Nonholonomic Control

EECS C106B Project 2

Osher Lerner Will Panitch Enya Xing

I. Abstract

We implement three different methods and modalities of planning algorithm in order to compare their results on a number of simulated and real-world tasks. These planners include a trajectory cost-optimization planner, a sampling-based planner (RRT), and a nonholonomic sinusoidal steering algorithm (steering with sinusoids). We test our planners on three simple point-to-point navigation tasks (a forward motion, sideways "parallel park" path, and a three-point turn), as well as two obstacle avoidance environments. We then provide visualizations of each planner's concrete performance on each task, demonstrating each model's approach to solving an environment. Finally, we discuss and evaluate each of the planners' performance in order to qualitatively understand each planner's strengths and weaknesses. This analysis allows us to demonstrate the applicability of each individual algorithmic approach to a variety of different motion planning tasks, and more intelligently choose a planner for any specific task.

II. Methods

A. Sinusoidal Planner

Our implementation of the sinusoid planner is based on the classic planner intruduced by Murray and Sastry in 1993 [2]. This planner attempts to create sinusoidal or parametric paths that align our state variables with the goal one at a time. As such, the paths that it created tented to be chains of smooth curves, with a high degree of volatility based on starting and ending states. To plan a path using this method, we first convert our state variables from x, y, θ, ϕ to a new state space in which the dynamics take on a chained form:

$$\begin{aligned}\dot{q}_1 &= u_1\\ \dot{q}_2 &= u_2\\ \dot{q}_3 &= f(q_2)u_1\\ \dot{q}_4 &= g(q_3)u_1\end{aligned}$$

This allows us to take advantage of sinusoidal inputs of the form

$$\begin{aligned}u_1 &= a_1 \sin(\omega_1 t)\\ u_2 &= a_2 \cos(\omega_2 t)\end{aligned}$$

using an appropriate choice of ω_1 and ω_2 to construct paths that vary a variable q_i without changing previous q_j for $j < i$. Thus by tuning the amplitudes a_1 and a_2 we may exactly control our resulting q_i, and repeat the process for each i sequentially to reach any desired state $\vec{q}$.

Particularly, we used the canonical model

$$\begin{aligned}\dot{x} &= v_1, & v_1 &= \cos(\theta)u_1\\ \dot{\phi} &= v_2, & v_2 &= u_2\\ \dot{\alpha} &= \tfrac{1}{l}\tan(\phi)v_1, & \alpha &= \sin(\theta)\\ \dot{y} &= \tfrac{\alpha}{\sqrt{1-\alpha^2}}v_1\end{aligned}$$

however, this model maps states with opposite headings to the same α and creates singularities when $\alpha = 1$ (when $\theta = \pm 90^\circ$). We attempted searching for other equivalent chained form dynamics but found none without singularities. Instead, we chose to dynamically adjust our chosen coordinates to maneuver these singularities away from the current trajectory. We did this by reflecting and rotating the coordinates of the canonical model. Reflecting about the $\theta = 45^\circ$ (the $x = y$ line), we get $x' = y$, $y' = x$, $\theta' = 90^\circ - \theta$, and $\phi' = -\phi$, we reproduce the alternate model

$$\begin{aligned}\dot{y} = \dot{x}' &= v_1, & v_1 &= \cos(\theta')u_1 = \sin(\theta)u_1\\ \dot{\phi}' &= v_2, & v_2 &= -u_2\\ \dot{\alpha}' &= \tfrac{1}{l}\tan(\phi')v_1, & \alpha' &= \sin(\theta') = \cos(\theta)\\ \dot{x} = \dot{y}' &= \tfrac{\alpha'}{\sqrt{1-\alpha'^2}}v_1\end{aligned}$$

Similarly, we can rotate our coordinates by some angle ψ to get $x' = \cos(\psi)x - \sin(\psi)y$, $y' = \cos(\psi)y + \sin(\psi)x$, $\theta' = \theta$, and $\phi' = \phi$. This allows us to adjust our antipodal θ singularities to any opposite angles, and we select these to be maximally beyond our initial and goal headings. In cases where our initial and final headings are exactly 180° apart, the singularities are still unavoidable. In these cases, we construct an intermediate goal with a heading halfway to the final goal, and plan a path to this state using the canonical model and from this state to the goal using the mirrored alternate model.

To compute the amplitudes a_1, a_2 needed to reach the goal state, we integrate the dynamics to find an expression for the change in state variables as a function of $a1$ and $a2$, then solve for the amplitudes. We analytically simplify this expression by expanding the Fourier Series of the dynamics, in which terms multiplying sinusoids of different frequencies integrate to 0 under our aptly chosen bounds (whole periods of the sinusoids). To numerically search solutions, we also take advantage of the fact that the change in state variables is monotonic in a_1 to solve for a_1 using binary search (setting a_2 to the largest permitted amplitude).

B. Optimization-based Planner

Next, we wrote an planner to find a trajectory by preforming constrained optimization on a quadratic cost function. We discretize the trajectory $(q(t), u(t))$ into N time-steps δt apart s.t. $q_i = q(i * \delta t)$ and $u_i = u(i * \delta t)$. We encode our state and input limits, dynamics, start and goal states, and obstacles as constraints on these variables, and construct a quadratic cost function to encourage the trajectory to reach the goal state as fast as possible using minimal input:

$$
\begin{aligned}
J(q,u) = & \sum_{i=1}^{N} \Big((q_i - q_{goal})^T Q \, (q_i - q_{\text{goal}}) \\
& + u_i^T R u_i + (q_{N+1} - q_{\text{goal}})^T P \, (q_{N+1} - q_{\text{goal}}) \Big) \\
q_i \geq & q_{\min}, \forall i \\
q_i \leq & q_{\max}, \forall i \\
u_i \geq & u_{\min}, \forall i \\
u_i \leq & u_{\max}, \forall i \\
q_{i+1} = & F(q_i, u_i), \forall i \leq N \\
\text{obs}_{j_r}^2 \leq & (x_i - \text{obs}_{j_x})^2 + \left(y_i - \text{obs}_{j_y}\right)^2, \forall i, j \\
q_1 = & q_{\text{start}} \\
q_{N+1} = & q_{\text{goal}}
\end{aligned}
$$

We solve this nonlinear programming problem using CasADI as the optimization backend. Regarding N and δt, we attempted to modify these values to improve optimization-based planning, but these modifications did not prove to be beneficial. Our conclusion was to leave N and δt as is. In general, poor results would occur due to low N and high δt, but a high N and low δt mostly just increase the dimensionality of the optimization problem (and thus the computation time). If we had been intersecting obstacles or other constraints, we would need higher fidelity to enforce constraints more frequently (low δt). Overs N increasing N can allow

C. Rapidly-Exploring Random Tree Planner

Our approach to RRT involved specific choices for distance heuristic, random sampling and local planning.

For the distance function, we ignored ϕ since it could be directly controlled once the goal state was reached. We initially used a Euclidean distance function of (x, y, θ) with modifications to account for the periodicity of the heading angle as in [3]. However, we found the RRT would succeed in reaching the correct x and y but struggle to match θ to the goal configuration. We fixed this by independently weighting each dimension x, y, and θ (and normalizing by the magnitude of the weights), but our tree would still occasionally get stuck at local minimums parallel to the target configuration. Thus we devised a metric to separately compute and weight the distance along the car's heading and the distance perpendicular to it.

For our starting configuration $c1$ at $(x_1, y_1, \theta_1, \phi_1)$ and $c2$ at $(x_2, y_2, \theta_2, \phi_2)$, we calculate $\Delta\vec{x} = (x_1 - x_2, y_1 - y_2)$ and $\overline{\theta} = \frac{\theta_1 + \theta_2}{2}$. We denote the perpendicular direction $\hat{e}_\perp$ between the configurations as $(\cos(\overline{\theta}), \sin(\overline{\theta}))$. Similarly, the parallel direction $\hat{e}_\parallel$ can be written as $(-\sin(\overline{\theta}), \cos(\overline{\theta}))$. We then use these two calculations to find the parallel distance $d_\parallel = |\Delta\vec{x} \cdot \hat{e}_\parallel|$ and perpendicular distance $d_\perp = |\Delta\vec{x} \cdot \hat{e}_\perp|$.

To combat differences in the periodic heading angle, we construct a term $t = (\theta_1(\mod 2\pi) - \theta_2(\mod 2\pi))^2$

We now utilize $d_\parallel$ and $d_\perp$ to construct our distance function with w_1, w_2, w_3 as the weights for the parallel distance, perpendicular distance, and θ respectively. Our distance function follows as

$$\text{distance} = \frac{\sqrt{w_1 d_\parallel + w_2 d_\perp + w_3 t}}{\sqrt{w_1 + w_2 + w_3}}$$

In the random sampling aspect, we decided to utilize goal zoom. With probability p we chose to sample in a circle centered around the goal g with a radius equivalent to $\min_i \text{distance}(x_i, g)$ for points $x_1, ...x_n$ of configurations in our graph. We ended up with $p = 0.5$ after some tuning.

We construct a diverse set of primitives defined by constant and sinusoidal inputs u_1, u_2 with varying magnitudes. After sampling a target configuration, we select the closest node in our graph and then find the best primitive local plan from it.

The local plan is computed by searching through our list of constructed primitives. For each primitive, we simulate its action schedule starting from configuration $c1$, then measure our custom distance function from the resulting configuration and the sampled goal. We use the first 1 second of the plan from the primitive with the minimum distance to generate a new node for our tree.

III. Experimental Results

A. Simple Motion Manipulation Task

For the simple motion task, we desire a path from our starting point of (1, 1, 0, 0) to (2, 1.3, 0.7, 0) to generalize any simple movement mostly in the forward direction with changes in orientation. We see in Fig.1a that our optimization-based planner is able to plan a straightforward path to reach the desired goal state. In Fig.1b our RRT planner is able to create a plan to the goal state with relative ease, with the RRT graph's edge paths illustrated in orange. Our sinusoidal planner shown in Fig.1c takes a larger footprint with part of the trajectory going to (2, 2, 0, 0), further than any other planner.

B. Parallel Park Motion Manipulation Task

This motion task challenges us to plan a motion in the y direction while keeping orientation and the x state the same. In real-world applications, this is a maneuver that many human drivers cannot claim mastery and fail to execute with ease. We have a starting point of (1, 1, 0, 0) and a goal state of (1, 3, 0, 0). Our optimization-based planner in Fig.2a leads us with a plan that moves in the y-direction upwards and then readjusts. The RRT planner in Fig.2b shows us the many edge paths that exist in the RRT graph. The generated result is fairly complicated, and creates a sideways M type movement. It makes a two adjustments in a shape that is similar to the optimization-based and sinusoidal planner. The RRT planner

travels the furthest in the x-direction than any other planner. The sinusoidal planner Fig.2 creates a path similar to the optimization-based planner, but has an even and horizontally symmetric path. An adjustment is first made toward the x-direction and then going further in the y-direction.

C. Point Turn Motion Manipulation Task

This motion manipulation task generalizes point turn as a change in orientation while position is constant. We begin at (1, 1, 0, 0) and have a desired goal state of (1, 1, π, 0). The optimization-based planner in Fig.3a creates a series of three curved edges that compose a triangle. Our RRT planner in Fig.3b creates a path similar to that of the optimization-based planner, but less even and with a larger maximum distance traveled in both the x and y-directions. We see through the RRT edge paths in orange that the graph has sampled points with values in the y-dimension up to 4, which occurs through randomness.

D. Navigation Tasks

In the first navigation task to get from (1, 1, 0, 0) to (9, 9, 0, 0) through two large obstacles, the optimization-based planner was able to reliably produce the same result between the two obstacles. RRT had a more difficult time as it occasionally generated a path that went above both obstacles.

In the second navigation task, the optimization-based planner is able to create a path of minimal distance expenditure. The RRT planner does more exploration and has sampled points between different obstacles.

IV. Discussion

A. Performance and Comparison

The three planners each had a number of strengths and weaknesses that differentiated their performance on our different tasks. The sinusoidal planner, for example, tended to create very smooth paths in those situations in which only the x or y would vary. This can be partially explained by the chaining method by which this planner develops its plans. By attempting to vary only one dimension of our state space during each planning phase, it is able to create relatively smooth cartesian motions that can accurately reach a desired goal. Additionally, by constraining the planner to offer up sinusoidal path segments, we gained additional smoothness during path execution. However, this planner struggled with variation in the angle of its planning, often resulting in plans with jaggedness or strange behavior near its singularities. To counter this, we enabled the method to switch between different expressions of the dynamics model in order to avoid its singularities, as described earlier, but this did not solve all of the problems introduced by our numerical instability. It also was not able to take obstacles into account when devising its plans.

The RRT planner, on the other hand, explored quite quickly, and came up with nice paths under many circumstances. However, due to the randomness involved, many of our paths were quite variable. The three point turn, especially, had a number of very disparate solutions to the same path problem, many of which were quite suboptimal. This unpredictability could limit its use in situations that require a high degree of consistency or precision, and occasionally resulted in paths that took a far higher number of actions than one would expect. Conversely, despite the potential for randomness arising from its initialization, the optimization planner tended to find relatively consistent plans across all three tasks, most of which were relatively direct and well-behaved compared to those found by the other planners. However, this was the slowest planner, which could drastically limit its applicability under time-sensitive conditions, such as real-time operation.

B. Applications

In the simple maneuver motion task Fig.1, our optimization-based planner was comparable to RRT in their ability to generate a simple trajectory. Here, the sinusoidal planner fell short with a more complex path planned that involved three different dramatic direction changes. Regarding the parallel parking motion, our sinusoidal planner excelled and our RRT planner had poor performance, with one extra adjustment needed. In the point turn, our sinusoidal planner performed significantly worse with a double-triangle path with large amounts of jaggedness. RRT can be improved through a more systematically generated set of motion primitives. In our current implementation, we have random constant and sinusoidal primitives. An example of an improvement would be computing a minimum set of primitives to span a lattice space [1]. This would ensure better computational performance through removing redundant primitives while still being able to reach every position. Each planner can be improved upon significantly with closed-loop control and feedback.

C. Simulation Performance

On balance, our open loop plans fared decently in simulation, reaching their planned waypoints with a relatively high rate of success. Despite this overall strong performance, there was still a great deal of variation between individual runs. Some plans tended to do significantly better than others; those with many switchbacks and a larger distance traveled were more likely to perform poorly in simulation than others, as these plans tended to be the most likely to experience cascading errors that compound as the path goes on. One notable failure occurred when the optimization-based planner running the navigation task in simulation successfully planned a path but went the opposite direction entirely upon activation. The robot was unable to move in exactly the desired way, and the lack of feedback resulted in a compounding error that eventually caused it to fail. We also had a few failures in which the robot made plans that hit the boundaries of the map, but we adjusted the state bounds to account for the robot dimensions.

D. Real-World Performance

On hardware, our open loop plans fared rather poorly. While we were able to plan and execute simple motions with gentle turns fairly well, the real world has physics and comes with

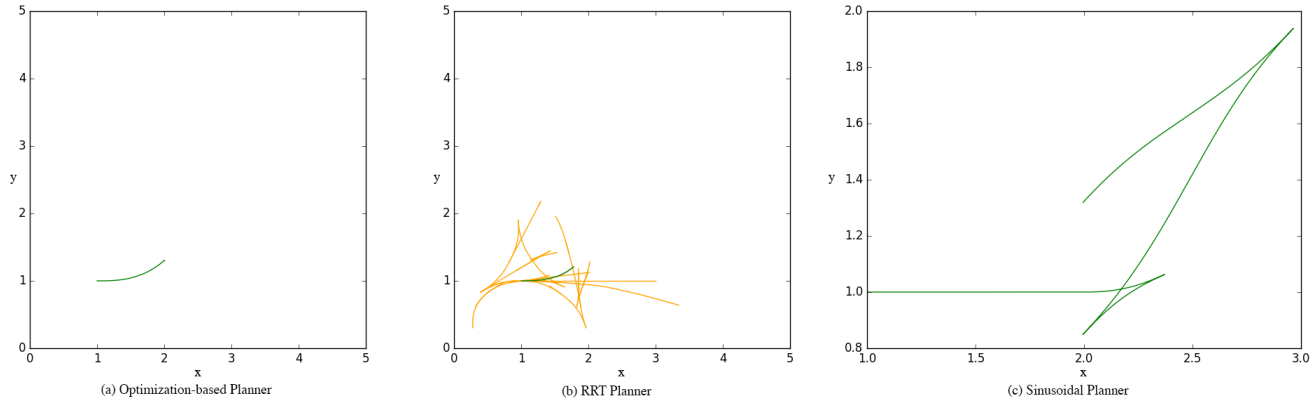

Fig. 1. A visualization of the simple maneuver task, and the approaches taken by our three different planners—optimization (left), RRT (center), and sinusoid (right)—in solving this environment. The planned path is in green, and the alternate paths explored by the RRT algorithm are in yellow.

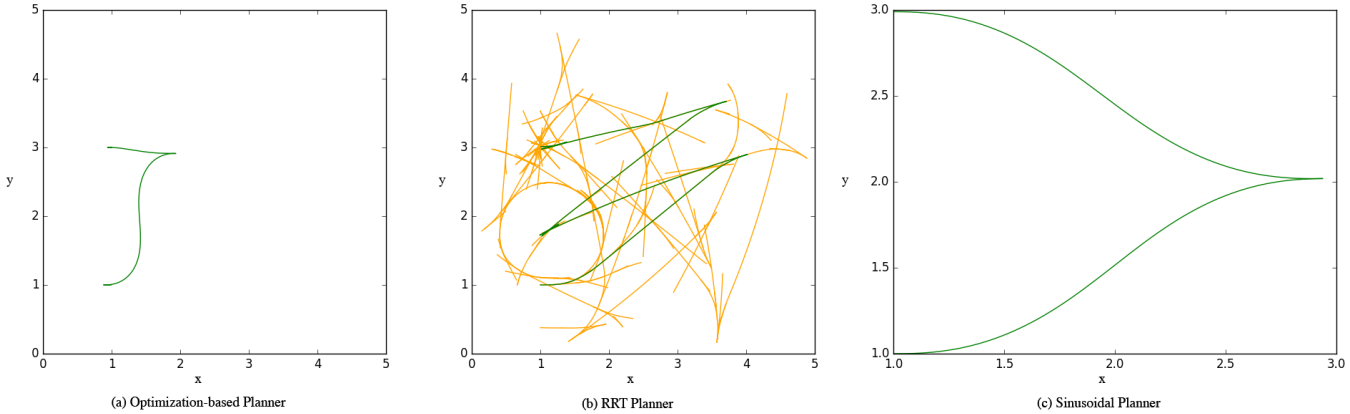

Fig. 2. A visualization of the lateral (parallel park) task, and the approaches taken by our three different planners—optimization (left), RRT (center), and sinusoid (right)—in solving this environment. The planned path is in green, and the alternate paths explored by the RRT algorithm are in yellow.

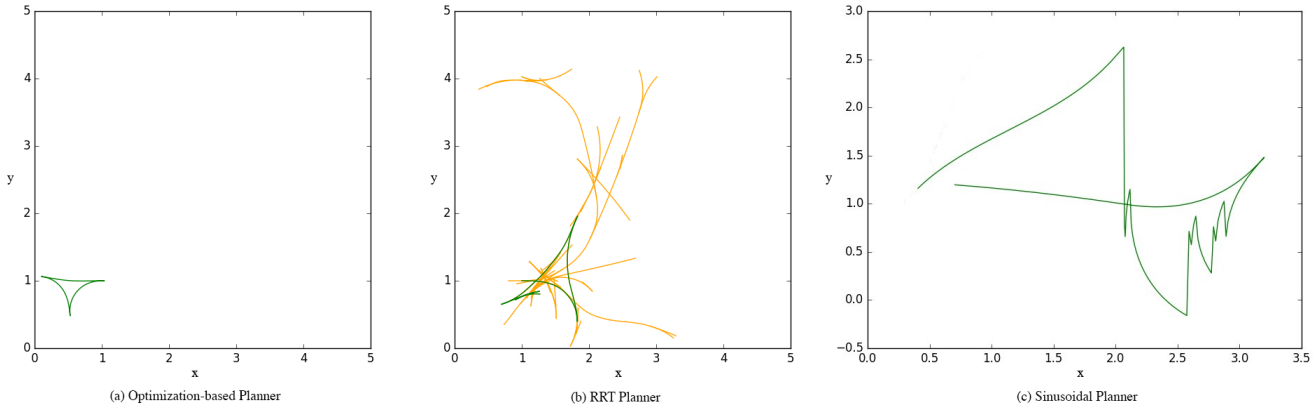

Fig. 3. A visualization of the three-point-turn task, and the approaches taken by our three different planners—optimization (left), RRT (center), and sinusoid (right)—in solving this environment. The planned path is in green, and the alternate paths explored by the RRT algorithm are in yellow.

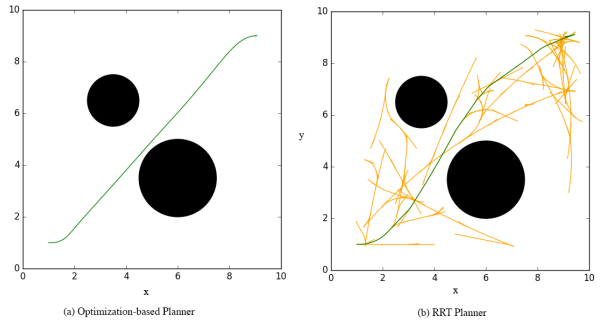

Fig. 4. A comparison between the optimization-based (left) and RRT (right) planner in solving the first navigation task (around two differently-sized objects). The planned path is in green, and the alternate paths explored by the RRT algorithm are in yellow.

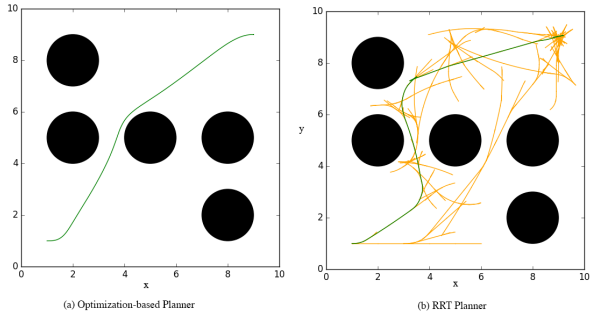

Fig. 5. A comparison between the optimization-based (left) and RRT (right) planner in solving the second navigation task (around five identically-sized objects). The planned path is in green, and the alternate paths explored by the RRT algorithm are in yellow.

friction. Regardless of planner, we were unable to execute the point turn well–the robot would try and turn a certain amount, but due to friction with the carpet, the amount of turning it actually did was incredibly underwhelming. A combination of multiple unsatisfactory turns as in some RRT and sinusoidal plans resulted in miserable failure as when we moved forward we moved too far in the wrong direction.

E. Note on the Sinusoidal Planner

Unfortunately, as implemented, the sinusoidal planner was not designed to take obstacle avoidance into account. This meant that we were unable to obtain a fair comparison between it and the other two planners on the navigation tasks. This issue arises from the sinusoid planner's process for developing a plan: the sinusoidal planner chains together a set of paths that adjust one of our state space components at a time until we reach our desired location. However, the existence of obstacles could result in a situation in which the planner cannot reach the goal state without making changes to those plans which it has already completed. This situation is somewhat reminiscent of the obstacle checking in our RRT algorithm, but it fails to allow for the backtracking and replanning that make RRT so successful at navigating through obstacle-strewn maps.

As a potential improvement, we propose modifying the steering with sinusoids algorithm so that it both accounts for obstacles, and allows for flexibility in planning to avoid them. We suggest using the same collision checking algorithm that we used for RRT to throw away solutions that intersect with objects, and using our multi-plan chaining approach to backtrack when we find that a plan intersects with an obstacle, and searching using intermediate points and different models until we are able to find an appropriate path. In doing so, we would form an RRT-like exploration system with our "primitives" being ordinally optimized paths that reach further and are smarter about reaching the goal. Thanks to this similarity with RRT, we could reuse much of our code concerning collision checking, graph building, and goal sampling, except with a more deterministic outcome. We worry that such a method would trade-off some computation time for these new capabilities, but in doing so would become a much more useful algorithm in real-world applications.

REFERENCES

[1] Botros, A. and Smith, S. L., "Multi-Start n-Dimensional Lattice Planning with Optimal Motion Primitives", 2021.
[2] Murray, R. and Sastry S., "Nonholonomic Motion Planning: Steering Using Sinusoids," IEEE Transactions on Automatic Control, 1999.
[3] LaValle, Steven M., "Planning Algorithms," Cambridge University Press, 2006.
[4] LaValle, Steven M., "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

V. APPENDIX

A. Supplementary Materials

Link to GitHub repository with implementations: https://github.com/ucb-ee106-classrooms/project-2-pinky

Link to Google Drive with Videos: https://drive.google.com/file/d/1kYiw5-KUZVam9YsDqBV5nuegIYcm5QwB/view?usp=sharing

B. Learning Goals

Over the course of this assignment, we implemented three different motion planning algorithms, with three very different paradigms for path construction. We got hands-on experience building, testing, and understanding an optimization-based path planning approach (CasADi optimization planner), a sampling-based approach (RRT), and an analytical approach (Steering with Sinusoids). These diverse paradigms for tackling a fundamental problem in robotics helped give us both a survey of the field, and allowed us to build a much deeper intuition about the problem as a whole. This assignment was quite challenging, but also very interesting and rewarding. We all came away feeling much more comfortable with the algorithms discussed here, and feel like we have a deeper understanding of why motion planning is such a difficult problem. The sinusoidal planner especially seemed very intimidating and abstruse before working through the project, but feels much more approachable now. Admittedly, the difficulty of this project felt at times like it undermined its pedagogical efficacy, as the amount of work required to implement some of the algorithms made it difficult to take the

time to step back and ensure our understanding. This is despite the fact that our group started very soon after the project was released, and felt relatively secure in the concepts—we are sure that this sentiment is felt even mroe acutely by thsoe who started the project later. However, this is a relatively minor complaint, and overall this project was really interesting and fun to tackle!