

Cloud-Connected Model Predictive Control for Autonomous Mobile Robots in the Presence of Network Latency and Message Losses

by

Prajwal Thakur

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2025

© Prajwal Thakur 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Autonomous driving systems have become increasingly popular due to their potential to reduce traffic accidents compared to human-operated vehicles, as well as their significant potential to optimize traffic flow and reduce pollution.

However, there remain cases in which autonomous driving system malfunctions lead to accidents. These failures often arise from the partial observability of the environment and the inherent limitations of onboard sensor systems.

To address these shortcomings, [Cloud-Connected Autonomous System \(CCAS\)](#) has emerged as a promising alternative, leveraging cloud-based computation and multiple distributed sensors to build a comprehensive understanding of the environment (including the ego vehicle) and make collective decisions for all controlled agents in the cloud. This approach improves decision-making and reduces the need to install costly onboard hardware on every vehicle by centrally sharing sensors and computational resources.

Despite its advantages, introducing cloud connectivity into autonomous systems presents significant challenges, particularly network latency and message loss. Such latencies can negatively impact vehicle control and safety, especially when they exceed the control sampling interval, leading to unstable maneuvers or potential collisions.

This work introduces a cloud controller designed to compensate for the effects of latencies longer than a single control sampling interval. A [Robot Operating System \(ROS\)](#)-based simulation environment was developed for rapid algorithm prototyping and seamless integration with a real testbed. The proposed solution was validated through both simulations and real-world tests involving an autonomous hospital bed using Ackermann steering in an indoor scenario. The experimental outcomes highlight the method’s effectiveness and practical potential.

Acknowledgements

I am grateful to my supervisors, Professor Yash Vardhan Pant and Professor Amir Khajepour for the opportunity to join their research groups and for their invaluable guidance, insight, and support throughout my Masters journey. Their feedback and our discussions have taught me so much about research and tackling problems as a researcher, and it has been a privilege to learn from them. I also want to thank the awesome members of the Mechatronic Vehicle Systems Lab and the Control, Learning and Logic Group, whose work has been an inspiration, and from whom I have learned so much. I wish to thank Yufeng Yang and Minghao Ning for their assistance during experiments. I also want to acknowledge Avraiem Iskandar, Arezoo Alipanah, Akash K, Da Saem Lee, Leroy DSouza, Kvk Praneeth, with whom I have shared so many wonderful memories.

Dedication

To my family.

Table of Contents

Author’s Declaration	ii
Abstract	iii
Acknowledgements	iv
Dedication	v
List of Figures	ix
List of Tables	xi
List of Abbreviations	xii
1 Introduction	1
1.1 Thesis Contribution	3
1.2 Thesis Outline	4
2 Literature Review	5
2.1 Introduction	5
2.2 Latency in Cloud-Connected Autonomous Systems	5
2.3 Control Strategies for Latency Mitigation	6

3	Background	9
3.1	Cloud-Connected Autonomous Systems	9
3.2	Model Predictive Control (MPC)	10
3.3	Communication Channel	11
3.4	Latency in the Cloud-Controlled System	12
3.5	Problem Statement	13
3.6	Robot Operating System (ROS)	15
3.7	Assumptions	15
4	Methodology	18
4.1	Closed-Loop Pipeline for Cloud-Controlled Autonomous Robot	18
4.2	Motion Model Description	19
4.3	Normalization	23
4.3.1	Scaling Matrices	23
4.3.2	Normalized State-Space Dynamics	24
4.3.3	Normalized Cost Function	24
4.4	Predictive Controller Design	26
4.5	Design of Cloud-Controller	27
4.5.1	State Estimation Function	29
4.6	Fire-and-Forget Communication	30
4.6.1	Constant latency	30
4.6.2	Lossy Network	32
4.7	Acknowledgment-Driven Communication	34
4.8	Algorithm for Computing Control Sequence	35
4.9	Control Sequence Handling in the Autonomous Mobile Robot (AMR)	36

5	Simulations and Experiments Results	38
5.1	Simulation Setup	40
5.2	Simulation Results	41
5.2.1	Fire-and-Forget Communication with Constant Mean Latency and Standard Deviation	42
5.2.2	Fire-and-Forget Communication with with Constant Latency and Message Dropouts	43
5.2.3	Acknowledgment-Driven Communication	50
5.3	Experiment Setup	55
5.4	Experiment Results	55
5.4.1	Fire-and-Forget Communication with Constant Latency and Dropouts	56
5.4.2	Acknowledgment-Driven Communication	58
6	Conclusion	62
6.1	Limitations and Future Work	63
	References	65

List of Figures

1.1	Example of a cloud-connected autonomous system.	2
1.2	Prototype of AMR and LiDAR.[6], [11]	3
1.3	The AMR is tracking the reference path, controlled by the cloud computer.	4
3.1	Timing diagram illustrating various delays in CCAS.	14
3.2	The cloud controller maintains two buffers: one storing sampled states from LiDAR (X_{LiDAR}) and the other storing AMR feedback (X_{AMR}). When the cloud-control algorithm is triggered, the messages are time-aligned, and the most recent time-aligned message (which carries the complete state of the AMR) is forwarded to the cloud controller.	17
4.1	Overview of the pipeline.	19
4.2	Kinematics model of AMR[34] using a three-degree-of-freedom bicycle model, showing the center of gravity (COG), front and rear wheel velocities (v_f, v_r), steering angles (δ_f, δ_r), and distances of the front and rear wheel from the COG (l_f, l_r).	20
4.3	Timing Diagram in ideal Case (when there's no latency in system).	28
4.4	Baseline1 and Baseline2 method's Timing Diagram in constant latency ($\tau_{\text{ul}}, \tau_{\text{dl}} = \Delta T_c^s$).	31
4.5	Baseline2 and Proposed method's Timing Diagram in lossy network ($\tau_{\text{ul}}, \tau_{\text{dl}} = \Delta T_c^s$).	33
4.6	Example of a message loss due large latency variation.	34
4.7	Timing diagram for Baseline2 and Proposed in case of acknowledgment based communication ($\tau_{\text{ul}}, \tau_{\text{dl}} = \Delta T_c^s$).	37

5.1	Lateral and yaw error with reference to the closest point on the reference track.	40
5.2	Rqt graph for the simulation stack.	41
5.3	Example of tracking performance at a constant mean latency of 0.1 seconds and a standard deviation of 0.02 seconds.	43
5.4	Performance, compared to the tracking cost, degrades as the dropout percentage increases in fire-and-forget communication.	46
5.5	Performance, compared to the Model Predictive Control (MPC) cost, degrades as the dropout percentage increases in fire-and-forget communication.	47
5.6	Comparison of performance against the MPC cost under different latencies and the same dropout percentage in fire-and-forget-based communication.	48
5.7	Example of tracking performance at a constant latency of (uplink: 0.15 sec, downlink: 0.15 sec) and a 60% dropout in fire-and-forget communication.	49
5.8	Example of tracking performance at a constant latency of (uplink: 0.25 sec, downlink: 0.25 sec) and a 60% dropout in fire-and-forget communication.	49
5.9	Example of tracking performance at an average latency of 0.05 seconds in acknowledgment-based communication.	52
5.10	Example of tracking performance at an average latency of 0.25 seconds in acknowledgment-based communication.	52
5.11	Performance, compared to the tracking-error cost, degrades as the average latency increases in acknowledgment-based communication. However, the proposed method still outperforms Baseline2.	53
5.12	Performance, compared to the MPC cost, degrades as the average latency increases in acknowledgment-based communication. However, the proposed method still outperforms Baseline 2.	54
5.13	Experimental setup.	56
5.14	Tracking performance of various methods under constant latency, with uplink and downlink latencies of (0.25 sec, 0.5 sec) respectively, in fire-and-forget based communication.	60
5.15	Tracking performance of various methods under uplink and downlink latencies following a truncated Gaussian distribution with means of 0.1 sec, 0.3 sec, and 0.5 sec, and a standard deviation of 0.02 sec, respectively, in acknowledgment-based communication.	61

List of Tables

5.1	AMR and MPC tuning parameters.	41
5.2	Tracking error and MPC cost breakdown across methods and latency (average over 12 simulation runs) in fire-and-forget-based communication. . .	42
5.3	Tracking error and MPC cost breakdown across methods and latency (average over 12 simulation runs) in fire-and-forget-based communication. . .	44
5.4	Tracking error and MPC cost breakdown across methods and latency (average over 12 simulation runs) in fire-and-forget-based communication. . .	45
5.5	Tracking error and MPC cost breakdown across methods for fixed mean latency (0.1s) with increasing variance (average over 12 simulation runs). .	50
5.6	Tracking error and MPC cost breakdown across methods and latency (average over 12 simulation runs) in acknowledgment-based communication. .	51
5.7	AMR and MPC tuning parameters.	55
5.8	Tracking error and MPC cost breakdown across methods and latency (average over 4 experiment runs) in fire-and-forget communication.	57
5.9	Tracking error and MPC cost breakdown across methods and latency (average over 4 Runs) in acknowledgment-based communication.	59

List of Abbreviations

ADS automated driving System [1](#)

AMR Autonomous Mobile Robot [vii](#), [ix](#), [xi](#), [2–4](#), [6](#), [9](#), [10](#), [12–15](#), [17–21](#), [24](#), [27](#), [28](#), [30–38](#), [41](#), [44](#), [50](#), [55](#), [62](#), [63](#)

CCAS Cloud-Connected Autonomous System [iii](#), [ix](#), [1](#), [3–6](#), [8–11](#), [13–15](#), [34](#), [38](#), [40](#), [62](#)

COG center of gravity [ix](#), [16](#), [19](#), [20](#), [39](#)

g(·) Cloud-Controller [27–29](#)

MEC Multi-access Edge Computing [2](#), [10](#), [18](#)

MPC Model Predictive Control [x](#), [xi](#), [4](#), [6](#), [7](#), [10](#), [11](#), [18](#), [22](#), [23](#), [26](#), [27](#), [32](#), [34](#), [35](#), [38](#), [41](#), [42](#), [44](#), [45](#), [47](#), [48](#), [50](#), [51](#), [54–59](#), [62](#), [63](#)

MQTT Message Queuing Telemetry Transport [11](#), [12](#), [55](#)

PID Proportional–integral–derivative controller [9](#), [16](#)

QoS Quality of Service [11](#), [12](#)

ROS Robot Operating System [iii](#), [3](#), [15](#), [38](#), [40](#), [55](#), [62](#)

TCP Transmission Control Protocol [11](#), [12](#)

UDP User Datagram Protocol [11](#)

Chapter 1

Introduction

In recent years, the demand for autonomous driving has surged, resulting in fewer accidents. A recent comparison between [automated driving System \(ADS\)](#) and human-operated vehicles [1] demonstrated an 80% reduction in reported injuries compared to human-operated vehicles. Additionally, the adoption of autonomous systems significantly optimizes traffic flow [2], and as reported in [3], it can lower greenhouse gas emissions by up to 34%. However, there are still cases in which [ADS](#) malfunctions lead to accidents. These failures often arise from partial environmental observability and the inherent limitations of onboard sensor systems.

To address these challenges, the adoption of [CCAS](#) has been increasing [4], [5]. In this approach figure 1.1, a central cloud computer gathers information from a network of multi-modal sensor nodes. It then fuses [6] the data from all sensors to create a comprehensive view of the environment, including the state of the ego vehicle. This enhanced context helps autonomous systems make better decisions by extending their sensing range and enabling multi-vehicle coordination. In [7], the authors demonstrate the advantages of cloud information sharing through examples such as pedestrians hidden by parked vehicles attempting to cross the road and vehicles obscured by buildings at intersections. Moreover, [CCAS](#) reduces the need for expensive onboard hardware such as LiDAR, 3D cameras, radars, and high-performance processing units by offloading complex computational tasks to the cloud. This minimizes the requirement for high-performance processors in every vehicle, thus cutting costs and reducing energy consumption.

Additionally, [CCAS](#) enables the use of shared sensing infrastructure, where high-end sensors such as HD LiDAR and cameras can be installed at fixed locations to cover the entire environment, including the states of the ego vehicle and other agents. This approach eliminates the need to equip every vehicle with expensive sensors while allowing the cloud

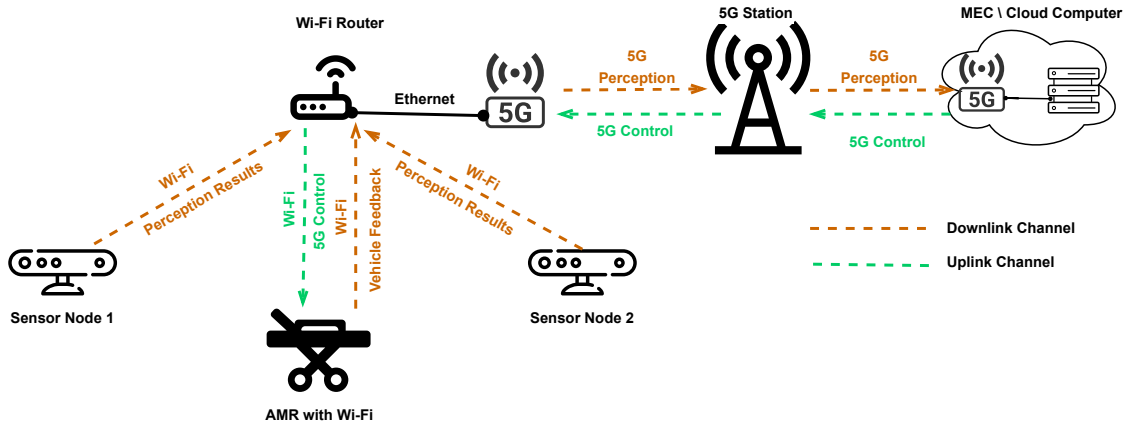


Figure 1.1: Example of a cloud-connected autonomous system.

The system consists of off-board sensor nodes that transmit processed perception data to the **Multi-access Edge Computing (MEC)**/cloud via Wi-Fi and 5G networks (shown as brown-dotted lines). The **MEC**/cloud control is responsible for sensor fusion to create a holistic view of the environment, including the state estimation of the ego vehicle and other agents, as well as trajectory planning. Control commands generated in the cloud are then transmitted back to the **AMR** over the network for execution (shown as green-dotted lines). Feedback from the **AMR**—including its speed and steering angle—is continuously sent to the cloud to help compute the next control action.

to maintain a comprehensive view of the environment, which improves its ability to plan the next action for each agent effectively.

Despite these advantages, cloud connectivity introduces new challenges. One primary concern is the added latency and message dropouts in closed-loop control caused by network constraints, which directly impact vehicle safety and performance. For fully autonomous driving, braking, acceleration, and lane changes must occur with near-instant response times. Even modest network delays—on the order of tens or hundreds of milliseconds—can degrade control quality. As shown in [8] for vehicle teleoperation, the total round-trip latency comprises sensor data acquisition, network transmission, cloud computation, and final actuation on the vehicle.

Another challenge in cloud-based control is security [9]. Security concerns in Connected and Autonomous Vehicle Systems (CCAS) include vulnerabilities to cyberattacks, such as hacking of communication channels and spoofing. To address this, an encryption-based acknowledgment protocol is essential to ensure data integrity and system reliability [10]. Although this thesis does not explicitly examine methods to mitigate security issues, the work outlined in [10] motivates the use of acknowledgment-based connections, which inherently induce latency—a topic discussed later in Chapter 4.

In [8], the uplink latency (section 3.4) was observed to be roughly constant at 60 ms,



(a) Prototype of the AMR.



(b) Prototype of an off-board LidAR.

Figure 1.2: Prototype of AMR and LiDAR.[6], [11]

whereas the downlink latency (section 3.4) follows a generalized extreme value distribution with a mean of 200 ms. Together, these result in an end-to-end latency (section 3.4) of approximately 250 ms. Such latencies significantly affect AMR motion-planning performance, especially when the control sampling time (ΔT_c^s) is much smaller (e.g., 100 ms) than the end-to-end latency. Under these conditions, the AMR may experience instability, poor tracking, or even collisions due to delayed feedback and command updates.

1.1 Thesis Contribution

This work focuses on the mobility of an AMR within a cloud-connected infrastructure, where the AMR in this study is an autonomous hospital bed figure 1.2a following an Ackermann steering model and operating in an indoor environment similar to the setup in figure 1.1. The primary contribution of this thesis is a latency-compensation scheme designed to handle network-induced delays exceeding a single control sampling period ΔT_c^s . This scheme is integrated into a real CCAS for indoor autonomous operations (figure 1.2, figure 1.3). A ROS-based simulation framework for CCAS was developed to streamline testing and validation of algorithms for latency. The proposed algorithm was tested under three network conditions: 1) a constant end-to-end latency greater than ΔT_c^s , 2) an end-to-end latency greater than ΔT_c^s with occasional message loss, and 3) an end-to-end latency greater than ΔT_c^s in an acknowledgment-based secure connection.

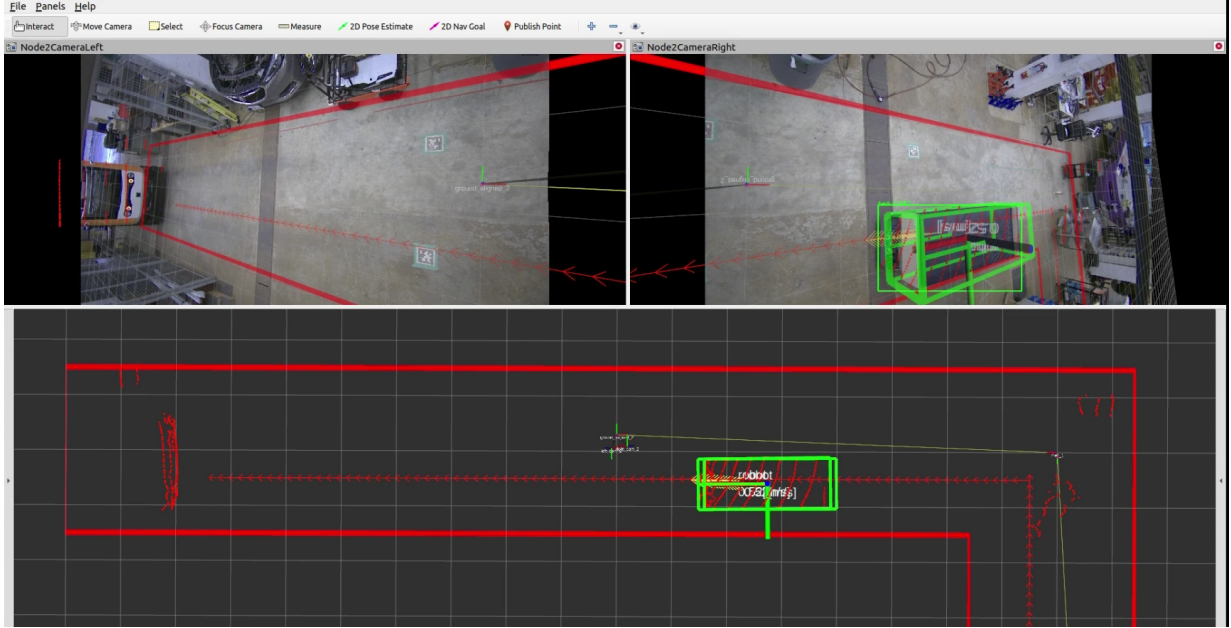


Figure 1.3: The [AMR](#) is tracking the reference path, controlled by the cloud computer.

1.2 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 reviews the literature and prior work on mitigating latency in [CCAS](#) and networked control systems. Chapter 3 presents the relevant technical background, clarifies key terminology, states the main assumptions, and formally defines the problem under consideration. Chapter 4 first outlines the design of the motion-planning algorithm, focusing on [MPC](#) under ideal conditions (i.e., without latency), and then extends this discussion by describing the proposed methodology for latency compensation. Chapter 5 presents simulation results for the scenarios discussed as part of the contributions, as well as results from real-world experiments. Finally, Chapter 6 concludes the thesis by summarizing key challenges and proposing directions for future research.

Chapter 2

Literature Review

2.1 Introduction

[CCAS](#) systems integrate distributed sensing, centralized computation, and localized actuation. The key challenge in such systems is managing latency, which arises from two primary sources: network communication and computational overhead.

Network communication delays occur in two main loops: the downlink and the uplink figure [1.1](#). Additionally, computational delays are introduced during the processing of data at individual sensors and in the cloud, where data from multiple sensors is fused and high-level decisions are made for each controlled agent figure [3.1](#).

Next, we briefly outline the various sources of latency in the context of [CCAS](#).

2.2 Latency in Cloud-Connected Autonomous Systems

- **Computation Delay at the Sensor Node:** In our current experimental setup, each sensor node processes its LiDAR data locally before transmitting the results to the central cloud. [\[12\]](#) investigated perception pipelines for high-speed autonomous racing, reporting that clustering and state estimation from the point cloud can take anywhere from 10 to 100 ms, largely depending on the number of objects and the overall complexity of the environment. In the current experimental setup, the computational time ranges from 10 to 25 ms.
- **Cloud Computation Delay:** At the cloud, latency stems from computation related to sensor fusion, trajectory planning, and control. Graph-based planning in a complex

environment could take approximately 100 to 110 ms [12]. Computing the control output also contributes to the latency, with a significant portion of runtime allocated to solving the optimization problem in MPC. The solver overhead typically scales with the number of iterations required and whether the underlying formulation is quadratic or nonlinear, adding about 10 to 20 ms.

- **Actuation Delay:** This delay represents the approximate dead time between when the control command is received by the AMR and when the actuator begins to apply it. For example, [13] measured a steering actuation delay of approximately 0.2 seconds.
- **Network Delay:** Network delay represents the time required to transmit data between different entities in the system, such as from the sensor node to the cloud computer or from the cloud computer to the AMR and vice versa. This delay depends on the processing time for encoding and decoding messages, available bandwidth, and the quality of service of the communication network. In the current experimental setup [6], communication between the sensor node, central node, and AMR occurs over 5G, where latency follows a Gaussian distribution with a mean of 50 ms and a standard deviation of 10 ms. However, as noted in [14], network latency can reach up to 150 ms when using 4G. For 5G, it could be up to 60 ms, and in some rare cases, as high as 200 ms.

2.3 Control Strategies for Latency Mitigation

As discussed in the previous section, the presence of various components in CCAS may result in undesirably large delays in the control loop, leading to a degradation in control performance [15], [16]. For instance, the work in [15] shows the effect of latency in the in-vehicle network on control performance, reporting that a closed-loop delay of 40 ms with the traditional scheme increases the response time to changes by 17%. This increase in response time is detrimental, especially at high speeds.

Prediction-based control is an intuitive method for actively managing systems with longer delays, primarily because it leverages predicted future system states to design a control strategy that compensates for the instability caused by the delay [17].

One of the earliest works in the literature on prediction-based control strategies for latency compensation dates back to the 1950s, when the “Smith predictor” was proposed [18] to compensate for the long dead time of a process and significantly improve control performance. The basic idea of the Smith predictor is to introduce an internal model

to predict the process’s delay-free response. This prediction is then used in feedback to calculate the difference from the desired output (output error) and adjust the corresponding control action.

Recently, state augmentation combined with MPC methods has become quite popular for mitigating latency within the in-vehicle network. For instance, [19] presents a closed-loop state-augmentation solution that compensates for the dead time between when a command is planned and when it reaches the actuator. The idea is to transform the delayed state-space dynamics into a delay-free system by augmenting the system state vector [20]. The work in [13] further leveraged this approach by proposing a state-augmentation method to compensate for steering lag that exceeds the sampling period. In addition, they integrated a predictor model to estimate the error after the delayed time steps and account for it when solving for the optimal control action at the current time. However, these approaches focus solely on input delays and do not explicitly account for feedback delays.

Strategies for mitigating the effects of feedback delays have also been explored, with prediction-based approaches becoming increasingly significant in this context. For example, the work in [21] proposed compensating for constant feedback delay due to sensor computation by transforming sensor readings so that they reflect the current state rather than the state at the time of recording. More recently, a popular technique is to forward-simulate the sampled state over the duration of the delay using an internal dynamics model to predict future states. This approach has been integrated with Kalman filter-based robust estimation in several autonomous driving stacks, such as [22].

Contrary to the state-augmentation method for compensating input delay, prediction-based delay compensation is widely used in network control system, which consists of complex robotic systems where linearization and solving the problem using state-space techniques are difficult and often produce suboptimal solutions. This approach predicts the system’s future state using past feedback and a buffer storing past control actions while accounting for both downlink and uplink latencies.

This method was first explored in [23], where state predictions were utilized to design a feedback control law that guarantees closed-loop stability for a linear time invariant system in a network controlled system. Extending this work, [24] addressed the remote control of a linear time invariant system over a lossy network under both state and control constraints and proposed an MPC framework to compute the optimal control for tracking a constant reference. [25] further extended this work by applying it to a nonlinear system.

More recently, such methods have been explored and integrated with learning techniques to compute the optimal control in the presence of both downlink and uplink latency. For example, [26] employed an autoencoder model to predict the future states of the leader

and follower manipulators in robotic tele-control operations under periodic bidirectional time delays.

Our work draws inspiration from prediction-based schemes in network control systems. We integrate sensor computational delay, network delay, and input delay while addressing control and state constraints — all within a unified optimization framework tailored for [CCAS](#).

An added benefit of our approach is its modular design, which is independent of the specific prediction-based control method employed. For instance, one could integrate the Sampling-Based Predictive Planner [\[27\]](#), [\[28\]](#) as part of the planning strategy instead of relying on the state-space optimization technique, which produces suboptimal solutions in some scenarios, such as in dense environments [\[28\]](#).

Chapter 3

Background

This chapter provides a concise overview of Cloud-Connected Autonomous Systems, defines key terminologies and assumptions relevant to this work.

3.1 Cloud-Connected Autonomous Systems

The following defines the structure, components, and characteristics of a [CCAS](#), highlighting its fundamental properties and operational principles. The overall structure can be divided into three parts:

- [AMR](#): The [AMR](#) executes control commands received from the cloud controller. It employs a low-level tracking controller, such as a [Proportional–integral–derivative controller \(PID\)](#), to track high-level reference commands. For example, the cloud controller computes the desired speed and steering angle references, which the on-board [PID](#) controller ensures the robot follows. The robot is equipped with encoders that measure both wheel speed and steering, transmitting this feedback to the cloud controller.
- Off-board sensors: The perception system relies on a network of multiple sensor nodes, which include LiDAR, cameras, and wireless communication modules [\[6\]](#). These sensors operate collaboratively to enhance perception in dynamic environments. Each sensor node processes data locally and transmits structured perception results to the cloud controller. The system ensures synchronization across sensor nodes, which is required for latency mitigation at the cloud controller.

- Cloud computer: The cloud computer consists of two primary components:
 - Global Perception and Localization (PL): This module receives processed feedback from off-board sensors and fuses data from multiple sources to generate a unified environmental model. It estimates the vehicle’s full-state representation, integrating perception data with vehicle feedback (speed and steering measurements).
 - Cloud Controller $g(\cdot)$: Using the estimated state from the PL module, the cloud controller runs the trajectory-planning algorithm to compute optimal control commands. The control strategy accounts for system latencies, ensuring robust and real-time decision-making in a delayed environment.
- MEC: MEC [5] is an emerging architecture that brings traditional cloud computing capabilities closer to data sources, typically at or near the base station or gateway. This proximity reduces end-to-end communication delays by offloading critical computation from a distant data center to an edge server.

Although this work describes a CCAS in terms of a generic cloud-based controller, the same structure and concepts apply when replacing a cloud data center with a MEC node. In practical terms, using a MEC server for sensor fusion, global perception, and control can reduce latency and mitigate the impact of network congestion while retaining the same high-level system architecture—namely, a remote controller, the AMR, and off-board sensors.

3.2 Model Predictive Control (MPC)

MPC is an optimal control method that minimizes a cost function for a constrained system over a finite, receding horizon. At each step, the controller estimates the system state $\zeta(t)$, solves the optimization problem (3.1) to compute a control sequence, and applies only the first action, repeating the process at the next step [29].

The terms $q(\zeta(k), u(k))$ and $p(\zeta(N))$ are referred to as the stage and terminal cost.

$$\begin{aligned}
J^*(\zeta(t), u(t)) = \min_U \quad & J(\zeta(t), u(t), U) = p(\zeta(N)) + \sum_{k=0}^{N-1} q(\zeta(k), u(k)) \\
\text{s.t.} \quad & \zeta(k+1) = f_a(\zeta(k), u(k)), \quad k = 0, \dots, N-1 \\
& \zeta(k) \in \mathcal{X}, \quad u(k) \in \mathcal{U}, \quad k = 0, \dots, N-1 \\
& \zeta(N) \in \mathcal{X}_f \\
& \zeta(0) = \zeta(t).
\end{aligned} \tag{3.1}$$

where $U = \{u(0), \dots, u(N-1)\}$ and $f_a(\zeta(k), u(k))$ represents the discretized system dynamics.

The procedure for the MPC is summarized in algorithm 1.

Algorithm 1 Model Predictive Control (MPC)

```

1: while true do                                ▷ Execute at each control sampling time  $\Delta T_c^s$ 
2:   Require  $\zeta(t), u(t)$ 
3:   Obtain  $U^*$  by solving the optimization problem (3.1)
4:   if problem infeasible then
5:     Stop
6:   end if
7:   Return the first element  $u^*(0)$  of  $U^*$ 
8:   Wait for the next sampling time  $\Delta T_c^s$         ▷  $\Delta T_c^s$  is the controller sampling time.
9: end while

```

3.3 Communication Channel

The communication channel in a CCAS is responsible for transmitting sensor data and control commands between the cloud controller and autonomous agents. Although specific protocols such as Message Queuing Telemetry Transport (MQTT), Transmission Control Protocol (TCP), or User Datagram Protocol (UDP) can be used for this exchange, the proposed latency-compensated CCAS design remains independent of any particular networking framework.

In many practical deployments, fire-and-forget-based communication (e.g., UDP or low MQTT Quality of Service (QoS) levels) is sufficient for time-critical messages such as

sensor updates and control signals. In this type of transmission method, the sender can transmit a message without waiting for confirmation or feedback from the receiver. In these cases, minimal delay and reduced overhead are often prioritized over guaranteed delivery. However, when latency exceeds one control sampling period, instability can arise, necessitating compensation strategies (discussed later in this work).

Certain safety-critical scenarios (addressed in Section 4.7) may benefit from acknowledgment-based protocols (e.g., higher MQTT QoS levels or TCP) to ensure reliable message transmission. Even under higher QoS levels, parameters such as *in-flight messages* can be configured to retain performance characteristics similar to best-effort protocols while still providing fault tolerance. We refer to this type of network communication as acknowledgment-driven communication.

In this work, experiments primarily employ MQTT for message transfer. Although higher QoS levels in MQTT inherently provide acknowledgments between the sender and broker, as well as between the broker and recipient, they do not directly confirm delivery from the sender to the recipient [30]. In principle, this limitation can be addressed by introducing an application-layer mechanism whereby the autonomous system sends explicit confirmation messages back to the cloud controller. However, this mechanism was not fully utilized in the experiments conducted here. While the vehicle does transmit status messages—indicating which control command is being executed and providing data such as speed—the control algorithm is triggered on a fixed schedule rather than waiting for explicit acknowledgments (section 4.7). This design choice was made to simplify the software implementation. Consequently, the framework remains highly adaptable across diverse network conditions, infrastructure constraints, and reliability requirements.

3.4 Latency in the Cloud-Controlled System

Taking the cloud controller as the reference point, latency in the system can be divided into two parts, as described below.

Downlink Latency: τ_{dl}

Downlink latency (figure 3.1a) represents the delay in transmitting a sensor’s message from sensor nodes and/or the feedback from the AMR to the cloud controller [31]. Factors such as network congestion, wireless interference, and processing overhead contribute to variations in this delay. Additionally, downlink latency may also include the sensor’s

computational delay. In this work, during the experimental setup of [CCAS](#), the sensor data undergoes preprocessing before being transmitted over the network to the cloud. This step reduces the bandwidth required compared to sending the raw sensor data. These delays can be treated as a single lumped contribution (figure 3.1b) [8].

Uplink Latency: τ_{ul}

Uplink latency (figure 3.1a) refers to the time required for control commands to propagate from the cloud controller to the autonomous system. τ_{ul} may include the computational delay of the PL module, the trajectory planning algorithm, and the actuation processing delay. These delays can also be treated as a single lumped contribution (figure 3.1b).

End-to-End Latency

End-to-end latency refers to the total time from when the sensor captures a message to when the corresponding control command, based on that message, is executed by the [AMR](#). It is the sum of the downlink and uplink latencies, including any additional processing delays at the cloud and actuation delays at the autonomous system.

3.5 Problem Statement

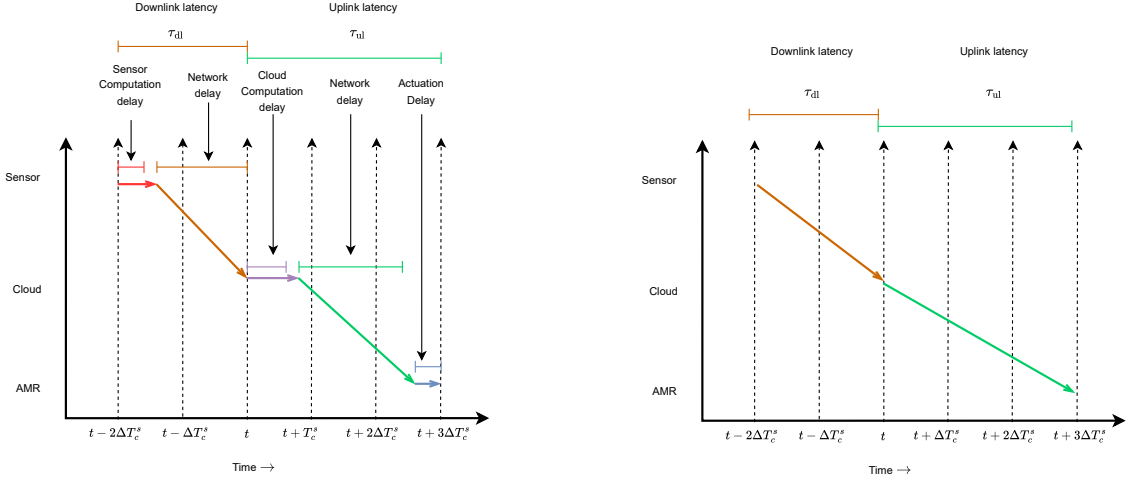
This work aims to develop a unified framework for latency-aware motion planning and control in [CCAS](#), capable of adapting to varying network conditions. The objective is to design a robust system that ensures stable and responsive vehicle operation across different network conditions.

Let $\zeta(t) \in \mathbb{R}^{N_\zeta}$ denote the state of an [AMR](#) at time t , where N_ζ represents the dimension of the state vector. The state evolves according to the following nonlinear dynamics:

$$\dot{\zeta}(t) = f(\zeta(t), u(t)) \quad (3.2)$$

where $u(t)$ is the control input. The robot operates in a [CCAS](#) subject to network uplink τ_{ul} and downlink τ_{dl} latencies, each characterized by a constant mean greater than one control sampling time (ΔT_c^s) and small deviation.

The [AMR](#) is tasked with tracking the desired reference trajectory $\zeta_{ref}(t)$, which defines the reference path and velocity profiles.



(a) Breakdown of uplink and downlink latency, showing contributions from sensor, network, cloud, and actuation delays.

(b) Simplified timing diagram illustrating the overall uplink and downlink latency between sensor, cloud, and AMR in CCAS.

Figure 3.1: Timing diagram illustrating various delays in CCAS.

Mathematically, we seek a control law $u(t)$ that solves the following optimization problem:

$$\min_{u(t)} J(e(t), u(t)) \quad \text{subject to} \quad \begin{cases} \dot{\zeta}(t) = f(\zeta(t), u(t)), \\ u(t) = l(\hat{\zeta}(t + \tau_{ul}), \hat{u}(t + \tau_{ul})), \\ \zeta(t) \in \mathcal{X}, \quad \forall t \in [0, T], \\ u(t) \in \mathcal{U}, \quad \forall t \in [0, T] \end{cases} \quad (3.3)$$

J is a user-defined cost function that balances tracking accuracy and control effort. The tracking error is defined as:

$$e(t) = \zeta(t) - \zeta_{\text{ref}}(t)$$

The set $\mathcal{X} \subseteq \mathbb{R}^{N_\zeta}$ defines the allowable state space, which captures physical constraints such as position, velocity, and acceleration limits. The set $\mathcal{U} \subseteq \mathbb{R}^{N_u}$ defines the allowable control input space, which includes actuator limits such as maximum steering angle and acceleration. Here, $l(\hat{\zeta}(t + \tau_{ul}), \hat{u}(t + \tau_{ul}))$ is the model predictive control law, which will be discussed in a later chapter.

The overall objective is to provide a real-time control solution that mitigates latency effects and can be executed reliably on a cloud-based infrastructure.

The proposed framework will be validated through both simulation and real-world deployment, demonstrating its effectiveness in handling real-time cloud-based control.

3.6 Robot Operating System (ROS)

ROS [32] provides a flexible and modular framework for developing robotic software. It consists of a comprehensive set of tools and libraries that streamline the integration of complex robotic systems. By enabling efficient communication between different components, such as sensors and actuators, ROS simplifies system development and enhances interoperability.

In this work, each component of the CCAS operates within its own ROS middleware. The sensor node runs a dedicated ROS instance to process data before transmitting it to the cloud computer. Similarly, the cloud computer utilizes its own ROS middleware to handle incoming sensor data and relay control commands to the AMR, ensuring a seamless flow of information across the system.

3.7 Assumptions

The running assumptions throughout this work are as follows:

- Synchronization of network components: All components of the network system, including off-board sensors, the vehicle, and the cloud controller, are assumed to be time-synchronized [6]. During experiments, synchronization was achieved by connecting all components to the same internet source, ensuring a common time reference.
- Modeling of network conditions: In this work, 5G communication network latency and message dropout conditions were artificially introduced in both the simulation and experiments based on insights from previous studies [6], [33]. Specifically, network-induced latency characteristics were defined as a Gaussian distribution, and packet loss was assumed to follow a Bernoulli distribution. This approach ensures that the simulation environment and experimental setup realistically reflect network behavior expected in cloud-connected autonomous systems. All network communications during the experiments relied solely on Wi-Fi.
- Time alignment of sensor and vehicle messages: The latency from the vehicle to the cloud controller and from the sensor nodes to the cloud controller is assumed

to be approximately equal or to differ only marginally. This ensures that the vehicle’s state representation—comprising its (x, y, yaw) coordinates (as processed by off-board sensors) and its wheel speed and steering angle (as directly measured on-board)—corresponds to the same physical state at a given point in time.

To address any discrepancies in latency, one could estimate the vehicle’s COG velocity and heading angle using off-board sensors such as LiDAR, which track the vehicle’s motion over time. This estimated COG velocity and heading can then be used in a kinematic model (equation 4.1) to infer the vehicle’s wheel speed and steering angle. Alternatively, a dual-buffer approach can be implemented:

- Buffer 1: Stores states sampled from off-board sensors (e.g., LiDAR) along with their respective timestamps.
- Buffer 2: Stores vehicle speed and steering feedback with timestamps.

These buffers can then be time-aligned to reconstruct an approximate representation of the vehicle’s state (figure 3.2).

- Perfect sensing: We assume sensor data is perfect—meaning that LiDAR readings for (x, y, yaw) and encoder measurements for steering angle and wheel speeds reflect exact values. Given the slow operating speeds during our experiments, minor sensor noise was considered negligible.
- Simplified kinematics model: In modeling the controller’s dynamics, we did not include the steering dynamics and assumed that steering is applied instantaneously. However, we imposed a bound on the change in the steering angle between two sampling times. During experiments, we used a PID controller to track the commanded reference steering and speed. Although this mismatch introduces some error, it was sufficient to demonstrate improved performance compared to the baseline. In simulation, we modeled the dynamics identically to the controller’s dynamics equation (equation 4.1). We observed that incorporating steering dynamics into the state space poses challenges that require further refinement, which we plan to explore in future work.

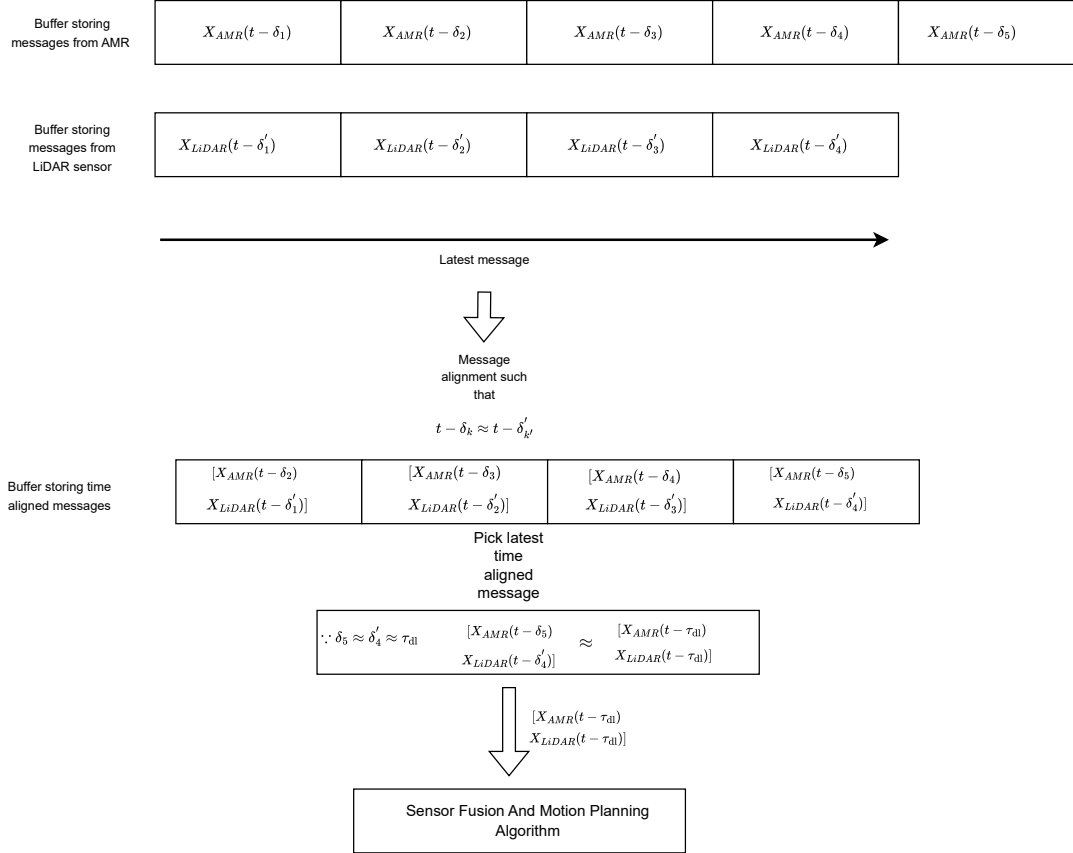


Figure 3.2: The cloud controller maintains two buffers: one storing sampled states from LiDAR (X_{LiDAR}) and the other storing [AMR](#) feedback (X_{AMR}). When the cloud-control algorithm is triggered, the messages are time-aligned, and the most recent time-aligned message (which carries the complete state of the [AMR](#)) is forwarded to the cloud controller.

Chapter 4

Methodology

In this chapter, we first present the nominal [MPC](#) design procedure for the considered [AMR](#) and then extend it to account for latency in the closed-loop control system.

4.1 Closed-Loop Pipeline for Cloud-Controlled Autonomous Robot

The closed-loop control pipeline considered and evaluated in this work is illustrated in figure [4.1](#). The pipeline comprises the following key steps:

1. Sensor data acquisition: Sensor nodes sample and process state information of the [AMR](#) and its surrounding environment, producing useful outputs such as bounding-box data for vehicles and obstacles.
2. Data transmission to the cloud: Processed sensor data (from off-board sensors such as LiDAR) and vehicle state information are packaged and transmitted to the [MEC](#) node over the network, incurring downlink latency.
3. Global perception and localization: The global perception and localization module integrates sensor data to form a global understanding of the environment. This module may also generate the reference path for the [AMR](#), which is assumed to be known for this work.

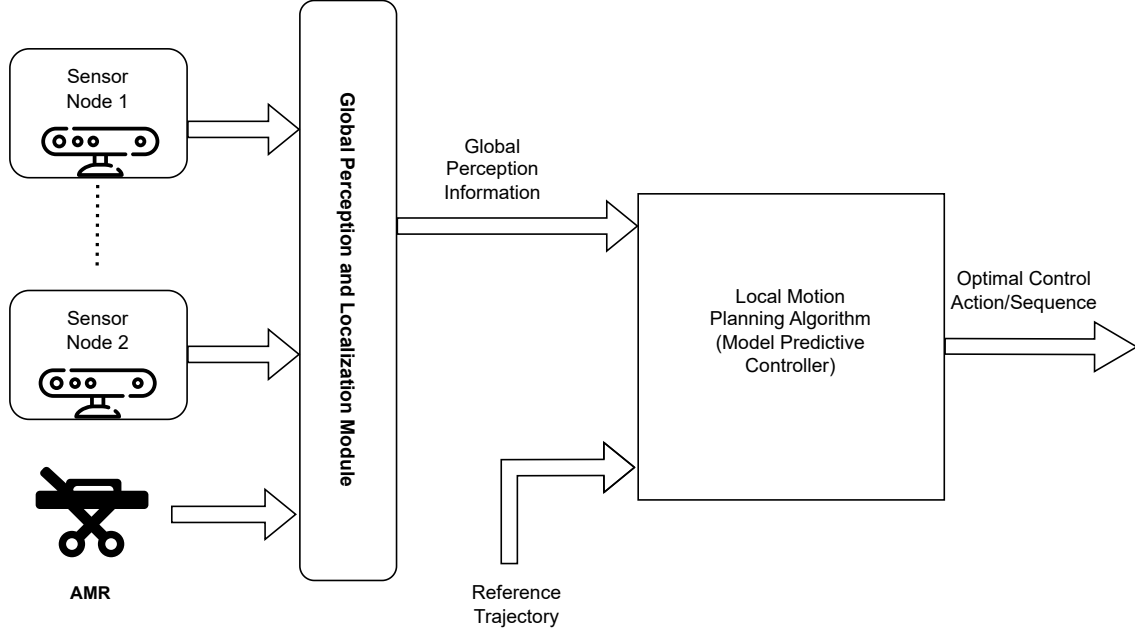


Figure 4.1: Overview of the pipeline.

4. Cloud-based control computation: A cloud-based control algorithm (e.g., a Model Predictive Controller) utilizes fused sensor data and compensates for both uplink and downlink latency to compute optimal control actions.
5. Control action transmission: The computed control actions are packaged and sent back to the [AMR](#), experiencing uplink latency, to execute local motion planning.

4.2 Motion Model Description

Our motion model for the [AMR](#) closely follows the work outlined in [11]. The motion of the [AMR](#) is represented using a three-degree-of-freedom bicycle model [34]. This model assumes a rigid body and neglects tire slippage (figure 4.2).

The variables X , Y , $\dot{X}$, $\dot{Y}$, and $\dot{\theta}$ represent the x-coordinate, y-coordinate, longitudinal velocity, lateral velocity, and angular velocity at the [COG](#) of the robot in the global coordinate frame.

The variables v_c and β denote the velocity at the **COG** and the sideslip angle in the body coordinate frame, respectively. l_f and l_r represent the distances from the **COG** to the front and rear wheels. The front and rear wheel velocities, steering angles, and accelerations are represented by v_f , v_r , δ_f , δ_r , a_f , and a_r , respectively.

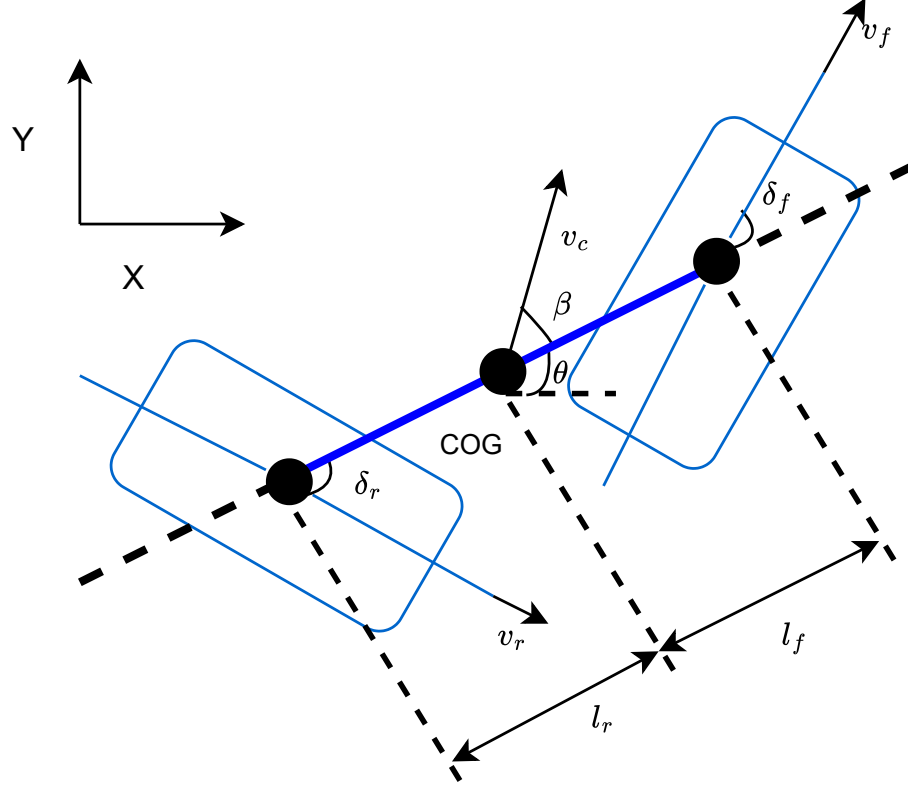


Figure 4.2: Kinematics model of **AMR**[34] using a three-degree-of-freedom bicycle model, showing the **COG**, front and rear wheel velocities (v_f, v_r), steering angles (δ_f, δ_r), and distances of the front and rear wheel from the **COG** (l_f, l_r).

The non-linear kinematic equations $\dot{\zeta} = f(\zeta, u)$, governing the motion of the robot is

given by the equation 4.1:

$$\begin{aligned}
\dot{X} &= v_c \cos(\theta + \beta) \\
\dot{Y} &= v_c \sin(\theta + \beta) \\
\dot{\theta} &= \frac{v_c \cos(\beta) [\tan(\delta_f) - \tan(\delta_r)]}{l_f + l_r} \\
\dot{v}_f &= a_f \\
\dot{v}_r &= a_r \\
v_c &= \frac{v_f \cos(\delta_f) + v_r \cos(\delta_r)}{2 \cos(\beta)} \\
\beta &= \tan^{-1} \left(\frac{l_r \tan(\delta_f) + l_f \tan(\delta_r)}{l_f + l_r} \right)
\end{aligned} \tag{4.1}$$

The system states ζ , outputs η , and control inputs u are defined as:

$$\zeta(t) = \eta(t) = [X(t) \quad Y(t) \quad \theta(t) \quad v_f(t) \quad v_r(t)]^T \tag{4.2}$$

$$u(t) = [a_f(t) \quad a_r(t) \quad \delta_f(t) \quad \delta_r(t)]^T \tag{4.3}$$

The nonlinear system equations are linearized around the operating point $\hat{\zeta}(t')$ and $\hat{u}(t')$, which represent the estimated state and the previously estimated control input, respectively. The next computed optimal control input will be applied at the estimated time t' .

In the ideal case, when there's no latency in the close-loop and where we have perfect knowledge of the system, these values correspond to the state and the control to the AMR at the current time t figure 4.3.

The linearized dynamics equation at time t is shown in equation 4.4

$$\begin{aligned}
\dot{\zeta}(t) &= A(t)\zeta(t) + B(t)u(t) + d(t) \\
A(t) &= \frac{\partial f(\zeta(t), u(t))}{\partial \zeta(t)} \\
B(t) &= \frac{\partial f(\zeta(t), u(t))}{\partial u(t)} \\
d(t) &= f(\zeta(t), u(t)) - A(t)\zeta(t) - B(t)u(t)
\end{aligned} \tag{4.4}$$

Since the [MPC](#) formulation in algorithm 1 is defined in discrete time, the linearized dynamics (equation 4.4) are discretized at every call to algorithm 1, as shown in equation 4.5.

$$\begin{aligned}\zeta(k+1) &= A(k)\zeta(k) + B(k)u(k) + d(k) \\ \eta(k+1) &= C(k)\zeta(k+1) \\ C(k) &= I\end{aligned}\tag{4.5}$$

where $k = t$, $k+1 = t + \Delta T_c^s$, $k+2 = t + 2\Delta T_c^s$, $\dots$, $k+N = t + N\Delta T_c^s$. Here, ΔT_c^s is the control sampling time, and N is the [MPC](#) horizon length defined by the formulation.

$A(k), B(k), d(k)$ remain constant throughout the horizon length defined by [MPC](#) since we keep the operating point ($\hat{\zeta}(t)$ and $\hat{u}(t)$) constant.

We define a constraint on the rate of change of the control input $u(k)$, i.e.,

$$\Delta \underline{u} \leq u(k) - u(k-1) \leq \Delta \bar{u}$$

through the state-space augmentation technique, where $\Delta u(k)$ will be the input to the augmented state-space and $u(k)$ will be part of the state vector.

Now, the control input follows the form

$$u(k) = u(k-1) + \Delta u(k),$$

so equation 4.5 can be rewritten as:

$$\begin{aligned}\tilde{\zeta}(k+1) &= \tilde{A}(k)\tilde{\zeta}(k) + \tilde{B}(k)\Delta u(k) + \tilde{d}(k) \\ \eta(k+1) &= \tilde{C}(k)\tilde{\zeta}(k+1)\end{aligned}\tag{4.6}$$

where,

$$\tilde{A}(k) = \begin{bmatrix} A(k) & B(k) \\ 0_{N_u \times N_\zeta} & I_{N_u} \end{bmatrix}, \quad \tilde{B}(k) = \begin{bmatrix} B(k) \\ I_{N_u} \end{bmatrix}, \quad \tilde{d}(k) = \begin{bmatrix} d(k) \\ 0_{N_u \times 1} \end{bmatrix}, \quad \tilde{C}(k) = [C(k) \quad 0_{N_\zeta \times N_u}]\tag{4.7}$$

We define N_ζ as the number of system states and N_u as the number of control inputs. The augmented state vector $\tilde{\zeta}(k)$ and incremental control input $\Delta u(k)$ are defined as:

$$\tilde{\zeta}(k) = \begin{bmatrix} \zeta(k) \\ u(k-1) \end{bmatrix}, \quad \Delta u(k) = u(k) - u(k-1)$$

4.3 Normalization

In this work, we have employed normalization techniques, including state-space normalization, constraint normalization, and cost normalization. The application of normalization in MPC has been studied and has demonstrated significant improvements. These improvements include mitigating ill-conditioning in optimization problems, enhancing numerical stability, faster and more reliable convergence, and preventing the disproportionate influence of specific state variables or control inputs in the cost function [35]. The normalization process is outlined as follows:

4.3.1 Scaling Matrices

To normalize the states and inputs, we use scaling matrices T_ζ , T_u , and $T_{\Delta u}$, such that following holds :

$$\begin{aligned} \zeta_{\text{norm}} &= T_\zeta \zeta, & u_{\text{norm}} &= T_u u, & \Delta u_{\text{norm}} &= T_{\Delta u} \Delta u \\ \zeta &= T_\zeta^{-1} \zeta_{\text{norm}}, & u &= T_u^{-1} u_{\text{norm}}, & \Delta u &= T_{\Delta u}^{-1} \Delta u_{\text{norm}} \end{aligned} \quad (4.8)$$

Here T_ζ , T_u , $T_{\Delta u}$ are the diagonal matrix defined as:

$$\begin{aligned} T_\zeta &:= \begin{bmatrix} \frac{1}{\max(X)} & & & & \\ & \frac{1}{\max(Y)} & & & \\ & & \frac{1}{\max(\theta)} & & \\ & & & \frac{1}{\max(v_f)} & \\ & & & & \frac{1}{\max(v_r)} \end{bmatrix} \\ T_u &:= \begin{bmatrix} \frac{1}{\max(a_f)} & & & & \\ & \frac{1}{\max(a_r)} & & & \\ & & \frac{1}{\max(\delta_f)} & & \\ & & & \frac{1}{\max(\delta_r)} & \end{bmatrix} \\ T_{\Delta u} &:= T_u \end{aligned} \quad (4.9)$$

The last definition ($T_{\Delta u} := T_u$) is required to follow the constraints, $u_k - u_{k-1} = \Delta u_k$

4.3.2 Normalized State-Space Dynamics

Substituting un-normalized states and control in equation 4.6 by equation 4.8, the discretized augmented state space becomes:

$$\begin{aligned}\tilde{\zeta}_{\text{norm}}(k+1) &= \tilde{A}_{\text{norm}}(k)\tilde{\zeta}_{\text{norm}}(k) + \tilde{B}_{\text{norm}}(k)\Delta u_{\text{norm}}(k) + \tilde{d}_{\text{norm}}(k) \\ \tilde{\eta}_{\text{norm}}(k+1) &= \tilde{C}(k)\tilde{\zeta}_{\text{norm}}(k+1)\end{aligned}\quad (4.10)$$

where, the normalized matrices are:

$$\tilde{A}_{\text{norm}}(k) = \begin{bmatrix} T_{\zeta}A(k)T_{\zeta}^{-1} & T_{\zeta}B(k)T_u^{-1} \\ 0 & I \end{bmatrix}, \quad \tilde{B}_{\text{norm}}(k) = \begin{bmatrix} T_{\zeta}B(k)T_{\Delta u}^{-1} \\ I \end{bmatrix}, \quad \tilde{d}_{\text{norm}}(k) = \begin{bmatrix} T_{\zeta}d(k) \\ 0 \end{bmatrix} \quad (4.11)$$

4.3.3 Normalized Cost Function

State reference Cost Term

This cost represents the weighted difference between the reference state and the predicted state of the AMR at time step k :

The original state reference cost is: $(\zeta(k) - \zeta_{\text{ref}}(k))^T Q (\zeta(k) - \zeta_{\text{ref}}(k))$

Substituting $\zeta(k) = T_{\zeta}^{-1}\tilde{\zeta}_{\text{norm}}(k)$

$$\begin{aligned}(\zeta(k) - \zeta_{\text{ref}}(k))^T Q (\zeta(k) - \zeta_{\text{ref}}(k)) &= \tilde{\zeta}_{\text{norm}}(k)^T (T_{\zeta}^{-T} Q T_{\zeta}^{-1}) \tilde{\zeta}_{\text{norm}}(k) \\ &\quad - 2\zeta_{\text{ref}}(k)^T Q T_{\zeta}^{-1} \tilde{\zeta}_{\text{norm}}(k) + \zeta_{\text{ref}}(k)^T Q \zeta_{\text{ref}}(k)\end{aligned}$$

Let $Q_{\text{norm}} := T_{\zeta}^{-T} Q T_{\zeta}^{-1}$

$$q_{\text{norm}} := (-2\zeta_{\text{ref}}(k)^T Q T_{\zeta}^{-1})^T$$

$$\begin{aligned}\therefore \text{The state reference cost becomes:} &= \tilde{\zeta}_{\text{norm}}(k)^T Q_{\text{norm}} \tilde{\zeta}_{\text{norm}}(k) + q_{\text{norm}}^T \tilde{\zeta}_{\text{norm}}(k) \\ &\quad + \zeta_{\text{ref}}(k)^T Q \zeta_{\text{ref}}(k)\end{aligned}\quad (4.12)$$

$\zeta_{\text{ref}}(k)$ is the reference state at time-step k .

Control Input Cost Term

This cost represents the weighted magnitude of the control effort:

$$\begin{aligned}
\text{The original control input cost is: } & u(k)^T R u(k) \\
\text{Substituting } u(k) = T_u^{-1} u_{\text{norm}}(k) & \\
u(k)^T R u(k) = (T_u^{-1} u_{\text{norm}}(k))^T R (T_u^{-1} u_{\text{norm}}(k)) & \\
= u_{\text{norm}}(k)^T (T_u^{-T} R T_u^{-1}) u_{\text{norm}}(k) & \\
\text{Let } R_{\text{norm}} := T_u^{-T} R T_u^{-1} & \\
\therefore u(k)^T R u(k) = u_{\text{norm}}(k)^T R_{\text{norm}} u_{\text{norm}}(k) & \quad (4.13)
\end{aligned}$$

Incremental Control Input Cost Term

This cost represents the weighted change in control input between two time steps:

$$\begin{aligned}
\text{The original incremental control cost is: } & \Delta u(k)^T R_d \Delta u(k) \\
\text{Substituting } \Delta u(k) = T_{\Delta u}^{-1} \Delta u_{\text{norm}}(k) & \\
\Delta u(k)^T R_d \Delta u(k) = (T_{\Delta u}^{-1} \Delta u_{\text{norm}}(k))^T R_d (T_{\Delta u}^{-1} \Delta u_{\text{norm}}(k)) & \\
= \Delta u_{\text{norm}}(k)^T (T_{\Delta u}^{-T} R_d T_{\Delta u}^{-1}) \Delta u_{\text{norm}}(k) & \\
\text{Let } R_{d, \text{norm}} := T_{\Delta u}^{-T} R_d T_{\Delta u}^{-1} & \\
\therefore \Delta u(k)^T R_d \Delta u(k) = \Delta u_{\text{norm}}(k)^T R_{d, \text{norm}} \Delta u_{\text{norm}}(k) & \quad (4.14)
\end{aligned}$$

Normalized Bounds

State Bounds

The original bounds are: $\underline{\zeta} \leq \zeta(k) \leq \bar{\zeta}$.

Normalize using $\zeta_{\text{norm}}(k) = T_{\zeta} \zeta(k)$,

$$\underline{\zeta}_{\text{norm}} = T_{\zeta} \underline{\zeta}, \quad \bar{\zeta}_{\text{norm}} = T_{\zeta} \bar{\zeta} \quad (4.15)$$

Control Input Bounds

The original bounds are: $\underline{u} \leq u(k) \leq \bar{u}$.

Normalize using $u_{\text{norm}}(k) = T_u u(k)$,

$$\underline{u}_{\text{norm}} = T_u \underline{u}, \quad \bar{u}_{\text{norm}} = T_u \bar{u} \quad (4.16)$$

Incremental Control Input Bounds

The original bounds are: $\Delta \underline{u} \leq \Delta u(k) \leq \Delta \bar{u}$

Normalize using $\Delta u_{\text{norm}}(k) = T_{\Delta u} \Delta u(k)$,

$$\Delta \underline{u}_{\text{norm}} = T_{\Delta u} \Delta \underline{u}, \quad \Delta \bar{u}_{\text{norm}} = T_{\Delta u} \Delta \bar{u} \quad (4.17)$$

Since we are using the augmented state-space equation, we redefine the weight matrices as shown in equation 4.18:

$$\begin{aligned} \tilde{Q}_{\text{norm}} &= \begin{bmatrix} Q_{\text{norm}} & \\ & R_{\text{norm}} \end{bmatrix}, \quad \tilde{q}_{\text{norm}} = \begin{bmatrix} q_{\text{norm}} \\ 0 \end{bmatrix}, \quad \tilde{R}_{\text{d,norm}} = R_{\text{d}} \\ \tilde{\underline{\zeta}}_{\text{norm}} &= [T_{\zeta} \underline{\zeta} \quad T_u \underline{u}], \quad \tilde{\bar{\zeta}}_{\text{norm}} = [T_{\zeta} \bar{\zeta} \quad T_u \bar{u}] \\ \Delta \tilde{u}_{\text{norm}} &= T_{\Delta u} \Delta \underline{u}, \quad \Delta \tilde{\bar{u}}_{\text{norm}} = T_{\Delta u} \Delta \bar{u} \end{aligned} \quad (4.18)$$

4.4 Predictive Controller Design

To this end, we define the final form of the MPC optimization problem with a control sampling interval ΔT_c^s . A finite-horizon optimal control problem $l(\zeta(t), u(t))$ (equation 4.19) is solved using the predicted state and control input as initial conditions (in the ideal case, i.e., when there is no latency, the predicted state equals the current state figure 4.3). The solution of equation 4.19 yields an optimal control sequence U^* , from which only the first control input is applied to the system. This process is repeated at the next sampling instance (algorithm 1).

$$\begin{aligned}
J^*(\zeta(t), u(t)) &= \min_{\Delta u_0, \dots, \Delta u_{N-1}} J(\zeta(t), \zeta_{\text{ref}}(t), u(t), \Delta u_0, \dots, \Delta u_{N-1}) \\
&= \tilde{\zeta}_{\text{norm}}(N)^T \tilde{Q}_{\text{norm}, N} \tilde{\zeta}_{\text{norm}}(N) + \tilde{q}_{\text{norm}, N}^T \tilde{\zeta}_{\text{norm}}(N) \\
&\quad + \sum_{k=0}^N \left(\tilde{\zeta}_{\text{norm}}(k)^T \tilde{Q}_{\text{norm}} \tilde{\zeta}_{\text{norm}}(k) + \tilde{q}_{\text{norm}}^T \tilde{\zeta}_{\text{norm}}(k) + \tilde{R}_{\text{d}, \text{norm}}^T \Delta u_{\text{norm}}(k) \tilde{R}_{\text{d}, \text{norm}} \right) \\
\text{s.t. } \quad &\tilde{\zeta}_{\text{norm}}(k+1) = \tilde{A}_{\text{norm}}(k) \tilde{\zeta}_{\text{norm}}(k) + \tilde{B}_{\text{norm}}(k) \Delta u_{\text{norm}}(k) + \tilde{d}_{\text{norm}}(k) \\
&\tilde{\eta}_{\text{norm}}(k+1) = \tilde{C}(k) \tilde{\zeta}_{\text{norm}}(k+1) \\
&0 = \delta_f(k) + \delta_r(k), \quad k = 0, \dots, N-1 \\
&\Delta \tilde{u}_{\text{norm}} \leq u(k) \leq \Delta \tilde{u}_{\text{norm}}, \quad k = 0, \dots, N-1 \\
&\tilde{\zeta}_{\text{norm}} \leq \zeta_{\text{norm}}(k) \leq \tilde{\zeta}_{\text{norm}}, \quad k = 0, \dots, N \\
&\zeta(0) = \zeta(t), \quad u(0) = u(t)
\end{aligned} \tag{4.19}$$

The resulting optimization problem is an Optimal Control Quadratic Problem, which is then solved using an off-the-shelf solver. In this work, we have used the HPIPM solver [36] to solve the optimization problem described in equation 4.19.

The constraint $\delta_f(k) + \delta_r(k) = 0$ is imposed to ensure that the AMR follows the Ackermann steering geometry. equation 4.19 includes the preprocessing of the states and inputs, as discussed in section 4.3, and the denormalization of the solution using the inverse of the normalization matrices (equation 4.8).

In the following, we outline the proposed design of the cloud controller. We begin by discussing the methodology for handling specific cases, addressing networks with varying quality of service. We then propose an algorithm that generalizes across different conditions. In particular, the controller solves the MPC problem initialized on the estimate of the states.

4.5 Design of Cloud-Controller

As shown in figure 4.3, in the ideal case, the sensor processes data and sends updates at a constant frequency. Similarly, the Cloud-Controller ($g(\cdot)$) receives this information and computes the control commands at a fixed rate.

Due to the presence of downlink (τ_{dl}) and uplink latency (τ_{ul}), the cloud controller might not have real-time access to the AMR's state at the moment when the next optimal control will be applied (figure 3.1).

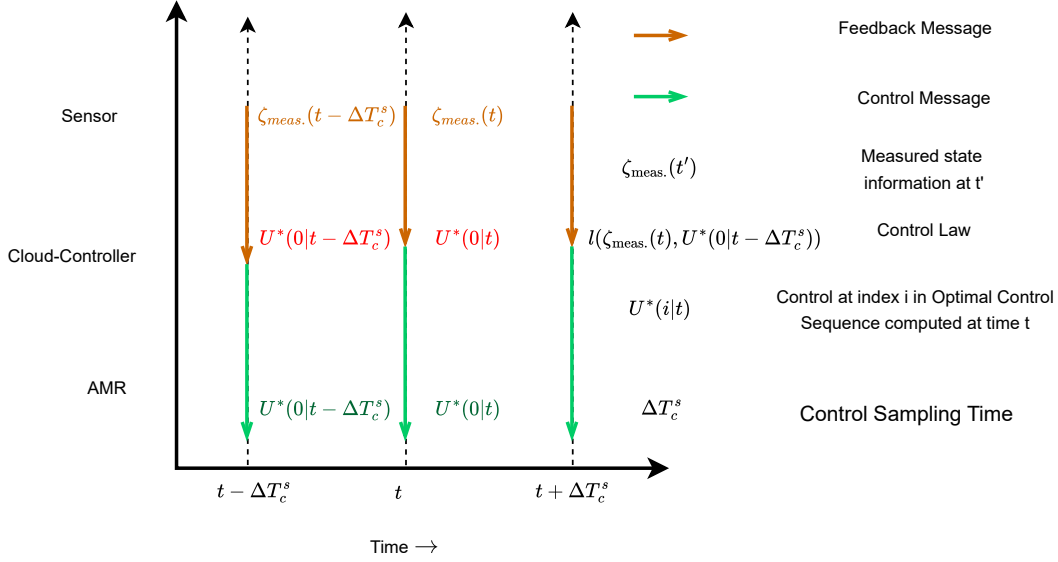


Figure 4.3: Timing Diagram in ideal Case (when there's no latency in system).

When only downlink latency τ_{dl} is present, the most recent state information available to the $\mathbf{g}(\cdot)$ at the current time t corresponds to the sensor's processed state from $t - \tau_{\text{dl}}$, denoted by $\zeta_{\text{meas.}}(t - \tau_{\text{dl}})$. During this interval $[t - \tau_{\text{dl}}, t]$, the AMR continues to evolve, resulting in a discrepancy between the true state and the available state information at the $\mathbf{g}(\cdot)$.

We also note that since there might be some uplink latency (τ_{ul}), the next optimal control input will be applied to the actuator at $t + \tau_{\text{ul}}$.

Since the true state $\zeta(t + \tau_{\text{ul}})$ is unavailable at the current time t , the cloud controller must first estimate the state $\hat{\zeta}(t + \tau_{\text{ul}})$ at $t + \tau_{\text{ul}}$ using the information available at time t , $\zeta_{\text{meas.}}(t - \tau_{\text{dl}})$.

To address this, we propose the following: First, the $\mathbf{g}(\cdot)$ stores the latest received state feedback along with its corresponding timestamp. Second, the $\mathbf{g}(\cdot)$ maintains a buffer of past computed control inputs along with their approximate application times (i.e., the approximate time at which the computed control will be executed). This enables it to predict how the vehicle has evolved despite latency.

The following section outlines the state estimation function, given access to the buffer of past computed control inputs.

4.5.1 State Estimation Function

Let's assume that the $\mathbf{g}(\cdot)$ has the information on what control input has been or will be executed in $[t - \tau_{dl}, t + \tau_{ul}]$. The control buffer at time t is denoted by $U_{\text{buffer}}(t)$, and defined as shown in equation 4.20.

$$U_{\text{buffer}}(t) := \begin{bmatrix} u(t - \tau_{dl}) & u(t - \tau_{dl} + \Delta T_c^s) & u(t - \tau_{dl} + 2\Delta T_c^s) & \cdots & u(t - \tau_{dl} + (\tau_{dl} - 1)\Delta T_c^s) \\ u(t) & u(t + \Delta T_c^s) & u(t + 2\Delta T_c^s) & \cdots & u(t + \tau_{ul}) \end{bmatrix} \quad (4.20)$$

At the current time, the cloud controller compares the current time t , to the timestamp of the latest available feedback message, where the difference between them is τ_{dl} . We define the state-estimation-function $h(\zeta_{\text{meas.}}, U_{\text{buffer}})$ as a recurrence relationship on the vehicle's non-linear kinematic model equation 4.1 as follows:

$$\begin{aligned} \hat{\zeta}(t - \tau_{dl} + \Delta T_c^s) &= \zeta_{\text{meas.}}(t - \tau_{dl}) + f(\zeta_{\text{meas.}}(t - \tau_{dl}), U_{\text{buffer}}(t)[0])\Delta T_c^s \\ \hat{\zeta}(t - \tau_{dl} + 2\Delta T_c^s) &= \hat{\zeta}(t - \tau_{dl} + \Delta T_c^s) + f(\hat{\zeta}(t - \tau_{dl} + \Delta T_c^s), U_{\text{buffer}}(t)[1])\Delta T_c^s \\ \hat{\zeta}(t - \tau_{dl} + 3\Delta T_c^s) &= \hat{\zeta}(t - \tau_{dl} + 2\Delta T_c^s) + f(\hat{\zeta}(t - \tau_{dl} + 2\Delta T_c^s), U_{\text{buffer}}(t)[2])\Delta T_c^s \\ &\vdots \\ \hat{\zeta}(t) &= \hat{\zeta}(t - \Delta T_c^s) + f(\hat{\zeta}(t - \Delta T_c^s), U_{\text{buffer}}(t)[N_{dl}])\Delta T_c^s \\ \hat{\zeta}(t + \Delta T_c^s) &= \hat{\zeta}(t) + f(\hat{\zeta}(t), U_{\text{buffer}}(t)[N_{dl} + 1])\Delta T_c^s \\ \hat{\zeta}(t + 2\Delta T_c^s) &= \hat{\zeta}(t + \Delta T_c^s) + f(\hat{\zeta}(t + \Delta T_c^s), U_{\text{buffer}}(t)[N_{dl} + 2])\Delta T_c^s \\ \hat{\zeta}(t + 3\Delta T_c^s) &= \hat{\zeta}(t + 2\Delta T_c^s) + f(\hat{\zeta}(t + 2\Delta T_c^s), U_{\text{buffer}}(t)[N_{dl} + 3])\Delta T_c^s \\ &\vdots \\ \hat{\zeta}(t + \tau_{ul}) &= \hat{\zeta}(t + \tau_{ul} - \Delta T_c^s) + f(\hat{\zeta}(t + \tau_{ul} - \Delta T_c^s), U_{\text{buffer}}(t)[N_{dl} + N_{ul} - 1])\Delta T_c^s \end{aligned} \quad (4.21)$$

ΔT_c^s is the controller sampling time. $N_{dl} := \lfloor \tau_{dl} / \Delta T_c^s \rfloor$, $N_{ul} := \lfloor \tau_{ul} / \Delta T_c^s \rfloor$. A more accurate integration method, such as Rk4, could be used instead of Euler integration. The next section outlines the design procedure for computing the control-buffer and presents the corresponding timing diagram for the network conditions mentioned in section 3.3.

4.6 Fire-and-Forget Communication

In this type of network configuration, the sender is allowed to send the next message without waiting to receive an acknowledgment from the recipient.

In this approach, the sensors and the cloud controller operate on a fixed schedule, meaning they perform computations at regular intervals (figure 4.4). We refer to this as a time-triggered system.

The following section discusses the computation of the control buffer in two sub-cases.

1. A constant end-to-end latency greater than ΔT_c^s .
2. An end-to-end latency larger than ΔT_c^s with occasional message loss.

4.6.1 Constant latency

In the figure 4.4, $\tau_{dl} = \Delta T_c^s$ and $\tau_{ul} = \Delta T_c^s$. $U^*(i|t)$ represents the control at i^{th} index, in an optimal control sequence $U^*(\cdot|t)$ computed at time t . As shown in figure 4.4, the $U_{buffer}(t)$ will be as follows:

$$U_{buffer}(t) := [U^*(0|t - 2\Delta T_c^s) \quad U^*(0|t - \Delta T_c^s)] \quad (4.22)$$

A general equation can be written for arbitrary known τ_{ul} and τ_{dl} equation 4.23.

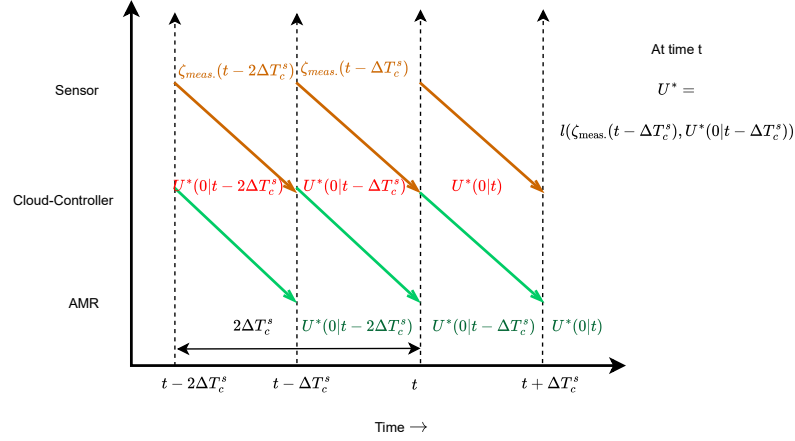
$$U_{buffer}(t) := \begin{bmatrix} U^*(0|t - (\tau_{dl} + \tau_{ul})) & U^*(0|t - (\tau_{dl} + \tau_{ul}) + \Delta T_c^s) & \dots \\ U^*(0|t - (\tau_{dl} + \tau_{ul}) + 2\Delta T_c^s) & \dots & \dots \\ U^*(0|t - (\tau_{dl} + \tau_{ul}) + ((\tau_{dl} + \tau_{ul}) - 3\Delta T_c^s)) & U^*(0|t - 2\Delta T_c^s) & U^*(0|t - \Delta T_c^s) \end{bmatrix} \quad (4.23)$$

The control algorithm is called at every constant control sampling time ΔT_c^s . First, it uses $U_{buffer}(t)$ to estimate the future state of the AMR and the control input that will be executed over the interval

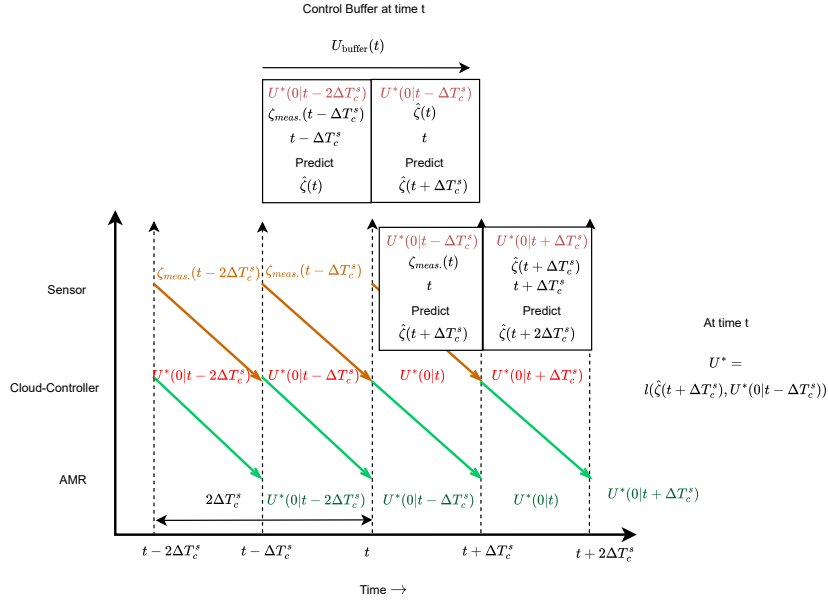
$$[t + \tau_{ul} - \Delta T_c^s, \quad t + \tau_{ul}] .$$

Second, it solves a finite-horizon optimal control problem to compute the optimal control sequence (equation 4.19) and then sends the first control input from this sequence (figure 4.4b). We refer to this method as *Baseline2*.

We compare the above methodology with the case where no future state or control estimation is performed, and the system works solely on the latest available feedback message. We refer to this method as *Baseline1* figure 4.4a.



(a) Baseline1 method: Control Algorithm is triggered at every fixed ΔT_c^s . Finite horizon Optimal Control problem equation 4.19 will be solved with the latest feedback message and the last computed control.



(b) Baseline2 method: A finite-horizon optimal control problem equation 4.19 is solved at every fixed ΔT_c^s using the estimated states and control inputs, and the first control from the computed optimal sequence is transmitted to the AMR.

Figure 4.4: Baseline1 and Baseline2 method's Timing Diagram in constant latency ($\tau_{\text{ul}}, \tau_{\text{dl}} = \Delta T_c^s$).

4.6.2 Lossy Network

Packet loss can occur due to large variations in latency or the inherent lossy nature of the communication network [33], where a message sent from the sender fails to reach the recipient, as shown in figure 4.5.

In the case of large latency variations, a control message computed at a later time may arrive before an earlier one, leading to out-of-order execution of control commands. In such cases, it is reasonable to discard the later message if it is known to have been generated before the previously received control message, since the earlier message would already reflect the latest available information (figure 4.6).

In the following section, and while reporting the results, we use a single metric for message dropout percentage. This metric combines the percentage of message loss caused by both the inherent lossiness of the network and large variations in network latency.

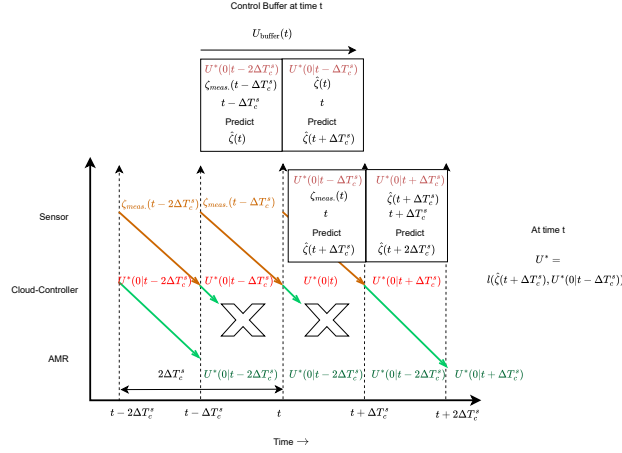
In this scenario, the controller uses the state estimation procedure under the assumption that the last control packets transmitted by the controller were or will be successfully executed by the system (figure 4.5).

We propose two changes. First, we propose adding a unique ID to each message sent from the cloud to the AMR, allowing the system to identify and discard out-of-sequence messages. This technique has also been applied to discard out-of-sequence messages from sensor-to-cloud transmissions and vehicle feedback messages to the cloud.

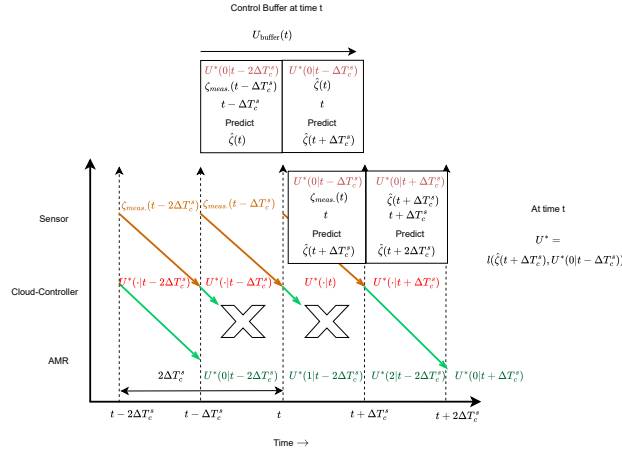
Second, instead of sending only the first control input from the optimal sequence computed by the MPC, we propose transmitting the entire control sequence (figure 4.5b). We refer to this approach as the *Proposed Method*.

This ensures that the AMR has a reference to follow in case of message dropout, as shown in figure 4.5b. In contrast, since the *Baseline2* method sends only the first control input from the sequence, the AMR would execute a constant control reference until the next control message arrives.

Since the controller is unaware of such message loss, the $U_{\text{buffer}}(t)$ will remain unchanged, as in the case of constant latency.



(a) Baseline2 method: A finite-horizon optimal control problem equation 4.19 is solved at every fixed ΔT_c^s using the estimated states and control inputs, and the first control from the computed optimal sequence is transmitted to the AMR.



(b) Proposed method: At every fixed ΔT_c^s , a finite-horizon optimal control problem equation 4.19 is solved using the estimated states and control inputs, after which the entire computed optimal sequence is transmitted to the AMR.

Figure 4.5: Baseline2 and Proposed method's Timing Diagram in lossy network ($\tau_{ul}, \tau_{dl} = \Delta T_c^s$).

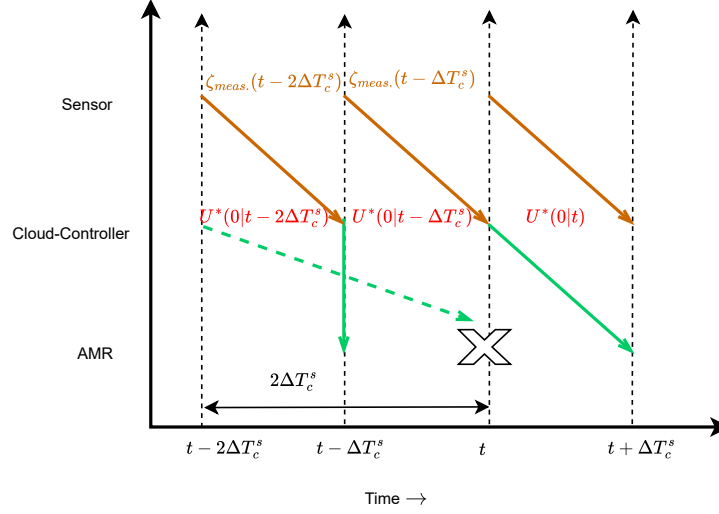


Figure 4.6: Example of a message loss due large latency variation.

4.7 Acknowledgment-Driven Communication

Security remains a critical challenge in CCAS, requiring robust protective measures, as outlined in [10]. An encryption-based acknowledgment protocol is essential to ensure data integrity and system reliability. In this approach, the controller waits for confirmation that the previous control input has been successfully executed before computing and transmitting the next command. This acknowledgment is facilitated through encrypted verification mechanisms, ensuring that only authenticated messages are exchanged between the cloud and the system [10]. In acknowledgment-driven communication, the controller operates in an event-triggered manner, meaning it computes and sends control commands only after receiving the necessary acknowledgment. figure 4.7 illustrates this case.

For brevity, this work assumes fire-and-forget-based sensor-to-cloud communication while focusing on a secure network connection from the cloud to the AMR. However, extending a secure connection from the sensor to the cloud is a straightforward adaptation.

In this case, the MPC algorithm will be called at every $\tau_{ul} + \tau_{dl}$ instead of ΔT_C^s . For the example illustrated in figure 4.7b the $U_{\text{buffer}}(t)$ is

$$U_{\text{buffer}}(t) := [U^*(0|t - 2\Delta T_c^s) \quad U^*(1|t - 2\Delta T_c^s)] \quad (4.24)$$

A generic expression for computing $U_{\text{buffer}}(t)$ is not straightforward in this case. Hence, during the software design, we use the timestamps of the sensor messages and the expected application times of the previously computed control messages to manage the buffer. This will be discussed next.

Remark 1. For simplicity, during simulation and experiments, we assume that the acknowledgment time is zero, meaning the MPC control algorithm is executed at intervals of τ_{ul} instead of $\tau_{\text{ul}} + \tau_{\text{dl}}$.

4.8 Algorithm for Computing Control Sequence

To this end, we propose an algorithm for computing the control sequence that unifies all the above cases into a single, coherent framework (algorithm 2).

We introduce specific notations to streamline the algorithm presentation:

- $\zeta_{\text{meas.}}(t') :=$ Latest sensor message available, containing state information for time t' .
- $U_{\text{buffer}}(t) :=$ A control buffer at time t storing $\{u_i, t_i\}$ tuples.
- $\{u_i, t_i\} :=$ Control input applied at the approximate time t_i , where t_i represents the estimated time at which the low-level controller executes this command.
- $t :=$ Current time.

At time t , algorithm 2 first obtains the most recent measured state, which arrives with a latency τ_{dl} . It then updates the list of planned control actions by discarding any actions that occurred before $t - \tau_{\text{dl}}$, as they are no longer relevant. Next, the algorithm uses both the measured state and the control actions from the buffer to predict the AMR state at $t + \tau_{\text{ul}}$, denoted as $\hat{\zeta}(t + \tau_{\text{ul}})$ using the recursive state-estimation function (equation 4.21).

With the predicted state and the anticipated control action that the AMR will execute between $t + \tau_{\text{ul}} - \Delta T_c^s$ and $t + \tau_{\text{ul}}$, denoted by $\hat{u}(t + \tau_{\text{ul}})$, it solves a finite-horizon optimization problem (equation 4.19) to determine the optimal control sequence from $t + \tau_{\text{ul}}$ onward. Finally, it replaces any outdated control sequence after $t + \tau_{\text{ul}}$ with the newly computed sequence. The algorithm outputs the new control actions (U^*) and waits for the next sampling instant (ΔT^s) before repeating the process.

The sampling time ΔT^s is equal to ΔT_c^s in the fire-and-forget based communication. For a secure acknowledgment-based connection, ΔT^s will be equal to τ_{ul} . see remark 1.

Algorithm 2 Control Sequence Computation

```
1: while true do                                     ▷ Execute at each sampling time  $\Delta T^s$ 
2:   Require  $\zeta_{\text{meas}}(t - \tau_{\text{dl}}), U_{\text{buffer}}(t)$                                      ▷ Current time is  $t$ 
3:   discard( $U_{\text{buffer}}(t)[0 : t_k]$ ), where  $t_k \leq t - \tau_{\text{dl}}$ 
4:    $U_{\text{seq}} \leftarrow U_{\text{buffer}}(t)[0 : t + \tau_{\text{ul}}]$ 
5:    $\hat{\zeta}(t + \tau_{\text{ul}}) \leftarrow h(\zeta_{\text{meas}}(t - \tau_{\text{dl}}), U_{\text{seq}}(t))$        ▷ Predict  $\hat{\zeta}(t + \tau_{\text{ul}})$  using  $h(\zeta_{\text{meas.}}, U_{\text{buffer}})$ 
6:    $\hat{u}(t + \tau_{\text{ul}}) \leftarrow U_{\text{seq}}[-1]$ 
7:    $U^* \leftarrow l(\hat{\zeta}(t + \tau_{\text{ul}}), \hat{u}(t + \tau_{\text{ul}}))$                                      ▷ Solve Eq.4.19 for  $U^*$ 
8:   if problem infeasible then
9:     Stop
10:  end if
11:  discard( $U_{\text{buffer}}(t)[t + \tau_{\text{ul}} : -1]$ )
12:   $U_{\text{buffer}}(t) \leftarrow \text{append}(U_{\text{buffer}}(t), U^*)$ 
13:  Return  $U^*$ 
14:  Wait for the next sampling time  $\Delta T^s$ 
15: end while
```

4.9 Control Sequence Handling in the AMR

In the work outlined in [6], [11], the AMR receives a single control input from the network at each time step. However, as discussed in the methodology, the cloud controller system now transmits an entire control sequence instead of a single command.

To accommodate this, the algorithm at the AMR has been modified to process and execute a sequence of control inputs, ensuring that the appropriate control command is applied at each step. This change allows the AMR to continue executing the planned sequence of control actions even in the presence of network latencies or intermittent communication losses.

Chapter 5

Simulations and Experiments Results

This chapter presents the simulation results conducted in [ROS](#), followed by the results from experiments conducted in [CCAS](#) controlling the actual [AMR](#).

To evaluate the performance of the methods, we use two metrics: (a) Total [MPC](#) cost and (b) Tracking-Error cost.

Total reference path length (total ref. path length) represents the total Euclidean distance of the reference path.

Total [MPC](#) Cost: The total [MPC](#) cost is defined as the sum of the following three individual costs:

$$\text{Total [MPC](#) Cost} = \text{State Reference Cost} + \text{Control Cost} + \text{Smoothness Cost}$$

- State Reference Cost (State Ref. Cost):

$$\frac{\sum_k (\zeta(k) - \zeta_{\text{ref}}(k))^T Q (\zeta(k) - \zeta_{\text{ref}}(k))}{\text{total reference path length}}$$

This cost denotes the normalized weighted cost of the difference between the reference state and the actual state of the [AMR](#) at time step k .

- Control Cost:

$$\frac{\sum_k u(k)^T R u(k)}{\text{total reference path length}}$$

This cost denotes the normalized weighted cost of the magnitude of the control effort.

- Smoothness Cost:

$$\frac{\sum_k (u(k-1) - u(k))^T R_d (u(k-1) - u(k))}{\text{total reference path length}}$$

This cost denotes the normalized weighted cost of the change in control between two time steps.

Tracking-Error Costs: In autonomous driving literature, tracking-error costs are often used to evaluate the performance of planning and control methods. In this work, we include the following three components as part of the tracking-error costs:

- Speed Error Tracking Cost (Speed Error Track. Cost): This is defined as the normalized L2 norm of the error between the actual COG speed ($v_c(k)$) at time step k from the reference speed ($v_{c,\text{ref}}(k)$) at time step k . The reference speed was constant in simulations and during the experiments. The units are in meters per second:

$$\frac{\sqrt{\sum_k (v_{c,\text{ref}}(k) - v_c(k))^2}}{\text{total reference path length}}$$

- Yaw Error Tracking Cost (Yaw Error Track. Cost): This is defined as the normalized L2 norm of the yaw angle deviation from the closest point on the reference path at time step k [37]. In figure 5.1, e_{yaw} represents the yaw error at some time instance. The units are in radians per meter:

$$\frac{\sqrt{\sum_k (\theta_{\text{ref}}(k) - \theta(k))^2}}{\text{total reference path length}}$$

- Lateral Error Tracking Cost (Lateral Error Track. Cost): This is defined as the normalized L2 norm of the lateral error to the closest point on the reference path at time step k [35], [37]. This calculation is performed in the path-aligned local coordinate frame, where the lateral error is measured perpendicular to the tangent defined at the closest point on the reference path. In figure 5.1, e_{lateral} represents the lateral error at some time instance. This is reported as a pure number with no unit:

$$\frac{\sqrt{\sum_k ((\sin \theta_{\text{ref}}(k))(x(k) - x_{\text{ref}}(k)) - (\cos \theta_{\text{ref}}(k))(y(k) - y_{\text{ref}}(k)))^2}}{\text{total reference path length}}$$

Remark 2. The latencies reported in the simulation and experimental results refer to the artificially added uplink and downlink latencies. In addition to these, extra latency may be introduced by sensor processing and cloud computation.

Remark 3. Due to space constraints, the units have been omitted while reporting the results.

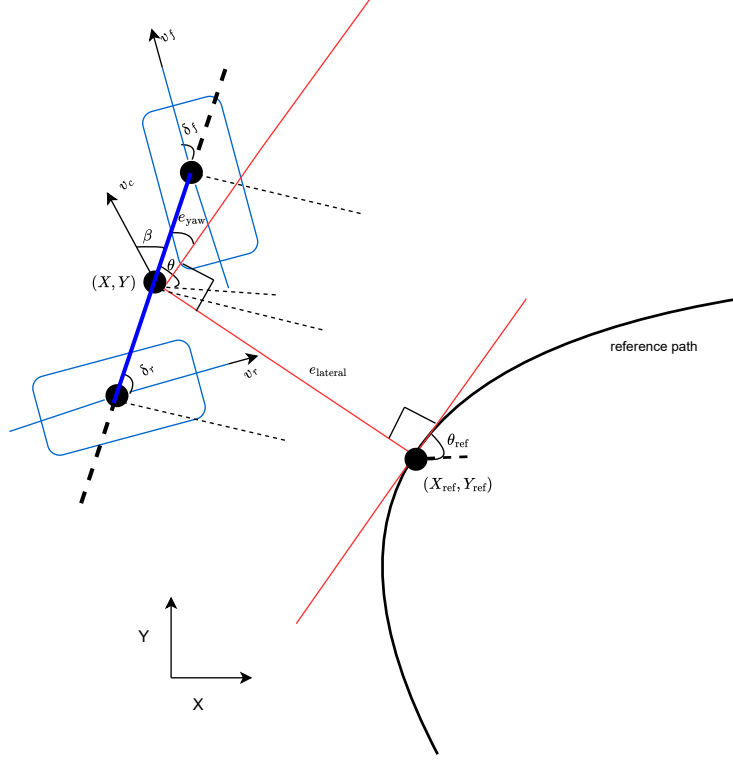


Figure 5.1: Lateral and yaw error with reference to the closest point on the reference track.

5.1 Simulation Setup

A simulation stack¹ based on the Robot Operating System (ROS) has been developed to model a CCAS, with artificial uplink and downlink latency added to reflect real-world network delays. The goal of the ego vehicle is to complete the reference track while maintaining a target reference speed. The key components of the software stack are illustrated in figure 5.2 as a graph. Rectangular boxes indicate the type of signal being exchanged between nodes (oval shaped), while arrows show the direction of message flow. The main components of the simulation stack include:

1. */trajectory_planning_node*: This node is responsible for generating the trajectory plan based on perception data received from the */network_sim_node*.

¹ Simulation code can be found at <https://github.com/prajwalthakur/CloudControlAvSim>

2. */hospital_bed_node*: This node represents the ego vehicle in the simulation. It receives control signals from the */control_interface_node* and executes them while also sending back state feedback (*/ego_state_feedback*) at a fixed frequency.
3. */network_sim_node*: This node simulates artificial network conditions by introducing communication delays for both control commands and perception feedback. It receives control commands from the */trajectory_planning_node* and forwards them to the */control_interface_node*, where the control messages are stored. The stored commands are then transmitted to the */hospital_bed_node* at a fixed frequency to drive the vehicle dynamics. Similarly, it receives feedback from the */hospital_bed_node* through */perception_to_network* and transmits it to the */trajectory_planning_node* after accounting for the effects of the simulated network conditions.

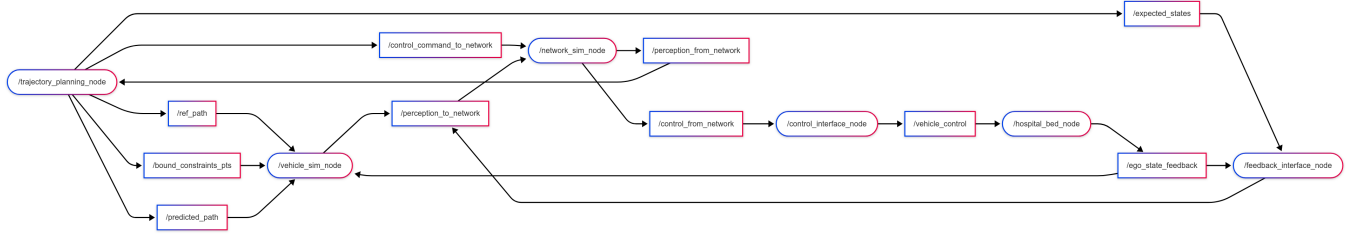


Figure 5.2: Rqt graph for the simulation stack.

5.2 Simulation Results

The table 5.1 shows the AMR and MPC parameters considered during simulations. N is the

Table 5.1: AMR and MPC tuning parameters.

Symbol	Value	Symbol	Value
l_f, l_r	1.2 m	N	12
u_{max}	$[2.5, 2.5, \frac{\pi}{4}, \frac{\pi}{4}]$	R_d	$1e^{-3} \cdot \text{diag}(0.1, 0.1, 1.0, 1.0)$
u_{min}	$-[2.5, 2.5, \frac{\pi}{4}, \frac{\pi}{4}]$	Q	$0.2 \cdot \text{diag}(15, 15, 2, 20, 20)$
Δu_{max}	$[6.0, 6.0, \frac{\pi}{10}, \frac{\pi}{10}]$	R	$\text{diag}(0.5, 0.5, 2, 2)$
Δu_{min}	$-[6.0, 6.0, \frac{\pi}{10}, \frac{\pi}{10}]$	$V_{f,ref}$	4.0 m/sec
ΔT_c^s	0.05 sec	$V_{r,ref}$	4.0 m/sec

control horizon, and $V_{f,ref}$ and $V_{r,ref}$ are the reference speeds of the front and rear wheels,

respectively. The relative magnitude of R_d has been kept significantly lower compared to Q and R to prioritize tracking the reference trajectory over the smoothness of the control actions.

5.2.1 Fire-and-Forget Communication with Constant Mean Latency and Standard Deviation

In this scenario, the cloud controller assumes that the latency in the system is constant (equal to the mean of the Gaussian distribution) and operates as described in section 4.6.1. The table 5.2 shows the Comparison b/w the *Baseline1* vs *Baseline2* vs *Proposed* Method across different Constant mean latencies. The results indicate that, in the case of constant

Table 5.2: Tracking error and MPC cost breakdown across methods and latency (average over 12 simulation runs) in fire-and-forget-based communication.

(a) Tracking Error Cost Breakdown

Latency	Method	Lateral Error Track. Cost	Yaw Error Track. Cost	Speed Error Track. Cost
$\mu = 0.05, \sigma = 0.02$	Proposed	0.037 ± 0.001	0.011 ± 0.000	0.123 ± 0.004
	Baseline2	0.040 ± 0.002	0.011 ± 0.001	0.129 ± 0.017
	Baseline1	0.066 ± 0.007	0.036 ± 0.007	0.138 ± 0.005
$\mu = 0.1, \sigma = 0.02$	Proposed	0.046 ± 0.006	0.015 ± 0.003	0.131 ± 0.007
	Baseline2	0.062 ± 0.023	0.028 ± 0.018	0.141 ± 0.015
	Baseline1	0.162 ± 0.010	0.128 ± 0.010	0.230 ± 0.007

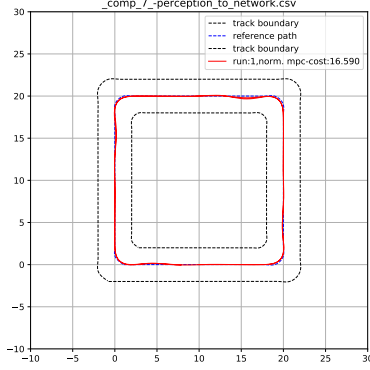
(b) MPC Cost Breakdown (scaled to 10^2)

Latency	Method	State Ref. Cost	Control Cost	Smoothness Cost	Total MPC Cost
$\mu = 0.05, \sigma = 0.02$	Proposed	0.150 ± 0.010	0.020 ± 0.000	0.000 ± 0.000	0.180 ± 0.010
	Baseline2	0.170 ± 0.060	0.030 ± 0.000	0.000 ± 0.000	0.190 ± 0.060
	Baseline1	0.170 ± 0.020	0.050 ± 0.010	0.000 ± 0.000	0.210 ± 0.020
$\mu = 0.1, \sigma = 0.02$	Proposed	0.180 ± 0.020	0.030 ± 0.000	0.000 ± 0.000	0.200 ± 0.020
	Baseline2	0.200 ± 0.040	0.030 ± 0.010	0.000 ± 0.000	0.230 ± 0.050
	Baseline1	0.330 ± 0.030	0.170 ± 0.020	0.000 ± 0.000	0.500 ± 0.050

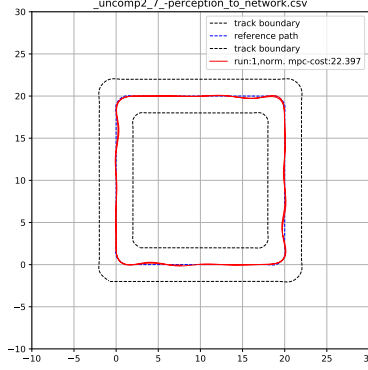
The costs are normalized by the total reference path length. Two cases were considered: (1) uplink and downlink latencies following a Gaussian distribution with a mean of 0.05 seconds and a standard deviation of 0.02 seconds, and (2) uplink and downlink latencies following a Gaussian distribution with a mean of 0.1 seconds and a standard deviation of 0.02 seconds.

The latencies are truncated at ± 3 times the standard deviation.

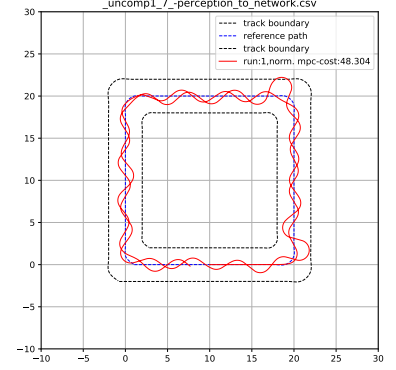
mean latency with minimal standard deviation, *Baseline1* Method performs the worst. *Baseline2* and the *Proposed* Method exhibit similar performance due to the low variance in latency. An example for the tracking performance is shown in the figure 5.3



(a) Tracking performance of proposed method.



(b) Tracking performance of baseline2 method.



(c) Tracking performance of baseline1 method.

Figure 5.3: Example of tracking performance at a constant mean latency of 0.1 seconds and a standard deviation of 0.02 seconds.

In the next sections, we will omit the results of the *Baseline1* method and focus only on the *Proposed* versus *Baseline2* method performance under different network conditions.

5.2.2 Fire-and-Forget Communication with with Constant Latency and Message Dropouts

In the following results, dropout follows a Bernoulli distribution with success probabilities determined by the specified dropout rate. For instance, a 10% dropout rate implies that each transmitted message has a 10% probability of failure. In this case, the cloud-control operates as described in section 4.6.2.

Uplink latency = 0.15 sec , Downlink latency =0.15 sec

Table 5.3: Tracking error and MPC cost breakdown across methods and latency (average over 12 simulation runs) in fire-and-forget-based communication.

(a) Tracking Error Cost Breakdown

Dropout %	Method	Lateral Error Track. Cost	Yaw Error Track. Cost	Speed Error Track. Cost
10	Proposed	0.024 ± 0.003	0.020 ± 0.003	0.130 ± 0.005
	Baseline2	0.035 ± 0.008	0.029 ± 0.007	0.131 ± 0.005
35	Proposed	0.028 ± 0.006	0.021 ± 0.003	0.131 ± 0.006
	Baseline2	0.055 ± 0.043	0.038 ± 0.014	0.135 ± 0.008
60	Proposed	0.042 ± 0.026	0.025 ± 0.010	0.133 ± 0.006
	Baseline2	0.131 ± 0.034	0.092 ± 0.025	0.164 ± 0.018

(b) MPC Cost Breakdown (scaled to 10^2)

Dropout %	Method	State Ref. Cost	Control Cost	Smoothness Cost	Total MPC Cost
10	Proposed	0.180 ± 0.020	0.030 ± 0.000	0.000 ± 0.000	0.210 ± 0.020
	Baseline2	0.180 ± 0.020	0.030 ± 0.000	0.000 ± 0.000	0.210 ± 0.020
35	Proposed	0.190 ± 0.020	0.030 ± 0.000	0.000 ± 0.000	0.210 ± 0.020
	Baseline2	0.210 ± 0.060	0.030 ± 0.010	0.000 ± 0.000	0.240 ± 0.070
60	Proposed	0.200 ± 0.030	0.030 ± 0.000	0.000 ± 0.000	0.230 ± 0.030
	Baseline2	0.290 ± 0.060	0.060 ± 0.010	0.000 ± 0.000	0.350 ± 0.060

The costs are normalized by the total reference path length. The dropout rate follows a Bernoulli distribution with varying success probabilities. For example, a 10% dropout rate implies that each transmitted message has a 10% probability of failure.

The results in table 5.3 and table 5.4 show that, on average, the proposed method performs better than *Baseline2*. This is because, in the *Proposed* method, a sequence of control references is sent, allowing the AMR to continue tracking the control reference even if a control message is missed. In contrast, in *Baseline2*, the AMR tracks a constant control reference until it receives the next updated control from the cloud. Increasing the dropout percentage reduces the performance of both methods, but the proposed method still outperforms *Baseline2*. figure 5.5 and figure 5.4 show the box plots of the MPC cost and tracking error cost, confirming the same trend.

Uplink latency =0.25 sec , Downlink latency =0.25 sec

Table 5.4: Tracking error and MPC cost breakdown across methods and latency (average over 12 simulation runs) in fire-and-forget-based communication.

(a) Tracking Error Cost Breakdown

Dropout %	Method	Lateral Error Track. Cost	Yaw Error Track. Cost	Speed Error Track. Cost
10	Proposed	0.041 ± 0.009	0.028 ± 0.005	0.141 ± 0.009
	Baseline2	0.049 ± 0.010	0.034 ± 0.006	0.142 ± 0.006
35	Proposed	0.045 ± 0.014	0.028 ± 0.007	0.141 ± 0.007
	Baseline2	0.086 ± 0.021	0.057 ± 0.014	0.153 ± 0.009
60	Proposed	0.078 ± 0.024	0.038 ± 0.013	0.145 ± 0.007
	Baseline2	0.192 ± 0.042	0.120 ± 0.025	0.184 ± 0.012

(b) MPC Cost Breakdown (scaled to 10^2)

Dropout %	Method	State Ref. Cost	Control Cost	Smoothness Cost	Total MPC Cost
10	Proposed	0.210 ± 0.020	0.030 ± 0.000	0.000 ± 0.000	0.240 ± 0.030
	Baseline2	0.210 ± 0.010	0.030 ± 0.010	0.000 ± 0.000	0.250 ± 0.010
35	Proposed	0.220 ± 0.020	0.030 ± 0.010	0.000 ± 0.000	0.250 ± 0.020
	Baseline2	0.250 ± 0.030	0.050 ± 0.010	0.000 ± 0.000	0.290 ± 0.030
60	Proposed	0.250 ± 0.030	0.030 ± 0.000	0.000 ± 0.000	0.280 ± 0.030
	Baseline2	0.420 ± 0.100	0.080 ± 0.010	0.000 ± 0.000	0.500 ± 0.110

The costs are normalized by the total reference path length. Uplink and downlink latencies were constant at 0.25 seconds. The dropout rate follows a Bernoulli distribution with varying success probabilities. For example, a dropout rate of 10% implies that each transmitted message has a 10% probability of being lost.

Furthermore, comparing both the tables and figure 5.6 illustrates that, under the same dropout percentage, an increase in constant latency degrades the performance of both *Baseline2* and the *Proposed*. This is because the cloud controller is unaware of the message dropout, and higher latency amplifies the impact of message dropout. Greater latency increases the divergence between the predicted state and the actual state of the vehicle under the same dropout rate, leading to higher costs.

Examples of the tracking performance are shown in figure 5.7 and figure 5.8, which illustrate the tracking performance under 0.15 and 0.25 seconds of constant latency and 60% message dropout.

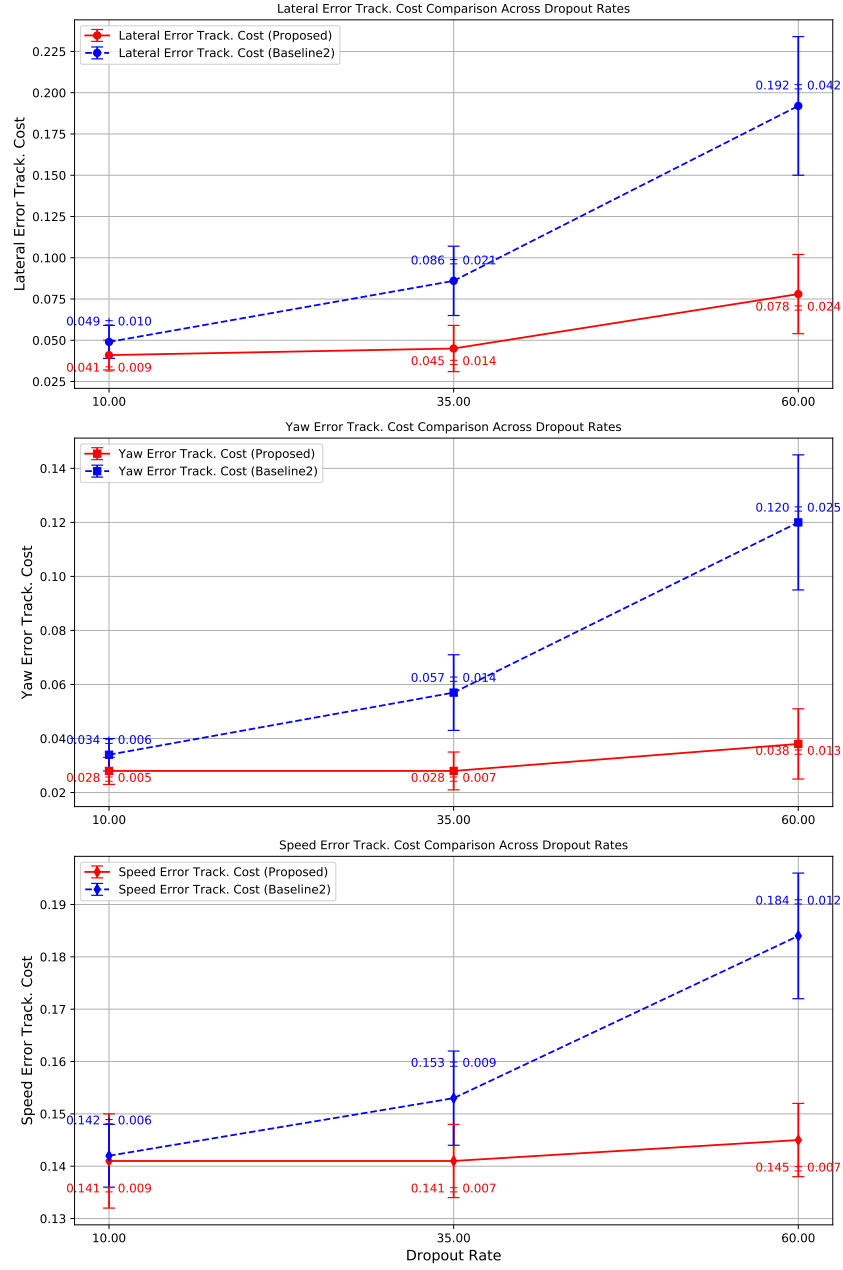


Figure 5.4: Performance, compared to the tracking cost, degrades as the dropout percentage increases in fire-and-forget communication.

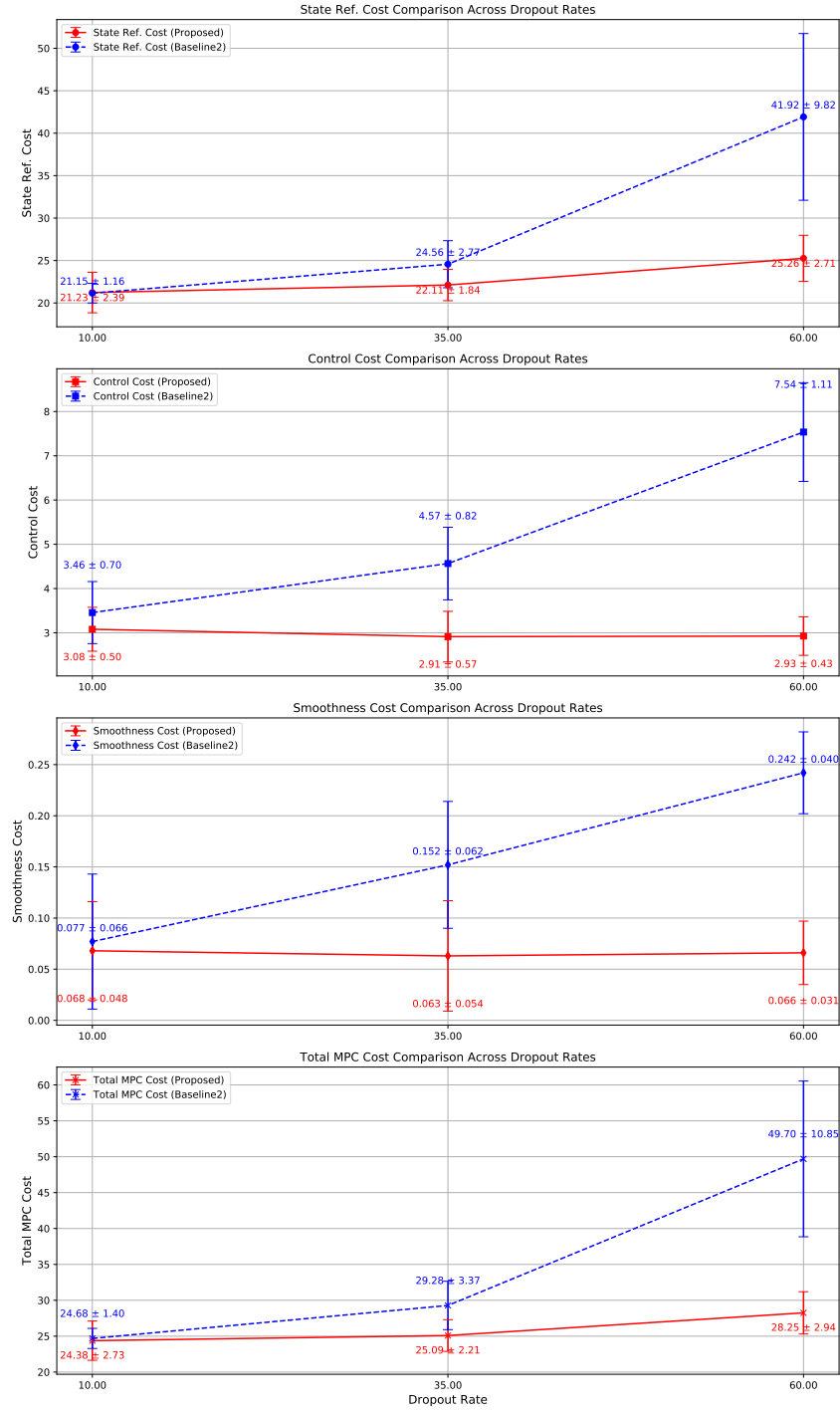


Figure 5.5: Performance, compared to the MPC cost, degrades as the dropout percentage increases in fire-and-forget communication.

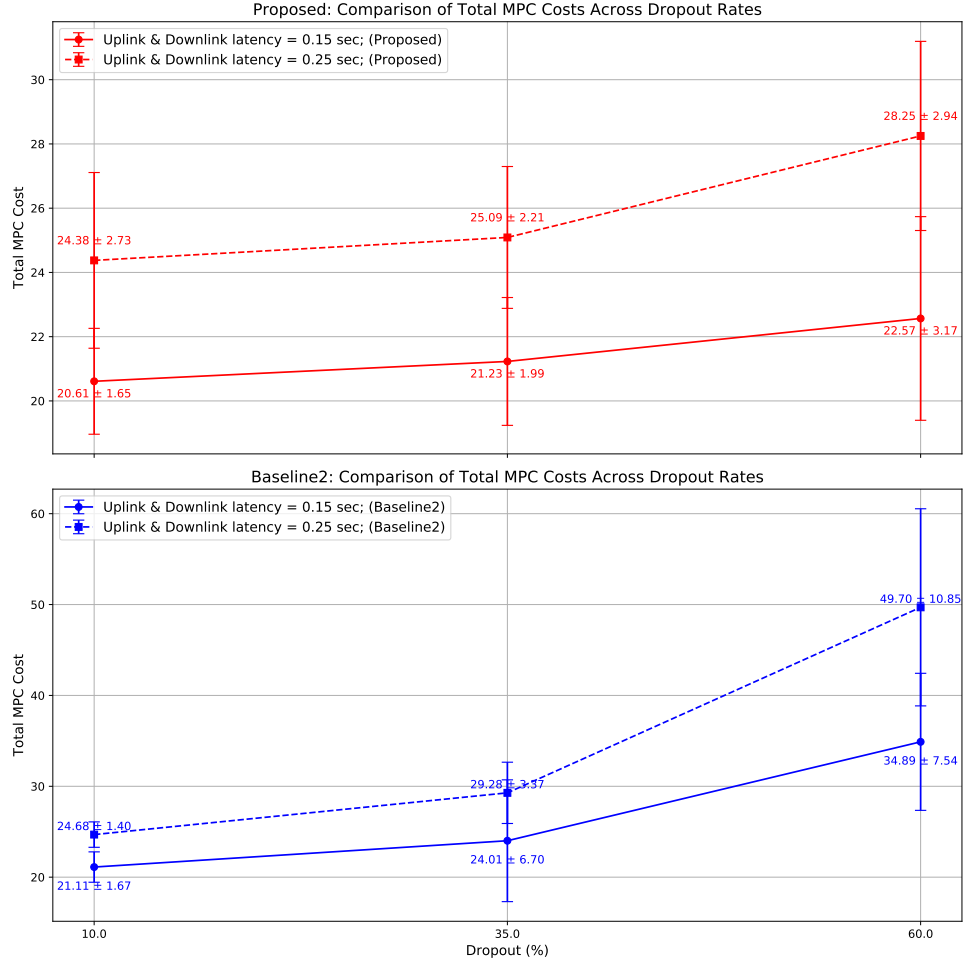
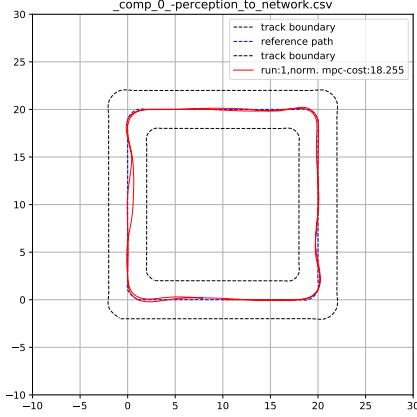
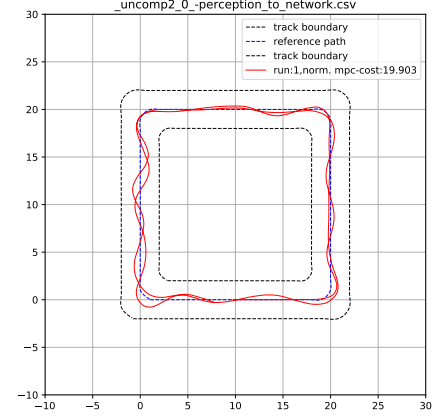


Figure 5.6: Comparison of performance against the MPC cost under different latencies and the same dropout percentage in fire-and-forget-based communication.

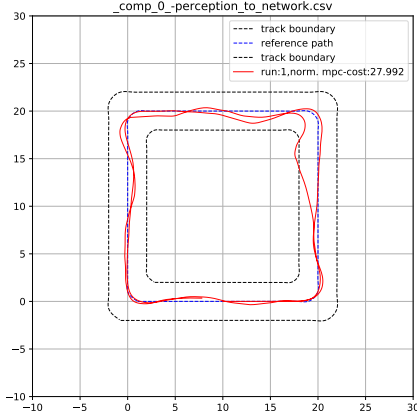


(a) Tracking performance of proposed method.

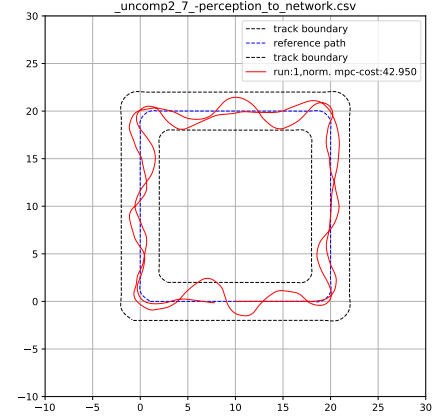


(b) Tracking performance of baseline2 method.

Figure 5.7: Example of tracking performance at a constant latency of (uplink: 0.15 sec, downlink: 0.15 sec) and a 60% dropout in fire-and-forget communication.



(a) Tracking performance of proposed method.



(b) Tracking performance of baseline2 method.

Figure 5.8: Example of tracking performance at a constant latency of (uplink: 0.25 sec, downlink: 0.25 sec) and a 60% dropout in fire-and-forget communication.

Fire-and-Forget Communication with High Variance

As discussed in section 4.6.2, packet loss can also occur due to significant deviations in latency from the mean value. table 5.5 presents the performance of the *Proposed* and *Baseline2* methods under increasing variance, with the mean latency consistently fixed at 0.1 seconds. Although the performance of both methods deteriorates as variance increases, the *Proposed* method consistently outperforms the *Baseline2* method.

Table 5.5: Tracking error and MPC cost breakdown across methods for fixed mean latency (0.1s) with increasing variance (average over 12 simulation runs).

(a) Tracking Error Cost Breakdown

Std Dev	Method	Lateral Error Track. Cost	Yaw Error Track. Cost	Speed Error Track. Cost
0.02	Comp.	0.017 ± 0.005	0.015 ± 0.003	0.131 ± 0.007
	Uncomp.2	0.033 ± 0.022	0.028 ± 0.018	0.141 ± 0.015
0.1	Comp.	0.021 ± 0.004	0.017 ± 0.003	0.220 ± 0.042
	Uncomp.2	0.038 ± 0.011	0.036 ± 0.009	0.249 ± 0.011

(b) MPC Cost Breakdown (Scaled to 10^2)

Std Dev	Method	State Ref. Cost	Control Cost	Smoothness Cost	Total MPC Cost
0.02	Comp.	0.18 ± 0.02	0.03 ± 0.00	0.00 ± 0.00	0.20 ± 0.02
	Uncomp.2	0.20 ± 0.04	0.03 ± 0.01	0.00 ± 0.00	0.23 ± 0.05
0.1	Comp.	0.59 ± 0.20	0.02 ± 0.00	0.00 ± 0.00	0.61 ± 0.20
	Uncomp.2	0.71 ± 0.06	0.04 ± 0.01	0.00 ± 0.00	0.75 ± 0.07

The costs are normalized by the total reference path length.

5.2.3 Acknowledgment-Driven Communication

In this scenario, the cloud controller assumes that the latency in the system is constant (equal to the mean of the Gaussian distribution) and operates as described in section 4.7. In this case, only the communication from the cloud to the AMR is acknowledgment-driven, while the sensor-to-cloud communication remains fire-and-forget based.

The table 5.6 compares the performance of the *Proposed* and *Baseline2* methods. On average, the *Proposed* method performs better across all performance metrics. The *Proposed* method outperforms *Baseline2* because, in the presence of large latency, the AMR has access to the full control sequence to track. In contrast, in *Baseline2*, only the first optimal control from the control sequence is sent, causing the AMR to track a constant control reference until the next control update is received from the cloud.

Table 5.6: Tracking error and **MPC** cost breakdown across methods and latency (average over 12 simulation runs) in acknowledgment-based communication.

(a) Tracking Error Cost Breakdown

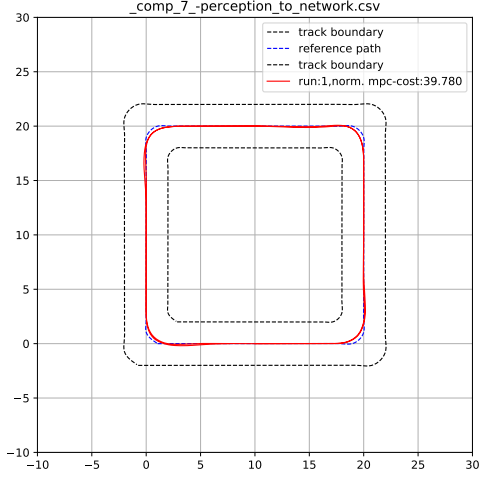
Latency	Method	Lateral Error Track. Cost	Yaw Error Track. Cost	Speed Error Track. Cost
$\mu = 0.05, \sigma = 0.03$	Proposed	0.022 ± 0.001	0.022 ± 0.001	0.140 ± 0.021
	Baseline2	0.024 ± 0.002	0.023 ± 0.002	0.141 ± 0.020
$\mu = 0.15, \sigma = 0.03$	Proposed	0.029 ± 0.002	0.027 ± 0.002	0.151 ± 0.008
	Baseline2	0.064 ± 0.009	0.044 ± 0.006	0.176 ± 0.004
$\mu = 0.25, \sigma = 0.03$	Proposed	0.061 ± 0.018	0.043 ± 0.006	0.179 ± 0.010
	Baseline2	0.115 ± 0.017	0.075 ± 0.009	0.222 ± 0.006

(b) **MPC** Cost Breakdown (scaled to 10^2)

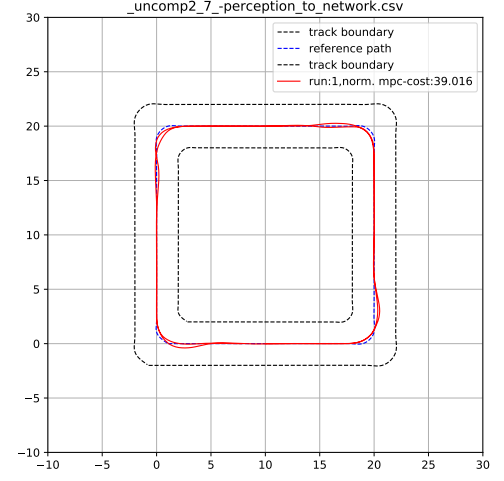
Latency	Method	State Ref. Cost	Control Cost	Smoothness Cost	Total MPC Cost
$\mu = 0.05, \sigma = 0.03$	Proposed	0.230 ± 0.080	0.030 ± 0.000	0.000 ± 0.000	0.260 ± 0.080
	Baseline2	0.230 ± 0.080	0.030 ± 0.000	0.000 ± 0.000	0.260 ± 0.080
$\mu = 0.15, \sigma = 0.03$	Proposed	0.260 ± 0.030	0.030 ± 0.000	0.000 ± 0.000	0.290 ± 0.030
	Baseline2	0.280 ± 0.020	0.070 ± 0.000	0.010 ± 0.000	0.350 ± 0.020
$\mu = 0.25, \sigma = 0.03$	Proposed	0.320 ± 0.040	0.050 ± 0.000	0.000 ± 0.000	0.370 ± 0.040
	Baseline2	0.400 ± 0.040	0.120 ± 0.010	0.010 ± 0.000	0.530 ± 0.040

The costs are normalized by the total reference path length. Three different cases were considered, where the uplink and downlink latencies followed a truncated Gaussian distribution with means of 0.05, 0.15, and 0.25 seconds, respectively, and a standard deviation of 0.03 seconds. The latencies were truncated at ± 3 times the standard deviation.

An increase in latency degrades the performance of both methods due to the increased error in state prediction through forward simulation of the dynamics. Examples of tracking performance under different methods and conditions are shown in figure 5.9 and figure 5.10. The figure 5.11 and figure 5.12 show the box plots of the tracking error and **MPC** costs.

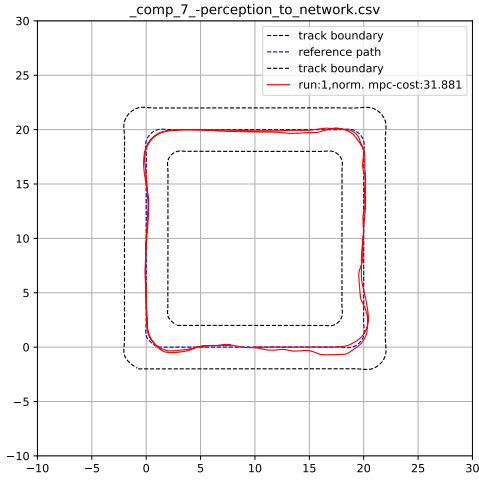


(a) Tracking performance of proposed method.

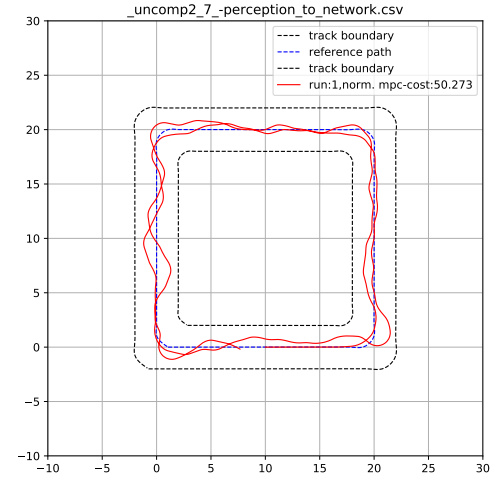


(b) Tracking performance of baseline2 method.

Figure 5.9: Example of tracking performance at an average latency of 0.05 seconds in acknowledgment-based communication.



(a) Tracking performance of proposed method.



(b) Tracking performance of baseline2 method.

Figure 5.10: Example of tracking performance at an average latency of 0.25 seconds in acknowledgment-based communication.

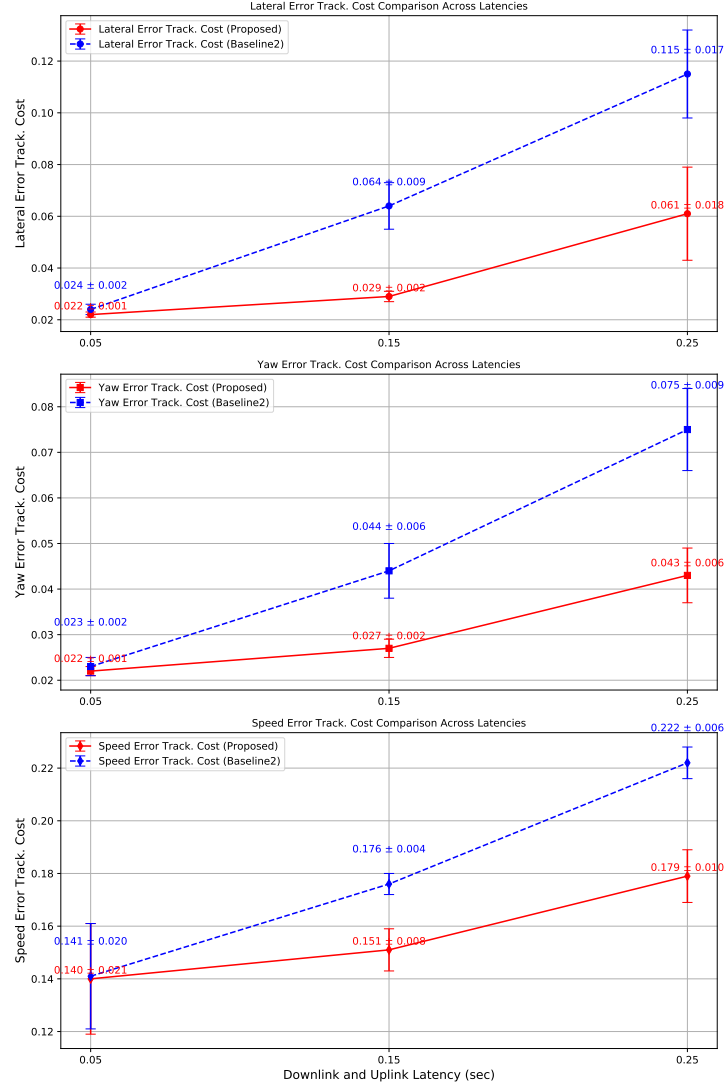


Figure 5.11: Performance, compared to the tracking-error cost, degrades as the average latency increases in acknowledgment-based communication. However, the proposed method still outperforms Baseline2.

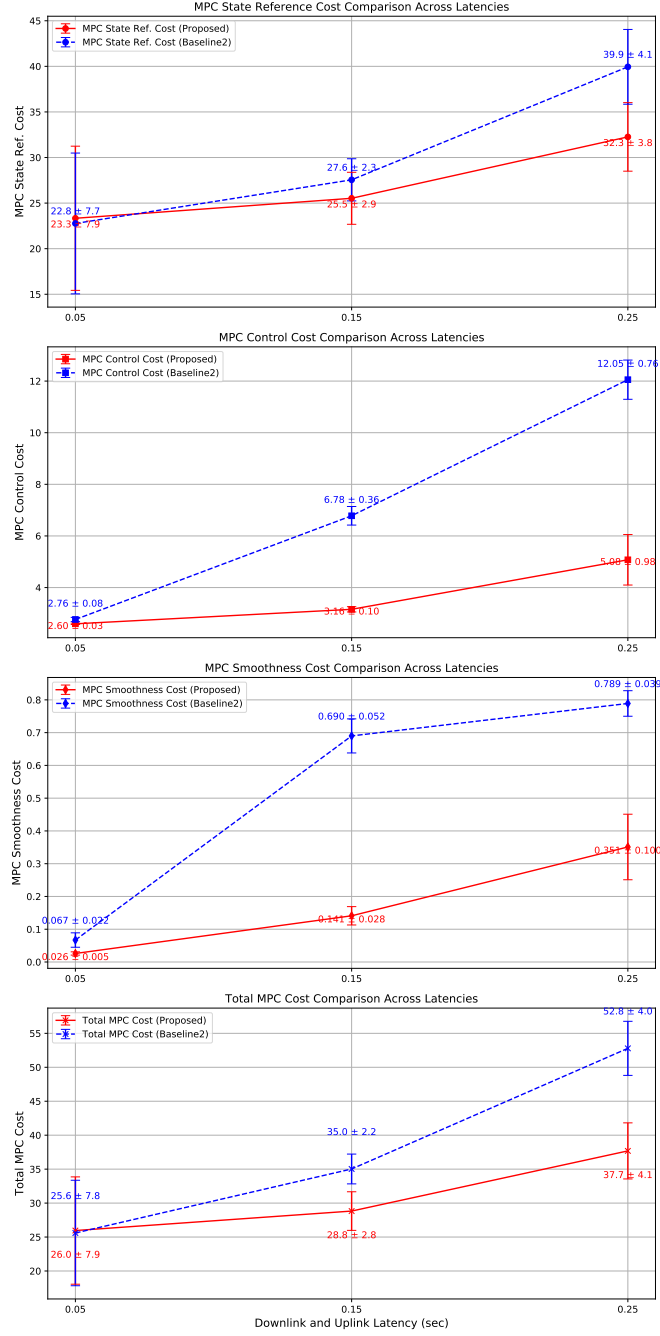


Figure 5.12: Performance, compared to the MPC cost, degrades as the average latency increases in acknowledgment-based communication. However, the proposed method still outperforms Baseline 2.

5.3 Experiment Setup

The overall experimental setup is illustrated in figure 5.13. Every off-board sensor nodes (Lidar as shown in figure 1.2b) is equipped with dual cameras, a LiDAR sensor, 5G/wireless communication capabilities, and a Jetson Orin NX for edge computing [6]. These nodes sample environmental information at a fixed interval, process it, and transmit it to the cloud-computer via Wi-Fi. Similarly, the vehicle sends its state information, such as wheel speed and steering angle, back to the cloud.

The cloud computer runs on a Linux-based platform equipped with an Intel Core i7 3.6GHz CPU, 32GB of RAM, and an Nvidia GeForce RTX2070 graphics card. In the cloud, simulated downlink latency and dropout are introduced to replicate network conditions. These feedback messages are then utilized by the global perception and MPC control modules. Finally, optimal control commands, incorporating simulated uplink latency, are transmitted back to the AMR via Wi-Fi.

The AMR has a ROS stack connected to the MQTT bridge and runs on a Linux-based platform. It receives the control commands from the server and then sends them to the low-level controller at a fixed frequency. The low-level PID controller tracks these references until it receives a new reference command.

During the experiments, as discussed in section 3.7, time synchronization was achieved by connecting all components to the same Internet source, ensuring a common time reference. The overall ROS network architecture is similar to that shown in figure 5.2.

5.4 Experiment Results

Table 5.7: AMR and MPC tuning parameters.

Symbol	Value	Symbol	Value
l_f, l_r	1.2 m	N	12
u_{max}	$[1.2, 1.2, \frac{\pi}{2}, \frac{\pi}{2}]$	R_d	$10 \cdot \text{diag}(10, 10, 50, 50)$
u_{min}	$-[1.2, 1.2, \frac{\pi}{2}, \frac{\pi}{2}]$	Q	$\text{diag}(5, 10, 12, 60, 60)$
Δu_{max}	$[5.5, 5.5, \frac{\pi}{12}, \frac{\pi}{12}]$	R	$10 \cdot \text{diag}(10, 10, 5, 5)$
Δu_{min}	$-[5.5, 5.5, \frac{\pi}{12}, \frac{\pi}{12}]$	$V_{f,ref}$	0.8 m/sec
ΔT_c^s	0.1 sec	$V_{r,ref}$	0.8 m/sec

The parameters used in the experiment are provided in table 5.7. N_c is the control horizon, $V_{f,ref}$ and $V_{r,ref}$ are the reference speeds of the front and rear wheel. The relative magnitude of R_d has been kept higher compared to Q and R to prioritize both tracking

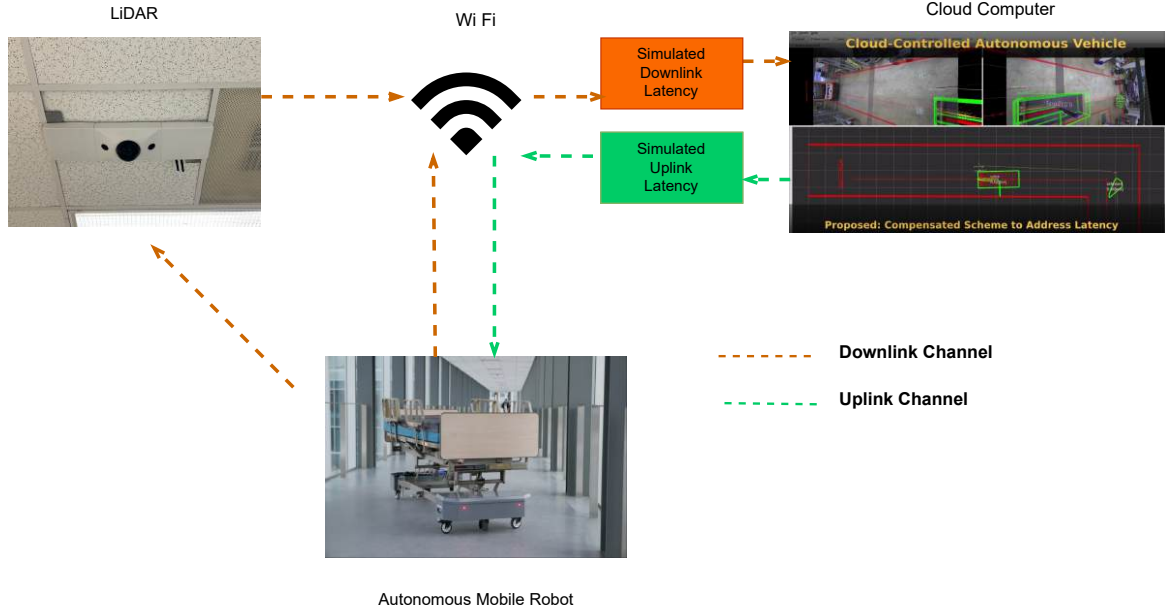


Figure 5.13: Experimental setup.

the reference trajectory and the smoothness of the control actions, thereby minimizing the jerk.

Remark 4. Due to the limited number of experiment runs, the reported results between the Proposed and Baseline2 methods are very close. More variations under different conditions, such as increased speed, latency and high message dropout along with a greater number of experiments, are required to demonstrate significant improvements.

Remark 5. Due to the high weights on smoothness R_d , the normalized Total MPC cost/smoothness cost of the proposed method is sometimes higher than that of the Baseline2 approach.

5.4.1 Fire-and-Forget Communication with Constant Latency and Dropouts

In the following results, dropouts follow a Bernoulli distribution with success probabilities determined by the specified dropout rate of 3%.

Results are summarized in table 5.8 from four experiments on tracking errors and MPC cost breakdowns under two latency conditions (0.25s and 0.5s uplink/downlink). At both

latency levels, the *Proposed* method consistently achieves lower lateral, yaw tracking, and velocity tracking errors compared to *Baseline1* and generally performs similarly to or better than *Baseline2*; see remark 4.

The breakdown of *MPC* costs is summarized in table 5.8b. The *Proposed* method demonstrates significantly lower state reference costs and total *MPC* costs compared to *Baseline1* at both latency levels, indicating superior overall performance. While *Baseline2* maintains competitive performance, it still trails behind the *Proposed* method, especially at lower latency conditions (0.25s). At higher latency, however, *Baseline2* outperforms the *Proposed* method; see remark 5.

The same conclusion can be drawn from figure 5.14, which shows an example of the tracking performance of the three methods under downlink and uplink latency of 0.25 sec, and uplink and downlink latencies of 0.5 sec.

The results shows the effectiveness of the *Proposed* method in reducing the impact of latency on tracking performance and overall control costs.

Table 5.8: Tracking error and *MPC* cost breakdown across methods and latency (average over 4 experiment runs) in fire-and-forget communication.

(a) Tracking Error Cost Breakdown

Latency	Method	Lateral Error Cost	Yaw Error Track. Cost	Speed Error Track. Cost
0.25	Proposed	0.226 ± 0.007	0.252 ± 0.009	0.142 ± 0.001
	Baseline2	0.233 ± 0.007	0.271 ± 0.008	0.152 ± 0.002
	Baseline1	0.250 ± 0.004	0.358 ± 0.018	0.314 ± 0.004
0.5	Proposed	0.233 ± 0.009	0.268 ± 0.005	0.154 ± 0.003
	Baseline2	0.227 ± 0.008	0.276 ± 0.004	0.165 ± 0.002
	Baseline1	0.314 ± 0.018	0.437 ± 0.015	0.369 ± 0.002

(b) *MPC* Cost Breakdown (scaled to 10^2)

Latency	Method	State Ref. Cost	Control Cost	Smoothness Cost	Total <i>MPC</i> Cost
0.25	Proposed	0.863 ± 0.014	0.307 ± 0.012	0.134 ± 0.014	1.304 ± 0.019
	Baseline2	0.963 ± 0.031	0.349 ± 0.024	0.217 ± 0.054	1.529 ± 0.087
	Baseline1	3.265 ± 0.109	0.541 ± 0.029	0.081 ± 0.009	3.886 ± 0.133
0.5	Proposed	1.012 ± 0.031	0.511 ± 0.031	0.628 ± 0.249	2.152 ± 0.248
	Baseline2	1.096 ± 0.028	0.388 ± 0.021	0.222 ± 0.038	1.706 ± 0.026
	Baseline1	4.587 ± 0.035	0.741 ± 0.057	0.105 ± 0.016	5.433 ± 0.088

The costs are normalized by the total reference path length. Two cases of constant latency were considered: (1) uplink and downlink latencies of 0.25 seconds each, and (2) uplink and downlink latencies of 0.5 seconds each.

5.4.2 Acknowledgment-Driven Communication

The results for the acknowledgment-driven communication have been summarized in table 5.9. The methods are evaluated under varying network latencies modeled by truncated Gaussian distributions with means of 0.1, 0.3, and 0.5 seconds, each with $\sigma = 0.02$ s.

The *Proposed* method consistently achieves lower Tracking Error Cost (lateral, yaw, and velocity) compared to *Baseline1* across all latency conditions and performs competitively or better compared to *Baseline2*; see remark 4.

The MPC cost comparisons in table 5.9b further show that the *Proposed* method yields significantly lower state reference and total MPC costs than *Baseline1*, particularly at higher latency conditions (0.3s and 0.5s). However, the smoothness cost of the *Proposed* method is higher than that of *Baseline1* and *Baseline2* due to the high weight on smoothness; see remark 5.

The same conclusion can be drawn from figure 5.15, which shows an example of the tracking performance of the three methods under downlink and uplink latency of 0.1 sec and uplink and downlink latencies of 0.5 sec.

These results highlight the *Proposed* method’s superior performance and effectiveness under acknowledgment-based communication.

Table 5.9: Tracking error and **MPC** cost breakdown across methods and latency (average over 4 Runs) in acknowledgment-based communication.

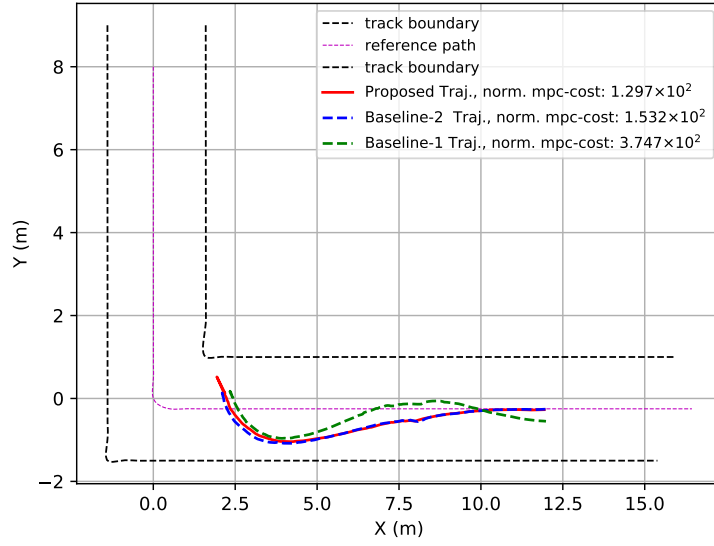
(a) Tracking Error Cost Breakdown

Latency	Method	Lateral Error Track. Cost	Yaw Error Track. Cost	Speed Error Track. Cost
$\mu = 0.1, \sigma = 0.02$	Proposed	0.170 ± 0.002	0.240 ± 0.006	0.144 ± 0.001
	Baseline2	0.165 ± 0.008	0.246 ± 0.010	0.161 ± 0.006
	Baseline1	0.183 ± 0.012	0.322 ± 0.023	0.284 ± 0.007
$\mu = 0.3, \sigma = 0.02$	Proposed	0.184 ± 0.017	0.240 ± 0.010	0.154 ± 0.002
	Baseline2	0.170 ± 0.009	0.262 ± 0.016	0.202 ± 0.002
	Baseline1	0.250 ± 0.033	0.360 ± 0.012	0.350 ± 0.025
$\mu = 0.5, \sigma = 0.02$	Proposed	0.183 ± 0.001	0.259 ± 0.004	0.174 ± 0.003
	Baseline2	0.181 ± 0.002	0.299 ± 0.003	0.233 ± 0.003
	Baseline1	0.311 ± 0.005	0.412 ± 0.013	0.383 ± 0.004

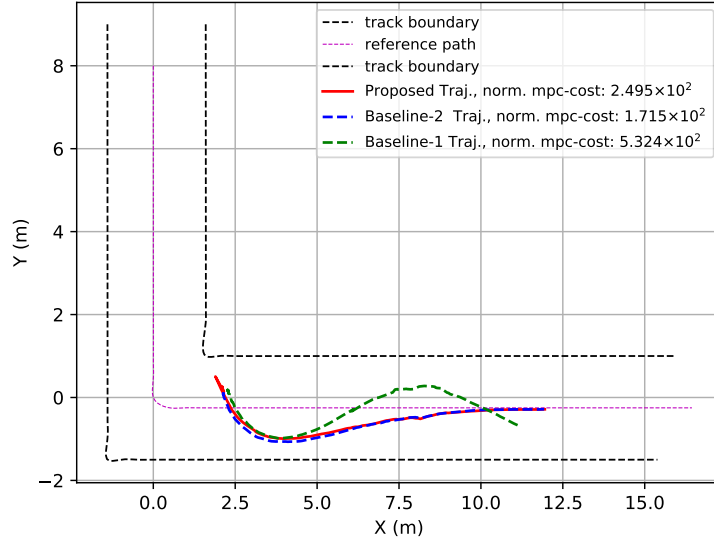
(b) **MPC** Cost Breakdown (scaled to 10^2)

Latency	Method	State Ref. Cost	Control Cost	Smoothness Cost	Total MPC Cost
$\mu = 0.1, \sigma = 0.02$	Proposed	0.795 ± 0.004	0.277 ± 0.018	0.129 ± 0.029	1.201 ± 0.038
	Baseline2	0.935 ± 0.050	0.301 ± 0.016	0.154 ± 0.031	1.390 ± 0.072
	Baseline1	2.630 ± 0.149	0.466 ± 0.074	0.111 ± 0.029	3.207 ± 0.232
$\mu = 0.3, \sigma = 0.02$	Proposed	0.911 ± 0.020	0.286 ± 0.039	0.155 ± 0.060	1.674 ± 0.072
	Baseline2	1.391 ± 0.042	0.304 ± 0.023	0.095 ± 0.025	1.790 ± 0.062
	Baseline1	4.007 ± 0.597	0.610 ± 0.160	0.367 ± 0.527	4.984 ± 1.284
$\mu = 0.5, \sigma = 0.02$	Proposed	1.138 ± 0.032	0.410 ± 0.051	0.370 ± 0.272	2.091 ± 0.355
	Baseline2	1.822 ± 0.050	0.387 ± 0.020	0.189 ± 0.054	2.398 ± 0.088
	Baseline1	4.860 ± 0.111	0.660 ± 0.030	0.126 ± 0.011	5.646 ± 0.120

The costs are normalized by the total reference path length. Three different cases were considered, where the uplink and downlink latencies followed a truncated Gaussian distribution with means of 0.1, 0.3, and 0.5 seconds, respectively, and a standard deviation of 0.02 seconds. The latencies were truncated at ± 3 times the standard deviation.

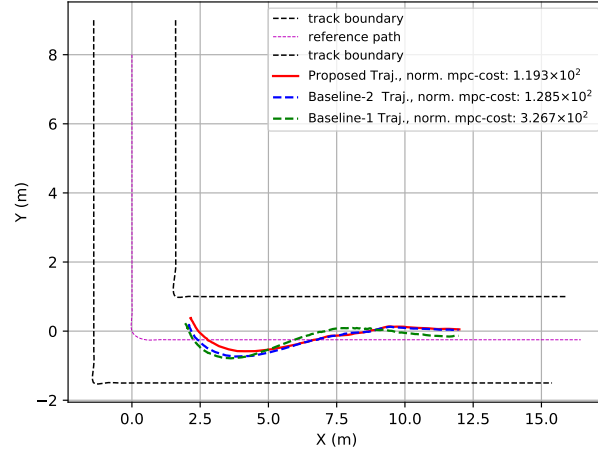


(a) Tracking performance of various methods under a constant latency of 0.25 seconds each for downlink and uplink, with a 3% dropout rate.

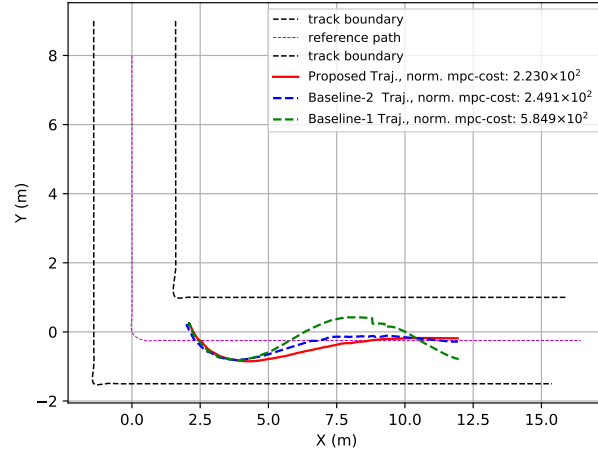


(b) Tracking performance of various methods under a constant latency of 0.5 seconds each for downlink and uplink, with a 3% dropout rate.

Figure 5.14: Tracking performance of various methods under constant latency, with uplink and downlink latencies of (0.25 sec, 0.5 sec) respectively, in fire-and-forget based communication.



(a) Tracking performance of various methods in acknowledgment-based communication, where the uplink and downlink follow a Gaussian distribution with a mean of 0.1 sec and a standard deviation of 0.02 sec.



(b) Tracking performance of various methods in acknowledgment-based communication, where the uplink and downlink follow a Gaussian distribution with a mean of 0.5 sec and a standard deviation of 0.02 sec.

Figure 5.15: Tracking performance of various methods under uplink and downlink latencies following a truncated Gaussian distribution with means of 0.1 sec, 0.3 sec, and 0.5 sec, and a standard deviation of 0.02 sec, respectively, in acknowledgment-based communication.

Chapter 6

Conclusion

This work investigated the motion-planning problem for [AMR](#) under uplink (τ_{ul}) and downlink latency (τ_{dl}) conditions exceeding one control sampling time (ΔT_c^s). A cloud-controller based on [MPC](#) is proposed, which (a) stores previously computed control actions along with their approximate application times (the estimated times at which the actuator will execute the stored control actions) in a buffer, (b) employs the state estimation function as a recursive relationship applied to the vehicle’s nonlinear discrete kinematic equations, predicting the vehicle’s state and corresponding control reference at a future time $t + \tau_{ul}$, where t is the current time, and (c) utilizes these predicted states and control references to linearize the dynamics and solve the [MPC](#) optimization problem. The resulting optimization problem is an optimal control quadratic problem, which is then solved using off-the-shelf solvers. The proposed cloud controller was evaluated in a [ROS](#)-based simulation environment under various network conditions and subsequently deployed in a real [CCAS](#) environment to demonstrate the algorithm’s effectiveness.

The proposed cloud-controller design was tested against two baseline methods. Different evaluation metrics were considered, including the closed-loop [MPC](#) cost and its components, as well as the tracking-error cost. The proposed design demonstrated superior performance across all simulation results in both metrics compared to both baselines.

From the experimental results, the proposed method showed superior performance over the baselines based on the defined metrics, but the results were sometimes inconsistent. This inconsistency stems from the limited number of experiments and the larger weight on the smoothness cost. Further adjustments to the [MPC](#) parameters are needed to clarify this comparison and potentially enhance performance.

6.1 Limitations and Future Work

In this work, a few key assumptions were made to simplify the design of the cloud-control system. These have been discussed in detail in section 3.7. For completeness, we list them here and discuss some of their implications.

First, the system assumes perfect synchronization among network components. In practice, it is unlikely that different vehicles will have synchronized clocks with the cloud. Therefore, it becomes important to handle data from different clock sources carefully. One possible approach could be to rely solely on off-board sensor nodes to estimate the full state of the ego vehicle. A future direction could involve developing methods to mitigate the effects of clock unsynchronization.

Second, the model assumes perfect sensor measurements. Sensor noise was neglected to simplify controller development. In practice, sensor data typically contains noise. Employing filtering or estimation methods such as Kalman filters would enhance the cloud-controller’s capability to estimate the future states of the ego vehicle. However, due to delays, state estimation using Kalman filters becomes challenging. One possible extension could involve incorporating the approach outlined in [38] into the current pipeline.

Third, the vehicle kinematic model used in this work is simplified. The model excluded detailed steering dynamics, which was sufficient for initial demonstrations. However, the assumption of instantaneous steering without dynamic modeling can compromise control accuracy, particularly at higher speeds or during aggressive maneuvers. Future work could involve using high-fidelity dynamics to better align the model with real-world behavior.

One major limitation of this work is that the current design of the cloud-control MPC optimization considers only single-step future latency. The work presented in [13], [19] explores state-augmentation techniques to account for delays over multiple future time steps. Considering successive future delays while solving the optimization at the current time could improve algorithm performance significantly. However, these approaches typically assume that control actions can be computed and sent at fixed intervals, generally at each control sampling time (ΔT_c^s), and that the standard deviation of latency is less than the control sampling time. As discussed earlier, these methods do not account for scenarios involving acknowledgment-based communication.

A potential extension would involve designing a unified framework that handles both fire-and-forget and acknowledgment-based communication models. The algorithm could predict the future states and control references of the AMR, anticipate future delays over multiple time steps, and solve the resulting optimization problem. However, incorporating these elements would significantly increase the complexity of the problem formulation.

Another promising extension could be inspired by reinforcement learning methods for robot control and planning. Reinforcement learning has shown effectiveness in solving complex control problems, including managing latency in robotic systems. The work in [39] proposed using an offline-based Actor-Critic algorithm to mitigate the effects of distributional latency. More recently, work in [40] demonstrated that uplink and downlink latency could be addressed under certain conditions. It proposed using Delay-Reconciled Training for the Critic and State Augmentation for the Actor. Actor learning leverages a control buffer (similar to the one used in this work) to predict future states and optimal control using complex neural networks.

Future research could explore adapting such methods to improve performance under variations in latency, potentially enabling direct prediction of optimal control actions rather than first predicting states and then solving a separate optimization problem.

References

- [1] K. D. Kusano, J. M. Scanlon, Y.-H. Chen, *et al.*, “Comparison of Waymo rider-only crash data to human benchmarks at 7.1 million miles,” eng, *Traffic Injury Prevention*, vol. 25, no. sup1, S66–S77, 2024.
- [2] O. H. Orieno, N. L. Ndubuisi, V. I. Ilojiana, P. W. Biu, and B. Odonkor, “The Future of Autonomous Vehicles in the U.S. Urban Landscape: A Review of Implications for Traffic, Urban Planning, and the Environment,” en, *Engineering Science & Technology Journal*, vol. 5, no. 1, pp. 43–64, Jan. 2024.
- [3] T. Ercan, N. C. Onat, N. Keya, O. Tatari, N. Eluru, and M. Kucukvar, “Autonomous electric vehicles can reduce carbon emissions and air pollution in cities,” *Transportation Research Part D: Transport and Environment*, vol. 112, p. 103472, Nov. 2022.
- [4] A. Caillot, S. Ouerghi, P. Vasseur, R. Boutteau, and Y. Dupuis, “Survey on Cooperative Perception in an Automotive Context,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14204–14223, Sep. 2022.
- [5] A. Rovid, V. Tihanyi, M. Cserni, *et al.*, “Digital Twin and Cloud Based Remote Control of Vehicles,” in *2024 IEEE International Conference on Mobility, Operations, Services and Technologies (MOST)*, May 2024, pp. 154–167.
- [6] M. Ning, Y. Cui, Y. Yang, *et al.*, *Enhancing Indoor Mobility with Connected Sensor Nodes: A Real-Time, Delay-Aware Cooperative Perception Approach*, arXiv:2411.02624, Nov. 2024.
- [7] Z. Li, T. Yu, R. Fukatsu, G. K. Tran, and K. Sakaguchi, “Proof-of-Concept of a SDN Based mmWave V2X Network for Safe Automated Driving,” in *2019 IEEE Global Communications Conference (GLOBECOM)*, ISSN: 2576-6813, Dec. 2019, pp. 1–6.
- [8] J. Prakash, M. Vignati, E. Sabbioni, and F. Cheli, “Vehicle Teleoperation: Successive Reference-Pose Tracking,” in *2022 IEEE Vehicle Power and Propulsion Conference (VPPC)*, arXiv:2206.10340 [cs], Nov. 2022, pp. 1–8.

- [9] A. T. Sheik, C. Maple, G. Epiphaniou, and M. Dianati, “Securing Cloud-Assisted Connected and Autonomous Vehicles: An In-Depth Threat Analysis and Risk Assessment,” en, *Sensors*, vol. 24, no. 1, p. 241, Jan. 2024.
- [10] Z. Xu and Q. Zhu, “Secure and Resilient Control Design for Cloud Enabled Networked Control Systems,” in *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or Privacy*, ser. CPS-SPC ’15, New York, NY, USA: Association for Computing Machinery, Oct. 2015, pp. 31–42.
- [11] Y. Yang, M. Ning, S. Huang, E. Hashemi, and A. Khajepour, *Intelligent Mobility System with Integrated Motion Planning and Control Utilizing Infrastructure Sensor Nodes*, arXiv:2410.22527, Nov. 2024.
- [12] T. Betz, P. Karle, F. Werner, and J. Betz, “An Analysis of Software Latency for a High-Speed Autonomous Race Car—A Case Study in the Indy Autonomous Challenge,” English, *SAE International Journal of Connected and Automated Vehicles*, vol. 6, no. 3, pp. 283–296, Feb. 2023, Number: 12-06-03-0018.
- [13] S. Xu, H. Peng, and Y. Tang, “Preview Path Tracking Control With Delay Compensation for Autonomous Vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, pp. 1–11, Apr. 2020.
- [14] S. B. Kamtam, Q. Lu, F. Bouali, O. C. L. Haas, and S. Birrell, “Network Latency in Teleoperation of Connected and Autonomous Vehicles: A Review of Trends, Challenges, and Mitigation Strategies,” en, *Sensors*, vol. 24, no. 12, p. 3957, Jan. 2024.
- [15] W. Cao, J. Liu, J. Li, Q. Yang, and H. He, “Networked Motion Control for Smart EV With Multiple-Package Transmissions and Time-Varying Network-Induced Delays,” *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 4076–4086, Apr. 2022.
- [16] J. Pan, Q. Xu, K. Li, and J. Wang, “A vehicle cloud control system considering communication quantization and stochastic delay,” en, *Asian Journal of Control*, vol. 25, no. 5, pp. 3616–3631, 2023.
- [17] Y. Deng, V. L  chapp  , E. Moulay, *et al.*, “Predictor-based control of time-delay systems: A survey,” *International Journal of Systems Science*, vol. 53, no. 12, pp. 2496–2534, Mar. 2022.
- [18] O. J. Smith, “A controller to overcome dead time,” *ISA J.*, vol. 6, pp. 28–33, 1959. en, Jan. 1959.
- [19] N. E. Kahveci and P. A. Ioannou, “Automatic steering of vehicles subject to actuator saturation and delay,” in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, ISSN: 2153-0017, Oct. 2011, pp. 119–124.

- [20] J. Chu, “Application of a discrete optimal tracking controller to an industrial electric heater with pure delays,” *Journal of Process Control*, vol. 5, no. 1, pp. 3–8, Jan. 1995.
- [21] J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, “A Multi-Stage Optimization Formulation for MPC-Based Obstacle Avoidance in Autonomous Vehicles Using a LIDAR Sensor,” en, in *Volume 2: Dynamic Modeling and Diagnostics in Biomedical Systems; Dynamics and Control of Wind Energy Systems; Vehicle Energy Management Optimization; Energy Storage, Optimization; Transportation and Grid Applications; Estimation and Identification Methods, Tracking, Detection, Alternative Propulsion Systems; Ground and Space Vehicle Dynamics; Intelligent Transportation Systems and Control; Energy Harvesting; Modeling and Control for Thermo-Fluid Applications, IC Engines, Manufacturing*, San Antonio, Texas, USA: American Society of Mechanical Engineers, Oct. 2014, V002T30A006.
- [22] S. Kato, S. Tokunaga, Y. Maruyama, *et al.*, “Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems,” in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, Apr. 2018, pp. 287–296.
- [23] G. P. Liu, “Predictive Controller Design of Networked Systems With Communication Delays and Data Loss,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 6, pp. 481–485, Jun. 2010.
- [24] M. Pezzutto, M. Farina, R. Carli, and L. Schenato, “Remote MPC for Tracking Over Lossy Networks,” *IEEE Control Systems Letters*, vol. 6, pp. 1040–1045, 2022.
- [25] L. Zhao, M. Nybacka, M. Aramrattana, *et al.*, “Remote Driving of Road Vehicles: A Survey of Driving Feedback, Latency, Support Control, and Real Applications,” *IEEE Transactions on Intelligent Vehicles*, pp. 1–22, 2024.
- [26] X. Zhou, Z. Yang, Y. Ren, W. Bai, B. Lo, and E. M. Yeatman, “Modified Bilateral Active Estimation Model: A Learning-Based Solution to the Time Delay Problem in Robotic Tele-Control,” *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2653–2660, May 2023.
- [27] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. Rehg, and E. Theodorou, “Robust Sampling Based Model Predictive Control with Sparse Objective Information,” en, in *Robotics: Science and Systems XIV*, Robotics: Science and Systems Foundation, Jun. 2018.
- [28] B. Sharma and A. K. Singh, *Trajectory Optimization Under Stochastic Dynamics Leveraging Maximum Mean Discrepancy*, arXiv:2501.19045 [cs], Jan. 2025.
- [29] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*, en, Jun. 2017.

- [30] *MQTT Publish, MQTT Subscribe & Unsubscribe – MQTT Essentials: Part 4*, en, Feb. 2024.
- [31] D. Jang, J. Yoo, C. Y. Son, H. J. Kim, and K. H. Johansson, “Networked Operation of a UAV Using Gaussian Process-Based Delay Compensation and Model Predictive Control,” en, in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, May 2019, pp. 9216–9222.
- [32] M. Quigley, “ROS: An open-source Robot Operating System,” in *IEEE International Conference on Robotics and Automation*, 2009.
- [33] M. Gheisarnejad, M.-H. Khooban, and T. Dragičević, “The Future 5G Network-Based Secondary Load Frequency Control in Shipboard Microgrids,” *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 8, no. 1, pp. 836–844, Mar. 2020.
- [34] D. Wang and F. Qi, “Trajectory planning for a four-wheel-steering vehicle,” in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, ISSN: 1050-4729, vol. 4, May 2001, 3320–3325 vol.4.
- [35] J. Kabzan, M. d. l. I. Valls, V. Reijgwart, *et al.*, *AMZ Driverless: The Full Autonomous Racing System*, arXiv:1905.05150 [cs], May 2019.
- [36] G. Frison and M. Diehl, *HPIPM: A high-performance quadratic programming framework for model predictive control*, arXiv:2003.02547 [math], Jun. 2020.
- [37] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, “Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing,” in *2007 American Control Conference*, ISSN: 2378-5861, Jul. 2007, pp. 2296–2301.
- [38] S. K. Nanda, G. Kumar, V. Bhatia, and A. K. Singh, “Kalman Filtering With Delayed Measurements in Non-Gaussian Environments,” *IEEE Access*, vol. 9, pp. 123 231–123 244, 2021.
- [39] Y. Bouteiller, S. Ramstedt, G. Beltrame, C. Pal, and J. Binas, “Reinforcement Learning with Random Delays,” en, in *International Conference on Learning Representations*, Oct. 2020.
- [40] W. Wang, D. Han, X. Luo, and D. Li, “Addressing Signal Delay in Deep Reinforcement Learning,” en, in *The Twelfth International Conference on Learning Representations*, Oct. 2023.