Lab #6 Report: Path Planning and Following for a Model Racecar

Team #5

Rohan Das
Kelsey Fontenot
Yifan Kang
Dante Suazo
He'yun Zhi

6.4200 Robotics: Science & Systems

April 24, 2026

1 Introduction (Yifan)

Bridging the gap between perception and physical movement is the primary domain of motion and path planning. The essential process of computing a continuous collision-free trajectory from an initial configuration to a target destination is what enables a robot to actively navigate its world. Even if an autonomous system boasts perfect localization and high-performance control mechanisms, it remains practically immobilized without a robust planning stack. This dependency is critical in modern robotics, where deploying unsafe or inefficient routes can swiftly result in mission failure, catastrophic collisions, and an inability to operate within unpredictable, dynamic spaces.

In this lab, we implemented and evaluated a comprehensive path planning system, serving as the critical decision-making module required to guide a robot through obstacle-dense environments. The system must reliably compute viable trajectories, balancing the need for optimal, shortest-distance paths against computational constraints and the complexity of high-dimensional configuration spaces.

Our technical approach centers on the comparative analysis of two distinct methodologies: Grid-based A* search and Optimized Rapidly-exploring Random Trees (RRT*). For structured, low-dimensional environments, we utilize Grid-based A*, a deterministic, resolution-complete algorithm that evaluates

discrete states based on a cost function $f(n) = g(n) + h(n)$ to guarantee an optimal path. To address high-dimensional, continuous spaces where grid methods succumb to the curse of dimensionality, we implemented RRT*. This sampling-based algorithm probabilistically builds a tree of feasible trajectories by drawing random samples x_{rand} from the configuration space, extending the tree from the nearest existing node x_{near} toward the sample, and rewiring by eliminating extraneous nodes.

We validated these algorithms side-by-side across a series of standard robotic navigation environments, demonstrating the underlying mechanics and computational trade-offs of each system. Our results establish that while A* excels at finding optimal routes in discretized grids, our RRT* implementation provides a highly adaptable, rapid alternative for exploring complex, continuous spaces.

2 Localization Quantitative Results

2.1 Localization Performance on the Real Car (Yifan)

To evaluate the particle filter's localization accuracy, we compared its pose estimate against raw VESC wheel odometry over a 75-second driving run in the Stata basement. The particle filter estimate was extracted from the `map` → `base_link` transform broadcast to `/tf`, while VESC odometry was taken from the `odom` → `base_link` transform. Since these two transforms live in different reference frames — the PF operates in the global map frame while the VESC integrates displacement in its local odom frame — a naive subtraction would conflate frame misalignment with genuine drift. We therefore applied a continuous heading correction: at each timestep, the incremental VESC displacement was rotated by the instantaneous heading difference between the PF and VESC yaw estimates before being accumulated. This isolates the purely translational component of odometry error, independent of frame initialization artifacts.

The resulting divergence between the PF trajectory and heading-corrected VESC odometry is shown in Figure 1. The mean positional error is 0.609m, with a peak of approximately 3m occurring around $t = 30$s, coinciding with a sharp turn where wheel slip is most pronounced. Outside of dynamic maneuvers, the error remains consistently below 1m, demonstrating that the particle filter successfully anchors the pose estimate against LiDAR-observed map features even as the underlying odometry drifts.

The trajectory comparison (Figure 2) further illustrates this: the VESC odometry path (blue) diverges visibly from the PF path (red) at the bottom cluster where the car reverses direction, and again near the top of the run. The PF trajectory, by contrast, traces a geometrically consistent path that aligns with the known map structure.

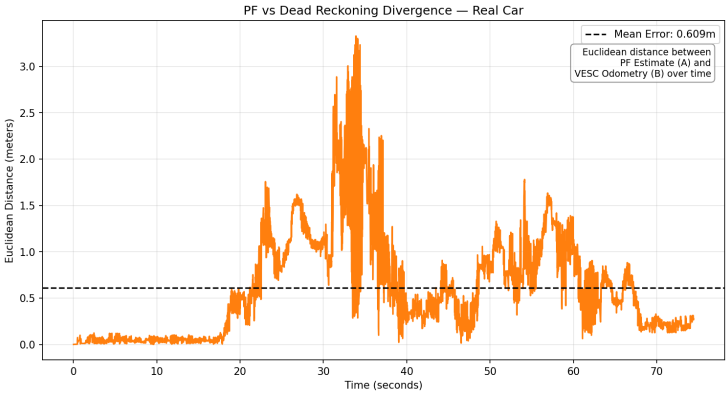

Figure 1: Divergence between the particle filter trajectory and heading-corrected VESC odometry over the 75-second run.

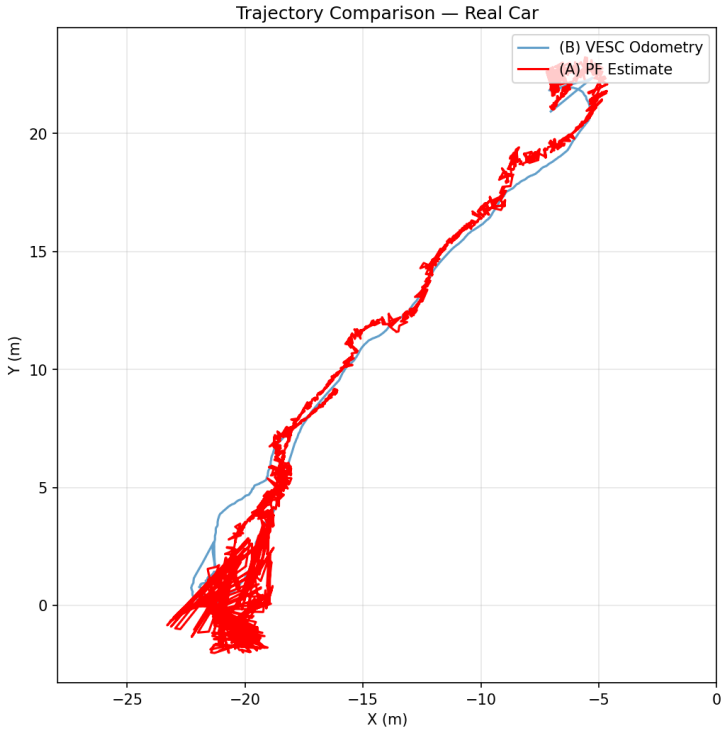

Figure 2: Trajectory comparison showing the VESC odometry path (blue) diverging from the PF path (red) during direction changes.

2.2 Limitations of VESC Odometry as Ground Truth (Yifan)

While this comparison effectively demonstrates that the particle filter outperforms dead reckoning, using VESC odometry as the baseline carries an important caveat: it is not true ground truth. The VESC measures wheel rotation, which introduces several sources of systematic error — tire slip on smooth floors, encoder quantization, and mechanical backlash — that accumulate monotonically over time. As a result, the divergence metric reported here measures how much the PF differs from dead reckoning, not how close the PF is to the car's true physical position.

A rigorous ground truth would require an external reference system such as a motion capture array or a high-precision RTK-GPS, neither of which was available in this setting. In the absence of such a system, the particle filter's self-consistency — the degree to which the laser scan aligns with the known map at the estimated pose — serves as a proxy for accuracy. Qualitatively, we observed strong scan-to-map alignment throughout the run, suggesting the PF estimate is reliable even though it cannot be independently verified against a true ground truth.

2.3 Simulation vs Real (Kelsey)

Simulation testing showed off a well working particle filter with large spikes correlating to turns. We found that our particle filter had a low mean error of 0.308m. In real life, we observed the mean error to be doubled as in Figure 2 and looked at other metrics to narrow down the cause. The number of effective particles over the course of the 75-second run is shown in Figure 3 and shows that the particle filter remains stable. However, several sharp drops that indicate temporary localization failures. These failures, particularly at the 32 second mark are in line with the large spike in mean error in Figure 2 at the same time.

To investigate this further, we looked at sensor model innovation over time. Innovation refers to the difference between what the sensors are reading as compared what our particle filter is predicting. Low innovation means the model is doing well, and high innovation means there is a mismatch between the prediction and the sensors, in this case our LIDAR readings. Figure 4 reveals that at the 32 second mark, innovation spikes from low to high, which matches with previous plots. During the run, this correlated when the robot was being driving in fast circles in an area of the map that did not directly match real life. There were several large boxes and bikes that the LIDAR picked up that weren't on the provided map that the model was predicting. In combination with fast

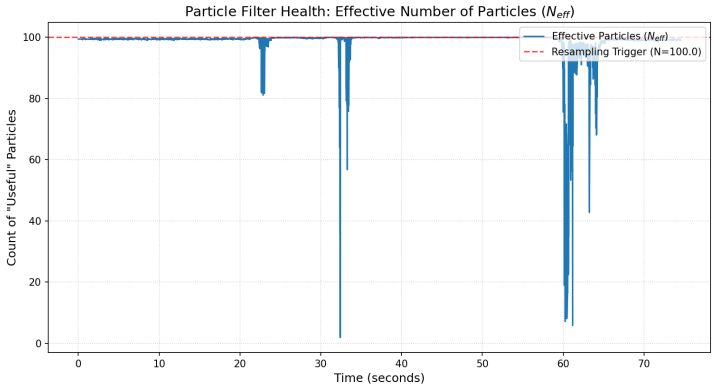

Figure 3: Stable particle filter with some drops that correspond to localization failures.

erratic driving, the model struggled with localization in this area. The particle filter eventually recovered when it drove back into the hallway around the 60 second mark and the improvement can be seen at that mark on the rest of the plots.

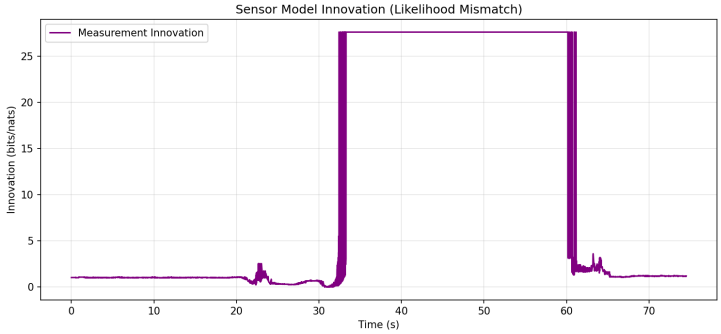

Figure 4: Model prediction mismatch over course of run gives indication as to why mean error has large spikes.

2.4 Runtime Performance (Dante)

Our particle filter achieved a sustained update rate of approximately 71 Hz on the Jetson Orin with 100 particles and 99 LiDAR beams, corresponding to a per-update latency of roughly 14 ms. This exceeds the 20 Hz real-time requirement by a factor of 3.5×. The dominant computational cost is the sensor model's ray-tracing step, which scales linearly with both particle count and beam count. The precomputed 201×201 lookup table eliminates per-beam

distribution computation, reducing the sensor model to array indexing and a single log-sum operation per particle. We did not conduct a systematic sweep of particle count versus update rate due to time constraints; however, the 3.5× headroom suggests the system could support approximately 350 particles before dropping below the real-time threshold, assuming linear scaling.

3 Localization Media

Please see our localization model visualized in gif form here and access the raw videos here.

4 Path Planning Technical Approach

4.1 Grid-Based Planner (Yifan)

While randomized planners are preferred for complex, high-dimensional spaces, Grid-based A* provides a deterministic, resolution-complete approach guaranteed to find the absolute shortest path in structured 2D environments. We implemented A* search directly on the provided simulator occupancy grid, which represents the Stata basement at a 0.05 m/cell resolution.

To guarantee safe vehicle clearance, we preprocess the map by morphologically dilating all obstacles (cells with occupancy > 50 or -1) by an 8-pixel radius (~ 0.4 m). Continuous world coordinates (x, y) are then accurately mapped to discrete grid pixels (u, v) via a rotational transform that accounts for the map origin's non-zero yaw.

The A* algorithm explores this inflated, 8-connected grid with the following process:

- Initialize a min-heap priority queue with the starting pose.
- While the queue is not empty, extract the node with the lowest total estimated cost $f(n) = g(n) + h(n)$.
  - **Accumulated cost (g):** Adds 1.0 for cardinal moves and $\sqrt{2}$ for diagonal moves.
  - **Heuristic (h):** The Euclidean distance to the goal. Because it never overestimates the true distance, the heuristic is admissible, guaranteeing an optimal solution.
- Expand to all valid neighbors. To prevent redundant computation, skip any neighbors where a path with an equal or lower g-value has already been recorded.
- If the goal node is expanded, backtrack through parent pointers to return the unique path.

By operating on the pre-inflated pixel grid, A* effectively computes an optimal, collision-free trajectory. The resulting discrete path is then transformed back into continuous world coordinates and published as a `PoseArray` to guide the vehicle.

4.2 Sampling-based Randomized Planners (Rohan)

The two main classes of randomized path planning algorithms are Probabilistic Roadmaps (PRMs) and Rapidly-exploring Random Trees (RRTs). PRMs rely on preprocessing to randomly build a graph, and then run a graph algorithm like A* search. The refined version, PRM*, is probabilistically complete and asymptotically optimal, meaning that within enough samples it finds a close-to-optimal solution with probability 1. However, it has a slow build time of $O(n^2)$ and is only preferrable to the A* approach when the graph construction is the main difficulty, such as in high-dimensional or complex spaces. Therefore, we elected to implement RRT*. We used the same map dilation approach as in our grid-based planner to prevent collision with obstacles.

RRT randomly expands a tree across thousands of iterations, with the following process:

- Initialize a tree with the starting pose.
- For n iterations:
  - Randomly sample a new point, which is the goal with probability $p_{goal\ bias}$.
  - Extend it into the tree, connecting it to the closest connectable node.
  - If the goal is in the tree, return the unique path.

This has $O(n \log n)$ runtime and is probabilistically complete, but doesn't guarantee an optimal solution. The refinement, RRT*, handles this by rewiring: when adding a new node, it iterates through all existing nodes within a distance d_{rewire} and checks if they can be reconnected to a closer node. This guarantees asymptotic optimality and allows for continued improvement after we first connect to the goal.
With 5000 iterations and with a 10% goal bias, RRT* demonstrates a qualitatively sound path across the entire Stata basement within a few seconds.

4.3 Pure Pursuit Controller (Dante)

Once a collision-free trajectory has been generated, the path-following module converts it into low-level steering commands using a Pure Pursuit controller. The controller runs at the odometry rate ($\sim$50 Hz) and produces an `AckermannDriveStamped` message at each tick.

Lookahead-point selection. At every pose update we find the point on the trajectory that the car should aim for — the *lookahead point* — in two stages:

1. **Nearest segment.** The car's pose is projected onto every piecewise-linear segment of the trajectory in a single vectorized NumPy operation. For segment i with endpoints $\mathbf{p}_1^{(i)}, \mathbf{p}_2^{(i)}$ and the car at $\mathbf{c}$,

$$t_i = \mathrm{clip}\left(\frac{(\mathbf{c} - \mathbf{p}_1^{(i)}) \cdot (\mathbf{p}_2^{(i)} - \mathbf{p}_1^{(i)})}{\|\mathbf{p}_2^{(i)} - \mathbf{p}_1^{(i)}\|^2}, 0, 1 \right),$$

   yields the closest projection on that segment; the segment with the smallest $\|\mathbf{c} - (\mathbf{p}_1^{(i)} + t_i(\mathbf{p}_2^{(i)} - \mathbf{p}_1^{(i)}))\|$ is selected as the nearest.

2. **Circle–segment intersection.** Starting from the nearest segment and walking forward, we intersect each segment with a circle of radius L_d (the lookahead distance) centred on the car. Solving the resulting quadratic yields up to two parameter values $t \in [0, 1]$; the larger valid root is chosen so the car always aims *ahead* of its current position along the path. On the segment containing the projection, intersections with $t < t_i$ are rejected to prevent the controller from targeting a point behind itself.

If no intersection exists (e.g. the car is further than L_d off-path near the goal), the controller falls back to the final waypoint.

Steering law. The lookahead point is expressed in the car's body frame as (x_ℓ, y_ℓ). The Pure Pursuit geometry gives a path curvature of $\kappa = 2y_\ell / L_d^2$, which we convert to a steering angle using the bicycle model:

$$\delta = \arctan(L\kappa), \qquad \delta \in [-\delta_{\max}, \delta_{\max}],$$

where $L = 0.325\,\mathrm{m}$ is the Racecar wheelbase and $\delta_{\max} = 0.34\,\mathrm{rad}$ ($\sim 19.5^\circ$) is the platform's mechanical steering limit.

Edge cases. Two behaviors proved important on the real car:

- *Lookahead behind the car.* If $x_\ell \leq 0$, the target has fallen behind the vehicle (e.g. after a spin-out or pose jump). We then reverse at half speed with mirrored steering $\arctan 2(-y_\ell, -x_\ell)$, mirroring the parking-controller behavior from Lab 3.

- *Goal arrival.* When the car comes within $0.5\,\mathrm{m}$ of the final waypoint, the controller publishes zero speed and zero steering to terminate the run cleanly.

Parameter tuning. We tuned the two dominant parameters — lookahead distance L_d and forward speed v — by sweeping in simulation and then validating on the car. A lookahead of $L_d = 1.5\,\text{m}$ was chosen as a compromise between stability on long hallway segments (where larger L_d damps oscillation) and responsiveness at tight corners (where a smaller L_d is required to track curvature). The speed was fixed at $v = 1.0\,\text{m/s}$; above roughly $1.5\,\text{m/s}$ the fixed lookahead produced noticeable overshoot at corners, and below $0.7\,\text{m/s}$ the mean cross-track error did not meaningfully improve. Table 1 lists the final values.

Parameter	**Value**	**Rationale**
Lookahead L_d	$1.5\,\text{m}$	Balances corner tracking vs. hallway stability
Speed v	$1.0\,\text{m/s}$	Below overshoot threshold; real-world headroom
Wheelbase L	$0.325\,\text{m}$	RACECAR platform geometry
Max steering $\delta_{\max}$	$0.34\,\text{rad}$	Mechanical steering limit
Goal tolerance	$0.5\,\text{m}$	Stops cleanly without overshooting past waypoint
Reverse speed	$-0.5\,v$	Used when lookahead falls behind the car

Table 1: Pure Pursuit parameters used for both simulation and real-world runs.

5 Path Planning Quantitative Results (Kelsey)

5.1 Simulation Testing

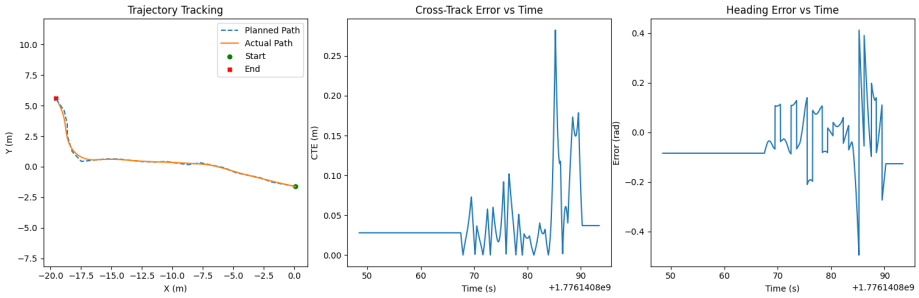

Figure 5: **Trajectory tracking performance with error metrics.** Left: Planned path (dashed) vs. actual robot trajectory (solid), with start (green) and end (red) markers. Middle: Cross-track error over time. Right: Heading error over time.

In simulation, the integrated path planning and following approach performed well, maintaining low-cross track error and heading error. Across three runs of paths longer than 20 m, the mean cross-track error was 0.0451 m and the mean heading error was 0.146 rad. The spikes that occur for the cross-track and heading error are due to an "imperfect" path follower. Since our car utilizes a

Pure Pursuit controller with a lookahead distance of 1.5 m, the planned path isn't always the path the car takes as seen by Figure 5.

5.2 Real World Testing

In real world testing, our path planner and follower continued to perform well. While the first tests in open space seemed promising, the car initially struggled to go around tight corners because of the overprotectiveness of the safety controller we implemented. This meant the car would stop extremely close to walls or pillars and stop motion entirely. However, after relaxing the constraints of the safety controllers, the car was able to maneuver smoothly down the hallways of Stata basement and take the tight turns in stride.

6 Path Planning Media

Please see our path planning performance visualized in gif form here and access the raw videos here.

7 Conclusion (Kelsey)

In this lab, we developed and implemented a localization system for our racecar using a particle filter. Our particle filter used a combined motion model and sensor model to estimate the racecar's position on the map.
In real life, our results showed that our particle filter performed best within the hallway outside the classroom because that section was most similar to the map. If there were large objects such as bikes or boxes in real life that did not appear in the map, our particle filter would struggle because it was not expecting those objects there.

Building upon our localization system, we created path planning and path following algorithms to drive the car between two points on a map. We explored both grid-based A* search and RRT*, ultimately deciding to implement RRT* on our racecar as well as a Pure Pursuit controller to move around the map.

Overall, our lab showed success in our particle filter both in simulation and real life, though there are still limitations to its performance in real life especially with erratic behavior. Further, the path planner performed well with low mean cross-track and heading error both in simulation and real world experiments. To improve further for the final challenge, we will actively look at stopping distance and ensure the robot is going to the desired position within 0.25 m.

8 Lessons Learned

8.1 Kelsey

In this lab, I continued to build on system integration skills, especially the transition from simulation to real world deployment. Through this, I learned to test extensively and not immediately rule our algorithms as not working. For instance, at the start our path follower struggled in real life to turn some corners where it was able to follow in simulation. By lowering the thresholds of the safety controller, car was able to follow paths around corners much better.

8.2 Yifan

In this lab, I learned that it is important to understand the theory behind each algorithms we are implementing. With a good understanding, we know what parameters to adjust when we see unexpected outcomes.

8.3 Dante

A major takeaway from this lab was that infrastructure decisions can consume as much debugging time as the algorithm itself. Docker volume mounts, TF frame naming, and RViz QoS settings each produced subtle failures that were difficult to diagnose remotely over an unstable WiFi link to the car. Documenting these operational details turned out to be just as important as documenting the code, because without a clear record of what worked, every reconnection risked repeating the same mistakes.

8.4 He'yun

I learned that getting started with planning evaluation metrics and simulation testing early paves the way for smooth real world testing. Furthermore, clever thinking about how to tweak and reuse previous code speeds things up, such as reusing my parking controller code for the behind-the-car case. However, it is necessary to ensure a conceptual understanding of the high-level fundamentals before debugging, testing, and evaluating. Our initial evaluation of A* versus RRT* misidentified a shared map dilation issue across both implementations, resulting in a misinformed comparative analysis.

8.5 Rohan

Throughout this lab, I learned the importance of communication and planning. In spite of joining this team midway through Lab 6, due to quick communication from my teammates, we were able to adapt easily. Furthermore, I worked on technical implementations of the randomized planner, where due to clear

communication I knew to reuse existing implementations for obstacle dilation. Also, because we set up a clear timeline, I was able to draft and debug my implementation in sync with the simulation test schedule.