl-C termination.

2.2.1 Wall Point Extraction. For each incoming scan, the node searches a 40-degree angular window centred at $+90^\circ$ (left) and -90° (right) to find the LiDAR beam closest to perpendicular. Range readings outside $[0.15\,\text{m},\ 10.0\,\text{m}]$ are rejected. The best beam (minimum angular error to the target direction) is selected on each side.

2.2.2 Coordinate Transformation. The selected wall hits are expressed in the vehicle local frame as $(r\cos\theta,\ r\sin\theta)$. They are transformed to the world frame using the odometry-derived pose (x, y, ψ):

$$x_w = \cos\psi\, x_l - \sin\psi\, y_l + x_{\text{odom}}, \tag{4}$$

$$y_w = \sin\psi\, x_l + \cos\psi\, y_l + y_{\text{odom}}, \tag{5}$$

where (x_l, y_l) is the local-frame point and ψ is the vehicle yaw recovered from the quaternion odometry message via:

$$\psi = \text{atan2}\Big(2(q_w q_z + q_x q_y),\ 1 - 2(q_y^2 + q_z^2)\Big). \tag{6}$$

2.2.3 Centerline and Width Computation. The world-frame midpoint of the left and right wall hits defines the centerline:

$$x_c = \tfrac{1}{2}(x_{\text{left}} + x_{\text{right}}), \quad y_c = \tfrac{1}{2}(y_{\text{left}} + y_{\text{right}}). \tag{7}$$

Track widths are stored as the Euclidean distances from the centerline to each wall:

$$w_{\text{left}} = \|p_{\text{left}} - p_c\|_2, \quad w_{\text{right}} = \|p_{\text{right}} - p_c\|_2. \tag{8}$$

Each new centerline point is checked against a spatial grid (cell size = 0.01 m) to suppress duplicates. The final file `track.npy` has shape $(N, 4)$: $[x_{\text{center}},\ y_{\text{center}},\ w_{\text{right}},\ w_{\text{left}}]$.

2.3 Stage 3: Raceline Optimizer

The optimizer node (`optimizer.py`) loads `track.npy`, runs a multi-step pipeline, and saves `raceline.npy` with shape $(N, 3)$: $[x,\ y,\ v_{\text{target}}]$.

2.3.1 Preprocessing. The raw track undergoes the following cleaning steps:

- **Lap detection:** the pipeline identifies the first index at which the car returns within 1.0 m of the start point (after completing at least 50% of collected points), trimming the dataset to a single clean lap.

- **Width filtering:** points with unrealistic widths (outside $[0.1\,\text{m},\ 3.0\,\text{m}]$) are discarded as LiDAR outliers.
- **Jump filtering:** consecutive points separated by more than 2.0 m are removed to eliminate pose-estimation discontinuities.
- **Resampling:** a closed periodic cubic smoothing spline is fitted to the cleaned centerline (`scipy.interpolate.splprep` with $s = 1.0$), and 300 uniformly spaced waypoints are sampled. Track widths are linearly interpolated to the new parameterisation.

2.3.2 Boundary Reconstruction. Left and right wall positions at each resampled centerline point are computed using local normal vectors. The unit normal pointing left is obtained by a 90° CCW rotation of the tangent:

$$\mathbf{n}_i = [-t_y,\ t_x]/\|\mathbf{t}\|, \tag{9}$$

$$p_{\text{left},i} = c_i + w_{\text{left},i}\,\mathbf{n}_i, \quad p_{\text{right},i} = c_i - w_{\text{right},i}\,\mathbf{n}_i. \tag{10}$$

2.3.3 Raceline-Like Seed Path. A heuristic seed path biases the solver toward an outside-inside-outside racing line. Signed curvature κ is computed at each waypoint using the Menger formula. A future-curvature signal is obtained by rolling the smoothed curvature forward by 50 waypoints. For each point, a composite lateral offset α moves the car to the outside before a corner (setup), toward the inside near the apex, and toward the outside on straights (exit). Strengths are normalized to $[0, 1]$ using the 88th percentile of absolute curvature. One hundred passes of exponential smoothing are applied before computing: $\text{seed}_i = c_i + \alpha_i\,\mathbf{n}_i$.

2.3.4 Minimum-Curvature QP Formulation. The optimization variable is

$$\mathbf{z} = [x_1, \ldots, x_N,\ y_1, \ldots, y_N]^\top \in \mathbb{R}^{2N}.$$

The objective minimizes:

$$\min_{\mathbf{z}}\ \mathbf{z}^\top \left(D^\top D \otimes I_2\right)\mathbf{z} + W_c\|\mathbf{z} - \mathbf{z}_{\text{center}}\|^2 + W_s\|\mathbf{z} - \mathbf{z}_{\text{seed}}\|^2, \tag{11}$$

where D is the $N \times N$ periodic second-difference matrix (curvature proxy), $\mathbf{z}_{\text{center}}$ is the resampled centerline, and $\mathbf{z}_{\text{seed}}$ is the seed path. Weights $W_c = 10^{-3}$ and $W_s = 0.08$ balance curvature minimization against centerline and seed attraction.

Corridor constraints enforce that each optimized point lies between the right and left walls, set back by safety margin $m = 0.15\,\text{m}$:

$$\mathbf{n}_i \cdot p_{\text{right},i} + m\ \le\ \mathbf{n}_i \cdot \mathbf{z}_i\ \le\ \mathbf{n}_i \cdot p_{\text{left},i} - m. \tag{12}$$

The QP is solved using `scipy.optimize.minimize` with `method='trust-constr'`, initialized from the seed path, and run for up to 500 iterations.

2.3.5 Speed Profile Generation. The speed at each waypoint is set using a curvature-based formula:

$$v_i = \text{clip}\left(\sqrt{\frac{a_{\text{lat,max}}}{\max(|\kappa_i|, 10^{-4})}},\ v_{\min},\ v_{\max}\right), \tag{13}$$

where $a_{\text{lat,max}} = 6.0\,\text{m/s}^2$, $v_{\min} = 1.5\,\text{m/s}$, and $v_{\max} = 4.0\,\text{m/s}$. Twenty-five passes of weighted smoothing are applied to prevent abrupt speed transitions.

2.4 Stage 4: Racing Control

2.4.1 Pure Pursuit. The Pure Pursuit controller (`USE_MPC = False` in `racing.py`) identifies the nearest raceline waypoint by scanning a forward window of 60 points from the last known nearest index. It then searches forward along the raceline for the first waypoint at the lookahead distance

($L_d = 1.5$ m) that lies ahead of the vehicle. The goal point is converted to the vehicle frame and the steering angle is computed as:

$$\delta = \text{atan2}(2L_{\text{wb}} \sin\alpha,\ L_d), \tag{14}$$

where $L_{\text{wb}} = 0.33$ m is the wheelbase and α is the angle to the goal in the vehicle frame. The target speed is read directly from the raceline at the goal waypoint index.

On first odometry receipt, the controller checks whether the raceline winding direction matches the car's heading; if the angular difference exceeds 90°, the raceline is reversed, preventing the car from chasing waypoints in the wrong direction.

2.4.2 Model Predictive Control (Implemented - performance is inconsistent, Not Demonstrated). An MPC controller was implemented using a kinematic bicycle model propagated over a 10-step horizon at $\Delta t = 0.05$ s. The control vector $\mathbf{u} = [\delta_0, a_0, \ldots, \delta_9, a_9]$ is optimized by `scipy` SLSQP at each timestep. The cost function penalizes cross-track error (CTE), heading error (e_ψ), speed error, steering magnitude, acceleration, and smoothness of both control channels:

$$\begin{aligned} J = \sum_{k=0}^{H-1} \Big[& W_{\text{cte}}\, \text{cte}_k^2 + W_{e_\psi}\, e_{\psi,k}^2 + W_v\, e_{v,k}^2 \\ & + W_\delta\, \delta_k^2 + W_a\, a_k^2 \\ & + W_{\Delta\delta}(\delta_k - \delta_{k-1})^2 + W_{\Delta a}(a_k - a_{k-1})^2 \Big]. \end{aligned} \tag{15}$$

The kinematic bicycle prediction is:

$$x_{k+1} = x_k + v_{\text{mid}} \cos\psi_k\, \Delta t, \tag{16}$$

$$y_{k+1} = y_k + v_{\text{mid}} \sin\psi_k\, \Delta t, \tag{17}$$

$$\psi_{k+1} = \psi_k + \tfrac{v_{\text{mid}}}{L_{\text{wb}}} \tan\delta_k\, \Delta t, \tag{18}$$

$$v_{k+1} = \text{clip}(v_k + a_k\, \Delta t,\ v_{\min},\ v_{\max}), \tag{19}$$

where $v_{\text{mid}} = \frac{1}{2}(v_k + v_{k+1})$ and the solution is warm-started from the previous optimized sequence. MPC produced correct steering in simulation but required additional tuning for hardware deployment, which was not completed within the project timeline.

2.5 Safety Node

A LiDAR-based safety node (`safety_node_lidar.py`) sits between `/drive_req` (planner output) and `/drive` (actuator input). It computes the minimum Time-To-Collision (TTC) across all forward LiDAR beams within a ±70° cone:

$$\text{TTC}_i = \frac{r_i}{\max(v_{\text{forward}} \cos\theta_i,\ \epsilon)}. \tag{20}$$

Four discrete braking levels are triggered by TTC thresholds (full stop at TTC < 0.7 s, hard brake at < 0.8 s, medium at < 0.9 s, soft at < 1.0 s). A minimum distance threshold of 0.2 m triggers an unconditional full stop. Hysteresis prevents oscillation: the brake level can only decrease once the nearest obstacle exceeds 0.35 m clearance.

3 Results and Validation

3.1 Mapping Quality

Mapping runs were conducted in both the F1TENTH simulator and on the physical track. In simulation, the node consistently accumulated clean centerline maps with few outlier jumps, owing to the near-perfect odometry available from the simulated environment. On the physical track, the car successfully completed a lap and produced a usable `track.npy`, with odometry drift remaining

within acceptable bounds—a result that was not universally achieved by peer teams, several of whom reported severe drift or map corruption when operating without external SLAM corrections.

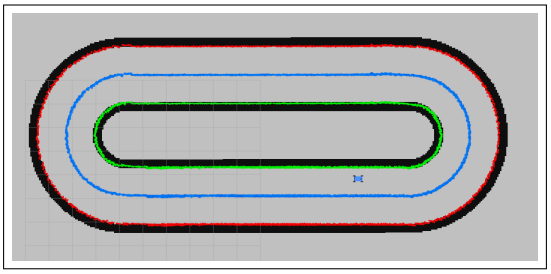

Fig. 2. Simulator mapping output. Blue dashed = centerline, green = left wall, red = right wall.

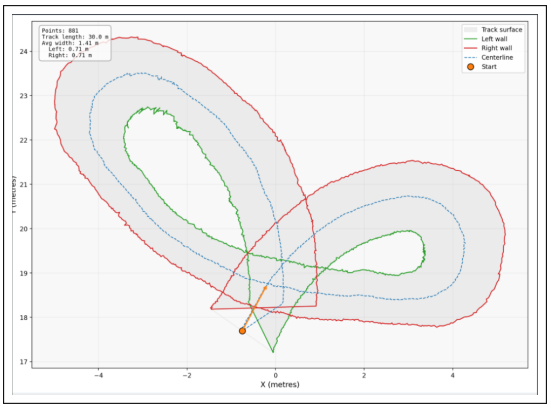

Fig. 3. Physical track mapping output. Mild odometry drift is visible near the start/end loop but the overall track shape is preserved.

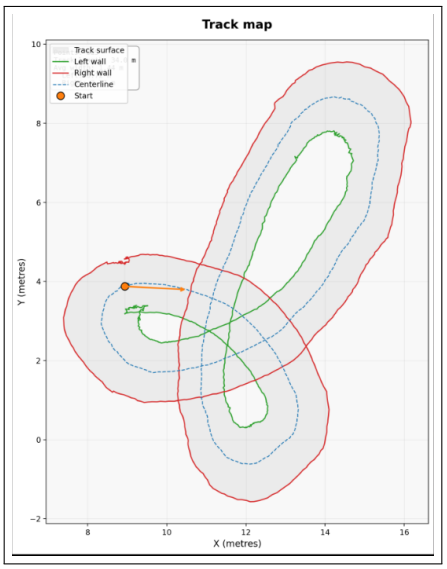

Fig. 4. Example of mapping with significant odometry drift on the physical track, illustrating the challenge addressed by careful tuning.

3.2 Raceline Optimization

The optimizer was run on both simulator and physical track data. The resulting racelines exhibit the characteristic outside-inside-outside profile of a proper racing line on curved sections, while the speed profile assigns higher velocities on straights and reduces speed through corners as expected from the curvature formula.

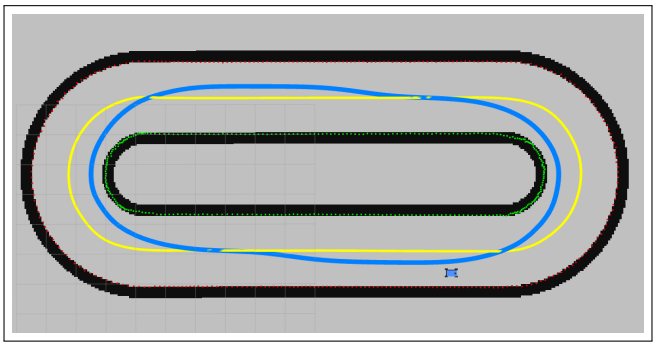

Fig. 5. Optimized raceline (blue) overlaid on the mapped track in the simulator. The line cuts the apex of corners and runs wide on straights.

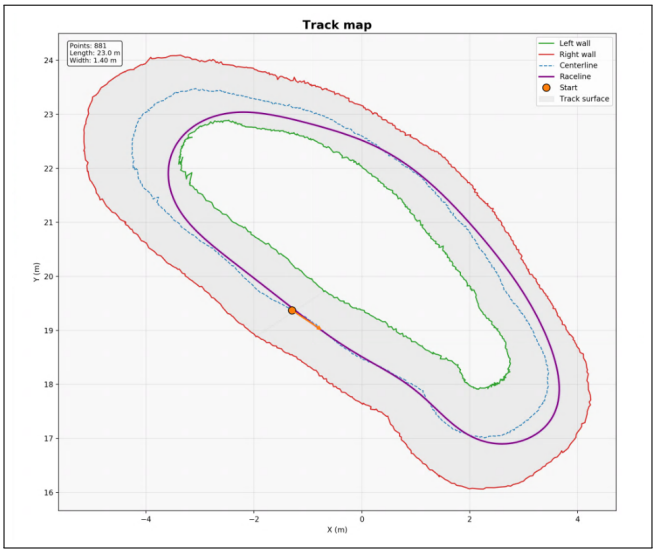

Fig. 6. Raceline computed from the physical track map. Despite mild drift in the input data, the optimizer produces a smooth, constraint-respecting line.

3.3 Pure Pursuit Tracking (Simulator)

Pure Pursuit was tested in the simulator with the optimized raceline as the reference. The car successfully completed multiple laps with stable tracking behavior at speeds between 1.5 m/s and 4.0 m/s. The controller required no manual tuning beyond the lookahead distance parameter.

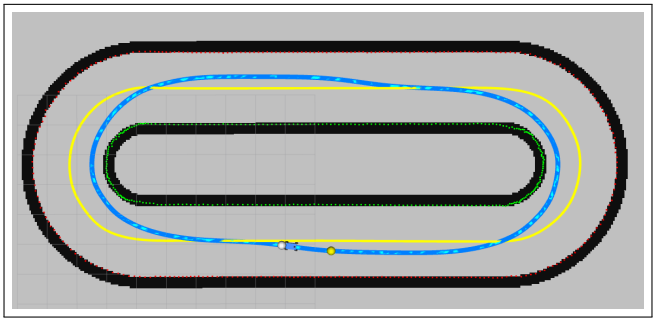

Fig. 7. Simulator view of Pure Pursuit tracking the optimized raceline. Yellow sphere = lookahead goal, cyan line = raceline.

4 Challenges

4.1 Odometry Drift on Hardware

Wheel odometry on the physical F1TENTH car accumulates error from wheel slip, especially in sharp corners. This drift distorts the coordinate frame within which wall points are recorded, causing the reconstructed track to appear skewed or disconnected. We mitigated this by tuning the wall-following speed to reduce slip, applying aggressive width and jump filtering in the optimizer, and capping the track to a single lap to prevent drift accumulation over multiple laps. Teams that relied solely on raw odometry without these filters experienced severe map corruption.

4.2 Simulator vs. Hardware Discrepancy

The simulator provides near-perfect odometry and deterministic LiDAR responses, producing maps that require almost no post-processing. The physical car, by contrast, exhibits sensor noise, latency, and mechanical tolerances that require all filtering parameters to be retuned. The `SIMULATOR` flag in each node switches topic names (e.g., `/ego_racecar/odom` vs. `/odom`) and enables or disables certain behaviors, but the fundamental algorithmic differences—particularly in LiDAR noise and odometry accuracy—required careful empirical parameter selection for hardware deployment.

4.3 Localization During Racing

Once the map is built and the raceline computed, the racing node must determine where the car is relative to the pre-computed waypoints. In the absence of a global localization module, we rely on odometry to match the car's starting position to the nearest waypoint at initialization. If the car is placed at a different start position, the automatic raceline-flip logic handles heading mismatches gracefully in most cases.

4.4 MPC Convergence and Compute Time

The SLSQP optimizer used for MPC must solve a 20-variable nonlinear program (10-step horizon, 2 controls per step) within the 50 ms control loop period. In simulation this was achievable with 80 iterations, but on the physical car the additional ROS overhead occasionally caused deadline misses, producing jittery control. This, combined with limited hardware testing time, led to MPC being excluded from the final demonstration. Warm-starting from the previous solution and reducing the horizon are natural remedies we would pursue in continued work.

4.5 Limited Hardware Time

Access to the physical car was constrained by shared lab resources and scheduling. As a result, MPC was not extensively tested on hardware, and Pure Pursuit received only partial on-track validation. This limited our ability to measure lap times and compare controllers empirically.

5 Individual Contributions

Table 1 summarizes each team member's primary responsibilities. All members contributed to integration, testing, and report writing.

Table 1. Individual contributions to the project.

Team Member	Contributions
Warotpete	Simulator map design and testing environments; improvements to mapping and optimizer pipelines; project documentation and setup instructions; debugging, parameter tuning, and integration across simulation and hardware.
Can	Raceline optimizer (`optimizer.py`): centerline optimisation logic, spline smoothing, boundary margin tuning, regularisation design, and track ordering/refinement to preserve driving consistency.
Ian	Initial mapping pipeline and centerline extraction; simulation validation; debugging mapping/runtime issues; hardware access management (SSH) and execution of real-world demonstrations.
Wit	Model Predictive Control (MPC): initial experimental implementation using centerline as a baseline; led debugging and integration across mapping, optimizer, and control pipeline; supported integration of MPC with the raceline optimizer and iterative tuning; simulator data collection and evaluation; contributions to mapping improvements and system-level stability.
Miguel	Mapping enhancements using scan-based techniques; wall-following improvements; contributions to optimizer tuning and raceline generation attempts; development of visualization tools for debugging and analysis.

6 Conclusions

This project demonstrates a complete autonomous racing pipeline from raw sensor data to tracked raceline, implemented without reliance on a third-party SLAM package. The key takeaways are:

- **Custom LiDAR mapping is feasible on F1TENTH hardware.** By restricting wall measurement to near-perpendicular beams and applying spatial deduplication, we obtained usable track maps on the physical car with manageable odometry drift—a result not universally achieved by peer teams.
- **Minimum-curvature optimization produces qualitatively correct racing lines.** The QP formulation with corridor constraints and a raceline-seed bias yields outside-inside-outside profiles consistent with standard racing theory.
- **Pure Pursuit is a robust baseline.** The lookahead-based controller tracked the optimized raceline reliably in simulation with no additional tuning, providing a solid validation of the upstream pipeline.
- **MPC offers higher performance potential but requires more development time.** The kinematic bicycle MPC implementation is correct in principle; the primary remaining challenges are computational budgeting and hardware-specific parameter tuning.

6.1 Future Work

- **MPC hardware validation:** reduce horizon length or use a faster QP solver (e.g., OSQP) to meet the 50 ms deadline.
- **Localization integration:** fuse odometry with ICP scan matching or a particle filter to reduce drift and enable reliable multi-lap racing.
- **Iterative raceline refinement:** update the raceline after each lap using lap time data or controller error signals (iterative learning control).
- **Dynamic obstacle avoidance:** extend corridor constraints to account for other vehicles detected by LiDAR during head-to-head racing.

References

[1] Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp, and Boris Lohmann. 2020. Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics* 58, 10 (2020), 1497–1527.

[2] University of Waterloo CL2 Lab. 2023. Raceline Optimization for F1TENTH. https://github.com/CL2-UWaterloo/Raceline-Optimization

[3] Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. 2015. Kinematic and dynamic vehicle models for autonomous driving control design. In *Proc. IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1094–1099.

[4] R. Craig Coulter. 1992. Implementation of the Pure Pursuit Path Tracking Algorithm. Technical Report CMU-RI-TR-92-01. Carnegie Mellon University Robotics Institute.

[5] Steve Macenski, Francisco Martin, Ruffin White, and Jonatan Ginés Clavero. 2020. The Marathon 2: A Navigation System. In *Proc. IEEE/RSJ IROS*.

[6] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic Robotics*. MIT Press, Cambridge, MA.