

MPC for Robots Under Communication Latency

Kush Patel (3041935783), Aaditya Shrivastava (3041936966), Siddharth Singh (3036713060),
Sarvesh Samadhan Vichare (3041966502)

Video: <https://youtu.be/UDsfPFw9gRc> , GitHub: <https://github.com/siddharths09/MPCForLatency.git>

December 2025

Abstract— Networked robotic systems operating under teleoperation or cloud based control are subject to communication latency, which can harm performance and compromise safety. This paper presents a supervisory Model Predictive Control framework for mobile robot motion control under guaranteed communication delays. When latency is detected, a localized MPC controller uses onboard state feedback, surrounding environmental information, and previously received control commands to predict dynamically feasible future trajectories. Rather than relying on a predefined safe state, the controller computes control actions that maintain stable and predictable motion while respecting system constraints. The approach is implemented in a ROS2 simulation environment using a simplified kinematic unicycle model with keyboard teleoperation, evaluated under varying latency conditions. Results demonstrate smooth trajectory tracking and improved robustness compared to direct command passthrough under delayed communication.

I. INTRODUCTION

Robotic systems that rely on teleoperation are fundamentally constrained by the reliability of the underlying communication network. Variable communication latency can lead to delayed commands and stale control inputs, which in safety critical scenarios may result in unstable or unpredictable motion. Traditional controllers are typically designed under the assumption of continuous command availability and therefore experience significant performance drop when communication delays increase. This motivates the use of predictive control strategies that allow the robot to reason about its future behavior using an internal model rather than relying solely on real time commands. Early work on teleoperation under communication delays was presented by Anderson and Spong [1], who developed a bilateral control framework that maintains stability between a human operator and a remote robot in the presence of time delay. Their approach assumes continuous communication and focuses on compensating for delayed feedback rather than interrupted command streams. In contrast, our work considers delayed network conditions and introduces a supervisory Model Predictive Control layer that enables stable and predictable robot motion when external

commands become unreliable. For our specific use case, we intentionally adopt an extremely simplified system model in order to demonstrate a minimum viable proof of concept and clearly illustrate the core workings of the proposed MPC based supervisory controller. This motivates the use of a simplified robot model implemented using turtlesim configurations, allowing the effects of communication latency and the resulting MPC behavior to be isolated and examined without additional system complexity. In this project, we propose a supervisory control architecture that improves a standard teleoperation pipeline with a localized Model Predictive Controller. When elevated latency is detected, the controller uses onboard state feedback, surrounding environmental information, and previously received commands to generate smooth and dynamically feasible trajectories. This design maintains continuity of motion, eliminates abrupt behavior under delayed communication, and improves overall control without modifying the external command source.

II. PAPER STRUCTURE

Section III describes the overall software architecture and node level implementation used to emulate communication latency. Section IV formulates the simplified Model Predictive Control problem, including the kinematic model, goal prediction strategy, objective function, and input constraints. Section V describes the results of our project. Finally, Section VI discusses future applications and potential extensions of the proposed framework.

III. SOFTWARE ARCHITECTURE AND NODE IMPLEMENTATION

In order to rigorously validate the proposed MPC architecture, we had to move beyond the ideal simulation conditions. Thus, we developed a custom “Fault Injection” middleware layer in ROS 2 Humble to replicate the stochastic nature of real time wireless networks such as 5G or WiFi systems

A. Network Fault Injection Nodes

To rigorously test the safety controller, we developed three Python-based nodes that manipulate packet timing and delivery using a First-In-First-Out (FIFO) buffer approach. The Network Delay node (Lag generator) simulates intermittent network congestion rather than simple constant latency. The node sits between the teleoperation input and robot, utilizing the FIFO buffer to manage packet flow. The burst logic listens to command inputs and increments a local counter for each packet it receives. Instead of delaying every packet, the system applies a massive latency spike (2000ms) only to every N^{th} (10^{th}) command.

All messages are stored in a deque. A high-frequency timer check (100Hz) releases standard packets immediately but holds the flagged lag packets until their age exceeds the 2.0-second threshold. This effectively simulates ‘bursty-lag’ which is common in cellular handovers. This creates a predictable but challenging failure mode where the robot periodically goes blind for 2 seconds while the operator continues to send in commands, thus forcing the MPC supervisor to rely on its prediction horizon to maintain safety.

B. Autonomous Safety Controller

The main intelligence of our system resides in the TurtleController, which integrates network monitoring and the control solver into a single entity.

For latency detection and prediction, the node subscribes to the /network_delay_active topic published by the fault nodes. When the delay flag is raised, the node switches from the ‘Passthrough’ mode to the MPC mode for prediction. Because the robot cannot see the current command during a delay, the controller predicts the robot’s future state by simulating the last 10 commands stored in its history buffer. This generates a predicted goal state that acts as the target for the MPC to navigate toward.

The system uses the Sequential Least Squares Programming (SLSQP) solver from the scipy.optimize library to solve for a control sequence over a 10-step horizon. The objective function minimizes the distance to the predicted goal and the heading error while smoothing control effort. A critical feature of the implementation is the integration of an exponential barrier function for obstacle avoidance. The controller defines a safety margin of 0.3m around known obstacles. If the predicted trajectory penetrates this zone, the cost function adds a penalty calculated as $1000 \cdot e^{5.0(r_{\text{safe}} - d_{\text{obs}})}$. This creates a mathematical repulsive force that pushes the optimal path away from obstacles, ensuring the robot autonomously navigates

around dangers even when the human operator is disconnected.

IV. MODEL PREDICTIVE CONTROL

When the delay flag is raised, the controller transitions into a localized Model Predictive Control (MPC) mode. The purpose of this supervisor is to maintain stable and predictable motion for the duration of the communication loss by computing a dynamically feasible control sequence using (i) the robot’s onboard state estimate, (ii) a lightweight kinematic model, (iii) known obstacle locations, and (iv) an inferred intent target derived from previously received commands.

Problem Formulation

At each control step, MPC solves a finite-horizon optimal control problem over a horizon of $N = 10$ timesteps for 1.0s at $\Delta t = 0.1s$. The decision variables are a sequence of twist commands- forward linear velocity and angular velocity. The optimizer predicts the resulting state trajectory over the horizon and chooses controls that minimize a cost function balancing goal-seeking behaviour with obstacle avoidance. After solving, only the first control twist command is applied. The entire optimization is then re-solved at the next timestep using updated state feedback. This receding-horizon execution provides closed-loop robustness even under modeling errors.

Kinematic Model

The robot is modeled as a planar unicycle operating in a two dimensional workspace. The system state is defined as:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Where x and y denote the Cartesian position of the robot and θ denotes its heading angle measured with respect to the global x axis.

The control input is defined as:

$$\mathbf{u} = \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Where v is the linear velocity and ω is the angular velocity. The continuous time kinematic model of the robot is given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cos \theta \\ v \sin \theta \\ \omega \end{bmatrix}$$

For use within the model predictive control frame, the continuous time dynamics are discretized using a forward Euler approximation with sampling time Δt . The resulting discrete time state equation is:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + v_k \cos \theta_k \Delta t \\ y_k + v_k \sin \theta_k \Delta t \\ \theta_k + \omega_k \Delta t \end{bmatrix}$$

Goal Prediction

Rather than predefining a fixed state, the controller infers a short-term target during the delay, which is based on the received teleoperation commands. This allows for the controller to have a more localized character that could intuitively make use of its environment and The node stores a history buffer of the last 10 commands. When delay begins, it simulates a forward motion for those 10 steps starting from the current state and uses the final position as the predicted goal state.

Objective Function

The MPC optimizer evaluates a candidate control sequence over a cost function based on the received goal prediction. It has two main components:

1. Goal tracking

Using a positional error based cost, the controller aims to navigate to the predicted goal state. At each predicted step k , the controller adds a squared Euclidean distance-to-goal penalty:

$$J_{goal} = \sum_{k=1}^N [(x_g - x_k)^2 + (y_g - y_k)^2]$$

2. Obstacle avoidance

Obstacles are represented as circles with known centers (x_o, y_o) and radius r_o . The controller defines a safety buffer $m = 0.3m$, creating a minimum allowed distance:

$$d_{safe} = r_o + m$$

For each predicted state, it computes the distance to each obstacle center using the Euclidean distance formula:

$$d_k = \sqrt{(x_g - x_k)^2 + (y_g - y_k)^2}$$

If the predicted trajectory approaches or penetrates the safety boundary, the function adds an increasing penalty. For d_k inside the safety region ($d_k < d_{safe}$), an exponential barrier -like penalty is applied to strongly discourage collision. For d_k near the safety boundary (within a small band outside d_{safe}), an additional repulsive penalty is applied to push the solution away from the close pass trajectory unless absolutely necessary, making this a soft constraint.

Input Bounds and Feasibility

The control inputs are constrained to lie within limits defined for the simulated system to ensure feasible and safe motion.

$$0 \leq v \leq v_{max}, \quad -\omega_{max} \leq \omega \leq \omega_{max} \\ v_{max} = 2.0, \quad \omega_{max} = 2.0$$

The initial guess for the optimizer is chosen to be a small forward velocity with zero angular rate across the horizon. This ensures persistent feasibility and improves solver stability and repeatability.

Numerical Solver

The goal of this localized MPC is to be lightweight, and run on onboard hardware, making computational efficiency key. The Sequential Least Squares Programming (SLSQP) solver provided by the scipy package supports these characteristics, performing well with support bound constraints and known for being computationally lightweight. [4]

Termination and Safety Behaviour

The supervisor stops the robot when the robot is sufficiently close to the predicted goal. In addition, further commands are not implemented until the network delay is solved, ensuring that delayed commands are not applied randomly and break system constraints.

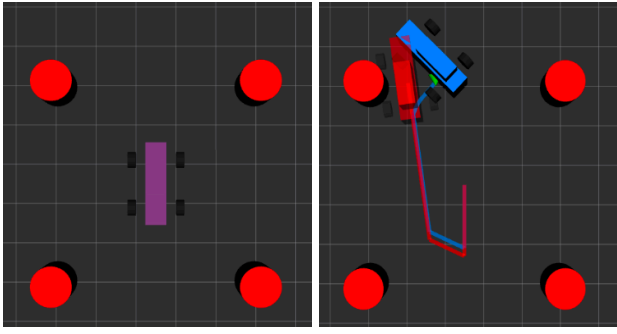


Fig. 1 (a) Start of simulation Fig. 1(b) Network Delay behaviour

V. RESULTS

The simulation window is visualized using Rviz where two robots are starting from the centre of the grid. The red robot is the standard robot taking tele-op commands directly from the keyboard. Whereas, the blue robot is running with the help of an MPC controller which is activated every time there is a delay in the tele-op signals.

When a two second network delay is introduced, the red robot stops at its current position as it does not receive any command (Fig 1b). On the other hand the Blue MPC robot uses its prediction horizon (Indicated by green line) to fill the delay and maintains a smooth continuous trajectory (Fig 1a). To show this graphically, Fig.2 has the red line, which indicates the red robot, drops to zero velocity at the delay showing failure, whereas the blue line stays stable. The system tries to mask the lag and keeps the robot's motion smooth and away from obstacles in real time. This allows the local MPC to solve latency issues and prevent the robot from breaking system constraints.

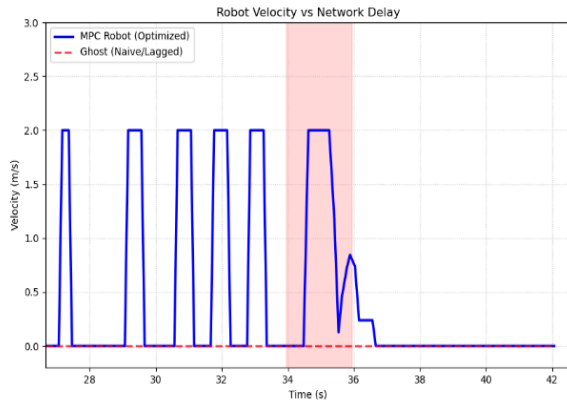


Fig. 2 Graphical Comparison of Velocity vs Network Delay

VI. FUTURE SCOPE AND APPLICATIONS

While with this project we successfully demonstrate a fundamental fail-safe mechanism for networked mobile robots, the supervisor-MPC architecture has broad capabilities across several high impact domains where communication reliability cannot be guaranteed

A. Space Teleoperation and Planetary Exploration:

One of the most critical applications for this technology is in space robotics, where communication delays are unavoidable. For example, Earth to Mars control may have transmission delays from 4 to 24 minutes, making direct joystick control impossible. Our project models the foundational solution for such missions: the human operator sends high-level waypoints, and the onboard MPC handles the real-time execution, effectively "closing the loop" locally to compensate for the massive communication gap.

B. Connected Autonomous Vehicles:

When platooning in convoy vehicle scenarios where trucks drive close together to save fuel, latency can disrupt spacing and cause collisions. This is particularly useful in such Vehicle to Everything (V2X) cases, where an emergency MPC such as our implementation, is required to maintain safe inter vehicle distances.

VII. CONCLUSION

This project uses Nonlinear Model Predictive Control to mitigate instabilities caused by latency issues during teleoperation. The system is able to bridge communication gaps by estimating goal position and simulating artificial fields for obstacle avoidance, where the standard reactive controllers malfunction. Our results demonstrate smooth velocity tracking and safety accomplished by the localized MPC controller during network lag, emphasising the importance of local predictive intelligence in degraded network environments.

VIII. RESULTS

- [1] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay," *IEEE Transactions on Automatic Control*, vol. 34, no. 5, pp. 494–501, May 1989, doi: 10.1109/9.24201.
- [2] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge, UK: Cambridge University Press, 2017.
- [3] Y. Hatori and Y. Uchimura, "Obstacle avoidance during teleoperation by model predictive control with time-varying delay," *IEEJ Journal of*

IndustryApplications,
https://www.jstage.jst.go.jp/article/ieejia/12/2/12_22004524/_article (accessed Dec. 20, 2025).

- [4] D. Kraft, *A software package for sequential quadratic programming*, Tech. Rep. DFVLR-FB 88-28, DLR German Aerospace Center — Institute for Flight Mechanics, Koln, Germany, 1988.