Final report PDM project

Group 44

Bram Kalkman
5646731
bakalkman@student.tudelft.nl

Mats Kasse
5572789
m.kasse@student.tudelft.nl

Lars Emmer
5110165
lemmer@student.tudelft.nl

Jesper van der Meulen
5640520
jespermeulen@student.tudelft.nl

***Abstract*—This report describes motion planning for a mobile service robot operating in a dynamic restaurant environment. Different global path planners are combined with a local Model Predictive Control (MPC) planner, to compare how the global planners perform. As a global planner, Rapidly-exploring Random Trees (RRT) is used as a baseline, and is compared to an RRT* and A* planner. The robot model is a non-holonomic differential-drive robot. The different planners are evaluated in a simulation on different performance metrics, including: success rate, path length, computation-time and the goal reach time of the MPC following the computed path.**

I. Introduction

A. Overview of the Motion Planning Pipeline

For motion planning, RRT was chosen in combination with a linear MPC controller as a baseline. RRT was chosen as a global planner. MPC is used as a local planner, making sure that the robot can anticipate and avoid dynamic obstacles such as people or other robots. RRT is chosen as a baseline because it can efficiently generate feasible paths in continuous configuration spaces with low computational cost. A standard linear MPC was chosen. A non-linear MPC is not used because we do not have the computational capacity. Input noise is not taken into account, so opting for an MPPI is also not the logical choice.

B. Simulation Environment

Two main options for the simulation framework were evaluated. The first option is ROS2 in combination with the Gazebo simulation environment. This approach provides an end-to-end robotics pipeline, as ROS2 enables testing of perception, navigation, mapping, and control modules. This reflects real-world development and is therefore advantageous for validating the software stack that will eventually be deployed on a physical platform. However, this architecture introduces considerable complexity into the system.

Contribution statement: Jesper created the restaurant environment, contributed to the paper and made/presented the presentation. Lars worked on RRT and RRT*, retrieved results for these algorithms and contributed to the paper. Mats was responsible for the setup and maintenance of the GitHub codebase, developed and implemented the MPC controller, integrated the full end-to-end robotic pipeline, conducted the end-to-end experimental evaluation and contributed to the paper. Bram worked on the implementation and retrieved the results of the A* Algorithm and contributed to the paper

GenAI statement: For the code of this project, GenAI was used to debug and get quick solutions. Also for the collection of the results code was written with the help of GenAI.

The alternative approach is PyBullet integrated with OpenAI Gym. This offers a more research-oriented environment. PyBullet provides a lightweight, high-performance simulation environment and low-level access to kinematic and dynamic states, which allows rapid prototyping and testing of motion control, reinforcement learning, and trajectory optimization algorithms. Importantly, this setup does not require the setup of a full ROS2 stack, thereby reducing development time and system complexity.

Given the objective of this project and the evaluation of control strategies for a mobile robot equipped with a manipulator in a custom environment, the PyBullet + OpenAI Gym ecosystem is selected. This environment is fully programmable in Python, supports direct loading of native URDF robot models (1), and enables simulations suitable for algorithmic research and validation.

C. Robot Application and Use Case

For the robot platform, the *Albert Robot* will be used. The Albert Robot is a mobile robotic manipulator which has two interesting robotic challenges: Ground Navigation and Manipulation. For this project we not only wanted to explore the basic control actions of any robot without a clear function, but work on a realistic scenario that is close to what we might encounter in our further careers as roboticists. The Albert Robot fits this description as there are already existing solutions for this robot in the real world.

The intended use-case corresponds to an autonomous service delivery robot in a dynamic environment, specifically a restaurant setting. The robot must navigate through the dining area, deliver food and drinks, and perform manipulation tasks at the corresponding tables. This report will focus on the navigation steps of this scenario:

1) Autonomous navigation in a static environment.
2) Autonomous navigation in a dynamic environment with moving obstacles (e.g., people or other delivery robots).

II. Problem Formulation

A. Robot Model and Assumptions

The Albert Robot is a non-holonomic differential-drive mobile platform equipped with an onboard manipulator. A non-holonomic drive system has limited possible movement directions at any given moment; it cannot move sideways without first changing its orientation.

B. Equations of Motion

Following the assumptions below,

- No slip
- No lateral movement
- The longitudinal speeds of the wheels on the same side are equal

The equations of motion can be defined as a simple two wheeled differential drive by the following equations taken from Lecture 9, Slide 16 of Robot Dynamics and Control (2)

$$\dot{\xi}_I = R^T(\theta) \begin{bmatrix} 1 & 0 & -l \\ 1 & 0 & l \\ 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} r\dot{\varphi}_A \\ r\dot{\varphi}_B \\ 0 \end{bmatrix} \tag{1}$$

$$\dot{\xi}_R = R(\theta)\,\dot{\xi}_I = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{2}$$

where $\dot{\xi}_I$ = velocities and rotation in robot frame, $\dot{\xi}_R$ = velocities and rotation in global frame, r = wheel radius, $\dot{\varphi}$ = wheel angular velocity and l = length from wheel to middle of vehicle

C. State-Space Representation

From the equations of motion, the state space model can be derived.

As a control input we use v and w, the forward speed of the robot and the angular velocity of the wheels.

D. Configuration Space and Planning Problem

The workspace of the robot is the 2D ground. So $\mathcal{W} = \mathbb{R}^2$. The configuration space of the robot is defined as $\mathcal{C} = \mathbb{R}^2 \times S^1$. The variables are the x and y coordinates and the heading angle θ. This is because the Albert robot is non-holonomic and not all motions in the configuration space are directly feasible. So the global planning problem is to compute a collision-free path in $\mathcal{C}_{free}$, from start configuration to goal configuration.

III. Motion Planning Method[1]

A. Global Path Planning

We use RRT as a baseline global planner and compare this to RRT* and A*. Each global planner generates a reference path for the local MPC.

B. A*

A* is deterministic and uses grid search to create a path. This algorithm was chosen, because of the efficiency using a heuristic, the guarantee for optimal results and the ability to reuse the grid-map. The reuse of the grid-map is especially useful in environments where obstacles do not move frequently, like in a restaurant with tables and chairs.

Parameters: The computing-speed and quality of the path depend on the resolution of the grid.

Distance-to-obstacle cost: A distance-to-obstacle cost was added next to the heuristic, creating paths which are more obstacle-avoiding and thereby minimizing the chance of collision. The effect of this cost is visualized in Figure 1.

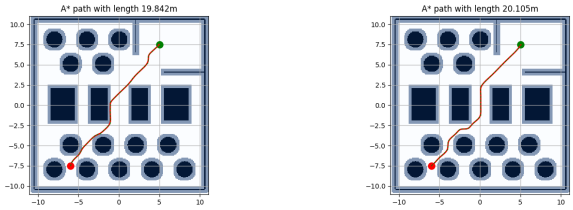

Fig. 1: Comparison of A* algorithm with (right) and without (left) obstacle avoidance cost.

C. RRT

RRT (Rapidly exploring Random Tree) is a sampling based path planning algorithm in continuous space. Which can create a feasible path quickly. For RRT, when using a small enough step size for the map, the probability of finding a path approaches one as max iterations are increased. This path however will never be optimal. For this standard version of RRT, the most important parameters are step size, max iterations and goal bias. Step size has to be small enough to be able to navigate the tightest corners in the map. A smaller step size means paths are more precise but more iterations are needed to produce a path, increasing computational time. Goal bias is used to tune how aggressively the tree expands in the direction of the finish point. High bias means a path is more quickly found in less complex environments, but can also cause the algorithm to find less straightforward routes less fast.

D. RRT*

RRT*'s most important addition to RRT is that the tree it produces can rewire itself during the process. If a new node that is added makes it possible to connect existing nodes through a shorter route, it will reroute the tree through itself to do so. Unlike standard RRT, when increasing the amount of iterations for RRT* to infinity, the path will converge to the optimal path. To determine how many nodes RRT* has to check for possible faster routes, the rewiring radius parameter is added. Using a higher rewiring radius means the algorithm will check more nodes and thus be 'smarter' in rewiring itself. But a higher rewiring radius also drastically increases computation time as the number of nodes increases.

E. Greedy pruning

For both RRT and RRT*, increasing step size and iteration amount results in smoother paths. But although the resulting paths also look shorter at first glance, when zooming in one

[1]For the implementation of the A* algorithm, the PythonRobotics (3) Toolbox was used. This was altered to accommodate an index grid-map instead of coordinates and to take into account the distance-to-obstacle cost. For RRT* and RRT no other libraries or toolboxes were used. The OSQP library was used as the quadratic programming solver, PyBullet and Gymnasium for physics-based simulation, Shapely for geometric operations.

can see that adding more and more short edges actually adds to the length of the path (Coastline Paradox). To address this problem, we added greedy pruning as a final step of each path planner. This pruning algorithm checks for pairs of nodes if the nodes in between them can be removed and replaced by a straight line without causing a collision. This resulted in significantly shorter paths. Figure 2 shows RRT paths with and without pruning.

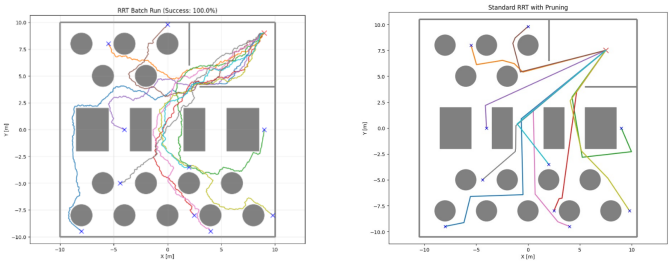

Fig. 2: Comparison of RRT paths with and without pruning.

F. System Model and Linearization Strategy

The control architecture consists of:

- a nonlinear unicycle model to represent the true system dynamics
- a linearized model within the MPC optimization at each control step
- a convex Quadratic Program (QP) solver for real-time feasibility

Nonlinearity of the System Model: The differential-drive robot is a nonlinear system due to trigonometric parts in the dynamics. It is modeled using unicycle kinematics with velocity-level control. The system state, x, and the input vector, u, are defined as:

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix}, \qquad u = \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{3}$$

$$\dot{p}_x = v\cos\theta, \qquad \dot{p}_y = v\sin\theta, \qquad \dot{\theta} = \omega$$

where p_x and p_y denote the robot position in the global reference frame, θ represents the heading angle, v is the forward linear velocity and ω is the yaw rate.
This choice reflects the actuation structure of a differential-drive robot, in which motion is controlled directly through translational and rotational velocities.

The discrete-time unicycle dynamics with sampling time T_s are given by:

$$p_{x,k+1} = p_{x,k} + T_s\, v_k \cos\theta_k, \quad p_{y,k+1} = p_{y,k} + T_s\, v_k \sin\theta_k,$$
$$\theta_{k+1} = \theta_k + T_s\, \omega_k.$$

The discrete-time system dynamics are expressed as:

$$x_{k+1} = f(x_k, u_k),$$

where $f(\cdot)$ is the nonlinear function.

Linearization for Linear MPC: To achieve feasibility, the nonlinear system model is linearized at each control step around the current operating point $(\bar{x}_k, \bar{u}_k)$:

$$x_{k+1} \approx A_k x_k + B_k u_k + c_k, \quad A_k = \left.\frac{\partial f}{\partial x}\right|_{(\bar{x}_k,\bar{u}_k)}, \quad B_k = \left.\frac{\partial f}{\partial u}\right|_{(\bar{x}_k,\bar{u}_k)}. \tag{4}$$

The term c_k compensates for linearization offsets and ensures local accuracy of the approximation.
After linearization, the MPC problem becomes a convex Quadratic Program which has benefits:

- global optimality of the solution,
- predictable iterative behavior,
- fast convergence optimal for real-time control.

G. MPC Formulation

Given a reference trajectory $\{x_k^{\text{ref}}\}_{k=0}^{N}$, the MPC objective is to avoid obstacles, minimize the tracking error, control effort, and to ensure stability and feasibility. The quadratic cost over the horizon N is:

$$J = \sum_{k=0}^{N-1} \left[(x_k - x_k^{\text{ref}})^\top Q (x_k - x_k^{\text{ref}}) + u_k^\top R u_k\right] + (x_N - x_N^{\text{ref}})^\top P (x_N - x_N^{\text{ref}}),$$

where $Q \geq 0$ is the weighting matrix that penalizes the state, $R > 0$ is the weighting matrix that penalizes control input effort, and $P \geq 0$ is the terminal cost weighting matrix.

There are also constraints in the system. These constraints guarantee physical, safety, environmental, and control constraints.

H. MPC constraints

Obstacle avoidance is formulated as a soft constraint in order to guarantee feasibility of the MPC problem under conflicting objectives such as tight environments, short prediction horizons, or aggressive reference tracking. Slack variables are introduced to allow controlled constraint violation.

Obstacle avoidance is implemented using linearized tangent half-space constraints around the nominal trajectory (4). For each obstacle prediction o_k at time step k, a separating hyperplane is constructed using the outward normal vector
where $\bar{p}_k$ denotes the linearization point of the robot position.

The collision avoidance constraint is formulated as

$$n_k^\top p_k \geq n_k^\top o_k + r_{\text{safe}} - s_k, \quad s_k \geq 0, \quad n_k = \frac{\bar{p}_k - o_k}{\|\bar{p}_k - o_k\|}, \tag{5}$$

where p_k is the robot position, r_{safe} is the required safety radius, and s_k is a slack variable allowing controlled constraint violation.

To discourage constraint violation, the slack variables are penalized in the MPC objective function where $\lambda_s > 0$ is a weighting parameter that determines the cost of violating the obstacle constraint (see equation 6).

As a result, the MPC remains solvable even in narrow passages or under conflicting objectives, while still strongly preferring collision-free trajectories (4).

The velocity constraint ensures the commanded velocities to remain within the capabilities of the robot and within safe operational limits. A high velocity could lead to unrealistic or unstable behavior. The velocity constraint is a hard constraint.

$$0 \leq v_k \leq v_{\max}, \qquad |\omega_k| \leq \omega_{\max}. \tag{6}$$

An overview of the cost function and the constraints are presented below:

$$\begin{aligned}
\min_{\{x_k,u_k\}_{k=0}^{N}} \quad & \sum_{k=0}^{N-1} \left[(x_k - x_k^{\text{ref}})^\top Q (x_k - x_k^{\text{ref}}) + u_k^\top R u_k + \lambda_s s_k^2 \right] \\
& + (x_N - x_N^{\text{ref}})^\top P (x_N - x_N^{\text{ref}}) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k), \qquad k = 0, \ldots, N-1, \\
& |\omega_k| \leq \omega_{\max}, \\
& 0 \leq v_k \leq v_{\max}, \\
& n_{j,k}^\top p_k \geq n_{j,k}^\top o_{j,k} + r_{\text{safe}} - s_{j,k}, \\
& s_{j,k} \geq 0, \qquad \forall j, k, \qquad s_{j,k} \in \mathbb{R}_{\geq 0} \\
& x_0 = x_{\text{init}}.
\end{aligned} \tag{7}$$

I. Solver Selection

Given the convex QP formulation of the linearized MPC problem, the OSQP solver is selected. OSQP is an ADMM-based[1] first-order solver designed for large-scale, sparse convex QPs and efficiently exploits problem structure by reusing matrices between MPC iterations. This makes it well suited for MPC problems with state, input, and slack variables.

Furthermore, OSQP supports warm starting, allowing each MPC iteration to be initialized from the previous solution, which accelerates convergence and enables reliable real-time performance in dynamic environments (5).

IV. Results

This section describes the simulation setup, testing methodology, and evaluation metrics used to assess the navigation performance of the mobile robot in a restaurant environment.

[1]ADMM (Alternating Direction Method of Multipliers)

A. Simulation Setup

All experiments are conducted in a simulated restaurant environment implemented using the PyBullet physics engine. The environment models a restaurant layout consisting of static and dynamic obstacles such as walls, tables and people, representing realistic navigation constraints for a service robot.

The robot is initialized at a fixed starting location corresponding to the kitchen area. Goal locations are positioned at tables throughout the restaurant. The simulation provides access to ground-truth robot states, including position, orientation, velocity, and collision information, which are used to compute the evaluation metrics described in Section IV-C.

B. Testing Method

The performance of a global planner highly depends on its settings. To make a fair end-to-end comparison of the different pipelines the settings of the different global planners should be approximately the same. So first the global planners performance is isolated and evaluated. In this way settings are determined that result in roughly the same computation time for the global planner. This global planner variables are then used in computation with an MPC for and end-to-end comparison. For each global planner 10 different endpoints are tested for 10 different algorithm variables. This will give a trend of how computation time vs path length is correlated.

C. Evaluation Metrics

To evaluate the navigation performance of the mobile robot acting as a waiter in a restaurant environment four different performance metrics are used. Two global planner metrics and two end-to-end metrics.

- **Path Length (global):** The total length of the planned path produced by the path planning algorithm. This metric allows comparison of planner optimality independent of execution effects.
- **Computation Time (global):** The average computation time required by the path planning algorithm per trial. This metric evaluates the suitability of the algorithm for real-time operation.
- **Success Rate (end-to-end):** The ratio of successful navigation trials to the total number of trials. This metric reflects the reliability and robustness of the navigation algorithm.
- **Goal Reach Time (end-to-end):** The elapsed time from the start of the trial until the robot reaches the goal. This metric is indicative of service efficiency and responsiveness.

D. Results global planner comparison

The global planners are tested for different settings. Every planner setting is evaluated at the same 10 goal positions. The

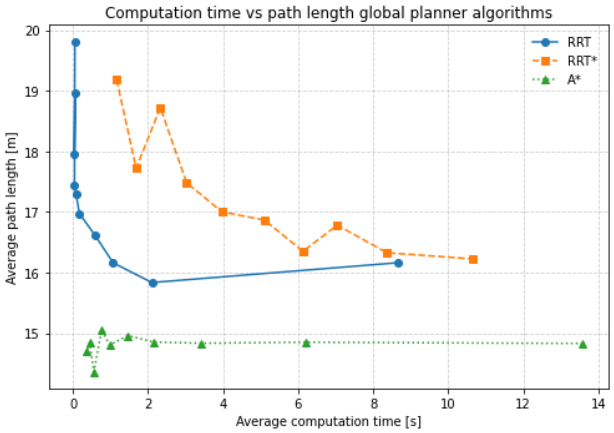

Fig. 3: Computation time (s) vs path length (m).

average path length and computation time is plotted for every setting in figure 3.

The settings that result in a computation time of roughly 2.2 seconds will be used in the end to end comparison. These settings are:

- **RRT:** Step size: 0.05. Max iterations: 15000. Goal bias: 5%.
- **RRT*:** Step size: 0.75. Iterations: 1500. Goal bias: 5%. Rewiring radius: 1.5
- **A*:** Resolution: 0.06

E. Results End-to-end comparison

To evaluate the performance of the global planners in a closed-loop navigation setting, an end-to-end simulation is performed in which the robot follows the paths generated by each planner using the same local MPC controller. All MPC parameters, including cost weights, horizon length, and constraint settings, are kept identical across all experiments to ensure a fair comparison between planners. The exact MPC configuration used in the experiments is available in the project repository (6).

A run is considered successful if the robot reaches the goal without colliding with either static or dynamic obstacles. If the robot fails to reach the goal within the predefined simulation time, the run is marked as unsuccessful and excluded from the average travel time. The reported Global Reach Time (GRT) therefore represents the average time required to reach the goal over successful runs only.

Each planner is evaluated on 10 different goal positions, with 10 repeated runs per goal, resulting in a total of 100 trials per planner. The results are presented below in table I.

Planning Algorithm	Success Rate	GRT* (s)
RRT	82.00%	16.40
RRT*	83.00%	15.47
A*	100.00%	15.04

TABLE I: End-to-End Navigation Performance of Global Planners with MPC Tracking.
*GRT (Global Reach Time) is the average simulation time

To provide qualitative insight into the closed-loop behavior of the system, a video showing representative simulation recordings for the A* planner combined with MPC is available at: https://www.youtube.com/watch?v=BRVX2FYVZRo

V. Discussion

The experimental results clearly show that A* is the most effective global planner for the considered restaurant navigation problem. As summarized in Table I, A* achieves a 100% success rate, outperforming both RRT (82%) and RRT* (83%) in the end-to-end tests.

This higher reliability can be explained both theoretically and practically. For a fixed grid resolution, A* is guaranteed to find the shortest path in the discretized configuration space. In contrast, RRT* only converges to optimal solutions asymptotically and RRT does not converge at all. This means that their solution quality depends strongly on the number of samples and thus on computation time.

RRT* performs slightly better than standard RRT in terms of goal reach time, but remains sub-optimal under the imposed iteration limit. As seen in Figure 3, within the allowed computation budget RRT* does not reach the same path optimality as A*, resulting in longer trajectories that are more difficult for the MPC to track. RRT, while not producing the shortest paths, remains the fastest planner in terms of initial solution generation.

The strength of RRT and RRT* lies in its random sampling of points in continuous space. This is a good characteristic when working with a large configuration space such as a robotic arm with multiple Degrees of Freedom, but for this workspace this feature does not work to its advantage.

An important consideration is that the MPC controller was tuned using A*-generated paths. This introduces a systematic bias in favor of A*. However, even accounting for this, since A* consistently scores better for both metrics, it seems to be the better option for a workspace of this size and dimension.

A. Planner improvement recommendations

Based on the observed weaknesses, the following extensions are recommended.

- **A*:** Extend the cost map with **time-dependent obstacle costs** to better account for moving people.
- **RRT*:** Implement an **adaptive rewiring radius** to improve early convergence while preserving real-time performance.
- **RRT:** Extend to **kinodynamic RRT** to generate trajectories that are easier for the MPC to track.

VI. Conclusion

For the considered restaurant service robot scenario, A* is the most appropriate solution. It provides the shortest paths, highest success rate, and lowest goal reach time, making it the most reliable and efficient planner when combined with MPC. However, RRT and RRT* are probably more suitable for bigger and more complex workspaces, where the deterministic approach of A* is not feasible.

References

[1] MicrocosmAI, "A review of nine physics engines for reinforcement learning research." Online article, Oct. 2024.

[2] B. Shyrokau, "Robot dynamics and control." Lecture slides (Lecture 9, slides 8 and 16), 2025. Fall 2025.

[3] A. Sakai, "Pythonrobotics." https://github.com/AtsushiSakai/PythonRobotics. GitHub repository.

[4] L. Ferranti, "Introduction to model predictive control part i." Lecture slides (Lecture 6, slides 48-51, 66, 70, 2025. Winter 2026.

[5] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, 2020.

[6] M. Kasse, B. Kalkman, L. Emmer, and J. van der Meulen, "Pdm project: Global path planning and mpc-based navigation." https://github.com/MatsKasse/PDM_project, 2025.