Project 1

EE4308: Autonomous Robot Systems

ID: A0299063U
ID: A0298841M
ID: A0296420E

March 2, 2025

Contents

Abstract

This study presents a comprehensive framework for enhancing autonomous robot navigation in static environments by integrating three core components: an advanced Regulated Pure Pursuit (RPP) controller, an A* path planner, and a Savitzky-Golay smoothing module. The RPP controller extends the traditional Pure Pursuit algorithm by dynamically adjusting the lookahead distance based on the robot's speed and path curvature, while incorporating dual heuristics—curvature-based and proximity-based—to regulate velocity. This adaptive mechanism not only minimizes lateral deviations and oscillations during sharp turns but also optimizes the robot's response to varying environmental constraints. In parallel, the A* algorithm is employed to generate optimal, collision-free paths on discretized grid maps. Recognizing that A* outputs a series of discrete waypoints unsuitable for smooth real-world navigation, the framework applies Savitzky-Golay Smoothing to convert these segmented paths into continuous, kinematically feasible trajectories. Extensive experimental evaluations conducted both in simulation and on a TurtleBot platform confirm that the integrated approach significantly improves trajectory tracking accuracy, enhances stability during high-curvature maneuvers, and ensures effective obstacle avoidance.

Keywords :Regulated Pure Pursuit, A-star Algorithm, Savitzky-Golay Smoothing

1 Introduction

The accuracy and robustness of autonomous robot navigation are among the core challenges in mobile robotics research. This project aims to enhance a robot's path-following and navigation capabilities in **static environments** by implementing the **Regulated Pure Pursuit (RPP) algorithm**, A* Path Planner, and Savitzky-Golay path smoothing techniques.

Regulated Pure Pursuit (RPP) is an improved control method based on the classical Pure Pursuit algorithm. While the traditional Pure Pursuit algorithm uses a geometric approach to guide the robot along a path, RPP enhances path-tracking smoothness and safety by incorporating heuristic rules and adaptive lookahead points.

The A* algorithm is a widely used heuristic search method for path planning that can determine the optimal route from a start point to a goal on a discretized grid map. While A* effectively generates collision-free paths, these paths typically consist of segmented straight lines, which are not ideal for smooth robot movement. Therefore, in this project, we apply the Savitzky-Golay smoothing algorithm to refine the paths generated by A*, making them better suited to the robot's kinematic constraints.

The project consists of the following key steps:

1. Development of the Regulated Pure Pursuit (RPP) controller: Computing velocity commands based on the robot's current state to ensure stable path tracking while maintaining obstacle avoidance capability.

2. Implementation of the A Path Planner*: Finding feasible paths from the start point to the goal within the environmental map, integrating cost maps for optimized path selection.

3. Path smoothing processing: Applying Savitzky-Golay smoothing to the A*-generated path to enhance trajectory feasibility.

4. Experimental testing and performance evaluation: Testing the navigation system in both a simulation environment and a real robot platform (TurtleBot) to analyze its performance across different scenarios.

This report provides a detailed discussion of the RPP controller, A* planner, and path smoothing techniques, including their algorithmic principles, implementation methods, improvements, and experimental results. Additionally, it explores potential future optimizations.

2 Basic Principe

2.1 Pure Pursuit

The Pure Pursuit algorithm represents a geometric path tracking methodology that is extensively utilized within the realms of mobile robotics and autonomous navigation. This algorithm empowers a robot to adhere to a pre-defined trajectory by persistently modifying its heading to align with a dynamically selected lookahead point. The algorithm calculates the curvature necessary for the robot to seamlessly approach the lookahead point, thereby ensuring efficient and stable motion control.

In this methodology, the robot identifies a lookahead point on the trajectory that is situated at a minimum specified distance. This lookahead point functions as an ephemeral objective for the robot to pursue. The fundamental principle underlying Pure Pursuit involves ascertaining the curvature of a circular arc that intersects both the robot's current location and the lookahead point. By computing this curvature, the robot generates an appropriate angular velocity command to align its heading with the selected point, thus facilitating smooth and efficient path tracking.

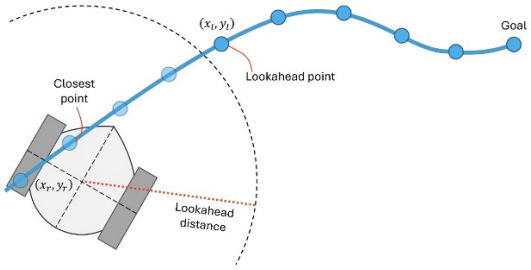

Figure 1: Lookahead Point Selection

To elucidate the mathematical underpinnings of this method, let the robot's global position be (x_r, y_r) denoted by, and the lookahead point be (x_l, y_l). The position of the lookahead point within the robot's frame can be ascertained through a coordinate transformation process. The displacement between the robot and the lookahead point is quantified by:

$$\Delta x = x_l - x_r \tag{1a}$$
$$\Delta y = y_l - y_r \tag{1b}$$

Upon rotating these coordinates into the robot's reference frame, the transformed coordinates (x', y') are derived as follows:

$$x' = \Delta x \cdot cos(\theta) + \Delta y \cdot sin(\theta) \tag{2a}$$
$$y' = -\Delta x \cdot sin(\theta) + \Delta y \cdot cos(\theta) \tag{2b}$$

where θ represents the robot's heading in the global frame. The curvature c, which dictates the required turning rate, is subsequently computed as:

$$c = \frac{2y'}{(x')^2 + (y')^2} \tag{3}$$

This curvature value directly influences the angular velocity ω of the robot, which is determined by:

$$\omega = v \cdot c \tag{4}$$

where v denotes the linear velocity of the robot. A larger curvature value results in more acute turns, whereas a smaller curvature value ensures smoother motion along the path. One of the pivotal parameters that impact the performance of the Pure Pursuit algorithm is the lookahead distance L_d. An extended lookahead distance prompts the robot to anticipate further along the path, leading to smoother but potentially less precise tracking. Conversely, a reduced lookahead distance enhances tracking accuracy but may introduce oscillations due to more frequent directional adjustments. The selection of L_d is thus critical and is contingent upon the application's requirements, such as the curvature of the path and the desired level of responsiveness.

In summary, the Pure Pursuit method offers an effective approach to navigate along a predefined path by continuously selecting a forward-looking target and calculating the requisite curvature to follow it. This method ensures a balance between smooth trajectory tracking and responsive maneuverability, rendering it a widely adopted strategy in robotic motion control.

2.2 Regulated Pure Pursuit

Despite the simplicity and computational efficiency of the Pure Pursuit algorithm, it has several drawbacks:

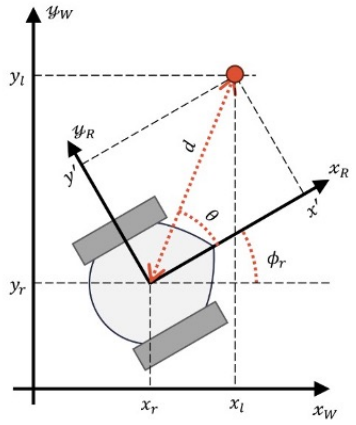

Figure 2:
Transformation of Lookahead Point

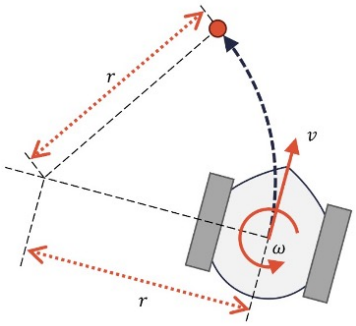

Figure 3:
Curvature Calculation and Pure Pursuit

1. On high-curvature paths, excessive deviation is likely to occur, causing the robot to veer off course or oscillate when making sharp turns.

2. It lacks speed control, meaning it cannot dynamically adjust velocity based on the environment, which may pose safety risks.

To address these limitations, Steve Macenski and his colleagues[2] proposed an improved algorithm called Regulated Pure Pursuit (RPP). This method enhances path-following performance in robot navigation by building upon the traditional Pure Pursuit approach. Designed specifically for service and industrial mobile robots operating in constrained and partially observable environments, RPP introduces heuristic-based speed regulation to facilitate obstacle avoidance while maintaining orientation constraints.

The first step in the Regulated Pure Pursuit algorithm involves transforming the input path $P_t\{p_0, ..., p_j, p_r, ..., p_n\}$ into the robot's base coordinate frame P'_t and pruning it. Any transformed path points P'_i that are significantly far away (i.e., when $dist(p'_r, p') >> L_t$) are removed from immediate consideration, as they are irrelevant to the current movement at time t. However, these points remain stored for future iterations until a new path update is received as the robot progresses.

To further enhance path-following performance, a fixed lookahead distance may not be the optimal one. Instead, dynamically adjusting the lookahead distance yields superior performance. The **dynamic lookahead distance** L_h is defined as

$$L_h = v'_t \, g_l \tag{5}$$

Where v'_t is regulated linear velocity, g_l is lookahead gain. Compared to a fixed lookahead approach, this adaptive method enables the robot to extend its lookahead distance at higher velocities while shortening it in low-speed or obstacle-avoidance situations. This improves path-tracking smoothness and mitigates instability arising from abrupt curvature variations.

The desired linear velocity v_t is next further processed by the curvature and proximity heuristics. Both heuristics are applied to linear velocity and we take the maximum of the two.

The purpose of the **curvature heuristic** is to slow the robot to v'_t when navigating sharp turns in partially observable environments, such as when entering or exiting hallways and aisles commonly found in retail, factories, and shopping malls. By reducing speed in such scenarios, the robot can traverse blind corners more safely. This heuristic is applied to the linear velocity v_t when the change in curvature C exceeds a predefined threshold

C_h, ensuring that minor turns or slight path variations do not unnecessarily trigger speed reductions. The curvature velocity v_t' returned by this heuristic is selected as

$$v_t' = \begin{cases} vt, & \text{Otherwise} \\ v_t \frac{C_h}{C}, & \text{if } C_h < C \end{cases} \tag{6}$$

The **proximity heuristic** is activated when the robot approaches dynamic obstacles or fixed infrastructure. This mechanism can slow the robot in constrained environments where the potential for collision risk is higher. Reducing velocity near static obstacles minimizes the impact of minor path deviations, thereby enhancing stability in narrow spaces. Additionally, in environments where the robot operates near humans or other dynamic agents, lowering speed ensures a quicker reaction time, thereby improving safety. The proximity heuristic regulates velocity based on the ratio of $\frac{d_o}{d_{prox}}$ is defined as

$$v_t' = \begin{cases} vt, & \text{Otherwise} \\ v_t \frac{d_o}{d_{prox}}, & \text{if } d_o < d_{prox} \end{cases} \tag{7}$$

Where d_{prox} is the proximity distance to obstacles to apply the heuristic, d_o is the current distance to an obstacle. The proximity threshold d_{prox} should be determined based on system requirements, ensuring that the robot only begins to reduce speed when approaching obstacles within a critical range.

2.3 A* Algorithm

The study of efficiently finding the minimum-cost path in graphs has long been a key research area in various engineering and scientific applications. This problem is widely relevant to fields such as telephone network routing, circuit board layout design, and robot path planning. Early graph search algorithms, such as Depth First Search (DFS), can quickly identify a path between two points but cannot guarantee optimality. In contrast, Breadth First Search (BFS) can find the optimal solution but is limited in its applicability. BFS is particularly suitable for path planning in dense grids but may become inefficient in sparse ones. Many excellent algorithms have successfully solved the minimum cost path problem, which can be mainly divided into two categories: mathematical approaches and heuristic approaches. The Dijkstra algorithm, derived from the optimization of DFS and BFS, is a typical mathematical method. It guarantees finding the minimum cost path but suffers from slow speed due to the large search range and high information storage requirements. On the other hand, Greedy Best First Search (GBFS) is a heuristic method that greatly improves computational efficiency but cannot guarantee finding the optimal solution, often falling into local optima. Therefore, a new path planning algorithm is needed that improves search efficiency while ensuring optimality — the A* algorithm proposed by Peter E. Hart and others in 1968.[1]

The A* algorithm combines mathematical and heuristic methods by using heuristic information to evaluate the current state and guiding the search direction with the optimal evaluation value, reducing unnecessary node expansions and achieving efficient minimum cost path searches. The core of the A* algorithm is the heuristic evaluation function, which combines the actual path cost $g(n)$ and the heuristic estimate $h(n)$ to guide the entire search process:

$$f(n) = g(n) + h(n) \tag{8}$$

The actual path cost $g(n)$ calculates the cost from the start node s to the current node n. It can be expressed as:

$$g(n) = g(p) + c(p, n) \tag{9}$$

Where $g(p)$ is the path cost from the start node to the parent node p, and $c(p, n)$ is the edge weight (path cost) from the parent node p to the current node n. During the algorithm's execution, each node's $g(n)$ is continuously updated to ensure path optimality.

The heuristic estimate $h(n)$ represents the estimated cost from the current node n to the target node e. The A* algorithm relies on $h(n)$ to guide the search, prioritizing the expansion of paths more likely to reach the target. The choice of $h(n)$ determines the search efficiency, and it must be an underestimate of the actual cost; otherwise, the A* algorithm may lose its optimality:

$$h(n) \leq h*(n) \tag{10}$$

Common heuristic functions include:

1. Euclidean Distance: Computes the shortest straight-line path in continuous 2D/3D space, suitable for applications such as aerial navigation and free-space movement.

$$h(n) = \sqrt{(x_n - x_t)^2 + (y_n - y_t)^2} \tag{11}$$

2. Manhattan Distance: Measures movement in a grid-based environment along horizontal and vertical axes, often used in urban road navigation and robot pathfinding.

$$h(n) = |x_n - x_t| + |y_n - y_t| \tag{12}$$

3. Diagonal Distance: Applicable to movement in eight possible directions, commonly used in grid-based motion planning.

$$h(n) = max(|x_n - x_t|, |y_n - y_t|) \tag{13}$$

Furthermore, adjusting the weight of h(n) can modify the search strategy. When w > 1 , the search becomes more greedy, increasing speed at the cost of potential non-optimality. When w $= 1$, the algorithm behaves as standard A*, ensuring optimality.

$$h(n) = w \cdot h(n) \tag{14}$$

A* not only guarantees finding the minimum-cost path but also significantly reduces the number of expanded nodes compared to other algorithms with the same heuristic estimate, greatly improving search efficiency. As a result, it has been widely applied in real-world scenarios, including GPS navigation route optimization and robot motion planning.

3 Code Implement

3.1 Pure Pursuit

3.1.1 Controller Architecture and Implementation

The Pure Pursuit controller within this project is realized as a bespoke plugin integrated into the Nav2 framework, comprising critical components such as the controller header, source files, and parameter configurations. The controller header delineates function prototypes and class variables, while the source file encapsulates the core logic. The configuration file specifies tuning parameters, notably the lookahead distance. The function *computeVelocityCommands*() is central to this process, as it calculates the curvature necessary for the robot to adhere to the trajectory and generates the corresponding velocity commands.

The algorithm operates through a structured sequence: initially, the global path is parsed to extract trajectory points. Subsequently, the lookahead point is ascertained based on a preset distance threshold. This point is then translated into the robot's local frame to facilitate curvature computation. Ultimately, the derived curvature is utilized to compute the angular velocity, which is subsequently applied to the robot via the velocity command publisher.

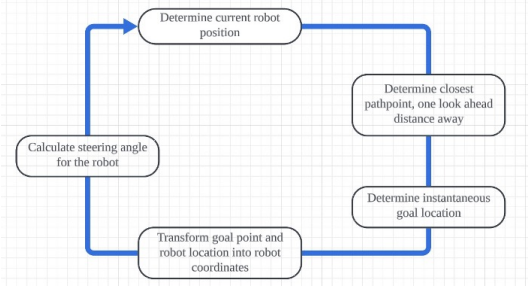

Figure 4: Pure pursuit algorithm

3.1.2 Lookahead Point Selection

The lookahead point is dynamically selected based on a predefined lookahead distance l_d, which is crucial for ensuring smooth and accurate path tracking. The distance metric is calculated using the Euclidean distance formula:

$$l_d = \sqrt{(x_l - x_r)^2 + (y_l - y_r)^2} \tag{15}$$

Where (x_r, y_r) and (x_l, y_l) represent the robot's current position and a candidate point on the path, respectively. If no point on the path meets or exceeds the lookahead distance, the last available point is designated as the lookahead point. This point is then translated into the robot's local frame using the previous transformations.2This transformation is essential for accurate curvature calculation.

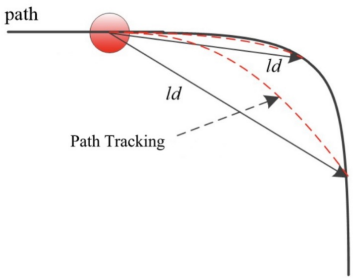

Figure 5:
Lookahead Point Geometry

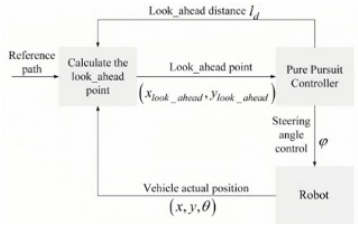

Figure 6:
Lookahead Point Calculation Process

3.1.3 Curvature Calculation and Velocity Command Generation

The curvature c derived from the transformed coordinates (x', y') 3 represents the inverse of the turning radius required for the robot to reach the lookahead point. The angular velocity ω, which dictates the robot's steering, is calculated based on the curvature and the robot's linear velocity v.4

To maintain stability and prevent excessive turning speeds, constraints are imposed on the angular velocity and the minimum turning radius:

$$|\omega| \leq \omega_{max} \tag{16a}$$

$$R_{min} \geq \frac{1}{c_{max}} \tag{16b}$$

These constraints are crucial for mitigating sudden oscillations in trajectory tracking and enhancing overall system stability.

3.1.4 Experimental Results and Observations

A series of experiments were designed to evaluate the Pure Pursuit method's performance under different lookahead distances l_d. The results indicated that l_d significantly influences the robot's trajectory tracking. A larger lookahead distance led to smoother trajectories but increased lateral errors during sharp turns, while a smaller lookahead distance improved accuracy but introduced oscillations due to frequent steering adjustments. An adaptive lookahead distance strategy, which dynamically adjusts based on speed and path curvature, demonstrated enhanced stability and responsiveness.

The experiments were conducted on a variety of path types, including straight-line trajectories, sinusoidal paths, and sharp turns. For straight paths, the robot maintained a consistent trajectory with minimal deviations. On curved paths, a larger lookahead distance reduced oscillations but resulted in slight deviations from the intended path. Sharp turns benefited from a shorter lookahead distance, improving accuracy but requiring high-frequency control adjustments. The method operated in real-time at a 10 Hz update frequency, ensuring high tracking accuracy and minimal latency.

To further analyze the performance, a comparative study was conducted between the Pure Pursuit method and other trajectory tracking algorithms, such as the Stanley Controller. The results showed that Pure Pursuit offers a better balance between smoothness and accuracy, especially in environments with complex geometries. Additionally, a sensitivity analysis on key parameters like lookahead distance and robot speed was performed to understand their impact on tracking performance.

The experimental results also highlighted the importance of integrating real-time sensor feedback for dynamic adjustments. Future work could explore the integration of machine learning techniques to predict and adapt to changes in the environment, further enhancing the robustness and efficiency of the Pure Pursuit method.

3.1.5 Conclusion

The implemented Pure Pursuit method effectively facilitates efficient trajectory tracking through dynamic lookahead point selection and curvature computation for smooth motion control. Balancing tracking accuracy and stability, significantly influenced by lookahead distance, poses the primary challenge. Experimental outcomes underscore that an adaptive lookahead distance mechanism, complemented by speed-dependent curvature adjustments, markedly enhances tracking performance. Future work may refine lookahead point selection for complex environments and integrate predictive controls to optimize navigation efficiency further.

3.2 Regulated Pure Pursuit

The standard Regulated Pure Pursuit (RPP) algorithm introduces the following adjustment mechanisms compared to the traditional pure pursuit algorithm:

1. Curvature adjustment: Limits the curvature to avoid large changes in angular velocity.
2. Obstacle avoidance heuristic: Dynamically adjusts speed based on the distance to obstacles to enhance the robot's ability to avoid collisions.
3. Adaptive lookahead distance: Dynamically adjusts the lookahead point position based on the robot's current speed and the environment.

The corresponding pseudocode for this implementation is as follows:

Algorithm 1 Regulated Pure Pursuit (RPP)

Load the global path and robot's current pose
while goal is not reached **do**
  Select a lookahead point (P_{target})
    Compute distance from the robot to all path points
    Select the nearest point with distance ≥ LookaheadDistance
  Compute the Pure Pursuit control command
    Transform P_{target} to robot's local coordinate frame
    Compute target heading angle $\alpha = \text{atan2}(y', x')$
    Compute curvature $\kappa = \frac{2\sin(\alpha)}{\text{LookaheadDistance}}$
    Compute desired angular velocity $\omega = \kappa v$
  Regulate velocity based on obstacle proximity and goal approach
    Set pose transition zone for orientation adjustment
    Check for obstacles near P_{target} using safety threshold
    Adjust linear velocity v based on goal and obstacle proximity
    Limit ω within $[-\omega_{max}, \omega_{max}]$
  Execute motion by sending (v, ω) to motion controller
  Check termination conditions
  if robot reaches goal or velocity approaches zero **then**
    Stop
  end if
end while

In our project, compared to the standard Regulated Pure Pursuit (Standard RPP), the following improvements are made:

1. Smooth orientation adjustmentWhen the robot is approaching the goal, we use a smooth quadratic function to gradually adjust the angle (orientation_weight = progresssš), instead of directly switching to angular velocity control. This approach reduces abrupt directional adjustments and improves the stability of the navigation.

2. Distance-based dynamic orientation control: When the robot is close to the goal (dist_to_goal $<$ xy_goal_thres * 2.0), the robot focuses more on angle alignment, especially when it is very close to the goal (dist_to_goal $<$ xy_goal_thres), completely ignoring linear velocity and only adjusting angular velocity. This method enhances the accuracy of alignment with the final goal and prevents jitter near the target point.

3.3 A* Algorithm

3.3.1 Implement A*

The core idea of the A* algorithm is to use a priority queue (Open List) to select the node that is most likely to reach the goal and maintain a closed list (Closed List) of visited nodes to avoid redundant calculations. Therefore, the first step is to initialize the Open List as a set of nodes to be expanded, and the Closed List as a set of already expanded nodes, with the first item in the Open List being the start node s and the Closed List initially being empty. For the start node s, we initialize $g(s)$ to 0, and calculate $h(s)$ and $f(s)$.

Once the initialization is complete, the algorithm enters the main loop. The first step in the loop is to remove the node with the smallest $f(n)$ from the Open List for the next operation. If this node is the goal node, the search terminates, and the optimal path is reconstructed by backtracking through the parent node chain. If this node is not the goal node, it is moved to the Closed List, and its neighboring nodes $neighbor(n)$ are explored. For each neighboring node m, $g(m)$ and $f(m)$ are calculated. If m is not in the Open List, it is added, and its parent node is set to n. If m is already in the Open List and the new path is better, $g(m)$ is updated, and the parent node is adjusted. This process is repeated until

the goal is reached or the Open List is empty (no solution). After the search is completed, the optimal path is reconstructed by backtracking through the parent node chain.

Algorithm 2 A* Algorithm

Input: Start node s, End node e
Initialize: Open set $\mathcal{O} \leftarrow \{s\}$,Closed set $\mathcal{C} \leftarrow \{\}$
Set $g(s) = 0, f(s) = h(s, e)$
while $\mathcal{O} \neq \emptyset$ **do**
    $n \leftarrow$ node in $\mathcal{O}$ with lowest f
    if $n == e$ **then**
        Return discrete sequence
    end if
    $\mathcal{O} \leftarrow \mathcal{O} \setminus \{n\}, \mathcal{C} \leftarrow \mathcal{C} \cup \{n\}$
    for each neighbor m of n **do**
        if $m \notin C$ **then**
            $g_{temp} = g(n) + distance(n, m)$
            if $m \notin \mathcal{O}$ or $g_{temp} < g(m)$ **then**
                $g(m) = g_{temp}$
                $f(m) = g(m) + h(m, e)$
                $m.parent \leftarrow n$
                $\mathcal{O} \leftarrow \mathcal{O} \cup \{m\}$
            end if
        end if
    end for
end while
Return failure

3.3.2 Savitsky-Golay Smoothing

The path generated by the A* algorithm is often discrete and contains sharp turns, especially in grid-based or rasterized maps. Since A* focuses solely on finding the shortest path without considering smoothness, the resulting trajectory may exhibit abrupt changes, discontinuities in control, and instability when executed by robots, drones, or autonomous vehicles. To address this issue, we apply Savitzky-Golay smoothing, which enhances path smoothness while maintaining optimality and avoiding significant deviations from the original route, thereby ensuring a more fluid and stable motion.

Savitzky-Golay smoothing (SG filtering) is a polynomial-based local smoothing method designed to reduce noise while preserving the trend, peaks, and higher-order derivative information of the signal. Compared to simple moving averages, SG filtering better retains the characteristics of the signal, making it particularly useful in spectral analysis, signal processing, and time series smoothing.

The core idea of SG smoothing is to achieve signal smoothness through local polynomial fitting while maintaining the overall trend and features. Given a window size W (typically an odd number), a k -th order polynomial is fitted using least squares within each sliding window:

$$P(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_k x^k \tag{17}$$

where $c_0, c_1, \ldots, c_k$ are the coefficients to be determined. The goal is to minimize the fitting error:

$$\sum_{j=-\frac{W-1}{2}}^{\frac{W-1}{2}} \left(y_{i+j} - P(x_j)\right)^2 \tag{18}$$

By solving the least squares equation:

$$\mathbf{Y} = \mathbf{XC} + \epsilon \tag{19}$$

yields the filtering weights a_j, and the final smoothed value is computed as:

$$\hat{y}_i = \sum j = -w^w a_j y_{i+j} \tag{20}$$

Applying Savitzky-Golay smoothing significantly improves the quality of the generated path, making it both near-optimal and more suitable for real-world execution. This ensures a smoother trajectory, reducing unnecessary oscillations and enhancing the feasibility of path-following applications in autonomous robot systems.

3.3.3 Implement *CreatePlan* ()

The implementation of CreatePlan() follows the same process as described earlier. Initially, it initializes PlannerNodes and a priority queue (openlist) to store nodes based on their cost. The function then converts the world coordinates of the start and goal positions into map coordinates, sets the cost parameters for the start node, and adds it to the open list.

During the main loop of the A* algorithm, the node with the lowest cost is extracted from the open list and marked as expanded. The function then explores its neighboring nodes, computes their respective costs, and updates the path accordingly. In this experiment, the heuristic estimation is based on the Euclidean distance, as the test environment involves free movement in a 2D plane. Since the Euclidean distance represents the shortest straight-line path between two points, it is well-suited for continuous spaces and provides an accurate cost estimation.

When expanding nodes, if a better path to a neighboring node is found, its cost is updated, and the node is added to the open list. Once the goal node is reached, the algorithm backtracks to reconstruct the path and generates a path message. Finally, the generated path undergoes Savitzky-Golay smoothing to enhance its smoothness before being returned as the final output.

4 Conclusion & Improvement

4.0.1 Conclusion

The project successfully integrates a Regulated Pure Pursuit controller, an A* path planner, and Savitzky-Golay smoothing to enhance autonomous navigation in static settings. Experimental evaluations confirm that the adaptive lookahead mechanism and heuristic-based speed regulation in the RPP algorithm significantly improve trajectory tracking, while the application of A* coupled with smoothing techniques yields continuous and feasible paths. These outcomes reflect a notable advancement in balancing tracking accuracy with stability, particularly in complex and high-curvature scenarios. Although the current system demonstrates robust performance under controlled conditions, further improvements are envisionedespecially in adapting to dynamic environments. Future research should explore the integration of advanced trajectory optimization strategies and predictive control methods, potentially incorporating machine learning to further elevate navigation performance and safety.

4.0.2 Improvement

A* is a widely used path planning algorithm that efficiently finds the shortest path in discrete space. However, in real-world applications, it has certain limitations, such as lack of smoothness in the generated paths and inability to adapt to dynamic environments. Recent

research has introduced various improvements to address these issues, including MINCO trajectory parameterization[4] and Signed Volume Swept Distance Field (SVSDF).[3]

MINCO (Minimum Control) was proposed by Wang Bo et al. in 2022 as an optimization-based approach to generate efficient and smooth trajectories. This method leverages polynomial parameterization to transform trajectory optimization into an unconstrained optimization problem, making it highly efficient for solving complex planning tasks.

SVSDF is a method for continuous collision-free trajectory optimization. Instead of checking discrete waypoints, it computes the signed distance field of the swept volume along a robots trajectory, ensuring safe clearance from obstacles throughout the entire motion. By formulating SVSDF computation as a generalized semi-infinite programming problem, this method eliminates the need for explicit surface reconstruction. SVSDF has been validated in complex environments and is applicable to robots with different dynamic properties, including rigid and deformable shapes, making it highly versatile for path planning applications.

5 Reference

References

[1] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[2] Steve Macenski et al. "Regulated pure pursuit for robot path tracking". In: *Auton. Robots* 47.6 (June 2023), pp. 685–694. ISSN: 0929-5593. DOI: 10.1007/s10514-023-10097-6. URL: https://doi.org/10.1007/s10514-023-10097-6.

[3] Jingping Wang et al. "Implicit Swept Volume SDF: Enabling Continuous Collision-Free Trajectory Generation for Arbitrary Shapes". In: *ACM Trans. Graph.* 43.4 (July 2024). ISSN: 0730-0301. DOI: 10.1145/3658181. URL: https://doi.org/10.1145/3658181.

[4] Zhepei Wang et al. "Geometrically Constrained Trajectory Optimization for Multicopters". In: *IEEE Transactions on Robotics* 38.5 (2022), pp. 3259–3278. DOI: 10.1109/TRO.2022.3160022.