MEAM 5200 - Lab 4 Report

Cédric Hollande, John D'Ambrosio

May 3, 2024

Abstract

This report details the development and testing of two path planning algorithms: a **Potential Fields (PF)** planner and a **Rapidly-exploring Random Tree (RRT)** planner, both tailored for the Panda robotic arm. The methodologies leverage distinct approaches to navigate through simulated environments with static obstacles. The performance of each planner is assessed based on their success rate, computational efficiency, and consistency across different test conditions. Our findings highlight the conditions under which each planner excels or struggles, providing insights into their practical applications in robotic navigation and manipulation.

1 Introduction

Path planning is a critical component in robotics, essential for autonomous navigation and manipulation in complex environments. This report delves into two prominent planning algorithms: the **Potential Fields** and the **RRT** (Rapidly-exploring Random Tree) planners, both tailored for the Panda robot arm—a versatile robotic system capable of performing intricate tasks.

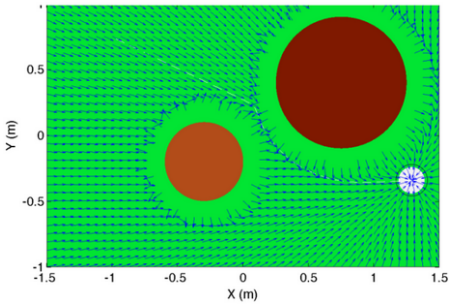

Figure 1: Potential Fields

Through a comparative analysis, we aim to assess their efficiency, reliability, and suitability under various conditions, providing a foundation for more informed algorithm selection and further development.

The **Potential Fields** method constructs a vector field in the robot's workspace, where each point exerts a force on the robot—either pulling it towards a goal or pushing it away from obstacles. This intuitive and straightforward approach facilitates quick decision-making, essential for real-time applications.

However, its effectiveness can be limited by the occurrence of local minima and the simplistic nature of the forces involved, which may not adequately handle complex environments. As the workspace becomes denser with obstacles, the effectiveness of the Potential Fields method diminishes.

Conversely, the **RRT** algorithm adopts a more explorative strategy, incrementally constructing a tree from the robot's initial state towards the goal. This method excels in environments with complex and high-dimensional obstacle configurations, providing a robust solution for spaces where PF might falter. The RRT's flexibility and adaptability make it a powerful tool for path planning in unpredictably dynamic settings.

As we explore these algorithms further in the subsequent sections, we will present a detailed examination of their implementation, including the strategic modifications and optimizations applied to enhance their performance. The comparative analysis will cover their operational strengths and weaknesses, culminating in a discussion on their practical applications in robotic navigation and manipulation. This comprehensive evaluation will not only underline the conditions under which each planner excels but also illuminate potential avenues for future research and development, aiming to bridge the gap between theoretical robustness and practical applicability.

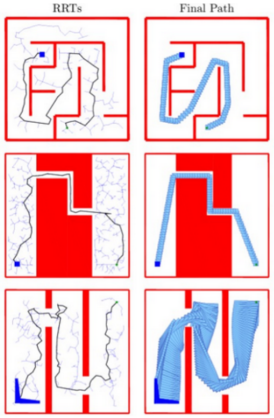

Figure 2: RRT

2 Methods

2.1 Potential Fields Planning Strategy

2.1.1 Attractive and Repulsive Force Calculation

The potential field planning was implemented using a combination of attractive and repulsive forces, designed to guide the robot towards a goal while avoiding obstacles. The attractive force $\mathbf{F}_{\text{att,i}}$ pulls the robot towards the target. When the distance is high we define it through the *Conic well Potential* (linear approach). And when the joint distance to the goal is below a set threshold $\mathbf{d}_{thresh}$ we define it by the *Parabolic well Potential* (smoothing the trajectory down): $\mathbf{d}_{goal} = \|\mathbf{o}_i(\mathbf{q}_f) - \mathbf{o}_i(\mathbf{q})\|$

$$\text{if} \quad \mathbf{d}_{goal} > \mathbf{d}_{thresh} : \ \mathbf{F}_{\text{att},i(Conic)}(\mathbf{q}) = \|\mathbf{d}_{goal}\|, \quad \text{else} \quad \mathbf{F}_{\text{att},i(Para)}(\mathbf{q}) = \frac{1}{2}\xi\|\mathbf{d}_{goal}\|^2$$

where ξ is the attractive gain, $\mathbf{o}_i(\mathbf{q})$ is the current position, and $\mathbf{o}_i(\mathbf{q}_f)$ is the final target position. The repulsive force $\mathbf{F}_{\text{rep,i}}$, activated when an obstacle is within a critical distance. We first calculate $\mathbf{d}_{b,i}(\mathbf{o}_i(\mathbf{q}))$, the smallest distance from a given joint center point to an obstacle, using **dist_point2box**. Then, to approximate the hit-box of the robot, we use a spheres approximation method detailed below, setting the joint radius $\mathbf{r_i}$. We then compute the actual distance: $\rho(\mathbf{o}_i(\mathbf{q})) = \mathbf{d}_{b,i}(\mathbf{o}_i(\mathbf{q})) - \mathbf{r}_i$, and ρ_0 is the influence range or distance threshold. We get:

$$\text{if} \quad \rho(\mathbf{o}_i(\mathbf{q})) < \rho_0 : \ \mathbf{F}_{\text{rep},i}(\mathbf{q}) = -\eta_i \left(\frac{1}{\rho(\mathbf{o}_i(\mathbf{q}))} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(\mathbf{o}_i(\mathbf{q}))} \nabla\rho(\mathbf{o}_i(\mathbf{q})), \quad \text{else} \quad \mathbf{F}_{\text{rep},i}(\mathbf{q}) = 0$$

See the figures below which were used to find the most optimal gains (Figure 4) and radii (Figure 5) for each joint. This allows us to have an offset and approximate a hit-box I developed in Blender for visual context, import the *.dae* files of each joint of the Franka Emika Panda Robot found from the internet, and placed each one of them based on my code computation of the FK for the given configuration in Figure 3.

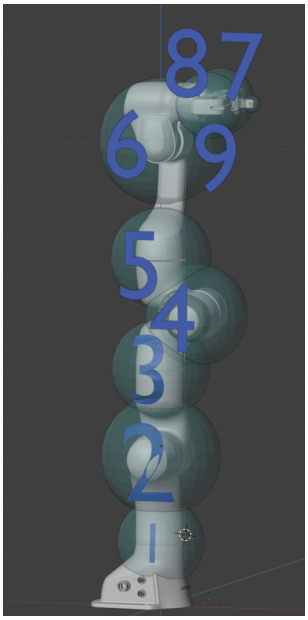

Figure 3: Spheres Hit-box Approximation

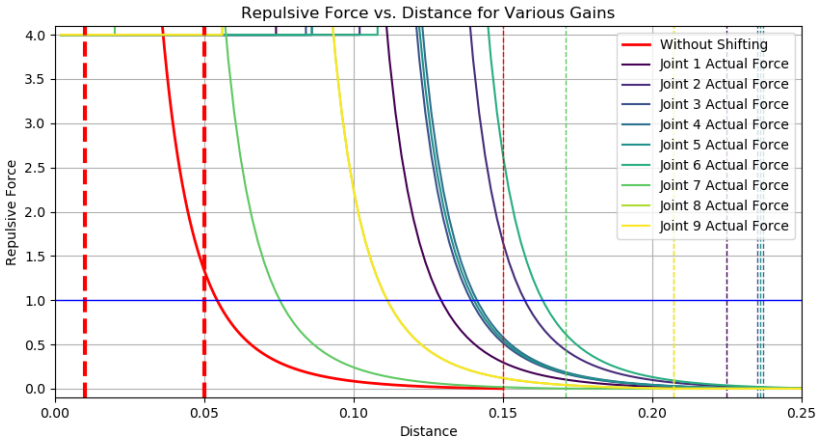

Figure 4: Force Graphing based on Distance

Figure 5: Radius of spheres based on joint index

$$\begin{bmatrix} 0.075 & 0.103 & 0.085 & 0.087 & 0.086 & 0.109 & 0.015 & 0.057 & 0.057 \end{bmatrix}$$

2.1.2 Torque Calculation and Gradient Descent Strategy

The torques required for robot motion are computed using the equation:

$$\tau(\mathbf{q}^i) = \sum_{i=1}^{N} \tau_i = \sum_{i=1}^{N} J_{v,i}^T F_i \in \mathbb{R}^{N\times 1}, \quad \text{for} \quad F_{\text{i}} = F_{\text{att,i}} + F_{\text{rep, i}}$$

where $J_{v,i}$ is the Jacobian velocity matrix corresponding to joint i, and F_i is the force vector. This formula integrates the forces through the robot's kinematic chain, converting them into joint-specific torques.

The planning algorithm employs a gradient descent strategy, which iteratively updates the robot's configuration:

1. Initialize q^0 to q_{init} and set $i = 0$.
2. While $q^i - q_{\text{final}} < tolerance$, perform the following:
   - Update $q^{i+1} = q^i + \alpha \frac{\tau(q^i)}{\|\tau(q^i)\|}$.
   - Increment i.
3. If a local minimum is detected, execute a random walk to escape, then continue from step 2.

Following this method, we then tuned the increment factor α, and found 1e-3 to work best with our setup.

2.1.3 Code Structure

The code structure, mostly implemented by Cédric Hollande, is designed to calculate the forces, torques and gradient descent strategy to describe the joint configurations the robot needs to go through to reach the goal position whilst avoiding obstacle collisions. The main functions used are:

- **attractive_force**: Calculates the attractive force between the current and target positions.
- **repulsive_force**: Computes the repulsive force based on the robot's proximity to obstacles.
- **compute_forces**: Aggregates all forces acting on the robot arm.
- **compute_torques**: Converts forces into joint torques using the Full Jacobian matrix of the robot.
- **compute_gradient**: Determines the gradient step in joint space to move the robot arm towards the target configuration.
- **plan**: Controls the iterative process of robot path planning until the goal is reached or a local minimum is encountered, calling helper functions to ensure each iteration remains within joint limits and does not lead to a collision.

In the planning function, the robot iteratively calculates the gradient of the potential field to determine the joint updates. If a local minimum is detected, a random walk is executed to escape, using the **random_walk** function which also goes through joint limit and collision checks. The process continues until the robot reaches the goal or the maximum number of iterations is exceeded.

This structure ensures a modular approach, where each component can be independently tested and optimized, contributing to the overall robustness and flexibility of the path planning algorithm.

2.2 RRT Planning

2.2.1 RRT Planning Strategy

The RRT (Rapidly-exploring Random Tree) planning strategy we implemented uses a simple tree structure to store parent-child node relationships. The planner uses functions to manage collision detection, random configuration generation, and path construction.

For each iteration, the algorithm begins by checking if the starting configuration is collision-free. If a collision is detected, an error is raised. Random configurations are then generated using a random uniform distribution between the upper and lower joint limits, ensuring that the paths always respect the physical constraints of the robot. We employ a goal biasing strategy, which will randomly select the goal node as the target for the tree to expand towards, which can help speed up computation time and increase the chance of path convergence. Once the random node has been generated, the algorithm selects the nearest node from the current tree and attempts to extend towards a new random configuration.

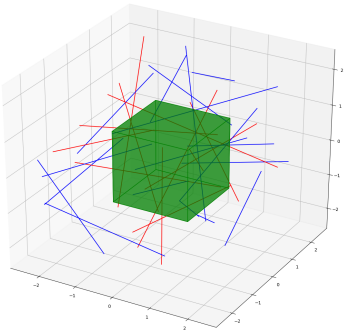

Figure 6: RRT Collision Tests

If this extension does not result in a collision, the new node is added to the tree. The algorithm continually checks if the goal is directly reachable from the latest node without encountering obstacles. If reachable, the goal is added to the tree, and the path is constructed by tracing back through the tree from the goal to the start. The planning algorithm terminates either when the goal is connected within the maximum number of iterations or if the goal cannot be reached, returning an empty path if unsuccessful. As for parameters, 1000 steps, for the machine used for testing, was typically enough to reach convergence and keep computation times low. We used a step size of 0.1 and a goal bias probability of 50%. This approach effectively explores the configuration space by incrementally building a tree directed towards the goal while avoiding collisions, leveraging random sampling to explore various paths efficiently.

2.2.2 Code Structure

John D'Ambrosio was responsible for most of the RRT planner code. The code defines several helper functions, such as `isRobotCollided`, which checks to see if the any robot is in collision at a given configuration q within certain safety margins from obstacles defined in a map object.

`random_config` generates a random configuration with respect to the joint limits. `distance` calculates the Euclidean distance between two nodes, and is directly used in `extend` to incrementally steer the tree towards a target configuration within predefined maximum step size. `reconstruct_path` builds the path by tracing back from the goal to the start using parent connections stored in a dictionary representing the tree structure. In this tree, each new node stores a reference to its parent node. We initialize the tree with the start configuration, and then randomly sample new nodes, using a Python lambda function to find the nearest neighbors according to distance, call the `extend` function, and append the outputted q_{new} to the tree with q_{near} as its parent. We continually check to see if the goal is in collision and if the path directly form q_{new} to the goal will cause a collision, and if not, we append the goal to the path and return the final path. If the goal is not established within the set iterations, the function returns an empty path. This implementation dynamically constructs a search tree that efficiently navigates around obstacles and directs towards the goal.

3 Evaluation

3.1 Potential Fields Planning

The evaluation of the Potential Fields planner revealed challenges in achieving consistency in path planning. Although the planner was capable of finding a path, it frequently encountered obstacles in between the center of each joint, affecting its reliability. Adjustments were made to enhance the planner's ability to consistently find a path; however, these adjustments often led to collisions with obstacles.

The sphere approximation method, in addition to being more efficient at detecting obstacles within the set boundaries, significantly reduced the running time from an initial average of 150 seconds down to approximately 15 seconds. This modification also improved the planner's proximity to obstacles without direct collisions, as depicted in Figure 7.

The paths generated by the planner were largely similar across runs, with slight variations due to inherent randomness in the planning process. Typically, the planner succeeded within 1200 iterations for the first map configuration.

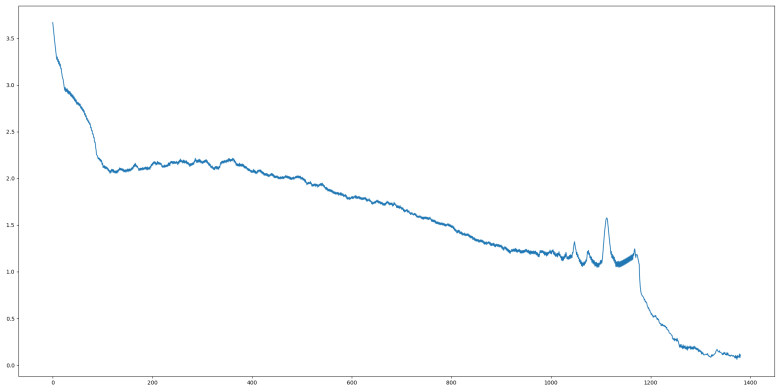

Figure 7: Graph showing the decrease in path length across iterations during a successful run of the PFP.

Additionally, to enhance our testing methodology and to better understand the interaction between the robot's trajectory and obstacles, we incorporated a 3D plotting technique with small interpolations to detect obstacle collisions. This technique was crucial for visualizing the trajectory of the central joint positions between successive states i and $i+1$. By plotting these trajectories in three dimensions, we could directly observe potential collisions and better identify specific points along the trajectory where the planning algorithm failed or encountered issues.

This visualization tool not only facilitated a more intuitive understanding of the spatial dynamics involved but also significantly aided in debugging and refining the planner's performance. The 3D plots allowed us to iterate the planning code more effectively, pinpointing the exact iterations where the path planning strategy diverged from expected behaviors or collided with obstacles. Figure 8 shows one such 3D plot, highlighting the trajectory and interaction with obstacles during one of the test runs.

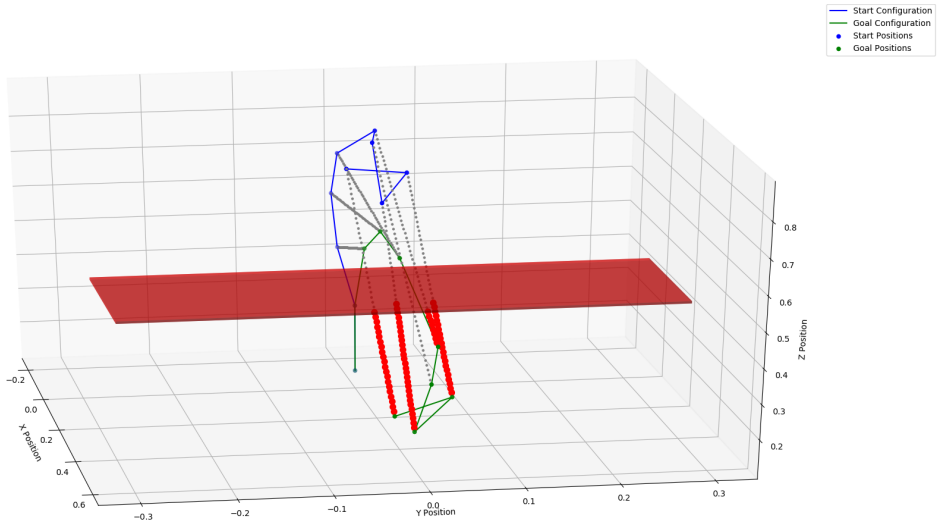

Figure 8: 3D plot of the central joint positions during a planner test run.

Despite these enhancements, the inherent limitations of the Potential Fields approach, such as its struggle with dynamic obstacle avoidance and susceptibility to local minima, were evident. These issues underscore the need for further optimization or alternative strategies to improve robustness and reliability.

3.2 RRT Planning

The evaluation of the RRT planner yielded great results and a promising algorithm which has great potential to be improved upon in the final project. The planner succeeded in finding a path at a very high rate, nearly 100% with finely tuned goal biasing. The runtime was low, usually less than 1 second and typically ranging from 0.15 to 0.80 seconds. Although the paths may have been found, they were usually not optimal, which is expected given the random nature of the planning strategy. These paths would very clearly reflected this random nature, often making steps in directions that had no clear benefit in terms of reaching the goal. The paths were similar in nature, probably due to the obstacles in the different environments tested, and maybe also the goal biasing, but the planner tended to create different paths every time it ran. This planner would probably be very fast in an obstacle free environment, but it may create too many branches that don't reach the goal. For uncluttered environments, Potential Fields would be prefered as it can quickly converge to goal configurations when there are little to no obstacles. Overall, our RRT implementation was a great success, and laid much of the ground work for our planning strategy in the final project.

4 Analysis

4.1 Effectiveness of each planner in various environments

The RRT planner tended to be more effective in almost all environments, thanks to its ability to rapidly sample all of the configuration space whilst remaining within joint limits. The gradient-descent based planning strategy for the potential fields would often cause the planner to get stuck or hover around

certain spots, resulting in a slow convergence time in cluttered or complex environments. Additionally, checking for collisions was harder and slower for potential fields, whereas in RRT you could use FK to see if the generated configuration would be in collision in real time. Overall, our testing not only revealed RRT to be more efficient, but much more reliable and safe in cluttered or complex environments, which is an essential for proper robotics path planning.

4.2 Situations where one planner might be preferred over the other

As mentioned earlier, the potential fields planner would be preferred if the environment were uncluttered. This is because it would have much less local minima, requiring less random walks and much quicker planning times. Also, you can be guaranteed to get more optimal paths thanks to the gradient-descent optimization based planning. RRT will almost always return a path, but will usually not be optimal, as it is random in nature, and uninformed about distance or cost. In sum, both planners have their pros and cons. RRT would be preferred in environments that are cluttered or where speed is prioritized over optimality. Potential fields would be the preferred choice in simple environments that require safety guarantees. This planner would also succeed in paths that are often repeated or need to be pre-computed.

4.3 Potential improvements if more time was available.

If we had more time, implementing and optimization layer into RRT, making it RRT*, would greatly improve our planner, as it combines the speed and breadth of RRT with the optimality of heuristic based methods like A*. As for artificial potential fields, we simply would've spent more time tuning our gains and other parameters, as the success of this algorithm is largely dependent on its highly sensitive attractive and repulsive gains in addition to other variables. Lastly, we would've made our collision checking more robust and created smarter safety margins so that our planner can perform well in environments where avoiding collisions is crucial

5 Conclusion

In conclusion, this report has analyzed the implementation and performance of two robotic path planning algorithms: the Potential Fields and the Rapidly-exploring Random Tree (RRT) planners. Our findings revealed that the RRT planner not only performed robustly in various environments but also operated faster than the Potential Fields planner. These results highlight the RRT's effectiveness and versatility in navigating through diverse scenarios. Future research could focus on hybrid models that combine the rapid planning capabilities of RRT with the optimality and intuitive force-based interactions of Potential Fields, potentially enhancing both the efficiency and adaptability of path planning strategies across different settings.