Implementation of Trajectory Planning and Control of a Non-holonomic Platform

I. Introduction

In this project, we focus on the trajectory planning and control of a non-holonomic robot. Non-holonomic systems pose challenges in motion planning and control due to their constraints. The objective is to develop an efficient trajectory optimization method and implement a controller for trajectory tracking, utilizing both nonlinear control and model predictive control (MPC). The system's kinematics, path planning strategies, and control laws are explored to achieve accurate tracking of the reference trajectory.

II. System Kinematics

The kinematic model of a non-holonomic system with state vector $(\phi, x, y)'$ and input vector $(v, \omega)'$ is used for motion planning and controller design in this project. The nonlinear transition function of the simplified non-holonomic canonical model is:

$$\begin{pmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \omega \\ v\cos(\phi) \\ v\sin(\phi) \end{pmatrix} \tag{1}$$

This model applies to three types of non-holonomic robots: unicycle, differential-drive, and car-like robots, each with different $(v, \omega)'$ control sets [1]. In this report, a unicycle robot is chosen due to its simple control constraints, where the control set is defined as $\mathcal{U} = \left\{(v, \omega)' \in \mathbb{R}^2 \;\middle|\; |v| \leq v_{\max}, |\omega| \leq \omega_{\max}\right\}$.

III. Graph-based Path Planning

A planned path provides an initial guess for nonlinear optimization, helping to avoid getting trapped in a local minimizer. The graph-based path planning algorithm uses a costmap, representing the environment as a graph where cost values between nodes indicate traversal difficulty. The algorithm involves creating the costmap and finding the path from the start node to the target node with the lowest cost.

In this project, both the Probabilistic Roadmaps (PRM) and Hybrid A* planners are explored for path planning. The PRM planner samples nodes in the obstacle-free space and only connects nodes when the edge does not intersect obstacles. [1] The Hybrid A* planner discretizes the obstacle-free space using a grid, searching for a kinematically feasible path with the lowest cost, applying different weights to straight and turning movements [2].

The PRM planner is implemented using `mobileRobotPRM` from the Robotics System Toolbox, with the number of sampling nodes set to 500. The Hybrid A* planner is implemented using `plannerHybridAStar` from the Navigation Toolbox. The grid resolution is set to 25 cells/m, and the forward and reverse costs are identical, with a minimum turning radius of 0.75 m. As trajectory optimization requires an initial guess with $N+1$ sets of states, where N is the finite horizon, the path obtained from the planner is interpolated using a cubic spline. The heading ϕ is calculated from the subsequent positions as $\phi_{i+1} = \arctan\left(\frac{y_{i+1}-y_i}{x_{i+1}-x_i}\right)$.

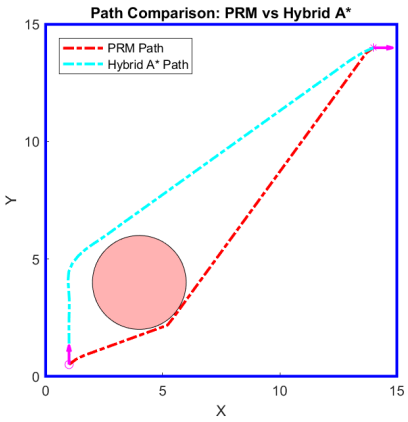


Fig. 1: Results of PRM and Hybrid A*; the circle and star indicate the start and target positions.

Figure 1 shows the paths generated by both planners with one obstacle exists. It can be observed that the PRM planner always finds the shortest path without considering non-holonomic constraints, which may lead to infeasible trajectories or sub-optimal results. Therefore, the Hybrid A* planner is chosen as a better option for path planning.

IV. Trajectory Optimisation

The path from the planner does not account for time or system dynamics. Therefore, a nonlinear optimization is formulated to derive the trajectory of both states and inputs by solving an optimal control problem over a finite horizon $N = T_f/T_s$, where $T_f = 5s$ is the total simulation time and $T_s = 0.1s$ is the sampling time. The optimal control problem is defined as:

$$\min_{\mathbf{x},\mathbf{u}} V(\mathbf{x}, \mathbf{u}) \tag{2}$$

$$\text{s.t.} \quad x_0 = x_{\text{start}}, \tag{3}$$

$$x_{i+1} = f(x_i, u_i), \quad \text{for } i = 0, 1, \ldots, N, \tag{4}$$

$$x_i \in \mathcal{C}_{\text{free}}, \quad \text{for } i = 0, 1, \ldots, N, \tag{5}$$

$$u_i \in \mathcal{U}, \quad \text{for } i = 0, 1, \ldots, N, \tag{6}$$

$$x_N \in \mathcal{X}_{\text{target}}. \tag{7}$$

where $\mathbf{x} = (x'_0, x'_1, \ldots, x'_N)'$ and $\mathbf{u} = (u'_0, u'_1, \ldots, u'_N)'$, x_{start} is the initial state, $\mathcal{C}_{\text{free}}$ represents the obstacle-free space, and $\mathcal{X}_{\text{target}}$ is a small region around the target. The function $f(x_i, u_i)$ is the discretized transition function using the midpoint rule from the system kinematics in Equation 1.

A. Cost Function

A quadratic cost function is chosen for its differentiability and ability to penalize large errors effectively. The cost function is expressed as:

$$V(\mathbf{x}, \mathbf{u}) = \sum_{i=0}^{N} (x_i - x_{\text{target}})^T Q (x_i - x_{\text{target}}) + \sum_{i=0}^{N-1} u_i^T R u_i$$

where $Q = \text{diag}(10, 10, 10)$ and $R = \text{diag}(1, 5)$. The cost on w is higher than on v, making the non-holonomic robot less prone to excessive turning actions.

B. Constraints

1) Box Constraint: The robot is constrained to operate within a box-shaped space using vector projection, as shown in Figure 2.

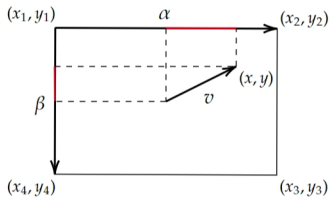


Fig. 2: The projection method used to define box constraints

For a point inside the rectangle, the projection of the vector from the geometric center of the rectangle to the point, onto each side of the rectangle, must be smaller than half the length of the respective side. Thus, the box constraint is written as:

$$\frac{|v \cdot s|}{||s||} \leq \frac{||s||}{2}, \quad -\frac{1}{2}||s||^2 \leq v \cdot s \leq \frac{1}{2}||s||^2, \quad \forall s \in \{\alpha, \beta\}$$

where v, α, and β are vectors illustrated in Figure 2. This method of handling the box constraint avoids singularity issues.

2) Ellipse Constraint: Elliptical obstacles are arranged within the box-shaped space. To avoid these obstacles, a nonlinear constraint is defined as:

$$1 - \frac{(x - x_c)^2}{a^2} - \frac{(y - y_c)^2}{b^2} \leq 0$$

where x_c and y_c are the center coordinates of the ellipse, and a and b are the semi-major and semi-minor axes of the ellipse, respectively. To increase robustness, the constraint is slightly tightened to prevent collisions.

C. Shooting Methods

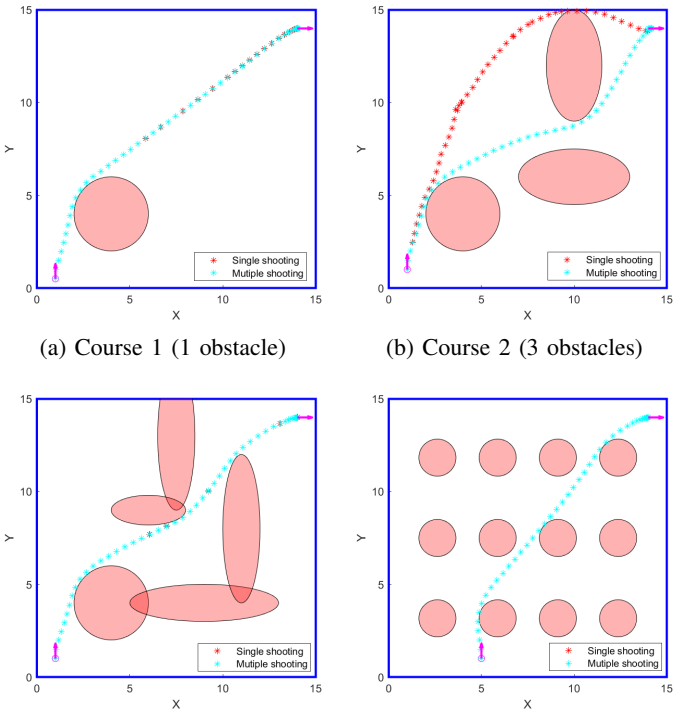
The nonlinear optimization problem is formulated using both direct single-shooting and direct multiple-shooting methods [3]. The single-shooting method forms a condensed optimization problem, where the decision variable is $w = \mathbf{u}$. In this case, $\mathbf{x}$ is a function of $\mathbf{u}$, obtained by solving an initial value problem with the initial condition $x_0 = x_{\text{start}}$.

In contrast, the multiple-shooting method formulates a sparse optimization problem with more decision variables, $w = (\mathbf{u}', \mathbf{x}')'$. The system kinematics are imposed as a nonlinear equality constraint: $x_{i+1} - f(x_i, u_i) = 0$.

The MATLAB function `fmincon` from the Optimization Toolbox is used to solve the nonlinear optimization problems formulated by both methods. The `active-set` algorithm is chosen for its robust performance in this application. The results of the trajectory optimization are shown in Figure 3, and the computational times are presented in Figure 4.

The single-shooting method is less robust and more sensitive to the accumulation of nonlinearities [3], resulting in failed trajectory planning in some courses, as shown in Figure 3b. In contrast, the multiple-shooting method passed all the tests in the simulation and is preferred due to its better initial guess setup, making it the more suitable method for this application.

Regarding computational time, each method outperformed the other in different courses, likely due to the characteristics of the cost function and initial guess.



(a) Course 1 (1 obstacle) (b) Course 2 (3 obstacles)

(c) Course 3 (5 obstacles) (d) Course 4 (12 obstacles)

Fig. 3: Results of the trajectory optimisation

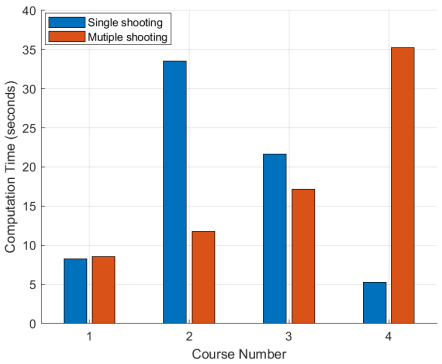


Fig. 4: Computational time of the trajectory optimisation

V. Controller Design

The robot is designed to track the planned reference trajectory using both nonlinear control and MPC.

A. Nonlinear Control

Given the state trajectory x_{ref} and input trajectory u_{ref}, the error state in the Frenet frame along the reference trajectory is defined as follows:

$$\begin{pmatrix} \phi_e \\ x_e \\ y_e \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_{\text{ref}} & \sin\phi_{\text{ref}} \\ 0 & -\sin\phi_{\text{ref}} & \cos\phi_{\text{ref}} \end{pmatrix} \begin{pmatrix} \phi - \phi_{\text{ref}} \\ x - x_{\text{ref}} \\ y - y_{\text{ref}} \end{pmatrix}, \quad (8)$$

For k_1, k_2, and k_3, the nonlinear feedforward and feedback control law is given by:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} (v_{\text{ref}} - k_1|v_{\text{ref}}|(x_e + y_e \tan\phi_e)) / \cos\phi_e \\ \omega_{\text{ref}} - (k_2 v_{\text{ref}} y_e + k_3|v_{\text{ref}}| \tan\phi_e)\cos^2\phi_e \end{pmatrix} \quad (9)$$

This control law is proven to ensure stable tracking when $|\phi_e| < \pi/2$ [1], which can be demonstrated using a Lyapunov function [4].

B. Model Predictive Control

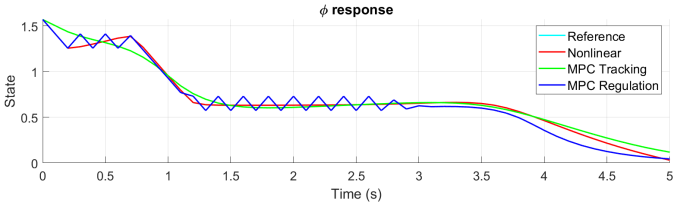
MPC is widely used for its robustness and reliability, especially in the presence of constraints. The MATLAB function `nlmpcmove` is utilized to implement a time-varying linear MPC with receding horizon for both tracking the planned reference trajectory and regulating the system to the target state. The nonlinear optimization setup is identical to the trajectory optimization discussed in the previous section.

Since MPC solves a similar optimization problem at each time step, the solutions are expected to be consistent if disturbances are not significant. This justifies using the previous solution as an initial guess for the next time step. For the receding horizon, the input is updated by removing u_1 and appending a copy of u_{N-1}.

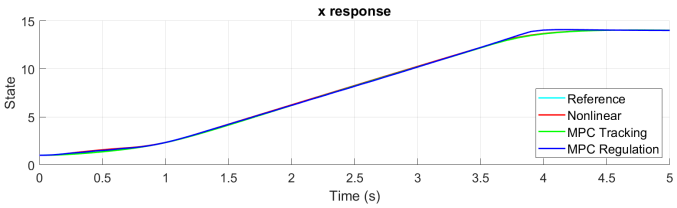
C. Results of Simulation

The system response of the non-holonomic robot is shown in Figure 5. It can be observed that the system tracks the reference trajectories of x and y well. However, for the ϕ response, the MPC tracking design demonstrates slower convergence to the target, and the MPC regulation exhibits oscillations, possibly caused by numerical errors during linearization.

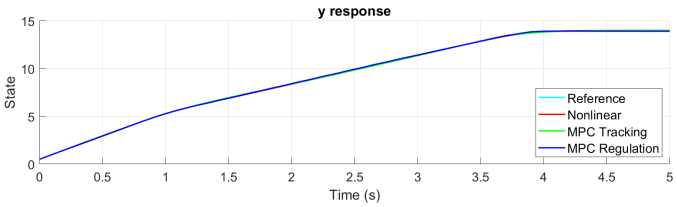
The nonlinear controller perfectly converges to the reference trajectory, as no disturbances were introduced in the simulation. This indicates that the optimized trajectory performs exceptionally well.



(a) Response of ϕ



(b) Response of x



(c) Response of y

Fig. 5: System response results

VI. Conclusion

In conclusion, the combination of trajectory optimization and control design for a non-holonomic robot proves effective in generating feasible paths and ensuring stable trajectory tracking. Future work could explore handling larger disturbances and further improving computational efficiency in real-time applications.

References

[1] Lynch KM, Park FC. Modern robotics : mechanics, planning, and control. University Press; 2017.
[2] Kurzer K. Path Planning in Unstructured Environments : A Real-time Hybrid A* Implementation for Fast and Deterministic Path Generation for the KTH Research Concept Vehicle; 2016.
[3] Rawlings JB, Mayne DQ, Diehl MM. Model predictive control theory, computation, and design. Madison, Wisconsin Nob Hill Publishing; 2017.
[4] Kanayama Y, Kimura Y, Miyazaki F, Noguchi T. A stable tracking control method for a non-holonomic mobile robot. 2002 12.