IMPLEMENTATION OF CONSENSUS ALTERNATING DIRECTION METHOD OF MULTIPLIERS IN A MODEL PREDICTIVE CONTROL PROBLEM WITH APPLICATIONS IN TRAFFIC NETWORKS

by

Joseph Tolone

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Mechanical Engineering

Charlotte

2023

Approved by:

_____

Dr. Amir Ghasemi

_____

Dr. Artur Wolek

_____

Dr. Scott Kelly

ABSTRACT

JOSEPH TOLONE. Implementation of consensus alternating direction method of multipliers in a model predictive control problem with applications in traffic networks. (Under the direction of DR. AMIR GHASEMI)

In this work, consensus Alternating Direction Method of Multipliers (ADMM) was implemented with the intention of improving the performance of Model Predictive Control (MPC) on an automobile traffic network. Automobile traffic networks must be engineered to be as efficient as possible in order to mitigate their contributions to climate change. In addition to global climate implications, more efficient traffic networks can benefit individual consumers. This work examines methods for controlling traffic networks with digital speed limits to enable these positive outcomes. Beyond practical interests, traffic network control is of theoretical interest because of the potential to apply traffic control schemes on other systems. MPC is one method for traffic network control. The main benefits of MPC in this application are that it can constrain inputs and handle disturbances. MPC is good for handling disturbances because it predicts future system behavior to inform control decisions. However, this also means that MPC can be computationally expensive. This work attempts to combat the computational expense by implementing consensus ADMM within the MPC optimization. Consensus ADMM will be used to break the computation down into smaller pieces that overlap in their effective range before recombining them into one solution. Subsequently, this method should lighten the computational load. In the context of traffic networks, a length of road will be broken down into multiple sections which can be referred to as "agents." An agent's control command will be the ideal speed limit(s) within its effective range. In this work, MPC and basic ADMM are first applied on a translational cart system. In the next case study, consensus ADMM will be implemented on a multi-agent, interconnected version of this cart system. Lastly, MPC with consensus ADMM will be applied to a traffic network simulation.

DEDICATION

The author dedicates this thesis to his parents, grandparents, and siblings, who have unceasingly supported him throughout this work. This work is also dedicated to the professors of the University of North Carolina at Charlotte, and Emory & Henry College whose dedicated instruction has helped enabled him to work on such a project.

ACKNOWLEDGEMENTS

The author acknowledges and thanks Professor Amir Ghasemi for his role as advisor in this work. His dedication to this project has been instrumental in its progression. The author also acknowledges and thanks Professor Artur Wolek and Professor Scott Kelly who have graciously agreed to serve on this defense committee, and whose efforts in the classroom are greatly appreciated. The author additionally acknowledges and thanks the members of The Human-Machine Interaction and Control lab.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADMM Alternating Direction Method of Multipliers

MPC Model Predictive Control

CHAPTER 1: INTRODUCTION

A common problem in control theory is the discrepancy between how long it takes for a control command to be calculated and when it can be implemented on a system. This issue can arise in different situations. For example, discrete-time systems receive measurements and output commands according to predetermined time steps. For a system controlled by Model Predictive Control (MPC), the control command is ideally calculated before the given command time interval passes. Thus, in the case of MPC, a controller loses its effectiveness if it is unable to calculate its control command update within this window of time. The same can be true for continuous systems updated in this way according to small time steps. As a trivial example, it would clearly be problematic if some kind of autonomous vehicle needed an hour to calculate input updates. At its very core, this problem of efficient command calculation is the problem that this research addresses.

The dynamic system examined in this research is an automobile traffic network. This system will be studied through a wide lens, concerning itself with broader traffic network performance, rather than the performance of a single vehicle. The specific characteristics by which this system will be quantified are average vehicle density, velocity, and flow. A more thorough discussion of these terms will take place in the following chapters. The overarching goal of this work is to implement an MPC-based method that enables improved efficiency and flow of these traffic networks. This method will incorporate the Alternating Direction Method of Multipliers (ADMM) in effort to accomplish this.

1.1 Problem Background

Control will be actuated on the automobile traffic network through digitally updating speed limits. These digital speed limits will span a section of highway so that some portion of the network can be controlled. This work examines the effectiveness of this strategy from a mathematical perspective.

This problem is being studied for a number of reasons. One of which has to do with the potential to positively influence consumers. The 2021 data given in [1] notes that the average household spending on transportation was $10,961, and that households spent the majority of their transportation allocation on private vehicles. This data demonstrates the nontrivial nature of transportation spending, and why it is worth investigating methods that might improve it. If this method of control is successful in increasing average traffic flow for commonly problematic sections of highway, a consumer may see a reduced amount of time spent on the road when looking at this example in isolation. It could be possible that this alone may help mitigate transportation costs.

This work will focus on the MPC control of traffic networks through the adjustment of the variable speed limit, but there are other approaches to improving traffic flow as well. In [2], some microscopic-level control approaches that do not focus on large numbers of vehicles are examined. The authors of [3] in part examine MPC control for ramp metering *and* digitally updating speed limits. The authors of [4] attempt to use a variation of MPC to control traffic flow, but this work is done in the context of urban networks. The work in [4] specifically focuses on non-distributed methods for mitigating the negative effects of computational complexities from MPC by modifying the MPC problem structure. The research that will be discussed here itself focuses on distributed methods for optimal control in part because the networks considered here are very large. This could cause centralized methods to be impractical.

Developing a control strategy that works for a large network could be advantageous

for future work for the potential of the distributed aspect of this technique to be applied to another computationally complex system. An example of one such system is given in [5]. In [5], ADMM is incorporated into the solution of a distributed optimization problem that seeks to control railway traffic networks.

MPC will be described in detail in chapter 2, but regular MPC itself is a control strategy that has potential to be computationally expensive. MPC chooses a control strategy by predicting future behavior of the system according to the equations that govern the system, as described in [6]. This allows MPC to make an informed decision on how best to control a system from a given system configuration. Certain MPC parameters can be tuned to trade relative degree of optimization for computational speed, as described in [6]. System characteristics will determine whether it is possible to achieve a desired level of optimization within the allowed computation time frame. This work seeks to optimize the overall command computation times to limit compromises to system performance.

1.2 Project-Specific Characteristics

This research examines a traffic network control problem similar to the one diagrammed by figure 1.1. More specifically, the traffic cells considered in this work have the same length, and only one lane (see figure 1.2). The system will be controlled by giving vehicles in the system a suggested velocity to follow. These modifications to the traffic model will simplify the development of a distributed control strategy. Once fundamental distributed control of this system has been attained, the model can be made more detailed in future work.

This traffic system will be divided into a number of agents. In the distributed control implementation, each of these agents will act with some degree of independence relative to a centralized optimization. The mechanics of how the distributed control algorithm will work is based on a variation of the Alternating Direction Method of Multipliers (ADMM), and will be discussed in the following chapters.

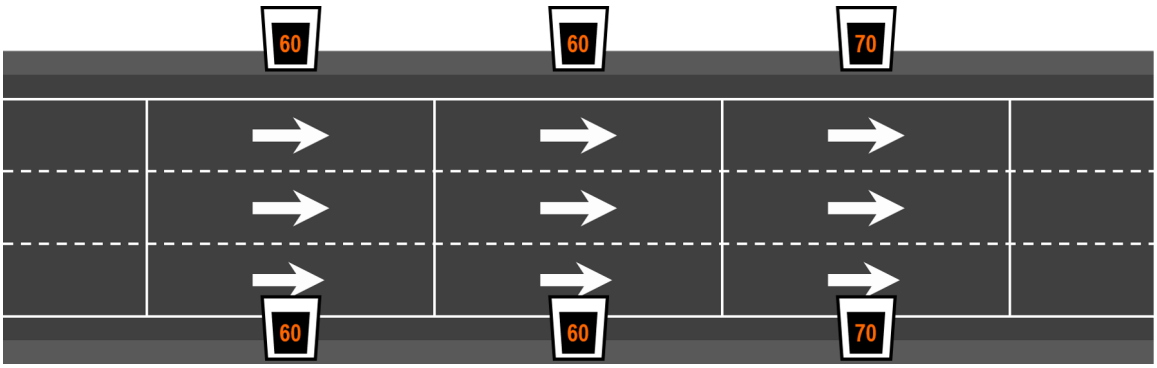

Figure 1.1: Example of a traffic network with multiple lanes. Dynamically updating speed limits are shown for each cell.

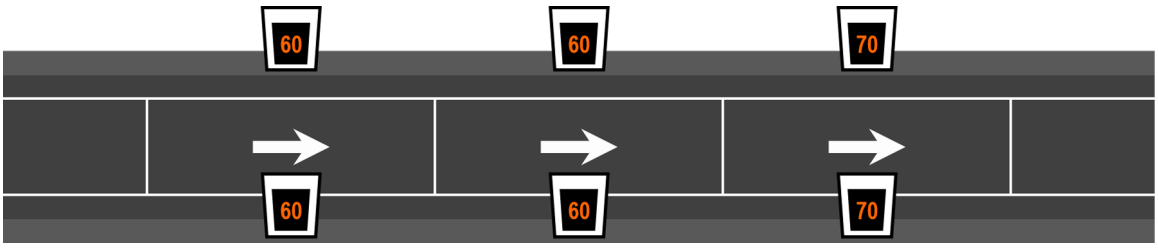

Figure 1.2: Example of a traffic network with a single lane. Dynamically updating speed limits are shown for each cell.

As discussed above, the distributed control strategy developed here focuses on a deterministic traffic network. This method seeks to control a traffic network by following a similar strategy to [7], which itself is based on mathematical principles of ADMM that are discussed in detail in [8]. This will be done by distributing the control of the automobile traffic network among various groupings of overlapping control agents. Each cell grouping will create a control strategy at each time step that will be combined with other local control strategies according to a variation of the ADMM method.

1.3 Discussion Overview

In the remainder of this discussion, chapter 2 will discuss the primary mathematical and conceptual topics that are especially prominent in this research. After laying the mathematical foundation, three case studies will be examined in chapters 3–5. Case study I in chapter 3 will formulate a problem that allows for the direct implementation

of the mathematical topics discussed in chapter 2. Chapter 4 will build on the work of chapter 3 by modifying the system and the control algorithm for case study II. Case study III will then attempt to implement the foundational elements of case studies I and II on a traffic network in chapter 5. Lastly, chapter 6 will conclude this discussion with an overarching discussion of results, their meaning, and future work.

CHAPTER 2: OVERVIEW OF TOPICS

The foundational mathematical and control theory concepts central to the work presented here will be discussed in this chapter. As noted in chapter 1, the general objective of this research is to develop a system that uses Model Predictive Control (MPC) to optimally control a traffic network. Because MPC can potentially require a large amount of computational resources, the Alternating Direction Method of Multipliers (ADMM) will be implemented to ensure the timely computation of control commands in part through the use of distributed optimization. It is important to note here that MPC will be used to control the system, and ADMM will be used to ensure the feasibility of MPC by speeding up the MPC calculations. In section 2.1, the foundational principles of ADMM will be discussed. Section 2.2 will then make explicit the ADMM algorithm, and section 2.3 will discuss the MPC algorithm. Lastly, section 2.4 will describe the manner through which ADMM is incorporated into MPC. This will help support the formulation of case study I in the following chapter.

2.1 Foundational Components of ADMM

As noted in [8], ADMM is an algorithm based on the combination of two simpler algorithms: the dual ascent method, and the method of multipliers. As shown in [8], both of these algorithms minimize some objective function with respect to some variable. ADMM combines the strengths of each algorithm in a way that mitigates the limiting factors inherent to each algorithm [8]. A brief discussion of the formation of this ADMM method is provided in this section, according to the formulation and

description in [8][1]. The reader is directed to the work in [8] for a more thorough description of this algorithm.

2.1.1 Dual Ascent and Dual Decomposition

As described in [8], the dual ascent method seeks to minimize the output of a function by minimizing a modified equation based on the objective function of the system. This modified equation is called the *Lagrangian* of the system. The dual ascent algorithm is given below, according to the formulation in [8].

$$\text{Minimize}: \quad f(x) \qquad \text{Subject to}: \quad Ax = b \tag{2.1}$$

$$\text{Lagrangian}: \quad L(x, y) = f(x) + y^{\top}(Ax - b) \tag{2.2}$$

$$\text{Algorithm}: \quad x^{k+1} := \underset{x}{\operatorname{argmin}}\ L(x, y^k) \tag{2.3}$$

$$y^{k+1} := y^k + \alpha^k(Ax^{k+1} - b) \tag{2.4}$$

In the algorithm given by equations (2.3) and (2.4), the variable x is the variable by which the Lagrangian of the objective function is minimized. The variable y in these equations is referred to as the *dual variable,* as in [8]. The dual variable in this case acts similarly to a penalty constant because as the constraint ($Ax = b$) is violated, the output cost is altered since $Ax - b \neq 0$. Also, α is a tuneable parameter, and k represents the iteration number. Dual ascent solves the problem given by (2.1) by iterating through equations (2.3) and (2.4), as described in [8]. First, the Lagrangian for the system, given by equation (2.2), is minimized with respect to x. During this minimization of x, the dual variable y is held constant. Following the Lagrangian minimization, y is updated in the dual update step. The iteration number k is then incremented before the algorithm is repeated.

As discussed in [8], the primary benefit of the dual ascent method is that it allows for

[1]Unless otherwise noted, the equations in the following subsections are completely removed from dynamic system analysis. All variables should be treated as purely mathematical symbols.

parallel optimization when the function f is separable. When f is separable, instead of being called dual ascent, the algorithm is referred to as *dual decomposition* [8]. The dual decomposition version of this algorithm is given below for $f(x) = \sum_{i=1}^{N} f_i(x_i)$, again according to the formulation in [8].

$$\text{Minimize}: \ f(x) \qquad \text{Subject to}: \ Ax = b$$

$$\text{Lagrangian}: \ \sum_{i=1}^{N} L_i(x_i, y) = \sum_{i=1}^{N} \left(f_i(x_i) + y^\top (A_i x_i - (1/N) y^\top b) \right) \tag{2.5}$$

$$\text{Algorithm}: \ x_i^{k+1} := \underset{x_i}{\operatorname{argmin}} \ L_i(x_i, y^k) \tag{2.6}$$

$$y^{k+1} := y^k + \alpha^k (A x^{k+1} - b) \tag{2.7}$$

As noted in [8], the new algorithm given by equations (2.6) and (2.7) is largely carried out in the same way as the dual ascent algorithm given by equations (2.3) and (2.4). Because of this, the same meanings are attributed to each symbol. The key difference, as noted in [8], is that the Lagrangian minimization step can be done in parallel for each Lagrangian term indexed by i, from 1 to N. This is crucial to this research because parallel optimization can greatly decrease computation times. Parallel optimization accomplishes this by breaking a problem down into smaller pieces, just like how the Lagrangian given by equation (2.5) can be separated by each index so that one local problem depends only on its own variables.

In the context of a traffic network, this parallelization is especially relevant because traffic cells depend primarily on their surrounding cells, rather than on distant cells. Thus, a collection of these smaller problems can collectively describe the whole system. If it is faster to solve the smaller problems than to solve the original large problem, this parallelization would be useful for the given system.

While the separability of dual decomposition is desirable for the traffic network application, there is a major drawback to this algorithm, as noted in [8]. This draw-

back is that dual ascent (and subsequently dual decomposition) require the objective function f to be convex [8]. Since the cost functions for the traffic system may vary according to system characteristics or the desired output of the system, some of these cost functions could potentially be nonconvex. Therefore, despite the potential for faster parallel computation, dual decomposition on its own is not a sufficient algorithm for the purposes of this research.

2.1.2 Method of Multipliers

The method of multipliers, also described in [8], is structured similarly to dual ascent and dual decomposition. Subsequently, it too seeks to minimize some objective function by repeatedly minimizing a modified version of this original function. The method of multipliers algorithm is given below, again according to the formulation in [8].

$$\begin{aligned} \text{Minimize}: \ & f(x) \qquad \text{Subject to}: \ Ax = b \\ \text{Lagrangian}: \ & L_\rho(x, y) = f(x) + y^\top (Ax - b) + (\rho/2)\|Ax - b\|_2^2 && (2.8) \\ \text{Algorithm}: \ & x^{k+1} := \underset{x}{\operatorname{argmin}}\ L_\rho(x, y^k) && (2.9) \\ & y^{k+1} := y^k + \rho(Ax^{k+1} - b) && (2.10) \end{aligned}$$

The primary benefit of the method of multipliers is that the function f is not required to be convex [8]. If the augmented Lagrangian term, $(\rho/2)\|Ax - b\|_2^2$, is not included, this algorithm would be equivalent to dual ascent [8]. All of the terms in equations (2.8), (2.9), and (2.10) are analogous to those given by the dual ascent and dual decomposition algorithms above with one exception. This exception is the term ρ, which is the penalty parameter [8]. As with α in dual ascent/decomposition, ρ scales the impact of the constraint violation, $Ax = b$.

As noted in [8], a key drawback of the method of multipliers is that the new Lagrangian is not separable. This is problematic for the traffic network system due to

its large size creating a computationally expensive control calculation. If this large problem cannot be broken down into simpler parts, real-time control implementation could be compromised. Therefore, another method is needed that combines the advantages of dual decomposition and the method of multipliers to solve separable problems that are not necessarily convex.

2.2 ADMM Assembly

As noted previously, ADMM builds off of the foundational algorithms described in sections 2.1.1 and 2.1.2. In so doing, ADMM takes advantage of the strengths of each algorithm [8]. ADMM is separable like dual ascent (dual decomposition), and it does not require the cost function to be convex like in the method of multipliers [8]. This algorithm is described in rigorous detail in [8], but a brief description and pictoral representation are given here.

2.2.1 ADMM Algorithm

ADMM, like the dual ascent method and method of multipliers, minimizes some cost function subject to a given set of constraints [8]. The pivotal difference between ADMM and both dual ascent and the method of multipliers is the splitting of the variable of interest, shown in equation (2.11) below [8]. Equation (2.11) demonstrates how the "original" objective function (some arbitrary $J(x)$) has been separated into two functions $f(x)$ and $g(z)$. The relationship between the variables x and z is also given by equation (2.11).

The splitting of variables described above enables the cost function to be broken down into smaller components. As an example, if one desired to minimize the cost function $J(x) = \|x\|_2^2 + \|x\|_1$, one could define the constraint, $Ax + Bz = c$, such that A is the identity matrix, B is the negative identity matrix, and c is zero. In so doing, $Ax + Bz = c$ is thus simplified to $x - z = 0$, which itself is equivalent to stating that $x = z$. With this choice of constraint relating x and z, the cost function

can be written in terms of both variables such that $J(x,z) = f(x) + g(z)$. Choosing $f(x) = \|x\|_2^2$ and $g(z) = \|z\|_1$, one can rewrite $J(x,z) = \|x\|_2^2 + \|z\|_1$ as the new cost function[2]. This cost function can be minimized in an alternating fashion according to the algorithm given by equations (2.13), (2.14), and (2.15). This example is based in part on the work of [9], and on the explanations in [8].

$$\text{Minimize}: \ f(x) + g(z) \qquad \text{Subject to}: \ Ax + Bz = c \tag{2.11}$$

$$\text{Lagrangian}: \ L_\rho(x,z,y) = f(x) + g(z) + y^\top(Ax + Bz - c) + \cdots$$

$$\cdots + (\rho/2)\|Ax + Bz - c\|_2^2 \tag{2.12}$$

$$\text{Algorithm}: \ x^{k+1} := \underset{x}{\operatorname{argmin}} \ L_\rho(x, z^k, y^k) \tag{2.13}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \ L_\rho(x^{k+1}, z, y^k) \tag{2.14}$$

$$y^{k+1} := y^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \tag{2.15}$$

The iteration through these equations can be thought of in the following way. First, the Lagrangian is minimized with respect to x, keeping the previous values for z and y constant. Second, using the newly determined x^{k+1} from equation (2.13), the Lagrangian is minimized with respect to z so that the x^{k+1} and y^k are held constant. Third, the dual variable y is updated using the newly determined x^{k+1} and z^{k+1} values from equations (2.13) and (2.14) respectively. This algorithm can be rewritten in another way, which will be used in case study I.

2.2.2 Scaled ADMM

When minimizing the Lagrangian first with respect to x and then with respect to z in equations (2.13) and (2.14), some terms were not factored in to the minimizations. For example, the variable z is held constant in equation (2.13). Since the z-term is additive, the value from $g(z)$ would have no impact on the system performance in

[2]Because $x = z$, this choice of J is equivalent to choosing $J(x,z) = \|z\|_2^2 + \|x\|_1$, where $f(x) = \|x\|_1$ and $g(z) = \|z\|_2^2$.

equation (2.13). This fact alone creates opportunity for the algorithm to be simplified notationally. In addition, further simplifications can be made by defining a new variable u, such that $u = (1/\rho)y$ [8]. The simplified algorithm, provided in [8] and also used in [9], is called the "scaled" ADMM algorithm[3]. This is algorithm is given below, and is equivalent to the ADMM algorithm given in subsection 2.2.1 [8]. This is the main algorithm to be used in case study I.

$$\text{Algorithm}: \quad x^{k+1} := \underset{x}{\operatorname{argmin}}\ f(x) + (\rho/2)\|Ax + Bz^k - c + u^k\|_2^2 \tag{2.16}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}}\ g(z) + (\rho/2)\|Ax^{k+1} + Bz - c + u^k\|_2^2 \tag{2.17}$$

$$u^{k+1} := u^k + Ax^{k+1} + Bz^{k+1} - c \tag{2.18}$$

It is easier to see the inner workings of ADMM when it is presented in the scaled form. According to this formulation, the x component of the cost function is first updated, followed by the z component update. The z component update uses the result of the x minimization in its computation. By minimizing $J(x, z) = f(x) + g(z)$ in this way, it is as if the objective function is being minimized in a partitioned fashion, as described in [8]. The subsequent dual update of the variable u can be thought of as the accumulation of discrepancy between x and z [8].

In the original version of the ADMM algorithm from subsection 2.2.1 and the scaled version of ADMM presented here, a fourth step to the algorithm has been omitted. This step will be examined in subsection 2.2.3.

2.2.3 ADMM Stopping Criteria

After the first three steps of the ADMM algorithm[4], it must be determined whether another iteration is required. This is done by checking the stopping criteria, as will

[3]The reader is again reminded that the variable u should be treated as a purely mathematical symbol with no relation to dynamic system control inputs. Furthermore, the A and B matrices also have nothing to do with system dynamics. These matrices only establish a relationship between the x and z vectors.

[4]...or before each iteration of ADMM, depending on how one views the algorithm...

be discussed below.

Before the algorithm starts, the user can define a maximum number of ADMM iterations allowed. Thus, one stopping criterion is whether this maximum number of iterations has been reached. If it has been reached, then the algorithm concludes after the last dual variable update, and the last value for z is the final output of the system[5].

The other stopping criterion will be referred to here as the *calculated* stopping criterion. This stopping criterion checks the values of the primal residual r, and the dual residual s against the feasibility tolerances ϵ^{pri} and ϵ^{dual} [8]. The definitions of these values are given below, as described in [8]. Here, P represents the number of rows of matrix A, and N represents the number of columns of matrix A. Additionally, ϵ^{abs} and ϵ^{rel} are determined by the user. For the simulations presented in this work, ϵ^{abs} and ϵ^{rel} were always 0.0001 and 0.01 respectively. This choice of feasibility tolerances was based on supporting materials provided with [8] by its authors.

$$r^{k+1} = Ax^{k+1} + Bz^{k+1} - c \tag{2.19}$$

$$s^{k+1} = \rho A^{\top} B(z^{k+1} - z^{k}) \tag{2.20}$$

$$\epsilon^{pri} = \sqrt{P}\epsilon^{abs} + \epsilon^{rel} \max\{\|Ax^{k+1}\|_2, \|Bz^{k+1}\|_2, \|c\|_2\} \tag{2.21}$$

$$\epsilon^{dual} = \sqrt{N}\epsilon^{abs} + \epsilon^{rel}\|\rho A^{\top} u^{k}\|_2 \tag{2.22}$$

It is important to note that r^{k+1} and s^{k+1} are vectors. This is important because the two-norm of these vectors is used later in the stopping criteria check. These equations are given for the scaled form of ADMM. This means that u^k in equation (2.22) is the scaled dual variable. The calculated stopping criterion as discussed in

[5]If the user were to select the final value for x as the output of the system, that would mean an incomplete iteration of the algorithm is performed in the last iteration. For the sake of consistency, the z-update step will be treated as the most meaningful final output of the system.

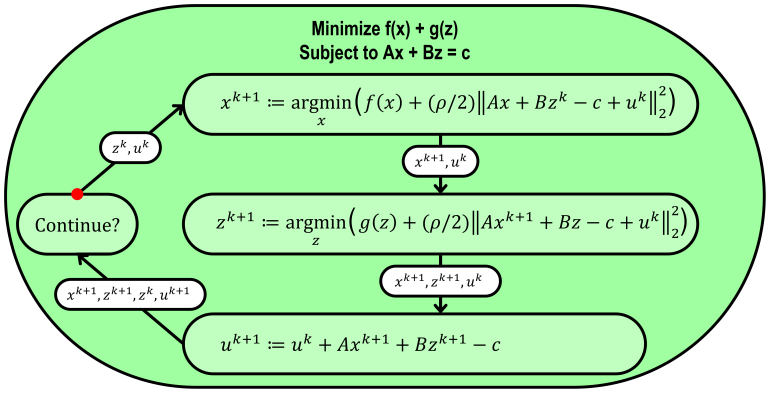

Figure 2.1: Basic ADMM algorithm. The minimization problem is given in bold text at the top of the diagram. The red dot represents the starting point of the algorithm.

[8] is given below by equation (2.23).

$$\text{if} \quad \left\{\|r^k\|_2 \leq \epsilon^{pri} \quad \text{AND} \quad \|s^k\|_2 \leq \epsilon^{dual}\right\} \quad \Longrightarrow \quad \text{end algorithm} \tag{2.23}$$

Each iteration of ADMM must check both stopping criteria. This ensures that the algorithm does not propagate unnecessarily.

2.2.4 ADMM Summary

The scaled ADMM algorithm in its most basic form is given by figure 2.1. The lighter green bubbles represent each step in the algorithm. The lighter green bubble labeled, "Continue?" is the stopping criteria check. Additionally, the white bubbles represent the main inputs to each step in the algorithm. This algorithm is foundational to the rest of the work presented in this discussion.

2.3 Model Predictive Control

The experimentation presented later in this work uses Model Predictive Control (MPC) control the dynamic system. This algorithm works by minimizing some cost function that correlates system outputs to a degree of control success, and is enabled by MPC making predictions about future system behavior [6]. These predictions generate the data that is given to a cost function. This algorithm is described in

detail below, based on the description given by [6]. MPC is also used in [7] and [9] in their ADMM and MPC applications.

When control of a given dynamic system begins, the MPC controller must first obtain measurement data from the system. From the measured configuration of the system, MPC creates and tests a single control strategy. In this work, this is done by running a simulation of the system behavior out to some time in the future. This creates a set of output prediction data, which is assigned a scalar value that describes the degree to which the given output is desirable. This scalar value is called the *cost* of that input strategy, and is based on the cost function for the system. For the sake of simplicity in this explanation, it will be assumed that the cost must always be greater than or equal to zero, with zero representing the optimal solution. Informed by the resulting cost of the given control strategy, the MPC controller then tests a new set of control inputs for the same initial conditions on the system.

Once the controller is satisfied it has found the solution that most minimizes the cost, the first time step of the control strategy is then implemented on the real system, as noted by [6]. It is important to emphasize that only the first time step of the control strategy is implemented on the actual system, and not the entire control strategy. A new system measurement is then taken, and the algorithm repeats. Despite the deletion of the unused control inputs, it is important to include them in this process to improve the predictions, as noted in [6]. The above process continues for as long as the given system is to be controlled. A graphical representation of this algorithm is shown in figure 2.2.

The interval of time for which the controller predicts the system behavior is called the *prediction horizon,* and will be denoted by p [6]. As described above, each MPC prediction is the product of some control strategy. The control strategy contains plans for the control of the system out to some time in the future. This length of time for which the control strategy is created is called the *control horizon*, and will be

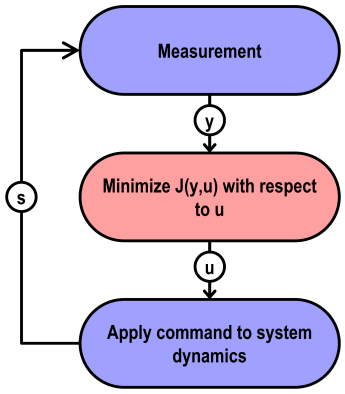

Figure 2.2: MPC algorithm. The measured output of the system is given by y. The variable u represents the control input strategy. Only the first time step of u is applied in the bottom bubble of the flow chart. The state of the system is represented by s. Lastly, $J(y, u)$ is the cost function to be minimized.

referenced with the variable m [6]. If the control horizon is shorter than the prediction horizon, the gap between the application of the last control input and the end of the prediction is filled by repeating the final control input until the end of the prediction, as noted in [4] and [6]. Thorough descriptions of the prediction and control horizons are given in [6].

As described specifically by [6], choosing adequate prediction and control horizons is paramount to creating a controller that effectively satisfies performance requirements. Longer prediction and control horizons greatly increase the likelihood that a more optimal solution will be found, but are also taxing computationally [6]. This is because longer prediction and control horizons require longer simulations to determine the effectiveness of a given control strategy. Furthermore, a longer control horizon increases the number of variables that can be manipulated by the solver. This adds to the complexity of the calculation. Therefore, much computation time is accumulated as longer simulations are performed to determine the optimal control strategy. Conversely, while shorter prediction and control horizons would decrease the computation time, they would also greatly reduce the likelihood that a more optimal solution will be found [6].

The selection of the prediction and control horizons depend largely on the system characteristics, performance requirements, and available computational power. Therefore, there is not a predetermined way to compute the *best* values for p and m for every dynamic system. Because of this, it is up to the user to manually determine the prediction and control horizons for a system. This determination will be examined in chapter 3, following the guidance from [6].

Before concluding the discussion on the basics of the MPC algorithm, it is important to discuss the meaning of the notation, $J(y, u)$, found in the red bubble in figure 2.2. The term $J(y, u)$ represents the cost function for the system. This cost function accepts as arguments the following quantities: the output data from the last measurement (y), and the control command (u)[6]. Since a simulation must be run before the algebraic cost function is used, one could consider this to be an implicit step in the cost function itself. Based on the notation provided by figure 2.2, y is the initial condition and u is the control input. This simulation, which technically occurs *within* J, outputs a vector of data points marking the new system trajectory. This trajectory vector is then transformed into some scalar cost according to regular cost function mechanics.

In the research presented here, a reference value is subtracted from each data point before this resulting vector has some norm operation performed on it[7]. In this work, the costs are always greater than or equal to zero, with zero representing the optimal output.

This structuring emphasizes the importance of choosing proper prediction and control horizons for a given dynamic system. The user must always try to find the right balance so that the control computations are not bloated, while not sacrificing too

[6]Some variables in sections 2.2 and 2.3 have been overloaded in an attempt to respect original notation for both concepts. These parameters will be redefined to mitigate this issue in a later section

[7]The output of simulations in this work is commonly a matrix. Much care is taken to parse the matrix into many vectors that represent a common component of a specific aspect of the system.

much performance. This is especially important as the cost function is repeatedly checked at each time step, leading to more simulations being performed. As alluded to previously, the determination of MPC parameters for case studies I and II, done according to the strategic guidelines offered by [6], will be examined in chapter 3.

2.4 Incorporation of ADMM into MPC

With ADMM and MPC independently defined above, the combination of these two algorithms can now be discussed. The combination of these two algorithms is based on the strategies of [7] and [9]. As alluded to in chapter 1, the intended purpose for this algorithm is to speed up computationally expensive control command calculations for a traffic network. With MPC controlling this traffic network, ADMM subsequently will be used to perform the MPC control command calculation. In the MPC algorithm depicted by figure 2.2, the step between the measurement step (top blue bubble) and control implementation step (bottom blue bubble) is the command calculation step. This step is depicted with a red bubble. It is in this potentially computationally expensive step of the MPC algorithm that ADMM will be implemented.

The adaptation of ADMM for use in the MPC control algorithm will discussed in the remainder of this section. First, in subsection 2.4.1, the notation of scaled ADMM will be adapted to match that of MPC for a general dynamic system. In subsection 2.4.2, the formal mapping of ADMM into the MPC algorithm will be illustrated. The algorithmic structures discussed here will be used in some form throughout much of the rest of this discussion.

2.4.1 Notation Update

In this method, ADMM as defined in subsection 2.2.2 will be implemented in the solve step of MPC. Before this can be done, the ADMM algorithm must first be redefined in terms of a general dynamic system. This has been done in figure 2.3. In this representation of ADMM, the variables with respect to which the cost function

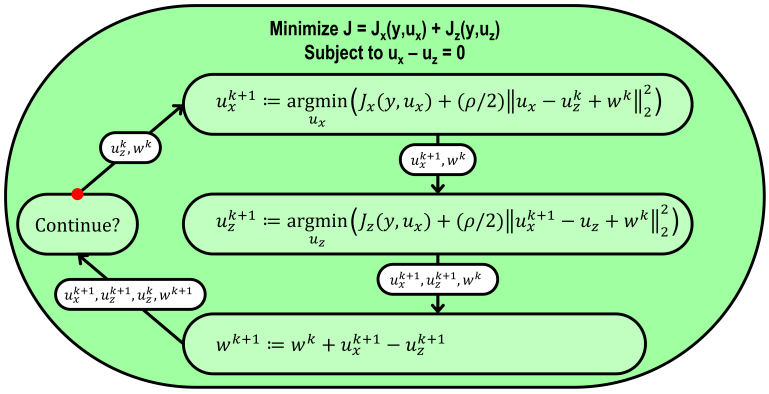

Figure 2.3: Basic ADMM algorithm. The terms u_x and u_z represent control inputs to the dynamic system. The variable w is the *scaled* dual variable. Measurement data is represented by y, and is used as the initial condition for the simulations. The functions $J_x(y, u_x)$ and $J_z(y, u_z)$ each represent one component of the overall cost function. The penalty parameter is ρ.

is minimized are given by u_x and u_z. These variables take the place of the generic x and z used in figure 2.1. The variable names u_x and u_z were chosen to express the idea that the cost function for this dynamic system will be minimized (following the steps outlined in section 2.3) with respect to the command input. Since control inputs are commonly referred to as u, this notation was preserved here. Furthermore, the symbols x and z were chosen as subscripts to preserve the previous notation of splitting the cost function in terms of the variables x and z. The relationship between u_x and u_z will be discussed more explicitly in the analysis of case study I in chapter 3.

The remaining symbols translate more directly from figure 2.1 to figure 2.3. The *scaled* dual variable is represented by w according to figure 2.3. This variable has the same units as u_x and u_z. In this representation, the variable y represents system output data. The cost function J has also been split into $J_x(y, u_x)$ and $J_z(y, u_z)$ as shown in figure 2.3. Here, $J_x(y, u_x)$ and $J_z(y, u_z)$ represent $f(x)$ and $g(z)$ in figure 2.1 respectively. It is important to note that $J_x(y, u_x)$ accepts as arguments the output data y and control command variable u_x. Here again, y is used as the initial condition for the prediction simulations that are embedded in the $J_x(y, u_x)$

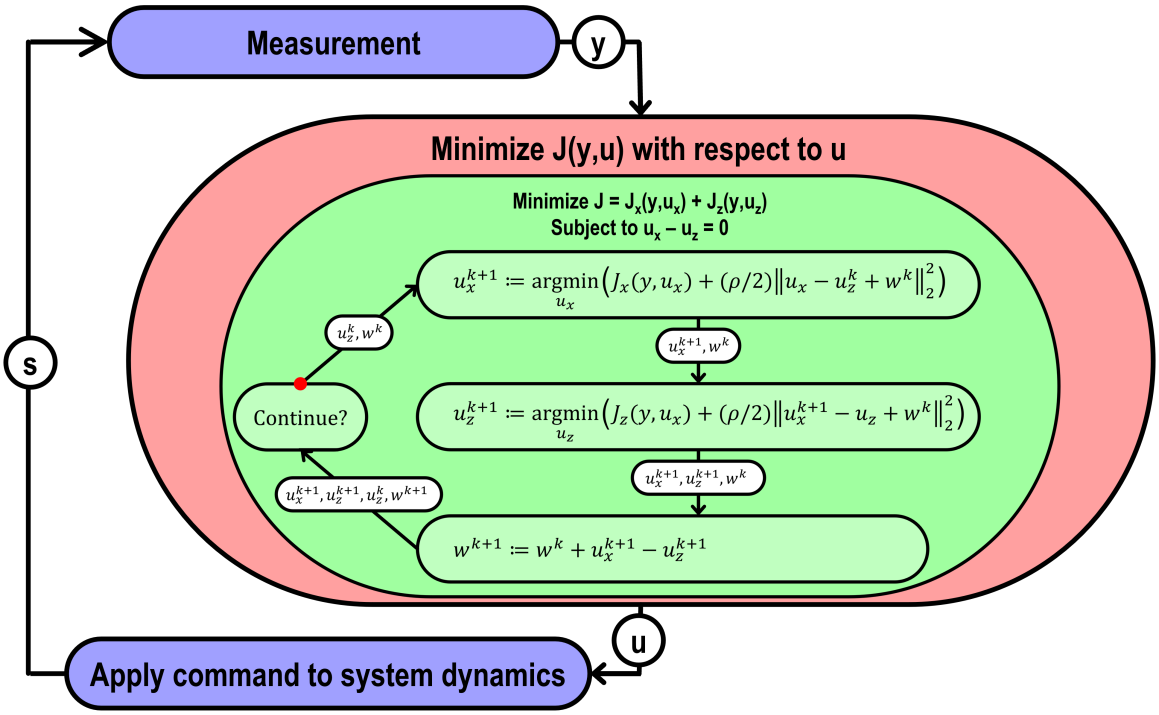

Figure 2.4: MPC-ADMM algorithm. Measured output: y. Control input strategy: u_x and u_z. Only the first time step of u_z is applied in the bottom bubble of the flow chart. System state: s. Cost function(s): $J(y,u)$, $J_x(y,u_x)$, and $J_z(y,u_z)$. Dual variable: w. Penalty parameter: ρ.

minimization step. Analogous statements about $J_z(y,u_z)$ can be made by symmetry. Lastly, it is important to note that that the stopping criterion variables can be directly translated to these variables.

2.4.2 Algorithm Combination

With ADMM in terms compatible with MPC (see figure 2.3), the combination of ADMM with MPC can now be discussed. To combine the use of ADMM with MPC, the entire ADMM algorithm must be placed inside of the solve step of MPC. In other words, to choose a new control command, an entire ADMM algorithm is run out to the stopping criteria. This will occur at each time step. The details of this method are described below. Figure 2.4 provides a visual aid that illustrates this process.

The first and last step of the new algorithm are largely unchanged from the first and last steps of the basic MPC algorithm. Subsequently, the new control algorithm

begins with a system measurement. After the measurement is obtained, the system must determine the most optimal control strategy based on this initial system configuration. If this were regular MPC, several system responses would be simulated to determine which system input is most ideal. This control strategy would be saved for implementation in the next step of the algorithm. However, with the introduction of ADMM, this solve step changes.

After the measurement is obtained, the system will iterate through the ADMM algorithm as defined for the given dynamic system and its cost function. Variable splitting and cost function separation will be illustrated by example in the examination of case study I. Regardless of these specifics, the ADMM algorithm iterates either until the maximum number of ADMM iterations are performed, or until the *calculated* stopping criterion is met.

In the u_x-update step, the scaled ADMM equation for u_x is minimized with respect to u_x, such that u_z^k and w^k are held constant. After the optimal u_x is identified, the algorithm proceeds to the u_z-update step. Similarly, the recently produced u_x^{k+1} variable and the q^k variable will be held constant while a round of minimizations are done with respect to u_z. After the most optimal value has been found, both u_x^{k+1} and u_z^{k+1} will be given to the dual variable update equation. Once the dual variable is updated, the stopping criteria are checked. If the stopping criteria are not met, k is incremented and the cylce repeats. Otherwise, the last copy of u_z^{k+1} will be taken as the most optimal control strategy. From here, the first time step only of command strategy u_z^{k+1} will be implemented on the system. Another measurement will be taken leading up to another repetition of the algorithm. When the algorithm repeats, in the next time step, the u_z^k and w^k variable will be preserved from the previous time step's iterations.

It is important to note that if, for example, a system controlled by MPC is to be simulated for 41 time steps, then there would be 41 separate instances of the ADMM

algorithm. In summary, the new algorithm is to take a measurement of the system, use ADMM to determine the optimal control command, implement the first time step of this optimal control command, and then repeat the process. As noted previously, this method is based on the combination of different forms of ADMM with MPC in [7] and [9]. Case study I explores an example application of this algorithm. As stated previously, the goal for this case study is to produce a faster computation method for MPC within the context of traffic networks. Case study I analysis will help determine whether this algorithm accomplishes this goal.

2.5 Overview of Topics Summary

In this chapter, ADMM and MPC were defined independently. First, it was discussed how ADMM is based on dual decomposition (and by inheritance, dual ascent) and the method of multipliers. The reader is referred to [8] for a thorough discussion of these topics. However, the discussion of ADMM included here provides a connecting point to MPC. Thus, after discussing some specifics of ADMM and its steps, the MPC algorithm was defined. Lastly, it was examined how ADMM and MPC could be woven together conceptually. An example implementation of this ADMM and MPC combination algorithm will be examined in chapter 3. In the following chapter, it will be shown that while this is a good foundation to understand ADMM and MPC, this implementation alone would likely not be sufficient for the purposes required in the context of traffic networks.

CHAPTER 3: CASE STUDY I – BASIC ADMM

Before implementing the ADMM and MPC combination algorithm developed in chapter 2 on a traffic network, this concept will be tested on a predictable dynamic system with much simpler behavior. To that end, a horizontal mass-spring-damper problem was created. In this dynamic system, a wheeled mass is attached to an anchor point with springs and dampers aiding/hampering its movement. An advantage of studying this system is that the system can be made progressively larger and more complex by linking additional wheeled masses together. These additional masses would be attached to each other with springs and dampers in the same way that the first mass is attached to the anchor point. For the sake of simplicity, this case study will consider only a single mass system. The term "cart" will henceforth be used to refer to this concept of a wheeled mass whose motion is affected by springs and dampers. A sketch of the single-cart system is given by figure 3.1.

Carts in this system can apply a torque directly to the wheels. The wheels do not slip, and the input torque will be treated as a direct force along the horizontal axis. The objective of this system is to move the cart a certain distance away from the anchor point, which itself is the origin of the system. The objective of the controller

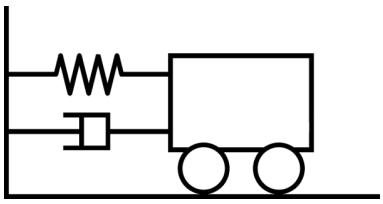

Figure 3.1: Mass-spring-damper system with only a single cart. Used in case study I. The left wall is a fixed anchor point (although no motion restrictions were placed on the system to keep the cart from passing through this location), and there are no restrictions to the right of the cart other than the dynamics of the spring and damper.

is therefore to determine (and apply) the correct input force to make the cart achieve this goal. Additionally, this must be achieved in as little time as possible.

In this chapter, two methods of control on this system will be analyzed. These methods are traditional Model Predictive Control (MPC), and the combination algorithm of MPC with the Alternating Direction Method of Multipliers (ADMM). After defining the equations of motion for the system, the grounds on which MPC parameters are determined will be discussed. Using these parameters, it will then be shown how this system can be controlled with traditional MPC. Lastly, the system will be controlled with the MPC-ADMM combination algorithm. The two performance metrics that will be used in this case study are elapsed time of the computation[1], and the cost function metric of how well each system performed.

3.1 Equations of Motion

In this section, the equations of motion for a system with N carts will be discussed. A diagram of this system is given by figure 3.2. In this implementation, each mass will be labeled with a positive whole number increasing successively when moving away from the origin (from left to right). The equations of motion and state matrices will first be defined for a system with N carts. Using this foundation, the equations of motion and state matrices for the system with one cart will be made explicit ($N = 1$). To do this, the system of N carts must first be put into the form of equations (3.1) and (3.2).

$$\dot{s} = As + Bu \tag{3.1}$$

$$y = Cs \tag{3.2}$$

Equation (3.1) is a general form used to describe dynamic behavior of a dynamic system. Equation (3.2) subsequently describes how the output data is managed by

[1] This will be a *rough* estimate of computational complexity of a given simulation. The justification for this metric will be given in analysis of the case study results.

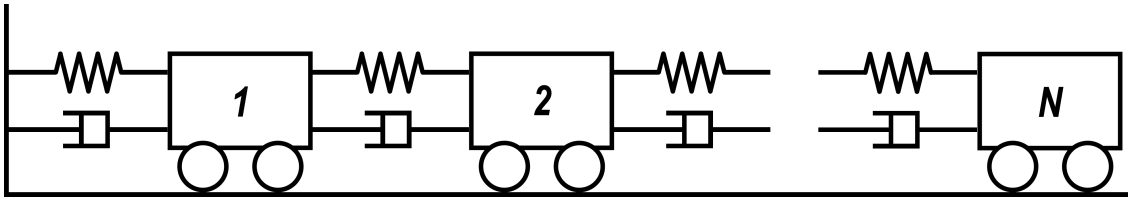

Figure 3.2: Mass-spring-damper system with N carts. The space between the connecting springs and dampers of carts 2 and N represent an arbitrary number of theoretical carts that can be placed in between. The left wall is an anchor point, and cart N has no restraints to its right.

measurement instrumentation. For the linked-cart system described above, the variable s is a $2N$-dimensional vector of potential states in the system. Furthermore, $\dot{s}$ represents the rate of change of these states and has the same size as s. The N-dimensional vector u is the input command vector, and the $2N$-dimensional vector y is the output vector. Lastly, A is a $2N \times 2N$ matrix, B is a $2N \times N$ matrix, and C is the $2N \times 2N$ identity matrix. Together, these matrices will help describe key components of this system.

The parameters that will be substituted in equation (3.1) can be obtained by forming matrices that contain information described by the system equations. This representation is made simpler because each cart in this system has identical mass (m), identical spring constants (k), and identical damper constants (b). Thus $m_\alpha = m_\beta$, $k_\alpha = k_\beta$, and $b_\alpha = b_\beta$ for any integers α and β, such that $1 \leq (\alpha, \beta) \leq N$. These properties greatly reduce the complexity of the equations that govern this system. In the following subsection, the equations of motion that govern this system will be derived.

3.1.1 Algebraic Equations of Motion

In the derivation of the system equations for this system, a pattern arises in the formulation of the equations of motion for each cart in the middle of the system. Because of this, the equations that govern these "middle" carts will first be defined. This will be done by determining the equations of motion for an arbitrary cart α,

where α represents neither the leftmost nor rightmost cart in the system ($\alpha \neq 1, N$; and $N \neq 1, 2$). The springs and dampers will be denoted from left to right, such that cart α has a set of springs and dampers directly to its left labelled α, and a set of springs and dampers directly to its right labelled $\alpha + 1$.

Following a derivation strategy similar to that found in [10], the forces affecting each cart are summed together. These forces are the cart's own input force (F_α), and the spring and damper forces to the left ($F_{k,\alpha}$ and $F_{b,\alpha}$) and to the right ($F_{k,\alpha+1}$ and $F_{b,\alpha+1}$) of the cart. As stated previously, the equal and constant parameters for the masses, springs, and dampers cause the "middle" carts to be governed by a single equation form derived below in (3.3), and (3.4). In these equations below, x is the position of the cart, $\dot{x}$ is the velocity of the cart, and $\ddot{x}$ is the acceleration of the cart.

$$\Sigma F_\alpha = m_\alpha \ddot{x}_\alpha = F_\alpha + F_{k,\alpha} + F_{b,\alpha} + F_{k,\alpha+1} + F_{b,\alpha+1} \tag{3.3}$$

$$F_{k,\alpha} = -k_\alpha(x_\alpha - x_{\alpha-1}) \qquad F_{b,\alpha} = -b_\alpha(\dot{x}_\alpha - \dot{x}_{\alpha-1})$$

$$F_{k,\alpha+1} = -k_{\alpha+1}(x_\alpha - x_{\alpha+1}) \qquad F_{b,\alpha+1} = -b_{\alpha+1}(\dot{x}_\alpha - \dot{x}_{\alpha+1})$$

$$\ddot{x}_\alpha = \big(F_\alpha - k_\alpha(x_\alpha - x_{\alpha-1}) - b_\alpha(\dot{x}_\alpha - \dot{x}_{\alpha-1}) - \cdots$$

$$\cdots - k_{\alpha+1}(x_\alpha - x_{\alpha+1}) - b_{\alpha+1}(\dot{x}_\alpha - \dot{x}_{\alpha+1})\big)/m_\alpha \tag{3.4}$$

The resulting pattern arises from the uniform cascade of constants through each state equation. This pattern will be made clearer in subsections 3.1.2 and 3.1.3, where the equations are more compactly organized.

The system dynamics for cart 1 and cart N follow no such pattern and must have their equations derived separately. The derivation of these equations is shown below, again following the formulation strategies found in [10]. First, the forces that affect each cart are summed together in equations (3.5) and (3.7). The full algebraic definitions of these forces are substituted into these equations, which are then reassembled in equations (3.6) and (3.8) below. The subscript "1" denotes the

first cart in the chain attached to the fixed anchor point and cart 2. The subscript "N" denotes the cart at the far end of the chain that is only attached to its neighboring cart on the left $(N-1)$, and is free of restrictions to the right.

$$\Sigma F_1 = m_1\ddot{x}_1 = F_1 + F_{k,1} + F_{b,1} + F_{k,2} + F_{b,2} \tag{3.5}$$

$$F_{k,1} = -k_1(x_1) \qquad F_{b,1} = -b_1(\dot{x}_1)$$

$$F_{k,2} = -k_2(x_1 - x_2) \qquad F_{b,2} = -b_2(\dot{x}_1 - \dot{x}_2)$$

$$\ddot{x}_1 = \big(F_1 - k_1(x_1) - b_1(\dot{x}_1) - k_2(x_1 - x_2) - b_2(\dot{x}_1 - \dot{x}_2)\big)/m_1 \tag{3.6}$$

$$\Sigma F_N = m_N\ddot{x}_N = F_N + F_{k,N} + F_{b,N} \tag{3.7}$$

$$F_{k,N} = -k_N(x_N - x_{N-1}) \qquad F_{b,N} = -b_N(\dot{x}_N - \dot{x}_{N-1})$$

$$\ddot{x}_N = \big(F_N - k_N(x_N - x_{N-1}) - b_N(\dot{x}_N - \dot{x}_{N-1})\big)/m_N \tag{3.8}$$

With the equations for a system with N number of carts written in this way, they can now be written in state space form. This will be achieved through the manipulation of equations (3.6), (3.4), and (3.8). It is important to note here that equation (3.4) actually represents a total of $N-2$ equations in a system with N carts. Furthermore, if $N = 2$, then only equations (3.6) and (3.8) would be needed to define the system. The case where $N = 1$ will be explored at the end of subsection 3.1.3.

3.1.2 State Space Form

In this subsection, the equations of motion defined in section 3.1.1 for a system containing N carts $(N > 2)$ will be transformed into state space. With some auxiliary updates, the state space form can be represented in one matrix equation like in equation (3.1). The intermediate steps presented in this section will help in achieving this compact formulation which is fully realized in subsection 3.1.3.

Before manipulating the equations of motion from subsection 3.1.1, some preliminary equations must first be introduced. In these equations, differently subscripted

versions of the s variable will be used as the state space variable. This variable will denote various states of the system that were originally denoted by various subscripted x and $\dot{x}$ variables. The derivation of these s terms is shown below.

$$\begin{array}{cccc} s_1 = x_1 & s_2 = x_2 & \cdots & s_N = x_N \\ s_{N+1} = \dot{x}_1 = \dot{s}_1 & s_{N+2} = \dot{x}_2 = \dot{s}_2 & \cdots & s_{2N} = \dot{x}_N = \dot{s}_N \\ \ddot{x}_1 = \dot{s}_{N+1} & \ddot{x}_2 = \dot{s}_{N+2} & \cdots & \ddot{x}_N = \dot{s}_{2N} \end{array} \tag{3.9}$$

Combining this notation with that of the equations of motion from subsection 3.1.1, the state space form can be written as shown below. The primary bases for this representation are equations (3.6), (3.4), and (3.8), combined with the relationships from equation set (3.9). Furthermore, the subscripts on the constants m, k, and b were dropped since all such constants are equivalent for this system. This allows for the combination of like terms which greatly simplifies the state space equations.

$$\dot{s}_1 = s_{N+1} \tag{3.10}$$

$$\dot{s}_2 = s_{N+2} \tag{3.11}$$

$$\vdots$$

$$\dot{s}_{N-1} = s_{2N-1} \tag{3.12}$$

$$\dot{s}_N = s_{2N} \tag{3.13}$$

$$\dot{s}_{N+1} = \big(F_1 - 2ks_1 + ks_2 - 2bs_{N+1} + bs_{N+2})\big)/m \tag{3.14}$$

$$\dot{s}_{N+2} = \big(F_2 + ks_1 - 2ks_2 + ks_3 + bs_{N+1} - 2bs_{N+2} + bs_{N+3})\big)/m \tag{3.15}$$

$$\vdots$$

$$\dot{s}_{2N-1} = \big(F_{N-1} + ks_{N-2} - 2ks_{N-1} + ks_N + bs_{2N-2} - 2bs_{2N-1} + bs_{2N}\big)/m \tag{3.16}$$

$$\dot{s}_{2N} = \big(F_N + ks_{N-1} - ks_N + bs_{N-1} - bs_{2N}\big)/m \tag{3.17}$$

In writing the system this way, all derivative terms from the right hand side of the equations have been eliminated. Eliminating these derivative terms is the primary benefit of state space form because it enables the equations given by (3.10)–(3.17) to be collected in one matrix[2].

3.1.3 Matrix Form

The first step to define the state matrix is to define the state vector, and the rate of change of state vector. This is shown below by equations (3.18) and (3.19) respectively.

$$s = \begin{bmatrix} s_1 & s_2 & \cdots & s_{N-1} & s_N & s_{N+1} & s_{N+2} & \cdots & s_{2N-1} & s_{2N} \end{bmatrix}^\top \tag{3.18}$$

$$\dot{s} = \begin{bmatrix} \dot{s}_1 & \dot{s}_2 & \cdots & \dot{s}_{N-1} & \dot{s}_N & \dot{s}_{N+1} & \dot{s}_{N+2} & \cdots & \dot{s}_{2N-1} & \dot{s}_{2N} \end{bmatrix}^\top \tag{3.19}$$

The state vector given by equation (3.18) and rate of change of the state vector given by equation (3.19) both have dimension $2N \times 1$. The vector of inputs is given below in equation (3.20). This vector has dimension $N \times 1$.

$$u = \begin{bmatrix} u_1 & u_2 & \cdots & u_{N-1} & u_N \end{bmatrix}^\top \tag{3.20}$$

Collectively, these vectors are used as guides in the formation of the matrix equation that governs the system. Using matrix multiplication, a relationship among these vectors is established, as shown by equation (3.21).

$$\dot{s} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{k} & \mathbf{b} \end{bmatrix} s + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} u \tag{3.21}$$

[2]Equation (3.14) corresponds to the leftmost cart whose motion is characterized by equation (3.6). Equations (3.15) and (3.16) correspond to "middle" carts, whose equation was first given by equation (3.4). Lastly, equation (3.17) corresponds to the rightmost cart with motion characterized by equation (3.8).

This equation condenses information from equations (3.10)–(3.17) into one matrix equation. It is important to note that some of the middle rows of the new matrix were extrapolated from the relationships given by equations (3.10)–(3.17). Furthermore, the carts used in both case study I and II will have a mass of $1kg$. For this reason, the mass term is dropped from equation (3.21), noting that dividing by the mass ($1kg$) will have an implicit impact on the units of the system.

In equation (3.21), $\mathbf{0}$ is an $N \times N$ zero matrix, and $\mathbf{I}$ is the $N \times N$ identity matrix. The definition of $\mathbf{k}$ and $\mathbf{b}$ are given below (with the mass term already removed). The $\mathbf{k}$ and $\mathbf{b}$ matrices are also $N \times N$.

$$\mathbf{k} = \begin{bmatrix} -2k & k & 0 & \cdots & 0 & 0 & 0 \\ k & -2k & k & \cdots & 0 & 0 & 0 \\ 0 & k & -2k & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -2k & k & 0 \\ 0 & 0 & 0 & \cdots & k & -2k & k \\ 0 & 0 & 0 & \cdots & 0 & k & -k \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -2b & b & 0 & \cdots & 0 & 0 & 0 \\ b & -2b & b & \cdots & 0 & 0 & 0 \\ 0 & b & -2b & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -2b & b & 0 \\ 0 & 0 & 0 & \cdots & b & -2b & b \\ 0 & 0 & 0 & \cdots & 0 & b & -b \end{bmatrix} \tag{3.22}$$

In equation (3.21) above, the matrix being multiplied by the state vector, s, will be referred to as the state matrix. This term will be denoted by A to resemble the form first given by equation (3.1). The matrix multiplied against the u input vector is called the input matrix, and is denoted by B. Together, these matrices organize the state space equations from (3.10)–(3.17) in a more condensed fashion. In this notation, the $2N \times 2N$ state matrix (A) governs all of the dynamics of the system, while the $2N \times N$ input matrix (B) governs all of the inputs to the system[3].

The matrices given by equation (3.22) explicitly show the underlying pattern in the state space equations. Each row in these matrices corresponds to the dynamic behavior of one cart, and illustrate how the dynamics of each cart depend on the carts directly next to them. Furthermore, these matrices show how the equations of

[3]These inputs were given as torques to the wheels of the carts.

motion for the first cart (represented by the top row) and the last cart (represented by the bottom row) follow a slightly different pattern than those in the middle of the system.

The last component to be defined is the output equation. This equation that governs how the system is measured is straightforward in this system, because each state of the system is known fully and directly. The form of this output equation was originally given above by equation (3.2). Since s has dimension $2N \times 1$, y also has the same dimension. This necessarily means that matrix C is $2N \times 2N$. Since all outputs are known directly, C is the identity matrix.

$$y = \mathbf{I}s \tag{3.23}$$

Equations (3.21) and (3.23), are the foundational equations for this system and will be used throughout the simulations in case studies I and II. Conveniently, the size of this system can be modified according to a choice of N, with the new system representation found by intuitively taking advantage of underlying patterns in the system matrix equations. For case study I, $N = 1$ is chosen as the size of the system[4]. Determining the state equation for the system of size $N = 1$ using equation (3.21) is not quite as clear as it might be for $N > 2$, so the exact equation is given below by equation (3.24). The output equation is shown by equation (3.25).

$$\begin{bmatrix} \dot{s}_1 \\ \dot{s}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k & -b \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_1 \tag{3.24}$$

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \tag{3.25}$$

This interpretation of the $N = 1$ cart system is primarily used in case study I.

[4]Case study II will use $N = 10$. Thus, the explanation for each system was given all at once.

Permanent values for k and b will be explicitly stated in the analysis of this case study. Lastly, when these equations are used to perform simulations in case study I and II, their discrete form will be used. The time step used to discretize these equations will be the same time step parameter used in the MPC controller.

3.1.4 Equations of Motion Summary

In this section, the equations of motion for a system with N carts were defined in a systematic way that took advantage of repetitive system structure. These equations were then put into state space form, which allowed the state equations to be condensed into a single matrix equation (given by equation (3.21)). The output equation was also written in this condensed fashion in equation (3.23). Together, these two equations form the foundation for case study I and II, and the work that follows in the rest of this chapter. Based on equations (3.21) and (3.23), equations (3.24) and (3.25) were constructed. These equations are the specific focus of case study I, where they will be used to make predictions and update the state of the system by discretizing them according to the MPC time step.

3.2 MPC Parameters

Before the full analysis of the MPC-only algorithm and the MPC-ADMM combination algorithm, the MPC parameters must be defined. The overall purpose of this case study is to compare advantages and disadvantages of the two control algorithms. Therefore, it is important to choose reasonably competitive MPC parameters to eliminate bias toward or against a particular control algorithm. While each control algorithm will use the same MPC parameters, the parameters themselves should be chosen carefully to avoid exaggerating different aspects of each algorithm.

In this section, the determination of MPC parameters will be discussed following the guidance of [6]. These parameters will be determined by analyzing select simulations based on the discretization of equations (3.24) and (3.25). The system used for these

simulations is the single cart system, where the objective is to move the cart some given distance away from the origin. For this system, $m = 1kg$, $k = 20N/m$, and $b = 1N/(m/s)$.

In subsection 3.2.1, the sampling time will be determined. This will allow for the determination of the prediction horizon in subsection 3.2.2, which in turn is the basis for the selection of the control horizon in subsection 3.2.3. All of these parameters will then be used in the full system analysis later in this chapter.

3.2.1 Sampling Time

The first parameter that must be defined is the sampling time, dt. The sampling time describes how often the MPC controller reassesses the system and determines a new control strategy [6]. As suggested by [6], this parameter will be based on the open loop rise time of the single cart system, which is defined as the time it takes for the open loop response to go from 10% to 90% of the steady state response [6]. No control is applied to the system in this open loop response. The sample time selection guidance offered by [6] is to choose a sampling time such that $\frac{T_r}{20} \leq dt \leq \frac{T_r}{10}$, where T_r is the rise time.

An open loop simulation on the single cart system was run, with the cart starting at a position $0.1m$ away from the origin with a velocity of $0m/s$. Since there is no input to the system, the steady state response of the cart will clearly be $0m$. In terms of the system model, the cart will be pulled back to the origin and stay there. Thus, the rise time describes the amount of time it takes for the system to go from a response of $0.9m$ (10% response) to $0.1m$ (90% response). Figure 3.3 illustrates the system's response.

For this simulation, a discrete time step of $0.0001s$ was used to ensure precise measurement. Based on this simulation, the elapsed time for the system to achieve 10% response is $t = 0.1027s$, and the 90% response time is $4.3753s$. Thus, the rise time is, $T_r = 4.3753s - 0.1027s = 4.2726s$. Following the guidance of [6], the sample

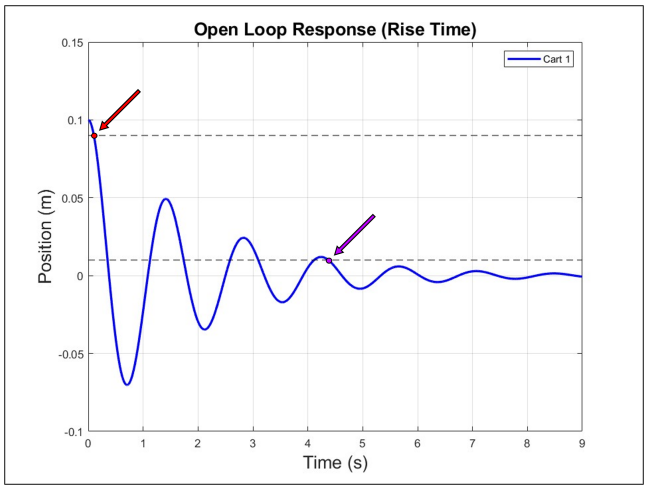

Figure 3.3: Open loop response for one cart. The red arrow indicates when the system achieves 10% of steady state response (defined by the top dashed line), and the purple arrow indicates when the system achieves 90% of steady state response (the bottom dashed line). No input was given to this system.

time should fit within $\frac{T_r}{20} \leq dt \leq \frac{T_r}{10}$, as shown below by equation (3.26).

$$\frac{4.2726s}{20} = 0.21363s, \qquad \frac{4.2726s}{10} = 0.42726s \qquad \Longrightarrow \qquad 0.21363s \leq dt \leq 0.42726s \tag{3.26}$$

Based on the recommendation of equation (3.26), the sampling time was chosen to be $dt = 0.3s$. This choice of sample time indicates that the MPC controller resets and cycles through its algorithm every $0.3s$. Or, as described by figure 2.2, a complete cycle through the flow map occurs every $0.3s$. This continues for the length of time that the system is to be controlled. The system will also be discretized according to this time step. With the sampling time established, the prediction horizon can now be determined.

3.2.2 Prediction Horizon

The prediction horizon, p, sets the length of the test simulations used to determine the optimal control strategy [6]. Like the determination of the dt parameter, the prediction horizon is also based on the open loop response of the system, but with the focus placed on the settling time instead of the rise time [6]. Settling time is

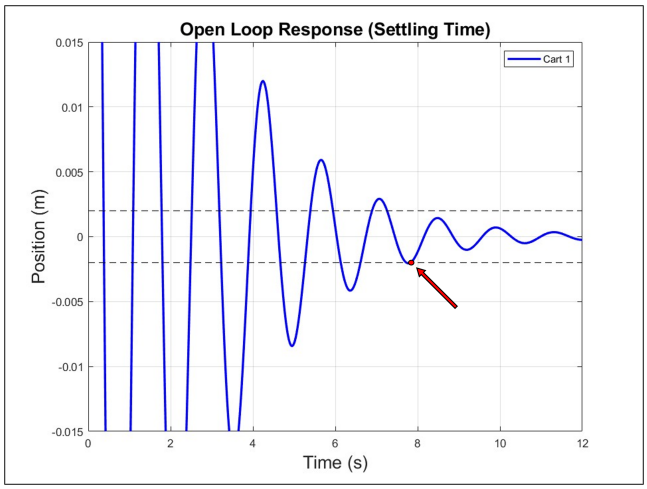

Figure 3.4: Open loop response for one cart. The red arrow indicates where the system achieves 102% of steady state response time for the last time (bottom dashed line). The top dashed line indicates 98% response. Since the system exceeds the $\pm 0.002m$ margin after it last achieves 98% response, the final time the system achieves 102% response time is used instead.

defined by [6] as the time where the open loop response is within 2% of the steady state response. The prediction horizon, as suggested by [6], should be selected such that $p \cdot dt \geq T_{settling}$ (or $p \geq T_{settling}/dt$).

The same simulation parameters used in the first simulation were used again in the determination of the settling time. This preserved the initial conditions and the steady state response, which is still $0m$. Being within 2% of the steady state response will be calculated as when the system achieves either 98%/102% of the steady state response. Based on the initial position of $0.1m$, the 98% or 102% system response will be achieved when the system is at $\pm 0.002m$. Figure 3.4 shows the open loop response of this system.

A time step of $0.0001s$ was used again in this simulation. Based on this simulation, the settling time is $T_{settling} = 7.8253s$. As [6] suggests, the prediction horizon is thus chosen such that $p \geq T_{settling}/dt$, as shown below by equation (3.27).

$$p \geq 7.8253s/0.3s \quad \Longrightarrow \quad p \geq 26.0843 \tag{3.27}$$

The prediction horizon was thus chosen to be $p = 27$. This means that at each MPC decision step, represented by the red bubble in figure 2.2, the test simulations that this controller uses have a duration of 27 time steps into the future ($p \cdot dt = 27 \cdot 0.3 = 8.1s$). This prediction horizon can now be used to determine the control horizon.

3.2.3 Control Horizon

The control horizon, m, sets the number of time steps for which the MPC controller must plan a control strategy [6]. The process of choosing a control horizon is much simpler than choosing the sampling time or prediction horizon. However, this choice is predicated off of the user's choice of prediction horizon, which subsequently depended on a good choice of sampling time. The control horizon is suggested by [6] to be chosen such that $0.1 \cdot p \leq m \leq 0.2 \cdot p$. The process of setting these boundaries is shown below in equation (3.28).

$$0.1 \cdot 27 \leq m \leq 0.2 \cdot 27 \quad \Longrightarrow \quad 2.7 \leq m \leq 5.4 \tag{3.28}$$

Based on this recommendation, the control horizon was chosen to be $m = 4$. Using these parameters case study I can now be analyzed in more detail. First, the MPC-only algorithm will be tested. Then the MPC-ADMM combination algorithm will be tested, allowing for the comparison of the two control strategies.

3.3 MPC-Only Algorithm

The purpose of this case study is to establish the connection between MPC and ADMM. To that end, a simulation using only MPC must first be analyzed to give a baseline of comparison. In this section, the single cart system will be controlled using a traditional MPC approach. The discrete simulation strategy employed here is the same one that was used in section 3.2 to determine the MPC parameters. The MPC parameters that were determined in section 3.2 will be used here as well ($dt = 0.3s$, $p = 27$, $m = 4$). The MPC algorithm process first introduced in section 2.3 will be

followed here. This algorithm was illustrated in figure 2.2.

In this section, the cost function that governs this system will first be analyzed in subsection 3.3.1. This will be followed by analysis of the MPC-controlled system in subsection 3.3.2. Collectively, this sets up part of the comparison with the MPC-ADMM combination algorithm which will be tested later in this chapter.

3.3.1 MPC-Only Algorithm: Cost Function

The cost function used in this simulation will be referred to as the lasso cost function, as it was in [9] (see equation (3.29) below). This cost function was selected for this analysis because it has only two terms and can help the system achieve the reference. The fact that this cost function has two terms is most relevant in the problem formulation and analysis in section 3.4. This style of cost function was also used in the analysis of an MPC-ADMM algorithm in [9]. This section is only concerned with MPC, so it is not directly related to the application in [9]. However, the problem formulation and work in [9] mirror some aspects of the MPC structure used here.

In section 2.3, the cost function was generally described from a programming and logical perspective so that the mechanics of the MPC algorithm could be more fully understood. In this section, only the algebraic cost function mechanics will be examined. This examination will primarily be based on how system data can be transformed into a meaningful scalar value. In the cost function given by equation (3.29) below, the y term in the cost function is understood to implicitly be the relevant output data of the prediction simulation, which itself accepted as arguments a control input u, and initial system configuration. This simplified cost function is given below by equation (3.29), with a description in the following paragraphs.

$$J(y) = \|y - \text{ref}\,\|_2^2 + \|y - \text{ref}\,\|_1 \tag{3.29}$$

This cost function (3.29) accepts the trajectory vector y as an input, and outputs a scalar as the cost associated with that trajectory. The y vector describes the predicted trajectory of the system as it progresses through time. There is one entry for each time step in the simulation that determines y. Therefore, each entry in y is a measurement of the system trajectory after each dt increment of time. Furthermore, the total number of entries is equal to one more entry than the prediction horizon ($27 + 1 = 28$ entries)[5]. The "added time step" is included because the first entry of the trajectory vector corresponds to the initial configuration of the system.

The output trajectory vector y then has a uniform vector of reference trajectories (ref) subtracted from it. In this work, the reference vector is a vector whose entries are all equal to one desired output value. Thus, the entry-wise difference between the output vector and reference vector provides a rough understanding of the discrepancy between the actual system output and the desired response. To more precisely quantify system performance, the norm of this discrepancy vector can be taken to transform this data into a non-negative scalar which concisely contains the collective discrepancy information.

The lasso cost function described by equation (3.29) sums the $\mathcal{L}_2$-norm squared of the difference between the output vector and reference vector with the $\mathcal{L}_1$-norm of the same quantity. Under the right circumstances, it is certainly possible for a given system to achieve some desired result using just one of these terms in the cost function(i.e., $J(y) = \|y - \text{ref}\,\|_2^2$ OR $J(y) = \|y - \text{ref}\,\|_1$). As described in [11], a cost function that uses the $\mathcal{L}_2$-norm gives larger discrepancies much more weight than smaller ones. This means that a system following a reference using the $\mathcal{L}_2$-norm can relatively easily be brought into the neighborhood of a solution. Conversely, [11] also indicates that a cost function that uses the $\mathcal{L}_1$-norm weights smaller discrepancies more than larger ones. Using this norm only would allow a system that is already

[5]When the cost function is applied to the system at the end of the simulation, this equation shifts to focus on each step of the simulation time, rather than on the prediction time.

close to the reference to achieve a more precise following of the reference. Though it could be possible under the right conditions for the reference to be achieved using either the $\mathcal{L}_2$-norm or the $\mathcal{L}_1$-norm, using both in one cost function takes advantage of the weight distributions of each norm[6].

As alluded to previously, the main reason this function was chosen was for the sake of relatively straightforward ADMM implementation. This implementation will be examined specifically later in this chapter. In the next subsection, the results of the MPC-only algorithm for a single-cart system will be presented.

3.3.2 MPC-Only Algorithm: Simulation and Analysis

In this subsection, the results of an MPC-only control algorithm on a single cart system (figure 3.1) using the lasso cost function (equation (3.29)) will be examined. The virtual cart in this simulation used an initial position $0.1m$ away from the origin in the positive direction and a velocity of $0m/s$. The reference that the cart should follow is $1m$ away from the origin. The MPC parameters used in this system were $dt = 0.3s$, $p = 27$, and $m = 4$. Because $dt = 0.3s$, the discrete version of equations (3.24) and (3.25) that are used in the simulation were determined according to $dt = 0.3s$. Since the cost function will accept only the position data vector from a simulation, the velocity has no impact on the cost. Because the output vector contains the position data measured every dt seconds, the cost of an output position is only calculated every dt seconds. This will be discussed further in the following paragraphs, after initial analysis of the system performance.

Figure 3.5 displays the position data from the MPC-only system simulation. This plot shows that the cart reasonably achieves the desired output reference which is denoted by the blue dashed line. After about $t = 2s$, the cart stays close to the reference. The cost that describes the performance of the system is given as $J =$

[6]It should also be noted that each norm term in equation (3.29) could be weighted according to problem type. In the work presented here, both terms are weighted equally.

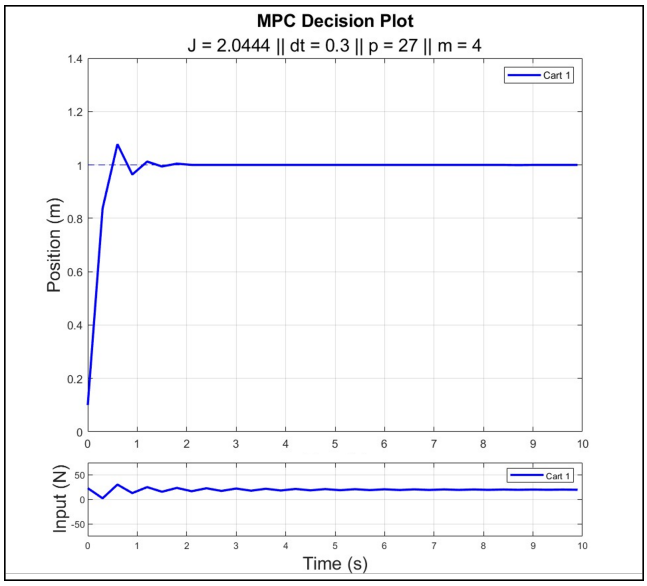

Figure 3.5: MPC Decision Plot. This is discrete data, so the connecting lines between points do not represent output values. The intersection/vertex of two lines represents a moment where actual data is measured, making some data points hard to distinguish. This plot style is more beneficial with a smaller time step. The reference for this data is noted by the dashed line at $1m$.

2.0444. This number means little without another with which to compare it, but a similar simulation will be performed in section 3.4 which will give another cost for comparison.

The jagged nature of the plot highlights the large time step of $0.3s$. In this discrete simulation, a new round of MPC optimizations were performed after each interval of $0.3s$. Because the MPC optimizations are done discretely and only updates the system configuration every $0.3s$, the cost function only has output data for every $0.3s$. In other words, the cost function is only able to *"see"* the system for an instant of time every $0.3s$. Furthermore, as figure 3.5 does not show the precise dynamics that occur between $t = 0.00...01s$ and $t = 0.29...9s$, neither does the cost function include the data from this interval in its calculation. This means that the cost of $J = 2.0444$ is obtained only from the precise data points indicated at every $0.3s$ on the plot.

Due to this discrete *vision* of the system, the plot in figure 3.5 is called the "MPC Decision Plot," because it only shows the configurations from which MPC based its

decision. A simulation with a smaller time step would show that the system response is much smoother and oscillatory. Also, after each MPC controller decision, the selected input is held constant for the duration of the time step until the next MPC decision is made. This characteristic is hidden by the given input plot in figure 3.5. A more detailed simulation was run to allow for more in-depth examination of system characteristics.

This second simulation preserves the MPC decisions from the previous time step and runs the discrete simulation with this same input strategy. The key distinction to be made about this simulation is that it uses a much smaller time step, $dt = 0.01s$. Therefore, to achieve the same duration used in figure 3.5 based on the shorter dt time interval, the new simulation must be performed for many more total time steps. This creates a problem because the vector of input commands used to generate the decision plot is much shorter than the number of entries in the input vector for this new simulation. Therefore, a new vector with many more inputs was created so that it mimicked the original step inputs from the decision plot. Since each input vector in the decision plot is held constant for the larger time step, the new input vector maintains this characteristic as well. The step-like result of this new vector is represented in input plot of figure 3.6.

Figure 3.6 shows the full results of this simulation. The data points that this simulation shares in common with the data in figure 3.5 are very close together in value. A possible reason for any variation could be due to rounding because many more updates are performed in the new simulation compared to the previous one, leading to opportunities for such an error. Figure 3.6 is included in this discussion to show a more detailed view of what is happening in the system than what the decision plot in figure 3.5 shows. This plot is labelled as the "MPC Simulation Plot" because of this characteristic.

The cost from feeding the entire position data vector from this simulation to the

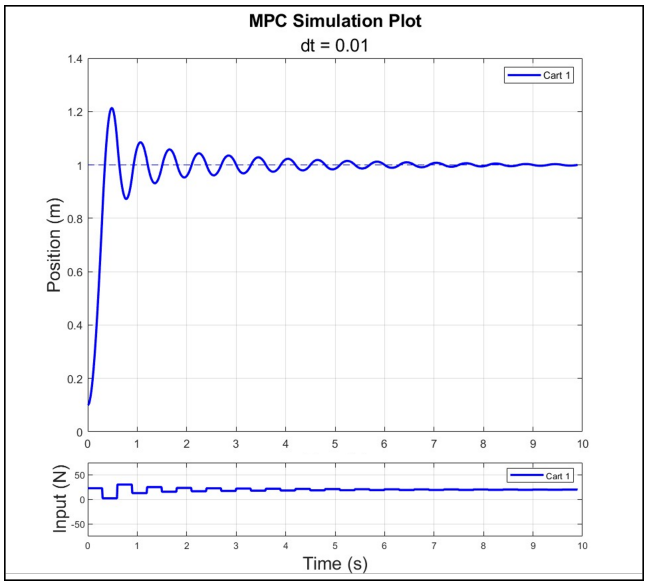

Figure 3.6: MPC Simulation Plot. This is discrete data. Because the time step is much smaller relative to the data represented by the decision plot, this plot gives a much more detailed view of the behavior of the system. The reference is still given by the dashed line at $1m$.

cost function is not included here because it could distort the characterization of the algorithm performance. One possible distortion is that giving many more data points to the cost function could result in an unfairly higher cost. Another could be that a specific control algorithm might make the system achieve the reference target by making the simulation oscillate back and forth, intersecting the reference at the instant that position data is measured. This would give a good cost according to the longer time step, but if the full data from a shorter time step simulation can be compared to the reference, the new cost would likely be higher. Furthermore, because the control algorithm was not basing its decision on the shorter time step, it is unreasonable to judge its performance as if it had.

The last simulation metric to discuss is the computation time. It took approximately $100s$ to run this MPC simulation. As with the cost metric, this metric needs to have some other simulation to compare against, which will be given explicitly in section 3.4. This metric is meant to serve only as an idea of the general computational

complexity/length of the algorithm, so it is reasonable to use only an approximation of the total time. The second reason a time approximation is used is the overall difficulty of running the simulation in complete isolation. This is because there are several variables that could affect the performance of a computer that are hard to quantify and reset before each iteration. Furthermore, the exact same simulation run under the same setup can take a slightly shorter or longer amount of time to run. Therefore, an approximation is used as the final time metric for the simulations in this research simply to give an idea of the level of computational complexity of a simulation. This additionally means that any difference between simulation times must be *substantial* to consider one simulation more or less computationally complex than another.

Viewed collectively, both the decision plot (figure 3.5) and the simulation plot (figure 3.6) characterize the performance of this system under the control of only the MPC algorithm. In the following section, the system will be controlled with the MPC-ADMM combination algorithm. This will enable a comparison between the two strategies that will provide insight on how ADMM can be implemented in the context of a traffic control optimization problem.

3.4 MPC-ADMM Combination Algorithm

With an understanding of the performance of the MPC-only algorithm, the MPC-ADMM combination algorithm can now be analyzed. The MPC-ADMM method formulation described in section 2.4 is used here in the simulation of the system. In this section, the cost function transformation used in ADMM will be discussed in detail in subsection 3.4.1. Then, the simulation itself will be analyzed in subsection 3.4.2. This will set up for a discussion later in this chapter analyzing both control strategies.

3.4.1 MPC-ADMM Combination Algorithm: Cost Function

In this section, the foundational cost function given by equation (3.29) will be transformed so that it can be used in the ADMM algorithm described by [8] for the lasso cost function. In [9], an MPC-ADMM combination algorithm is formulated that is similar to the one used here. The work of [9] provides a good outline for the derivation that follows. The algorithm used here was first given by figure 2.3, and is given below by equations (3.30)–(3.33). This follows the scaled ADMM algorithm from subsection 2.2.2.

$$\text{Minimize}: \ J_x(y,u_x) + J_z(y,u_z) \qquad \text{Subject to}: \ u_x - u_z = 0 \tag{3.30}$$

$$\text{Algorithm}: \ u_x^{k+1} := \underset{x}{\operatorname{argmin}} \ J_x(y,u_x) + (\rho/2)\|u_x - u_z^k + w^k\|_2^2 \tag{3.31}$$

$$u_z^{k+1} := \underset{z}{\operatorname{argmin}} \ J_z(y,u_z) + (\rho/2)\|u_x^{k+1} - u_z + w^k\|_2^2 \tag{3.32}$$

$$w^{k+1} := w^k + u_x^{k+1} - u_z^{k+1} \tag{3.33}$$

For ADMM to be incorporated into this system analysis, a few adjustments must be made to the current MPC algorithm, as first outlined in section 2.4. First, the cost function given by equation (3.29) must be separated. Following the strategy from section 2.4.1, the $J_x(y,u_x)$ cost function will be defined such that $J_x(y,u_x) = \|y - \text{ref}\,\|_2^2$, where y is implicitly the position output of a discrete simulation which accepted u_x as an argument. Furthermore, the $J_z(y,u_z)$ cost function will be defined such that $J_z(y,u_z) = \|y - \text{ref}\,\|_1$, where y is implicitly the position output of a discrete simulation which accepted u_z as an argument. In both of these modifications, u_x and u_z are command vectors given to a discrete "simulation function" whose output is y, and is not explicitly listed in these representations of the cost function. The variables u_x and u_z are therefore equal as they are both commands given to the system ($u_x = u_z \implies u_x - u_z = 0$).

With $J_x(y, u_x)$ and $J_z(y, u_z)$ defined, the u_x and u_z minimization steps of ADMM can now be discussed. These steps minimize $J_x(y, u_x)$ and $J_z(y, u_z)$ respectively, where each J function has a term added to it. Using a non-changing ρ value of 10, the scaled version of ADMM will be implemented (refer to subsection 2.2.2). This means that the added term to the u_x minimization will be $(\rho/2)\|u_x - u_z^k + w^k\|_2^2$, where w^k represents the dual variable from the previous iteration. The added term to the u_z minimization is very similar. Upon further examination, it becomes clear that there is potentially a substantial difference in magnitude between these two terms combined in the u_x minimization step (and similarly in the u_z minimization step). This is expanded upon below.

The inputs given to this system are added or subtracted forces[7]. The input force is bounded such that it does not exceed $\pm 75N$. These boundaries were never close to being exceeded in the MPC-only simulation, but they do give a sense of the scale of the vectors in the added term. Conversely, in the original $J_x(y, u_x)$ term, the input is a position, which itself has a range of magnitudes of about ± 1 m. This potentially substantial discrepancy in magnitude could place more weight than is feasible on one term. Therefore, a new constant must be introduced.

The new constant is meant to bring the magnitude of each J term closer to the magnitude of the extra terms in the u_x and u_z minimization equations. Therefore, each J term is multiplied by a constant of 10000. It should be noted that choosing this type of constant for the J terms has a similar effect as choosing a smaller ρ parameter. However, to keep the identity of ρ fixed, this method of weighting the cost function was used[8]. The full form of scaled ADMM that will be used in this MPC-ADMM combination algorithm is shown below by equations (3.34)–(3.36), and

[7] These "forces" are technically torques applied to the wheels. However, since the wheels do not slip, it is simpler to consider these inputs to be forces. One can think of these forces as pushing the cart in one direction or another.

[8] A simple way of updating the ρ parameter is offered in [8], but is not used here for the sake of simplicity.

mirrors the algorithms in figures 2.3 and 2.4.

$$\text{Minimize}: \; J_x(y, u_x) + J_z(y, u_z) \qquad \text{Subject to}: \; u_x - u_z = 0$$

$$\text{Algorithm}: \; u_x^{k+1} := \underset{x}{\operatorname{argmin}} \; 10000\|y - \text{ref}\,\|_2^2 + (10/2)\|u_x - u_z^k + w^k\|_2^2 \tag{3.34}$$

$$u_z^{k+1} := \underset{z}{\operatorname{argmin}} \; 10000\|y - \text{ref}\,\|_1 + (10/2)\|u_x^{k+1} - u_z + w^k\|_2^2 \tag{3.35}$$

$$w^{k+1} := w^k + u_x^{k+1} - u_z^{k+1} \tag{3.36}$$

Lastly, although omitted from this description, the stopping criteria check follows the rules described in section 2.2.3. The following updates can be made to equations (2.19)–(2.23) which govern the *calculated* stopping criterion. The A matrix can be replaced with the identity matrix, and the B matrix can be replaced with the negative identity matrix. The c constant is replaced with 0. Lastly, the P and N constants represent the prediction horizon, plus one. This is the size of the output vector from each iteration.

Using this algorithm, new simulations were run. The goal of these simulations was to find a solution that is faster to compute than the MPC-only algorithm, and relatively close to the performance cost of the MPC-only algorithm.

3.4.2 MPC-ADMM Combination Algorithm: Simulation and Analysis

In this subsection, the single-cart simulation using the MPC-ADMM algorithm will be analyzed. This simulation will be run under parameters very similar to those used in section 3.3, with some added terms that can vary. The MPC parameters ($dt = 0.3s$, $p = 27$, $m = 4$) are preserved, as are the initial conditions of the system (position: 0.1 m; velocity: 0 m/s). The cost function has the same form, but with the changes discussed in subsection 3.4.1. However, there are some additional parameters associated with ADMM that must also be set.

As discussed previously, the ρ penalty parameter was set to always be constant and

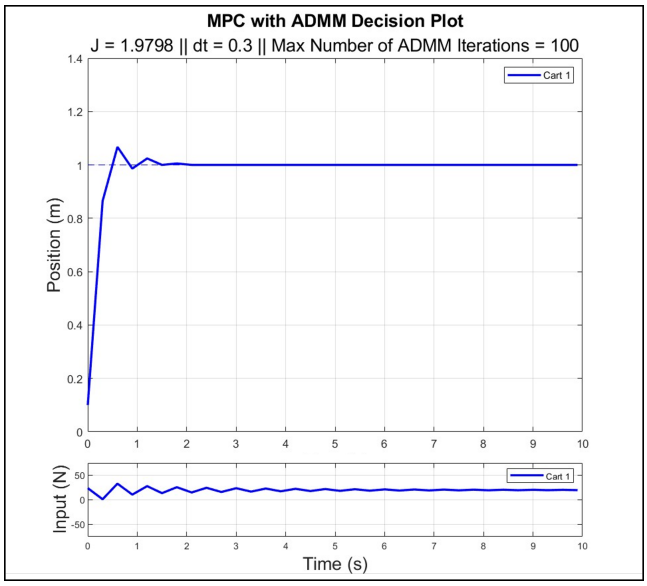

Figure 3.7: MPC with ADMM Decision Plot. The maximum number of ADMM iterations for this simulation was 100.

equal to 10. Furthermore, the maximum number of ADMM iterations allowed by the system will be set to 100. It is important to note that not every single iteration of ADMM must be used each time. The point where the ADMM algorithm ends, if not due to reaching the maximum number of ADMM iterations, is subject to the *calculated* stopping criterion. This stopping criterion will be checked according to subsection 2.2.3, using an absolute tolerance of 0.0001, and a relative tolerance of 0.01. These parameters were based off of parameters from additional material provided with [8], in which several test cases of ADMM are run[9]. The results from this simulation are given in figure 3.7 for the $dt = 0.3s$ time step.

As this figure shows, the cost of this simulation and control strategy is $J = 1.9798$, which is very close to the cost from the MPC-only algorithm in subsection 3.3.2 (which was $J = 2.0444$). It is important to note that this final cost ($J = 1.9798$) was calculated according to the same cost function as the MPC-only algorithm, instead of the ADMM style of cost function, which was only used to decide on the control

[9]These additional materials applied ADMM in a much different context than how it is applied to a dynamic system here.

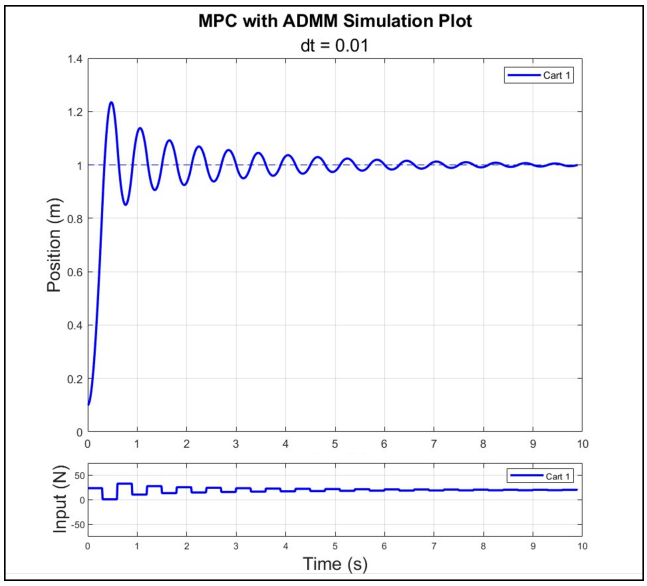

Figure 3.8: MPC with ADMM Simulation Plot. This is a more detailed representation of what is happening within the decision plot for this MPC-ADMM simulation.

strategy. Thus, the basic lasso cost function given by equation (3.29) was used to calculate the total cost for this system. Like the MPC-only algorithm, this cost is calculated based only on the data from the decision plot (with time step $dt = 0.3s$).

Despite the competitive cost, the approximate time of computation was $500s$, which is much higher than it was in the MPC-only simulation ($100s$). This shows that the current form of the MPC-ADMM combination algorithm is likely not desirable for the traffic network dynamics because the computation time is significantly longer. Furthermore, even if the maximum number of ADMM iterations allowed at each time step is reduced, the performance is still not desirable. The specifics of these other results will be analyzed in subsection 3.5. The simulation plot that uses a smaller time step is given by figure 3.8. It is important to note that the observations made in subsection 3.3.2 regarding the discrete characteristics of the MPC-only algorithm plots and data can be made here as well.

For this simulation, the ADMM parameters u_z and w were not reset between time steps. Many of the final input commands for this system end up being repeated. Because of this, not clearing the u_z^k and w^k values at each time step may help the

Table 3.1: Data for the MPC-only algorithm and the MPC-ADMM combination algorithm. This data is for a single-cart system.

Simulation	MPC-Only	MPC-ADMM Combination	MPC-ADMM Combination	MPC-ADMM Combination
Maximum Number of ADMM Iterations	N/A	1	2	100
Cost	$J = 2.0444$	$J = 2.2111$	$J = 2.0917$	$J = 1.9798$
Approximate Time	$100s$	$140s$	$285s$	$500s$

ADMM algorithm start closer to a reasonable answer. This would be due to the influence exercised by the persisting choices of u_z and w.

Additional simulations comparing different allowable numbers of ADMM iterations were also performed using identical conditions otherwise. The results of these simulations are given in table 3.1. These results are thematically the same compared to the plots presented in this subsection. The cost of each simulation was competitive, but not so much as to have drastically outperformed the MPC-only algorithm. Furthermore, the approximate computations times indicated that selecting fewer maximum allowable ADMM iterations was still slower than only using the MPC-only algorithm. These results will be discussed in detail in section 3.5 in the context of the other simulations.

3.5 Case Study I Summary

As sections 3.3 and 3.4 show, the current form of the MPC-ADMM combination algorithm does not outperform the MPC-only algorithm. Additionally, limiting the maximum allowable ADMM iterations for each MPC optimization step did not provide overwhelmingly positive results. As table 3.1 indicates, the best performing simulation from a cost perspective was the MPC-ADMM combination algorithm with the maximum allowable ADMM iterations set to 100. However, this performance came with a substantial computation time. The MPC-only algorithm outperformed

all of the other MPC-ADMM combination simulations in terms of performance, and outperformed *all* of the MPC-ADMM combination algorithms in terms of time.

Perhaps the reasoning for this is the total number of minimizations performed at each time step. For one time step of the MPC-only algorithm, there is one set of minimizations. However, for the MPC-ADMM combination algorithm, there are *at least* two sets of minimizations per time step (u_x and u_z). Although these minimizations are aided by the added terms that can bring them to a solution faster, the overhead time-loss does not seem to be overcome. Therefore, the MPC-ADMM algorithm described in section 2.4 and used in the simulations in section 3.4 essentially increases the computational load without speeding up the overall computation time. While there could be other factors that might contribute to these shortcomings, the data in table 3.1 bluntly indicates that no matter the reason, a new strategy is needed. Chapter 4 will outline a new approach that builds off of the ADMM formulation discussed in this chapter.

CHAPTER 4: CASE STUDY II – CONSENSUS ADMM

In this chapter, a revised method for controlling the mass-spring-damper system will be explored. This new method will combine the consensus form of the Alternating Direction Method of Multipliers (ADMM) with Model Predictive Control (MPC), following many of the underlying principles explored in chapter 3. It will be shown that consensus ADMM leads to faster computation times in this case study.

As the single-cart system has nearly been exhausted in terms of analysis and experimentation, an expanded system is needed. The model that this chapter will examine is a 10-cart system based on the formulation described from section 3.1. The 10-cart system will showcase the strengths of consensus ADMM and illustrate its potential application to traffic networks. As a traffic network problem has sequentially interdependent agents that each have their own dynamics, so also do multiple carts chained together by springs and dampers. Therefore, developing this method on a 10-agent interconnected system will mirror potential applications to traffic networks.

This chapter will specifically examine the combination of *consensus* ADMM with MPC. Since the equations of motion for an N-cart system have already been defined in section 3.1, the equations that govern the 10-cart system will only be briefly discussed in section 4.1. Then, the pure consensus ADMM algorithm will be discussed in section 4.2. The full implementation of consensus ADMM into an MPC problem will then be discussed in section 4.3. Section 4.4 will then establish a baseline for comparison by simulating MPC control of the 10-cart system. In section 4.5, the performance of the MPC and consensus ADMM algorithm will be assessed. Lastly, section 4.6 will discuss the results of sections 4.4 and 4.5, and set up for the application of this method on traffic networks.

4.1 Problem Formulation and Equations of Motion

In this section, the equations of motion that govern this system will be briefly discussed. As shown in equation (3.21), the state matrix equation for this system changes as the number of carts in the system increases. In this chapter, a 10-cart system will be used. Therefore, the position and velocity vectors are both 10×1. The combined state vector variable and rate of change of this variable are 20×1 (s and $\dot{s}$ respectively in equation (3.21)). Furthermore, the $\mathbf{0}$, $\mathbf{I}$, $\mathbf{k}$, and $\mathbf{b}$ matrices are all 10×10. Equation (3.21) is given again below for convenience.

$$\dot{s} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{k} & \mathbf{b} \end{bmatrix} s + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} u$$

The form of the cost function that this system will use is the sum of $\mathcal{L}_2$-norms squared of each cart's position vector. This means that for the 10-cart system, there are 10 terms being summed together in the basic form of the cost function[1]. A condensed version of the cost function is shown below by equation (4.1). This simple style of cost function was chosen so that consensus ADMM could be applied in a relatively straightforward manner.

$$J(y) = \|y_1 - \text{ref}_1\|_2^2 + \|y_2 - \text{ref}_2\|_2^2 + \cdots + \|y_{10} - \text{ref}_{10}\|_2^2 \tag{4.1}$$

It is important to note that, as illustrated by equation (4.1) above, each cart is following its own constant reference trajectory. This means that each cart in the system is seeking to move to a unique position away from the origin. One approach to setting up the system in this way would be to set the initial position of each cart as (cart number)·(some constant distance), and set the given reference for each cart

[1]Recall that the y vector in the cost function is implicitly the result of some simulation based on the command input, u, and initial state.

as (cart number)·(another constant distance). The velocities of each cart could be chosen to be constant and equal. However, a key characteristic of this system is that each cart has the same parameters for mass, the spring constant, and the damper constant. Therefore, to avoid conditioning the system so that a "trivial" solution can be found[2], another approach to setting the initial conditions and reference trajectories is needed.

The approach used here is to set each cart's initial configuration and reference position pseudo-randomly. In this work, each cart has its initial position set according to the following equation, given by equation (4.2).

$$y_i(t_0) = (\text{space}) \cdot rand + (\text{space}) \cdot (i - 1) \tag{4.2}$$

Here, *rand* represents a random number between 0 and 1, chosen from a uniform distribution. This random number acts as a percentage and is multiplied against the spacing interval (space) which is chosen to be $1m$. The product of the random number with the spacing variable is summed with the product of the spacing variable with an index that is one index less than the current index. This can be thought of as assigning each cart a region of space the same size as the spacing variable, and choosing a random reference point somewhere within this space. The reference positions are set according to the same formula given by equation (4.2), but with a new set of random numbers for *rand*. The spacing interval was also chosen to be $1m$ for the reference calculation.

The initial velocities are set according to equation (4.3) below, following similar logic as equation (4.2).

$$\dot{y}_i(t_0) = 2 \cdot (\text{range}) \cdot rand - (\text{range}) \tag{4.3}$$

[2]Trivial in this sense means that each cart can largely use the same control strategy as its surrounding carts to achieve the reference.

The term, "range," is meant to evoke the notion that the velocity is set to fall within a window, such that $-\text{range} \leq \dot{y}_i \leq \text{range}$. The range term in this work was set to be $10m/s$. Thus, all initial velocities are somewhere within $\pm 10m/s$.

The same parameters used for case study I will be used again here ($m = 1kg$, $k = 20N/m$, and $b = 1N/(m/s)$). In the following section, the new MPC and consensus ADMM control algorithm will be defined.

4.2 Consensus ADMM Algorithm: Principles

The combination of MPC and consensus ADMM follows a similar pattern as the combination of MPC with "regular" ADMM first discussed in chapter 2. Before this formulation is fully described, the consensus ADMM algorithm itself must first be discussed. This will be done in subsection 4.2.1. The new stopping criterion for consensus ADMM will be briefly discussed in subsection 4.2.2. Afterward, subsection 4.2.3 will prime the algorithm so that it can be incorporated into MPC in section 4.3.

4.2.1 Consensus ADMM: Formulation

The use of consensus ADMM in this work is similar to its usage in [7]. In [7], consensus ADMM helps the distributed MPC control of a flocking problem. Much of the approach used here is based off of this approach because of shared characteristics between the mass-spring-damper system, and the system used in [7]. The approach in [7] and the work presented in this discussion is also largely based on the description provided by [8]. In [8], a detailed description of the consensus ADMM method is provided. Thus, both [7] and [8] are foundational to the analysis provided here.

The description of consensus ADMM here is heavily based on the formulations of ADMM described in chapter 2, and subsequently on the explanations provided in [7] and [8]. For that reason, some shared characteristics between consensus ADMM and the chapter 2 formulation of ADMM will be discussed in less detail here. The primary purpose of consensus ADMM is to minimize a separable cost function with several

terms that can be analyzed apart from other variables [8]. In the cost function given by (4.1), each additive term has no impact on the terms around it. Consensus ADMM takes advantage of this property by separating the large optimization problem into several smaller local optimizations [8]. The general definition of consensus ADMM in this type of application is given below[3] by equations (4.4)–(4.8), as described in [8].

$$\text{Minimize}: \ \Sigma_{i=1}^{N} f_i(x_i) \qquad \text{Subject to}: \ (x_i - \tilde{z}_i) = 0, \tag{4.4}$$

$$\text{Where}: \ i = 1, ..., N$$

$$\text{Augmented Lagrangian}: L_\rho(x_1, ..., x_N, z, y) = \Sigma_{i=1}^{N}\big(f_i(x_i) + y_i^\top (x_i - \tilde{z}_i) + \cdots$$

$$\cdots + (\rho/2)\|x_i - \tilde{z}_i\|_2^2\big) \tag{4.5}$$

$$\text{Algorithm}: \ x_i^{k+1} := \underset{x_i}{\operatorname{argmin}} \ f_i(x_i) + y_i^{k\top} x_i + (\rho/2)\|x_i - \tilde{z}_i^k\|_2^2 \tag{4.6}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \ \Sigma_{i=1}^{m}\big(-y_i^{k\top}\tilde{z}_i + (\rho/2)\|x_i^{k+1} - \tilde{z}_i\|_2^2\big) \tag{4.7}$$

$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - \tilde{z}_i^{k+1}) \tag{4.8}$$

The x and y variables above are generally understood as they were in section 2.2.1. However, because consensus ADMM breaks the problem into smaller local problems, there are more than one x and y variable. The different versions of these variables are denoted with the i subscript. Each represents a local problem to be solved. The $\tilde{z}_i$ variable differs from the original definition of z in section 2.2.1. In consensus ADMM, the z term is indicative of a global variable that contains information about each local variable [8]. The index on the $\tilde{z}_i$ term refers to which parts of the global variable correspond to the i-th local variable [8]. Therefore, the $\tilde{z}_i$ notation represents the notion of the proper value of local variable, x_i, as noted in [7] and [8]. Thus, the whole problem can be summed up as the minimization of a cost function subject to

[3]The variables in equations (4.4)–(4.8) do not have anything to do with the dynamics of the system. The consensus ADMM algorithm will be reformulated with the dynamics variables in the following paragraphs.

the constraint given at the end of equation (4.4). This constraint indicates that the local variable (x_i) and its corresponding counterpart in the global variable ($\tilde{z}_i$) should match [8]. This will be expanded upon further in the rest of this chapter.

Equation (4.4) demonstrates that the cost function is separable in terms of the variable x_i. As alluded to in the previous paragraph, the x-minimization step given by equation (4.6) can be broken down into N separate x_i-minimizations. These operations essentially minimize one of the additive terms which helps collectively make up the augmented Lagrangian (equation (4.5)). This is shown explicitly in [7]. Although it is not explored in this work, it is possible that each of these x_i-minimizations could be done in parallel. The level of system complexity and available computational power will help determine if this distributed style of computation improves the computation time. In this work, the distributed nature of the x_i-minimizations will be particularly helpful because it limits the local problem size, thereby improving computation time without having to resort to parallel computation. This will be discussed in detail in section 4.5.

After all N x_i-minimizations have happened, the z-update can take place. As equation (4.7) shows, there is only one z variable to be updated. This is because in consensus ADMM, the z variable is a global variable. As it will be used in this work, this step can be rewritten from its general form in equation (4.7) to a new form given by equation (4.9), as shown in [8].

$$z_g^{k+1} := (1/k_g)\Sigma_{G(i,j)=g}(x_i^{k+1})_j \tag{4.9}$$

As noted in [8], the i and j indices indicate which part of the global z variable is being operated on. The term $G(i,j)$is an indexing term that describes how portions of one local variable may overlap with another [8]. This will be discussed in greater detail with modified notation in a later section. Because k_g represents how many

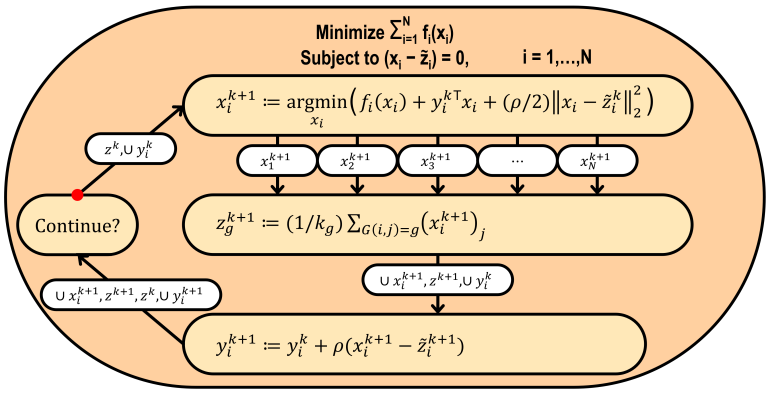

Figure 4.1: Basic form of consensus ADMM. The x_i term is a local variable, and the $\tilde{z}_i$ term is the corresponding part of the global variable. The z_g term represents a similar quantity as $\tilde{z}_i$, but with different notation. The dual variable is y_i. The term k_g indicates how many local variables contribute to a part of the global variable.

local variables contribute to a specific portion of the global variable, this new representation indicates that the z-update is really the collection of local averages of all these contributing terms [8]. The new representation of the algorithm is given by figure 4.1.

As figure 4.1 shows, there are N outputs from the first set of x_i-minimizations, one for each local variable. These outputs are all collected in the global variable update. Each individual dual variable is then updated before the stopping criteria are checked. The algorithm then chooses whether or not to repeat based on the result of the stopping criteria check. A concise summary of this process is given in [7].

The calculated stopping criterion for consensus ADMM is slightly different than that for "regular" ADMM discussed in chapter 2. These differences, as discussed in [8], will be noted briefly in subsection 4.2.2.

4.2.2 Consensus ADMM: Stopping Criterion

The calculated stopping criterion for consensus ADMM is very similar to the original stopping criterion described in subsection 2.2.3. The actual check itself, given originally by equation (2.23), remains the same. As do the formulations of ϵ^{pri} and

ϵ^{dual}. Equation (2.23) in chapter 2 is repeated below for convenience.

$$\text{if} \quad \left\{ \|r^k\|_2 \leq \epsilon^{pri} \quad \text{AND} \quad \|s^k\|_2 \leq \epsilon^{dual} \right\} \quad \Longrightarrow \quad \text{end algorithm}$$

The calculated stopping criterion differs though in the construction of the primal and dual residuals, which are again denoted as r and s respectively, as shown below. Instead of using the form presented in chapter 2, r and s can be determined according to equations (4.10) and (4.11) below, which are based on the discussion in [8].

$$r^{k+1} = (x_1^{k+1} - \tilde{z}_1^{k+1}, ..., x^{k+1} - \tilde{z}_N^{k+1}) \tag{4.10}$$

$$s^{k+1} = -\rho(z^{k+1} - z^k, ..., z^{k+1} - z^k) \tag{4.11}$$

This calculated stopping criterion will be used in conjunction with the limit on how many ADMM iterations can be performed to determine when the algorithm should stop.

4.2.3 Consensus ADMM: Notation Update

Before consensus ADMM can be incorporated into MPC, the consensus ADMM algorithm must be placed in similar terms with the MPC algorithm. This can be done by modifying the variables from figure 4.1. The results of doing so are shown in figure 4.2. In this figure, u_x and u_z still represent commands that can be given to a dynamic system, like they did in the previous MPC-ADMM combination algorithm. However, the u_x term has the i index introduced to indicate the separable nature of this term. Furthermore, u_z is still both a command and a global variable to the system. The i index on $\tilde{u}_z$ extracts the information from u_z that corresponds to local variable $u_{x,i}$. Lastly, the cost function is broken down in the same way as the local variables[4].

[4]The cost function maintains the implicit nature of accepting an initial condition and input command, in order to output a scalar cost, like before.

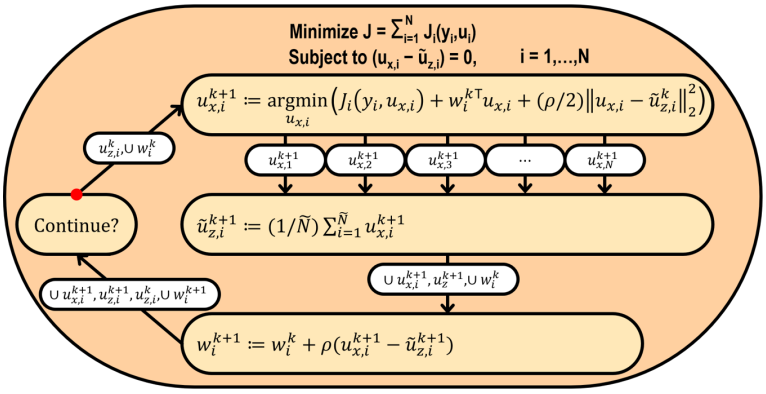

Figure 4.2: Consensus ADMM with simplified representation of the $\tilde{u}_z$-update.

It is important to note that the u_z-update step has been greatly simplified from its original form in equation (4.7). Originally, the u_z-update step had a term related to the dual variable in it. However, both [7] and [8] claim that since each dual variable is zero after one iteration of this algorithm, the u_z-update step can be reduced to a form resembling that in figure 4.2. Here, the $\tilde{u}_z$-update for a given index i is the average of overlapping suggestions from each local x_i variable. Included in this average are only the suggestions for the common local variable components. Subsequently, the " ~ " notation on the variable $\tilde{u}_z$ and the $\tilde{N}$ term are meant to denote that a local average is taking place[5].

Using this notation, consensus ADMM can now be conceptualized in the study of a dynamic system. In section 4.3, consensus ADMM in the context of the mass-spring-damper system will be explored. Then, the consensus ADMM algorithm will be combined with MPC in the context of the mass-spring-damper cart example.

4.3 Incorporation of Consensus ADMM into MPC

In this section, the incorporation of consensus ADMM into MPC will be discussed. Many characteristics of this new method are similar to the MPC-ADMM combination

[5]The " ~ " notation is *not* an operator. Here, $\tilde{u}_z$ indicates that the global variable itself, u_z, is obtained by averaging local variables together. The term, $\tilde{N}$, indicates the number of local variables included in a local average. Depending on system characteristics, this $\tilde{N}$ might not be uniform for each average.

method first assembled in section 2.4. First, consensus ADMM will be situated in a dynamic system. Then, the full MPC and consensus ADMM algorithm formulation will be discussed.

The mass-spring-damper system from chapter 3 will be used again in this analysis. As with the original MPC-ADMM combination algorithm, consensus ADMM takes place in the optimization step of MPC. In other words, consensus ADMM can be used instead of a regular solver at each optimization step of MPC. Consensus ADMM will be placed in the context of dynamic system by examining a single optimization step of the MPC algorithm in the paragraphs below.

As previously noted, the objective in this multi-cart system is to have each cart move to, and stay at, a given reference trajectory. Since each cart in the system has the ability to apply a torque to its wheels, each cart must identify a proper control input. The methods used in chapters 2 and 3 would suggest that all inputs could be determined simultaneously in one calculation. However, the consensus ADMM strategy will break up this single computation. A simple way of doing this would be to have each individual cart calculate *only* its own trajectory in the u_x-minimization step. However, consensus ADMM suggests a more interconnected method.

In this context, consensus ADMM divides the calculation as shown in figure 4.3. Each row of carts in figure 4.3 is the map of an agent and its range of influence. Each cart in green represents a main cart of interest in a local problem, while the yellow carts represent the rest of the neighborhood of influence. Consensus ADMM in this case will work by performing one optimization for each *green* cart and its surrounding neighborhood of carts. This will calculate the suggested control strategy that would allow each cart in the smaller problem to achieve its reference trajectory. Together, this constitutes the $u_{x,i}$ minimization steps. The overlapping control strategies from each $u_{x,i}$ are then averaged in the $\tilde{u}_{z,i}$ updates. An overlapping control strategy is defined as when two carts each make a suggestion for the same cart. This will be

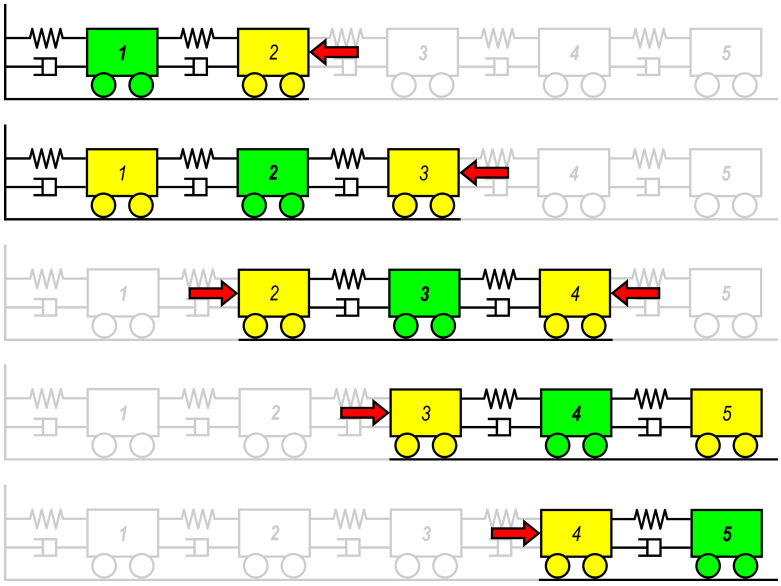

Figure 4.3: Consensus ADMM diagram for a dynamic system. The green cart in each row is the central agent. The yellow carts are its neighbors. The yellow carts have their dynamics affected by the red arrows, which represent constant disturbance forces. The light gray carts contribute to these disturbance forces.

explained further below.

The averaging step for cart 1 (figure 4.3) will be detailed here. Figure 4.3 shows that cart 1 creates a control strategy for itself[6], and cart 2. These control strategies are collectively denoted as $u_{x,1}$. Meanwhile, cart 2 creates a control strategy for cart 1, itself, and cart 3. This is collectively denoted as $u_{x,2}$. No other carts in the system create a control strategy for cart 1. Therefore, to calculate the average control for cart 1, the component of $u_{x,1}$ that relates to the control of cart 1 must be averaged with the component of $u_{x,2}$ that relates to the control strategy of cart 1. In this $\tilde{u}_z$-update equation, $\tilde{N} = 2$ because there are two suggested control strategies in the average. This resulting average is stored in the global variable u_z. If this example had been for any of the carts in the middle of the system, then there would have been three control strategies to include in the average step (and $\tilde{N}$ would have been equal to 3 in the update equation).

With consensus ADMM conceptualized in this way, the algorithm given by figure

[6]Each green cart can be thought of as a processor that solves its own local problem.

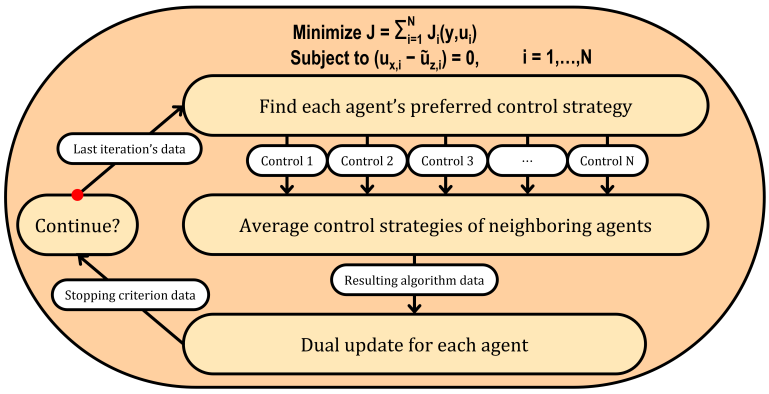

Figure 4.4: Summarized consensus ADMM algorithm. This diagram puts into words what happens at each step in consensus ADMM. It also more clearly shows how there are many results from the $u_{x,i}$ steps.

4.2 can be reformulated as shown in figure 4.4. Figure 4.4 restates what happens at each step in the algorithm with words instead of symbols. As noted here, each agent in the system performs a local optimization where it calculates a control plan for itself and its neighbors. The overlapping control commands are then averaged together and stored in a single global parameter. Then, each local agent's dual variable is updated with this new information before the stopping criteria are checked in order to see if the algorithm should repeat. This cycle is followed for each step in the MPC control. This process is also summarized in [7]. Consensus ADMM as described here fits in the solve step of MPC in a similar way as the original ADMM algorithm did in chapter 2. This combination of MPC and consensus ADMM is shown in figure 4.5. The algorithm depicted in figure 4.5 will be used in the simulations in section 4.5.

In this section, the combination of MPC with consensus ADMM was discussed. First, consensus ADMM was conceptualized in terms of a dynamic system. This understanding was then applied to the original representation consensus ADMM. This led to the formal incorporation of consensus ADMM into MPC. In the following section, the control of a 10-cart system will be examined, using only MPC control. This will establish a baseline from which to compare the newly defined MPC with consensus ADMM algorithm.

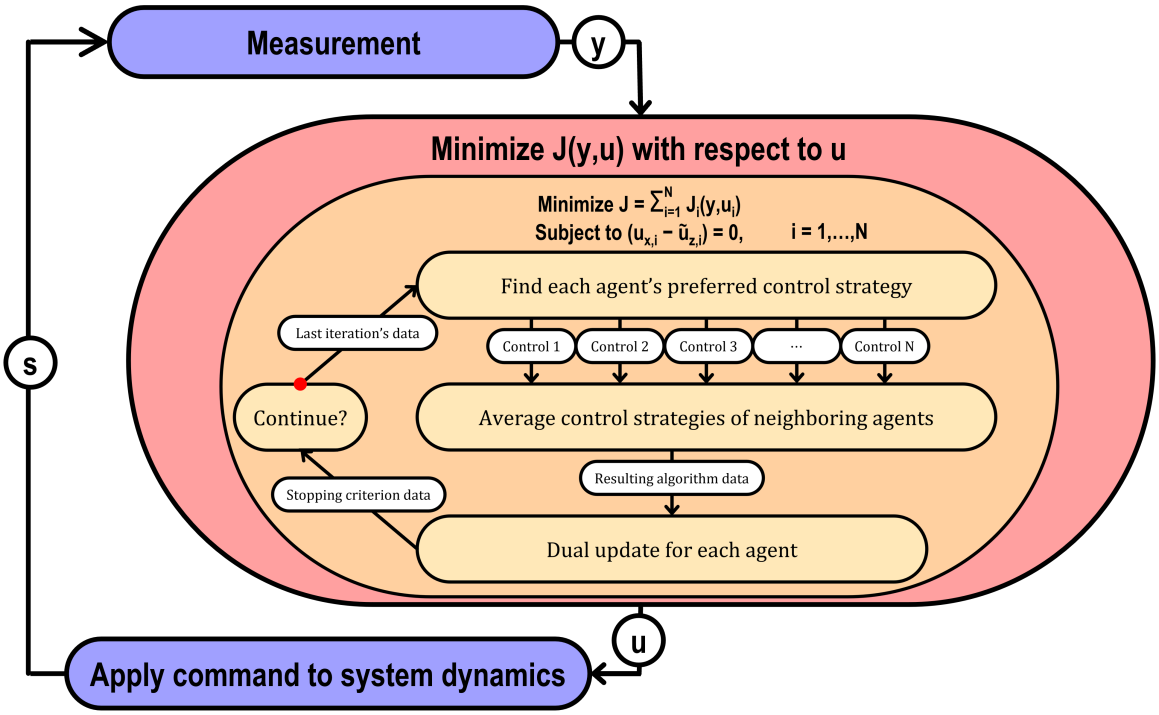

Figure 4.5: Formal incorporation of consensus ADMM into the MPC control algorithm.

4.4 MPC-Only Algorithm: 10-Cart System

In this section, MPC control of a 10-cart system will be explored to give a baseline from which to compare the performance of the MPC with consensus ADMM combination algorithm. This simulation uses the equations of motion and state matrix equations discussed in section 4.1. The position cost function is equally weighted among the different carts, and the reference values each cart follows were determined pseudo-randomly, as described in section 4.1.

One performance metric is the overall cost function at the end of the simulation, which collectively quantifies each individual cart's performance. The purpose of this metric is primarily to give a binary answer as to whether a control strategy offers a competitive performance. Thus, it is not the primary metric of interest. Instead, the approximate computation time is the primary performance metric. All of the caveats made about the meaning of this *approximate* metric in section 3.3.2 hold here as well.

For a control strategy to be deemed "better" than another, it must have a cost in the neighborhood of its rival's, while having a *substantially* improved computation time. The improvement must be sufficiently large enough so as to work around any peculiarities in the approximate time measurement.

For this simulation, the same mass, spring constants, and damper constants used in case study I were used again on each cart. The modular nature of this system also allows for reasonable system performance when the MPC parameters for a single agent are used for the whole system. In this sense, it is like each cart is considered an individual unit as in case study I, but subject to some disturbance caused by its neighboring carts. Therefore, the same time step (dt), prediction horizon (p), and control horizon (m) used in case study I will be used again here.

The results of the simulation described above are shown by figure 4.6. Figure 4.6 displays the results of the decision plot, which measured the system once every time step ($dt = 0.3s$). This simulation used a spacing interval of $1m$ to set the random initial position and references. The carts also had a random initial velocity chosen between $-10m/s$ and $10m/s$.

This simulation took approximately $1240s$ to run, which is quite substantial relative to the simulations in case study I. Equation (4.1) was used as the cost function. The cost was calculated from data taken once every time step ($dt = 0.3s$), and amounted to $J = 1.8779$. The results displayed in figure 4.6 show that the system was able to achieve the references for each cart reasonably well. The simulation plot (figure 4.7) gives a more accurate depiction of the behavior of the system. As with case study I, the simulation plot uses the same input strategy as the decision plot with commands adjusted to match the step input pattern.

Figures 4.6 and 4.7 indicate that the MPC-only control algorithm provides the system with a reasonable ability to follow each cart's reference. However, as expected, computing a control command for all 10 agents at once is taxing with respect to

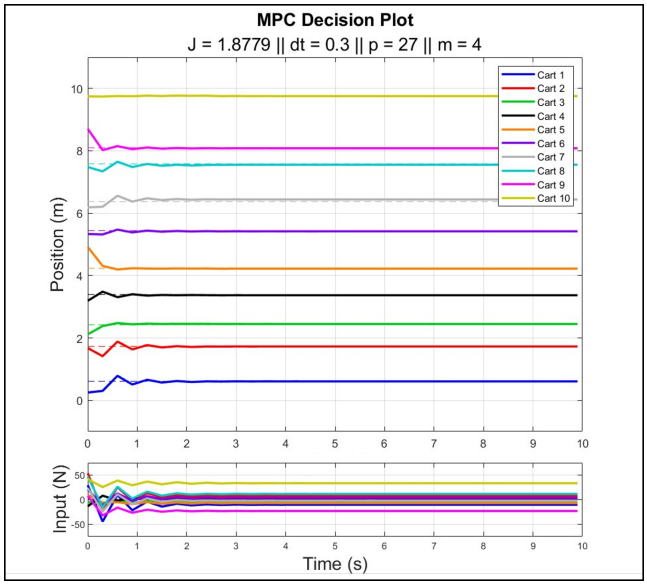

Figure 4.6: Decision plot for MPC-only control on a 10-cart system. Approximate calculation time was 1240s. Each cart's position is noted with a solid line, with corresponding colors noted in the legend. Each cart follows a reference trajectory plotted with a dashed line in the same color as its cart position line.

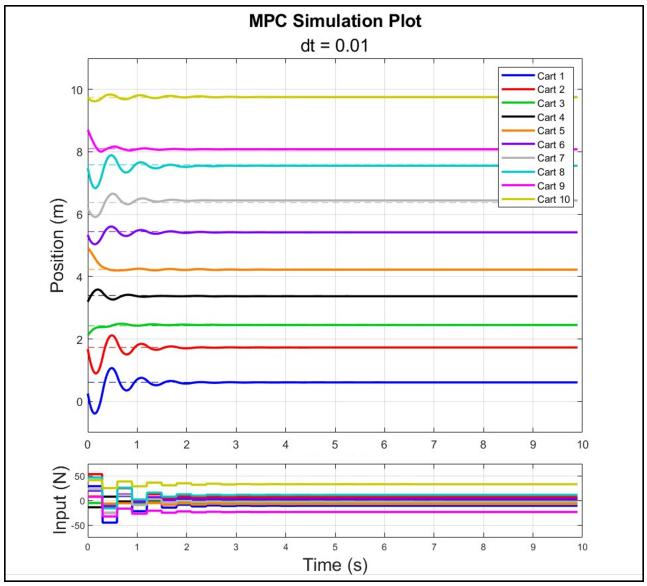

Figure 4.7: Simulation Plot for MPC-only control on a 10-cart system.

time. Section 4.5 will examine the performance of the MPC and consensus ADMM combination control algorithm on the system.

4.5 MPC and Consensus ADMM Algorithm

In this section, the performance of the MPC with consensus ADMM algorithm will be explored. As noted previously in this chapter, this control strategy divides the optimization problem into smaller problems that will be solved serially, rather than in parallel. Subsection 4.5.1 will first give more details about the cost function. Then, subsection 4.5.2 will analyze the results of the simulation.

4.5.1 MPC and Consensus ADMM Algorithm: Cost Function

The cost function given by equation (4.1) is used as the foundational cost function in this simulation. As with before, the cost function maintains the implicit action of running a simulation that outputs a position vector to be used in the cost calculation. However, this cost function is divided and weighted in a new way according to consensus ADMM. Each of these $u_{x,i}$ minimization steps considers the additive $\mathcal{L}_2$-norm term for itself (i), and the additive $\mathcal{L}_2$-norm term(s) for its neighbors ($i-1$, and $i+1$). For the first and last carts in the system, only one adjacent cart ($i+1$ and $i-1$, respectively) will be considered in addition to the current cart. The $u_{x,i}$-minimization steps shown in figure 4.2 show that there are additional terms to be added to each local agent's cost function as well.

The additional terms given to each agent here are the relevant portions of the global variable and the local dual variable. The control commands must be within $\pm 75N$, which can be used as an estimate of the maximum magnitude of the augmented term before the $\mathcal{L}_2$-norm squared is taken. As with before, the position terms in the cost function are likely within roughly $1m$ of each reference trajectory. This creates a similar phenomena as was seen in the MPC-ADMM algorithm. In the case of the MPC-ADMM algorithm, the position data was multiplied by a constant to counteract

the potentially over-impactful augmented term. The same strategy is used again here to solve this problem.

It would not make sense to use the same multiplicative constant again however, because the incorporation of the agent's neighbors in the cost function could exaggerate the discrepancy between the two terms. Therefore, instead of using a weight of 10000 like in case study I, a weight of 200000 will be used here instead. It is important to note that this weighting is only used in the decision-making process for MPC. Once the simulation has run completely, the position data is then fed through equation (4.1) to obtain the output cost. This ensures a fair comparison with the MPC-only output cost.

4.5.2 MPC and Consensus ADMM Algorithm: Simulation and Analysis

With the cost function more fully defined, the new 10-cart simulation may now be discussed in detail. For this simulation, the same MPC parameters were used again ($dt = 0.3s$, $p = 27$, $m = 4$). The same pseudo-randomly generated references from the MPC-only simulation in section 4.4 were used here. A constant ρ was used again for this simulation ($\rho = 10$). The maximum number of allowable ADMM iterations for this simulation was set to be 2. The decision plot for this simulation is given by figure 4.8.

Figure 4.8 illustrates that this control strategy enables the reference to be achieved with a reasonable amount of accuracy. The overall cost of this simulation was $J = 1.7497$, which is marginally better than the MPC-only control algorithm's cost. Most importantly though, this simulation only took about $195s$ to run. This is *substantially* less time than the MPC-only algorithm ($1240s$). This improvement is likely due to the inherent principles of consensus ADMM since a large optimization problem has been broken down into modular components. Figures 4.6 and 4.8 are validation that the new MPC and consensus-ADMM control strategy not only has the potential to match the performance of the MPC-only control strategy, but vastly improves upon

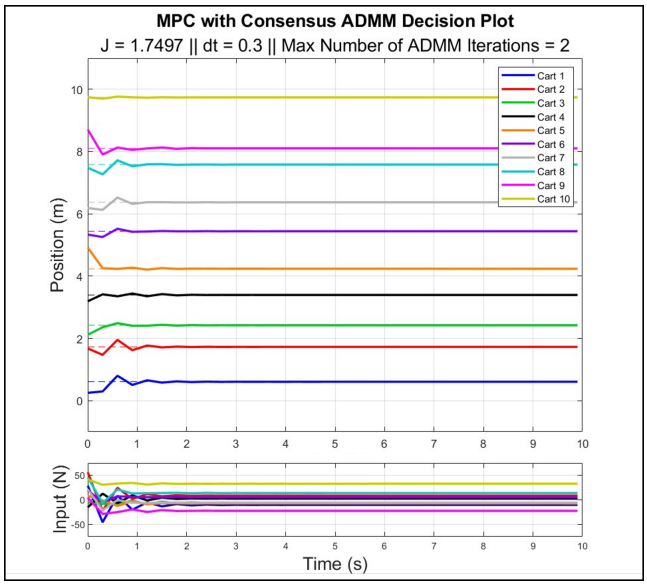

Figure 4.8: MPC with consensus ADMM combination algorithm. Elapsed time was approximately 195s. The reference trajectories are represented by the dashed lines and are color-coded to match their corresponding cart position data.

it with respect to computation time.

The simulation plot for this MPC with consensus ADMM application is shown by figure 4.9. Again, the "simulation plots" are only meant to clarify system behavior by re-running the simulation with a smaller time step and control inputs that are based on the final collective control vector from the decision plot. The simulation plot shown in figure 4.9 illustrates this more detailed behavior.

The system response varies according to the maximum allowable number of ADMM iterations. To examine this variation, two additional simulations were run. One set the maximum number of allowable ADMM iterations to be 1, while the other set this parameter to be 100 iterations. The results of these simulations, and the two simulations already discussed, are shown in table 4.1.

As table 4.1 shows, increasing the number of allowable ADMM iterations did not necessarily mean that better performance was achieved. Allowing 100 iterations forced the system to take around 2350s to calculate a solution. Using only 1 iteration gave the fastest performance, but this cost was marginally worse than the MPC-

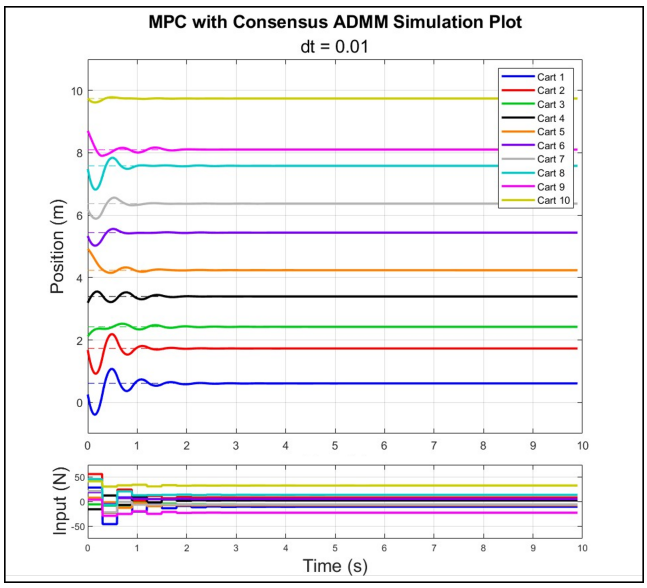

Figure 4.9: Simulation plot for consensus ADMM.

Table 4.1: Data for the MPC-only algorithm and the MPC and consensus ADMM algorithm. This data is for a 10-cart system with a simple $\mathcal{L}_2$-norm cost function.

Simulation	MPC-Only	MPC and Consensus ADMM	MPC and Consensus ADMM	MPC and Consensus ADMM
Maximum Number of ADMM Iterations	N/A	1	2	100
Cost	$J = 1.8779$	$J = 1.9636$	$J = 1.7497$	$J = 1.9114$
Approximate Time	$1240s$	$140s$	$195s$	$2350s$

only algorithm. Thus, setting the maximum allowable ADMM iterations as 2 gave arguably the best performance out of the four simulations.

As noted previously, a similar problem is discussed in [7]. This work also used consensus ADMM as it was described in [8], but applied to a different dynamic system. In [7], it was found that a solution for its dynamic system could be found within 1.5% of the centralized solution after using just 2 iterations of a similar method. While these exact percentages might not be precisely found here, setting the maximum allowable ADMM iterations to 2 did yield performance better than the centralized MPC-only algorithm.

4.6 Case Study II Summary

In this chapter, the difference between a regular MPC-only algorithm and an MPC with consensus ADMM algorithm were examined. First, the equations of motion for the 10-cart, multiagent system were defined. Then, consensus ADMM was described in detail. The MPC-only algorithm was then used in the 10-cart system simulation to establish a baseline control performance. It was then shown that allowing up to 2 ADMM iterations per time step in an MPC with consensus ADMM algorithm gave performance that improved the computation time by a large margin, with no loss in performance. In chapter 5, this strategy will be applied to a traffic network.

CHAPTER 5: CASE STUDY III – TRAFFIC NETWORK

As discussed in chapter 1, the purpose of this work is to implement a control method for traffic networks that is based on Model Predictive Control (MPC). Since the limiting factor of MPC in this context is computation time, the tool in development must manage the overall computational load. Chapter 2 laid a foundation from which to explore various strategies for making such a tool. The main ideas explored in this chapter were the concepts of MPC and the Alternating Direction Method of Multipliers (ADMM). The most important concept explored in chapter 2 was the combination of MPC and ADMM together. Chapter 3 (case study I) explored the use of an MPC-ADMM combination algorithm, to limited success. A new strategy was explored in chapter 4 (case study II) that was a combination of MPC with *consensus* ADMM. Using this strategy, promising results were found for an interconnected mass-spring-damper system with multiple carts. The principles explored throughout the development of this last algorithm will now be applied to a traffic network with modular characteristics, similar to the multiple-cart system explored in case study II.

First, this chapter will explore the dynamics of a traffic network in section 5.1. The dynamics that govern a traffic network are not quite as straightforward as the simple mass-spring-damper system. Thus, much care must be taken in exploring the behavior of the system before applying control to it. After the dynamics are discussed in detail, the constraints on these dynamics will be discussed in section 5.2. There is much nuance to how this system applies constraints, and much care will be exercised in discussing this process. Section 5.3 will then further examine key system characteristics and parameters, and section 5.4 will discuss logistical details of the simulation method. Sections 5.5 and 5.6 will analyze and discuss the results of two

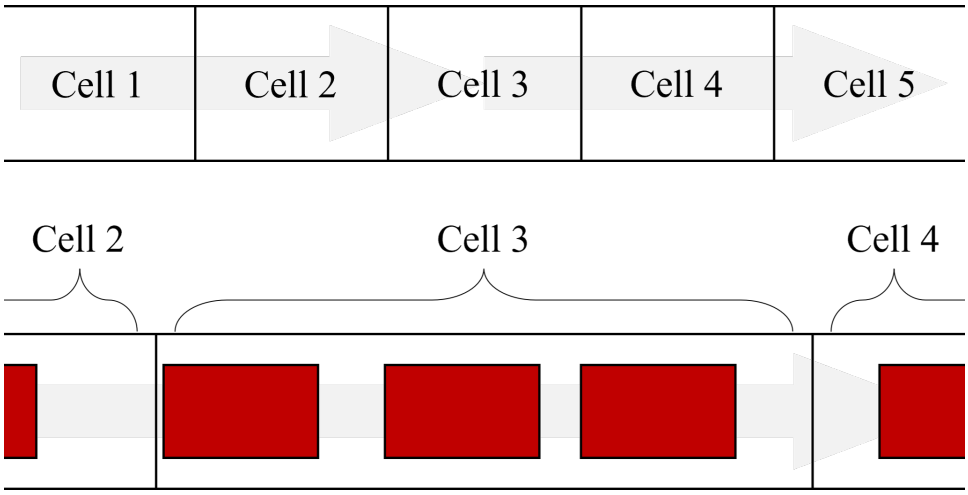

Figure 5.1: Representation of a traffic network. Each cell is of equal length and has a single lane. There are no on/off-ramps. Traffic flows from the left (upstream) to the right (downstream). The red rectangles represent vehicles, although not drawn to scale.

different experimentation setups. Lastly, section 5.7 will summarize key results from both of the experiments in this chapter.

5.1 Traffic Dynamics

In this section, the dynamics of a simplified traffic network will be explored. In [12], many different network characteristics that can affect the dynamics of such a system are examined. Some of the characteristics examined in [12] include vehicle type, movement cooperation among vehicles, and overall model scope. The main characteristics studied in the system explored in this work are relatively simple compared to these, in order to highlight the relative effectiveness of the control scheme in development. First, the model used here is for a single-lane section of interstate, with no on/off-ramps. Furthermore, the vehicles all have the same type of dynamics. A pictoral representation of the system is given by figure 5.1. These simplifications will enable a more focused analysis of the algorithmic tools in development here by limiting extraneous factors that could affect the system. With these assumptions established, the behavior of the system can now be discussed.

In this work, a section of single-lane interstate is divided into many cells of equal length. The system measurements are the vehicles in each cell (per unit length) and the average velocities of each cell. Furthermore, the entry/exit of vehicles with respect to each cell is monitored. These measurements are quantified in terms of average vehicle density, average velocity, and average flow respectively. Density and velocity are the main parameters used to characterize the system based on equations found in [3], [13], [14], [15], and [16]. Proper understanding of these terms is necessary in order to understand their implications in the dynamic equations that follow.

In this work, the term *density* refers to the number of vehicles in a given cell per unit of length. Therefore, density is measured with units of $vehicles/(lane \cdot m)$. Since this work only considers single lane traffic networks, the *lane* unit is dropped from the density representation, with the knowledge of its implicit existence ($vehicles/m$). This simplified notion of density is used throughout the rest of this work. Average velocity is measured in m/s.

The last system variable pertaining to this work is traffic *flow*. This parameter describes the number of vehicles that exit a cell per unit of time. Thus, flow is measured in terms of $vehicles/(lane \cdot s)$. As with the density units, the *lane* dependence from the units of flow will be dropped for simplicity ($vehicles/s$) since this work only considers one lane systems. This is done with the implicit knowledge that there is a lane dependence in this term. Flow has the special characteristic as being the product of the average density of a current cell with the average velocity of the current cell. This is shown below by equation (5.1), where flow, density and velocity are represented by q, k, and v respectively[1]. These variables are foundational to the equations that govern the system.

$$q = k \cdot v \tag{5.1}$$

[1]Although the work in [3] uses $q = \lambda \cdot k \cdot v$ to represent this relationship, the λ term is superfluous here because the number of lanes is set as $\lambda = 1$.

Understanding this relationship between density, velocity, and flow is key to understanding the performance of this system. It should be understood that these quantities are averages, although this will not be explicitly stated each time these terms are used. In subsection 5.1.1, the primary equations which govern the system will be discussed in detail. These equations are nonlinear, which will require a different method for solving relative to the methods used for the linear systems examined in case studies I and II. In case study III, these equations are solved with a solving tool directly, rather than using discrete simulation strategies to update the system based on the $\dot{s} = As + Bu$ and $y = Cs$ equations from chapter 3.

5.1.1 Traffic Dynamics: Rate of Change

In this work, there are two primary equations that will be used to characterize the dynamics of the system. These equations are the rate of change of density and the rate of change of velocity. Although the works, [3], [13], and [14], each use the discrete version of the rate of change of density, the continuous form will be used in this work. The continuous version of the equation is given in [15] and [16]. This continuous-time representation is shown below by equation (5.2)[2].

$$\dot{k}_i(t) = \frac{1}{L\lambda}\big(q_{i-1}(t) - q_i(t)\big) \tag{5.2}$$

In this representation, k is the variable used for average density. It follows that $\dot{k}(t)$ represents the rate of change of average density with respect to time. The quantity L refers to cell length, which is the same for all cells. The quantity λ refers to the number of lanes in the cell. Since this work only considers single-lane networks, this λ term can essentially be ignored since its value is always 1. The term $q(t)$ represents flow. Since the i index indicates a cell reference, the $i-1$ term refers to an adjacent upstream cell, and $i+1$ refers to an adjacent downstream cell. By this indexing,

[2]Additive terms related to disturbances from traffic ramps have been omitted.

equation (5.2) indicates that the change of density of any given cell depends only on itself and the cells preceding it. This dependence is related strictly to the flow variable[3]. The rate of change of velocity equation is quite different from this equation, and more complex.

Both [3] and [13] use the discrete form of the rate of change of velocity equation. Again, in this work, the continuous form of the equation is used instead. This form of the equation is given in [15] and [16]. There are no disturbance terms in the representation used here. This is given below by equation (5.3).

$$\dot{v}_i(t) = \frac{1}{L}\left[v_i(t)(v_{i-1}(t) - v_i(t)) - \frac{\eta}{\tau}\frac{k_{i+1}(t) - k_i(t)}{k_i(t) + \zeta}\right] + \frac{1}{\tau}(U_i(k_i) - v_i(t)) \tag{5.3}$$

In equation (5.3), $v(t)$ refers to the average velocity of a given cell, and $k(t)$ refers to the density of a given cell. The variables η and ζ are constant and govern specific characteristics of the system. The τ constant is a time constant. In this representation, $U_i(k_i) = (1 - \beta_i)V_i(k_i)$, as shown in [15] and [16]. Although k_i is implicitly a function of time, for a single instant this value can be treated as a constant. Thus, the time dependence is not explicitly noted. As referred to in [3] and [13], $V_i(k_i)$ represents an equation that relates current cell density to the average desired velocity for the vehicles in that cell. This term will be discussed in explicit detail in subsection 5.1.2, but it is important to note that β_i is a control command subject to the constraint that $0 \leq \beta_i \leq 1$. The definition of $U_i(k_i)$ is substituted into equation (5.3), and the result is given below in equation (5.4).

$$\dot{v}_i(t) = \frac{1}{L}\left[v_i(t)(v_{i-1}(t) - v_i(t)) - \frac{\eta}{\tau}\frac{k_{i+1}(t) - k_i(t)}{k_i(t) + \zeta}\right] + \frac{1}{\tau}(V_i(k_i) - v_i(t)) - \frac{V_i(k_i)\beta_i}{\tau} \tag{5.4}$$

Using this form of the equation is beneficial because it more directly shows the

[3]With the knowledge that flow is actually the product of a cell's density with its velocity, this equation can be rewritten as, $\dot{k}_i = \frac{1}{L\lambda}(k_{i-1}v_{i-1} - k_i v_i)$

effect $V_i(k_i)$ has on the system. In subsection 5.1.2, the meaning of this term will be explored in detail.

5.1.2 Traffic Dynamics: Desired Velocity

The equation for $V_i(k_i)$, based on its representation in [3] and [13], is given below by equation (5.5). This representation is more similar to the $U_i(k_i)$ term usage, but viewing it in this way will help contextualize the control command later.

$$V_i(k_i) = \min\left(v_{ff}\,\exp\left[\frac{-1}{a_{m,i}}\left(\frac{k_i}{k_c}\right)^{a_{m,i}}\right], (1+c)\,V^{(*)}\right) \tag{5.5}$$

The leftmost argument to the minimization function describes the system's desired response under no control influence. This term could be thought of as the "dynamics' desired velocity," which expresses the idea that the term represents system behavior under no control influence. The rightmost argument describes the theoretical effective response to a suggested velocity. Full examination of these terms is necessary to understand much of the work that follows.

In equation (5.5), the term v_{ff} is the free-flow velocity. The free-flow velocity term governs how fast vehicles in a given cell can go when they have no restrictions to their speed outside of the traditional speed limit, as noted in [14]. The constant k_i is the density in a given cell at a specific instant, and k_c is the critical density. The critical density describes the maximum number of vehicles in a given cell such that their average velocity is not restricted due to the quantity of vehicles in the cell, allowing them to travel at the free-flow velocity [2]. This density is therefore where the maximum flow can be achieved, as noted in [2] and [12]. The term $a_{m,i}$ is a constant that is dependent on the density of the given cell. The manner in which this constant is updated will be discussed in subsection 5.1.3, and is based on the method used in [15] and [16]. Lastly, the controller's suggested velocity, as given in [3] and [13], is noted here by $V^{(*)}$. The term, $(1+c)$, is a compliance parameter. The

rightmost argument in equation (5.5) will be modified for this research.

In this research, as with the work presented in [13], the compliance parameter c is set to 0. In other words, the vehicles comply with the suggested velocity $V^{(*)}$. Furthermore, this work sets the suggested velocity $V^{(*)}$ to always be equal to the dynamics' desired velocity. This implementation is similar to that in [15] and [16]. Importantly, making this modification does not change the output of equation (5.5). This is because no matter the value for suggested velocity, the output of (5.5) can never be greater than the dynamics' desired velocity. Therefore, this sets an effective range in which the right-hand side argument can operate effectively. The modified right-hand side argument can be multiplied by $(1-\beta_i)$, which effectively incorporates the control command into the original $U_i(k_i)$ term. The β term is the control command[4], which itself is restricted to be between 0 and 1. The right-hand side argument can be summarized as being "some percentage of the dynamics' desired velocity." The new equation, with these changes implemented, is given by equation (5.6) below[5].

$$V_i(k_i) = \min\left(v_{ff}\,\exp\left[\frac{-1}{a_{m,i}}\left(\frac{k_i}{k_c}\right)^{a_{m,i}}\right], (1-\beta_i)\,v_{ff}\,\exp\left[\frac{-1}{a_{m,i}}\left(\frac{k_i}{k_c}\right)^{a_{m,i}}\right]\right) \tag{5.6}$$

The leftmost argument in (5.6) is fixed for a given instant. Subsequently, the rightmost argument must be somewhere between zero and the left-hand side fixed value because of the restriction on β_i to be between 0 and 1. One could therefore argue that the output of this minimization will always be given by the right-hand side of the equation[6]. Therefore, the new desired velocity equation can be simplified as shown below by equation (5.7). This form is most similar to the representation of this term in [15] and [16]. As noted in [15] and [16], a choice of $\beta > 0$ corresponds with

[4]This enables the controller to work by selecting a parameter for β_i, rather than choosing a suggested velocity from scratch.

[5]Technically, the $V_i(k_i)$ term in equation (5.4) itself is given by the left-hand side argument of equation (5.6) only. However, this representation is given to help contextualize the control command.

[6]If the two arguments are equal (if β_i is zero) then the output is equivalently given by either argument.

commanding the vehicles to slow down, while choosing $\beta = 0$ is the same as doing nothing to the system. Writing the desired velocity equation in this way greatly simplifies it notationally.

$$V_i(k_i) = (1 - \beta_i)\, v_{ff}\, \exp\left[\frac{-1}{a_{m,i}}\left(\frac{k_i}{k_c}\right)^{a_{m,i}}\right] \tag{5.7}$$

Intuitively, the result of this minimization will always be between 0 and v_{ff}. This is because the term inside the exponential is negative, and is therefore restricted to output something between 0 and 1. This reduces the output desired average velocity to being some percentage of v_{ff}, multiplied by the control command term $(1 - \beta_i)$. This representation is foundational to understanding the meaning of the control command given by β_i.

5.1.3 Traffic Dynamics: Density Versus Desired Velocity

The last component of the desired velocity equation to be discussed is the $a_{m,i}$ term. In [13], this parameter is chosen to be fixed. However, this could lead to some exaggerated system dynamics. To combat this, $a_{m,i}$ is selected discretely based on the density of the current cell. This strategy is based on that used in [15] and [16]. The values used for $a_{m,i}$ are given in tables 5.1–5.2. These tables base the $a_{m,i}$ constants off of k_c and k_j. As alluded to previously, k_c refers to the critical density, while k_j refers to the jam density. The jam density is the maximum allowable density of a given cell. For this case study, v_{ff} was chosen to be $110 km/h$ (or about $30.5555 m/s$), k_c was chosen to be 0.01875 $vehicles/m$, and k_j was chosen as 0.1125 $vehicles/m$. In other words, in each $1000m$ long cell, the maximum number of vehicles permitted was 112.5, while the critical number of vehicles was 18.75. These parameters were chosen based on similar parameters in [16]. These parameters in [16] are very similar to those in [15]. The work of [13] also had some influence these choices, but to a lesser degree.

Table 5.1: Table of $a_{m,i}$ constants. The values for $a_{m,i}$ depend on the density k_i at the current cell (i). Data is continued in table 5.2.

	$k_i < k_c$	$k_c < k_i \leq 1.5k_c$	$1.5k_c < k_i \leq 2.5k_c$	$2.5k_c < k_i \leq 3.5k_c$
$a_{m,i}$	30	4	2	1.8

Table 5.2: Data from table 5.1, continued.

	$3.5k_c < k_i \leq 4.5k_c$	$4.5k_c < k_i \leq k_j$	$k_j < k_i$
$a_{m,i}$	1.5	1.2	1

Figure 5.2 depicts the plot of $V_i(k_i)$ versus k_i, with $V_i(k_i)$ determined according to equation (5.7). This figure was generated using the $a_{m,i}$ constants from tables 5.1–5.2. In this plot, region 1 corresponds to the section where $a_{m,i} = 30$. Region 2 corresponds to $a_{m,i} = 4$ and so on. The justification for this choice of arguments will be given by counter example.

If a fixed value for $a_{m,i}$ was used, as it was in [13], the plot would resemble figure 5.3. This plot illustrates the response when $a_{m,i}$ was always set to be 1.8. Importantly, figure 5.3 shows how the desired velocity of the system is more aggressively decreasing prior to exceeding the critical density in this plot, relative to the behavior in figure 5.2. This is in contradiction with the notion of a critical density being a value for which vehicles can travel at the free-flow velocity when $k_i < k_c$. By choosing $a_{m,i} = 30$, the desired velocity stays closer to the free-flow velocity for longer in region 1 of figure 5.2. The main drawback from varying the $a_{m,i}$ constants is that it induces discontinuities at the points where the constant changes. This trade off is accepted in this work because more value is placed on the theoretical behavior of the vehicles compared to smoothness of this plot.

Lastly, it is important to note that there is a region that exists for $k_j < k_i$. While it may seem counter intuitive to have a parameter that exists for a seemingly improbable system configuration—the density is greater than the jam density—it is important to implement this region of the plot (region 7 of figure 5.2) due to a quirk in the simulation method. This idiosyncrasy has to do with the applications of constraints

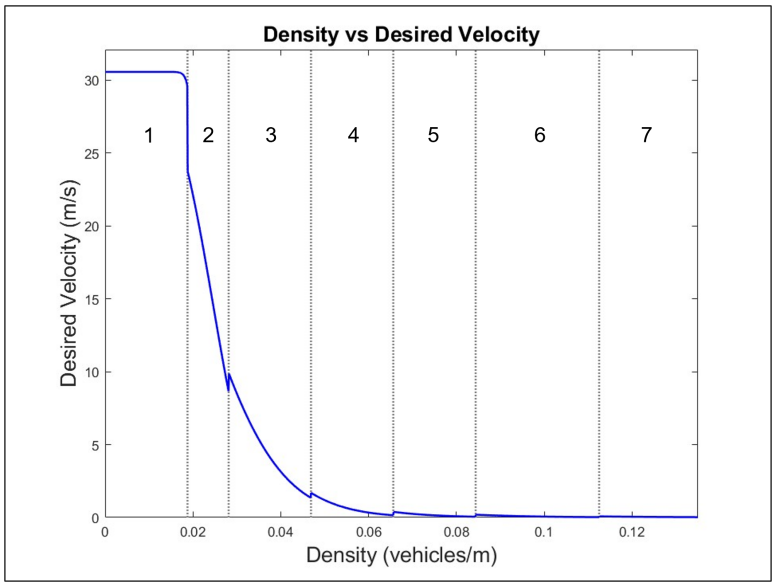

Figure 5.2: Average desired velocity plot for non-constant $a_{m,i}$. The gray dotted vertical lines are successively placed at k_c, $1.5k_c$, $2.5k_c$, $3.5k_c$, $4.5k_c$, and k_j. The free-flow velocity is approximately $30.5555m/s$.

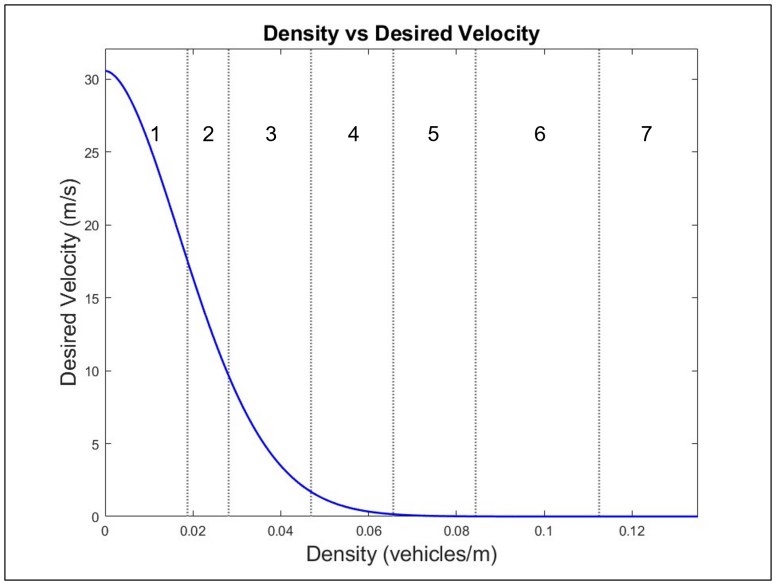

Figure 5.3: Average desired velocity plot for constant $a_{m,i} = 1.8$. The gray dotted vertical lines are successively placed at k_c, $1.5k_c$, $2.5k_c$, $3.5k_c$, $4.5k_c$, and k_j. The free-flow velocity is approximately $30.5555m/s$.

to the simulated system. The specifics of this method will be discussed in detail in section 5.2. However, it can simply be stated here that if a simulation cell exceeds the allowable density, unrealistic as it may be, the simulation must have some value that it can use to complete the calculation.

5.2 Traffic Constraints

In this section, the constraints placed on the traffic system will be discussed. The parameters that must be constrained in the simulations presented here are the density and velocity. Before discussing these constraints themselves, some background on the simulation strategy is needed.

To update the state of this system, the first order differential equations that govern the system (equations (5.2) and (5.4)) must be solved from a given initial condition for a given interval of time. Due to the continuous nature of the equations, a continuous differential equation solver was used to update the state of the system. However, this prevented constraints from being applied by the programmer *manually* in the solution step following a discrete pattern[7]. This was primarily because the solver chooses its own time step in a way that hinders access to the time step by the user. Subsequently, a new method of manual constraint application was developed.

In this new simulation strategy, rather than performing one solve step for a given start and end time, the results of multiple sub-solutions are strung together. For example, if a differential equation is to be solved from $0s$–$10s$, this time is divided into multiple subintervals. The length of each subinterval is determined by the user. For this example, if a subinterval time of $1s$ was chosen, then the solutions of ten, sequential, $1s$-long time intervals would be combined into one answer. These intervals would be from $0s$–$1s$, $1s$–$2s$, and so on, all the way up to $9s$–$10s$. For each of these subintervals, a continuous differential equation is solved in order to update the

[7]The fact that a method was not found to apply constraints *manually* does not mean they do not exist. It can only be said that this proposed method will not work.

system. This is computationally expensive, but the benefit of using this method is that system constraints can be applied manually by the user after each sub-solution. This constraint application can also be done more frequently than if constraints were only applied at the end of one long simulation, and ensures a more accurate representation of the system. In the $0s$–$10s$ example, the constrained solutions to each $1s$-long subinterval are used as the initial conditions for the problem in the next time step. In the following subsections, the density and velocity constraints applied in these intermediate steps are described in detail.

5.2.1 Traffic Constraints: Density

The first density constraint is that the density of a given cell must never be negative. To apply this constraint, cells are checked from upstream to downstream (from left to right in figure 5.1). If a given cell has a negative density, then the density quantity required to make it have a density of zero is siphoned from the downstream cell back into the current cell. Simply put, because the downstream cell accepted more vehicles than was theoretically available to it, it must *restore* them to the preceding cell before the density of the preceding cell can be overwritten to 0. This process continues downstream until the end of the simulation range.

In a similar way, cells are also checked to ensure that they never have density greater than the jam density, k_j. For this constraint, the cells are checked from downstream to upstream (right to left in figure 5.1). If a cell has density greater than k_j, then the excess is added back to the upstream cell, and the density of the cell originally exceeding the constraint is set to k_j. This is because the downstream cell has taken more vehicles than it theoretically is allowed to accept. The constraint described here essentially *stacks* vehicles when they are not permitted to enter downstream cells.

The method in which these density constraints are applied is very important. If one were to only overwrite disallowed cell densities to the nearest acceptable value, there would be discrepancies in how the system updates. This is primarily because

density represents a physical quantity. Therefore, it is not reasonable to *only* overwrite data that does not comply with the density constraint because it allows vehicles to spontaneously appear and disappear from the system. By considering the states of neighboring cells, a representation of the system more rooted in reality can be achieved.

The density constraints discussed here are quite simple: the density must never exceed a fixed set of boundaries. However, the implementation of these constraints is more complex because it takes into account the state of surrounding cells. In the following subsection, the velocity constraints will be discussed.

5.2.2 Traffic Constraints: Velocity

The method of applying velocity constraints on the system is much different than the density constraint application. For each cell, the restrictions are that the average velocity must never be negative, and that the average velocity must not exceed the maximum allowable value. Unlike the density constraints, it *is* reasonable to "cap" velocities that exceed boundaries because velocity does not represent a tangible quantity like density. A velocity is "capped" by overwriting its value to be the nearest acceptable value, without making changes to surrounding cells. Using this method makes the implementation of these constraints simpler relative to the methods of *restoration* and *stacking* used to apply density constraints. However, this constraint is not straightforward to implement because the maximum velocity constraint is not fixed, and depends on the system configuration[8]. The definition of this varying constraint will be discussed in the following paragraphs.

There are multiple ways that the maximum velocity constraint can be defined. One way of defining the maximum velocity is derived from a plot of density versus flow. This plot is constructed by defining a maximum allowable flow for a given cell at each density. In [14], the *triangular fundamental diagram* governs this relationship

[8]The minimum velocity is fixed, and is set to be $0m/s$.

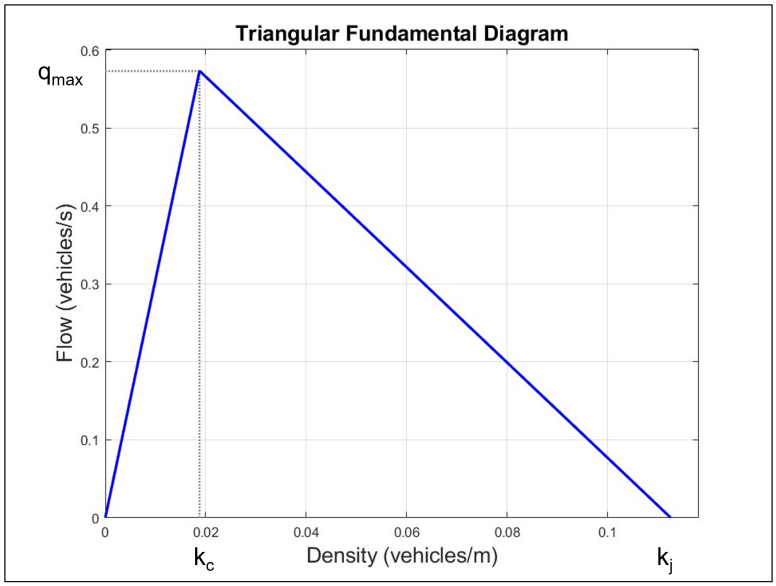

Figure 5.4: Triangular fundamental diagram. The maximum flow, critical density, and jam density are labelled by q_{max}, k_c, and k_j respectively.

between flow and density. A representation of this plot is shown by figure 5.4. Multiple system characteristics influence the formation of the plot that governs this density-flow relationship. The work of [12] in particular examines the impact of using different vehicle types in the system, in the context of this diagram. A version of this diagram is provided in [12]. The authors of [2] also use the triangular fundamental diagram in the development of their own model for a system with connected and autonomous vehicles that test platooning strategies. For the work presented in this chapter, the simple triangular diagram, as described in [2] and [14], will be used. This relationship is plotted in figure 5.4.

Figure 5.4 indicates the maximum potential flow for a traffic cell at its current density value. As figure 5.4 shows, the critical density (k_c) is placed where the maximum flow is achieved [2]. This value is labeled by k_c in figure 5.4. The governing equations

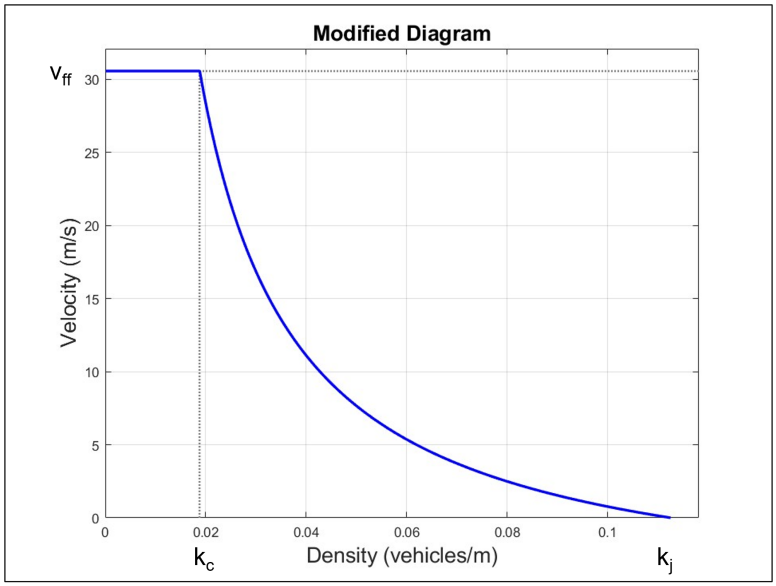

Figure 5.5: Modified triangular fundamental diagram (flow-divided-by-density versus density). The free-flow velocity, critical density, and jam density are given by v_{ff}, k_c, and k_j respectively.

for this plot are given below, based on the description in [2].

$$q_i = k_i \cdot v_{ff}, \qquad \text{if } k_i \leq k_c \tag{5.8}$$

$$q_i = \frac{-q_{max}}{k_c - k_j}\big(k_j - k_i\big), \quad \text{if } k_i > k_c \tag{5.9}$$

Since flow is the product of density and velocity (equation (5.1)), it is also true that flow divided by density gives the velocity. Because of this fact, figure 5.4 can be modified to plot velocity versus density. This effectively plots "flow-divided-by-density" versus density. In effect, this relationship gives the maximum possible velocity for a given density. The plot and equations that govern this relationship are given by figure 5.5 and equations (5.10)–(5.11) respectively.

$$v_i = v_{ff}, \qquad \text{if } k_i \leq k_c \tag{5.10}$$

$$v_i = \frac{-q_{max}}{k_c - k_j}\left(\frac{k_j}{k_i} - 1\right), \quad \text{if } k_i > k_c \tag{5.11}$$

Importantly, if $k_i \leq k_c$, then the maximum velocity is the free-flow velocity, as equation (5.10) shows. Otherwise, the maximum velocity for a given density is given by polynomial equation (5.11). While these relationships are important and useful, they only describe the behavior of a cell in isolation.

One way to address the concept of velocity constraints with respect to other traffic cells is given in [14]. In [14], constraints are defined that restrict the maximum allowable flow that can enter a downstream cell at a given time step. The equation that defines this relationship is given below by equation (5.12), based on the representation in [14].

$$q_i = \min \left[\left(\min \left(v_{ff} k_{i-1},\ q_{max} \right) \right),\ \left(\min \left(q_{max},\ \frac{-q_{max}}{k_c - k_j} \cdot (k_j - k_i) \right) \right) \right] \tag{5.12}$$

Equation (5.12) indicates that the maximum flow allowed into a given cell is the minimum between the potential inflow from an upstream cell and the capacity of the current cell to receive this flow. The potential inflow is given as the left-most argument of the bracketed minimization of equation (5.12). This argument itself is a minimization, and is determined based off equation (5.8) for $i - 1$. Additionally, the capacity of the current cell is given by the right-most argument of the bracketed minimization. This argument is also the result of a minimization and is based off of equation (5.9) for congested flow[9].

Building off these equations from [14], equation (5.12) can be manipulated into a velocity constraint that relates cells to their neighboring cells. Just as equations (5.8)–(5.9) were divided by density to create a set of velocity equations (equations (5.10)–(5.11)) for the plot in figure 5.5, the same is done here to equation (5.12). The

[9]Because the cells in this system are all of the same dimension, all $q_{max,i}$ are equal for any i. Thus, the notion of an index is omitted from q_{max} in equation (5.12).

results of dividing this equation by density are shown below in equation (5.13).

$$v_i = \min\left[\left(\min\left(v_{ff},\ \frac{q_{max}}{k_{i-1}}\right)\right),\ \left(\min\left(\ \frac{q_{max}}{k_i},\ \ \frac{-q_{max}}{k_c - k_j}\cdot\frac{k_j - k_i}{k_i}\ \right)\right)\right] \tag{5.13}$$

The result of equation (5.13) sets the maximum velocity of a cell for a given configuration. It is important to note here that this is only the *maximum* velocity for a given cell in relation to its neighbors. Thus, if the velocity of a given cell is greater than zero and less than the result of equation (5.13), then no constraint is applied on this velocity. Otherwise, a velocity that exceeds this value is overwritten to be the result of this equation.

As this subsection has illustrated, the implementation of the velocity constraint is quite simple—one needs only to directly overwrite disallowed velocities. However, the maximum velocity for each cell itself is more intricate as it dynamically changes with the system. Updating this constraint is crucial to ensuring that the system behaves as close as possible to the intended system. One of the most important implementations of this constraint is related to the boundary traffic cells in this simulation. The characteristics of these boundary cells, among other specifics of the system, will be discussed in section 5.3.

5.3 System Characteristics

In this section, key system parameters and characteristics will be discussed. First, subsection 5.3.1 will discuss the method for updating boundaries of the traffic network. The parameter that governs driver aggression, η, will then be examined in subsection 5.3.2.

5.3.1 System Characteristics: Boundary Cells

When simulating a traffic network, a beginning and end point for a stretch of road must be chosen. Additionally, the manner in which the dynamics of the endpoints evolve must also be established. A simple traffic network with five cells will be used

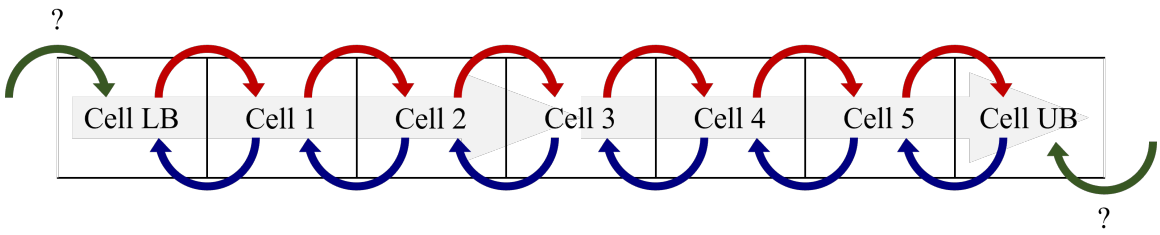

Figure 5.6: Surrounding cell influences. The red arrows indicate the effect of an upstream cell on a downstream cell. The blue arrows indicate the effect of a downstream cell on an upstream cell. The green arrows represent the undefined influences that act on cell LB and cell UB.

as a model in this discussion and is given by figure 5.6. As this figure shows, the term "LB" is used to indicate the Lower Boundary cell on the leftmost side of the diagram. Likewise, the term "UB" is used to indicate the Upper Boundary cell on the rightmost side of the diagram. Cell LB is located upstream, and cell UB is located downstream. The arrows in figure 5.6 indicate the influence of the cells on each other. As this figure indicates, there is an undetermined influence from the upstream direction on cell LB, and an undetermined influence from the downstream direction on cell UB. These influences are important because equations (5.2) and (5.4) require information from surrounding cells in order to update the current cell. Therefore, to fully define the system, a method for updating the density and velocity of these edge-case cells must be defined.

Clearly, augmenting the edge-case cells with more outer cells only exasperates the problem. To establish the influences on the system without augmenting cells LB and UB, a new way of updating these cells must be defined. As subsection 5.2.2 indicated, it is not reasonable to hold the states for these cells constant because of the potential for a fixed velocity state to suddenly exceed the dynamically updated velocity constraint. Therefore, because the states these cells cannot be held uniformly constant throughout the simulation, cells LB and UB were updated according to a constant rule. This way, although the actual number indicating the system state is

not held constant, the method for updating these cells is constant.

The method in use here has several parts. First, because the density constraints *are* constant, the density of cell LB and UB may be fixed with no worry of the density constraints being violated. Also, examining equations (5.2) and (5.4) indicates that there is no downstream velocity dependence. In other words, there is no v_{i+1} term in these equations. Therefore, the velocity of cell UB has no effect on the system. With the density of cells LB and UB defined as constant, and the velocity of cell UB clearly having no effect on the system, the only parameter left to define is the velocity of cell LB.

The method used to update the velocity of cell LB is based on equation (5.13). In this method, the right-most argument in equation (5.13) indicating the "capacity to receive velocity" is calculated for cell 1. After computing the maximum theoretical velocity that can enter cell 1, the velocity of cell LB is set to be that velocity. In effect, this method determines the maximum velocity that can be transferred to cell 1, and forces cell LB to use that as its velocity.

These boundaries can be conceptualized as follows. First, the flow of the cells on the *outside* of cells LB and UB (not included in figure 5.6) are dynamically updated such that the density of cells LB and UB are constant. Then, the velocity of cell LB is updated to be the maximum value it can be while affecting change in cell 1. Lastly, the velocity of cell UB varies to enable it to remain at a constant density. This is made simpler because this value is not included in the state update equations. Though these update rules may be arbitrary, they help put the system on solid footing for the edge cell cases. Therefore, even if the behavior near the outer edges of the system is not desirable, or not what one might expect in a system, it can at least be ensured that this behavior is *not* because of a violation of system principles.

Table 5.3: Table of parameters used in these simulations.

L	λ	v_{ff}	k_c	k_j	a_m	τ	ζ
1000 m	1 $lane$	110 km/hr 30.5555 m/s	0.01875 veh/m	0.1125 veh/m	see tables 5.1–5.2	$3s$	$6.22 \cdot 10^{-6}$ veh/m

5.3.2 System Characteristics: Driver Aggression

Returning to system equations (5.2) and (5.4), parameters must now be chosen for the system. The free-flow velocity, critical density, jam density, and a_m parameters have already been discussed. These parameters were primarily chosen based on work in [13], [15], and [16]. The τ variable is chosen as $\tau = 3s$, based on the work in [16]. This choice of τ is similar to that in [15]. By choosing $\zeta = 6.22 \cdot 10^{-6}\ veh/m$ to be relatively small (like in [15] and [16]), it has little impact on the system outside of ensuring that that if $k_i(t)$ is 0, there will not be a divide-by-zero error in equation (5.4). The system tested here is a one lane system ($\lambda = 1$). This choice of parameters is formally given by table 5.3.

There is a prominent omission from table 5.3 because the η term has yet to be defined. This term was found to be *crucial* to the performance of the system. For reasons that will be made explicit below, η can be thought of as the "driver aggression parameter," and has units of m^2/s. Reexamining the rate of change of velocity equation (5.4) it can be seen that η is a multiplicative constant on the term associated with density. This is highlighted below in equation (5.14), which is an excerpt from equation (5.4).

$$\dot{v}_i(t) = \cdots - \frac{\eta}{L \cdot \tau}\left[-\frac{k_{i+1}(t) - k_i(t)}{k_i(t) + \zeta}\right] + \cdots \tag{5.14}$$

This density term is vital to the determination of the velocity update equation. Since this term is negative, it essentially regulates the speed update based on the downstream density. In other words, if the downstream density (k_{i+1}) is high, then this term detracts more from the speed change. Conversely, a downstream density less than the density of the current cell would make the speed change more positive.

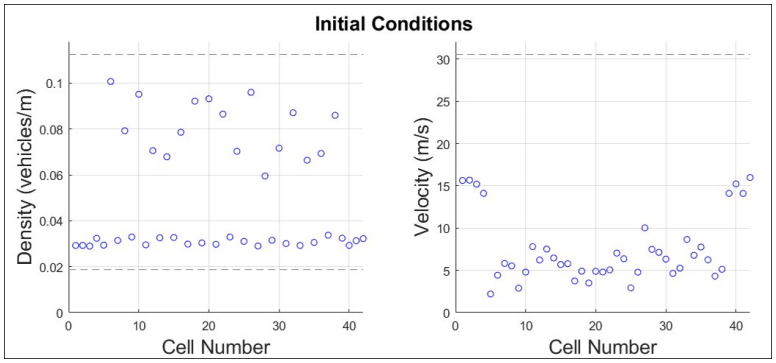

Figure 5.7: Initial condition for simulation in figures 5.8, 5.9, and 5.10. This was chosen such that every other cell in the range of interest was jammed or congested to some degree. Density values were chosen pseudo-randomly, similar to the method used to set initial conditions in case study II. Velocities were chosen based on the densities. This network has 42 cells.

This relationship makes the role of η much more important. Although τ and ζ are tuneable parameters in this term, τ affects other terms in equation (5.4) and the effect of ζ for this equation is relatively obscure compared to the role of η. Therefore, since η has the most direct influence on the impact of the density term highlighted by equation (5.14), its choice is vital to system performance.

Conceptually, if η was very small, that would mean that the velocity update for each cell depends very little on the density of the adjacent downstream cell. If this were the case, vehicles in an upstream cell would not adjust their velocity even if there was clear congestion in front of them. These upstream vehicles would be forced to rapidly decelerate upon entering the downstream cell in order to avoid causing an accident. Based on the initial condition given by figure 5.7 (42-cell network), this behavior is illustrated in figure 5.8 for $\eta = 0m^2/s$.

As figure 5.8 shows, some of the cells immediately become jammed, and over the course of the 45-minute simulation, are unable to return to a more optimal state. This behavior is indicative of *highly aggressive* drivers because vehicles move as fast as they can, without regulating their speed based on their surroundings. This behavior should be modified to be more realistic for the final simulations.

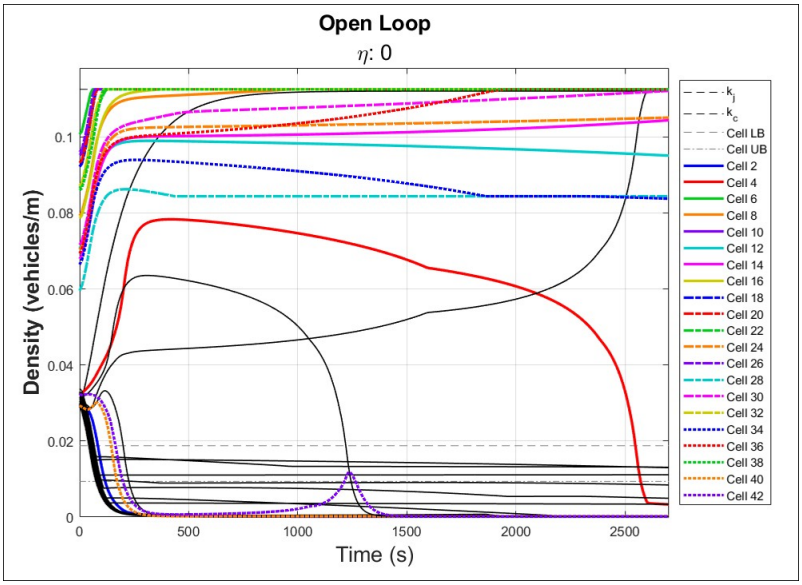

Figure 5.8: Open loop response from a congested state for $\eta = 0m^2/s$. This represents the system behavior when the velocity rate of change has no dependence on downstream cells.

If η was chosen to be extremely large, then the system would behave much differently. Rather than aggressively advancing, the vehicles would all behave much more cautiously because their speed is more dramatically reduced by the density of downstream cells. As a consequence, rather than causing a backup by forcing cells into a jammed state, a backup would be caused by drivers generally moving much more slowly. This behavior is illustrated by the open loop response in figure 5.9 for $\eta = 20000m^2/s$. This plot too used the initial condition given by figure 5.7.

This plot itself presents a problem because if all vehicles in the system behave like this, there may be little effect in adding control to the system. This could be because the vehicles are already travelling at reduced speeds. Therefore, an η parameter was chosen as a compromise between these two extreme values. This value was chosen as $\eta = 5000m^2/s$, and it allows for a relatively minor backup to occur in the system. The open loop simulation for this η value is given by figure 5.10, again using the initial condition from figure 5.7.

In choosing this η and system configuration, the traffic network is now in a state

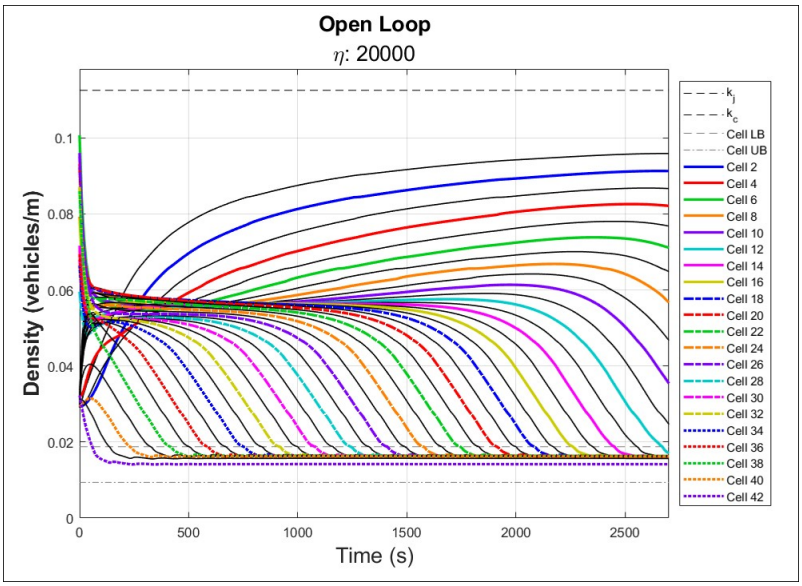

Figure 5.9: Open loop response from a congested state for $\eta = 20000m^2/s$. This represents a system full of drivers that are abundantly cautious so that their velocity greatly depends on the density of the downstream cell.

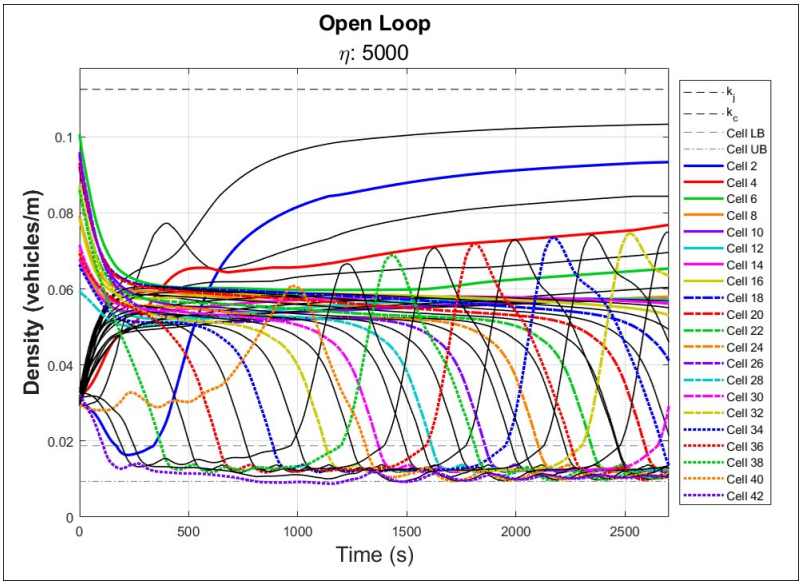

Figure 5.10: Open loop response from a congested state for $\eta = 5000m^2/s$. There is a backup of cars that can be followed down the cells before they reach steady state response.

where it can be helped by the control scheme. Before applying the MPC-only and MPC with consensus ADMM control algorithms to this system, some final details about the system must be made explicit.

5.4 Simulation Method

With the dynamics of the traffic network system defined, the final simulation strategy may now be discussed. These characteristics deal with how this problem is solved, how the system is organized, and how consensus ADMM can be applied most effectively. Subsection 5.4.1 will broadly discuss the method used by the simulations in section 5.5 and 5.6. Subsection 5.4.2 will describe how system agents organized for the experiment in section 5.5. Lastly, subsection 5.4.3 will define how consensus ADMM will be implemented in this system.

5.4.1 Simulation Method: Decision Strategy

The same simulation structures followed in case studies I and II are followed in this case study. Regular MPC control will be used as a baseline for comparison, and the MPC with consensus ADMM control will be tested against this baseline. In keeping with the methods developed in case study I and II, the solve step operates in the following way. First, a system measurement is obtained. Then, a cost function is minimized that characterizes the performance of predicted states of the system (based on the previous system measurement). After finding the optimal control by making several such predictions, the first time step of the selected control strategy is implemented on the system. The system then obtains a new measurement and repeats the process.

Both the MPC-only and MPC with consensus ADMM versions of this method follow the implicit method for determining cost, as outlined in case studies I and II. Thus, the minimization method itself works by running a brief simulation of the system out to a prediction horizon to obtain theoretical measurement data on system

performance for a given control strategy. The resulting performance cost of this command is then determined according to simple cost function methods. This means that a prediction simulation is implicitly run each time this cost function is called in the minimization step.

The MPC with consensus ADMM version of this code will have an augmented term to the cost function, as it did in case study II. However, the basic form of the cost function in use here is given below by equation (5.15).

$$J = \Sigma_{i=1}^{N} \|k_i - \text{ref}\|_2^2 \tag{5.15}$$

In equation (5.15), the variable k_i is the prediction data vector for cell i. When consensus ADMM is in use, this equation will have terms augmented to it similar to the representation in figure 4.2 in section 4.2.3. Although not noted in equation (5.15), the cost function was normalized through the multiplication of a constant so that if a density vector deviated from the reference by the maximum allowable value, the resulting cost would be 1.

An important nuance to the prediction step has to do with constraint application. In the actual simulation step used to update the state of the system, many shorter simulations are run with constraints applied after each implementation, as detailed in section 5.2. However, chaining together the results of many simulations in this way would be too expensive computationally to include in the prediction step. Therefore, rather than waste computational resources finding a relatively more realistic prediction simulation output, no constraints are applied here. In addition, only *one* solver is used in this step, rather than many solvers with results chained together. This saves computational resources at the expense of potentially finding an answer that might not be the most ideal.

This method will help stave computational resources. It is also not guaranteed

that the lack of constraint application in the prediction simulation will have a negative impact on the system. This is because the reference values chosen for the cost functions will always be feasible values. In other words, the system is not asked to violate constraints in order to achieve a reference. Although there might cases where not applying constraints hurts the effectiveness of the calculated control command, it stands to reason that violating constraints may be enough of a tax on the prediction cost that it deters the controller from choosing a command that would lead to a constraint violation. By this reasoning, it is not necessarily guaranteed that leaving out constraints from prediction simulations is a detriment to the system performance.

Before concluding this discussion on decision strategy, there are a couple of final points to discuss. First, this strategy is made computationally simpler by using a short prediction horizon, control horizon, and time step. This choice was made in part because the sheer level of computational complexity for this system is much higher than for the system in case studies I and II. Lastly, it is important to note that the solver used here is not guaranteed to find a global minimum, and might get stuck in a local minimum. The authors of [3] discuss a method to help avoid such a scenario, by strategically selecting an initial optimization values. However, no such method is used here. In subsection 5.4.2, final notes about the system will be discussed in preparation for the later simulation analysis.

5.4.2 Simulation Method: Agent Profile

In this subsection, the manner in which the MPC with consensus ADMM algorithm is applied on this system will be discussed. The discussion here parallels that of section 4.3 in case study II. As with the interconnected cart example, consensus ADMM in the context of a traffic network seeks to break the minimization computation down into many subcomponents of the larger problem.

For the interconnected cart example in case study II, consensus ADMM was relatively straightforward to implement. Each cart—or agent—came up with a control

strategy for itself and its adjacent neighbors, leading to the formation of a nearly uniform communication network. The traffic network system is not as straightforward. While an agent in the 10-cart system was defined as a single mass, capable of inputting forward and reverse thrust, an agent in a traffic network is not considered to be a single cell. This is primarily due to the broader characteristics of the dynamics of this system. As discussed in section 5.1, the digitally updating speed limit can only tell vehicles in a given cell to slow down. Under this restriction, the only direct way for a system to increase the speed of the vehicles in a given cell would be to apply no control on this cell. A method that can more effectively increase the speed of a given cell relative to the open loop response could improve controller flexibility.

One way to take advantage of the system structure is by including neighboring cells in an agent's control range. If one wanted to speed up the vehicles in cell i, there are three primary places that control can be applied. These places are cell $i+1$, cell i, and cell $i-1$. As discussed in subsection 5.3.2, an increase in the density of cell $i+1$ would result in a direct decrease in the velocity of cell i, as shown by equation (5.4) (specifically due to the term highlighted in equation (5.14)). Thus, cell $i+1$ is not necessarily the best place to apply control to increase the velocity of cell i. As discussed in the preceding paragraph, applying control on cell i itself is also not a feasible option because applying control on cell i directly lowers the velocity of the vehicles in this cell. There is potential in applying control on cell $i-1$ though.

If the speed limit of cell $i-1$ is reduced by the controller, that would result in a reduced outflow from cell $i-1$. In a sense, this relieves pressure on cell i, allowing its speed to be increased. This strategy will be employed in the simulations that follow. This also is a key system characteristic because it adds another dimension of control to cell i. Instead of only being able to slow down the vehicles in cell i, a method now exists that indirectly speeds them up quicker than is theoretically possible under the open loop response.

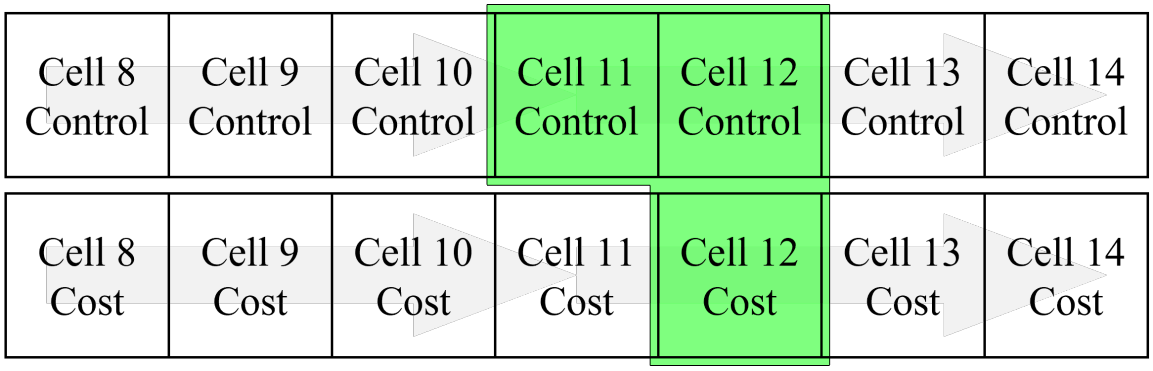

Figure 5.11: Diagram of a single agent (cell couplet). Vertically aligned cells only represents a single cell in the traffic network. In this diagram, cells 11 and 12 coordinate control strategies to help cell 12 better achieves the reference.

With this knowledge, an agent in the system will be defined as being a couplet of two traffic cells. Each cell couplet determines its performance cost based only on the performance of the downstream cell in the couplet. This allows the agent (cell couplet) to take advantage of the control principle discussed in the previous paragraph. A representation of this style of agent is given by figure 5.11.

In case study II, an agent was defined as a single cart. As noted here, an agent in case study III is defined as a cell couplet. When the MPC-only control algorithm is applied on this system, it will control each cell in the cumulative range of all cell couplets from a single optimization. Even in this case, the cost of the applied control is based only on the downstream cell of each agent. For a continuous definition of non-overlapping agents in a system, this equates to basing the cost on every other cell. Based on modular structure, the application of consensus ADMM in this system can now be discussed.

5.4.3 Simulation Method: ADMM Definition

As stated in [7], ADMM in this application can be thought of as a communication algorithm. In case study II, the method for communicating messages was relatively simple to define. In this 10-cart example, each agent determined a control plan for itself and its neighbors, and communicated this plan to each agent in its effective

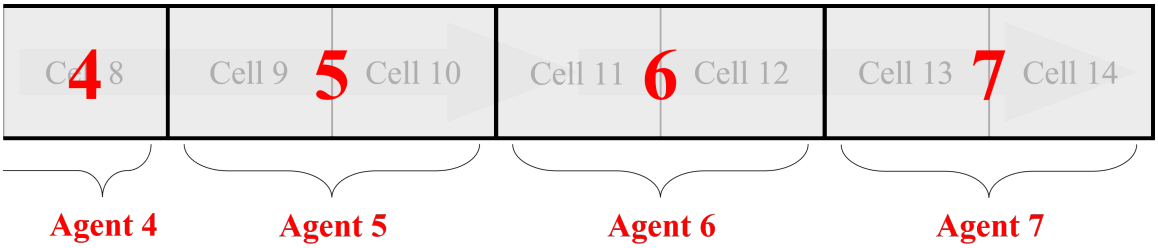

Figure 5.12: Agent definition for an arbitrary traffic network. An agent is composed of two cells.

range. The communication network in this case study is slightly different.

As stated in subsection 5.4.2, an agent in the system is defined based on the notion of a cell couplet. Comparing the 10-cart system in case study II and the traffic network here, one could say that a single cart (or agent) is equivalent to a single cell couplet (which is also an agent). In this system, there are no overlapping couplets. Therefore, if there are 10 cells in a traffic network, the maximum number of agents in this system is given by $10 \div 2 = 5$ cells. This principle is sketched in figure 5.12.

The communication network established for ADMM will build upon this notion of agents being non-overlapping cell couplets. As a point of reference, case study II examined a 10-cart system that subsequently had 10 agents. A "processor" here was defined as the control range for which a single agent computed a control plan. Based on this definition, there were 10 processors for this system because all 10 carts came up with a local control plan. Carts 2–9 created a local control plan that involved 3 carts in total, one for itself and its two neighbors. Carts 1 and 10 created plans for only two carts in total because they each had only one neighboring cart.

For comparison, a traffic network with 10 agents will be examined. A 4-cell buffer will be implemented on either side of the 20 cells that constitute the 10-agent system[10]. This effectively creates a 28 cell system, where the middle 20 cells constitute the 10 agents. A processor in this network will be defined differently than the mass-spring-

[10]This will allow the system dynamics near the edges of the network to evolve more naturally so that the system behavior near the boundary cells is not overly affecting the system.

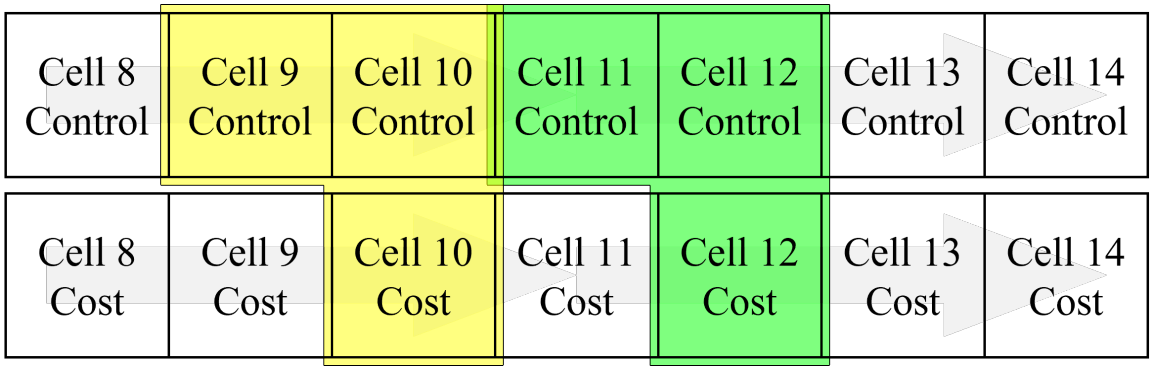

Figure 5.13: Representation of a single 4-cell processor in a traffic network. This processor is "located" at cell 12, and uses cells 9–12 to compute a control plan that minimizes the performance cost of cells 10 and 12.

damper example. Rather than each processor computing a control for the upstream agent, current agent, and downstream agent, a local control plan will only consist of control on the upstream and current agents. As a result, instead of having 3 overlapping control plans, each agent (except the first/last agents) will have only 2 overlapping control plans. This will save on the computational expense of this method. A diagram of this processor definition is given by figure 5.13.

One way of conceptualizing this system as directed by figure 5.13 is by considering that only a yellow box and green box can overlap in *this* 4-cell processor configuration. There should be no overlap of green boxes among different processors in this system. In the example given by figure 5.13, the processor is said to be "located" at cell 12. This cell can be considered the focal point of the processor because the offshoot highlighted cells to the left are determined relative to their proximity to cell 12. Furthermore, this processor would create a control plan to improve the cost of cells 10 and 12 by using cells 9 through 12. The control plan that this processor makes for cells 9 and 10 would be shared with the processor that is located at cell 10. Additionally, this current processor accepts a control plan for cells 11 and 12 from the processor located at cell 14.

It is important to note here that each processor bases its control plan only on data

about itself, and on the disturbance data of cells that neighbor this processor. This is noted in the highlighted areas of figure 5.13. As with case study II, the data from the cells that surround a processor will be counted as a disturbance. Furthermore, as a processor performs a minimization, it will dynamically update the disturbance cells as if they were cells LB and UB. In other words, throughout the prediction step, the density values of the surrounding cells will be held constant. Meanwhile, the velocity of the upstream disturbance cell will be dynamically updated to provide the maximum velocity possible to the first cell in a processor. In the example system given by figure 5.13, cells 8 and 13 would be treated as if they were cells LB and UB respectively in this processor's local control plan computation.

The range of cells whose full dynamics are actively updated in a local control plan computation can be considered the processor's "sight." This "sight" of a system describes what data a processor has available to it. This is a key concept that must be considered when comparing the MPC-only algorithm with the MPC with consensus ADMM algorithm. To make sure that both methods are roughly equivalent, the MPC-only method uses the cumulative sum of each consensus ADMM processor's sight for a given system layout. This means that if two consensus ADMM processors have a sight of cells 9-12 and around 11-14 (note the overlap on cells 11 and 12), then the comparable MPC-only algorithm will control cells 9-14 from a single optimization step[11].

Success of this consensus ADMM method hinges on the computational expense for solving these processors' optimization problems individually as being collectively cheaper than solving an analogous problem with centralized MPC. If this is the case, assuming no dramatic drop in system performance, then this method of distributed computation method will have been shown to be more practically feasible than the original MPC-only method. This would mean that this system is able to compute

[11]For case study II, the cumulative system sight was equal to the full system. This allowed the MPC-only algorithm to do a single full optimization for the whole system.

and apply control commands faster, and that the time between system measurement and control implementation would be smaller. This is beneficial for a real-world system because the system would evolve much less in this time gap before control is implemented.

Furthermore, the MPC with consensus ADMM method can measure computation time based on the maximum processor solve time from each time step. In other words, if processor 8 solves its local control problem faster than processor 9, the computation time from processor 8 need not be counted in the elapsed time measurement. This is because the first minimization step of consensus ADMM could theoretically be done in parallel before results are combined in the averaging step. While the simulations presented in case study II and the simulations that will be presented in case study III were all done serially[12], data was gathered at each computation time step that allowed for such a comparison to be made.

5.5 Experiment 1 – Variable Simulation Window

In this section, the collective results of applying the MPC-only and MPC with consensus ADMM algorithm on a traffic network will be discussed. First, a smaller-scale simulation will be examined in order to establish a baseline by which to compare these two algorithms. Then, building off of this result, several simulations will be examined that each have different organizational layouts covering different windows of control. For all of the MPC with consensus ADMM simulations that will be presented here, the maximum allowable number of ADMM iterations will be set to 2, based on the results of case study II. Comparing the data from all of these simulations will offer insight into the performance of these algorithms.

[12]These cases studies implemented the consensus ADMM algorithm serially by computing the local control plan for each processor one at a time, before combining these results in the averaging step.

Table 5.4: Parameters used for simulations throughout section 5.5 and 5.6.

L	λ	v_{ff}	k_c	k_j	a_m	τ	ζ	η
1000 m	1 $lane$	110 km/hr, 30.5555 m/s	0.01875 veh/m	0.1125 veh/m	see tables 5.1–5.2	$3s$	$6.22 \cdot 10^{-6}$ veh/m	5000 m^2/s

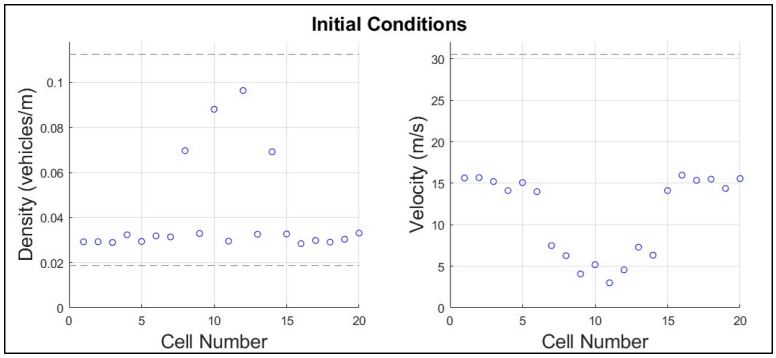

Figure 5.14: Initial conditions for the sample problem in section 5.5.1. The upper and lower dashed lines of the density initial condition plot are the jam and critical densities, respectively. The dashed line in the velocity initial condition plot is the free-flow velocity. Density values were chosen pseudo-randomly, similar to the method used to set initial conditions in case study II. Velocities were chosen based on the densities.

5.5.1 Experiment 1: Initial Implementation

In this subsection, a 20-cell traffic network is simulated and controlled. Key system parameters are given in table 5.4. These parameters are the same that were originally used in subsection 5.3.2 in the testing of different η parameters. The initial condition used for this simulation is given by figure 5.14. This initial condition was chosen so that the particular cells whose cost will be calculated were especially congested, with the remaining cells congested to a lesser extent. The initial velocity of each cell was chosen to be close to the maximum allowed given the system configuration.

The decision interval for this system was chosen to be $60s$, with the prediction and control horizons set to $p = 3$ and $m = 2$ respectively. In the prediction simulations that take place in the solve step, data was obtained based on smaller interval than the $60s$ time step used for making the control strategy. Therefore, although control commands were updated every $60s$, the predictions themselves used a time step of

$10s$. This follows a strategy used in [13], which is also briefly alluded to in [3] and [4]. This strategy was chosen so that the foresight of the prediction could be increased without sacrificing simulation detail. This method does not require the control horizon to be increased to achieve this greater level of prudence[13]. Using this strategy, a relatively short prediction and control horizon were chosen for the sake of computational efficiency. It should be noted that a longer prediction and control horizon would likely give better performance, although at the expense of computation time.

Processors in this simulation were defined as they were in subsection 5.4.2. Each processor contained four cells, with the processor being indexed according to the furthest downstream cell. Each processor can be thought of as the grouping of two adjacent cell couplets (agents), meaning that each processor had two control objectives, and four cells from which control could be applied to achieve these objectives. Furthermore, processors were placed in intervals of two cells, meaning that each processor had at least one cell couplet overlap with another cell couplet from a neighboring processor. This is shown in figure 5.15. It should also be noted that the disturbances to each processor (as shown by the red arrows in figure 5.15) were the most recent density values, but with dynamically updating velocities. In other words, the simulations that each processor ran to determine its control command were based off of static boundary densities, but the velocity of the upstream disturbance was updated dynamically to be the maximum allowed by the constraints discussed in subsection 5.2.2.

In the following simulation, cells 8, 10, 12, and 14 are the reference cell locations used for four processors. The cells whose cost will be calculated that fall in this processor's window of control are cells 6, 8, 10, 12, and 14. In other words, starting from the congested initial condition given by figure 5.14, the traffic network must

[13]If the control and prediction horizon were based on the smaller $10s$ time step, an increase in p and/or m would be required to achieve a more prudent control, adding to the computational complexity. The varying time step method takes advantage of having a more detailed prediction, without adding so much to the computational load.

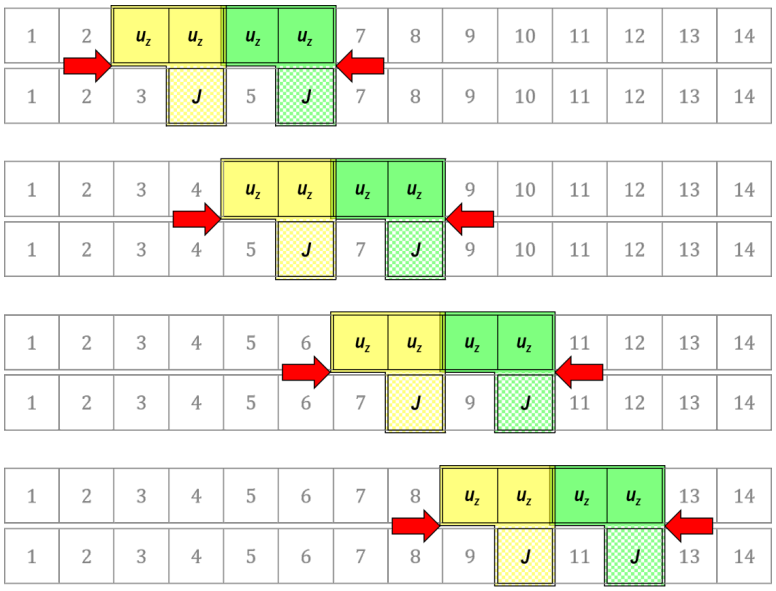

Figure 5.15: Mapping of four, 4-cell processors in an example system. The red arrows represent disturbances to each processor. This figure represents a similar concept to figure 4.3.

apply control on the system so that cells 6, 8, 10, 12, and 14 have their densities reduced to the critical density. The open loop response for this system is given by figure 5.16. As noted in this figure, the overall cost of performance for this response is 0.7377. This cost is calculated based on the performance of every other cell, starting from cell 2 and ending on cell 20, and it is a measure of how the overall system performed in achieving the reference value of k_c.

MPC was applied to the simulation, giving a final cost of performance of 0.60456, which is better relative to the baseline open loop response (0.7377). Figure 5.17 displays the density data of this system as well as the control inputs. This simulation based its sight and control layout on the one that will be used in the MPC with consensus ADMM method. Inspection of figure 5.17 shows that MPC initially succeeds in more quickly bringing the density of cells 10, 12, and 14 closer to the reference relative to the open loop performance. However, there is some undesirable variability in this performance. While cells 6 and 8 especially struggle to achieve the reference, the reduced cost indicates better overall performance of the whole system.

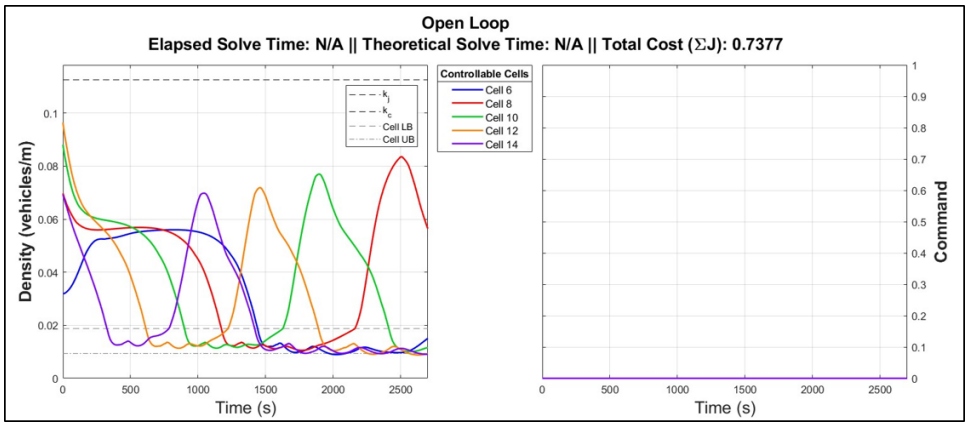

Figure 5.16: Open loop response, 20-cell system. The overall cost for this system is calculated based on every other cell, starting from cell 2 and ending on cell 20. Data only shown for cells whose cost is being calculated.

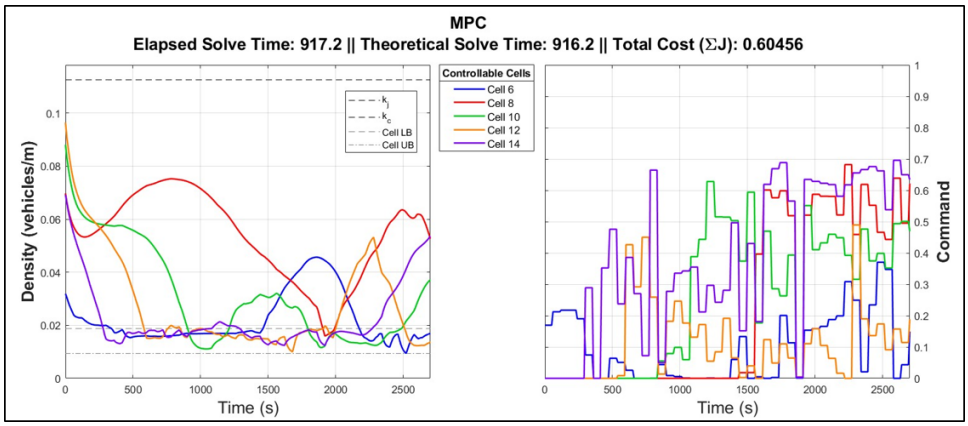

Figure 5.17: MPC-only control algorithm. This method improves on the overall cost of the open loop response. Data is only shown for cells whose cost is calculated in the prediction steps.

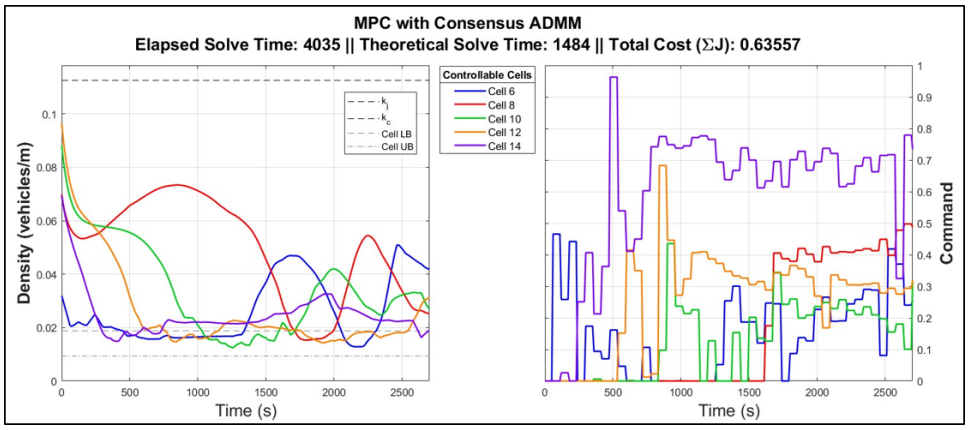

Figure 5.18: MPC with consensus ADMM control algorithm. This method improves on the overall cost of the open loop response, but is not better than the MPC-only response. Data is only shown for cells whose cost is calculated in the prediction steps.

Lastly, the MPC with consensus ADMM version of this method was applied with results shown in figure 5.18. The ρ parameter for the consensus ADMM portion of the cost function was held constant at 10, and the density cost function weight was chosen to be 80000. The cost describing this method's performance is 0.63557. Similar comments to those made about the performance of the MPC-only method can be made about the performance of this algorithm.

As figures 5.16–5.18 demonstrate, both the MPC-only and MPC with consensus ADMM versions of the method improve on the overall cost performance of the system. The traffic back-up shown in figure 5.16 is removed to some degree. However, this performance is far from perfect. Figures 5.17 and 5.18 indicate that the system was able to reel in some of the unfavorable performance, but not all of it. In particular, both of the methods struggled to make some cells remain at the k_c reference. However, the performance cost was improved overall in each of these control methods. Having established that the control improved the behavior of the system, although not perfectly, the computation times can now be examined.

As figure 5.17 indicates, the MPC-only algorithm took approximately 917s to run. In this run time, approximately 916s was taken to run the code minimiza-

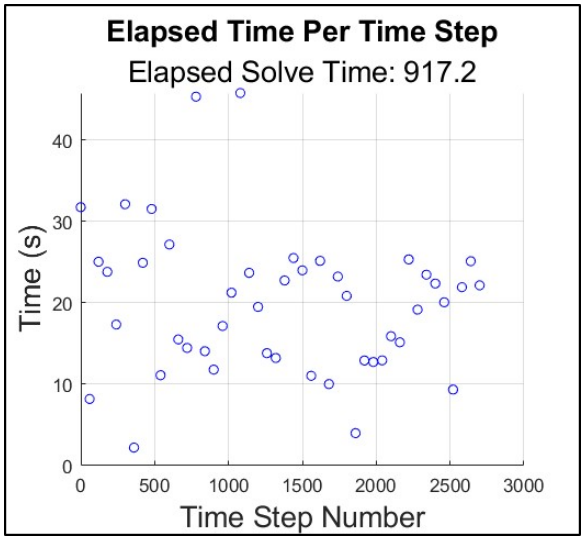

Figure 5.19: MPC-only timing data showing solve time per time step in the simulation.

tions/predictions. In other words, very little time was wasted executing other components of the program outside of the minimization. For this reason, the "Elapsed Solve Time" shown in figure 5.17 is the primary performance metric for system timing. The solve times for this MPC-only system are shown in figure 5.19.

The MPC with consensus ADMM method performed reasonably well with respect to cost, but the serial (overall) elapsed time was approximately $4035s$. This time is representative of the amount of time it took to run this algorithm from start to finish without distributing the work of each processor at each time step. As noted previously, the "Theoretical Solve Time" for this system was then calculated by taking the overall elapsed time for the serial computation ($4035s$), and subtracting extraneous timing data from it. The extraneous timing data is defined as the cumulative solve time of non-relevant processors from each solve step. The adjective "non-relevant" in this sense describes processors whose solve time was less than the maximum processor solve time in a given time step[14]. Including only the performance-relevant timing

[14]For example, in a two processor network, if processor 1 took $10s$ to solve its minimization problem, and processor 2 took $5s$ to solve its minimization problem, then the total theoretical computation time would be $10s$. This eliminates the "non-relevant" timing data from processor 2 because it performed well enough that it would not delay the system.

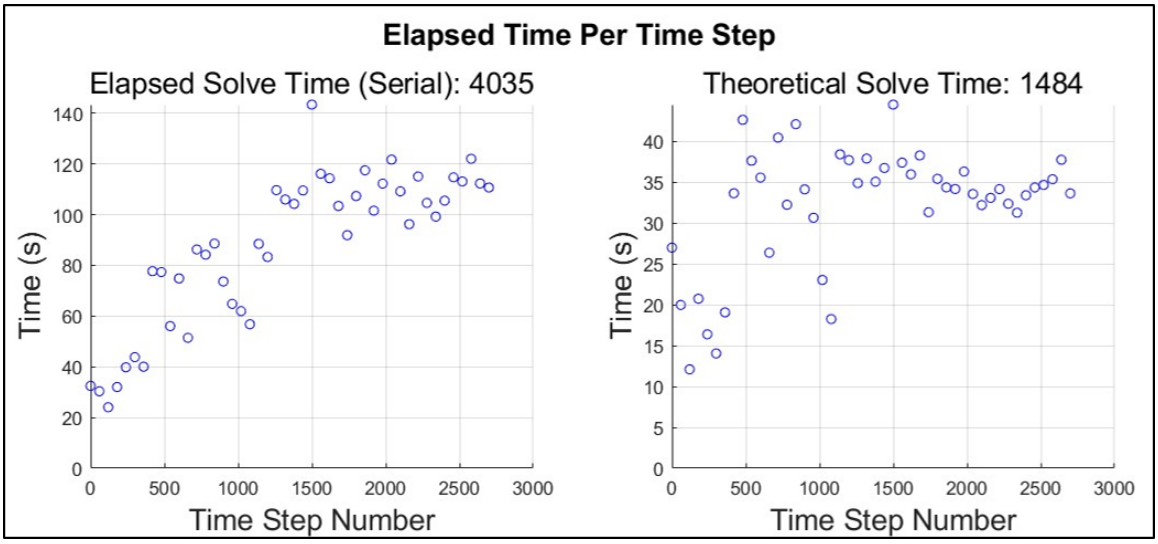

Figure 5.20: MPC with consensus ADMM timing performance. The plot on the left represents the total serial evaluation time at each time step of the simulation. The plot on the right indicates the maximum solve time out of each of the four processors per time step.

data indicates that the MPC with consensus ADMM calculation took approximately $1484s$. This performance is shown in figure 5.20.

Based on this timing data, the MPC with consensus ADMM algorithm was worse in every key metric relative to the MPC-only strategy. However, this is a relatively simple system, which shows that using a distributed computation method on such a relatively simple system is not the best strategy. This may be because each local computation does not maximize its potential effectiveness.

Having established that the performance of this system improved under both control methods, the system can now be adjusted and made more difficult to control. In subsection 5.5.2, it will be shown that the MPC with consensus ADMM algorithm outperforms the MPC-only algorithm with respect to theoretical elapsed time for more complex systems. The plots in figure 5.19 and figure 5.20 will become more prominent in subsection 5.5.2.

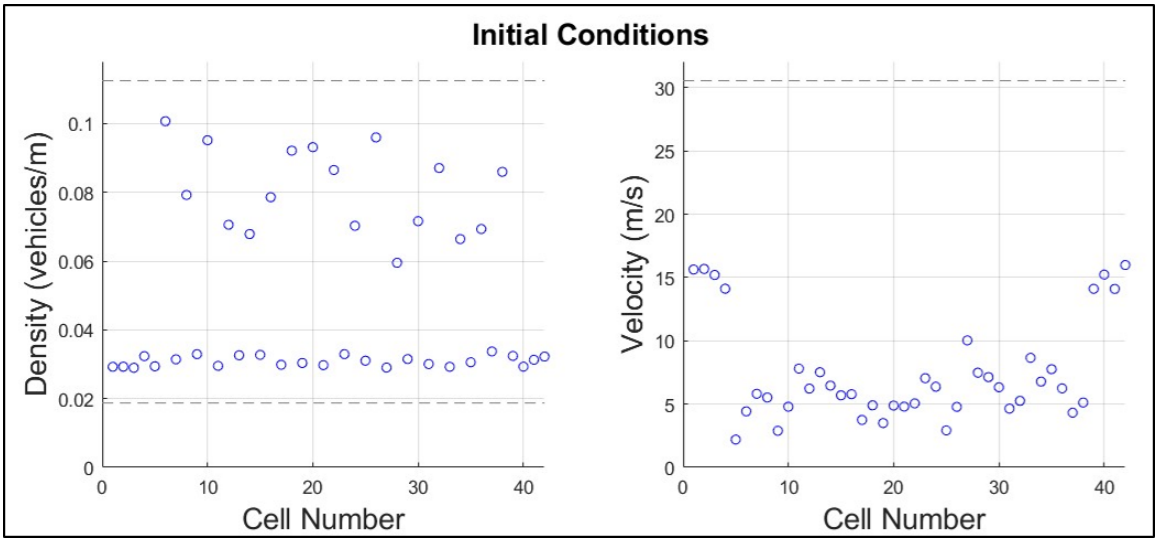

Figure 5.21: Initial condition for the simulations in subsection 5.5.2. The upper dashed line of the plot on the left is the jam density, and the lower dashed line is critical density. The dashed line on the right side plot is the free-flow velocity. Density values were chosen pseudo-randomly, similar to the method used to set initial conditions in case study II. Velocities were chosen based on the densities. This network has 42 cells.

5.5.2 Experiment 1: Complex Simulations

The simulations examined in this subsection use a 42-cell traffic network. The initial condition on this system is the same for each simulation, and was first noted in figure 5.7. The initial condition is repeated in figure 5.21 for convenience. The open loop response of this system is also included below in figure 5.22.

The same system parameters, MPC parameters, processor definition, cost functions (weights), and ADMM ρ values from subsection 5.5.1 were used again for these simulations. A key difference between this set of simulations and that of 5.5.1 is that the initial condition of the simulations in this section is *much* more congested. This means that there is a lot of capacity for this system to improve its performance. Because of this, it could take longer than the 45-minute simulation allows to see the final responses of the system. Therefore, in these simulations, a successful control strategy is defined as one that simply reels in and mitigates undesirable open loop

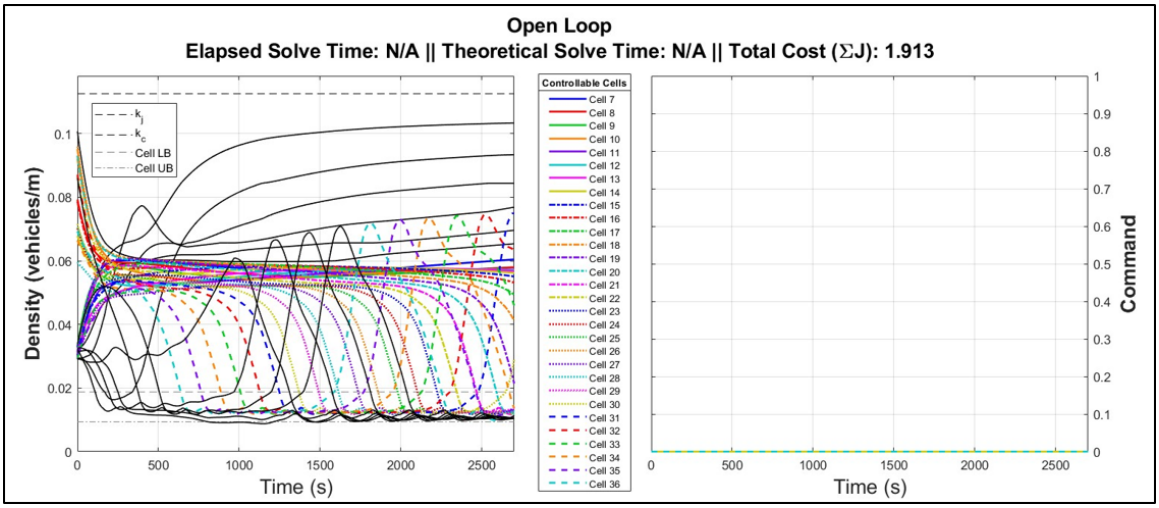

Figure 5.22: Open loop response for simulations in subsection 5.5.2. Data for all cells included.

performance characteristics.

Tables 5.5 and 5.6 display data from each of the 8 simulations performed here. The number of processors and their location for each simulation are shown in table 5.5. The density and command plots from trials **D** and **G** are included here. In addition, the timing plots from trials **C**, **D**, **G**, and **H** are included as well. It is important to note again that the initial conditions were the same for each simulation. Therefore, some of the smaller simulations with fewer processors struggled to manage this large amount of congestion relative to some of the other simulations. The results shown in these plots and tables will be discussed in detail in subsection 5.5.3.

5.5.3 Experiment 1: Discussion

There are several key points to make about the behaviors depicted in the plots from figures 5.23–5.26. First, while the controller naturally struggles to obtain the reference of k_c for such a high level of congestion, it is important to note that primarily the cells at the end of the control range seem to be impacted more immediately. This could be because being at the end of the control window gives these cells the advantage of having each of the previous cells before them reduce their speed. Although ideal

Table 5.5: Experiment 1 setup data with preliminary results. The final costs shown here are calculated based on every other cell, starting at cell 8 and ending on cell 38.

Trial	Number of Processors	Processor Location (cell)	Controllable Cells	Open Loop Cost	MPC Cost	ADMM Cost
A	2	22, 24	19–24	1.913	1.8854	1.8819
B	4	20, 22, 24, 26	17–26	1.913	1.8984	1.882
C	6	18, 20, 22, 24, 26, 28	15–28	1.913	1.7925	1.781
D	8	16, 18, 20, 22, 24, 26, 28, 30	13–30	1.913	1.752	1.7626
E	10	14, 16, 18, 20, 22, 24, 26, 28, 30, 32	11–32	1.913	1.7677	1.7409
F	12	12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34	9–34	1.913	1.7081	1.6656
G	14	10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36	7–36	1.913	1.6848	1.651
H	16	8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38	5–38	1.913	1.7044	1.6902

Table 5.6: Approximate timing data, experiment 1. "MPC Time" is the elapsed time for the MPC-only algorithm. "ADMM Serial Time" is the total time for doing the MPC with consensus ADMM algorithm. "ADMM Theoretical Time" is the *ideal* time for the MPC with consensus ADMM algorithm, calculated only based on the maximum processor solve time per time step.

Trial	MPC Time (s)	ADMM Serial Time (s)	ADMM Theoretical Time (s)
A	413	930	648
B	899	1953	924
C	1932	2784	1028
D	2357	3825	1188
E	6399	4488	1272
F	4388	5616	1422
G	5406	7433	1680
H	8249	7854	1651

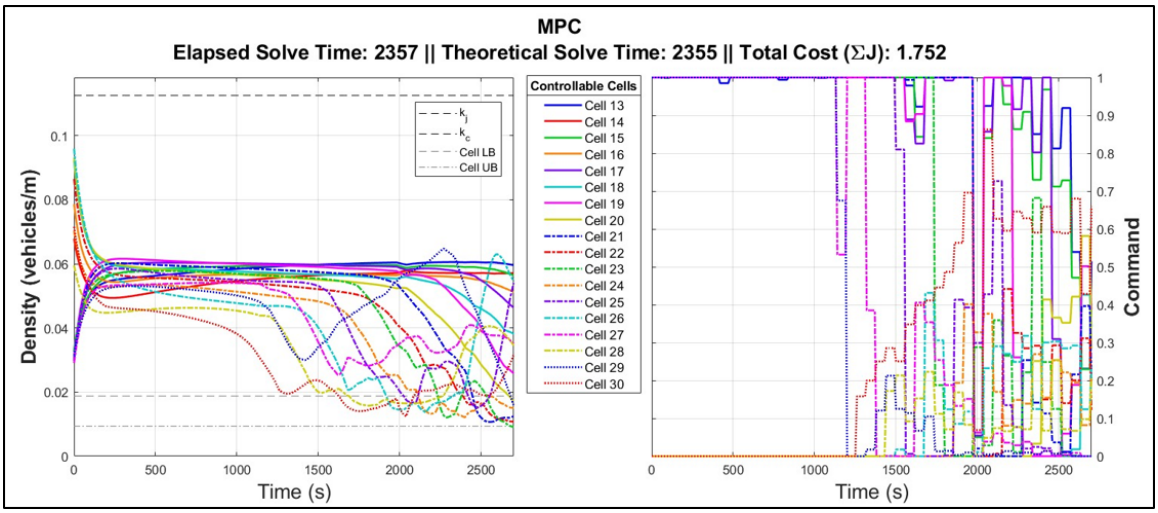

Figure 5.23: Trial **D** data for the MPC-only method.

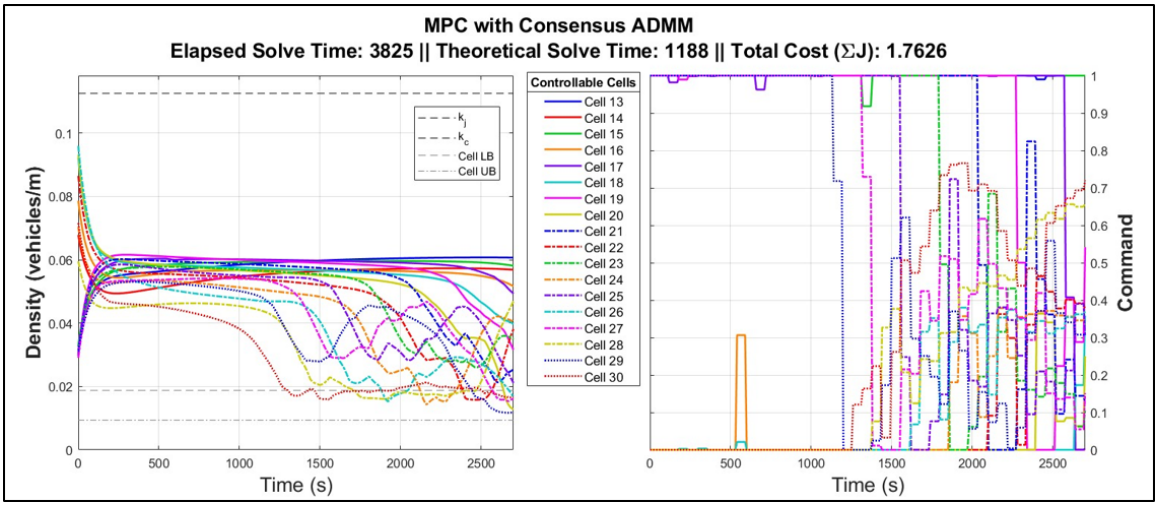

Figure 5.24: Trial **D** data for the MPC with consensus ADMM method.

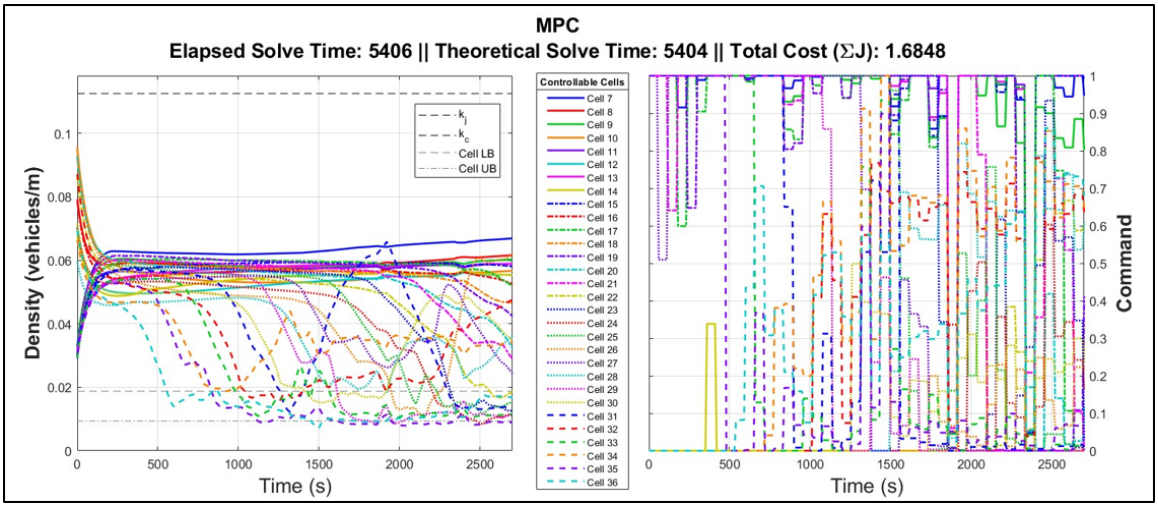

Figure 5.25: Trial **G** data for the MPC-only method.

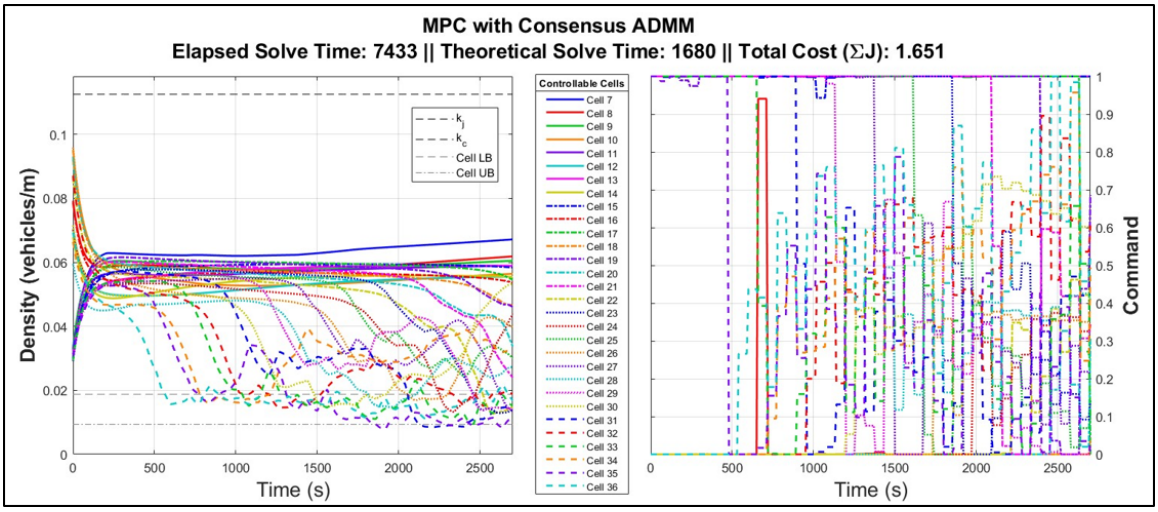

Figure 5.26: Trial **G** data for the MPC with consensus ADMM method.

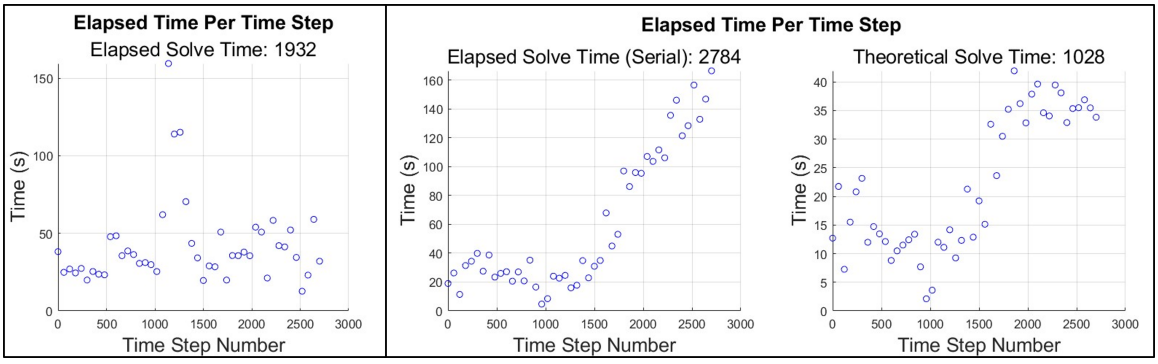

Figure 5.27: Trial **C** timing information. The leftmost plot is the MPC timing data. The two plots on the right are the ADMM timing data.

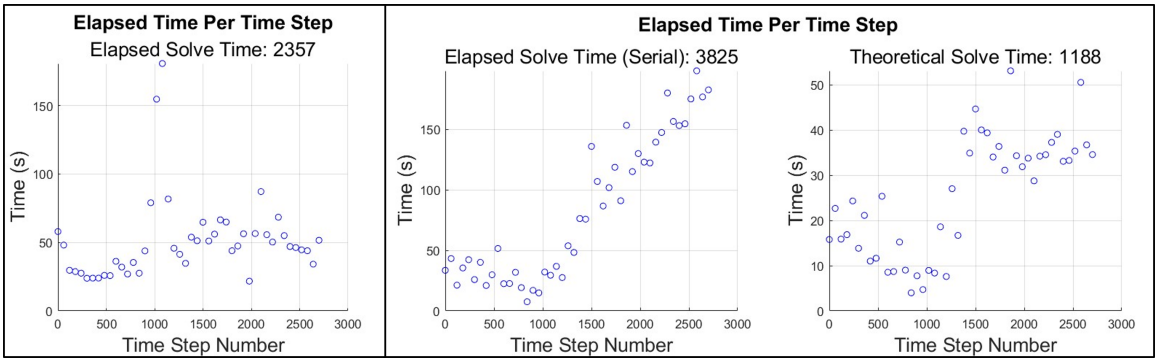

Figure 5.28: Trial **D** timing information. The leftmost plot is the MPC timing data. The two plots on the right are the ADMM timing data.

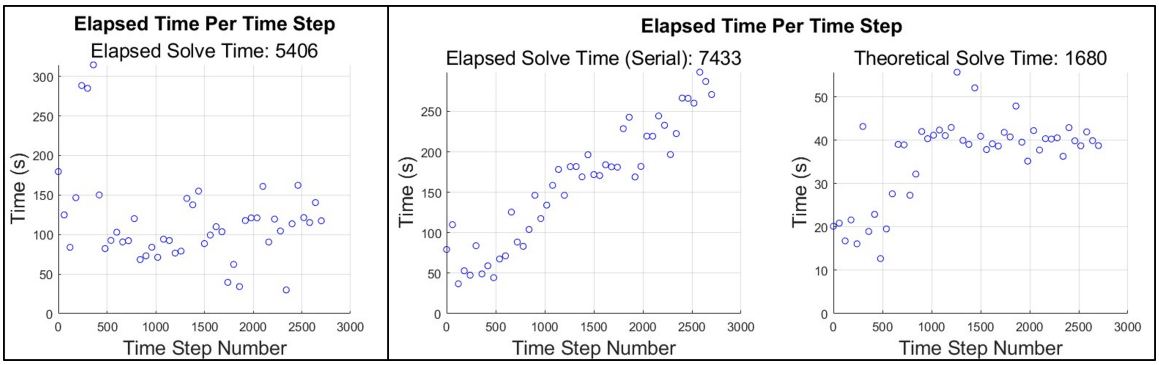

Figure 5.29: Trial **G** timing information. The leftmost plot is the MPC timing data. The two plots on the right are the ADMM timing data.

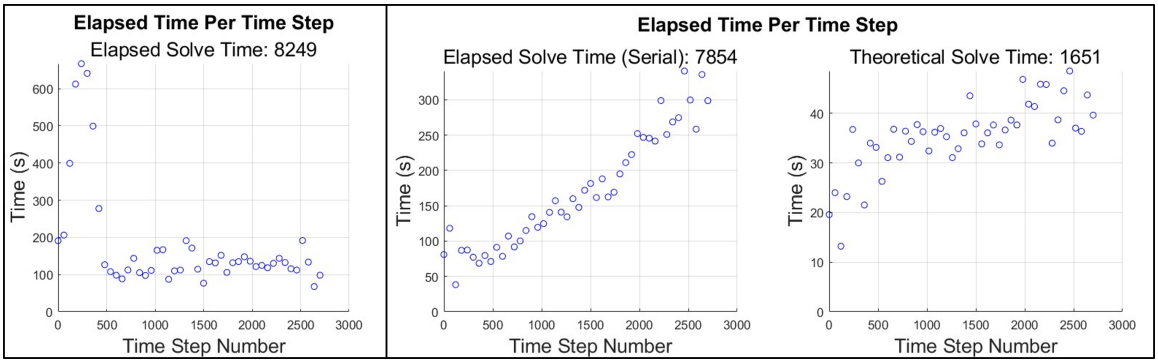

Figure 5.30: Trial **H** timing information. The leftmost plot is the MPC timing data. The two plots on the right are the ADMM timing data.

performance is not achieved within the 45-minute simulation window, the MPC-only and MPC with consensus ADMM algorithms remove the sharp congestion spikes seen in the open loop response.

This behavior is reflected in each cost of the controlled systems (see table 5.5) being less than the open loop cost. This shows that even from an undesirable initial condition like this, the control strategies do help overall system performance. Furthermore, each of the costs from a controlled system shown in table 5.5 are competitive with each other. That is, the MPC-only and MPC with consensus ADMM costs are roughly the same for each trial. This is important because it establishes that each method has a relatively similar level of performance. Because of this, the discussion may now shift to the primary metric of interest: computation time.

Since the performance of the MPC with consensus ADMM algorithm has been established as roughly the same as the MPC-only algorithm, relative success of the MPC with consensus ADMM algorithm depends on computation time. The overall timing data for each trial is given in table 5.6. Figures 5.27–5.30 show more detailed data for trials **C**, **D**, **G**, and **H**. There are several key trends in this data that will be discussed here.

First, the computation times of the MPC with consensus ADMM algorithm are generally not better than the MPC-only algorithm for the trials with low numbers of processors. Furthermore, the serial computation time of ADMM is greater than the MPC computation in all cases except for trials **E** and **H**. However, from trial **C** through trial **H**, the MPC with consensus ADMM algorithm substantially outperforms the MPC algorithm with respect to the *theoretical* computation time. One possible reason for this is illustrated by figures 5.27–5.30.

Figures 5.27–5.30 show that as the number of processors increases, the initial time it takes the MPC-only algorithm to calculate a control plan increases. This is especially prominent in the beginning of each simulation. As the time steps progress, the MPC-

only algorithm seems to work itself into a rhythm such that its computation time is much less than the time spikes seen in the MPC-only plots of figures 5.27–5.30. Conversely, the MPC with consensus ADMM algorithm data shown here do not have these time spikes. In addition, after the algorithm initially begins, the theoretical computation time at each time step is less than the same time step for the MPC-only algorithm.

The MPC with consensus ADMM theoretical time plots also seem to have a relative maximum solve time across these trials. That is that they all seem to settle around having 40–45s computation times. This could possibly just be the maximum solve time needed for an ill-conditioned processor state for a processor of this size. No matter the reason, this benefits this algorithm by placing an artificial limit on computation difficulty at a given time step. This could be the reason that the centered plot in figures 5.27–5.30 (which represents the serial solve time of the MPC with consensus ADMM algorithm) displays a linear relationship. This would imply that as the system needs more control to be actuated upon it, more processors take longer to compute their solution. This means that they contribute more to the serial time, but the maximum processor solve time at a given time step remains relatively constant. Based on this assumption that the maximum time it will take to solve a single processor is about 40–45s, the linear relationship in the middle plot of figures 5.27–5.30 is logical. However, more testing is required to verify/reject this idea.

Trial **H** shows a particular example of this artificial cap on computational complexity. The timing data of this trial (figure 5.30) shows that when the MPC-only algorithm first starts out, there is approximately a 600s computation time spike as the simulation begins. However, as the MPC with consensus ADMM data for this trial shows, such computation time spikes are non-existent. Regardless of the precise reason, the absence of these spikes in computation is beneficial for the MPC with consensus ADMM algorithm.

The last component of this simulation to discuss is the overall timing data itself. For each of these trials (and each of the trials in case studies I and II), the time and state of the simulations was frozen as the control commands were being calculated. Because of this, the results presented in each of these case studies are for a theoretical, instantly computed control strategy. In a more realistic simulation, the control commands should be implemented as they are calculated with the system evolving during this computation. While not doing this makes these simulations more unrealistic, the primary purpose of these simulations was to show the feasibility of the MPC with consensus ADMM method.

There is much potential in the MPC with consensus ADMM method since its computation time per time step is generally much shorter than the MPC-only computation time depending on the system configuration. Therefore, if a delay for computing control commands was introduced to these simulations, it may have a more pronounced effect on the MPC-only algorithm. Simulations testing this could then show the degree to which this is or is not a hindrance to the computation.

As a final note, it is important to note that the theoretical computation time of the MPC with consensus ADMM method is in fact *theoretical*. There are several factors that could prevent this theoretical computation time from being achieved. One such obstacle could be the communication time between processors. In this simulation, it was assumed that this communication time was nearly instantaneous. Based on this fact alone, the elapsed time should increase in real-world implementation. Because the theoretical computation time was shown to be so much less for the MPC with consensus ADMM method, it is possible that this would not have that much of an impact. More testing is need to establish this communication time relationship.

5.6 Experiment 2 – Variable Processor Shape

In section 5.5, experiments were performed to test the overall computation time for a growing network of identical processors. In the round of experimentation discussed

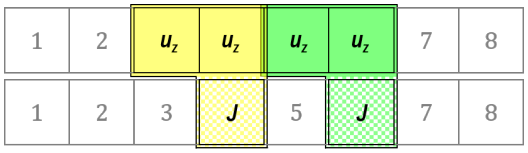

Figure 5.31: A processor of size "4." All 4 cells in this processor coordinate controls so that the second and fourth cells in the processor can more closely achieve the reference. This is indicated by the labels on each cell (u_z, J). Vertically stacked cells represent only one cell. This processor was used throughout experiment 1.

in that section, the profile and size of each processor was fixed between simulations. This allowed for the effect of adding more processors to the system to be tested. In addition, the MPC-only algorithm was executed on each processor layout to give another performance reference point. In this section, several simulations are run to test the effect of keeping the effective range of processors fixed while manipulating the size and number of processors that fill the effective range. Therefore, rather than keep processor profiles and sizes fixed while changing the effective range between simulations, these processor profiles and sizes will change, but their effective range will be fixed. Each of these simulations for the MPC with consensus ADMM algorithm will set a maximum allowable number of ADMM iterations to 2. Subsection 5.6.1 will first explain the experiment setup in detail, and present the results. Then, subsection 5.6.2 will analyze and discuss the results in the context of the larger traffic network problem.

5.6.1 Experiment 2: Simulation Structure and Results

In the experimentation in section 5.5, the profile of a processor was defined as the pairing of two agents. These agents themselves were defined as a couplet of cells that worked together in order to ensure that the cost of the more downstream cell of the two achieved some reference. This is shown here by figure 5.31, in a similar way as it was first shown in figure 5.13. The reader is again reminded that figure 5.31 indicates that all four cells in the processor are controlled (u_z), and only the second and fourth cells' data is used in the cost calculation (J).

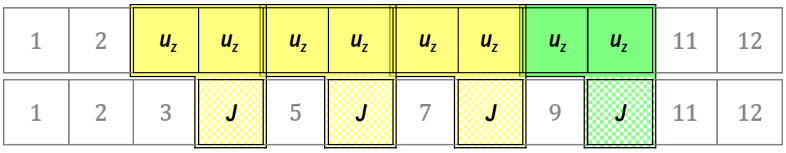

Figure 5.32: A processor of size "8." All 8 cells in this processor coordinate controls so that the second, fourth, sixth, and eighth cells in the processor can more closely achieve the reference. This is indicated by the labels on each cell (u_z, J). Vertically stacked cells represent only one cell.

Another potential processor configuration is given by figure 5.32. As this figure shows, this processor has 8 cells with a similar structuring pattern as that shown in figure 5.31. This type of modular processor expansion will be used for the rest of the simulations tested here. In particular, two of the processor configurations tested here are given by figures 5.31 and 5.32 (given below by trials **D** and **C**, respectively)[15]. Another processor configuration is given by placing 3 of the original 4-cell processors next to each other (trial **B**). The largest processor configuration tested here is given by placing 4 of the original 4-cell processors next to each other (Trial **A**). It is important to note that trials **A**–**D** in this section refer to different simulations than those discussed in subsections 5.5.2–5.5.3.

The last piece of the experimental setup to discuss is the processor placement. All processor configurations presented here are assigned a location based on the most downstream cell in the processor. In section 5.5, processors were always placed so that they could overlap with *half* of the cells from adjacent processors, as shown by figure 5.15. This pattern will be preserved for expanded processors. Using the processor in figure 5.32 as an example, since this processor is located at cell 10 and has 8 cells, the adjacent downstream processor would be placed at cell 14. This would allow it to overlap with half of the original processor.

Using these formatting techniques, simulations were run using the same overall number of cells, initial conditions, MPC parameters, cost functions (weights), and

[15]The processor in figure 5.32 can be thought of as placing two 4-cell processors next to each other.

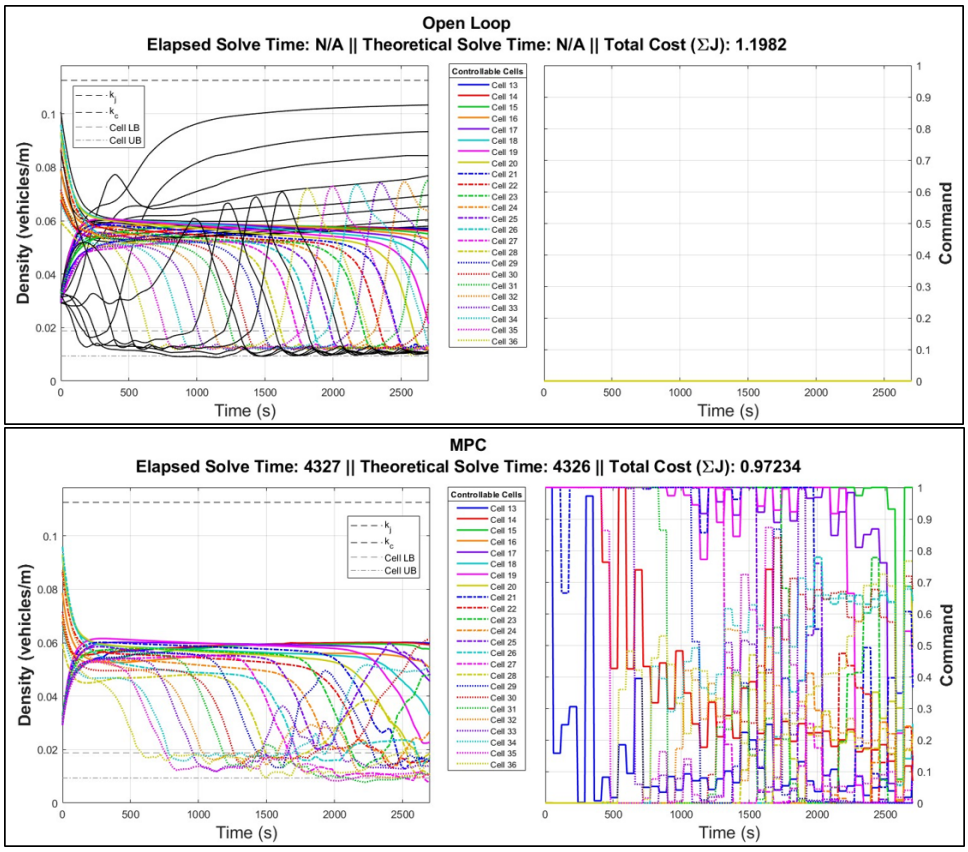

Figure 5.33: Open loop data (top) and MPC-only data (bottom) for the simulations presented in this section. The open loop simulation shows data for all 42 cells in the simulation, but the MPC-only data only shows data for cells in the effective control range.

ADMM ρ values, as were used in subsection 5.5.2. As noted previously, the changing variables of interest in the simulations presented here are the profile and size of the processors used in the simulation, and how they cover the effective control range. The effective control range in this experiment will always be cells 13–36, as indicated in table 5.7. Because of this fact, only one MPC simulation is needed for comparison because all processor configurations have the same effective range. The open loop and MPC-only simulation results are presented in figure 5.33.

Building off of this result, four simulations (trials **A**–**D**) are run with varying processor sizes, as noted above. The simulation setups and their resulting data are shown in tables 5.7 and 5.8. The output data for trials **A** and **C**, as well as the timing

Table 5.7: Cost calculated for every other cell starting from 16 and ending at 36. While MPC technically does one computation for the whole system, its number of processors is denoted as "(1)" because this processor does not use the consensus ADMM cost function. The location of this MPC "processor" is noted in parentheses for the same reason since the MPC-only algorithm does not use processors in the same way as the ADMM algorithm.

Trial	Processor Size	Number of Processors	Processor Location (cell)	Controllable Cells	Cost
OL	–	0	–	–	1.1982
MPC	24	(1)	(36)	13–36	0.97234
A	16	2	28, 36	13–36	0.96667
B	12	3	24, 30, 36	13–36	0.9443
C	8	5	20, 24, 28, 32, 36	13–36	0.92225
D	4	11	16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36	13–36	1.0124

Table 5.8: Approximate timing data for experiment 2.

Trial	Elapsed (Serial) Time (s)	ADMM Theoretical Time (s)
MPC	4327	–
A	11010	7467
B	10340	5485
C	8971	3496
D	6290	1482

plots for each trial are given by figures 5.34–5.40. Analysis and discussion of these plots will take place in subsection 5.2.2[16].

5.6.2 Experiment 2: Discussion

As table 5.7 indicates, the performance of each controller for this simulation improves on the open loop response, but not by a dramatic margin. As the MPC data in figure 5.33 shows, as well as the data from figures 5.34 and 5.35, the spiked densities were removed from the open loop response. A way of improving overall perfor-

[16]Note that experiment 2 used the same initial condition as experiment 1.

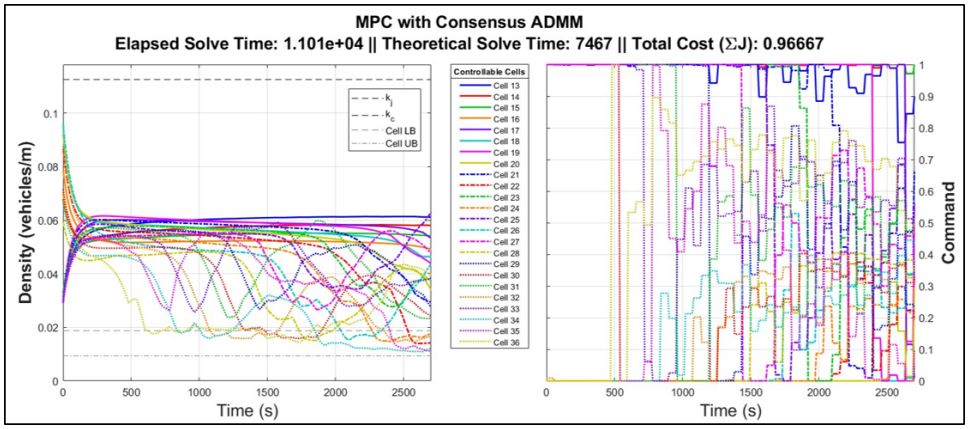

Figure 5.34: Trial **A** data.

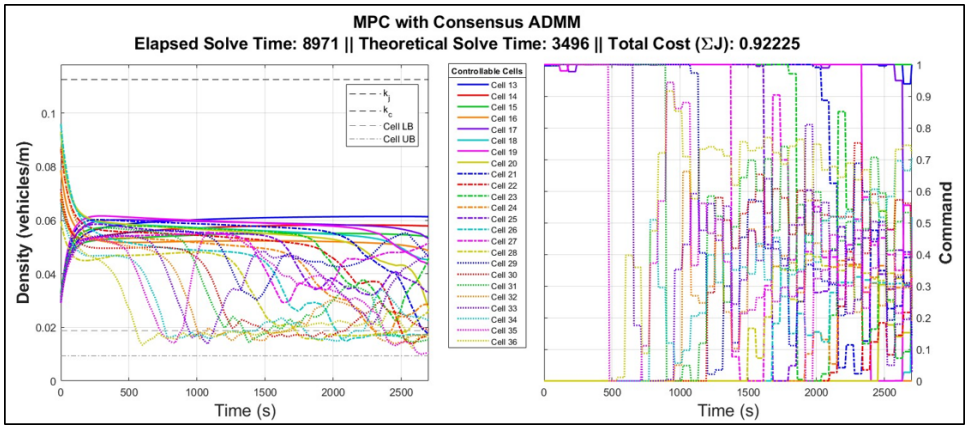

Figure 5.35: Trial **C** data.

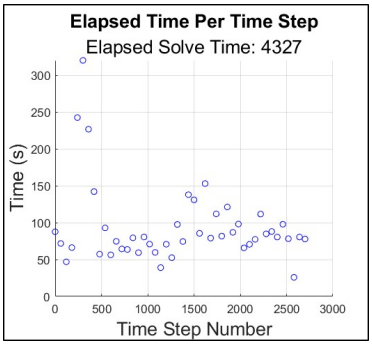

Figure 5.36: MPC timing data.

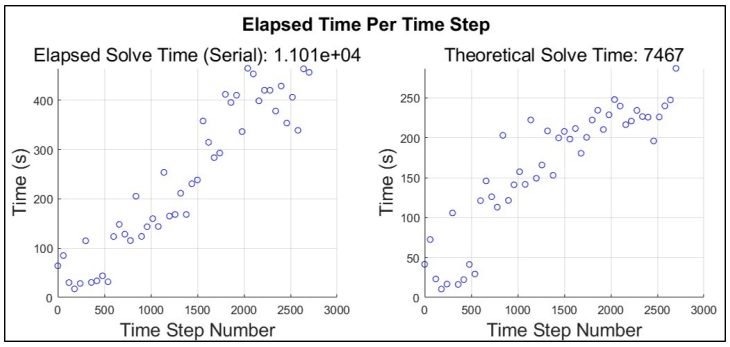

Figure 5.37: Trial **A** timing data.

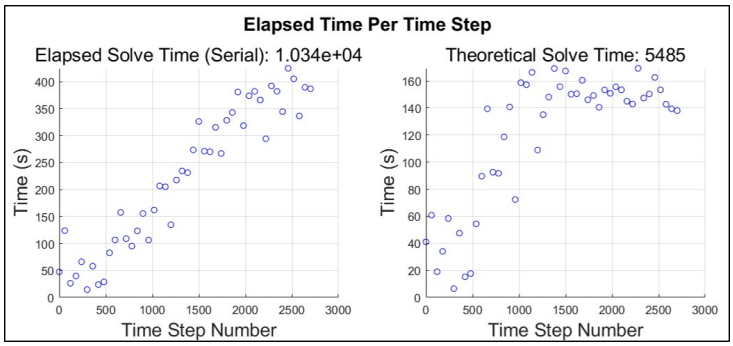

Figure 5.38: Trial **B** timing data.

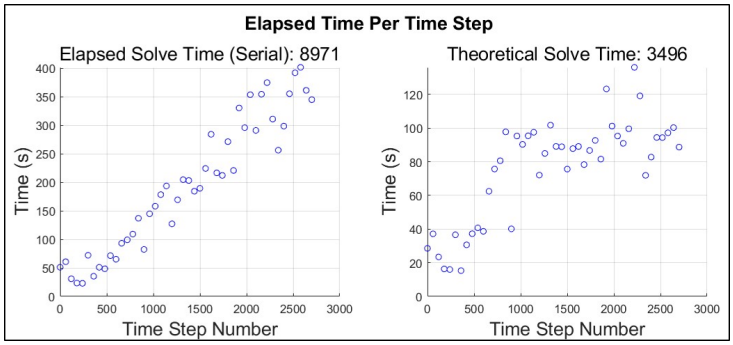

Figure 5.39: Trial **C** timing data.

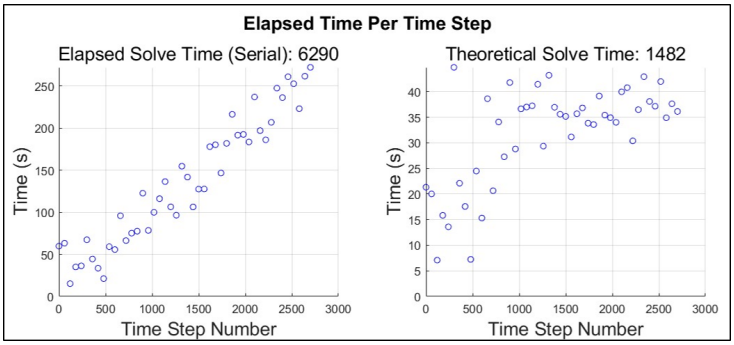

Figure 5.40: Trial **D** timing data.

mance might be to increase the prediction and control horizons so that the controller can avoid more undesirable dynamics, rather than only abrupt spikes. However, this would come at the expense of computational efficiency. With the control performance examined, the timing data for each simulation can be discussed.

Figures 5.36–5.40 support the notion suggested by the results of experiment 1 that having a smaller processor size places an artificial cap on the solve time at a single time step (when viewing the *theoretical* solve time). These plots all show the trend that as the size of the processor increases, so also does the typical computation time for a complex calculation. In other words, because small processors are used in trial **D**, the theoretical timing data indicates an approximate maximum calculation time of about $40s$. This approximate maximum is judged visually based on the apparent collection of data points around $40s$ after the initial stages of the simulation. As the size of the processor increases from 4 cells in trial **D**, to 16 cells in trial **A**, this approximate maximum processor solve time also increases. In trial **C** the approximate maximum processor solve time is about $100s$. In trial **B**, it is about $160s$. In trial **A**, about $250s$ is the maximum approximate processor solve time[17].

Importantly, this approximate time for the MPC-only trial (figure 5.36) is about $75s$. Based on this observation, as well as the theoretical timing results from table 5.8, it is clear that ADMM only offered a benefit with respect to time for processors of size 4 and 8. In a real application of this method, this would suggest that one should ideally invest in several traffic measurement and control devices with relatively shorter ranges to address this problem. In addition, if it was only possible to have a few control stations, this data suggests that using one control optimization (like with the MPC-only simulation) is more optimal than building a small number of collaborative stations.

[17]This data for trial **A** does not appear to as sharply plateau as the data for trials **B, C,** and **D,** suggesting perhaps that the maximum computational complexity for these processors is not reached as fully for this trial as it was in the others.

5.7 Case Study III Summary

In this chapter, the characteristics of this system were defined and discussed in sections 5.1–5.3. The method of simulation was discussed in section 5.4, before experimentation in sections 5.5 and 5.6. The MPC with consensus ADMM simulations in section 5.5 all used 4-cell processors to achieve control over a varying range of influence. This data suggested that when the range of cells to be controlled was large, having many of these relatively smaller processors would offer great benefits according to the theoretical computation time. The experimentation in section 5.6 altered this experiment by keeping the overall range of control fixed, while testing various processor sizes. The results from this experiment indicated that larger processors were found to generally not even be as effective as a centralized computation. Therefore, based on the data presented here, the MPC with consensus ADMM method has its greatest advantage over a regular MPC-only method when the effective range of control is large, and when the processors that cover this range are relatively small.

CHAPTER 6: CONCLUSIONS

In this work a method for distributed optimization based on the Alternating Direction Method of Multipliers (ADMM) was developed for use in traffic networks. This method was embedded within the traditional Model Predictive Control (MPC) control algorithm. The use of ADMM in this application primarily addresses the issue of having a potentially computationally expensive MPC algorithm. At its core, this research therefore addressed the problem of how to ensure a control computation time is not so excessive as to drastically hinder control performance. In this application, approximate timing data was used to judge the degree to which a problem was computationally expensive. For reasons discussed previously, only an *approximate* time was available based on practical limitations. This metric was an appropriate metric however, because the proposed method sought large time improvements. In addition, this proposed method was required to have a *similar* level of performance compared to the MPC-only algorithm. Therefore, for this implementation to be deemed successful, it must solve complex problems much faster than an MPC-only algorithm, and it must do so without a dramatic loss in performance.

This work examined three case studies. The first case study examined the impact of the basic ADMM algorithm when applied to a single-agent MPC problem. The dynamic system studied here was a single horizontal mass-spring-damper, capable affecting change on its velocity through input forces (which were technically torques). The goal was to test how this agent could be moved to a given reference position when ADMM was incorporated into the solve step of an MPC controller. This was done based on the work of [8] and [9]. Experimentation showed that traditional MPC was generally a much better control algorithm for this specific system. Thus, a new

method and problem structure needed to be tested in case study II.

Case study II built upon case study I by implementing a different version of the ADMM algorithm on a more complex mass-spring-damper problem. In this case study, ten interconnected mass-spring-dampers were to be controlled with the goal of having each achieve its reference position within a relatively short computation time. Rather than implementing the same basic ADMM algorithm from case study I, the consensus version of ADMM was applied within the MPC solve step, based on the work of [7] and [8]. A main benefit of using this algorithm is that it can enable parallel optimizations. In this implementation, this essentially divided the work of optimizing the performance of all ten agents in the system to optimizing ten smaller local optimization problems. Significantly, experimentation showed that even executing the MPC with consensus ADMM algorithm on the system *serially* was faster than the MPC-only computation. Furthermore, the performance relative to system cost was roughly the same, showing the viability of this algorithm. After the successful application of this algorithm to this problem, it was then applied to a traffic network.

The traffic network system examined here was primarily governed by principles and equations discussed to various degrees in [2], [3], [12]–[16]. After defining the system, the MPC with consensus ADMM algorithm implemented in case study II was applied. The application of this algorithm to a traffic network was not straightforward. The primary difference between the interconnected mass-spring-damper system and the dynamics of the traffic network were how agents were defined in this system. This work defined an area of control to be a cell couplet, rather than a single cell. In addition, processors were defined as being multiple cell couplets. After reframing the structure of the problem in this way, several control ranges were tested in experiment 1.

Although the serial computation for this method was not shown to be better than

the MPC-only computation time as it was in case study II, the theoretical computation was significantly better than the MPC-only algorithm for a sufficiently large system. This theoretical computation time was calculated based on the notion that only the most time-consuming processor calculation per time step should be included in the theoretical computation time. Based on this metric, experiment 1 showed that as the overall size of the control problem increased, the consensus ADMM application was most useful. Experiment 2 then tested a fixed control window, with varying processor profiles and sizes collaborating to achieve control within this window. The results of experiment 2 emphasized the importance of choosing a relatively small processor size for the MPC with consensus ADMM algorithm. Based on the results of experiments 1 and 2, the suggested ideal system structure for the implementation of MPC with consensus ADMM was to have many relatively small processors that cover a large effective window of control.

For future work, this MPC with consensus ADMM traffic network application should be implemented in a parallel fashion to verify the degree to which the theoretical computation times are achievable. A potential obstacle that is not considered in the theoretical computation time is the time lost to communication between processors. However, since the difference between theoretical computation time and serial computation time for the MPC with consensus ADMM method was largest as the size of the control range increased, it is much more likely that communication times factor in less to larger systems.

An area of this work that could benefit from more exploration is the weighting of the cost function in the MPC with consensus ADMM algorithm. In this work, the objective function weights and ρ weights were fixed. However, [8] discusses a method for dynamically updating ρ that may improve computations. Additionally, the prediction horizon, control horizon, and time steps could be more rigorously optimized for the traffic network application.

In another direction, the cost function itself should be rigorously tested and experimented upon. Since the density reference value was chosen to be the critical density, a cell had a density less than this would be inclined to slow traffic in an effort to increase its density. A method that uses density as the performance indicator while mitigating this undesirable stoppage should be tested. In addition, using the other system metrics like velocity and flow in the cost function may have potential as well.

Lastly, different processor configurations and layouts should be tested for this network. Further experimentation might reveal a more optimal problem structure in this respect. One potential improvement is to test the allowed overlap among processors. It is possible that more or less overlap than was tested here could be beneficial. Changing the degree of overlap among processors could mitigate cases where one cell completely sacrifices its own performance to help a neighboring cell. Another potential improvement would be to broaden the sight of each processor. This would allow a processor to have more data on its surroundings without requiring control commands to be computed for these extra cells. This increased sight may improve the processor's suggested control commands.

The work presented here only begins to grasp the potential for this method. Future work on the MPC with consensus ADMM algorithm for traffic networks, or other similar systems, could realize this potential more fully.

REFERENCES

[1] Bureau of Transportation Statistics, "Household Spending on Transportation: Average Household Spending," May 2023. https://data.bts.gov/stories/s/ida7-k95k.

[2] R. W. Whalin and G. Hu, "Macroscopic Fundamental Diagram Approach to Traffic Flow with Autonomous/Connected Vehicles," tech. rep., University of Florida, March 2020. Southeastern Transportation Research, Innovation, Development and Education Center (STRIDE).

[3] J. R. D. Frejo, A. Núñez, B. D. Schutter, and E. F. Camacho, "Hybrid model predictive control for freeway traffic using discrete speed limit signals," *Transportation Research Part C: Emerging Technologies*, vol. 46, pp. 309–325, September 2014.

[4] J. Jeschke and B. D. Schutter, "Parameterized Model Predictive Control Approaches for Urban Traffic Networks," in *IFAC-PapersOnLine* (M. Ghazel, ed.), vol. 54, (Lille, France), pp. 284–291, June 2021. Issue 2. 16th IFAC Symposium on Control in Transportation Systems CTS 2021.

[5] X. Luan, B. D. Schutter, T. van den Boom, F. Corman, and G. Lodewijks, "Distributed optimization for real-time railway traffic management," in *IFAC-PapersOnLine* (A. Ferrara, ed.), vol. 51, (Savona, Italy), pp. 106–111, June 2018. Issue 9. 15th IFAC Symposium on Control in Transportation Systems CTS 2018.

[6] The Mathworks, Inc., "Understanding Model Predictive Control," May 2023. https://www.mathworks.com/videos/series/understanding-model-predictive-control.html.

[7] T. H. Summers and J. Lygeros, "Distributed Model Predictive Consensus via the Alternating Direction Method of Multipliers," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, (Monticello, IL, USA), pp. 79–84, IEEE, October 2012.

[8] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[9] M. Annergren, A. Hansson, and B. Wahlberg, "An ADMM Algorithm for Sovling $\mathcal{L}_1$ Regularized MPC," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, (Maui, HI, USA), pp. 4486–4491, IEEE, December 2012.

[10] M. A. Davies and T. L. Schmitz, *System Dynamics for Mechanical Engineers*. Springer New York, NY, 1 ed., November 2014.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004. Seventh printing with corrections 2009.

[12] H. Yu, R. Jiang, Z. He, Z. Zheng, L. Li, R. Liu, and X. Chen, "Automated vehicle-involved traffic flow studies: A survey of assumptions, models, speculations, and perspectives," *Transportation Research Part C: Emerging Technologies*, vol. 127, pp. 1–22, June 2021.

[13] J. Chen, Y. Yu, and Q. Guo, "Freeway Traffic Congestion Reduction and Environment Regulation via Model Predictive Control," *Algorithms*, pp. 1–23, October 2019. Special Issue. Model Predictive Control: Algorithms and Applications.

[14] L. Muñoz, X. Sun, R. Horowitz, and L. Alvarez, "Traffic Density Estimation with the Cell Transmission Model," in *Proceedings of the 2003 American Control Conference, 2003*, (Denver, CO, USA), pp. 3750–3755, IEEE, June 2003.

[15] P. K. Shahri, B. HomChauduri, S. S. Pulugurtha, A. Mesbah, and A. H. Ghasemi, "Traffic Congestion Control Using Distributed Extremum Seeking and Filtered Feedback Linearization Control Approaches," *IEEE Control Systems Letters*, vol. 7, pp. 1003–1008, December 2022.

[16] P. K. Shahri, B. HomChauduri, S. S. Pulugurtha, and A. H. Ghasemi, "Freeway Traffic Control using Filtered Feedback Linearization (UP)." See [15]. Guidance in initial definition of system dynamics/parameters.