8-2020

Formation Control Using Vehicle Operational Envelopes and Behavior-Based Dual-Mode Model Predictive Control

Brian Merrell
Utah State University

FORMATION CONTROL USING VEHICLE OPERATIONAL ENVELOPES AND BEHAVIOR-BASED DUAL-MODE MODEL PREDICTIVE CONTROL

by

Brian Merrell

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Greg Droge, Ph.D. Major Professor	Randy Christensen, Ph.D. Committee Member
Jonathan Phillips, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2020

ABSTRACT

Formation Control using Vehicle Operational Envelopes and Behavior-based Dual-mode Model Predictive Control

by

Brian Merrell, Master of Science

Utah State University, 2020

Major Professor: Greg Droge, Ph.D.
Department: Electrical and Computer Engineering

This thesis presents a dual-mode, behavior-based model predictive control (MPC) framework for formation control. Given an initial desired trajectory from a deliberative planner for a virtual leader, a framework is presented to generate follower trajectories for each agent within the formation. When combined with operational envelopes, a designated area for each vehicle to maneuver, the multi-vehicle problem decomposes into a single vehicle problem. A single vehicle framework is presented to track the respective trajectory when possible, or stay near it when it passes through previously unknown obstacles. Arc-based behaviors are used to rapidly produce desirable robot controls while a zero-error epsilon trajectory tracking behavior is used to ensure convergence when the trajectory is obstacle free. As the zero-error epsilon tracking control can be defined in terms of a linear system with an algebraic mapping to the actual control inputs of the system, the terminal cost in the MPC framework can be immediately designed using the infinite horizon cost of the terminal controller at the terminal state. This allows for linear system theory to be used to establish convergence characteristics for a nonlinear, nonholonomic system. The resulting formation control framework is illustrated through a real-time simulation with trajectories

passing through obstacles. The simulated robot is able to seamlessly balance tracking with obstacle avoidance.

(72 pages)

PUBLIC ABSTRACT

Formation Control using Vehicle Operational Envelopes and Behavior-based Dual-mode Model Predictive Control

Brian Merrell

This thesis presents a control framework for formation control. Given an initial desired trajectory, a framework is presented to generate trajectories for each vehicle within the formation. When combined with an operational envelope, a designated area for each vehicle to maneuver, for each vehicle the multi-vehicle formation control problem can be redefined into a single vehicle problem. A single vehicle framework is presented to track the respective trajectory when possible, or stay near it when it passes through previously unknown obstacles. Arc-based motions are used to rapidly produce desirable robot controls while a trajectory tracking motion is used to ensure that the vehicle tracks the trajectory when it is obstacle free. The resulting formation control framework is illustrated through a real-time simulation with trajectories passing through obstacles. The simulated robot is able to seamlessly balance tracking with obstacle avoidance.

CONTENTS

LIST OF FIGURES

Figure Page

ACRONYMS

ARE	Algebric Riccati Equation
CCC	Circle Circle Circle
CCPath	Continuous Curvature Path
CCTraj	Continuous Curvature Trajectory
CLC	Circle Line Circle
CLF	Control Lyapunov Function
DWA	Dynamic Window Approach
FC	Formation Control
FONC	First Order Necessary Condition
LQR	Linear Quadratic Regulator
LTI	Linear Time Invariant
MPC	Model Predictive Control
RHC	Receding Horizon Control
SLSQP	Sequential Least-Squares Quadratic Programming

CHAPTER 1

Introduction

Multi-agent robotic systems are becoming increasingly prevalent in a host of applications, including security and surveillance [2], factory automation [3], inspection operations [4], transit [5] and convoying applications [6], distributed sensing [7], and target tracking networks [8], to name a few. A nearly universal requirement of multi-agent systems is that, at some point in their mission, they move together in coordinated motion, often in tight formation. Thus, formation control forms a fundamental enabling capability for multi-agent operations.

Formation control is also a canonical problem for multi-agent control as there is an inherit coupling between individual and collective decision making. A vehicle must simultaneously consider individual systems due to satisfying local level requirements, such as dynamics constraints or obstacle avoidance, while achieving desirable collective outcome, like formation position or trajectory tracking. This coupling between individual and collective decision making can sometimes present conflicting objectives. For example, if a single agent is required to execute an obstacle avoidance maneuver, it may have to break formation, which can interfere with neighboring agents and may even risk a collision between agents.

While there are a host of approaches to formation control, three are particularly relevant to this work. First, in a virtual structure approach the entire formation is treated as a single entity. By defining a single oriented point within the structure, the desired position of each agent is defined [9]. Second, the leader-follower approach defines an agent as a leader that all other agents within the formation follow. This provides a centralized interface for planning a formation: an operator commands the leader and the formation follows. The first and second methods can be combined by defining a virtual leader (an artificial vehicle that an operator can control perfectly), which provides centralized structured movement.

Third, a behavior-based approach achieves formation control by defining specific behaviors for individual agents. For instance, artificial potential fields can be designed and combined to have agents avoid each other, avoid obstacles, and converge to a desired position within the formation [10].

While these formation approaches have distinct advantages, each approach has some deficiency. The virtual structure approaches have been applied to open environments with little to no obstacles. Virtual leader trajectories are also typically preplanned or very local (i.e. a single arc). The leader-follower approaches have convergence and tracking guarantees, but also neglect obstacles and typically do not provide inter-vehicle collision avoidance guarantees. On the other hand, behavior-based approaches fundamentally consider obstacles and inter-agent collisions, but are difficult to analyze and ensure that guarantees are met (either in collisions or convergence). Moreover, each of these approaches typically assume that the agents are point masses and do not directly account for dynamic constraints nor the executability of the trajectories.

The purpose of this work is to combine these approaches to develop a moving formation control framework that allows each individual vehicle to adapt its control based upon dynamic and environmental limitations while safely following a coordinated trajectory within the formation. This will be accomplished by combining four major elements. The first is the application of Model Predictive Control (MPC), which is control methodology that repeatedly solves a finite-horizon optimal control problem to add feedback to the otherwise open-loop optimal control solution [11]. The second is the use of virtual structures, which allow the desired positions of vehicles in the formation to be defined in terms of relative displacement to a single oriented point, e.g., [9], [12]. This allows the formation to be treated as a single virtual entity and enables the motion of the formation to be defined by the motion of that entity, referred to as a virtual leader, e.g., [13]. The virtual structure will be decomposed into time-varying operational envelopes that will enable each individual vehicle to determine its own control trajectory, which will be done using a behavior-based model predictive control approach. The third major element is the development of motion

for that virtual leader using continuous-curvature paths [14], [15]. For a single vehicle, continuous-curvature paths respect constraints related to the curvature and change in curvature of a vehicle. Paths between waypoints can be found through the combination of straight lines, circular arcs of maximum curvature, and clothoids which smoothly transition between the straight lines and circular arcs. The fourth element is the use of zero-error trajectory tracking control laws, [15] [16], which allow vehicles to converge to and track a sufficiently smooth trajectory. To accomplish the proposed work, the following technical objectives will be accomplished:

1. Develop virtual leader motion from desired waypoints that will result in continuous trajectories for each agent
2. Utilize the virtual structure to create individual vehicle operational envelopes
3. Create an optimization framework that will bound individual vehicle movement to the operational envelope
4. Develop first order necessary conditions (for numerical solutions) and convergence theorems to ensure agents move as desired
5. Provide theoretic results using Matlab and real-time implementation using C++ and the Robot Operating System (ROS)

CHAPTER 2

Preliminaries

This chapter presents key background information for the development of the zero-error moving formation control approach discussed in subsequent chapters. First, the dynamic motion models for the individual agents and the virtual leader are presented. Next, the relevant background on behavior-based control and MPC is discussed. The formation structure is then defined along with the Voronoi tessellation that will be utilized as a time varying operational envelope. The chapter will then concluded with a brief overview of clothoid-based trajectory generation.

2.1 Dynamic Motion Models

The motion for each agent is modeled using a smooth, unicycle kinematic model, e.g., [17]. As the formation motion will be defined for a homogeneous group of agents, agent indices are omitted for sake of clarity. The states include the two-dimensional position of the vehicle, (q_1, q_2), its orientation ψ, and translational and rotational velocities, (v, ω), respectively. It is assumed that the accelerations are directly controlled with the dynamics taking the form:

$$\dot{x} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \psi \\ v \\ \omega \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ \omega \\ u_v \\ u_\omega \end{bmatrix}, \tag{2.1}$$

where u_v and u_ω are the translational and rotational accelerations. The velocities and accelerations can be grouped together as $\boldsymbol{v} = \begin{bmatrix} v & \omega \end{bmatrix}^T$ and $\boldsymbol{a} = \begin{bmatrix} u_v & u_\omega \end{bmatrix}^T$, respectively.

The notation $\mathcal{C}^k$ is used to denote the set of functions that are k-times continuously differentiable. In Chapter 4, a virtual leader trajectory in $\mathcal{C}^4$ will be produced to enable

executable trajectories for each agent. Three degrees of smoothness are needed due to the tracking control used. The need for the fourth degree of continuity comes from producing the desired follower trajectories from the virtual leader trajectory.

The model used to define the motion of the virtual leader takes a similar form to (2.1) with three significant differences to enable a more direct consideration of the $\mathcal{C}^4$ smoothness requirements. First, the translational velocity, v, is considered a constant parameter instead of a time-varying state. Second, the rotational velocity is replaced by a curvature term, $\omega = v\kappa$, which allows the curvature constraint to manifest directly into the dynamic model. Third, derivatives of the curvature are included as additional states, so that the curvature can be directly controlled to satisfy the $\mathcal{C}^4$ smoothness requirement. Thus, the full leader state is given as follows.

$$x_l = \begin{bmatrix} q_{l_1} \\ q_{l_2} \\ \psi_l \\ \kappa_l \\ \sigma_l \\ \gamma_l \end{bmatrix}, \dot{x}_l = \begin{bmatrix} v_l \cos(\psi_l) \\ v_l \sin(\psi_l) \\ v_l \kappa_l \\ \sigma_l \\ \gamma_l \\ u_l \end{bmatrix}, \tag{2.2}$$

where κ_l is the curvature of the leader trajectory and its first, second, and third derivatives are σ_l, γ_l, and u_l. It is assumed that $\kappa_l \in [-\kappa_{max}, \kappa_{max}]$, $\sigma_l \in [-\sigma_{max}, \sigma_{max}]$, and $u_l \in [-u_{max}, u_{max}]$.

It is also assumed the vehicle has a range sensor, such as lidar, that provides evenly spaced range measurements originating from the center of the vehicle, as depicted in Figure 2.1.

2.2 Behavior-based Control

The term behavior in this work is used to describe a feedback control law defined for a specific task, namely tracking a trajectory and executing an arc. These specific control laws can be developed independently and combined to achieve an overall desired objective. This section will describe the two controllers that will be used in the subsequent chapters

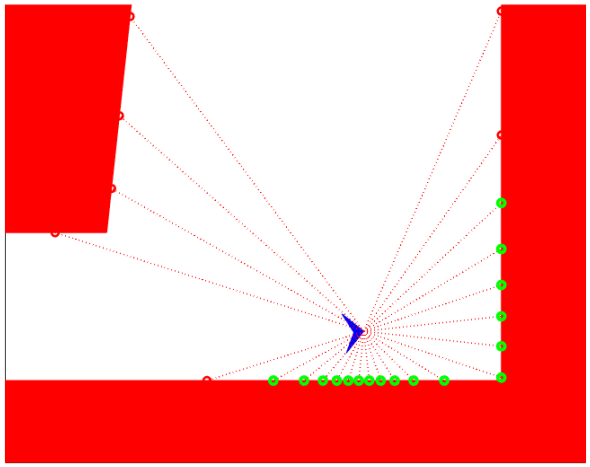

Fig. 2.1: The agent is shown as a blue triangle and the obstacles are shown as red polygons. The range sensors are depicted as lines extending from the vehicle with circles at the points of detection. The green detection circle depict a detected contiguous region.

within a behavior-based MPC framework.

2.2.1 Trajectory Tracking

While there are numerous techniques for tracking a trajectory (i.e., have $q(t) \longrightarrow q_d(t)$ as $t \longrightarrow \infty$) [18–22], a simplified form[1] of the control law developed in [16] and modified in [15] is used as it allows the system to be treated as a linear, time-invariant system. This property is later exploited to prove convergence of the MPC formulation. The control law in [15] is summarized in what follows for sake of clarity and to introduce notation used throughout the subsequent chapters.

The basic premise behind the control law is that a position directly in front of the vehicle is completely controllable. The point in front of the vehicle is referred to as the ϵ-point and calculated as

$$q_\epsilon(t) = q(t) + \epsilon \begin{bmatrix} \cos(\psi(t)) \\ \sin(\psi(t)) \end{bmatrix}, \tag{2.3}$$

[1]The ϵ variable is allowed to decrease to an arbitrarily small value in [16] whereas it is constant in [15].

where $\epsilon > 0$ is some parameter to be chosen. It was shown in [16] that any $q_d(t) \in \mathcal{C}^2$ can be tracked arbitrarily close, defining ϵ-tracking as $||q_\epsilon(t) - q_d(t)|| \longrightarrow \epsilon$ as $t \longrightarrow \infty$. Further developments in [15] show that if $q_d(t) \in \mathcal{C}^3$ then a new trajectory, $q_{\epsilon_d}(t) \in \mathcal{C}^2$, could be designed from $q_d(t)$ such that as $q_\epsilon(t) \longrightarrow q_{\epsilon_d}(t)$ then $q(t) \longrightarrow q_d(t)$. Moreover, since (2.1) forms a differentially flat system, $q_d(t)$ can be used to define a desired state for the full state trajectory, i.e, $x_d(t)$. A bi-product of the proof in [15] shows that as $q(t) \longrightarrow q_d(t)$ then $x(t) \longrightarrow x_d(t)$.

Thus, the full state trajectory tracking controller can be defined in the following steps:

1. Reshape $q_d(t)$ to $q_{\epsilon_d}(t)$
2. Assume the ϵ-point is not constrained by the vehicle dynamics and design a control law for $q_\epsilon(t)$ to converge to $q_{\epsilon_d}(t)$
3. Algebraically map the ϵ-point control inputs into the acceleration control inputs, $\boldsymbol{a}$, of the vehicle

The *first step* is to create a desired trajectory for the ϵ-point to follow. This is done by defining a trajectory at a distance in front of $q_d(t)$ in the same fashion as the definition of the ϵ-point, i.e.,

$$q_{d_\epsilon}(t) = q_d(t) + \epsilon \begin{bmatrix} \cos(\psi_d) \\ \sin(\psi_d) \end{bmatrix} \tag{2.4}$$

where $\psi_d(t)$ is the desired orientation at time t. Assuming that $q_d(t) \in \mathcal{C}^3$, [15] derives the ϵ-trajectory from the desired trajectory as:

$$\begin{aligned} q_{\epsilon_d}(t) &= q_d(t) + \epsilon \begin{bmatrix} \cos\psi_d \\ \sin\psi_d \end{bmatrix} \\ \dot{q}_{\epsilon_d}(t) &= R_\epsilon(\psi_d)\boldsymbol{v}_d \\ \ddot{q}_{\epsilon_d}(t) &= R_\epsilon(\psi_d)\hat{\omega}_d\boldsymbol{v}_d + R_\epsilon(\psi_d)\boldsymbol{a}_d \end{aligned} \tag{2.5}$$

where $q_d = \begin{bmatrix} x_d & y_d \end{bmatrix}^T$ and the remaining desired states can be written as

$$\begin{aligned} \psi_d &= atan2(\dot{y}_d, \dot{x}_d) \\ v_d &= \sqrt{\dot{x}_d^2 + \dot{y}_d^2} \\ \omega_d &= (\dot{x}_d \ddot{y}_d - \dot{y}_d \ddot{x}_d) v_d^{-2} \\ a_d &= (\dot{x}_d \ddot{x}_d + \dot{y}_d \ddot{y}_d) v_d^{-1} \\ \alpha_d &= (\dot{x}_d \dddot{y}_d - \dot{y}_d \dddot{x}_d) v_d^{-2} - 2 a_d \omega_d v_d^{-1} \end{aligned} \quad . \tag{2.6}$$

The velocities and accelerations can be grouped as $\boldsymbol{v_d} = \begin{bmatrix} v_d & \omega_d \end{bmatrix}^T$ and $\boldsymbol{a_d} = \begin{bmatrix} a_d & \alpha_d \end{bmatrix}^T$, respectively.

The *second step* is to design a control for the ϵ-point assuming that $\ddot{q}_\epsilon = u_\epsilon \in \mathbb{R}^2$, where $u_\epsilon(t) \in \mathbb{R}^2$ can be chosen freely. Allow the ϵ-state dynamics to be defined as:

$$\begin{aligned} x_\epsilon = \begin{bmatrix} q_\epsilon \\ \dot{q}_\epsilon \end{bmatrix}, \; \dot{x}_\epsilon = A x_\epsilon + B u_\epsilon, \\ \text{s.t. } A = \begin{bmatrix} 0_2 & I_2 \\ 0_2 & 0_2 \end{bmatrix}, \quad B = \begin{bmatrix} 0_2 \\ I_2 \end{bmatrix}, \end{aligned} \tag{2.7}$$

where 0_2 and I_2 are the 2×2 zero matrix and identity matrix, respectively. To have x_ϵ converge to the desired trajectory, $x_{\epsilon_d} = \begin{bmatrix} q_{\epsilon_d}^T & \dot{q}_{\epsilon_d}^T \end{bmatrix}^T$, u_ϵ can be defined as

$$u_\epsilon = \ddot{q}_{\epsilon_d} - K z, \; z = x_\epsilon - x_{\epsilon_d}. \tag{2.8}$$

For convergence, any stabilizing method for choosing K could be used. In the subsequent chapters, a linear-quadratic regulator (LQR) is used as the calculation of the gain matrix, K. LQR is a control technique in which a feedback control law is generated by minimizing

the cost

$$\min_{u_\epsilon(\cdot)} \int_{t_0}^{\infty} (x_\epsilon^T Q x_\epsilon + u_\epsilon^T R u_\epsilon) dt \tag{2.9}$$

$$\text{s.t. } \dot{x}_\epsilon = A x_\epsilon + B u_\epsilon, \tag{2.10}$$

where $R \in \mathbb{R}^{2\times 2}$ is a positive definite weighting matrix on the control, and $Q \in \mathbb{R}^{4\times 4}$ is a positive semi-definite weighting matrix on the state (e.g., [23, 24]). The optimal control is given by

$$u_\epsilon = -Kz, \quad K = R^{-1} B^T P, \tag{2.11}$$

where P is the solution to the Algebraic Riccati Equation (ARE):

$$A^T P + PA - PBR^{-1}B^T P + Q = 0, \tag{2.12}$$

The LQR formulation can be particularly important for MPC as it allows for the immediate expression of the cost-to-go as $x_\epsilon^T P x_\epsilon$ [25].

The *third step* of the trajectory tracking controller is to provide an algebraic mapping between u_ϵ and $\boldsymbol{a}$. The mapping is given in the following lemma.

Lemma 1. *Assuming the dynamics in (2.1), a desired trajectory $q_d(t) \in \mathcal{C}^3$, and the reshaping of the desired reference trajectory in (2.5), the following control law will achieve asymptotic tracking of $q_d(t)$ by $q(t)$.*

$$\boldsymbol{a} = R_\epsilon^{-1} u_\epsilon - \hat{\omega}_\epsilon \boldsymbol{v}, \tag{2.13}$$

where

$$\hat{\omega}_\epsilon = \begin{bmatrix} 0 & -\epsilon\omega \\ \frac{\omega}{\epsilon} & 0 \end{bmatrix}, \; R_\epsilon^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\epsilon} \end{bmatrix} \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix}$$

Proof. See [15]. □

This tracking control law will be used as the primary mode of operation when the trajectory is free of obstacles.

2.2.2 Arc-Based Control

Arc motions are natural motions for mobile platforms, often corresponding to constant inputs. For example, if the smooth unicycle were moving at $v(t_0) = v_0$ and $\omega(t_0) = \omega_0$ and $u_v(t) = u_\omega(t) = 0 \ \forall t \geq t_0$, then the vehicle would move in an arc with radius $\frac{v_0}{\omega_0}$. An arc-based control for (2.1) thus consists of defining a desired velocity pair (v_d, ω_d) and controlling the vehicle such that $v(t) \longrightarrow v_d$ and $\omega(t) \longrightarrow \omega_d$. This can be accomplished through the control

$$\begin{aligned} u_v &= k_1(v_d - v(t)) \\ u_\omega &= k_2(\omega_d - \omega(t)) \end{aligned} \tag{2.14}$$

where $k_1, k_2 > 0$.

This arc-based control law will be used as the mode of operation when the trajectory is near obstacles.

An example of using arc-based motions can be seen in the dynamic window approach (DWA) that was presented in [26]. DWA is a navigation algorithm that accounts for the vehicle dynamics and obstacle avoidance by repeatedly selecting the "best" (v_d, ω_d) pair. The "best" pair at a given time instant is determined through an optimization routine that considers the current vehicle position, desired velocity, obstacle locations, and an execution horizon. This optimal pair is executed for a set amount of time and then recalculated.

Various techniques have offered improvements to DWA to provide information on the connectivity of the goal through planned paths [27], the use of navigation functions [28], and the addition of other further control modalities to ensure connectivity of the arcs to a path [29]. There are two common themes in each of the works cited: first, a receding horizon is used to ensure obstacle avoidance while considering vehicle motion constraints; second, basic feedback control laws, such as the arc-based control, are used to reduce the

required computation in the optimization.

2.3 Dual-Mode MPC

The receding horizon optimization process of the DWA can be seen as an example of MPC. Each computation of the MPC algorithm minimizes a cost objective to find the optimal control over a finite horizon, executes control for a small horizon, and repeats the process. Following the notation of [29], this work denotes the starting time of optimization as t_0 and the length of the time horizon as Δ. To accommodate the fact that MPC simulates the dynamics forward in time, a double time notation is used, such that $x(t;t_0)$ and $u(t;t_0)$ denote the state and input predicted for time t as computed at time t_0. The actual state and input at time t is written as $x(t;t)$ and $u(t;t)$. It is assumed $x(t;t_0) \in X \subset \mathbb{R}^n$ and $u(t;t_0) \in U \subset \mathbb{R}^m$, with the dynamics expressed as $\dot{x}(t;t_0) = f(x(t;t_0), u(t;t_0))$. The general optimization problem can be written as:

$$\begin{aligned} &\min_{u(\cdot;t_0)} \int_{t_0}^{t_0+\Delta} L(x(\xi;t_0), u(\xi;t_0))d\xi + \Phi(x(t_0+\Delta;t_0)) \\ &\text{s.t. } \dot{x}(t;t_0) = f(x(t;t_0), u(t;t_0)), x(t_0+\Delta;t_0)) \in X_f, \end{aligned} \tag{2.15}$$

where ξ represents the variable of integration, L is known as the running or instantaneous cost, Φ represents the terminal cost at the final time, $x(t_0;t_0)$ is a known initial state, and $X_f \subset X$ is a terminal constraint set. The instantaneous cost largely deals with the transients of the system, such as obstacle avoidance or deviating from a nominal velocity, while the terminal cost consists of planning objectives such as the distance from a goal.

The resultant control is only executed for a small portion of time, δ. The optimization is repeated again by increasing t_0 by δ. This repeated optimization allows the vehicle to respond to previously unknown variables, such as obstacles, sensing errors, or motion modeling errors.

Dual-mode MPC is a strategy that uses a stabilizing control law inside an MPC framework to aide in the convergence proof, although the control law itself may never be executed,

e.g. [29], [30]. It is assumed there exists a controller, $u = \kappa_f(x)$, that will render the desired equilibrium locally stable for all $x \in X_f$. The conditions for asymptotic stability are stated in [30] and are provided here for sake of completeness:

C1: X_f *is closed*

C2: $\kappa_f(x) \in U$, $\forall x \in X_f$

C3: $f(x, \kappa_f(x)) \in X_f$, $\forall x \in X_f$

C4: $\frac{\partial \Psi}{\partial x}(x) f(x, \kappa_f(x)) + L(x, \kappa_f(x)) < 0$, $\forall x \in X_f, x \neq 0$

The first two conditions state the terminal controller is well-defined in the terminal set X_f. One implication of $C3$ is that the terminal controller can be utilized as a warm start for the optimization. The last condition, $C4$, states that the cost forms a Control Lyapunov Function (CLF) when using the terminal controller, invoking asymptotic convergence.

There are two major difficulties in MPC: cost design and computational intensity. Cost design borderlines art as the cost must simultaneously produce desired transients while satisfying convergence requirements. The computational burden of MPC is due to computing the optimal control solution repeatedly. This typically requires either direct or indirect solutions to a two-point boundary value problem [11], which can be computationally intensive.

2.4 Behavior-based MPC

The application of a behavior-based MPC scheme, such as [31], reduces the number of parameters to be optimized and produces an intuitive cost design. Instead of choosing the entire control trajectory, the optimization can choose the set points and the time to switch between different controllers. For example, optimizing over a series of three arc-based controllers would reduce the optimization to eight variables (two for each arc and two for the switch times) and the resulting trajectory would be a series of arcs, a natural motion for wheeled vehicles.

The i^{th} control law could be a function of the state and a vector of parameters, $\theta_i \in \Theta$, and is denoted as $\kappa_i(x(t), \theta_i)$, where Θ is the set of valid parameters for θ_i. As mentioned, κ_i and κ_j may actually be the same control law with different values for θ_i and θ_j, or

they could be different controllers where θ_i and θ_j could even have different dimensions. A sequence of these controllers can be represented as, $(\kappa_0, \tau_0), (\kappa_1, \tau_1), \ldots, (\kappa_N, \tau_N)$, where τ_i is the time for the system to begin executing $\kappa_i(x(t), \theta_i)$ [32]. This allows the system to have piecewise dynamics, which are written as:

$$\begin{gathered}\dot{x}(t; t_0) = f\big(x(t; t_0), \kappa_i(x(t; t_0), \theta_i)\big) \text{ for } \tau_i \leq t < \tau_{i+1}. \\ i \in [0, 1, \ldots, N]\end{gathered} \tag{2.16}$$

For the sake of brevity, the dynamics are expressed as $\dot{x}(t; t_0) = f_i(x(t; t_0), \theta_i)$.

To reflect the change from the infinite dimensional problem to a finite parameter optimization problem, the form of the instantaneous cost and dynamics in (2.15) are updated. In the subsequent chapters, the instantaneous cost does not depend upon the parameter vectors, θ_i. Thus a slightly simpler form of the cost found in [31] is used:

$$\begin{gathered}\min_{(\theta,\tau)} J(\theta, \tau) = \int_{t_0}^{t_0+\Delta} L(x(\xi; t_0))d\xi + \Phi(x(t_0 + \Delta; t_0)) \\ \text{s.t. } \dot{x}(t; t_0) = f_i(x(t; t_0), \theta_i) \\ 0 \leq v_i \leq v_{max}, \quad -\omega_{max} \leq \omega_i \leq \omega_{max} \\ \forall t \in \tau_i \leq t < \tau_{i+1},\end{gathered} \tag{2.17}$$

where $\theta = [\theta_0, \theta_1, \ldots, \theta_N]$ and $\tau = [\tau_0, \tau_1, \ldots, \tau_N]$.

With this final parameterized form of the cost equation defined, various parameter optimization techniques can be used to solve for θ and τ. Often the gradients are used to improve performance and are thus given in the following Lemma.

Lemma 2. *The first order necessary conditions for optimality of* (2.4) *are*

$$\begin{aligned} \frac{\partial J}{\partial \tau_i} &= (\lambda^T(f_{i-1} - f_i))\Big|_{\tau_i} = 0 \\ \frac{\partial J}{\partial \theta_i} &= \xi_i(\tau_i) = 0, \end{aligned} \tag{2.18}$$

where

$$\begin{aligned} \dot{\lambda} &= -\frac{\partial L_i}{\partial x}^T - \frac{\partial f_i}{\partial x}^T \lambda \\ \textit{for } \tau_i \leq t < \tau_{i+1},\ i &= 0, \ldots, N \\ \lambda(\tau_{N+1}) &= \frac{\partial \Phi}{\partial x}(x(\tau_{N+1})) \\ \dot{\xi}_i &= -\frac{\partial f_i}{\partial \theta_i}^T \lambda \\ \xi(\tau_{i+1}) &= 0 \end{aligned} \tag{2.19}$$

Proof. The exact derivation is given in [31] can be followed with one simplification: the costs do not depend on θ. This results in $\frac{\partial L}{\partial \theta}$ being zero and the instantaneous cost being continuous across switching times. □

2.5 Formation Definition

The MPC framework discussed thus far is pivotal to the development of the individual vehicle within a formation. Formation control is the coordinated effort of multiple vehicles to achieve or travel in some desired configuration. While there are a myriad of methods to define and achieve formation control, this work takes a rigid, geometric approach to defining the desired relative placement of the vehicles. Meaning that, at any snapshot in time, the desired relative position of vehicles forms a particular geometric shape.

A virtual structure approach, [9], is used to allow the formation to move about the environment. The virtual structure approach creates an image of the desired formation and uses this image to create references of what the desired formation state should be as depicted in Figure 2.2. The information of this structure is assumed as common knowledge for the agents within the formation. The placement of each agent within the structure can

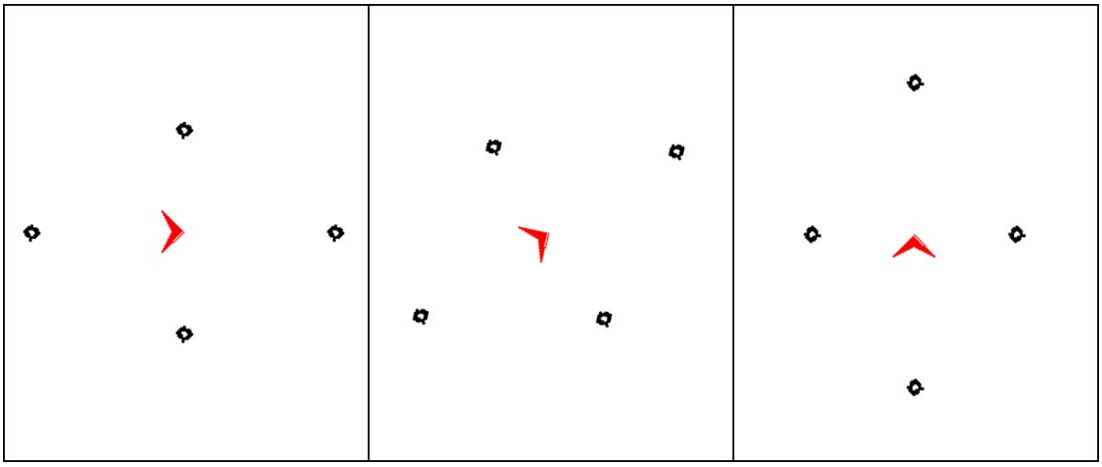

Fig. 2.2: Example of a virtual structure shown in different configurations. The red icon denotes a position and orientation of the virtual leader and the circles represent the desired follower positions.

be defined in terms of the "virtual leader" configuration consisting of a reference position, q_l, and an orientation, ψ_l. This configuration is referred to as a "virtual leader" since it may not be an actual physical agent. Following the developments in [13], the desired position of agent i at time t, $q_{d_i}(t)$, can be defined with respect to the leader as:

$$\begin{aligned} q_{d_i}(t) &= R(\psi_l(t))r_i + q_l(t) \\ R(\psi_l(t)) &= \begin{bmatrix} \cos\psi_l(t) & -\sin\psi_l(t) \\ \sin\psi_l(t) & \cos\psi_l(t) \end{bmatrix}, \end{aligned} \tag{2.20}$$

where r_i is the desired relative offset of agent i from the virtual leader.

In Section 4.2, a virtual leader trajectory in $\mathcal{C}^4$ will be produced to enable executable trajectories for each agent. Three degrees of smoothness are needed due to the tracking control used. The fourth will be shown to come from the third derivative of (2.20).

2.6 Voronoi Tessellations

To further define a virtual structure, this work proposed to incorporate a time-varying operational envelope for each vehicle to maneuver. A method for determining the boundary of this envelope is via a Voronoi tessellation. A Voronoi tessellation is defined as the cells

that are formed by the area that lie within the region bounded by equidistant lines between a set of vertices which is shown as

$$\mathbb{V}_i \cap \mathbb{V}_j = \emptyset, \forall i, j \in n, i \neq j \tag{2.21}$$
$$\text{s.t. } \mathbb{V}_i = \{w \in \mathbb{V}_{\sqsupset} | dist(q_{d,i}, w) \leq dist(q_{d,j}, w), \forall j \neq i\},$$

where $\mathbb{V}_{i/j}$ represent a Voronoi cell and $q_{d,k}$ represents desired position the for the k^{th} vehicle. In [33–35], a method to add and remove agents from a formation was achieved by using a virtual structure to define safe operational envelopes. To accomplish this, a Voronoi tessellation was defined dynamically from the positions of the formation agents. The mentioned research uses these Voronoi tessellations for the purpose of allowing a formation to be fault tolerant. Whenever an agent is added to or removed from the formation, the distributed objective is to have each agent maximize the distance to its neighbors.

For this work, the Voronoi tessellation provides a method for developing the time-varying operational envelopes. The cell walls of the Voronoi tessellation would be modeled as constraints in the optimization framework because of this the a cell walls are defined as

$$\mathbb{V}_{lines,i} \cap \mathbb{V}_{lines,j} = \emptyset, \forall i, j \in n, i \neq j \tag{2.22}$$
$$\text{s.t. } \mathbb{V}_{lines,i} = \{w \in \mathbb{R}^2 | dist(q_{d,i}, w) = dist(q_{d,j}, w), \forall j \neq i\}$$

The key differences being that this equation is an equality rather then the inequality in (2.21) and that the sample set is taken from the Voronoi cell. With these lines a barrier cost may be developed to retain the vehicles within their designated cell in the subsequent chapters. Note, that for a 2-D environment, $\mathbb{V}_{lines,i}$, will consist of straight lines and points as depicted in Figure 2.3.

2.7 Continuous Curvature Paths

The motion of the virtual leader is another characteristic of the formation definition,

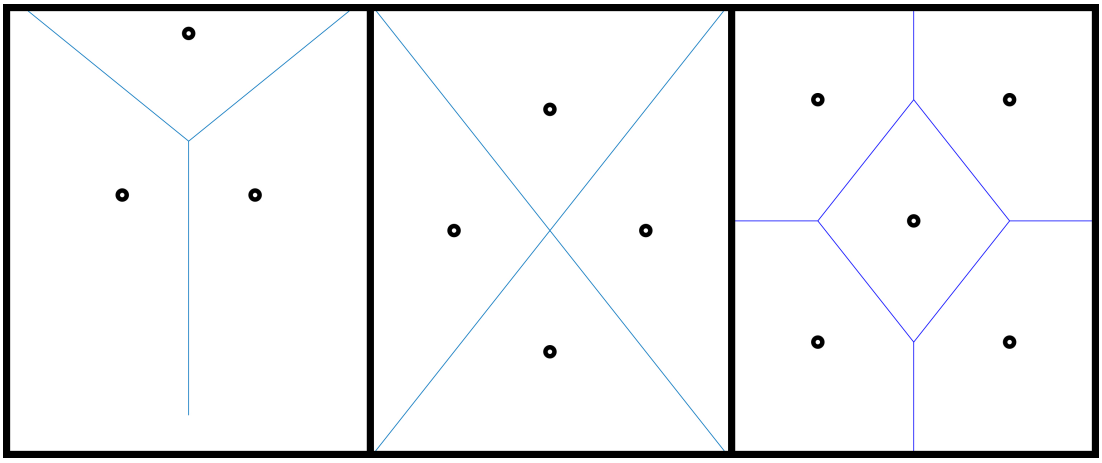

Fig. 2.3: This figure shows Voronoi tessellations of different configurations. The configurations include triangle (left), diamond (middle), and cross (right) formations. Originally the Voronoi diagram was developed to approximate a desirable location for a post office or store by applying the diagram to a map and population estimates. It assumed that a consumer would utilize the closest establishment. [1]

with a objective to develop continuous trajectories from the virtual leader. The virtual leader motion model shown in (2.2) allows for direct consideration of the path curvature. To directly consider the maximum curvature in planning, Dubins paths were developed to find the shortest path between any two oriented waypoints [36]. Given a constant forward velocity, minimum paths are found by either executing three maximum curvature circles (CCC) or a circle, line, then a circle (CLC). However, Dubins paths assume that the vehicle curvature can be changed instantaneously. Continuous curvature paths extend the idea of Dubins paths by employing a smooth transition between desired curvatures, known as clothoids, as shown in Figure 2.4. The smooth transition are achieved by linearly changing the curvature at a maximum curvature rate. In [14] constant curvature turns (CCTurn) replace the Dubins circle, although planning is nearly identical.

It was shown in [15] that a truncated form of (2.2) could be used to develop continuous curvature paths (CCPaths)[2]. Using a motion model for the clothoid provides the added benefit of time-indexing, enabling the paths to be used in trajectory tracking control laws.

While the discussion of planning shortest paths is left to [14], the generation of the

[2]The truncation comes from a lesser requirement for smoothness than $\mathcal{C}^4$.

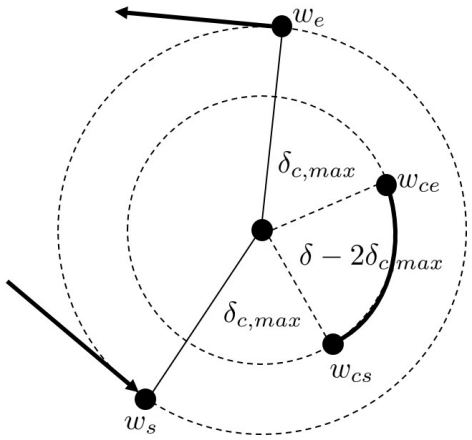

Fig. 2.4: Example of a continuous curvature turn where w_s represents the start of the transition clothoid, w_{cs} is where the path begins a circular arc, w_{ce} is where the circular arc ends and a clothoid begins, and w_e is the point where the clothoid ends.

CCTurn is important for understanding of the development of sufficiently smooth trajectories in Section 4.2. As shown in Figure 2.4, a CCTurn has the vehicle turn as quickly as possible from one straight line to another while considering constraints on curvature and its derivatives. This change in orientation is given by the deflection angle, β. The turn consists of three phases. The first phase is to move through a clothoid from zero curvature to the maximum curvature. The second phase is to execute a circular arc at maximum curvature. The third phase is to reflect the first clothoid and transition from maximum curvature to zero curvature. If β is small enough, the vehicle never reaches maximum curvature before transitioning back to zero curvature, thus never executing the second phase consisting of the maximum curvature arc.

Three parameters are key in constructing a CCTurn: the maximum curvature, κ_{max}, the maximum curvature rate, σ_{max}, and the maximum clothoid deflection angle, $\beta_{c,max}$. The latter being the amount of change in orientation when executing a clothoid to move from zero curvature to κ_{max}. Thus, the first and final phases of the CCTurn cause an aggregate change of orientation of $2\beta_{c,max}$ leaving the maximum curvature arc to effect the remaining $\beta - 2\beta_{c,max}$ of the deflection angle. Figure 2.4 depicts these phases by defining four waypoints.

In Chapter 4 the clothoid will extend from $\mathcal{C}^2$ to $\mathcal{C}^4$.

2.7.1 A Note on Using CCTurns

While CCTurns do not have a closed form solution, the clothoid in a CCTurn need only be computed a single time. Given a nominal value of w_s set to zero position and orientation, the clothoid of a right turn can be achieved by reflecting the clothoid position about the x-axis and changing the sign on κ and σ. To start the clothoid at a different orientation and position, all position values can be rotated by the desired orientation and then translated to the desired position. The curvature and curvature-rate values need not be adjusted.

Thus, the trajectory in a CCTurn can be quickly calculated by determining the deflection angle and starting point and then piecing together the individual elements (i.e., initial clothoid, circle, and terminal clothoid).

CHAPTER 3

Single Vehicle Design

The purpose of this Chapter is to develop the capabilities of the individual vehicles of the formation. The focus on the single vehicle design is due to the nature of the original objective of this work. That formation control can be accomplished by designating an operational area for each vehicle to maneuver freely in without being considered as breaking from the formation. The single vehicle capabilities that will be developed in this chapter will be as if the vehicle had no limitations on where it can maneuver. An operational envelope constraint will then be applied as part of the virtual structure designed in Chapter 4.

The first section of this chapter develops the dual-mode behavior-based MPC trajectory tracking algorithm that will be applied each vehicle of a formation. This algorithm is designed to track a trajectory when there are no detected obstacles and stay near a reference trajectory when there are obstacles impeding or close to the reference trajectory. The second section analyzes the convergence of the developed algorithm. This analysis provides two theorems that addresses what the robot will do when the desired trajectory is free from obstacles and when there are detected obstacles near the desired trajectory. The last section will then demonstrate the single vehicle capabilities in two examples. These examples show that the developed algorithm will control a single vehicle to track a reference trajectory in the presence of obstacles without collision.

3.1 Behavior-based Trajectory Tracking MPC Algorithm

The behavior-based MPC formulation is now used to create a trajectory tracking capability while simultaneously considering obstacle avoidance. The need to track a trajectory stems from the method in which formation control is implemented in Chapter 4.

The fundamental components of the algorithm are first discussed. This includes the sequencing of the behaviors, an approach for initializing behavior parameters, and cost

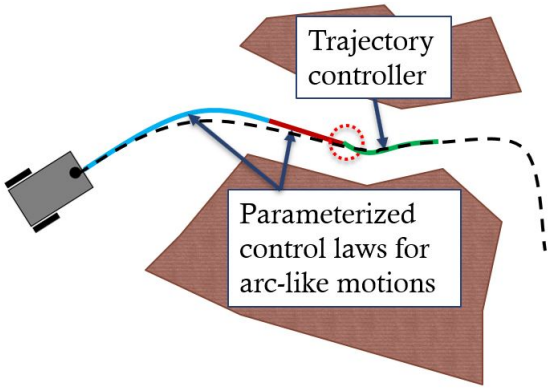

Fig. 3.1: This figure shows an example of a dual-mode arc-based trajectory. The dotted line denotes the desired trajectory for the robot. The robot plans its control using two arc-based controllers (blue and red lines) followed by a trajectory following controller (green line). Obstacles to be avoided are shown in brown.

definitions. This section ends with a statement of the trajectory tracking algorithm.

3.1.1 Behavior Sequence

The sequence of controllers consists of N arc-based controllers followed by a single trajectory tracking controller. The i^{th} arc controller is parameterized by $\theta_i = [v_i, \omega_i]^T$, denoting the desired set-point velocities for the i^{th} window. The final controller is the trajectory tracking controller. It does not have any parameters to be optimized. An illustration of a trajectory produced from this sequence can be seen in Figure 3.1. As is typical in dual-mode approaches, the properties of the final trajectory tracking mode are exploited for convergence.

3.1.2 Initialization of Behaviors

The defined sequence of the controllers allows for various initializations of the optimization process. This is done to warm start the optimization. This warm start is a method to get closer to the optimal value by quickly evaluating several key behaviors to formulate an educated guess of the optimal control.

For example, one behavior type used to initialize θ_i and τ_i is done via the scaling approach developed in [29]. This approach tests a series of arcs by setting $\theta_i = \theta_j$ and

dividing τ_i evenly over the horizon. Each (v, ω) pair is simulated and examined for a obstacle collision. In the event that a velocity pair is projected to collide with an obstacle, the amount of time until the collision is utilized to scale the velocities such that the same arc is executed at a slower speed, thus avoiding collision. This approach examines a subset of the velocity space and whichever (v, ω) pair returns the lowest cost is used as the pair to warm start the optimization.

More complex initializations could potentially be developed, but the following initialization behaviors were implemented and proved sufficient for the example:

1. Single-arc behavior: all horizons are set to execute the same velocity pair and $\tau_N = \tau_{N+1}$ (i.e., the terminal controller is not executed)

2. Single-arc and trajectory tracking behavior: same as single-arc except $\tau_N < \tau_{N+1}$ (i.e., the behavior ends with a tracking mode)

3. Pure tracking behavior: only the tracking window is evaluated $\tau_i = 0$

4. Repeat behavior: the previous optimization output is repeated

This initialization also assists in escaping local minima that are inherent in gradient-based optimization methods. By examining a wider breadth of the command the chances having the optimization get stuck in a local minima is lowered.

3.1.3 Cost Definitions

Fundamental to MPC is the definition of the cost to be optimized. The cost is used to both represent the objectives (desired values) and soft constraints (i.e., cost barriers to avoid undesirable values). The aggregate cost is developed through the combination of three individual costs, namely obstacle avoidance, instantaneous velocity, and trajectory convergence costs. Each cost is weighted by a value $\rho_i \in \mathbb{R}$. The first two costs are evaluated over the entire time horizon as part of the instantaneous cost, and the third cost is only a terminal cost.

Obstacle Avoidance

The obstacle avoidance cost used in this paper is defined as a logarithmic barrier function. This log barrier function relies on a minimum and maximum distance, d_{min} and d_{max}, respectively. The resulting cost is infinite when the vehicle is within d_{min} of an obstacle and a zero when d_{obs} is greater than d_{max}. This forms a soft constraint, such that it will not be numerically violated due to the infinite cost. This log barrier is written as

$$L_{j,avoid}(x) = \begin{cases} \rho_1[ln(d_{max} - d_{min}) - ln(d_{obs} - d_{min})] & d_{min} < d_{j,obs} \leq d_{max} \\ \infty & d_{j,obs} \leq d_{min} \\ 0 & d_{max} < d_{j,obs}, \end{cases} \tag{3.1}$$

where $d_{j,obs} = ||q(t;t_0) - q_{j,obs}||$ which is the distance from the vehicle to each detected obstacle. Figure 3.2 demonstrates this barrier function. It is important to note that this cost is summed over all of the points detected by the range sensor, meaning that each detected obstacle has it's own associated cost. This can be seen by re-examining the Figure 2.1, where the green detection points fall within d_{min} and d_{max}.

Instantaneous Velocities

A quadratic cost is implemented to encourage the vehicle to track the reference trajectory velocities, penalizing large deviations. This cost is defined as

$$L_{vel}(x) = \frac{\rho_2}{2}(v(t;t_0) - v_{traj}(t))^2 + \frac{\rho_3}{2}(\omega(t;t_0) - \omega_{traj}(t))^2, \tag{3.2}$$

where $v_{traj}(t)$ and $\omega_{traj}(t)$ represent the translational and rotational velocities of the trajectory at time t as defined in (2.6).

Terminal Cost

The terminal cost includes a penalty for not being at the desired state and a barrier

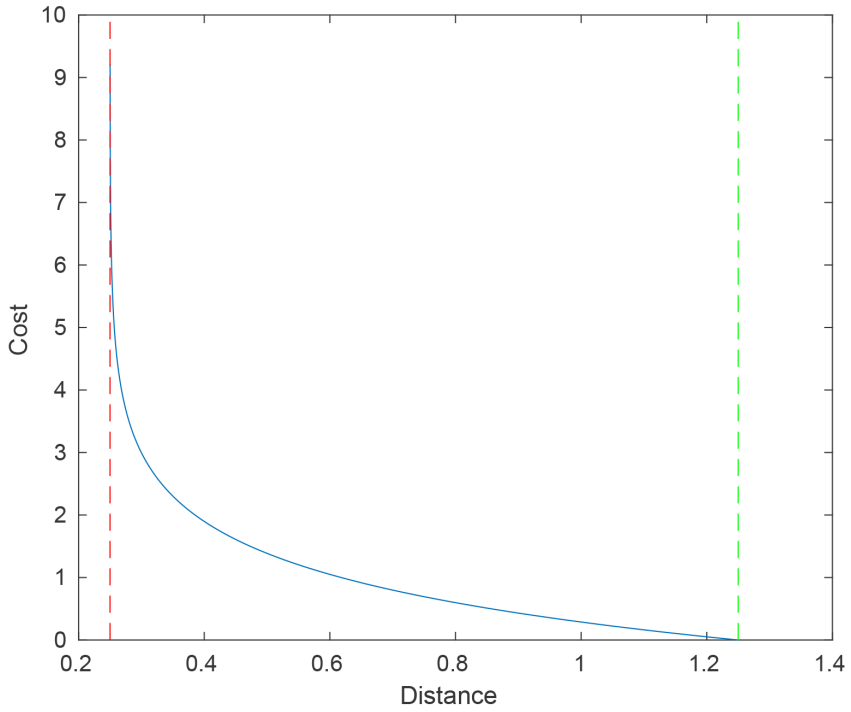

Fig. 3.2: This figure shows the obstacle avoidance logarithmic barrier cost as a function of distance from a detected obstacle. The blue line represents the cost at a specific distance. The red dashed line represents d_{min}, when the cost evaluates as infinity. The green dashed line represents d_{max}, which is when the barrier transitions to zero cost.

function to constrain the terminal position to be within μ of the desired terminal position. The terminal penalty is written as

$$\begin{aligned} \Phi_{traj} &= \rho_4 z^T P z, \\ \text{s.t. } z &= x_\epsilon - x_{\epsilon_d}, \end{aligned} \tag{3.3}$$

where P has already been defined in Section 2.2.1 as the solution to the ARE as in (2.12) to determine the feedback gain in (2.11). The terminal barrier is written as

$$\Phi_{barr} = \begin{cases} \rho_5[ln(\mu_{max} - \mu_{min}) - ln(\frac{\mu_{max}-\mu}{\mu_{max}})] & \mu_{min} < \mu \leq \mu_{max} \\ 0 & \mu \leq \mu_{min} \\ \infty & \mu_{max} < \mu \end{cases}, \tag{3.4}$$

where $\mu = ||q - q_d||$, μ_{max} is the distance from q_d that the terminal barrier allows, and μ_{min} represents the radius around q_d that is free from the barrier's influence. This is again a logarithmic barrier function to form a soft constraint for the optimization. This will generate a finite cost when with μ_{max} and an infinite cost when $\mu \geq \mu_{max}$.

Cost Summation

The total cost to be minimized thus takes the form:

$$\min_{(\theta,\tau)} J(\theta,\tau) = \int_{t_0}^{t_0+\Delta} \Big(\sum_{j=1}^{n_{obs}} L_{j,avoid}(x(\xi;t_0) + L_{vel}(x(\xi;t_0)) \Big) d\xi + \Phi_{traj}(x(t_0+\Delta;t_0)) + \Phi_{barr}(x(t_0+\Delta;t_0)) \tag{3.5}$$

where n_{obs} is the detected number of obstacles points.

3.1.4 Dual-mode Arc-Based MPC Tracking Algorithm

With the behavior sequencing and costs defined, the dual-mode arc-based MPC tracking algorithm is stated in Algorithm 1. Note, it is assumed that a higher-level planner exists to provide a desired trajectory.

Algorithm 1 Dual-Mode Arc-based MPC

---

1. Initialize Parameters θ and τ:
   (a) Calculate cost using previous values of θ and τ.
   (b) Calculate cost using variety of predefined values for θ and τ.
   (c) Choose parameters from step 1a and 1b that result in lowest cost.
2. If no initial solution can be found with an associated finite cost
   (a) Flag to higher-level planner that $x_d(t)$ is infeasible
   (b) Execute an avoidance behavior until new trajectory is planned
   (c) Move to step 1
3. Minimize $J(\theta, \tau)$ with respect to θ and τ, using parameters from 1c as an initialization.
4. Execute optimal control sequence for $\delta < \Delta$ seconds.
5. Repeat steps 1 through 5, updating t_0 by δ.

Algorithm 1, Step 2b refers to executing an avoidance behavior. This could be any number of strategies to either stop, execute a contingency plan, or attempt to stay near the trajectory. Step 2c is a similar case in that a number of higher-level planning methods could be executed. The specifics of the avoidance behavior must unavoidably consider the inter-action with the chosen higher-level planner. In this work these algorithms were not developed because the rest of Algorithm 1 proved to track a trajectory sufficiently in the examples provided.

3.2 Convergence Analysis

This section evaluates the steady-state characteristics of the trajectory tracking MPC formulation of Section 3.1. The formulation is evaluated in terms of the two general goals:

track the trajectory when it is sufficiently far from an obstacle and stay near it otherwise. The requirements for each goal are summarized by enumerating the characteristics that are required for each goal as assumptions. A theorem is stated for trajectory tracking when certain obstacle conditions are met and another theorem is stated for staying near the trajectory when the obstacle conditions are not met.

3.2.1 Converging to a trajectory

To converge to a trajectory, the following assumptions are given to provide a succinct overview of the problem setup:

The first three assumptions consider the characteristics of the costs that are used to show convergence. They can be seen to be valid through inspection of the costs in Section 3.1.3.

A1: (Strictly Positive Costs) Both the instantaneous and terminal costs are to be strictly positive for all time.

A2: (Cost Convexity in Terminal State Set[1]) The instantaneous cost is defined as strictly convex in the terminal state set, X_f, with a zero minimum when $x(t) = x_d(t)$.

A3: (Obstacle Avoidance) The instantaneous cost forms a cost barrier around obstacles such that $L(x, \kappa_i) \to \infty$ as $dist(q, \mathcal{B}^c_{free}) \to 0$, where $\mathcal{B}_{free}$ represents the collision free space and is finite outside $\mathcal{B}^c_{free}$. All other terms in the instantaneous cost produce finite costs.

An additional assumption on the instantaneous cost is used to ensure that it does not overpower the terminal cost in the convergence evaluation. Instead of being an characteristic of the cost definition, it is a characteristic of the choice on the gain values.

A4: (Small Gain) The gains ρ_2 and ρ_3 are sufficiently small for convergence.

A2 and A4 will be used to enable the velocity costs to be neglected during the convergence proof. However, in application it is show that these costs can have non-negligible gains.

The next two assumptions deal with the terminal cost and controller and are used for convergence analysis.

[1] Defined in A8

A5: (Convergent Terminal Controller) Employing $u(t) = \kappa_N(x(t))$ *when* $x(t) \in X_f$ *results in asymptotic*[2] *convergence of the* ϵ*-state.*

A6: (Terminal Cost) Within X_f*, the terminal cost is the infinite horizon cost for tracking.*

The following two assumptions declare that the controllers and terminal set are well-defined.

A7: (Executable Control Inputs) Employing $u(t) = \kappa_i(x(t), \theta_i) \in U \ \forall x \in X$ *and* $\theta_i \in \Theta$.

A8: (Terminal Set) X_f *is defined as a ball around zero,* $\mathcal{B}_\mu$*, where the terminal state is considered in terms of the* ϵ*-point as* $z = x_\epsilon - x_{\epsilon_d}$.

Assumptions on obstacle detection are now given. These are purely for trajectory tracking and will often not be satisfied. When they are violated, it implies that the desired trajectory runs close to or through an obstacle. In such a case, the vehicle should not converge to the trajectory, but rather just stay near.

A9: (X_f *is free of obstacles) The terminal state set,* X_f*, contains the reference trajectory and is a minimum distance of* d_{max} *from any obstacles.*

A10: (No New Obstacle Detected) No new obstacle within d_{max} *of the trajectory from one time step to the next.*

A11: (Terminal Position Barrier) The terminal cost forms a cost barrier around the desired terminal position, such that $\Phi(x) \to \infty$ *as* $dist(q, \mathcal{B}_\mu^c) \longrightarrow 0$ *and* $\Phi(x, \kappa_i) \to 0$ *as* $dist(q, \mathcal{B}_\mu) \to 0$

Finally, an assumption is given on the ability to produce at least one feasible solution. A theorem will then be stated to show that if this one solution is found then future solutions can be found.

A12: (Initialization of solution) An initial solution for the parameters satisfying tracking requirements and obstacle avoidance requirements can be found.

Theorem 1. *Given A1 through A12, Algorithm 1 will cause the state to asymptotically converge to the desired trajectory.*

[2] Note that the controller used actually has exponential convergence, but asymptotic is all that is used in the proof

Proof. The proof comes in two parts: feasibility and convergence. The feasibility of the trajectory is concerned with obstacle avoidance and executability of the control inputs. The latter is given through A7. Obstacle avoidance is guaranteed through a combination of A3, A5, and A7 through A12. Given an obstacle free trajectory, as in A12, the cost of that trajectory will be finite. At the next iteration, a finite-cost trajectory can be found by appending the control from the terminal controller to the previous control trajectory. A7 through A10 ensure that X_f will be obstacle free and A5 ensures that the tracking control can stay in X_f.

The convergence of the control is established by showing that C1 through C4 are satisfied. C1 through C3 are satisfied by A5, A7, and A8, respectively. To satisfy C4, it must be shown that $\frac{\partial \Phi}{\partial x} f(x, \kappa_N) + L(x, \kappa_N) < 0$. The instantaneous cost term is dealt with using A2 and A4. A2 informs us that the cost will go to zero as $x(t) \to x_d(t)$. As L is strictly positive, it gives a lower bound on convergence of $\frac{\partial \Phi}{\partial x} f(x, \kappa_N)$ as $\frac{\partial \Phi}{\partial x} f(x, \kappa_N) < -L(x, \kappa_N) < 0$. A result of A2 is that $L \to 0$ and, since A4 states that the gain is small, the influence of L can be neglected. It is now left to show that $\frac{\partial \Phi}{\partial x} f(x, \kappa_N) < 0$. Since $\Phi = z^T P z$, with P defined as the solution to the ARE used to calculate the terminal controller, it is well-established that Φ is a CLF under the controller developed using $u = -R^{-1} B^T P z$ [24], where in this case u is actually u_ϵ. This shows that $x_\epsilon(t) \to x_{\epsilon_d}(t)$. In [15] it was shown that as $x_\epsilon(t) \to x_{\epsilon_d}(t)$, $x(t) \to x_d(t)$. □

Remark 1. *The arguments made in the proof could be made for any trajectory tracking controller if the terminal cost is updated to be a CLF under the tracking controller.*

3.2.2 Maintaining Proximity to Trajectory

As mentioned, in the presence of obstacles, assumptions A9 and A10 will quickly be violated. It is important that when these assumptions are violated that the optimization produces a trajectory that stays near the desired trajectory.

Theorem 2. *Given A3, A7, and A11, the resulting trajectory produced by Algorithm 1 will either result in an infinite cost or an obstacle-free trajectory that can move the vehicle to within μ_{max} of the desired trajectory.*

Proof. A3 will ensure any resulting solution that produces a finite cost will be obstacle free. A7 ensures the resulting trajectory is executable. A11 ensures the resulting trajectory comes within μ_{max} of the desired trajectory. □

A result of Theorem 2 is that any solution returned by the optimization routine can be quickly evaluated. If the resultant cost is finite, then the vehicle will plan to move "close" to the trajectory. If it is infinite, then the higher-level planner can be notified that the trajectory must be re-planned.

3.3 Example

This section demonstrates the dual-mode arc-based MPC trajectory tracking algorithm in two examples of different complexity. These simulations are developed in Matlab with the optimization being done via gradient descent with an Armijo line search [37]. Both examples are executed with the same cost weights and values for d_{min} and d_{max}.

$$\rho_1 = 0.15, \quad \rho_2 = 0.15, \quad \rho_3 = 0.60, \quad \rho_4 = 0.65,$$
$$d_{min} = 0.25, \quad d_{max} = 1.00$$

In both examples the trajectory tracking/following is possible without re-planning by a higher-level planner, despite significant portions of the trajectories being inside an obstacle. The weighting on the terminal barrier, ρ_5, is not presented because, in these examples, it was never utilized and therefore weightings could not be tested.

3.3.1 Example 1

The first example is a typical environment with a poorly placed reference trajectory that passes through obstacles, as shown in Figure 3.3. The trajectory was intentionally

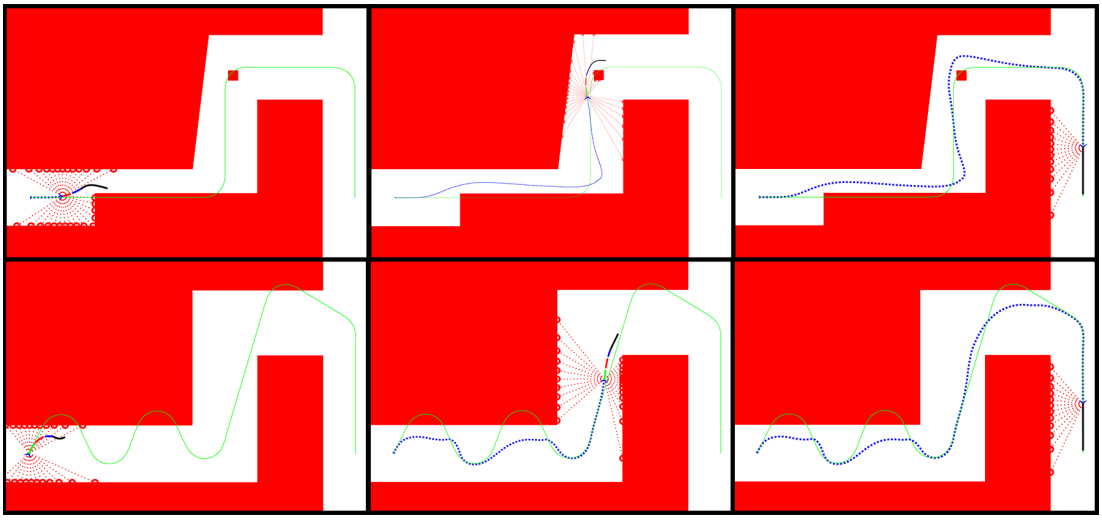

Fig. 3.3: This figure displays the two examples. The top row reflects the first example and the bottom row the second. The left and middle image displays more obstacle avoidance instances from each example. The right image shows an overview of the whole simulation. The green line represents the desired trajectory. The blue line in this image shows the path that the vehicle traveled, showing that no collisions occurred during the simulation.

embedded into an obstacle to demonstrate that the framework can successfully direct the vehicle along the obstacle to the desired trajectory. An error plot can be examined in Figure 3.4. The large position error that can be seen in the first 15 seconds is due to the measurement model. When in a narrow corridor the lidar sensor will detect multiple objects near the vehicle, creating a high cost to remain in the area. And the horizon predicts a much smaller cost in the future since there is not a high number of detected obstacles.

This example shows the importance of the initialization step in Algorithm 1. If there was no initialization step then the optimization cycle would encounter a local minimum at the first wall. Similarly, the obstacle placed in the middle of the trajectory later in the simulation tested the ability of the initialization of different behaviors. The behaviors provided in Section 3.1.2 proved sufficient to navigate around the obstacle. It can be examined that as the trajectory is free of obstacles the vehicle can converge via the final controller of dual-mode MPC framework, the trajectory tracking controller.

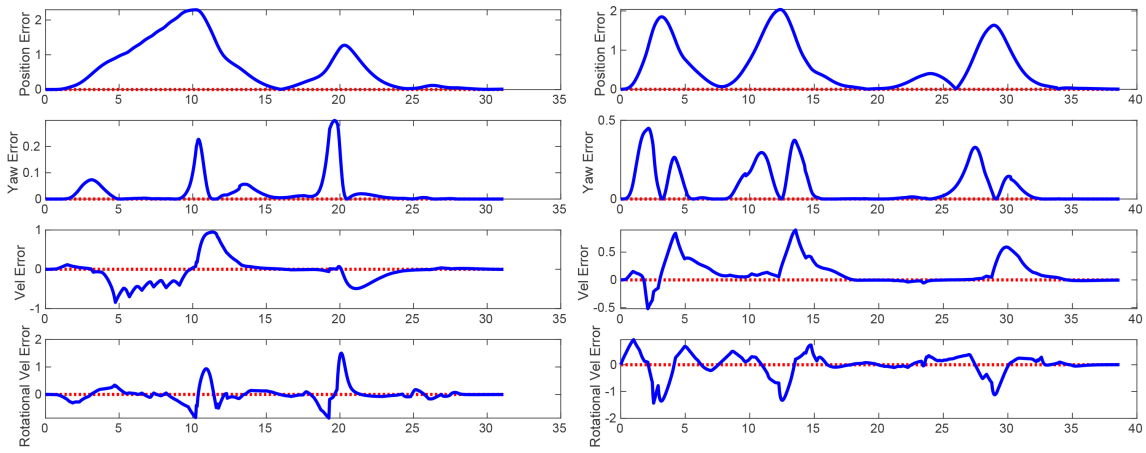

Fig. 3.4: This figure displays the state errors of both examples. The first example is on the left and the second example is on the right. Note, a negative velocity in this plot corresponds to a velocity greater than the desired trajectory velocity.

3.3.2 Example 2

The second example shows a open corridor with a trajectory generated more carelessly, as shown in Figure 3.3. When the trajectory runs through a wall in this scenario the tracking algorithm slows the vehicle down to stay near the trajectory and continues tracking once the desired position emerges from the wall. In the case where the trajectory goes near a wall, not through, the algorithm moves off the trajectory because of the influence of the obstacle avoidance costs. An error plot of this example can be examined in Figure 3.4. The large position errors in this example can be directly linked to each time the trajectory passes through a wall.

CHAPTER 4

Multi-Vehicle

The previous Chapter focused on the development of the single vehicle. This Chapter focuses on developing the necessary attributes of the Multi-Vehicle system to add onto the single vehicle design. Section 4.1 develops the trajectory generation of each vehicle within the formation. Section 4.2 discusses the generation of the trajectory for the virtual leader. Section 4.3 introduces the operational envelope for each vehicle to maneuver with an example of the formation control method shown in Section 4.4.

4.1 The Virtual Structure

This section derives the generic trajectory that each agent should nominally follow assuming a sufficiently smooth virtual leader trajectory. It is also shown that the structure of the virtual leader motion model can be exploited to evaluate characteristics of the resulting follower trajectories.

4.1.1 Follower Trajectory Generation

Assuming that the virtual leader motion is sufficiently smooth, $q_l(t) \in \mathbb{C}^4$, the desired trajectories for each follower can be derived using $q_l(t)$, its derivatives, and the desired offset of the follower. Denoting the reference trajectory for agent i as $q_{d_i}(t)$, Lemma 3 gives its derivatives. As a matter of notation, given a scalar ν and the $\frac{\pi}{2}$ rotation matrix, J, the notation $\hat{\nu}$ forms the skew symmetric matrix

$$\hat{\nu} = \nu J = \nu \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}. \tag{4.1}$$

Lemma 3. *Given a desired relative offset for agent i, denoted r_i, and a leader trajectory, $q_l(t) \in \mathbb{C}^4$, a three-times continuously-differentiable trajectory for agent i can be produced*

using the following equations.

$$\begin{aligned} q_{d_i} =& R(\psi_l) r_i + q_l \\ \dot{q}_{d_i} =& R(\psi_l) \hat{\omega}_l r_i + \dot{q}_l \\ \ddot{q}_{d_i} =& R(\psi_l)(\hat{\omega}_l^2 + \hat{\alpha}_l) r_i + \ddot{q}_l \\ q_{d_i}^{(3)} =& R(\psi_l)(\hat{\omega}_l^3 + \hat{\zeta}_l + 3\hat{\omega}_l\hat{\alpha}_l) r_i + \dddot{q}_l \end{aligned} \tag{4.2}$$

where $\dot{\psi}_l = \omega_l$ *and* $\ddot{\psi}_l = \alpha_l$ *can be obtained as in (2.6). The third derivative introduces* $\psi_l^{(3)} = \zeta_l$*, which can be obtained by differentiating* $\ddot{\psi}_l$ *and written as*

$$\begin{aligned} \zeta_l =& (\dot{x}_l y_l^{(4)} + \ddot{x}_l y_l^{(3)} - x_l^{(3)} \ddot{y}_l - x_l^{(4)} \dot{y}_l) v^{-2} \\ & - 2a_l (\dot{x}_l y_l^{(3)} - x^{(3)} \dot{y}) v_l^{-3} + 2a_l^2 \omega_l v_l^{-1} \\ & - 2j_l \omega_l v_l^{-1} - 2a_l \alpha_l v_l^{-1} \end{aligned} \tag{4.3}$$

where $j = \dot{a}$ *and can be written as*

$$j_l = (\ddot{x}_l^2 + \dot{x}_l x_l^{(3)} + \dot{y}_l y_l^{(3)} + \ddot{y}_l^2) v_l^{-1} - a_l^2 v_l^{-1} \tag{4.4}$$

Proof. Note first that q_{d_i} in (4.2) is a direct copy of (2.20), provided again for convenience. The summation in q_{d_i} allows for the left and right portions to be considered separately, where the derivatives of $q_l(t)$ are trivial. The left portion of each part of (4.2) uses the time derivative of a rotation matrix as shown in [38] and provided here for convenience,

$$\frac{d}{dt}(R(\psi_l)) = R(\psi_l)\hat{\omega}_l, \quad \hat{\omega}_l = \begin{bmatrix} 0 & -\omega_l \\ \omega_l & 0 \end{bmatrix}. \tag{4.5}$$

And both (4.3) and (4.4) are obtained using the time derivative of the acceleration equations in (2.6), which is derived by utilizing the product rule. □

It is important to recall that the necessity of the third derivative for the follower stems from the use of the zero error ϵ-trajectory tracking controller discussed in Section 2.2.1.

4.1.2 Smoothness of Virtual Leader Motion Model

The virtual leader motion model in (2.2) is restated for convenience:

$$x_l = \begin{bmatrix} q_{l_1} \\ q_{l_2} \\ \psi_l \\ \kappa_l \\ \sigma_l \\ \gamma_l \end{bmatrix}, \dot{x}_l = \begin{bmatrix} v_l \cos(\psi_l) \\ v_l \sin(\psi_l) \\ v_l \kappa_l \\ \sigma_l \\ \gamma_l \\ u_l \end{bmatrix}.$$

Which is now evaluated in terms of its smoothness properties and the associated curvature properties for each follower. Using a direction vector, $h_l = \begin{bmatrix} \cos(\psi_l) & \sin(\psi_l) \end{bmatrix}^T$, the virtual leader model will produce trajectories in $\mathcal{C}^4$ as expressed as in the following lemma.

Lemma 4. *Trajectories produced using (2.2), will exist in $\mathcal{C}^4$.*

Proof. The derivatives of q_l can be written as

$$\begin{aligned} \dot{q}_l &= v_l h_l \\ \ddot{q}_l &= v_l^2 \kappa_l J h_l \\ q_l^{(3)} &= v_l^2 \sigma_l J h_l - v_l^3 \kappa_l^2 h_l \\ q_l^{(4)} &= v_l^2 \gamma_l J h_l - 3 v_l^3 \sigma_l \kappa_l h_l - v_l^4 \kappa_l^3 J h_l \end{aligned} \tag{4.6}$$

The value for $\dot{q}_l$ comes directly by substituting h_l into (2.2). Further derivatives use the time derivative of h_l which is derived as,

$$\frac{d}{dt} h_l = \dot{\psi}_l \begin{bmatrix} -\sin\psi \\ \cos\psi \end{bmatrix} = v_l \kappa_l J h_l. \tag{4.7}$$

These further derivatives rely repeatedly on the product rule and the relation that $J^2 = -I_2$. All terms in $q_l^{(4)}$ are continuously differentiable, thus satisfying $q_l \in \mathcal{C}^4$. □

An immediate result of (4.6) is that the generic form of the follower trajectory in (4.2) can be updated to directly account for the structure of the virtual leader motion. We state the first and second derivatives of $q_{d_i}(t)$ as they are used in the development of the follower trajectory properties. The derivatives can be stated as

$$\begin{aligned} \dot{q}_{d_i} =& \kappa_l v_l R(\psi_l) J r_i + v_l h_l \\ \ddot{q}_{d_i} =& R(\psi_l)(-v_l^2 \kappa_l^2 I_2 + v_l \sigma_l J) r_i + v_l^2 \kappa_l J h_l \end{aligned}, \tag{4.8}$$

where $\omega_l = \kappa_l v_l$, $\alpha_l = \sigma_l v_l$, and the derivatives of (4.6) are employed.

4.1.3 Curvature Properties for Follower Trajectories

The virtual leader motion model allows for direct limitations on the leader's curvature and its derivatives. As trajectory curvature can directly correlate to executability of the trajectory for physical systems, it is important to understand how the virtual leader's curvature relates to the curvature of the follower vehicles.

Recall that curvature can be defined as $\kappa = \frac{\omega}{v}$ where the velocities can be derived directly from a trajectory through (2.6). Note that these velocities can also be represented in vector form as

$$v_{d_i} = \sqrt{\dot{q}_{d_i}^T \dot{q}_{d_i}}, \; \omega_{d_i} = \frac{-\dot{q}_{d_i}^T J \ddot{q}_{d_i}}{\dot{q}_{d_i}^T \dot{q}_{d_i}}. \tag{4.9}$$

An immediate concern for the follower trajectory is to understand when the virtual structure motion will demand an agent to execute infinite curvature, which is addressed in the following lemma.

Lemma 5. *Given a desired offset for agent i, written as $r_i = \begin{bmatrix} r_{i_1} & r_{i_2} \end{bmatrix}^T$, an infinite curvature will be commanded for agent i if and only if (i) $r_{i_1} = 0$ and (ii) $\kappa_l = \frac{1}{r_{i_2}}$.*

Proof. An infinite curvature command will only come when $v_{d_i} = 0$ (or equivalently if $v_{d_i}^2 = 0$). Equation (4.9) for v_{d_i} can be used with (4.8) and $v_{d_i}^2$ can be simplified algebraically to

$$v_{d_i}^2 = v_l^2 \kappa_l^2 r_i^T r_i - 2 v_l^2 \kappa_l r_{i_2} + v_l^2. \tag{4.10}$$

Solving for the value of κ_l where (4.10) is zero, v_l^2 can be factored out and the quadratic equation used to produce

$$\kappa_l = \frac{r_{i_2} \pm \sqrt{-r_{i_1}^2}}{r_i^T r_i}, \tag{4.11}$$

which only has a real solution when $r_{i_1} = 0$ (condition *(i)*). In that case, condition *(ii)* directly falls out by substituting $r_{i_1} = 0$ directly into (4.11) which is written as

$$\kappa_l = \frac{r_{i_2} \pm \sqrt{0}}{\begin{bmatrix} 0 & r_{i_2} \end{bmatrix} \begin{bmatrix} 0 \\ r_{i_2} \end{bmatrix}} = \frac{1}{r_{i_2}}, \tag{4.12}$$

□

An interesting result from Lemma 5 is that only the vehicles that are to move directly "along side" the virtual leader run the risk of having a zero velocity, or, equivalently, turning in place.

However, it may be desirable to limit the virtual leader motion in such a way that the desired curvature for any follower agent lies below some threshold. This is addressed in the following lemma.

Lemma 6. *The curvature for agent i will stay below the maximum threshold κ_{max_i} if, for all time t,*

$$\kappa_{max_i} > \frac{v_l \kappa_l^3 r_i^T r_i + [\sigma_l,\ 2v_l\kappa_l^2] r_i + v_l \kappa_l}{v_l \left(\kappa_l^2 r_i^T r_i - 2\kappa_l r_{i_2} + 1\right)^{\frac{3}{2}}} \tag{4.13}$$

where time indices on $\kappa_l(t)$ and $\sigma_l(t)$ have been omitted for sake of brevity.

Proof. The proof is trivial in nature as the right-hand side of (4.13) is the equation for agent i's desired curvature. It can be obtained by combining (4.8) with (4.9), using the relation $\kappa_{d_i} = \frac{\omega_{d_i}}{v_{d_i}}$, and simplifying algebraically. □

Lemma 6 may appear difficult to satisfy at first glance since it requires solving for all possible combinations of κ_l and σ_l through the entirety of the trajectory. This is true, generally. However, it will be shown in Section 4.2 that by using a clothoid approach to

virtual leader planning, all possible combinations of κ_l and σ_l occur during the clothoid, so only the clothoid transition need be evaluated.

4.2 Continuous Curvature Virtual Leader Trajectory

The virtual leader trajectory is required to have both the position and orientation be three times continuously differentiable as per (4.2). An example method to create a sufficiently smooth trajectory is now shown using clothoids as an inspiration. In this section, two minor contributions are made to the clothoid techniques. First, a sufficiently smooth clothoid is created to satisfy the continuity conditions for a virtual leader. Second, a simplified waypoint planning technique is presented to rapidly plan paths given a sequence of desired position-based waypoints.

4.2.1 A Sufficiently Smooth Clothoid

The virtual leader requires the generation of a trajectory that is three-times continuously-differentiable in orientation and position, which can simultaneously be accomplished with a trajectory in $\mathcal{C}^4$ as described in Lemma 3. As (2.2) is sufficiently smooth, it is used to design the clothoid. Recall that the clothoid is used to transfer the vehicle from a straight line along the tangent of an outer circle to a smaller concentric circle where the vehicle is then able to execute its maximum curvature as depicted in Figure 2.4. The straight line corresponds to κ_l and its derivatives being equal to zero and the inner circle corresponds to $\kappa_l = \pm\kappa_{max}$ with the derivatives again equal to zero.

Thus, once started, the clothoid need only consider κ_l, its derivatives, and the amount of time to stay on the inner circle. The final three elements of x_l in (2.2) consisting of κ_l and its derivatives is denoted as $x_{\kappa_l} \in \mathbb{R}^3$. The resulting state dynamics form a triple-integrator linear system. For such a constrained system, the fastest way to achieve κ_{max} is to use bang-bang control, e.g. [24]. The control strategy will proceed as follows[1].

1. Use bang-bang control to get from $\begin{bmatrix}\sigma & \gamma\end{bmatrix}^T = \begin{bmatrix}0 & 0\end{bmatrix}^T$ to $\begin{bmatrix}\sigma & \gamma\end{bmatrix}^T = \begin{bmatrix}\sigma_{max} & 0\end{bmatrix}^T$

[1] Note that only the convergence to κ_{max} is considered. To move from κ_{max} to zero or from zero to $-\kappa_{max}$ the control inputs need only be reversed.

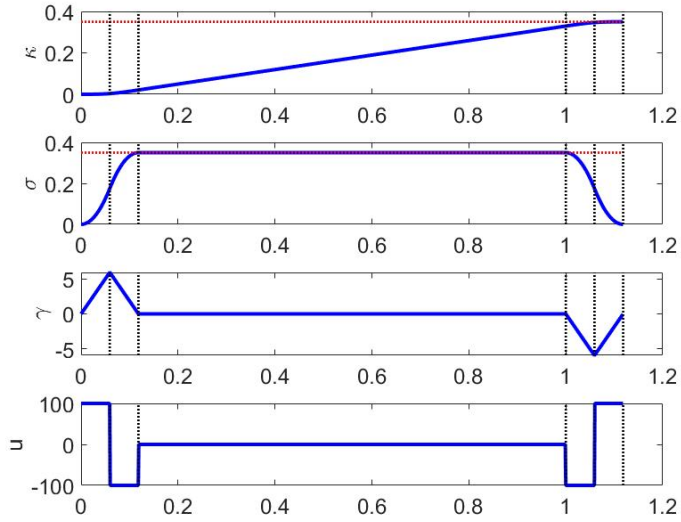

Fig. 4.1: An illustration of the construction of x_{κ_l} using bang-bang control. The figures from top to bottom are κ_l, σ_l, γ_l, and u_l. The blue line shows the value over time, the red shows bounds, and the black lines show the switching times.

2. Coast at $\sigma = \sigma_{max}$
3. Use bang-bang control to get from $\begin{bmatrix}\sigma & \gamma\end{bmatrix}^T = \begin{bmatrix}\sigma_{max} & 0\end{bmatrix}^T$ to $\begin{bmatrix}\sigma & \gamma\end{bmatrix}^T = \begin{bmatrix}0 & 0\end{bmatrix}^T$ at the precise time that $\kappa_l = \kappa_{max}$

To define the control, four parameters are first introduced.

- τ_{bb}: The time that the control is held constant during bang-bang maneuvers
- $\Delta\kappa_{bb}$: The change in curvature during a bang-bang maneuver
- $\Delta\kappa_{\sigma_{const}}$: The change in curvature during coast when $\sigma_l = \sigma_{max}$
- $t_{\sigma_{const}}$: The time at which coast ends

These parameters, and the resulting trajectory for x_{κ_l}, are shown in Figure 4.1. The control to achieve κ_{max} is stated in the following lemma.

Lemma 7. *Given the dynamics for κ_l in (2.2) and associated constraints on σ_l and u_l, the following control will move the system from $x_{\kappa_l} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ to $\begin{bmatrix} \kappa_{max} & 0 & 0 \end{bmatrix}^T$ in the minimum time.*

$$u_l(t) = \begin{cases} u_{max} & t \in [0,\ \tau_{bb}) \\ -u_{max} & t \in [\tau_{bb},\ 2\tau_{bb}) \\ 0 & t \in [2\tau_{bb},\ t_{\sigma_{const}}) \\ -u_{max} & t \in [t_{\sigma_{const}},\ t_{\sigma_{const}} + \tau_{bb}) \\ u_{max} & t \in [t_{\sigma_{const}} + \tau_{bb},\ t_{\sigma_{const}} + 2\tau_{bb}) \end{cases} \tag{4.14}$$

where

$$\tau_{bb} = \sqrt{\frac{\sigma_{max}}{u_{max}}},\ t_{\sigma_{const}} = \frac{\Delta\kappa_{\sigma_{const}}}{\sigma_{max}} + 2\sqrt{\frac{\sigma_{max}}{u_{max}}} \tag{4.15}$$

and

$$\Delta\kappa_{bb} = u_{max}\left(\frac{\sigma_{max}}{u_{max}}\right)^{\frac{3}{2}},\ \Delta\kappa_{\sigma_{const}} = \kappa_{max} - 2\Delta\kappa_{bb} \tag{4.16}$$

Proof. It is important to recall the constraints, $|u_l| \leq u_{max}$, $|\sigma_l| \leq \sigma_{max}$, and $|\kappa_l| \leq \kappa_{max}$. The solution to get to κ_{max} can be viewed as changing σ_l as quickly as possible to σ_{max}, executing σ_{max} for an interval, and, at the last moment possible, moving from σ_{max} to zero. The proof hinges on the fact that bang-bang control is the fastest way to move a linear system from one state to the next considering constraints on the control inputs [24]. Bang-bang control has two control intervals. In each interval, the control is held constant at an extreme. Due to the symmetry in the upper and lower constraints, these intervals are equivalent (τ_{bb} represents the length of a single time interval).

To find the switching times and curvature change values, the solution to a linear, time-invariant (LTI) system is employed, e.g., [23]. Namely, given $\dot{x}(t) = Ax(t) + Bu_{const}$, the solution at time t for $x(t)$ is given by the equation

$$x(t) = \underbrace{\exp^{A(t-t_0)}\, x(t_0)}_{\text{zero-input}} + \underbrace{\int_{t_0}^{t} \exp^{A(t-\nu)}\, B d\nu u_{const}}_{\text{zero-state}}. \tag{4.17}$$

In the case of $x_{\kappa_l} = \begin{bmatrix} \kappa_l & \sigma_l & \gamma_l \end{bmatrix}^T$, the exponential and integral portions can be written as

$$\begin{aligned} \exp^{A(t-t_0)} &= \begin{bmatrix} 1 & t-t_0 & \frac{1}{2}(t-t_0)^2 \\ 0 & 1 & t-t_0 \\ 0 & 0 & 1 \end{bmatrix} \\ \int_{t_0}^{t} \exp^{A(t-\nu)} B d\nu &= \begin{bmatrix} \frac{1}{6}(t-t_0)^3 \\ \frac{1}{2}(t-t_0)^2 \\ t-t_0 \end{bmatrix} \end{aligned} \tag{4.18}$$

Assuming that the initial time is 0 (which can generally be done since this is an LTI system), τ_{bb} can be found by using (4.17) and (4.18) over the first two time intervals. The solution from the first time interval can be fed directly into the zero-input portion of the second time interval as:

$$\sigma(2\tau_{bb}) = \sigma_{max} = \left(\frac{\tau_{bb}^2}{2} + \tau_{bb}^2 - \frac{\tau_{bb}^2}{2}\right) u_{max}, \tag{4.19}$$

where the first two τ_{bb} terms come from propagating the solution of the first horizon through the zero-input portion on the second horizon. The amount of change in curvature over the bang-bang maneuver, $\Delta\kappa_{bb}$, can then be solved for in a similar fashion using the solution for τ_{bb}.

The bang-bang control will be reversed to send σ_l back to zero by the end of the horizon. Over the reversed bang-bang control, the change in curvature will be the same as the as the original bang-bang control, resulting in $2\Delta\kappa_{bb}$ occurring over the combined bang-bang maneuvers. Thus, to achieve κ_{max} a total curvature change of $\kappa_{max} - 2\Delta\kappa_{bb}$ must occur during the constant σ_{max} interval. Since σ_l is constant over that interval, the curvature in that interval can be written as $\kappa_l(t) = \sigma_{max}(t - 2\tau_{bb}) + \Delta\kappa_{bb}$. The solution of $t_{\sigma_{const}}$ can be solved for from the following equation.

$$\begin{aligned} \kappa_l(t_{\sigma_{const}}) &= \kappa_{max} - \Delta\kappa_{bb} \\ &= \sigma_{max}(t_{\sigma_{const}} - 2\tau_{bb}) + \Delta\kappa_{bb}. \end{aligned} \tag{4.20}$$

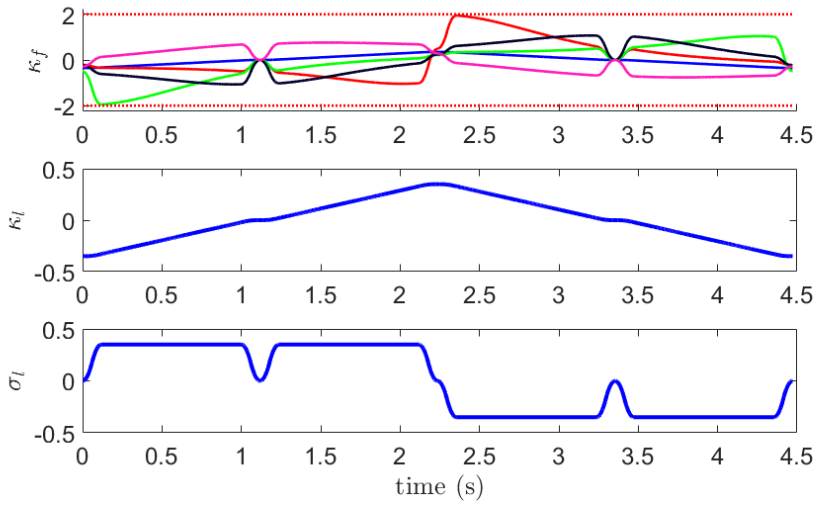

Fig. 4.2: This figure depicts the curvature verification for the formation in Section 4.4. The bottom and middle images show σ_l and κ_l over the four clothoid windows. The top image shows the resulting curvature for each agent in the formation, calculated using (4.13).

□

One of the major benefits to using a CCTurn approach for the virtual leader path is that all possible combinations κ_l and σ_l are contained in the within the clothoid (both for achieving κ_{max} and κ_{min}). Thus, to see if the trajectory will violate the curvature restraints in Lemma 6, one only need to verify each offset over four clothoid intervals: $-\kappa_{max}$ to 0, 0 to κ_{max}, κ_{max} to 0, and 0 to $-\kappa_{max}$. Figure 4.2 depicts the verification for the example in Section 4.4.

4.2.2 Virtual Leader Trajectory For Structured Environments

While Dubins paths and CCPaths can be used to find the shortest path that moves between a series of waypoints, often passing exactly through each waypoint is not completely necessary. For example, consider a series of waypoints that are taken from a graph representing building corridors. The waypoints may correspond to the center-points of corridor intersections. A planned path through the corridors to get from point A to point B could include multiple intermediary points. It may not be necessary to move directly through those points, but simply to move "near" them as depicted in Figure 4.3. Thus, this

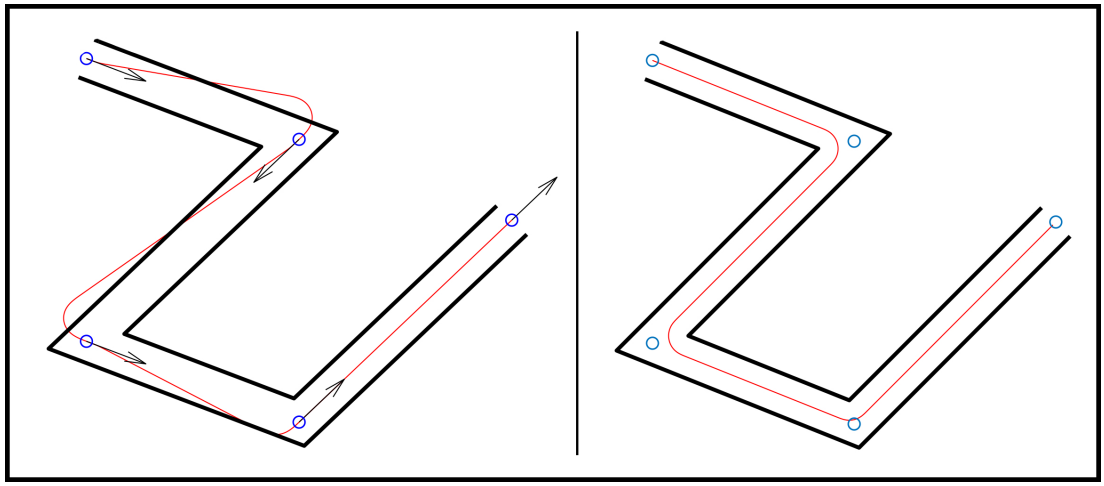

Fig. 4.3: Example of path points and a CCPath. The left depicts a shortest path between oriented waypoints and the right shows the path generated using position only waypoints. The blue circles represent the waypoints used to generate the CCPath and red shows the path itself. The bold black lines were generated by offsetting the line crated by the waypoints.

section presents a method to use clothoids to plan a trajectory from point A to B using intermediary points to define required turns, not points that have to be traversed exactly.

The nominal path is defined as an ordered set of n positions to be visited, denoted as Q, where

$$Q = \{q_1, q_2, \ldots, q_n\},$$

and the i^{th} waypoint is in $\mathbb{R}^2$.

As mentioned, the CCTurn forms a fundamental component in the shortest paths in CCPaths [14]. In this work, CCTurns are used to perform a turn between two subsequent line segments made from the ordered set of positions.

The process for rapidly assembling the turns at each waypoint can be summarized as:

1. Determine the deflection angle around the turn i, denoted δ_i. This defines the nominal value for the end waypoint, q_e^{nom}.

2. Rotate the CCTurn so the starting point tangent line is parallel to the angle from q_{i-1}-to-q_i, denoted $\psi_{i-1,i}$.

3. Find the vector that will point from $q_{s,i}$ to $q_{e,i}$.

4. Use the law of sines to solve for the distance between q_i and $q_{e,i}$.

5. Translate the CCTurn into place.

These parameters and their relations are shown in Figure 4.4. Defining the start and end points of each turn is sufficient for designing the trajectory as straight line segments will be used to connect the turns.

The *deflection angle* at waypoint i can be defined using the unit vectors parallel to line segments q_{i-1}-to-q_i and q_i-to-q_{i+1} as

$$\bar{\eta}_{i-1,i} = \frac{q_i - q_{i-1}}{d_{i-1,i}}, \; \bar{\eta}_{i,i+1} = \frac{q_{i+1} - q_i}{d_{i-1,i}} \tag{4.21}$$

where $d_{i-1,i} = ||q_i - q_{i-1}||$. The deflection angle about waypoint i can be defined as

$$\delta_i = acos(\bar{\eta}_{i-1,i}^T \bar{\eta}_{i,i+1}). \tag{4.22}$$

The deflection angle defines the nominal ending waypoint, $w_{e,i}^{nom}$, on the nominal CCTurn. To orient the CCTurn so that $w_{e,i}$ will align with $\bar{\eta}_{i,i+1}$, the nominal *CCTurn is rotated* by the orientation of $\bar{\eta}_{i-1,i}$, denoted as $\psi_{i-1,i}$. This allows for the direct calculation of the *vector that will point* from $q_{s,i}$ to $q_{e,i}$ as

$$\eta_{e,i} = R(\psi_{i-1,i})\big(q_{e,i}^{nom} - q_{s,i}^{nom}\big), \tag{4.23}$$

where $q_{s,i}^{nom}$ is typically at the origin.

Allowing θ_i to be the angle between $\eta_{e,i}$ and $\bar{\eta}_{i-1,i}$, the law of sines can be used to *find the distance* between $q_{e,i}$ and q_i as

$$m_i = \frac{\sin(\theta_i)}{\sin(\delta_i)} ||\eta_{e,i}||, \tag{4.24}$$

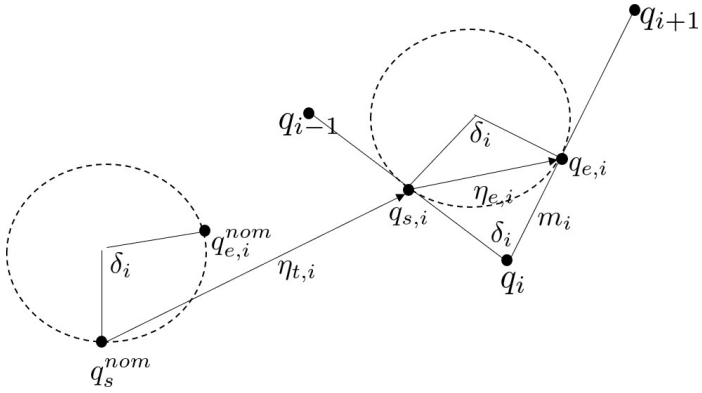

Fig. 4.4: An illustration of several of the parameters used to rotate and translation the CCTurn into position for maneuvering past the i^{th} waypoint.

where $q_{e,i} = q_i + m_i\bar{\eta}_{i,i+1}$ Finally, the *translation vector* for the CCTurn can be calculated as

$$\eta_{t,i} = q_{e,i} - \eta_{e,i}. \tag{4.25}$$

4.3 The Voronoi Constraint

In further defining the virtual structure, an operational envelope for each agent is defined to allow each agent to maneuver without breaking formation. The operational envelope in this research is derived as a static Voronoi tessellation which is defined with the set of desired follower positions $Q = [q_{f_{d_l},1}, q_{f_{d_l},2}, \ldots, q_{f_{d_l},n}]$, where n is the total number of agents in the formation. Previously (2.21) and (2.22) defined the criteria of a Voronoi cell and the boundaries of a cell. The entire Voronoi tessellation for a formation will be referred to as $\mathbb{V}(Q)$. The boundary of each cell consists of a line definition where $l_{k_i} = q_{k_i,1}, q_{k_i,2})$, where each $(q_{k_i,1}, q_{k_i,2})$ pair denote the two position vertices needed to define the line segment and k_i is the total number of line segments for the i^{th} cell.

4.3.1 Voronoi Dynamics

While this Voronoi tessellation is statically defined based on the virtual structure, it is necessary to redefine the Voronoi tessellation with respect to any change to the virtual leader. A single vertex of the cell boundaries can be updated similarly to how the follower

position is updated:

$$q_{k_i,1}(t) = R(\psi_l(t))q_{k_i,1}(t_0) + q_l(t), \tag{4.26}$$

To more simply state this in terms of the entire Voronoi tessellation it is necessary to redefine $\mathbb{V}_{lines,i}$ in terms of the entire tessellation. This is written as

$$\mathbb{V}_{lines} = \{\mathbb{V}_{lines,1} \;\; \mathbb{V}_{lines,2} \;\; \dots \;\; \mathbb{V}_{lines,n}\}, \tag{4.27}$$

where n represents the total number of cells. $\mathbb{V}_{lines,i}$ can be defined in terms of a set of points that can be translated and rotated based upon the virtual leader configuration. The set of pre-transformed points of cell i can be rewritten as

$$\mathbb{V}^0_{lines,i} = \{q^0_{1,1} \;\; q^0_{1,2} \;\; q^0_{2,1} \;\; q^0_{2,2} \;\; \dots \;\; q^0_{k_i,1} \;\; q^0_{k_i,2}\}, \tag{4.28}$$

where $(q^0_{k_i,1}, q^0_{k_i,2})$ represent the two points that define the k^{th} boundary line for the i^{th} Voronoi cell. This allows (4.26) to be used to describe the Voronoi tessellation at an given time. At time t, $\mathbb{V}^0_{lines,i}$ is transformed as

$$\mathbb{V}_{lines,i}(t) = \{q_{i,j} = R(\psi_l(t))q^0_{i,j} + q_l(t) | q^0_{i,j} \in \mathbb{V}^0_{lines,i}\}. \tag{4.29}$$

This formulation is important as the Voronoi tessellation can be expensive to compute. Equation 4.29 allows the heavy computation to be preformed as a preproccessing step.

4.3.2 Voronoi Boundary Avoidance

To incorporate the operational envelopes into the virtual structure a cost is added to the costs defined in Section 3.1.3. The purpose of this cost is to restrain the vehicles to within their respective Voronoi cell. This cost is included as an instantaneous cost to restrain the vehicles for all time. This cost is designed as a logarithmic barrier to create a

soft constraint to avoid vehicles leaving their designated Voronoi cell. This is written as

$$L_{k,voro}(x) = \begin{cases} \rho_5(ln(d^v_{max} - d^v_{min}) - ln(d_{k,voro} - d^v_{min})) & d^v_{min} < d_{k,voro} \leq d^v_{max} \\ \infty & d_{k,voro} \leq d^v_{min} \\ 0 & d^v_{max} < d_{k,voro} \end{cases}, \quad (4.30)$$

where $d_{k,voro}$ is the distance from the the vehicle to the lines that define the Voronoi boundary. This cost operates similarly to the obstacle barrier, with the main difference being how distance to the barrier is calculated. This distance is calculated via a unit vector, η_k, which is perpendicular to the k^{th} boundary line and the current agent position localized to the virtual structure, q_f. This distance is written as

$$d_{k,voro} = \eta_k^T q_f. \quad (4.31)$$

The logarithmic barrier has an infinite cost when the vehicle is outside and is finite when within the designated cell. This barrier is also convex when within the cell. Like the avoidance cost the Voronoi boundary cost is summed over all of the sides of the boundary.

4.3.3 Formation Convergence Analysis

Previously in Chapter 3 the convergence of the single vehicle was analyzed. In this Chapter this single vehicle model has been augmented to facilitate formation coordination. To show convergence to each vehicle's respective trajectory the following assumption is made, which is validated by examining the cost defined in (4.30):

A13: (Operational Envelope) The instantaneous cost forms a cost barrier around the operational area of each vehicle, such that $L(x, \kappa_i) \to \infty$ *as* $dist(q_i, V_i^C) \to 0$, *where* V_i^C *is the complementary space of* V_i, *which represents the space within a Voronoi cell and is finite in* V_i.

Theorem 3. *Given Theorem 2 and A13, the resulting trajectory produced by Algorithm 1 will either result in an infinite cost or an obstacle-free trajectory that can guide the vehicle*

to remain in the designated Voronoi cell.

Proof. Theorem 2 showed that Algorithm 1 would stay near a given trajectory. With the addition of A13, each vehicle within the formation will remain within their designated operational envelope while staying near the given trajectory. If a infinite cost is a result of A13, this would imply that the executable space of the operational envelope is no longer a feasible area of operation. This would cause a trigger to alert a higher-level planner to re-evaluate the desired trajectory. □

4.4 Example

This section demonstrates the formation tracking capability of the tracking algorithm developed in Chapter 3 with the added developments discussed in this Chapter. This simulation is developed in Matlab with the optimization being done via gradient descent with an Armijo line search. The weights have been adjusted considering the new developments.

$$\rho_1 = 0.3, \quad \rho_2 = 0.3, \quad \rho_3 = 0.6, \quad \rho_4 = 0.6, \quad \rho_6 = 0.3,$$
$$d_{min} = 0.25, \quad d_{max} = 0.75, \quad d^v_{min} = 0.25, \quad d^v_{max} = 0.75.$$

In this example the trajectory tracking/following is possible without re-planning by a higher-level planner, despite significant portions of the trajectories being inside an obstacle because of this values for the weight on the terminal barrier were not able to be tested.

This example is a representation of a corridor that is too narrow for the desired formation, as shown in Figure 4.5. There is also a obstacle placed in the direct path of the center agent to show that the vehicles have the full maneuverability previously displayed in Chapter 3. The state error of each agent can be seen in Figure 4.6. The vehicles on the sides of the virtual leader can be seen as having larger position errors while the desired position and trajectory is passing through walls. A spike in position error also occurs for the front and left agent due to the obstacle placed in the formation's path.

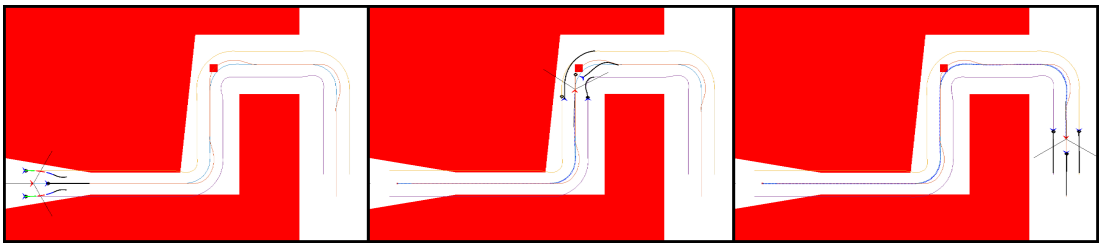

Fig. 4.5: This figure displays an example of the multi-vehicle formulation. The blue arrowheads represent the vehicles of the formation, while the red arrowhead represents the virtual leader. The circular dots represent the desired position of each vehicle and the thin black lines represent the Voronoi tessellation for this formation. The left most image shows the outer vehicles moving inwards to avoid the walls. The middle image displays obstacle avoidance of the forward vehicle. The right image shows an overview of the whole simulation. No collisions occurred and all three vehicles remained in their designated cell.

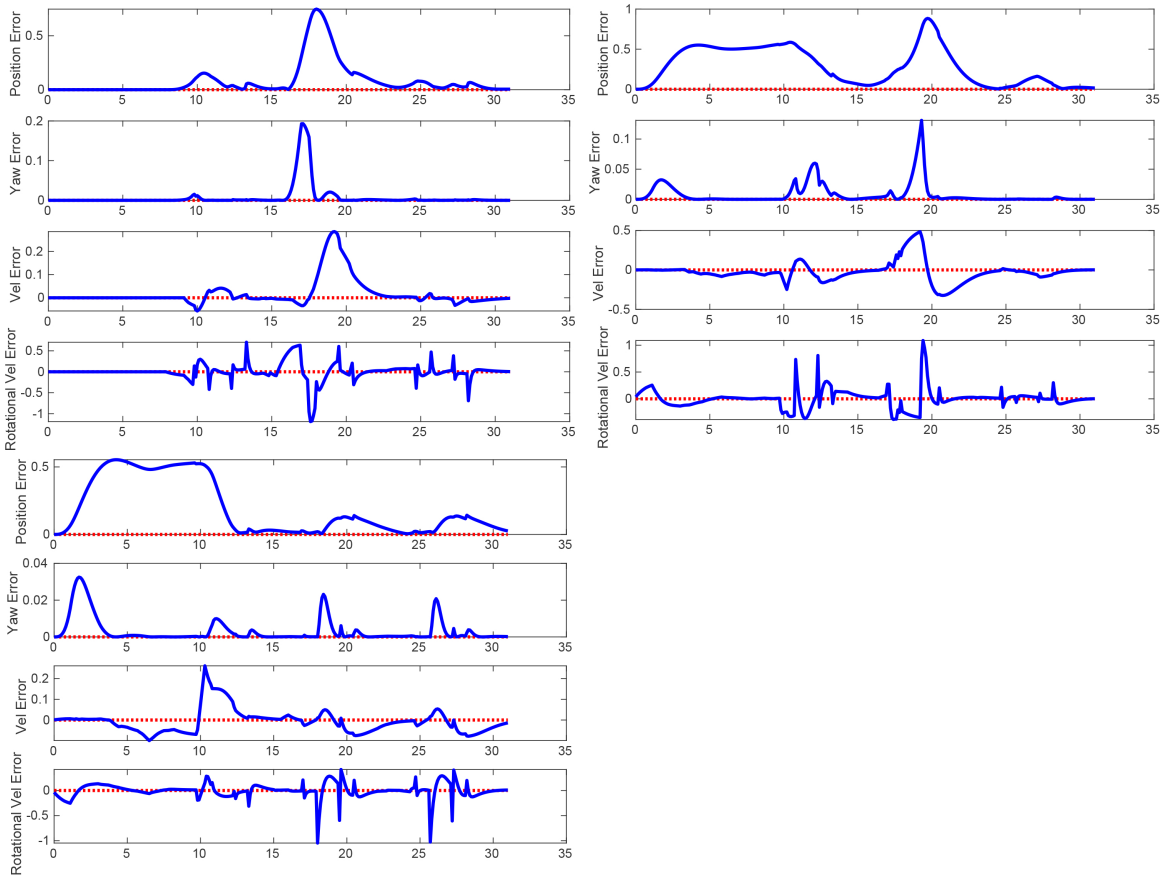

Fig. 4.6: This figure shows the error of each agent for the multi vehicle example. The error for the agent in front of the virtual leader is the top left. The agent to the left of the virtual leader is on the top right. The agent to the right of the virtual leader is on the bottom left.

CHAPTER 5

Examples

To further demonstrate the capabilities of the developments from Chapters 3 and 4, a simulation has been developed in C++ to show that the developed system can be simulated and implemented on real-time systems. This demonstrates that the formation control method of this work is viable for implementation on an actual system.

To develop for real-time systems C++ was chosen for its breadth of use and speed at execution. This simulation utilizes the Robotic Operating System (ROS) as a basis of the architecture, such that the effort to implement on hardware is minimal [39]. As the name implies, ROS is an architecture created for the purpose of developing and operating robotic systems that allows for parallel processing with a robust message system for communicating between processes. Within ROS, a basic independent program is referred to as a node. ROS also contains several visualization tools that are useful for simulation; this work relies on ROS's rviz. An example this can be seen in Figure 5.1, which depicts a turtlebot, the vehicle model used in this simulation. Another visualization tool, rqt_graph, displays the interactions of the node and message structure. This plot for a single vehicle in the simulation is given in Figure 5.2 and Figure 5.3 shows the interactions between the virtual leader and a single formation vehicle. The "/robot2/param_opt" node waits for a message from the virtual leader node to start executing the developed MPC algorithm. While waiting

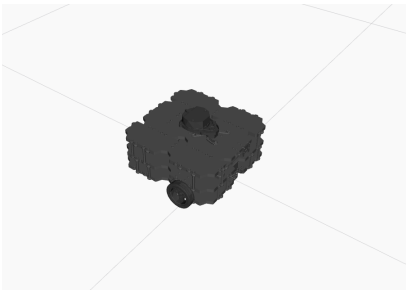

Fig. 5.1: A close-up image of the simulated robot as it appears in Rviz.

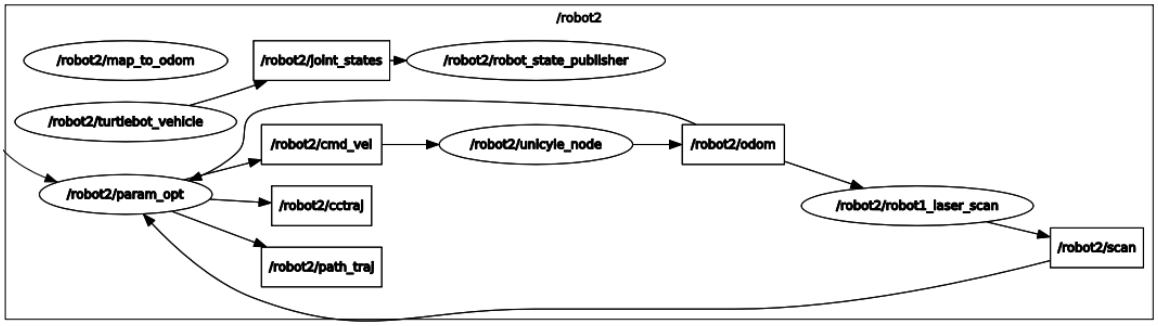

Fig. 5.2: This figure shows architecture of a given vehicle. Each ellipse represents a node within ROS and each rectangle is a message between nodes.

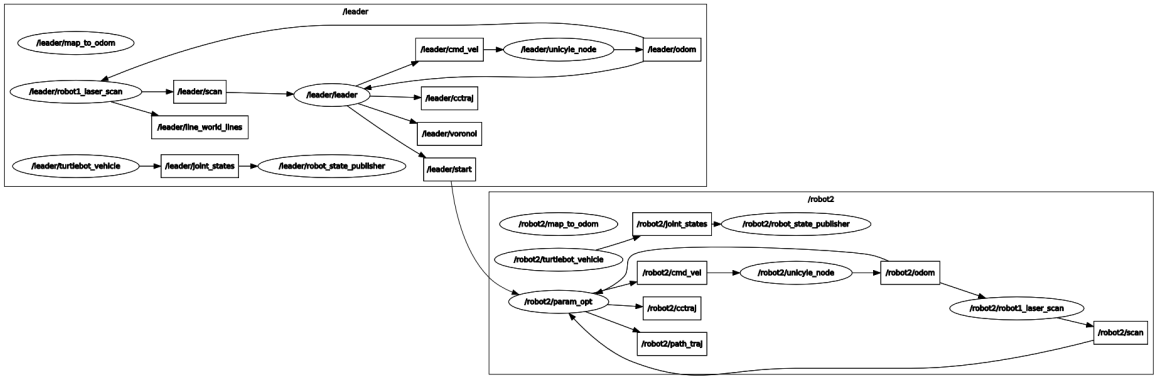

Fig. 5.3: This figure shows architecture of virtual leader and a single formation vehicle. Each ellipse represents a node within ROS and each rectangle is a message between nodes.

for the message from the virtual leader this node computes the reference trajectories that it will need to execute. This node publishes a command message for the vehicle to actuate. This actuation is completed by "/robot2/unicycle_node" which simulates the dynamics of a vehicle and continually publishes odometry data. The simulated lidar data and world details is produced in "/robot2/robot1_laser_scan". Both the odometry and lidar data are fed back into the optimization node to create a new command.

The simulation also utilized various third party libraries. One of the libraries used in this work is NLopt, which stands for nonlinear optimization. NLopt is an open source optimization library that was chosen for the optimization process in the MPC framework [40]. Within NLopt there are a host of optimization algorithms to choose from, for this work the sequential least-squares quadratic programming (SLSQP) algorithm was chosen. The Eigen C++ library was used for matrix types and arithmetic [41]. The Boost C++ numeric library was used for numerical integration [42]. This allowed the capability of testing different modes of numerical integration. The Boost libraries were also used to generate the Voronoi tessellation for the operational envelope, which can be found within the Boost geometry library [43].

5.1 Simulation Parameters

Chapter 3 focused on the development of the individual vehicles that would be used as the fundamental building block of later formation developments. The desired capabilities of the single vehicle are to avoid obstacles and to track a reference trajectory. To achieve these attributes, costs for an MPC framework were developed: obstacle avoidance, velocity matching, and a terminal convergence to the trajectory cost.

The C++ implementation of the single vehicle developments has the same parameter definitions with the given values:

$$\rho_1 = 0.15, \quad \rho_2 = 0.15, \quad \rho_3 = 0.60, \quad \rho_4 = 0.65,$$
$$d_{min} = 0.25, \quad d_{max} = 1.00$$

where ρ_1 and ρ_2 are the weights on the velocity matching, ρ_3 is the weight on avoidance of obstacles, ρ_4 is the terminal state weight, and d_{min} and d_{max} are the minimum and maximum distance for the logarithmic barrier on the obstacle avoidance cost. The timing parameters are that are inherent of MPC are set as follows:

$$\delta = 0.15, \quad dt = 0.10, \quad t_f = 3.00,$$

where δ represents the period of time a command is executed, dt represents the discrete step size that is considered for the numerical integration, and t_f is the span of the receding horizon. Note, δ is an adjustable parameter based on the rate at which the optimization can produce a result. The average optimization time in this simulation was roughly 0.17 seconds. However, NLopt provides an option to specify a maximum execution time, which was set to 0.10 as the optimization was typically very close to a local minimum. This allowed an execution loop time of $\delta = 0.15$.

The parameters required to generate the reference trajectory are:

$$\epsilon = 0.20, \quad v_{traj} = 0.50,$$
$$\kappa_{max} = 0.625, \quad \sigma_{max} = 0.625, \quad \gamma_{max} = 50,$$

where ϵ is the distance at which a ϵ-trajectory is created, v_{traj} is the constant velocity of the virtual leader, and κ_{max}, σ_{max}, γ_{max} represent the maximum curvature, curvature rate, and curvature acceleration allowable. Note, each of these parameters depend upon the vehicle's capabilities. The values specified imply that the vehicle can complete a $1.6m$ turn at a speed of $0.50m/s$.

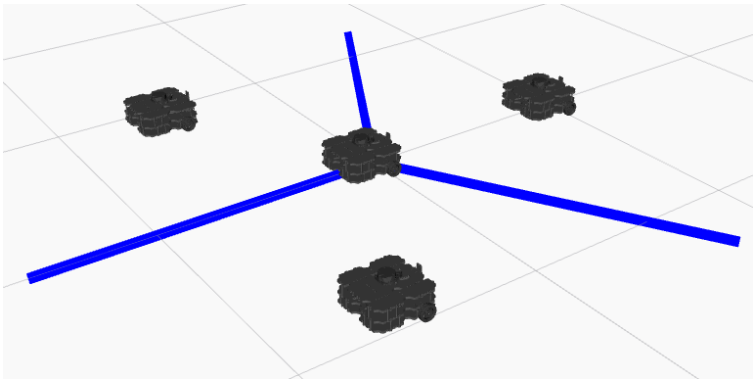

Fig. 5.4: This figure shows the formation in the ROS simulation. The center robot depicts the virtual leader. The blue lines depict the Voronoi tessellation boundaries. While each wall for this particular formation should extend infinitely, the walls were displayed with a finite length for simplicity.

In Chapter 4 a few more parameters were introduced for the development of the Multi-Vehicle formation capabilities. These consist primarily of the necessary parameters of the formation configuration and the Voronoi cost. The formation configuration that was used was a triangle formation which consists of three vehicles with one in front and two behind the virtual leader to the left and right. This is shown by the given formation positions:

$$q_l = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad q_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad q_2 = \begin{bmatrix} -0.5 \\ 0.866 \end{bmatrix}, \quad q_3 = \begin{bmatrix} -0.5 \\ -0.866 \end{bmatrix}.$$

A visualization of this formation configuration is depicted in Figure 5.4.

This formation configuration is used to produce trajectories for each vehicle to follow. The offset from the leader is utilized to generate each vehicles' own reference trajectory. Another use of this formation configuration is to generate a Voronoi tessellation, where each cell of the tessellation is used as an operational envelope for each respective vehicle. These operational envelopes are applied to the single vehicle developments as soft constraint in the form of a logarithmic barrier. The parameters for this barrier are:

$$\rho_5 = 0.60, \quad d^v_{min} = 0.25, \quad d^v_{max} = 0.75.$$

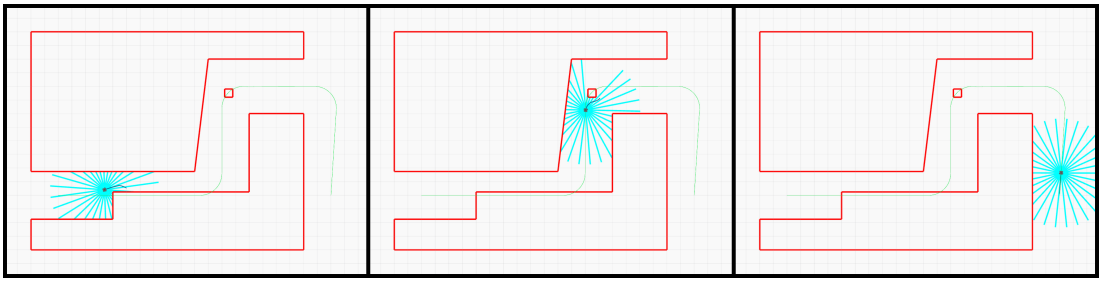

Fig. 5.5: This figure displays the single vehicle simulation in C++. The left and middle image displays the obstacle avoidance instances. The right image shows an overview of the whole simulation. The green line represents the desired trajectory. The red lines depict the boundary of obstacles. The teal lines represent the on-board lidar of the vehicle.

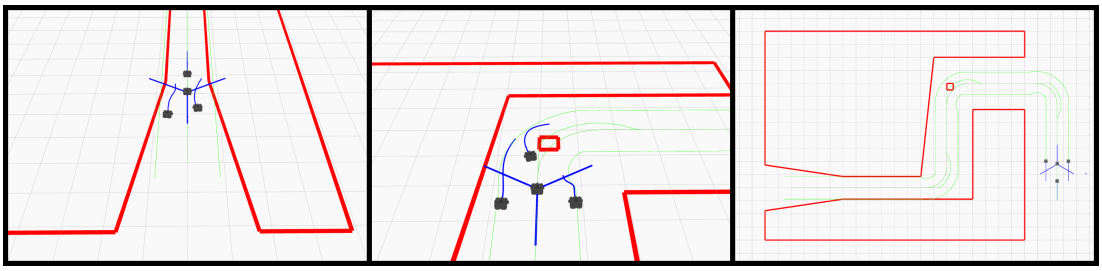

Fig. 5.6: This figure displays the multi-vehicle simulation in C++. The left and middle images show the formation successfully navigating around the obstacles in the environment. The right image shows the vehicles finishing the simulation fully converged on tracking their respective reference trajectories.

5.2 Results

The C++ simulation of the single vehicle developments can be seen in Figure 5.5. This example was designed to show that a typical environment can be navigated with a poorly placed reference trajectory. This simulation operated at real-time and succeeded in avoiding collision with any obstacles and tracked the desired trajectory when sufficiently far from any obstacles.

The results from the multi-vehicle simulation can be seen in Figure 5.6. This example was executed with the same parameters as the single vehicle simulation with the addition of the necessary development. This simulation was run at real-time and all vehicles succeeded in avoiding obstacles while staying in their designated Voronoi cells.

The complexity of operating at real-time introduces the problem of the time required to calculate a command. In the Matlab examples in Chapters 3 and 4, the command is

calculated at each discrete time step, this however proved to be infeasible for the real-time case. The speed at which the C++ simulation could, on average, produce a command averaged roughly every 0.17 seconds. This delay required the prediction of the vehicle state at the time that a command is issued. A limit was placed on the optimization process such that the vehicle was given commands on a regular integral to further prevent collision due to any uncertainties, whether environmental or dynamic. Previously it was stated that the execution loop time δ was set to 0.15 seconds and that the optimization was limited to 0.10 seconds. This was done to assist in overcoming these uncertainties. A sub-optimal command is produced, but produced more rapidly then a optimal command, thus resulting in a more stable system.

CHAPTER 6

Conclusion

The purpose of this work was to develop a moving formation control framework that allows each individual vehicle to adapt its control based upon dynamic and environmental limitations while safely following a coordinated trajectory within the formation. This was accomplished by combining four major elements. The first is the application of MPC as the method of determine the control of the formation vehicles. The second was the use of virtual structures to coordinate the vehicles by deriving operational envelopes and relating the virtual leaders motion to each vehicle. The third major element was the development of motion for the virtual leader using continuous-curvature paths. The fourth element was the use of zero-error trajectory tracking control laws which allowed vehicles to converge to and track a sufficiently smooth trajectory.

A dual-mode behavior-based MPC trajectory tracking formulation for a single vehicle was developed in Chapter 3. This MPC formulation created an optimization framework that provides the first order necessary conditions for numerical solutions and convergence theorems to ensure vehicles move as desired. This single vehicle development was finished with a showcasing of the capabilities of the developed MPC framework.

In Chapter 4, multiple-vehicle attributes were developed and applied to the MPC framework. These attributes included the generation of the virtual leader trajectory, producing follower trajectories from a leader trajectory, and the creation of the operational envelopes. An example was provided to demonstrate these multiple vehicle developments. This example succeeded in tracking the reference trajectory in an obstacle laden environment, while at the same time maintaining vehicles within their designated operational envelopes.

In Chapter 5, a simulation was developed in C++ to operate the developed moving formation control for real-time systems. The simulation was successful in producing similar results to the previous Matlab examples. This work successfully developed a moving forma-

tion control framework that allows each individual vehicle to adapt its control based upon dynamic and environmental limitations while safely following a coordinated trajectory as a collective whole within a formation.

REFERENCES

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. [Online]. Available: http://link.springer.com/10.1007/978-3-540-77974-2

[2] X. Zhou, W. Wang, T. Wang, X. Li, and T. Jing, "Continuous patrolling in uncertain environment with the UAV swarm," *PLoS ONE*, vol. 13, no. 8, pp. 1–29, 2018.

[3] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Magazine*, vol. 29, no. 1, pp. 9–19, 2008.

[4] T. Suzuki, T. Sekine, T. Fujii, H. Asama, and I. Endo, "Cooperative formation among multiple mobile robot teleoperation in inspection task," *Proceedings of the IEEE Conference on Decision and Control*, vol. 1, pp. 358–363, 2000.

[5] A. Soni and H. Hu, "Formation control for a fleet of autonomous ground vehicles: A survey," *Robotics*, vol. 7, no. 4, 2018.

[6] S. Wu, Y. Xia, M. Lin, and Y. Luo, "Leader-following Consensus and Trajectory Tracking for Nonholonomic Mobile Robots," in *Proceedings 2018 Chinese Automation Congress, CAC 2018*. Institute of Electrical and Electronics Engineers Inc., jan 2019, pp. 3678–3683.

[7] S. Sarno, M. D'Errico, J. Guo, and E. Gill, "Path Planning and Guidance Algorithms for SAR Formation Reconfiguration: Comparison between Centralized and Decentralized Approaches," *Acta Astronautica*.

[8] H. A. Almurib, P. T. Nathan, and T. N. Kumar, "Control and path planning of quadrotor aerial vehicles for search and rescue," in *Proceedings of the SICE Annual Conference*. Society of Instrument and Control Engineers (SICE), 2011, pp. 700–705.

[9] R. W. Beard, J. Lawton, and F. Y. Hadaegh, "A coordination architecture for spacecraft formation control," *IEEE Transactions on Control Systems Technology*, vol. 9, no. 6, pp. 777–790, 2001.

[10] C. Kownacki and L. Ambroziak, "Local and asymmetrical potential field approach to leader tracking problem in rigid formations of fixed-wing UAVs," *Aerospace Science and Technology*, vol. 68, pp. 465–474, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.ast.2017.05.040

[11] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model predictive control*, 2nd ed. Nob Hill Publishing, 2019.

[12] P. Ogren, E. Fiorelli, and N. E. Leonard, "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," *IEEE Transactions on Automatic control*, vol. 49, no. 8, pp. 1292–1302, 2004.

[13] G. Droge, "Distributed virtual leader moving formation control using behavior-based MPC," in *Proceedings of the American Control Conference*, vol. 2015-July, 2015.

[14] T. Fraichard and A. Scheuer, "From Reeds and Shepp's to Continuous-Curvature Paths," *IEEE Transactions on Robotics*, vol. 20, no. 6, pp. 1025–1035, 2004.

[15] C. Ferrin, G. Droge, and R. Christensen, "Zero-Error Epsilon-Tracking for Autonomous Vehicles through Epsilon-Trajectory Generation," 2020.

[16] R. Olfati-Saber, "Near-identity diffeomorphisms and exponential ϵ-tracking and ϵ-stabilization of first-order nonholonomic SE(2) vehicles," in *Proceedings of the American Control Conference*, vol. 6, 2002, pp. 4690–4695.

[17] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006, vol. 9780521862.

[18] A. De Luca, G. Oriolo, C. Samson, and J.-P. Laumond, "Feedback Control of a Nonholonomic Car-like Robot Robot Motion Planning and Control Feedback Control of a Nonholonomic Car-Like Robot," Sapienza University of Rome, Tech. Rep., 1998.

[19] R. M. Murray and S. S. Sastry, "Nonholonomic motion planning: steering using sinusoids," *IEEE Transactions on Automatic Control*, vol. 38, no. 5, pp. 700–716, May 1993.

[20] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *Proceedings of the American Control Conference*, 2007.

[21] J. M. Snider, "Automatic Steering Methods for Autonomous Automobile Path Tracking," Ph.D. dissertation, Carnegie Mellon University, 2009.

[22] K. Soltesz, "Trajectory tracking control of an autonomous ground vehicle," Ph.D. dissertation, 01 2008.

[23] J. P. Hespanha, *Linear systems theory*. Princeton university press, 2018.

[24] D. Liberzon, *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2012.

[25] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[26] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[27] W. Chung, S. Kim, M. Choi, J. Choi, H. Kim, C.-b. Moon, and J.-B. Song, "Safe navigation of a mobile robot considering visibility of environment," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3941–3950, 2009.

[28] P. Ogren and N. Leonard, "A convergent dynamic window approach to obstacle avoidance," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 188–195, 2005.

[29] G. Droge, "Dual-mode Dynamic Window Approach to Robot Navigation with Convergence Guarantees," *Journal of Control and Decision*, 2020. [Online]. Available: http://arxiv.org/abs/1808.05869

[30] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[31] G. Droge, P. Kingston, and M. Egerstedt, "Behavior-based switch-time mpc for mobile robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012.

[32] G. Droge and M. Egerstedt, "Optimal decentralized gait transitions for snake robots," School of Electrical and Computer Engineering, Georgia Institute of Technology, http://gritslab.gatech.edu/home/wp-content/uploads/2012/01/Dist_Gaits.pdf, Tech. Rep., January 2012.

[33] T. Chevet, C. S. Maniu, C. Vlad, and Y. Zhang, "Voronoi-based UAVs Formation Deployment and Reconfiguration using MPC Techniques," *2018 International Conference on Unmanned Aircraft Systems, ICUAS 2018*, pp. 9–14, 2018.

[34] ——, "Guaranteed voronoi-based deployment for multi-agent systems under uncertain measurements," *2019 18th European Control Conference, ECC 2019*, pp. 4016–4021, 2019.

[35] T. Chevet, C. Vlad, C. S. Maniu, and Y. Zhang, "Decentralized MPC for UAVs Formation Deployment and Reconfiguration with Multiple Outgoing Agents," *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2019.

[36] L. E. Dubins, "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, vol. 79, no. 3, p. 497, 1957.

[37] E. Polak, *Optimization: algorithms and consistent approximations*. Springer New York, 1997, vol. 7.

[38] Z. S. S. Murray, Richard M Li, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[39] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009.

[40] "NLopt Algorithms." [Online]. Available: https://nlopt.readthedocs.io/en/latest/NLopt{_}Algorithms/

[41] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[42] "The Boost.Numeric.Odeint Library." [Online]. Available: https://www.boost.org/doc/libs/1_66_0/libs/numeric/odeint/doc/html/index.html

[43] "The Boost.Polygon Voronoi Library." [Online]. Available: https://www.boost.org/doc/libs/1{_}52{_}0/libs/polygon/doc/voronoi{_}main.htm