MPC-Based Moving Obstacle Avoidance with Prediction Uncertainty

Aayush Dulal

***Abstract*— The problem of trajectory tracking for mobile robots in the presence of moving obstacles is evident in applications such as self-driving cars. Avoiding a moving obstacle is already a complicated task for any scale of operation. The problem worsens when we consider that the obstacle's position is not exactly known, which is the case in most real-life scenarios. Most solutions to the problem of an uncertain moving obstacle avoidance just consider the expected value of the obstacle's exact position. In this report, a novel approach to solving this problem is implemented. The distance between the distribution of the obstacle's position and the robot's position is leveraged, instead of only considering the mean obstacle position.**

I. INTRODUCTION

Dynamic programming (DP) is the process of finding the optimal solution in the form of the current state of the system. The mathematical implementation of dynamic programming is driven by the principle of optimality, which states, "An optimal policy has the property that Whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy about the state resulting from the first decisions." [1] [2]

Model Predictive Control (MPC) is widely used for real-time control of systems with constraints because it optimizes system behavior over a finite prediction horizon while explicitly accounting for system dynamics and input limitations [3]. One important application of MPC in robotics is trajectory planning and collision avoidance in environments containing moving obstacles [4]. A major challenge in dynamic environments is that the future motion of obstacles cannot be known exactly. Traditional obstacle avoidance methods often assume that obstacle positions are known precisely or remain static. However, in realistic scenarios, obstacles move, and their future motion must be predicted with uncertainty. Ignoring this uncertainty may lead to overly aggressive trajectories or potential collisions. This project implements MPC for safe navigation in the presence of moving obstacles whose future motion is uncertain. The controller incorporates predictions of obstacle motion together with a representation of prediction uncertainty to ensure that the robot maintains a safe distance from regions where obstacles are likely to be present. Several studies have investigated MPC-based approaches for obstacle avoidance in dynamic environments. [5] proposes a trajectory planning framework that incorporates prediction uncertainty using probabilistic constraints based on the Mahalanobis distance. This paper demonstrates that incorporating uncertainty into the MPC formulation leads to safer and smoother trajectories compared to deterministic methods. Other works have applied MPC to mobile robot navigation and path tracking using online linearization [6]. Simplified vehicle models, such as the unicycle or kinematic bicycle models, are frequently used. These models allow efficient optimization while still capturing the essential motion constraints of mobile robots.

II. PROBLEM INTRODUCTION

A path can be considered as a series of waypoints that the robot ideally takes. In many cases, the path will account for static obstacles present from the starting position to the end position of the robot. This path is used by a high-level controller that generates a trajectory given the current state of the system. Then, a low-level controller directly controls the actuators so that the robot follows the trajectory generated. The trajectory generation for the robot is an optimal control problem that can be solved using dynamic programming. In the discrete dynamic algorithm, the terminal cost J_T is initialized, and the optimal control problem is solved for the previous time step to go to the current state position. In the presence of some disturbance w, optimal control π corresponding to the optimal expected cost J_l is evaluated recursively from time step $l = T$ to 1 where T is the length of the prediction horizon.

$$J_T(x_T) = g_T(x_T)$$

$$J_l(x_l) = \min_{u_l} \mathbf{E}_{w_l}\left[g_l(x_l, u_l, w_l) + J_{l+1}(f_l(x_l, u_l, w_l))\right]$$

$$\pi_l^*(x_l) = \arg\min_{u_l} J_l(x_l)$$

The dynamic programming algorithm has a closed-form solution if the system happens to be unconstrained and the dynamics of the system are linear, considering the cost g to be quadratic. In other words, the solution to a discrete dynamic programming algorithm with a linear quadratic (LQ) system is a closed, algebraic equation in the form of $u = f(x)$ that gives the optimal control given a certain state. This closed-form algebraic solution is called the Discrete-time Riccati Equation (DARE). Hence, in theory, robot control with no obstacles in its feasible space is as easy as solving the dynamic programming algorithm shown above. Likewise, if the robot has linear dynamics with no constraints, the DARE can be solved to find the optimal control.

The dynamic programming algorithm becomes intractable in many cases when the domain becomes continuous instead of a discrete domain, because in theory, there could be infinitely many states that a system can take. But for an LQ system, the DP algorithm still has a closed algebraic solution known as the Algebraic Riccati Equation (ARE). The implementation of the DP algorithm becomes challenging outside

LQ dynamics, given a continuous domain. This difficulty is more pronounced when constraints must be enforced to find the optimal control. Even with an LQ dynamics, with the inclusion of constraints, the solution given by ARE ceases to be optimal, and the solution instead becomes piece-wise linear with a piece-wise quadratic locally optimal cost.
In the context of trajectory tracking of a robot, any obstacles in the robot's path make the feasible space of the robot non-convex. Likewise, to enforce this non-convex feasible space, non-convex constraints need to be enforced in addition to linear constraints inherent in any system. Therefore, the LQ setting quickly breaks, and the DP implementation in the continuous domain becomes intractable. As a result, approximate methods must be deployed to approximate the global optimal control solution found using DP.

A. Model predictive Control

One of the most popular methods of approximating the DP optimal control solution is reformulating the DP in the form of the current system state. This reformulation involves solving an optimization problem to determine the next N optimal control inputs, where N denotes the prediction horizon length. From this sequence of N optimal inputs, only the first control action is applied to the system. At the subsequent time step, the prediction horizon shifts forward by one step, and the entire optimization process is repeated. This receding-horizon strategy is known as model predictive control (MPC).

The most important part of model predictive controllers is the optimization problem being solved at every time step. The type of the problem and the time required to solve the problem completely determine whether the controller can be used in real-life applications or not. For an application mimicking robot trajectory tracking without any obstacles, the optimization problem will almost always be a convex problem.

$$J_0^*(x_0) = \min_{\{u_k\}_{k=0}^{N-1}} \left[\sum_{k=0}^{N-1} \left(x_k^\top Q x_k + u_k^\top R u_k \right) + x_N^\top P x_N \right]$$

$$\begin{aligned} \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, \quad k \in \{0, 1, \dots, N-1\} \\ & x_0 = x(0) \\ & u_k \in \mathcal{U}(x_k), \quad k \in \{0, 1, \dots, N-1\} \end{aligned}$$

A great feature about MPC is that the problem does not have to be convex or linear. Non-linear constraints can be enforced, which can make the problem non-convex. Consider a robot with state $x \in \mathbb{R}^n$, input $u \in \mathbb{R}^m$, and nonlinear discrete-time dynamics

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

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $k \geq \mathbb{N}_0$ denotes the current discrete time.

Over a prediction horizon $N \in \mathbb{N}_{>0}$, define the state and input trajectories as

$$X_k = (x_k^\top, x_{k+1}^\top, \dots, x_{k+N}^\top)^\top \in \mathbb{R}^{nN},$$

$$U_k = (u_k^\top, u_{k+1}^\top, \dots, u_{k+N-1}^\top)^\top \in \mathbb{R}^{mN}.$$

The general Model Predictive Control (MPC) problem for trajectory planning can then be written as:

$$\begin{aligned} \min_{\{U_k, X_k\}} \quad & f_0(X_k, U_k) && \text{(1a)} \\ \text{s.t.} \quad & x_{k+i+1} = f(x_{k+i}, u_{k+i}) && \text{(1b)} \\ & x_{k+i} \in X_{k+i} && \text{(1c)} \\ & u_{k+i-1} \in U_{k+i-1} && \text{(1d)} \end{aligned}$$

The model of the system shown in $1a$ can be linear or non-linear. Likewise, other constraints can be added, but the core structure of an MPC loop will stay the same. Hence, when obstacles are present in the robot's path, the robot's feasible space becomes non-convex, and while this makes the DP extremely difficult to implement, the MPC is fairly simple to implement. In addition, the optimization problem is solved online at each time step, enabling MPC to react to external disturbances.

B. Problem formulation

In a two-dimensional setting, robot workspace $\mathcal{W} \subset \mathbb{R}^2$, where only the position

$$p_{k+i} = (x_{k+i}, y_{k+i})^\top \in \mathcal{W}$$

as a part of the state

$$x_{k+i} = (\cdots, p_{k+i}, \cdots)^\top \in \mathbb{R}^n$$

is restricted. The other states include velocities in both x and y directions, but these states are usually not a part of the decision variable set. If the problem includes obstacle avoidance, the feasible position state space becomes

$$X_{k+i+1} = \left\{ x_{k+i} \in \mathbb{R}^n \;\middle|\; p_{k+i} \in \left\{ S^C \cap D_{k+i}^C \right\} \right\} \qquad (2)$$

where S describes the set of static obstacles and D_{k+i} describes the set of dynamic obstacles at time $k+i$. Hence, the feasible space becomes non-convex. For static obstacles that can be represented as polyhedra, their boundaries can be encoded as hyperplanes, which in turn can be represented by linear constraints. Because of the special nature of the obstacles, for the static obstacle-avoidance problem, binary variables can be introduced into the optimization problem, converting a non-convex optimization problem into a convex mixed-integer program. In theory, this procedure can be extended to moving obstacle avoidance as well by solving a different mixed integer problem every time step.
A major hurdle in applying these ideas in real-life applications is that the robot must know the exact position of the obstacle to solve for optimal control. However, this is not usually the case in real-life. There will be noise in the sensors; the sensors themselves might be some approximation, and there will be a lag in signal transmission. This project focuses on the implementation of the theory established for it. More specifically, the project implements a trajectory tracking for moving obstacle avoidance by a robot given the distribution of the obstacle's position. Considering the difficulty in applying the DP for this purpose, an MPC is

used to approximate the optimal solution.

The project considers linear dynamics for the robot, and the distribution of the obstacle's position is considered to be normal. If the obstacle's exact position is available, a good candidate for the cost function would include a term penalizing the distance between the center of masses of the robot and the obstacle. Likewise, a region around the obstacle can be created, and this no-go region for the robot can be established by encoding the region as a non-convex and non-linear constraint in the optimization problem. When a distribution of the obstacle's position is known instead, the Mahalanobis distance can be used to quantify the proximity. The Mahalanobis distance is used to evaluate how far a point is from a distribution, which is exactly the metric of interest for this project as well. The Mahalanobis distance d_M of a point x to a normal distribution $\mathcal{N}(\mu, \Sigma)$ is defined as

$$d_M^2(x; \mathcal{N}(\mu, \Sigma)) = (x - \mu)^T \Sigma^{-1} (x - \mu)$$

The next step is to use the Mahalanobis distance as a metric to create a set around the obstacle that will act as the infeasible region for the robot when solving for optimal control. Mathematically, $\mathbb{D}$ is found and used as the set the robot will try to avoid, where d_M^2 is the Mahalanobis distance.

$$\mathbb{D} = \{\mathbf{x} \in \mathbb{R}^n \mid d_M < s\}$$

As a result of the Mahalanobis distance being chi-squared distributed, the resulting set $\mathbb{D}$ becomes an ellipse. An additional padding can also be done to the ellipse by increasing the radii of the ellipse. This is clear by writing the eigenvalue decomposition of the Mahalanobis distance as

$$d_M^2(\mathbf{x}; \mathcal{N}(\mu, \Sigma)) = (\mathbf{x} - \mu)^T R \Lambda^{-1} R^{-1} (\mathbf{x} - \mu) \tag{3a}$$

$$= (\mathbf{x} - \mu)^T R \Lambda^{-1} R^T (\mathbf{x} - \mu) \tag{3b}$$

where Λ is a diagonal matrix with Σ's eigenvalues λ_1 and λ_2 as its diagonal entries. R's columns are Σ's eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ with length 1 and $R^T = R^{-1}$, since R is orthogonal. Using the coordinate transformation $\mathbf{x}' = R^T(\mathbf{x} - \mu)$ and (5), $d_M^2 < s^2$ can be expressed based on (4) as

$$\mathbf{x}'^T \Lambda^{-1} \mathbf{x}' < s^2 \tag{4a}$$

$$\left(\frac{\mathbf{x}_1'}{s\sqrt{\lambda_1}}\right)^2 + \left(\frac{\mathbf{x}_2'}{s\sqrt{\lambda_2}}\right)^2 < 1 \tag{4b}$$

to describe the ellipse in a coordinate frame which axes align with the axes of the ellipse and whose origin is in μ. Equation (6b) shows that the ellipse's axes have the lengths $s\sqrt{\lambda_1}$ and $s\sqrt{\lambda_2}$, which can be enlarged by redefining Λ^{-1} as

$$\Lambda_{d,k+i}^{-1} = \text{diag}\left\{\frac{1/\left(s\sqrt{\lambda_{d,1,k+i}} + r + r_d\right)^2}{1/\left(s\sqrt{\lambda_{d,2,k+i}} + r + r_d\right)^2}\right\} \tag{5a}$$

where r and r_d are the radii of the robot and the obstacle. Hence, the size of the ellipse depends on the variable s and the eigenvalues of the covariance matrix of the obstacle position. With

$$R_{d,k+i} = [v_{d,1,k+i} \quad v_{d,2,k+i}] \tag{6a}$$

$$\Delta p_{d,k+i} = p_{k+i} - \mu_{d,k+i} \tag{6b}$$

The set $\mathbb{D}$ becomes

$$\mathbb{D}_{d,k+i} = \{p_{k+i} \in \mathbb{R}^2 \mid \Delta p_{d,k+i}^T R_{d,k+i} \Lambda_{d,k+i}^{-1} R_{d,k+i}^T \Delta p_{d,k+i} < 1\} \tag{7a}$$

The constraint that enforces $7a$ as the no-go region for the robot is as follows

$$\mathbb{D}_d^C = \{p_{k+i} \in \mathbb{R}^2 \mid \Delta p_{d,k+i}^T R_{d,k+i} \Lambda_{d,k+i}^{-1} R_{d,k+i} \geq 1\} \tag{7b}$$

The variable s is crucial in defining the robot's feasible space. A viable approach is to use the equation $7b$ as one of the inequality constraints, with s fixed. This is equivalent to enforcing the robot to never come close to the obstacle's vicinity within some distance. Taking this approach makes the inequality constraint a hard one, and the problem might become infeasible at some point. This is especially a danger if model disturbance is also considered. Hence, instead of a fixed s, s is made an additional decision variable and the optimization problem chooses the value. s_{ref} is chosen to be the reference for the variable s, and their difference will add to the cost at every optimization step. As a result, the inequality constraint softens, and the optimal size of the no-go region is evaluated with both close and far interactions penalized. Hence, the final optimization problem that is solved is as follows.

The final optimization problem that is solved at every time step is shown in equation 8.

$$\underset{\{U_k, X_k, s\}}{\text{minimize}} \quad \sum_{i=1}^{N} (x_{k+i} - x_{\text{ref},k+i})^T Q (x_{k+i} - x_{\text{ref},k+i}) + \sum_{i=1}^{N} u_{k+i-1}^T P u_{k+i-1} + (s - s_{\text{ref}})^2$$

(8a) subject to

$$x_{k+i+1} = A x_{k+i} + B u_{k+i} \tag{8b}$$

$$\Delta p_{d,k+i}^T R_{d,k+i} \Lambda(s)_{d,k+i}^{-1} R_{d,k+i}^T \Delta p_{d,k+i} \geq 1 \tag{8c}$$

$$s_{\min} \leq s \leq s_{\max} \tag{8d}$$

$$u_{\min} \leq u_{k+i-1} \leq u_{\max}, \quad i = 1, \ldots, N \tag{8e}$$

If more obstacles are present in the robot's feasible space, more inequality constraints like equation $8c$ are added to the optimization problem, but additional decision variables would also need to be added.

C. The dynamic programming formulation

As explained before, theoretically, the trajectory tracking with a moving obstacle given uncertainty in its position can be solved using the DP algorithm. Mathematically, the DP formulation, considering a deterministic setting, becomes

$$J_T(x_T) = (x_T - x_{\text{ref},T})^T Q_f (x_T - x_{\text{ref},T})$$

$$J_t(x_t) = \min_{u_t,\ldots,u_{t+N-1},s} \sum_{i=0}^{N-1} \Big[(x_{t+i} - x_{\text{ref},t+i})^T Q (x_{t+i} - x_{\text{ref},t+i}) + u_{t+i}^T R u_{t+i}\Big] + (s - s_{\text{ref}})^2 + J_T(x_{t+N})$$

$$x_{t+i+1} = Ax_{t+i} + Bu_{t+i}, \quad i = 0, \ldots, N-1$$

$$-u_{\max} \le u_{t+i} \le u_{\max}$$

$$(p_{t+i} - \mu_{t+i})^T E_{t+i}(s)(p_{t+i} - \mu_{t+i}) \ge 1$$

$$p_{t+i} = \begin{bmatrix} x_{1,t+i} \\ x_{3,t+i} \end{bmatrix}$$

$$E_t(s) = R_t \begin{bmatrix} \frac{1}{\left(s\sqrt{\lambda_{1,t}} + r + r_d\right)^2} & 0 \\ 0 & \frac{1}{\left(s\sqrt{\lambda_{2,t}} + r + r_d\right)^2} \end{bmatrix} R_t^T$$

Equation 8 is the deterministic MPC approximation of the DP formulation shown above.

III. MODEL PROPERTIES

A. Robot Dynamics

Consider a mobile robot moving in a 2D workspace $\mathcal{W} \subset \mathbb{R}^2$. The robot's state is defined as $\mathbf{x} = [x, \dot{x}, y, \dot{y}]^\top \in \mathbb{R}^4$, where (x, y) denotes the position and $(\dot{x}, \dot{y})$ the velocity. The control input is $\mathbf{u} = [u_x, u_y]^\top \in \mathbb{R}^2$, representing acceleration commands.

The discrete-time dynamics with sampling time $\Delta t = 0.2$ s are given by

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k, \quad k = 0, 1, \ldots, N-1 \tag{9}$$

where

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0.2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$B = \begin{bmatrix} \frac{\Delta t^2}{2} & 0 \\ \Delta t & 0 \\ 0 & \frac{\Delta t^2}{2} \\ 0 & \Delta t \end{bmatrix} = \begin{bmatrix} 0.02 & 0 \\ 0.2 & 0 \\ 0 & 0.02 \\ 0 & 0.2 \end{bmatrix}.$$

B. Obstacle Motion and Uncertainty

A moving obstacle travels along a circular track of radius $R = 8$ m. Its angular position is given by $\theta_k = \theta_0 + \omega_{\text{obs}} k \Delta t$, where $\omega_{\text{obs}} = 0.12$ rad/s is the constant angular velocity. The obstacle's true position is

$$\mathbf{p}_{\text{obs},k} = \begin{bmatrix} R\cos(\theta_k) \\ R\sin(\theta_k) \end{bmatrix} = \begin{bmatrix} 8\cos(\theta_k) \\ 8\sin(\theta_k) \end{bmatrix}.$$

However, the robot does not have access to the exact obstacle position. Instead, it receives noisy measurements, and the obstacle's position is modeled as a random variable following a Gaussian distribution:

$$\mathbf{p}_{\text{obs},k} \sim \mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k), \quad \boldsymbol{\mu}_k \in \mathbb{R}^2,\ \Sigma_k \in \mathbb{R}^{2\times 2}. \tag{10}$$

The initial mean position is $\boldsymbol{\mu}_0 = [8\cos(55^\circ), 8\sin(55^\circ)]^\top \approx [4.588, 6.553]^\top$ m, and the initial position covariance is

$$\Sigma_{\text{pos},0} = \begin{bmatrix} 0.5 & 0.3 \\ 0.3 & 0.2 \end{bmatrix}.$$

The initial mean velocity is $\boldsymbol{\mu}_{v,0} = [-8 \cdot 0.12\sin(55^\circ), 8 \cdot 0.12\cos(55^\circ)]^\top \approx [-0.786, 0.551]^\top$ m/s, with velocity covariance

$$\Sigma_{v,0} = \begin{bmatrix} 0.3 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}.$$

The mean $\boldsymbol{\mu}_k$ and covariance Σ_k evolve according to the obstacle's predicted motion. For prediction horizon $N = 30$, the sequence of mean positions $\{\boldsymbol{\mu}_{k+i}\}_{i=1}^N$ and covariance matrices $\{\Sigma_{k+i}\}_{i=1}^N$ are propagated using the obstacle's velocity distribution:

$$\mu_{k+i} = \mu_{k+i-1} + \Delta T \frac{\mu_{v,k+i} + \mu_{v,k+i-1}}{2} \tag{11a}$$

$$\Sigma_{k+i} = \Sigma_{k+i-1} + \frac{\Delta T^2}{4}(\Sigma_{v,k} + \Sigma_{v,k+i-1}) \tag{11b}$$

The velocity covariance at future steps is propagated as $\Sigma_{v,k+i} = (1 + 0.10i) \begin{bmatrix} 0.08 & 0.05 \\ 0.05 & 0.03 \end{bmatrix}$.

C. Mahalanobis Distance and Safe Set

To quantify the proximity between the robot and the uncertain obstacle, the Mahalanobis distance is used to create the no-go region. For a robot position $\mathbf{p}_{\text{rob}} = (x, y)^\top$, the squared Mahalanobis distance to the obstacle's distribution is written below, referring to equation $3b$

$$d_M^2(\mathbf{p}_{\text{rob}}; \mathcal{N}(\boldsymbol{\mu}, \Sigma)) = (\mathbf{p}_{\text{rob}} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{p}_{\text{rob}} - \boldsymbol{\mu}). \tag{12}$$

The region where the robot is considered unsafe is defined as

$$\mathbb{D} = \left\{\mathbf{p} \in \mathbb{R}^2 \mid d_M(\mathbf{p}; \mathcal{N}(\boldsymbol{\mu}, \Sigma)) < s\right\}, \tag{13}$$

where $s > 0$ is a confidence parameter that scales the ellipse size. Larger s corresponds to a larger safe margin.

Taking into account the physical radii of the robot ($r_{\text{rob}} = 0.15$ m) and the obstacle ($r_{\text{obs}} = 0.20$ m), the effective safe distance is enforced by inflating the ellipse. Let $r_{\text{sum}} = r_{\text{rob}} + r_{\text{obs}} = 0.35$ m. The constraint

$$\Delta\mathbf{p}^\top R\Lambda^{-1}R^\top\Delta\mathbf{p} \geq 1, \quad \Delta\mathbf{p} = \mathbf{p}_{\text{rob}} - \boldsymbol{\mu}, \qquad (14)$$

defines the complement of an ellipse with semi-axes lengths $s\sqrt{\lambda_1} + r_{\text{sum}}$ and $s\sqrt{\lambda_2} + r_{\text{sum}}$, where λ_1, λ_2 are the eigenvalues of Σ, and R is the orthogonal matrix of eigenvectors. This constraint ensures that the robot stays outside the inflated uncertainty ellipse.

D. Reference Trajectory

The robot is tasked with tracking a circular reference trajectory of radius $R = 8$ m with constant angular velocity $\omega_{\text{ref}} = 0.52$ rad/s. The reference linear velocity is $v_{\text{ref}} = R\omega_{\text{ref}} = 4.16$ m/s.

E. Optimal Control Problem

At each time step k, given the current robot state $\mathbf{x}_k$, the predicted obstacle mean positions $\{\boldsymbol{\mu}_{k+i}\}_{i=1}^N$ and covariances $\{\Sigma_{k+i}\}_{i=1}^N$, the robot solves the non-convex, nonlinear optimization problem shown in equation 8 over a prediction horizon $N = 30$:

where:

- $U_k = [\mathbf{u}_k^\top, \mathbf{u}_{k+1}^\top, \ldots, \mathbf{u}_{k+N-1}^\top]^\top \in \mathbb{R}^{60}$ is the sequence of control inputs.
- $Q = \text{diag}([0.2, 0.05, 0.2, 0.05]) \in \mathbb{R}^{4\times4}$ and $P = \text{diag}([0.001, 0.001]) \in \mathbb{R}^{2\times2}$ are positive definite weighting matrices for state tracking error and control effort, respectively.
- $w_s = 0.005$ is a weighting factor for the slack variable s.
- $s_{\text{ref}} = \sqrt{5.991} \approx 2.448$ corresponds to a 95% confidence level for a χ^2 distribution with 2 degrees of freedom.
- $s_{\min} = s_{\text{ref}}$, $s_{\max} = 10.0$ are bounds on the slack variable.
- $\mathbf{u}_{\min} = -3$, $\mathbf{u}_{\max} = 3$ are input bounds (in m/s^2).

The decision variable s is introduced to soften the collision avoidance constraint. When s is close to s_{ref}, the robot maintains a 95% confidence safety margin. Larger s values create a larger safe region, while smaller values allow a closer approach. The term $(s - s_{\text{ref}})^2$ in the cost function penalizes deviation from the reference safety margin.

F. Closed-Loop Implementation

The closed-loop simulation runs for $N_{\text{sim}} = 600$ time steps (120 seconds). At each time step k:

1) The robot measures its current state $\mathbf{x}_k$.
2) The obstacle's future mean positions $\{\boldsymbol{\mu}_{k+i}\}_{i=1}^N$ and covariances $\{\Sigma_{k+i}\}_{i=1}^N$ are predicted using the obstacle motion model.
3) The optimization problem is solved using sequential quadratic programming (SQP) with MaxIterations $= 500$ and ConstraintTolerance $= 10^{-6}$.
4) The first control input $\mathbf{u}_k$ is applied to the system.
5) The robot state is updated using the true (possibly disturbed) dynamics. A time-correlated disturbance $w_k = \alpha w_{k-1} + w_{\text{std}} \cdot \mathcal{N}(0,1)$ with $\alpha = 0.85$ and $w_{\text{std}} = [0, 0.08, 0, 0.08]^\top$ is added to the nominal dynamics.
6) The obstacle advances according to its true motion with angular velocity $\omega_{\text{obs}} = 0.12$ rad/s.
7) The procedure repeats at time $k + 1$.

This receding-horizon strategy enables the robot to safely navigate while accounting for uncertainty in the obstacle's position.

IV. EXPERIMENTS

A. Scenario 1: Perfect model

In this scenario, the robot's model is completely deterministic with no noise at all. Figure 1 shows the trajectory the robot takes throughout the simulation time of 600 seconds. As is clear from the trajectory plot, the robot successfully avoids the moving obstacle. The trajectory plot is also a very good example of why the use of MPC is a good idea for these kinds of applications. The robot can be seen to be beginning to avoid the obstacle before encountering it, which is possible because of the proactive nature of MPC, which takes into consideration the future events as the optimization problem solves to minimize cost accumulated from the current time step to the end of the horizon.

The feasibility of the solution is guaranteed by the decision variable s. The s_{ref} acts as the reference size for the no-go ellipse for the robot. Figure 2 shows how the decision variable s changes with the simulation. The change is minimal, which is due to the large penalty associated with trying to break the barrier established by $s_{ref} = \sqrt{5.99} = 2.4474$. Again, this value of s_{ref} corresponds to the most likely obstacle position with 95% confidence. In other words, the robot is penalized if it tries to decide on a small s and maneuver close to the obstacle to avoid it. Analyzing the s vs time plot, it is evident that the robot can mostly stay outside the 95% confidence zone without any infeasibility.

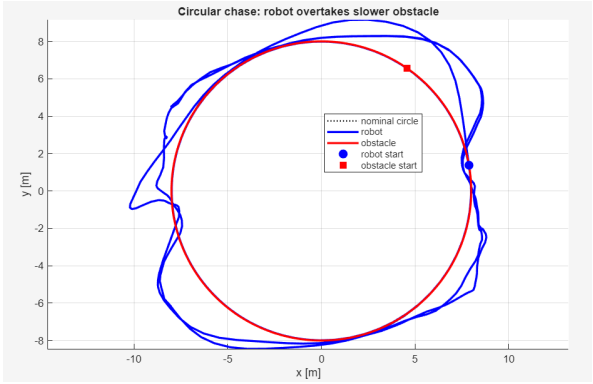

Fig. 1: Robot trajectory in scenario 1

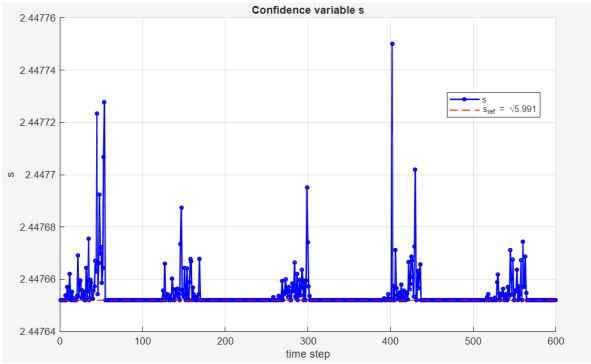

Fig. 2: s trajectory in scenario 1

The satisfaction of the bounding constraint is evident from Figure 3, which shows the input applied during the experiment.

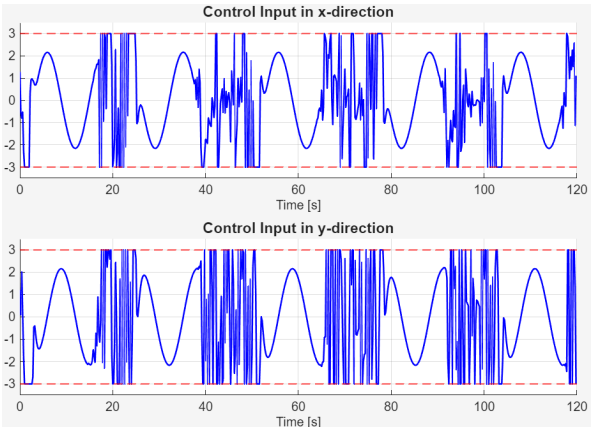

Fig. 3: Optimal control in x and y directions for scenario 1

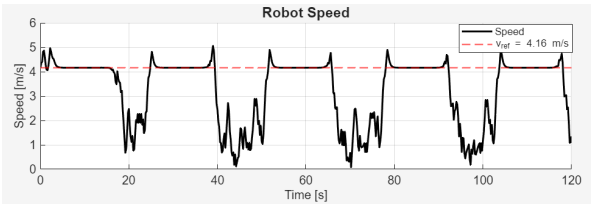

Fig. 4: Velocity trajectory in scenario 1

B. Time taken for solution

While the results above show that MPC performs well in the simulated scenarios, the applicability of MPC remains an important topic of discussion. Figure 5 shows the time taken every simulation step to find the optimal input. At some time steps, the controller takes up to 2500 milliseconds to find the solution. It is evident that with the current parameters of the system and simulation, the time taken to solve for optimal control input is too large, given the sampling rate of 200 milliseconds. Hence, the proposed theoretical method is not ideal for real-life applications.

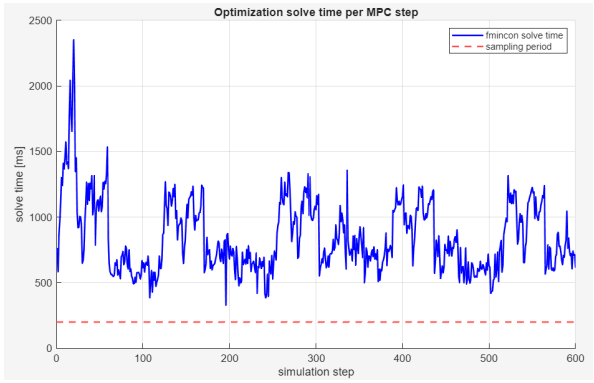

Fig. 5: Time taken every simulation step with N=30

C. Scenario 2: Reduced horizon length

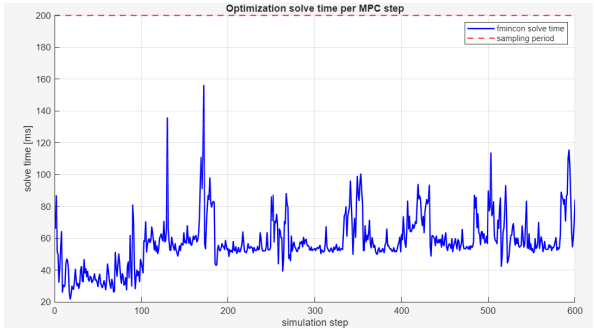

Fig. 6: Time taken every simulation step with N-8

This project focused on a single moving obstacle and not multiple ones. As a result, several simulation parameters can be catered to this specific scenario. An important parameter is the prediction horizon. A larger prediction horizon means a more complex optimization problem, and usually, the prediction horizons are preferred to be long in real-life cases. Since the project only considers a single moving obstacle, the prediction horizon can be decreased up to $N = 8$. At this horizon length, the computation time significantly decreases and is well below the sampling rate of 200 milliseconds, as shown in Figure 6, and the maximum computation time is around 156 milliseconds.

Likewise, as seen in Figures 7 and 8, the robot's trajectory tracking is still excellent despite a lower prediction horizon. Because of a smaller prediction horizon, the robot is short-sighted, and hence the trajectory is influenced more because of its momentum. This is exactly what is seen in Figure 8: the optimal solution discourages a change in direction. Another important realization from these experiments is that a larger prediction horizon does not equal best performance, and in cases like the one in this project, a smaller prediction horizon makes the implementation practical.

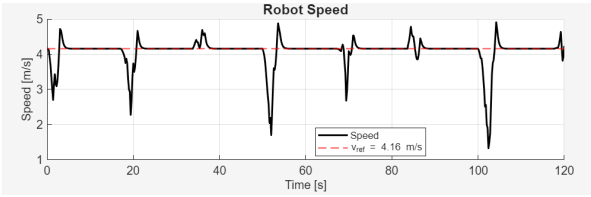

Fig. 7: Velocity trajectory in scenario 2

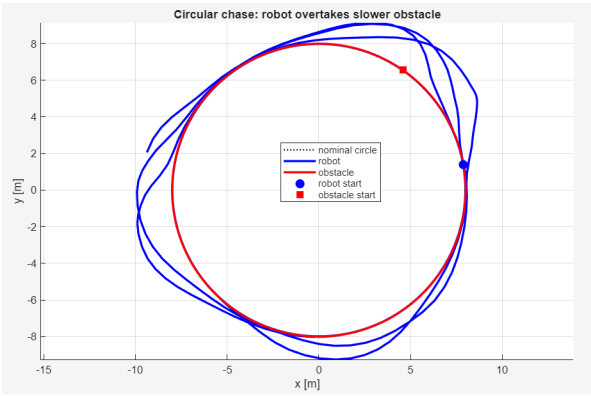

Fig. 8: Robot trajectory in scenario 2

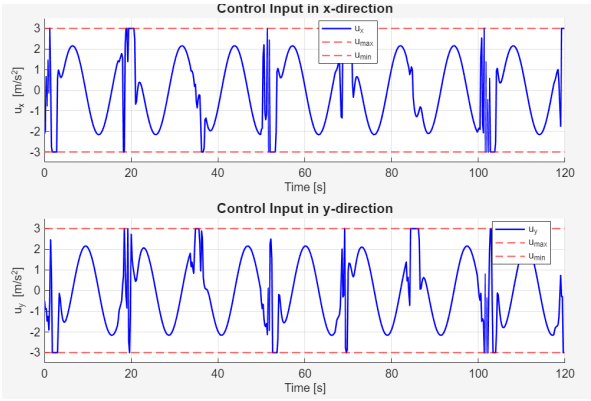

Fig. 9: Optimal control in x and y directions for scenario 2

D. Scenario 3: Time correlated noise in the model

As seen from the sections before, deterministic MPC performed well with a deterministic model of the system, and is also implementable for a 2D case. However, a deterministic system is not a very accurate representation of the system. Hence, the next step is to see how a more practical MPC with $N = 8$ performs with a time-correlated disturbance added to the system dynamics. The MPC is still solving the deterministic problem, but is now used in a system with noise. This disturbance can be imagined as a wind blowing in the simulation environment that displaces the robot from its actual trajectory. The robot trajectory with the same reference path and identically moving target is shown in Figure 10, and the deterministic MPC does not perform well in this case. Because the length of the horizon is small ($N = 8$), the control is short-sighted, and with the noise in the system, the prescribed sub-optimal trajectory accumulates a high cost. This is because by the time the controller decides on the best control, it is already too late, and the robot is too far off the reference trajectory. As a result, there are many instances when the robot wanders off the path without prioritizing the accumulated tracking cost.

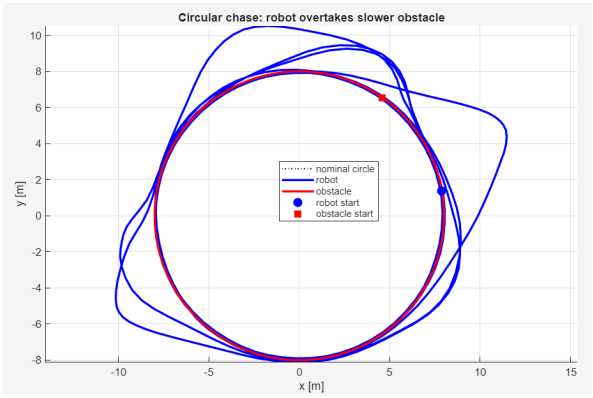

Fig. 10: Robot trajectory in scenario 3

E. Scenario 4: Time correlated noise in the model with a longer prediction horizon

In the previous section, a limitation of a smaller prediction horizon is evident, the limitation being the inability to handle time-correlated noise in the system. Hence, reducing the prediction horizon might work in simulation because of lower computational cost, but in reality, with the noise inherent in the system, the system is bound to fail. Even if the dynamics are known perfectly, if there are multiple obstacles in the robot's path, the length of the horizon cannot be small. To neutralize the limitation shown in section 3, the length of the prediction horizon is increased to $N = 30$ again for this section's experiments. The same deterministic MPC is applied in a more realistic model, in which an additive time-correlated noise is injected into the dynamics.

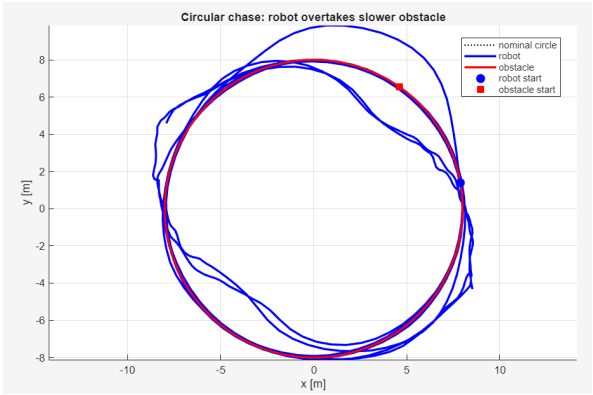

Fig. 11: Robot trajectory in scenario 2

Figure 11 shows the optimal trajectory for the robot. It is evident that with the addition of noise into the system, the robot's motion is not deterministic, and as a result, the optimal trajectory also changes. With a longer prediction horizon, the robot tracks the path more closely in comparison

to scenario 3. The MPC considers a longer future cost, which means the sub-optimal solution is closer to the optimal solution. This realization aligns with the theory on the infinite horizon variant of the optimal control problem. Even with a stochastic element to the dynamics, the obstacle avoidance is still successful for the entirety of the simulation period.

The prescribed optimal control is identical to scenario 1 with a deterministic model. The only difference is that the control is applied even just to make the robot track its original path. This is evident from a non-smooth control trajectory seen in Figure 12. This effect is also evident from comparing Figure 4 and Figure 13, which show the robot's velocity trajectory in both scenarios. In a deterministic setting, the robot's velocity becomes smooth after avoiding the obstacle, but in a stochastic setting, there is always an external disturbance trying to push the robot away from the path, and the MPC works to counter this perpetually. Also, the feasibility of the solution is always established, as seen by the input trajectories in Figure 12, as the no-go region dynamically changes, as evident from Figure 14. Hence, by increasing the length of the prediction horizon, the limitation seen in scenario 3 is mitigated, but as seen in scenario 1, the computational cost increases, making the system impractical. This is one of the forms of the curse of dimensionality that is prevalent in MPCs.

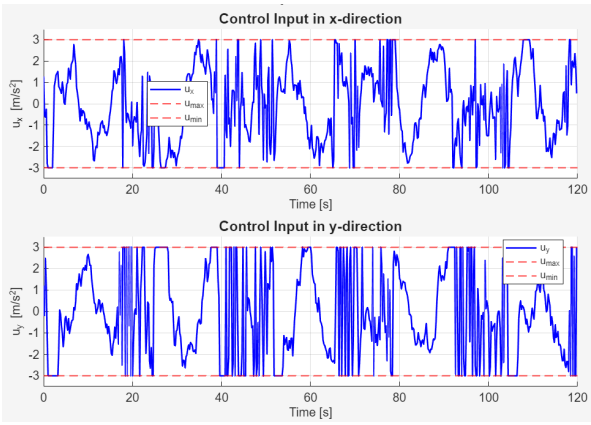

Fig. 12: Optimal control in x and y direction for scenario 4

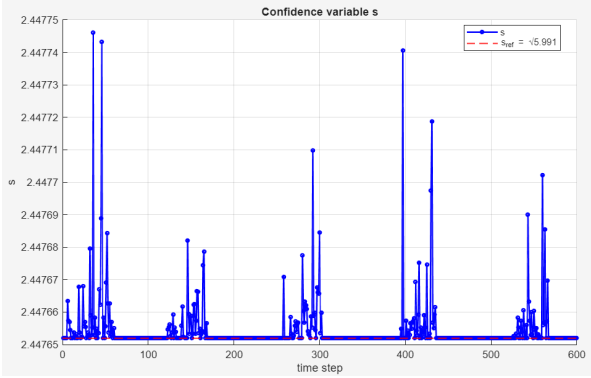

Fig. 13: s trajectory in scenario 4

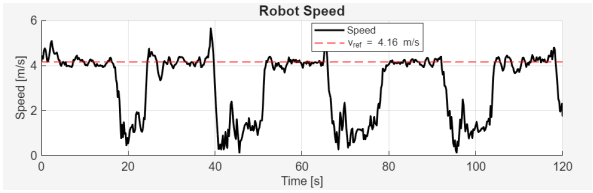

Fig. 14: Velocity trajectory in scenario 4

V. CONCLUSION

The project successfully implements trajectory tracking with dynamic obstacle avoidance, given the knowledge of the obstacle's position with uncertainty. Dynamic programming in a continuous domain can quickly become intractable, and with the addition of constraints, the problem becomes extremely difficult. This is also the case with the problem of trajectory tracking with uncertain moving obstacles, as shown in Section II, subsection C. Hence, this project opted for a reformulated problem because DP is too difficult in this case. This reformulation, which is also known as MPC, gives a sub-optimal solution to the optimal control problem. Despite being sub-optimal, MPC still provides a good way around the complexity of DP in a constrained and continuous domain. As evident from experiments and analysis shown in Section IV, MPC is more than capable of performing trajectory tracking.

The MPC implementation centers around solving a non-convex, non-linear optimization problem. The non-linearity and non-convexity arise from the inequality constraint that enforces the no-go region for the robot given some Mahalanobis distance. The results of the experiments are a good example of how MPC can provide a good approximation to the optimal control given by the DP algorithm, especially when the prediction horizon is long enough. While the practical implementation of the DP becomes difficult, the MPC formulation can instead be used. In addition to providing a good approximation to the global optimal control, the MPC formulation allows for an easy addition of constraints to enforce certain behavior in the system. The approach taken in this project, which involves enforcing a dynamic infeasible region instead of a fixed one, shows that MPC is flexible given different applications and considerations. Likewise, in this project, assumptions were made on the deterministic nature of the dynamics, which is usually not the case. Scenarios 2 and 4 in Section IV show that such assumptions are not always bad, and in many cases, making these assumptions might be the best way to approach an optimal control problem.

Despite all of the advantages, MPC is not a free upgrade to the DP algorithm. MPC still solves an optimization problem in its core, which determines the applicability of MPC in a real-life setting. As seen in section IV, a more computationally expensive optimization problem can make MPC too slow for operation. This can be mitigated by further simplifying the assumptions, but these simplifications are generally not ideal, and sacrifices must be made in the performance. Hence, a careful analysis of the system and the

environment is necessary even for the approximation of the optimal control problem with an MPC.

VI. Future works

A natural extension of the implementation in this project is obstacle avoidance, given multiple moving obstacles. The formulation of the MPC is intuitive enough to extend this to the case with multiple obstacles. It could be as simple as adding more infeasible regions for multiple obstacles, and as a result, more inequality constraints enforcing this. Likewise, extension to a three-dimensional scenario should also be straightforward. The major hurdle in extending a simple problem like the one demonstrated in this project is the complexity of the optimization problem. For a simple problem like this, the solution time taken for each simulation step is 1.5 seconds, which might be too large for many applications. Hence, the complexity of the optimization problem must be taken into consideration before deciding on using MPC. If MPC is the best way to go about it, its variation, such as explicit MPC, can also be a good option. Likewise, the disturbance injected into the dynamics for this project is not the most realistic one. Even though the nominal MPC developed works well with a noisy model definition, a more robust MPC is almost always preferred.

VII. Acknowledgment

The implementations in this project are the ideas presented in the research article [5]

References

[1] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[2] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60:503–515, 1954.

[3] Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[4] David Q. Mayne. Model predictive control: Recent developments and future promise. *Automatica*, 50(12):2967–2986, 2014.

[5] Eric Schöneberg, Michael Schröder, Daniel Görges, and Hans D. Schotten. Trajectory planning with model predictive control for obstacle avoidance considering prediction uncertainty. *IFAC-PapersOnLine*, 59(18):349–354, 2025. 14th IFAC Symposium on Robotics ROBOTICS 2025.

[6] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. Predictive active steering control for autonomous vehicle systems. *IEEE Transactions on Control Systems Technology*, 15(3):566–580, 2007.