BEHAVIOR-BASED MODEL PREDICTIVE CONTROL FOR NETWORKED MULTI-AGENT SYSTEMS

A Thesis
Presented to
The Academic Faculty

by

Greg N. Droge

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2014

BEHAVIOR-BASED MODEL PREDICTIVE CONTROL FOR NETWORKED MULTI-AGENT SYSTEMS

To my wife and children,

Without whom I would not be

who I am today.

ACKNOWLEDGEMENTS

I would first like to acknowledge my adviser, Dr. Magnus Egerstedt, for his continuous support and example over the past several years. I will forever be grateful for his help and encouragement in my research endeavors and for the love of learning that he has instilled in me. Not only has he been an incredible help in guiding me through my research, but has been a tremendous example in the way he treats others and maintains his family life.

I am grateful for the members of the GRITS lab for their helpful insights and discussions which led me through my research. They are truly a set of great minds, some of the brightest people I have known. I am grateful for their friendship and help for the past several years. They have been an irreplaceable group of people, allowing me to bounce ideas off of them as well as get inspiration for possible avenues of work.

Most of all, I would like to thank my incredibly wife for her love and support and for my children for being helping me to stay focused on the more important things in life each day when I go home. They are amazing. I truly would not be the person I am today without my family, the most important part of my life.

TABLE OF CONTENTS

DEDICATION . . . . **iii**

ACKNOWLEDGEMENTS . . . . **iv**

LIST OF TABLES . . . . **ix**

LIST OF FIGURES . . . . **x**

SUMMARY . . . . **xv**

I BACKGROUND **1**

1.1 Distributed Optimization 1

1.1.1 Distributed Optimization Formulation 2

1.1.2 Existing Methods for Distributed Optimization 3

1.2 Model Predictive Control 5

1.2.1 Dual-mode MPC 6

1.2.2 Computation of Optimal Control Trajectories 7

1.3 Behavior-based Model Predictive Control 8

1.3.1 Behavior-based MPC Background 9

1.3.2 Motion Control and Behavior-based MPC 10

1.3.3 The Dynamic Window Approach to Navigation 11

1.4 Distributed Multi-Agent MPC 13

1.5 Organization 15

II PROPORTIONAL-INTEGRAL DISTRIBUTED OPTIMIZATION FOR NETWORKED SYSTEMS WITH STATIC TOPOLOGIES **16**

2.1 Introducing the Constrained Optimization Problem 17

2.1.1 Problem Formulation 18

2.1.2 Networked Multi-Agent Systems 19

2.1.3 PI Control as Gradient Method for Constrained Optimization . . . 21

2.1.4 PI Performance Metrics 24

2.2 Dual Decomposition 25

2.2.1 Dual-Decomposition for Networked Systems 25
2.2.2 Integral Control 29
2.2.3 Distributed Implementation 30
2.2.4 Example 31
2.3 Consensus Based Distributed Optimization 32
2.3.1 Consensus Based Algorithm 32
2.3.2 Consensus Method and Constrained Optimization 34
2.3.3 Example 35
2.4 PI Distributed Optimization 36
2.4.1 PI Distributed Optimization Algorithm 36
2.4.2 Connections to PI Control 38
2.4.3 Example 39
2.4.4 Scalable Multi-Agent Formulation 40
2.4.5 PI Distributed Optimization of a Non-Convex Function 44
2.5 Conclusion 48

III PROPORTIONAL-INTEGRAL DISTRIBUTED OPTIMIZATION FOR NETWORKED SYSTEMS WITH SWITCHING TOPOLOGIES 50

3.1 Network Multi-agent Systems with Switching Topologies 51
3.2 Alternative Proof of Convergence for Static Topologies 52
3.3 Extending Convergence for Dynamic Topologies 54
3.4 Index-free PI Distributed Optimization 55
3.5 Example: Formation Control 58
3.6 Conclusion 61

IV BEHAVIOR-BASED MPC 63

4.1 Behavior-based MPC Formulation 64
4.1.1 MPC Framework 65
4.1.2 First Order Optimality Conditions 66
4.2 Example: Vector-field Orbiting for Nonholonomic Vehicle 69
4.2.1 Non-Linear Unicycle Control 69

4.2.2 Partials for Cost Optimization 72
4.2.3 Orbit Example 73
4.3 Example: Vector-field Navigation for Inverted Pendulum Robot 75
4.3.1 Inverted Pendulum Robot 77
4.3.2 Straight Path Following 80
4.3.3 Navigation Examples 82
4.4 Conclusion 86
V DUAL-MODE DYNAMIC WINDOW APPROACH TO NAVIGATION . . . 87
5.1 Dual-mode Arc-based MPC 88
5.1.1 Dual-mode Arc-baseed MPC Algorithm 88
5.1.2 Convergence 91
5.1.3 Behavior-based Optimization 96
5.2 Control and Costs for Unicycle Motion Model 100
5.2.1 Reference Following Control 100
5.2.2 Cost Definition 102
5.3 Case Study: Magellan Robot 104
5.3.1 First Order Model 104
5.3.2 Reference Following Control 105
5.3.3 Results 107
5.4 Case Study: Inverted Pendulum Robot 109
5.4.1 Control Laws 109
5.4.2 Maintaining Balance 111
5.4.3 Results 112
5.5 Conclusion 113
VI MULTI-AGENT DISTRIBUTED BEHAVIOR-BASED MPC 114
6.1 Preliminaries 115
6.1.1 Problem Definition 115
6.1.2 Communication of Trajectories 116

6.1.3 Notation 117
6.2 Multi-agent Distributed Parameterized MPC 118
6.2.1 Gradients 118
6.2.2 MPC Framework 120
6.3 Induced Information Structure 120
6.4 Example 123
6.5 Conclusion 125

VII DISTRIBUTED VIRTUAL LEADER FORMATION CONTROL 127

7.1 Virtual Leader Formation Control Background 128
7.2 Arc-based Leader-follower Control 130
7.2.1 Epsilon-tracking Formation Control 130
7.2.2 Perfect Tracking Using Approximate Control 135
7.2.3 Obstacle Avoidance Control 137
7.3 Virtual Leader Behavior-based MPC 138
7.3.1 Virtual Leader Algorithm 138
7.3.2 A Note on Information Topology 140
7.3.3 Cost Definition 141
7.4 Results 142
7.4.1 Measuring Formation Performance 142
7.4.2 Virtual Leader Formation Results 144
7.5 Conclusion 146

REFERENCES 148

LIST OF TABLES

1 The results of performing proportional, integral, and PI distributed optimization for the convex optimization problem 32

2 The results of performing proportional, integral, and PI distributed optimization with each agent optimizing over the full state vector. 44

3 The results of performing proportional, integral, and PI distributed optimization with each agent optimizing over a subset of the state vector. . . . 45

4 The results of performing proportional, integral, and PI distributed optimization with $\theta_2 = \frac{\pi}{4}$ 48

5 The results of performing proportional, integral, and PI distributed optimization with $\theta_2 = \frac{3\pi}{4}$ 48

6 This table shows the resulting parameters as well as the average and standard deviation of distances for each simulation. The nominal distance, d_N, refers to the distance from agent's starting position to their nominal position. The travel distance, d_T, is how far the agents actually traveled to reach formation. 60

7 This table defines the symbols used in the dynamics of the two-wheel inverted pendulum robot. The numeric values are given in [42]. 79

LIST OF FIGURES

1 This figure shows the results of using the cost $f(z) = (x_1 - 1)^2 + (x_2 + 1)^2$. Left: Dotted line shows the equality constraint and the arrows show the gradient and projected gradient. Right: Result of performing the PI gradient method for optimization given in (13). The trajectory of the two states is shown ending in the final condition denoted by the solid circle and the constraint is shown as a dotted line. The arrows show the final gradient and Lagrange multiplier multiplied by the constraint. As expected, these are equal in magnitude, but opposite in direction. 22

2 This figure depicts the "Line" network structure used for the examples in Sections 2.2, 2.3, and 2.4 31

3 This figure shows the results from the convex optimization example using dual-decomposition 32

4 This figure shows the result of optimizing using consensus for the problem given in (27) for both a constant and fading value for k_G on the left and right respectively 35

5 This figure shows the results from the convex optimization example using PI distributed optimization 40

6 This figure depicts the "Ring" network structure used in Section 2.4.4 . . . 43

7 This figures shows the results of applying the formulation of Sections 2.2, 2.3, and 2.4 on the top row and 2.4.4 bottom row to solve the problem in (42). The left, middle, and right images of each row correspond to consensus, dual-decomposition, and PI distributed optimization techniques. The results shown are for variable 10. The solutions in the top row require 20 versions of this variable to converge to the optimal value where the solutions in the bottom row require only 3. 44

8 From left to right in each row, this figure shows the cost, consensus, dual-decomposition, and PI distributed optimization methods applied to the problem in (43). The top row corresponds to results when $\theta = \frac{\pi}{4}$ and the bottom row corresponds to $\theta = \frac{3\pi}{4}$. The axis label z corresponds to each agents version of the variable, k_G is the gain on the gradient for the consensus method, and μ is the Lagrange multiplier for the other methods. 46

9 The task of moving into formation can be defined as finding a translation (τ), rotation (θ), and scaling (γ) from the nominal formation, shown in the upper left, to the desired position of the agents,shown in the bottom right. . 50

10 This figure shows the optimal point to the cost $f(z) = (x_{11} - 1)^2 + (x_{21} + 1)^2$ with constraint $x_{11} = x_{21}$. The unconstrained gradient, $\frac{\partial f}{\partial z}^T(z^*)$, is balanced by the vector $\frac{\partial h}{\partial z}^T(z^*)\lambda^*$ 56

11 On the top left is shown snapshots of the agents while converging to the diamond formation. On the bottom left is shown the resulting diamond and line formations. The lines between agents show the communication topology and the bottom left of each figure is shown the nominal configuration. On the right is shown each agents' version of γ while converging to the diamond formation. Other variables are not shown as this plot is indicative of the convergence characteristics of the variables in each simulation. . . . 60

12 This figure shows 60 agents assigned to spell out the word 'GRITS'. On the left is shown the initial positions and on the right is the final position. . . 61

13 The images show the result of different gains on an orbiting vector field. The left image shows the result of gains that move the robot directly towards the orbit and the middle shows the result of gains that very smoothly transition into orbit. The right image shows the resulting distance and orientation of starting at the same point and executing the direct and smooth vector fields as well as the result of adapting the vector field using behavior-based MPC. 64

14 This figures shows a diagram of the states of a unicycle robot. (x_1, x_2) gives the position and ψ gives the orientation. 70

15 This figure shows different snapshots in time of the adaptation of the vector-field given in equation (86). A series of two behavior was executed at each time instant, each being an implementation of the control law in (84) with different parameters for the vector field. The robot is shown with its planned trajectory extending from it in each case. The middle two images are actually the same time instance where the middle-left image shows the vector field produced in the first time window and the middle-right image shows the vector field produced in the second time window. 75

16 This figure shows the costs associated with different numbers of switches with each behavior executing the control law in (84) with different parameters. The costs are normalized so that the largest cost is scaled to one. . . . 75

17 On the left is shown the line and orbit vector fields. On the right is shown an example of the vector fields being concatenated together to guide the robot on a path through the environment. 76

18 Shown is a diagram of the inverted pendulum robot with the symbols defined in Table 7. 78

19 This figure shows the definition of a line through a point p and with a given angle ψ. It also shows the coordinate frame of the line used to create a line following vector field. 81

20 The trajectory and underlying orbits for the vector field navigation examples are shown above. 84

21 Above are shown three states during the vector field navigation of the environments shown in Figure 20. 84

22 On the left is shown the average computation time per gradient step while executing the figure-eight shown in Figure 21. On the right is shown the normalized cost per gradient step of a figure-eight. The cost is normalized so that no optimization has a cost of 1. 85

23 This figure shows an example of a dual-mode arc-based trajectory. The robot is shown as a triangle with a planned trajectory extending from it. The trajectory is created from three arc-based controllers appended back to back with a reference tracking controller at the end. The different portions of the trajectory are differentiated by color and line styles. A reference trajectory is also shown as a dashed line. 89

24 On the left is shown a contour plot of a cost when choosing a single (v, ω) pair. Shown on the contour plot are 20 points tested along a circle of radius 1 with a line connecting each point to a collision free pair of velocities executing the same arc, with a different speed. On the top-right is shown the initial arcs before scaling and on the bottom-right is shown the arcs after scaling with $\alpha = .8$. The straight lines denote walls. 98

25 This figure demonstrates possible behaviors based on arc-based control. From left to right is shown a single arc, arc then reference follower, turning, and point-to-point behaviors. In each image the robot is represented as a triangle, with the resulting trajectory extending from it. The arcs are differentiated through color and line style, with the final reference tracking mode denoted as a solid line at the end of the trajectory (only visible in the arc-then-reference behavior). The straight lines denote walls. 99

26 Shown is a diagram of a two-wheeled robot with the ϵ-point to be controlled. 101

27 Shown is a picture of the IRobot Magellan-pro. 104

28 The left two images show the two environments completely mapped with the Magellan in the center. The right two images show the Magellan executing Algorithm 2. As the Magellan approaches the goal, the map is erased so that the Magellan can traverse the next environment as if it were unknown. 108

29 This figure shows results from the reference tracking control, optimization without gradient descent, and various times allowed for gradient-based optimization. From left to right is shown the total cost (normalized so that the reference tracking cost equals one), average execution time for a loop of Algorithm 2, and the average velocity for a window time excluding acceleration and deceleration periods. On both the middle and right plots is shown the standard deviation for each trial. 108

30 On the left is shown an executed path through the environment. The right three images show the robot and current map at different positions along the trajectory. The robot is shown as a triangle. The line extending from it is the trajectory created by the arc-based MPC framework. 112

31 This figure shows results for the inverted pendulum robot using the reference tracking control, optimization without gradient descent, and various times allowed for gradient-based optimization. From left to right is shown the total cost (normalized so that the reference tracking cost equals one), average execution time for a loop of Algorithm 2, and average and maximum tilt angle for each trial. 112

32 This figure shows how dynamic dependencies make the required communication increase as needed information propagates through the edges in dynamic dependency graph. Cost dependence ($\mathcal{G}^c$), dynamic dependence ($\mathcal{G}^d$), and the induced information graph ($\mathcal{G}^I$) are shown in red, blue, and green respectively 123

33 This illustrates the utility of the proposed MPC framework. From left to right is shown $\mathcal{G}^I$, the starting configuration, result of agents not negotiating, and the result of agents using dual-decomposition to negotiate. The result of consensus with constant step size is not shown as it is very similar to using dual-decomposition. The result of consensus with diminishing step-size is not shown as it does not converge. 124

34 This figures shows each agents opinion of what the velocity should be of the blue colored agent. From left to right are the results from dual-decomposition, consensus with constant step-size, PI, and no distributed optimization. 125

35 This figure shows the resulting cost versus time for each of the four simulations, namely, distributed optimization using dual-decomposition, the consensus based method with constant step-size, the consensus method with diminishing step size, and no distributed optimization. 125

36 This diagram shows the leader position, q_l, and orientation, ψ_l, the actual and desired follower positions, q_f and q_{f_d}, and orientations, ψ_f and ψ_{f_d} as well as the actual and desired follower ϵ-points, q_{f_ϵ} and $q_{f_{d_\epsilon}}$. The leader position and desired follower position are located on circles as the motion of each will be on circles with the same center, but different radii. 131

37 The left and right images show the desired orientation of the follower for $\omega < 0$ and $\omega > 0$ respectively. The arrow-heads on the circles denote the direction of motion around the circle. 133

38 This figure shows the set of all positions that could satisfy $q_{f_\epsilon} = q_{f_{d_\epsilon}}$. The set forms a circle of radius ϵ, with q_{f_d} lying on the circle and ψ_{f_d} pointing towards the center of the circle. 136

39 The left and middle images show the Lyapunov function, V, given in (145), as well as its derivative, $\dot{V}$. V is plotted as a solid line and $\dot{V}$ is plotted as a dotted line. The left image shows the results for $v_{f_d} = .5$, $\epsilon = .1$, and $\omega_{f_d} = 5$ and the middle shows results for $v_{f_d} = .5$, $\epsilon = .1$, and $\omega_{f_d} = 100$. The right image shows convergence times for the values in the middle image for all possible deviations in orientation, showing that V may be conservative. . 137

40 This figure shows the actual positions of the agents plotted as circles with solid lines, the desired positions using the global variables as circles with dotted lines, and the desired positions using the process in Section 7.4.1 as solid circles. 143

41 This figure shows the results from the equal gain trial (i.e. $\rho_1 = \rho_2 = \rho_3 = \rho_4 = 1$) for the line formation on the top row and GT formation on the bottom row. The left most image shows the initial environment and nominal positions of the agents. The right three images show zoomed in images of the agents at different points in the environment. The agents are shown as triangles, the obstacles are shown as solid circles, the waypoints are shown as dotted circles, and the trajectory of the leader agent around the environment is plotted as a solid line. 145

42 This figure shows the formation error for the line formation on the top row and the GT formation on the bottom row. On the left is shown the maximum and average distance for each agent for $\rho_1 = \rho_2 = \rho_3 = \rho_4 = 1$. On the right is shown a bar graph with the maximum and average distance for both trials. The left set of bars corresponds to the results from leader-only control (i.e. $\rho_1 = \rho_2 = \rho_3 = 0$, $\rho_4 = 1$). The right set of bars corresponds to the results from using $\rho_1 = \rho_2 = \rho_3 = \rho_4 = 1$. The value for the maximum distance in the leader-only control trial was 1.23 for the line formation and 0.497 for the GT formation. 146

SUMMARY

We present a motion control framework which allows a group of robots to work together to decide upon their motions by minimizing a collective cost without any central computing component or any one agent performing a large portion of the computation. When developing distributed control algorithms, care must be taken to respect the limited computational capacity of each agent as well as respect the information and communication constraints of the network. To address these issues, we develop a distributed, behavior-based model predictive control (MPC) framework which alleviates the computational difficulties present in many distributed MPC frameworks, while respecting the communication and information constraints of the network. In developing the multi-agent control framework, we make three contributions. First, we develop a distributed optimization technique which respects the dynamic communication restraints of the network, converges to a collective minimum of the cost, and has transients suitable for robot motion control. Second, we develop a behavior-based MPC framework to control the motion of a single-agent and apply the framework to robot navigation. The third contribution is to combine the concepts of distributed optimization and behavior-based MPC to develop the mentioned multi-agent behavior-based MPC algorithm suitable for multi-robot motion control.

CHAPTER I

BACKGROUND

This chapter introduces the background material necessary for a concise development of a behavior-based approach for model predictive control of multi-agent systems. As a fundamental component in the behavior-based MPC is the optimization, the chapter begins with an introduction to existing distributed optimization techniques in Section 1.1. Focus is then shifted to address MPC. An overview of a general optimal control-based MPC framework is given in Section 1.2, followed with background on the behavior-based approach in Section 1.3. We then discuss different techniques that have been employed for multi-agent MPC in Section 1.4. The chapter ends with a brief outline of the remainder of the thesis.

1.1 Distributed Optimization

We begin our discussion by considering the problem of having multiple agents optimize over a number of parameters in a distributed fashion. The distributed element requires agents to work together without any one agent doing the bulk of the work. The problem is complicated due to the fact that the optimization must be done while respecting the communication restraints present in the underlying communication network. Distributed parameter optimization forms a key element of our research as the multi-agent behavior-based MPC formulation developed in the thesis will require agents to solve a distributed parameter optimization problem.

Focus is given to gradient-based methods for distributed optimization as they are readily applicable to the MPC scenario. Multi-agent MPC will require each agent to simulate its state as well as its neighbors' states into the future in order to optimize its actions and form an opinion on its neighbors' actions. By limiting the method of distributed optimization to a gradient-based method, each agent will only be required to simulate the states into the future once at each time step. This will more readily permit the agents to meet computational and communication restraints present when executing in real-time.

1.1.1 Distributed Optimization Formulation

We address the distributed optimization methods in terms of the problem formulation presented in [61] and continued in [45, 88, 89, 84, 26, 46]. Specifically, assume that the function to be optimized is a summation of convex costs, i.e.

$$\min_{x} \sum_{i=1}^{N} f_i(x), \tag{1}$$

where $x \in \mathbb{R}^n$ is the parameter vector being optimized, N is the number of agents, and agent i only knows its individual cost, $f_i(x)$. The individual costs can be derived naturally from a distributed problem as done in [68] for resource allocation, or can be "designed" as done, for example, in [39, 18], where a central cost is split into separable components which are then assigned to the individual agents.

To be able to establish convergence to the collective minimum, certain convexity assumptions are made on the cost. Namely, each individual agents' cost, f_i, must be convex in x, while the summation, $\sum_{i=1}^{N} f_i(x)$, must be strictly-convex (see, for example, [5] for a thorough overview of convex functions and their properties). Without the assumptions on convexity, convergence is not guaranteed, although additional assumptions about local convexity may be made to ensure that agents converge to some local minimum.

For sake of clarifying the notation, one key point must be stressed. To perform distributed optimization, each agent will maintain its "own version" of the variables, denoted as $x_i \in \mathbb{R}^n$, with the constraint that $x_i = x_j \; \forall \; i, j \in \{1, ..., N\}$. This will allow (1) to be expressed as

$$\begin{gathered} \min_{x_i, i=1,...,N} \sum_{i=1}^{N} f_i(x_i). \\ \text{s.t. } x_i = x_j \; \forall \; i, j \in \{1, ..., N\} \end{gathered} \tag{2}$$

To perform the optimization in a distributed manner, the equality constraints are relaxed. Algorithms differ in the manner that they force agents to return to the constraint set.

1.1.2 Existing Methods for Distributed Optimization

While many classifications of distributed optimization algorithms may exist, a distinction has recently been made in both [90] and [45] that distributed optimization techniques can be divided between two categories: consensus-based gradient methods and decomposition or primal-dual methods.

The consensus-based approach is characterized by algorithms where, at each time step, every agent takes a gradient step along with an averaging or consensus step, e.g. [45, 88, 61, 89]. This concept was first developed in discrete time for dynamic networks in [61], for discrete-time random networks in [52], and recently extended to continuous-time static networks in [45]. Communication restraints introduced by the network are satisfied as each agent need only know its own cost and the parameter vector maintained by neighboring agents. The gradient step guides the agents to the optimal point, while the consensus step is exploited as a means of bringing agents into agreement.

While elegant in its simplicity, the consensus-based method can be slow to converge to the collective optimum [45, 88, 61]. This is due to the fact that the influence of the gradient step must go to zero as time goes to infinity in order for agents to reach agreement. However, [45] has shown that with a constant step-size, the centroid will converge to the optimum value with very desirable transient characteristics. Therefore, strategies have been developed in [45] and [61] which give tradeoffs between optimality and convergence rate. Alternatively, [88, 89, 27] have extended the method using a different consensus equation which will allow for better convergence characteristics at the cost of additional communication.

In contrast, decomposition methods distributedly reach agreement by exploiting the dual of the problem, e.g. [85, 74, 75], which requires the added collaborative update of pricing or dual variables, e.g. [62, 53, 5]. By solving the dual problem, the constraint is relaxed and a $\max\min$ optimization technique is used to solve for both the original variables as well as their respective Lagrange multipliers. Of particular interest is the decomposition method for multi-agent systems presented in [85], along with the gradient-based solution for dual problems first presented in [3]. When combined, these methods allow for a gradient-based multi-agent distributed optimization technique that, for simplicity, we refer to as dual-decomposition.

Despite the fact that dual-decomposition is guaranteed to achieve the collective optimal value, it has been shown to have poor transient convergence characteristics, such as oscillation [88, 75, 23]. This is not surprising as oscillation has been observed to be a characteristic of dual-formulations in general, e.g. [6]. Apart from undesirable convergence characteristics, it is not clear how to interpret Lagrange multipliers for dynamic networks when agents move into, and out of, communication range with each other.

A possibly third classification of distributed optimization algorithms is emerging which can be seen as combining aspects from both the consensus and decomposition methods. A method was presented in [88] as a consensus-based method using an alternative consensus equation. It was later shown in [89] for undirected networks, and [27] for directed networks, that this method combines aspects from both consensus and decomposition methods. The method is guaranteed to reach the optimal solution with constant gains, but requires agents to communicate both their opinions of the parameters as well as a vector of Lagrange multipliers. It has also been developed assuming static communication topologies.

A major contribution of our work is to design a distributed optimization framework for dynamic networks which has the desirable transient characteristics of the consensus-based approach as well as the convergence guarantees of the dual-decomposition approach. The dynamic aspect allows agents to move about the environment and change with whom they can communicate. Also, this is done without the need for shuffling an extra Lagrange multiplier vector between agents.

1.2 Model Predictive Control

We now shift focus to model predictive control for the remainder of the Chapter. First for the single agent case and then for the multi-agent case. We first introduce the general concept of MPC in this section, a behavior-based approach to MPC in Section 1.3, and a brief background on MPC for multi-agent systems in 1.4.

MPC is a method of control which allows the benefits of the, typically open-loop, optimal control solution to be realized while adding feedback into the optimization. This method was first introduced in industrial applications where engineers desired to capitalize on the benefit of optimal control being able to handle state and input constraints, but needed feedback to deal with uncertainties present in the processes [56]. As this method became successful, it caught the eye of the academic community, which has now introduced methods and conditions to ensure convergence and stability of the underlying dynamic system. In this section we will give a brief overview of the MPC algorithm, but the reader is encouraged to see [56] which provides an extensive review of the history and theory behind MPC.

At each iteration of an MPC algorithm a cost is minimized to find the optimal control over a certain horizon. We denote the time at which the optimization takes place as t_0 and the length of the horizon as Δ. To explicitly account for the fact that MPC requires the system to simulate forward in time, we introduce a double notation for time: $x(t;t_0)$[1] and $u(t;t_0)$ are simulated at time t_0 to be the state and input at time t. Note that $x(t;t)$ and $u(t;t)$ denote the actual state and input at time t. It is assumed that $x(t;t_0) \in X \subset \mathbb{R}^n$ and $u(t;t_0) \in U \subset \mathbb{R}^m$ with the dynamics denoted as $\dot{x}(t;t_0) = f(x(t;t_0), u(t;t_0))$. The

[1]Note that we use different notation when discussing distributed optimization and MPC to reflect the notation in the literature for each. In Chapters 2 and 3 where distributed optimization is discussed, x represents a parameter vector and f represents a cost. In the rest of the thesis, x represents a state, f represents dynamics, J, L, and Ψ represent costs, and θ is used to represent a parameter vector.

optimization problem can then be written in the following, generalized form:

$$\min_{u(\cdot;t_0)} \int_{t_0}^{t_0+\Delta} L\big(x(t;t_0), u(t;t_0)\big)dt + \Psi(x(t_0+\Delta;t_0)), \tag{3}$$

$$\text{s.t } \dot{x}(t;t_0) = f(x(t;t_0), u(t;t_0)),\ x(t_0+\Delta;t_0) \in X_f,$$

where $X_f \subset X$ is a terminal constraint set and $x(t_0;t_0)$ is known. With this setup in hand, the MPC Algorithm can be stated as follows:

Model Predictive Control

1. Minimize (3) with respect to the control $u(t;t_0)\ \forall\, t \in \big[t_0, \quad t_0+\Delta\big]$.
2. Apply $u(t_0;t_0)$ to the system.
3. Repeat at the following time instant.

This setup represents the constrained, nonlinear, continuous-time case, e.g. [56, 35, 58]. Other variants include linear dynamics, additional constraints, and/or discretization of the cost and dynamics. Also, as a note on implementation, step 2 states the ideal case where the cost is minimized at each time instant, but in actuality the control found in step 1 is applied for a small amount of time.

While intuitively this cost minimization may seem to guide the system in a desirable fashion, stability is not necessarily guaranteed. For the nonlinear case, different techniques for proving stability almost invariably use the cost in (3) as a Control Lyapunov function to stabilize the system to a set of goal states, e.g. [56, 35]. In such an evaluation, conditions are given on the terminal cost, terminal constraints, and often the existence of a control law which can be used to maintain the system in the goal state once the goal state has been reached (or is close to being reached), e.g. [56, 58, 54].

1.2.1 Dual-mode MPC

As control laws can often be defined to stabilize the system in a region around the desired goal state, a common technique known as dual-mode MPC utilizes a stabilizing control law to help ensure stability in the MPC formulation, e.g. [56, 58, 71, 54]. It is assumed that a controller, $u = \kappa_f(x)$, exists which will render the desired equilibrium locally stable. Without loss of generality, we assume this point to be the origin. The way $\kappa_f(x)$ is incorporated is through an interplay between $\kappa_f(x)$, X_f, and the cost to give asymptotic stability of the origin. We present the conditions on these components, given in [56], for sake of completeness:

B1) X_f closed

B2) $\kappa_f(x) \in U, \forall x \in X_f$

B3) X_f is positively invariant under $\dot{x} = f(x, \kappa_f(x))$

B4) $\frac{\partial \Psi}{\partial x}(x) f(x, \kappa_f(x)) + L(x, \kappa_f(x)) < 0 \; \forall \, x \in X_f, \, x \neq 0$

along with modest conditions on the dynamics (i.e. continuity, uniqueness of solutions, etc, e.g. [56, 12]).

The method is called "dual-mode" MPC as an optimal control mode of operation is combined with a stabilizing control mode of operation; although the stabilizing controller is never actually used to control the system. Intuitively, the controller is useful for a couple of reasons. Condition B3 allows the controller to be used as a hot start for optimization as, at the current optimization time, the solution from the previous time appended with the control generated by the terminal mode will satisfy the input and terminal constraints. Also, condition B4 states that the cost forms a CLF when using the terminal controller, making convergence somewhat expected.

1.2.2 Computation of Optimal Control Trajectories

Finally, once stability guarantees are met, the system must be able to accomplish the needed minimization at each time step. There are two groups of methods which are often used to solve for the optimal control trajectory. The first involves solving a two point boundary value problem where some of the differential equations depend on the known initial state, and the rest depend on the terminal cost and constraint, e.g. [43, 11]. Another family of methods known as "Direct Methods" involve approximating the state and/or control trajectories with other functions which may be easier to evaluate and apply to optimization methods, e.g. [11, 16]. However, even Direct Methods may not always sufficiently ease the burden of computing the optimal control solution at each time instant, e.g. [1], in which case other methods for simplification must be applied.

1.3 Behavior-based Model Predictive Control

As computation of an optimal solution at each time instant can be prohibitive, the state trajectory generation can be outsourced to behaviors designed to accomplish the desired task. While behavior-based control schemes constitute an entire class of robotic control paradigms [2], we will only consider those behaviors which can be considered as parameterized feedback control laws. In other words, we assume that under a behavior-based approach, the dynamics can be expressed as:

$$\dot{x}(t;t_0) = f\big(x(t;t_0), \kappa(x(t;t_0), \theta)\big), \tag{4}$$

where $\kappa(x, \theta)$ is a controller and θ is a vector of parameters.

Utilizing this form of behaviors, a wealth of different control applications can be built upon which all use some form of parameterized control. Examples include schema-based behaviors [2], gait design for robotic snakes [86] and legged locomotion [36], orbiting for unmanned aerial vehicles [63], ground vehicle obstacle avoidance [41], and even potential fields methods which are used in a wide variety of robotic motion applications [48, 76], just to name a few.

By using a behavior-based approach, the optimization is done over the switch times and parameters. This replaces an infinite dimensional control trajectory optimization problem with a finite dimensional parameter optimization problem. While there are numerous approximation techniques for solving the MPC problem, (see [1, 13] and the references therein), two are presented in Section 1.3.1. They are closely related to our work and provide a background on which our contributions can be seen. As the behavior-based MPC framework is developed for robot motion control, we then discuss in Section 1.3.2 how the behavior-based approach fits into the motion control paradigm. We end this section with a discussion of a specific navigation algorithm which will be extended in Chapter 5 as an example of behavior-based MPC.

1.3.1 Behavior-based MPC Background

The approach given in [13] is related to the method we develop as it utilizes pre-designed control laws at its core. The approach builds upon a technique presented in [87] where the nonlinear model to be fit to a linear parameter-varying model and control laws are designed to stabilize the linear system in different regions of the state space. After this has been done off-line, MPC is then used on-line as a tool to find optimal deviations from the output of these control laws. By so doing, the optimization is reduced to solving a QP problem. However, the cost to be minimized is restricted to a cost on the norm of the deviation from the original control input, which may be a limiting factor when applied to navigation. Also, the full nonlinear dynamics are not considered during the optimization.

The authors in [32] and [92] give an alternative method which is quite related to the proposed method. The basic idea being that a previously designed control law can be tuned on-line using MPC to achieve a desired result. This is fundamentally different from [13] in that instead of tuning the output from the control, the actual control law is tuned. The affects of tuning will be taken into account as the full dynamic model is considered in the optimization.

While the method we propose is closely related to that found in [32, 92], several contributions are made to the existing work. First, an analytic form to the gradient will be derived, whereas [32] uses a numerical approximations to the gradient and [92] proposes using methods such as pattern search and genetic algorithms. Second, switch time optimization techniques will be included to allow for multiple modes to be considered. This will permit this technique to be more readily applicable to hybrid control scenarios. Finally, a dual-mode approach, closely related to dual-mode MPC will be given to establish stability guarantees.

1.3.2 Motion Control and Behavior-based MPC

As the proposed method for behavior-based MPC is being developed to control robot motions, it is important to address where it will fit in terms of the motion control literature. Robot motion control design often consists of two interdependent pieces: planning of a trajectory and execution of the plan. There is a large spectrum of different techniques which range from addressing these pieces separately to executing them simultaneously.

On one end of the motion control spectrum is deliberative planning where a series of actions to take a robot from an initial state to a goal state is first planned and then acted upon. This encompasses a vast array of different planning methods. Some, such as graph based methods like Dijstra's algorithm and its derivatives, e.g. [47, 14], consist of finding an optimal path, but often suffer from the curse of dimensionality. Other approaches, such as sampling based methods, e.g. [14, 48], sacrifice a degree of optimality but are often able to significantly reduce the planning time, especially on higher dimensional spaces. However, these methods do not entirely solve the curse of dimensionality, and often neglect the dynamics of the robot [48].

On the other end of the motion control spectrum lie reactive planners. These types of planners decide upon an action based almost entirely upon local information. This incorporates a wide range of concepts from vector field approaches, e.g. [48, 2, 63, 76] to Lyapunov based control laws, e.g. [77]. The common thread being the ability to quickly compute a reaction to the current information available to the robot.

Often, reactive and deliberative planners are used together to leverage capabilities from each. For example, reactive planners can be used when a robot is in a precarious situation and the deliberative planner is not quick enough to avoid danger, e.g. [14, 48]. They may also be used to accommodate the dynamics as in way-point following, e.g. [4]. It is in the context of reactive planners working in collaboration with deliberative planners that we present the behavior-based MPC framework.

Behavior-based MPC could be considered a "glue" between a reactive controller stabilizing the dynamics and a deliberative plan directing the robot. Chapter 5 gives an example where a dual-mode approach to behavior-based MPC provides guarantees for convergence to a goal location while considering the full dynamics of the robot. It allows the deliberative planner to plan on the position space of the robot, avoiding the curse of dimensionality. The behaviors, or control laws, consider the full dynamics of the vehicle and are able to guarantee obstacle avoidance. The framework fits into a widely accepted paradigm for decomposing trajectory tracking into path-planning and execution, e.g. [29, 4]. This paradigm has even become the standard formulation in the navigation package of the increasing popular robot operating system (ROS), [72], where a global navigator finds a path and a local navigator follows that path.

1.3.3 The Dynamic Window Approach to Navigation

For a navigation algorithm to be successfully applied in unknown environments, it is essential to have guarantees on both obstacle avoidance and progression towards the goal location. These guarantees need to take the dynamic constraints of the vehicle into account, especially as the speed of the vehicle increases, e.g. [29, 82]. Moreover, the navigation scheme cannot consume too much computational power as the robot must perform other tasks, such as process sensor information and map the environment. To incorporate all of these demands, we build upon the dynamic window approach (DWA) presented in [24], which possesses all of the mentioned qualities, albeit without a guarantee of progression towards the goal (as noted, for example, in [64, 7, 82]).

DWA provides a direct way of incorporating dynamic constraints for fast navigation through an unknown environment, but lacks general convergence guarantees, [24, 64, 7]. The basic concept of the DWA algorithm first presented in [24] is, at each time step, to choose an arc for the robot to execute based on some predefined cost. This directly deals with the dynamic constraints of the vehicle as most wheeled vehicles contain a nonholonomic constraint which can be expressed using the unicycle motion model, e.g. [48]:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\psi}_3 \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ \omega \end{bmatrix}, \tag{5}$$

where (x_1, x_2) is the two dimensional position of the robot, ψ is the orientation, and v and ω are the translational and rotational velocities of the vehicle. Thus, arc-based motions are natural for most wheeled mobile platforms as they can be realized by commanding constant values for v and ω.

Commanding constant v and ω values has two advantages worth mentioning. First the (v, ω) pair can be chosen such that the transients due to the real dynamics of the vehicle can be ignored after a small window of time. This allows for quick simulation into the future to ensure obstacles are avoided. Second, by choosing to execute the (v, ω) pair over the simulated horizon, instead of solving for a trajectory of inputs on that horizon (as is typical in MPC), the computational burden is greatly reduced, [24, 19, 69]. Thus, DWA is able to guarantee obstacle avoidance while taking into account the dynamic constraints without imposing unreasonable computational burdens. These benefits have lead DWA to be a default "local planner" in the increasingly popular robot operating system's (ROS) navigation package, [72].

The term "local planner" is used as DWA does not incorporate information about the connectivity of the free space when planning its action, [7]. As such, it is known to get stuck in local minima, noted, for example, in [64, 7, 82]. To modify the algorithm to have guarantees of convergence, [64] utilized a control scheme based on incorporating navigation functions into the cost and used MPC stability techniques to ensure convergence to the goal. The navigation functions were chosen as they allow the cost to form CLF. They also provide an intuitive worst-case scenario: the worst the robot will do is simply follow the navigation function to the goal.

DWA is the focus of Chapter 5 as it provides an excellent example where dual-mode MPC concepts can be incorporated into the behavior-based MPC framework to ensure convergence of a DWA-like algorithm. By combining an arc-based controller with a terminal reference-tracking controller, information about the connectivity of the free space to the goal can be realized with a generic path-planner instead of the need for navigation functions. A guarantee on the convergence to the goal location is established based on properties of the reference tracking controller and conditions imposed on the cost.

1.4 Distributed Multi-Agent MPC

Distributed multi-agent MPC introduces an added level of difficulty to the MPC problem as network communication constraints must be taken into account while optimizing. This difficulty is compounded by the fact that the predictive element of MPC requires the ability of an agent to communicate or simulate neighboring agents' trajectories into the future. Additionally, to achieve optimality, agents must generally be able to influence neighbors' trajectories, not solely a vector of parameters as discussed in Section 1.1 for distributed optimization. The basis of several different formulations for multi-agent MPC is to address one or both of these problems.

In [10], authors present a method for distributed MPC. At each time step, agents iteratively broadcast their planned trajectory and control, optimize their own trajectory according to the communicated trajectories from all other agents, and then repeat the process until convergence. Under convexity conditions and linear dynamics, this method will guarantee an optimal solution. However, it comes at the cost of a large amount of communication at each time instant.

To mitigate the difficulty of optimizing the trajectory of neighboring agents, a method was developed in [20] and extended in [21]. The basic idea behind this method is to have agents communicate their trajectories before the optimization step. Then, an additional cost is added to each agents' individual cost to penalize deviation from the trajectories that the agents' told neighbors they would execute. This cost eliminates the need to repeatedly communicate trajectories at each time instant, albeit at a possible sacrifice to optimality.

The authors in [28, 75] propose a method for multi-agent, distributed MPC for agents executing linear dynamics in discrete time. The algorithm involves reformulating each agents dynamics and introducing pricing variables corresponding to the influence that agents have on each other. In this way, the number of times that agents are required to share state and control vectors at each iteration can be reduced while maintaining optimality and stability guarantees.

In [15], authors evaluated the decomposition and design of MPC formulations for linear systems with decoupled control. A discussion is given on the design of the costs and constraints such that stability and convergence guarantees are met for the collective system while respecting the communication topology of the underlying network. The methods were extended in [38] to include coupled control as well as a form of dynamic topology. It is important to note that distributed optimization must be employed between neighboring agents to solve for the control trajectories, requiring communication of entire trajectories, albeit only between neighboring agents.

In an attempt to overcome the difficulty of communicating trajectories, authors in [39] have presented a method where agents communicate only initial conditions. They are able to do this by introducing constraints on what agents are permitted to do. Agents then solve the optimal control problem assuming neighboring agents are working to minimize the same cost. Again, stability is established, but at the cost of optimality.

Finally, is it worth mentioning that an entire class of hierarchical methods also exist which do not strictly conform to the outlined multi-agent MPC architecture, e.g. [79, 80, 83]. These methods concentrate on the distributed optimal control of interconnected systems with distributed computing components which exhibit a hierarchy of knowledge and decision making. In particular, both [80] and [83] assume linear system dynamics and exploit the properties of the dynamics to formulate sound methods of respecting the hierarchy to come to a collective minimum.

By capitalizing on a behavior-based approach to MPC, we develop a method which relies on agents executing primitive, parameterized behaviors, with the optimization being done over the parameters. Agents are able to simulate each others trajectories after communicating solely the initial conditions and the parameters used in the behaviors. Agents are also able to influence each others trajectories by performing distributed parameter optimization. Moreover, each agent need only share information with neighboring agents defined by the underlying communication network, thereby respecting the communication constraints imposed by the network.

1.5 Organization

Each of the three key contributions are discussed and developed in the following chapters. We begin with a development of a new distributed optimization method for static communication topologies in Chapter 2, and extend it to dynamic topologies in Chapter 3. The focus is then shifted to developing the behavior-based MPC formulation. Chapters 4 and 5 discuss the behavior-based MPC formulation for the single agent scenario. Chapter 4 details the formulation of the behavior-based MPC approach, including gradient strategies and vector field examples. Chapter 5 gives an extended example where a dual-mode formulation is utilize to prove convergence when quickly navigating an unknown environment. Chapters 6 and 7 detail the development for the behavior-based MPC formulation for multiple agents. Chapter 6 gives a detailed derivation of the algorithm. Chapter 7 presents an example of a virtual leader approach to multi-agent formation control.

CHAPTER II

PROPORTIONAL-INTEGRAL DISTRIBUTED OPTIMIZATION FOR NETWORKED SYSTEMS WITH STATIC TOPOLOGIES

In general, tasks solved by multi-agent systems pose challenging problems as they require the agents to realize collective objectives using solely information local to each agent in the communication network, e.g. [57, 59]. Additionally, many real-world examples complicate task completion as they require agents to move about the environment, changing which agents can communicate. To give structure to the problem, tasks are often defined in terms of a cost, where task completion corresponds to minimizing the cost, e.g. [59]. Therefore, this chapter focuses on the development of a distributed optimization technique which allows a system of agents to converge to a collective minimum under static communication topologies. Chapter 3 will extend this work to cope with dynamic topologies.

The distributed optimization framework developed in this chapter and the next prove to be an essential component to the multi-agent behavior-based MPC algorithm developed in subsequent chapters. The reason being is that the behavior-based approach changes required optimization problem into a parameter optimization problem. This allows distributed parameter optimization to be directly applied to solve the behavior-based MPC algorithm.

As mentioned in Section 1.1 distributed optimization techniques have recently been categorized into consensus methods and dual methods. In this chapter, we show that the consensus-based and decomposition gradient algorithms are actually very closely related when examined in context of the underlying constrained optimization problem that is solved by these methods. Specifically, we formulate both the dual-decomposition method in [85] and the consensus-based method in [45] in control theoretic terms to draw parallels and gain intuition behind why they can naturally be joined together. In fact, it will become apparent that dual-decomposition is very closely related to integral (I) control, and the consensus method is closely related to proportional (P) control.

The relation to proportional and integral control explains the effects of using such methods. Much like integral control, dual-decomposition is notorious for being oscillatory, but guarantees convergence to the constraint, e.g. [88, 75]. Similar to proportional control, the consensus-based methods with constant gains are much more damped in their transient response, but do not converge. To achieve convergence, a diminishing stepsize on the gradient is necessary, but also slows down settling times, [45, 88, 61].

The main contribution made in this chapter is to combine these two methods to form a new, proportional-integral (PI) distributed optimization method. The constrained optimization problem being solved is first examined in Section 2.1 to give necessary background and intuition to the approach taken. Sections 2.2 and 2.3 examine two existing distributed optimization techniques under the constrained optimization approach. Section 2.4 then presents a new PI distributed optimization approach for fixed topologies, formulated by combining the approaches mentioned in Sections 2.2 and 2.3. The chapter ends with concluding remarks in Section 2.5.

2.1 Introducing the Constrained Optimization Problem

This section introduces the background information necessary to characterize PI distributed optimization and give intuition behind its formulation. It begins with the formulation of the distributed optimization problem that is addressed in this chapter. Following this, the graph-based model of the multi-agent network will be introduced. Gradient-based constrained optimization is then discussed from a high level viewpoint to develop intuition about the underlying relationship between dual-decomposition and the consensus based method. As similarities to PI control will become readily apparent, this section ends with a brief introduction of the PI control metrics that are used to compare these methods.

2.1.1 Problem Formulation

As mentioned in Section 1.1, we present the distributed optimization problem in the same formulation as [61, 45, 88, 89, 84, 26, 46]. Specifically, assume that the function to be optimized is a summation of strictly-convex costs, i.e.

$$\min_x \sum_{i=1}^{N} f_i(x), \tag{6}$$

where $x \in \mathbb{R}^n$ is the parameter vector being optimized, N is the number of agents, and agent i only knows its individual cost, $f_i(x)$.[1]

To be able to establish convergence to the collective minimum certain convexity assumptions are made on the cost. Note that a convex function is defined as a function that satisfies:

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y) \tag{7}$$

where $0 < \theta < 1$. A function is strictly-convex if strict inequality holds in (7) (see, for example, [5] for a thorough overview of convex functions and their properties). The following assumptions about the costs are used throughout the paper:

Assumption 1. *$f_i(x) : \mathbb{R}^n \to \mathbb{R}$, $i \in \{1, ..., N\}$, are convex, twice continuously differentiable functions and the summation $\sum_{i=1}^{N} f_i(x)$ is strictly-convex, twice continuously differntiable function.*

Assumption 2. *The solution $f^* = \min_x \sum_{i=1}^{N} f_i(x)$ and respective optimal parameter vector, x^*, exist and are finite.*

Remark 1. *We note that the differentiability assumption has been relaxed in many of the references to address subgradient optimization. However, we do not concern ourselves with relaxing this assumption as it does not add to the development of the paper.*

[1]Note again that to maintain consistent notation with the distributed optimization literature, we use a different notation than in other chapters. When discussing MPC, x will correspond to the state vector and f will denote a dynamic function.

Once again, for sake of clarifying the notation, one key point must be stressed. To perform distributed optimization, each agent will maintain its "own version" of the variables, denoted as $x_i \in \mathbb{R}^n$, with the constraint that $x_i = x_j \; \forall \, i, j \in \{1, ..., N\}$. This will allow (1) to be expressed as

$$\min_{x_i, i=1,\ldots,N} \sum_{i=1}^{N} f_i(x_i). \tag{8}$$
$$\text{s.t. } x_i = x_j \; \forall \, i, j \in \{1, ..., N\}$$

To perform the optimization in a distributed manner, the equality constraints are relaxed. Algorithms differ in the manner that they force agents to return to the constraint set.

While much of the work on distributed optimization has been developed in discrete-time formulations, which are amenable for implementation, e.g. [90, 61, 85, 51], a great deal of work recently has been made in continuous-time [45, 88, 89, 74, 75, 46, 26]. Continuous-time analysis has proven useful as it allows Lyapunov stability conditions to be directly applied to the update-equations for convergence analysis. It also allows for an intuitive connection between the optimization algorithm proposed in this paper and proportional-integral control. Moreover, a discretization of the framework proposed in this paper does not pose a significant contribution. The proportional element has been evaluated in discrete time in [61] and the integral element has been evaluated in discrete time in [85, 74, 75]. Furthermore, a closely related PI distributed optimization algorithm developed in [88, 89, 26] (discussed further in Section 2.4.1) was discretized in [88].

2.1.2 Networked Multi-Agent Systems

We now introduce the terminology and properties of multi-agent systems that will be used to formulate the distributed optimization algorithms and discuss their convergence. The term "agent" is used to refer to a computing component and it is assumed that agents only communicate with each other through a defined, static network topology. This is representative of a great number of different multi-agent systems, from communication networks to teams of robots, e.g. [57, 81, 75].

The interconnections of the network are represented through an undirected graph $\mathcal{G}(\mathcal{V},\mathcal{E})$. The set of nodes, $\mathcal{V}$, is defined such that $v_i \in \mathcal{V}$ corresponds to agent i. Communication constraints are represented through the set of edges in the graph, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, where $(v_i, v_j) \in \mathcal{E}$ iff agents i and j can directly communicate. The number of agents is then given by $|\mathcal{V}| = N$ and the number of communication links is given by $|\mathcal{E}| = M$. To prove convergence of the distributed optimization methods, the following assumption on the graph topology is made:

Assumption 3. *The graph $\mathcal{G}(\mathcal{V},\mathcal{E})$ is connected.*

Associated with this graph are two important, and related matrices. The first is the incidence matrix, $D \in \mathbb{R}^{N\times M}$ which is formed by arbitrarily assigning an orientation to each edge and can be defined as

$$D = [d_{ik}] = \begin{cases} 1 & \text{edge } k \text{ points to node } i \\ -1 & \text{edge } k \text{ originates at node } i \, \cdot \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

The second matrix, the graph Laplacian, is closely related to D and can be defined as $L = L^T = DD^T \in \mathbb{R}^{N\times N}$. Note that the resulting values for the elements of L are independent of the orientation assigned to each edge, [57].

We utilize both the incidence matrix and the graph Laplacian to form larger, aggregate matrices to incorporate the fact that each agent will be maintaining an entire vector of values. First, let x_{ij} denote the j^{th} element of x_i, $z_j \triangleq [x_{1j}, x_{2j}, ..., x_{Nj}]^T$ is the combination of all the j^{th} elements, and $z \triangleq [z_1^T, ..., z_n^T]^T \in \mathbb{R}^{Nn}$ is the aggregate state vector. The aggregate matrices can then be written as $\mathbb{D} \triangleq I_n \otimes D$ and $\mathbb{L} \triangleq I_n \otimes L$. This notation expresses the concept that an aggregate graph is formed where there are n replicas of $\mathcal{G}$, each corresponding to one of the elements of the vector being optimized. The aggregate graph will not be connected, but have n connected components, given Assumption 3. Therefore, the aggregate Laplacian will have the following properties (see, for example, [57]):

1. $\mathbb{L} = \mathbb{L}^T = \mathbb{D}\mathbb{D}^T$
2. $\mathbb{L} \succeq 0$
3. The eigenvectors associated with the zero eigenvalues of the aggregate Laplacian are $\alpha \otimes \mathbf{1}$, where $\alpha \in \mathbb{R}^n$
4. If $\dot{z} = -\mathbb{L}z$, the solution, $\bar{z} = z(t)$ as $t \longrightarrow \infty$, will be the projection of $z(0)$ onto the set $\alpha \otimes \mathbf{1}$ for $\alpha \in \mathbb{R}^n$. Moreover, the vector $-\mathbb{L}z$ will point along a line orthogonal to the set $\{\alpha \otimes \mathbf{1} | \alpha \in \mathbb{R}^n\}$.

One further property that will be exploited throughout the paper comes from the incidence matrix. The constraint in (2) that $x_i = x_j \; \forall \; i, j \in \{1, ..., N\}$ can be written as $\mathbb{D}^T z = 0$. This can be verified by first considering the scalar case where $n = 1$ and $\mathbb{D} = D$. $D^T z = 0$ will enforce that $x_{k_1} - x_{k_2} = 0$, where k_1 and k_2 correspond to the verticies associated with edge k. Then through Assumption 3, $x_i = x_j \; \forall \; i, j \in \{1, ..., N\}$. The same argument can be extended to $n > 1$ case by noting that $\mathbb{D}^T z = \begin{bmatrix} D^T z_1 \\ \vdots \\ D^T z_n \end{bmatrix}$.

Finally, this notation allows the distributed optimization problem to be presented in a compact form:

$$\begin{aligned} &\min_z f(z) \\ &\text{s.t. } h(z) = 0. \end{aligned} \tag{10}$$

where $f(z) = \sum_{i=1}^N f_i(x_i)$ and $h(z) = \mathbb{D}^T z$.

2.1.3 PI Control as Gradient Method for Constrained Optimization

We now take note of the structure of (10) to give intuition to the relationship between the gradient methods presented in Sections 2.2 and 2.3. The development in this section will not dwell on the details of constrained optimization, as it has been a widely studied area,

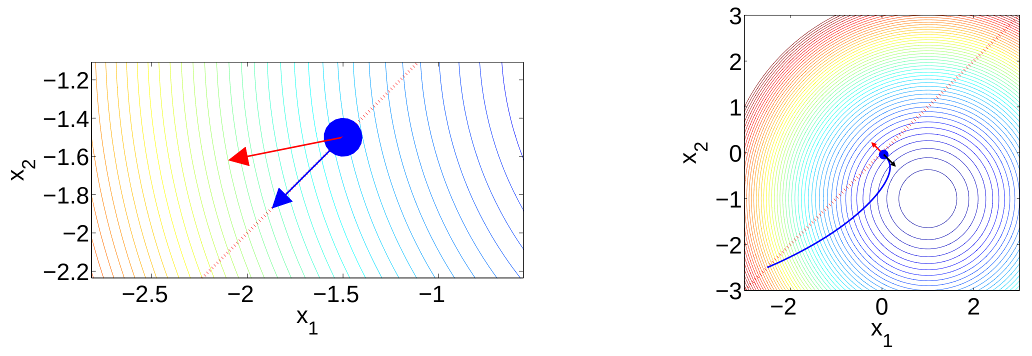


Figure 1: This figure shows the results of using the cost $f(z) = (x_1 - 1)^2 + (x_2 + 1)^2$. Left: Dotted line shows the equality constraint and the arrows show the gradient and projected gradient. Right: Result of performing the PI gradient method for optimization given in (13). The trajectory of the two states is shown ending in the final condition denoted by the solid circle and the constraint is shown as a dotted line. The arrows show the final gradient and Lagrange multiplier multiplied by the constraint. As expected, these are equal in magnitude, but opposite in direction.

e.g. [53, 5]. Rather, it will be focused on forming a control law to return the state to the constraint set when the constraints are relaxed.

Without constraints, a gradient method for optimization of the problem would simply take the form $\dot{z} = -k_G \frac{\partial f}{\partial z}^T$, where $k_G \in \mathbb{R}^+$ is some gain. However, when the optimization includes constraints, the update to the variables being optimized cannot be in any arbitrary direction. The update can only occur in a direction that will allow the state to continue to satisfy the constraint. As the constraints in (10) are linear, this involves taking the gradient and projecting it onto the constraint space, as shown in Figure 1.

It should be noted that the difference between an unconstrained gradient and a constrained gradient could be written in terms of the addition of a term perpendicular to the constraint set. This could be expressed as $\frac{\partial f}{\partial z} + \lambda^T \frac{\partial h}{\partial z} = \frac{\partial f}{\partial z} + \lambda^T \mathbb{D}^T$. The dynamics of the resulting optimization would then be

$$\dot{z} = -k_G(\frac{\partial f}{\partial z}^T + \mathbb{D}\lambda(t)). \tag{11}$$

However, computing $\lambda(t)$ in a distributed fashion could be difficult as it may require knowledge from the entire network.

Alternatively, if the gradient method is permitted to violate the constraint, control terms can be added to guide the state back to the constraint at the optimal point. The first term we consider is a term proportional to the error from the constraint. Allow $\lambda(t) = \frac{k_P}{k_G}e(t)$ where $e(t) = \mathbb{D}^T z(t)$ is the error at each edge of the graph. This can be seen to be a logical choice because, as mentioned in Section 2.1.2, $-\mathbb{D}e(t) = -\mathbb{L}z(t)$ will point along a line orthogonal to the constraint set. In other words, it points in the right direction, but with possibly the wrong magnitude. This gives the dynamics

$$\dot{z} = -k_G \frac{\partial f}{\partial z}(z) - k_P \mathbb{D}e(t) = -k_G \frac{\partial f}{\partial z}(z) - k_P \mathbb{L}z(t). \tag{12}$$

As will be discussed in Section 2.3, the similarity of (12) to proportional control is perpetuated in that the steady-state solution will have a constant error from the desired optimal point. Basically, the effort produced by introducing an error term proportional to the deviation from the constraint will fall short of the needed effort to drive the state all of the way to the constraint set.

To compensate for the steady state error, it is common to add an integral term to the control, e.g. [25]. This would lead to a $\lambda(t)$ of the form $\lambda(t) = \frac{k_P}{k_G}e(t) + \frac{k_I}{k_G}\int_{t_0}^{t} e(s)ds$. Over time, the integral term will build up the necessary effort to reach the constraint. With this additional term, the dynamics of the system can be expressed as

$$\dot{z} = -k_G \frac{\partial f}{\partial z}^T - \mathbb{D}\Big(k_P e(t) + k_I \int_{t_0}^{t} e(s)ds\Big) = -k_G \frac{\partial f}{\partial z}^T - k_P \mathbb{L}^T z - k_I \mathbb{L} \int_{t_0}^{t} z(s)ds. \tag{13}$$

It will be shown in Section 2.4 that under Assumptions 1, 2, and 3, the dynamics in (13) will indeed converge to a collective minimum, as shown in Figure 1.

While this method for obtaining a gradient strategy to arrive at the desired optimal value may seem somewhat trivial or ad-hoc, it will be seen in Section 2.2 that the dual-decomposition distributed optimization method will exactly correspond to adding an integral term. Similarly, in Section 2.3, it is shown that the consensus-based method will be exactly the proportional term. Therefore, we combine the two methods in Section 2.4 to form a PI distributed optimization method.

2.1.4 PI Performance Metrics

As the distributed control laws developed throughout the remainder of this paper are closely related to proportional and integral control laws, we give a brief introduction to the performance metrics that will be employed for comparison. These metrics are important as there really is no single metric which best determines which control law is most suitable. For example, as discussed in [25], proportional control can converge quickly, but may result in a steady-state error. As the proportional gain is increased, the steady-state error will typically decrease up to the point where the system becomes unstable. On the other hand, integral control can be introduced to eliminate steady-state error, but dampening will be decreased and this will result in greater oscillation, overshoot, and slower convergence.

To say that one method is "better" would required a reference to a specific application. To be able to judge which method is more suitable for the given application, the following performance metrics, typical for classic control evaluation (e.g. [25]), are used:

- Percent overshoot (M_p): The percentage of the distance that the state overshoots the final value, given as $\frac{x_{max}-x_f}{x_f-x_0} \times 100$.
- Settling time (t_{10} and t_1): Time it takes for the state to converge to within 10 percent and 1 percent of the final value. For example, t_{10} is the smallest t such that $.9\frac{x_f}{x_f-x_0} < x(t) < 1.1\frac{x_f}{x_f-x_0} \; \forall \, t > t_{10}$.
- Percent error (% error): The percentage of error from the optimal value ($\frac{|x^*-x_f|}{x_f-x_0} \times 100$).

where x_0 is the initial value, x_f is the final value, and x_{max} is the maximum value reached. For simplicity, we have assumed $x_{max} \geq x_f > x_0$. As these values are measures of scalar states, the worst case over all agents will be presented in each evaluation.

Also note that numerical results depend upon the value of the gains and initial conditions. To allow for a fair comparison between examples throughout the paper, all gains

($k_G, k_P,$ and k_I) are assigned a value of 1. Similarly, all initial conditions are assigned a value of 0, unless otherwise stated.

2.2 Dual Decomposition

This section introduces the concept of gradient-based distributed optimization through the introduction of dual-decomposition, which has been used in a variety of different applications, e.g. [68, 88, 85, 18, 75, 28]. Notation, examples, and proofs are given which will allow for a concise development of the distributed optimization methods in Sections 2.3 and 2.4.

As already mentioned, dual-decomposition will be akin to integral control for constrained optimization. However, to provide intuition as to the origins and the theoretical underpinnings of this method, it is presented here in a more typical fashion relying upon the theory of dual-optimization, e.g. [53, 5]. The formulation introduced here is closely related to that found in [85], except that we use Uzawa's saddle point method, [3], to update both the parameters and dual variables simultaneously. This permits a continuous-time formulation where Lyapunov methods can be readily applied to establish convergence. After presenting the algorithm, the relation to integral control will be evaluated. This section will end with a distributed implementation and a numerical example.

2.2.1 Dual-Decomposition for Networked Systems

The basic idea behind dual decomposition is to introduce n copies of the variables, with the constraint that the copies be equal. The dual is then formed to relax the added constraints and a $\max \min$ optimization technique is used to solve the dual problem. In this paper, we use a gradient method introduced in [3] for saddle point finding. This will allow for a distributed solution to the problem where each agent uses only local information defined by the network graph, $\mathcal{G}$.

The dual problem to (2) can be formed by introducing a Lagrange multiplier vector,

$\mu_k \in \mathbb{R}^n$, $k = 1, ..., M$, for each edge in $\mathcal{G}$. It can be written as:

$$\max_{\mu_k, k=1,...,M} \min_{x_i, i=1,...,N} \left\{ k_G \sum_{i=1}^{N} f_i(x_i) + k_I' \sum_{k=1}^{M} \mu_k^T (x_{k_1} - x_{k_2}) \right\} \tag{14}$$

where, again, the subscripts k_1 and k_2 correspond to the agents which make up the k^{th} edge and $k_G, k_I' > 0$ are constant gains. Note that due to the constraint equaling zero, k_I' has no influence and k_G scales the cost, but does not change the location of the optimal point. Equation (14) can be simplified by forming an aggregate Lagrange multiplier vector, $\mu \in \mathbb{R}^{Mn}$, in the same fashion that the aggregate state, z, was formed. This allows us to reintroduce the constraint as $\mathbb{D}^T z = 0$ and rewrite (14) as:

$$\max_{\mu} \min_{z} F(z, \mu) = k_G f(z) + k_I' z^T \mathbb{D} \mu. \tag{15}$$

To solve this max-min problem, we use a technique first developed in [3] for saddle point finding and has more recently gained attention for its applicability to distributed optimization, e.g. [88, 89, 75, 74]. The basic idea behind this approach is that dynamics can be assigned to the variables being optimized and convergence can be established using control methods such as Lyapunov stability.

For a saddle point finding problem, where $F(z, \mu)$ is strictly-convex in z and strictly concave in μ, [3] shows that applying the dynamics

$$\dot{z} = -\frac{\partial F}{\partial z}^T, \qquad \dot{\mu} = \frac{\partial F}{\partial \mu}^T, \tag{16}$$

the system will converge asymptotically to the saddle point. Taking the partials of (15), the dynamics can be expressed as:

$$\dot{z} = -k_G \frac{\partial f}{\partial z}^T - k_I' \mathbb{D} \mu \tag{17}$$

$$\dot{\mu} = k_I' \mathbb{D}^T z. \tag{18}$$

However, we note that (15) is not strictly concave in μ, rather, it is linear. This requires further evaluation, which is done in the proofs of Theorems 1 and 2. While there exist proofs for dual-decomposition, e.g. [68, 23], we present an alternative proof here to show the relationship of dual-decomposition to the underlying constrained optimization problem. This will allow us to easily extend these proofs in Section 2.4 for the PI distributed optimization method that will be developed. The proofs use the same Lyapunov candidate function as [74, 23] to prove convergence, but differ in the application of Lasalle's invariance principal and the proof that the equilibrium reached is the collective minimum.

Theorem 1. *Given Assumptions 1, 2, and 3 as well as the dynamics in (17) and (18), the saddle point* $(\dot{z}, \dot{\mu}) = (0, 0)$ *is globally asymptotically stable.*

Proof. Using the candidate Lyapunov function $V = \frac{1}{2}(\dot{z}^T\dot{z} + \dot{\mu}^T\dot{\mu})$, $\dot{V}$ can be written as:

$$\begin{aligned}\dot{V} &= \dot{z}^T\ddot{z} + \dot{\mu}^T\ddot{\mu} = -\dot{z}^T\big(k_G\frac{\partial^2 f}{\partial z^2}\dot{z} + k_I'\mathbb{D}\dot{\mu}\big) + k_I'\dot{\mu}^T\mathbb{D}^T\dot{z} \\ &= -k_G\dot{z}^T\frac{\partial^2 f}{\partial z^2}\dot{z} = -\dot{z}^T H(z)\dot{z} \leq 0 \,\forall \dot{z}, \dot{\mu}\end{aligned} \tag{19}$$

where $H(z) = k_G\frac{\partial^2 f}{\partial z^2} \succ 0$ due to strict convexity given by Assumption 1. As there is no dependence upon $\dot{\mu}$ in $\dot{V}$, LaSalle's invariance principle must be used to show convergence to $(\dot{z}, \dot{\mu}) = (0, 0)$.

Let the set where $\dot{V} = 0$ be denoted as

$$S = \{(\dot{z}, \dot{\mu}) | \dot{V} = 0\} = \{(\dot{z} = 0, \dot{\mu} \in \mathbb{R}^{Mn})\} \tag{20}$$

To see that that the only solution in which the complete state $(\dot{z}, \dot{\mu})$ can remain in S is the equilibrium $(0, 0)$, use the fact that to stay in $S \Rightarrow \dot{z} = 0 \,\forall t \Rightarrow \ddot{z} = 0$. From this we see that

$$\ddot{z} = -H(z)\dot{z} - k_I'\mathbb{D}\dot{\mu} = -k_I\mathbb{D}\mathbb{D}^T z = -k_I\mathbb{L}z = 0,$$

where $k_I'^2 = k_I$. For the connected graph, the only z such that $-\mathbb{L}z = 0$ is $z = \alpha \otimes \mathbf{1}$, $\alpha \in \mathbb{R}^n$. This shows two things:

1. $x_i = x_j \,\forall i, j$ which means that the agents reach consensus.

2. $\dot{\mu} = k_I' \mathbb{D}^T(\alpha \otimes \mathbf{1}) = 0$ which shows that the only possible value for $\dot{\mu}$ which stays in S is $\dot{\mu} = 0$.

Since V is radially unbounded, this completes the proof. □

Theorem 2. *Given Assumptions 1, 2, and 3 as well as the dynamics in (17) and (18), the saddle point* $(\dot{z}, \dot{\mu}) = (0, 0)$ *corresponds to the collective minimum.*

Proof. To validate that a feasible solution is a local extremum, z^*, of a constrained optimization problem *it is sufficient to show that* z^* *corresponds to a regular point (i.e. rows of* $\frac{\partial h}{\partial z}(z^*)$ *are linearly independent) and there exists* λ^* *such that*

$$0 = \frac{\partial f}{\partial z}(z^*) + \lambda^{*T} \frac{\partial h}{\partial z} \tag{21}$$

where $h(z) = 0$ *is the constraint and* $f(z)$ *is the cost*, (see [53] for a discussion on local extremum and regular points). Due to Assumption 1, the only extremum is the collective minimum. Therefore, this proof is performed in two steps. First, we show that the saddle point corresponds to a feasible point satisfying (21), then we show that the saddle point is indeed a regular point.

The proof of Theorem 1 showed that $(\dot{z}, \dot{\mu}) = (0, 0)$ implies that consensus is reached. Thus, the constraints are satisfied and the saddle point is feasible. Also, by noting that $\frac{\partial h}{\partial z} = \mathbb{D}^T$ for the problem at hand, (17) gives us

$$\dot{z} = 0 = -k_G \frac{\partial f}{\partial z}^T - k_I' \frac{\partial h}{\partial z}^T \mu \Rightarrow 0 = k_G \frac{\partial f}{\partial z} + k_I' \mu^T \frac{\partial h}{\partial z}. \tag{22}$$

Allowing $\lambda = \frac{k_I'}{k_G}\mu$, (21) is satisfied.

The saddle point must now be shown to be a regular point. To do so, we show that the convergent point is a regular point to the problem in which edges are removed from $\mathcal{G}$ to form a minimum spanning tree (for undirected graphs, a minimum spanning tree is a connected graph with N nodes and $N - 1$ edges, e.g. [57]). Due to Assumption 3, a minimum spanning tree, $\mathcal{G}_T$, exists such that $\mathcal{E}_T \subset \mathcal{E}$. The saddle point is shown to be regular by first showing that the representation of the constraints using $\mathcal{G}_T$, i.e. $\mathbb{D}_T^T z = 0$, is linearly independent and then showing that if a λ can be found to satisfy (21) for $\mathcal{G}$, a λ_T can be found to satisfy (21) for $\mathcal{G}_T$.

Let $D_T \in \mathbb{R}^{N \times N-1}$ be the incidence matrix associated with $\mathcal{G}_T$. The graph Laplacian for a connected graph with N nodes always has rank $N-1$, [57]. Therefore, $\mathbb{D}_T$ has full rank, which for $n = 1$, gives that $D_T^T z = 0$ is a linearly independent set of constraints. For $n > 1$, $\mathbb{D}_T = I_n \otimes D_T$, and, as noted in Section 2.1.2, $\mathbb{D}_T^T z = \begin{bmatrix} D_T^T z_1 \\ \vdots \\ D_T^T z_n \end{bmatrix}$ which will also be linearly independent.

Without loss of generality, we can assume that $D = \begin{bmatrix} D_T & D_R \end{bmatrix}$ where D_R containts the "redundant" edges not contained in $\mathcal{G}_T$. Since D_T has the same rank as D, the columns in D_R can be expressed as linear combinations of the columns of D_T. In other words, $D_R = D_T \delta$, where $\delta \in \mathbb{R}^{N-1 \times M-N+1}$.

Without loss of generality, we can assume the elements in z have been rearranged to write $D = \begin{bmatrix} D_T & D_R \end{bmatrix}$, where $\mathbb{D}_R = I_n \otimes D_R$. Since $D_R = D_T \delta$, $\mathbb{D}_R$ can be expressed as $\mathbb{D}_T \Delta$, where $\Delta = \mathbf{1} \otimes \delta$. We can separate λ as $\lambda = \begin{bmatrix} \lambda' \\ \lambda'' \end{bmatrix}$ which allows us to write $\mathbb{D}\lambda = \mathbb{D}_T \lambda' + \mathbb{D}_R \lambda'' = \mathbb{D}_T \lambda' + \mathbb{D}_T \Delta \lambda''$. Therefore, if a λ is found such that (21) is satisfied for $\mathcal{G}$, λ_T can be defined as $\lambda_T = \lambda' + \Delta \lambda''$. Thus, the solution is a regular point for the constraint $\mathbb{D}_T^T z = 0$. $\square$

2.2.2 Integral Control

With the optimization framework in hand, the loop can be closed on the discussion begun in Section 2.1.3 by relating the dynamics in (17) and (18) to integral control. Note that the Lagrange multiplier, μ, can be expressed as follows (assuming $\mu(t_0) = 0$):

$$\mu(t) = \int_{t_0}^{t} \dot{\mu}(\tau) d\tau = \int_{t_0}^{t} k_I' \mathbb{D}^T z(\tau) d\tau = k_I' \mathbb{D}^T \int_{t_0}^{t} z(\tau) d\tau. \tag{23}$$

This allows $\dot{z}$ to be expressed as:

$$\dot{z}(t) = -k_G \frac{\partial f}{\partial z}^T - k_I \mathbb{D}\mathbb{D}^T \int_{t_0}^{t} z(\tau) d\tau = -k_G \frac{\partial f}{\partial z}^T - k_I \mathbb{L} \int_{t_0}^{t} z(\tau) d\tau, \tag{24}$$

which gives the same result obtained in (13) assuming $k_P = 0$. After closer inspection of (18), one can see that the Lagrange multiplier, μ is indeed the integral of the weighted error referred to in Section 2.1.3.

2.2.3 Distributed Implementation

While the analysis of this method has been performed from the point of view of the entire system, its utility as a distributed optimization technique would be questionable if it were not possible for the algorithm to be executed by each agent using only local information. Therefore, we now present the algorithm in terms of implementation of a single agent and discuss the information and communication requirements.

Equations (17) and (18) can be written in terms of execution by a single agent, i, as follows:

$$\dot{x}_i = -k_G \frac{\partial f_i}{\partial x}^T (x_i) - k_I' \sum_{j \in \mathcal{N}_i} \mu_i^j, \tag{25}$$

$$\dot{\mu}_i^j = k_I'(x_i - x_j), \tag{26}$$

where for simplification we have introduced the Lagrange multiplier variables $\mu_i^j = -\mu_j^i = d_{i,k_{ij}} \mu_{k_{ij}}$ where k_{ij} is the edge connecting agents i and j and it is assumed that $\mu_k(0) = 0$, $k = 1, ..., M$.[2] Note that $\mathcal{N}_i$ denotes agent i's neighborhood set, or agents with which agent i can communicate. By inspection, agent i can compute $\dot{x}_i$ and $\dot{\mu}_i^j \ \forall \ j \in \mathcal{N}_i$ using only its own state and the states of its neighbors. Therefore, we emphasize that *the only piece of information that an agent needs to communicate with its neighbors is its version of the state vector.*

[2]By uniqueness of solutions to differential equations, $\dot{\mu}_i^j(t) = -\dot{\mu}_j^i(t) \ \forall t$

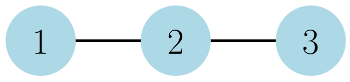


Figure 2: This figure depicts the "Line" network structure used for the examples in Sections 2.2, 2.3, and 2.4

2.2.4 Example

To illustrate behaviors typical of dual decomposition, we give a numerical example. Let the individual costs be defined as follows:

$$\begin{aligned} f_1(x_1) &= (x_{11}-1)^2 + \frac{1}{3}(x_{11}-x_{12})^2, \\ f_2(x_2) &= (x_{22}-3)^2 + \frac{1}{3}(x_{21}-x_{22})^2, \\ f_3(x_3) &= (x_{31}-6)^2 + \frac{1}{3}(x_{31}-x_{32})^2. \end{aligned} \tag{27}$$

where $x_i = \begin{bmatrix} x_{i1} & x_{i2} \end{bmatrix}^T$ and the network structure takes the form of the line graph shown in Figure 2. In other words, agents 1 and 2 as well as 2 and 3 can communicate, but agents 1 and 3 cannot. The collective cost is given by $\sum_{i=1}^{3} f_i(x_i)$, where $x_1 = x_2 = x_3$, has the optimal solution of $x^* = \begin{bmatrix} 3.4 & 3.2 \end{bmatrix}^T$.

Figure 3 and Table 1 show the results of employing these dynamics. As seen in Figure 3, there is oscillation in the solution as the different agents communicate and vary their values. This oscillation is quite typical of dual-decomposition [74], and it will be seen that the oscillation increases with an increase in problem complexity and number of agents in Sections 2.4.4 and 2.4.5. Table 1 shows that the I control (corresponding to dual-decomposition) has a large overshooot and slower settling times when compared with the P and PI control laws (which are discussed in Sections 2.3 and 2.4). This is to be expected as the integral term will decrease the dampening of the system [25]. Moreover, as expected, Table 1 shows that there is zero steady-state error when using dual decomposition.

Table 1: The results of performing proportional, integral, and PI distributed optimization for the convex optimization problem

	P: $\gamma = 1$	P: $\gamma = \frac{1}{1+.1t}$	I	PI
M	0.11%	34.66%	24.24%	14.95%
t_{10}	3.54	103.73	5.61	5.14
t_1	6.66	869.32	15.04	13.19
% error	43.58%	1.97%	0%	0%

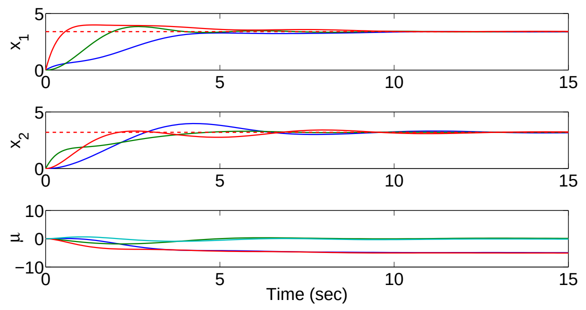


Figure 3: This figure shows the results from the convex optimization example using dual-decomposition

2.3 *Consensus Based Distributed Optimization*

This section introduces the consensus-based distributed optimization technique, first outlined in [61], which will give the proportional component in the new PI distributed optimization method. After formulating the algorithm in terms of notation presented in previous sections, characteristics of the convergence are discussed in terms of the constrained optimization problem. This section will end by resuming the example started in the previous section to present a comparison between the distributed optimization methods.

2.3.1 Consensus Based Algorithm

While originally given in discrete time, we present the consensus based distributed optimization problem in continuous time as done in [45] to maintain consistent notation. In stark contrast to the development of dual-decomposition, the consensus-based method was not designed from existing optimization methods. Rather, it was directly developed for networked, multi-agent systems. The foundation of this concept is that the consensus equation, a core equation in many multi-agent designs, e.g. [57, 66, 34], can be used to force agreement between different agents. Therefore, the basic idea is for each agent to combine a step in the gradient direction with a step in the direction of consensus.

As the consensus method was developed for the multi-agent scenario, it can immediately be expressed in a distributed fashion as

$$\dot{x}_i = -k_G \frac{\partial f_i}{\partial x}(x_i) - \sum_{j \in \mathcal{N}_i} \alpha_{ij}(x_i - x_j), \tag{28}$$

where α_{ij} is the weighting that agent i associates with the edge of the graph connecting itself to agent j. Assuming equal weighting on all edges, i.e. $\alpha_{ij} = k_P \ \forall \ (v_i, v_j) \in \mathcal{E}$, the consensus based method can be stated for the aggregate state dynamics as:

$$\dot{z} = -k_P \mathbb{L} z - k_G \frac{\partial f}{\partial z}^T. \tag{29}$$

From this expression of the aggregate dynamics, we immediately see that the consensus term is the proportional term given in (13).

We do not present a proof of this method as it does not add to the development in this paper. For the discrete-time analog to (29), using a diminishing or adaptive step-size rule[3] for determining k_G at each iteration of the optimization would cause the agents to converge to the optimal value. For the continuous case, [45] proves that agents can come arbitrarily close to the optimum by choosing $\frac{k_G}{k_P}$ to be “sufficiently small.”

The diminishing step-size condition has been observed to be a possible deterrent of quick convergence of the algorithm, e.g. [88, 61, 89]. To balance a tradeoff between convergence and optimality, [61] proposed a scheme of changing k_G during execution to get closer to the optimal point. The basic idea is that a constant gain often will result in the state approaching a steady-state value in relatively few steps. Once the state is “close enough” to the steady-state then the gain is changed to zero to allow the agents to reach

[3]Section 2.3 is the only section which consideres the gain k_G to be time-varying. Throughout the rest of the paper, all gains (k_G, k_I, and k_P) are considered constant.

consensus. They prove that the longer the agents wait to switch to the zero gain, the closer they will come to the optimal value, but will suffer in convergence rate.

2.3.2 Consensus Method and Constrained Optimization

We now examine this tradeoff further in terms of the underlying constrained optimization problem given in Section 2.1.3. This will give insight into the effect of the contribution of the proportional term and the benefit of including an integral term, which is done in Section 2.4.

To perform this analysis, assume that $\bar{z}$ is the steady-state result of executing (29) as $t \longrightarrow \infty$. Such a $\bar{z}$ is known to exist due to the analysis in [45]. At $\bar{z}$, (29) will give

$$\dot{z} = 0 = -k_P \mathbb{L}\bar{z} - k_G \frac{\partial f}{\partial z}^T (\bar{z}) \Rightarrow 0 = \frac{k_P}{k_G}\mathbb{L}\bar{z} + \frac{\partial f}{\partial z}^T (\bar{z}). \tag{30}$$

Using the fact that $\mathbb{L} = \mathbb{L}^T = \mathbb{D}\mathbb{D}^T$, (30) can be expressed as $0 = \frac{\partial f}{\partial z} + \frac{k_P}{k_G}\bar{z}^T\mathbb{D}\mathbb{D}^T$. Now, let $\lambda^T = \frac{k_P}{k_G}\bar{z}^T\mathbb{D}$ and recall that $\frac{\partial h}{\partial z} = \mathbb{D}^T$, where $h(z) = 0$ is the equality constraint. This gives $0 = \frac{\partial f}{\partial z} + \lambda^T \frac{\partial h}{\partial z}$ as in (21). While this satisfies part of the condition for determining an extreme point, $\bar{z}$ will not be optimal as consensus will not be reached, resulting in the constraints not being met, [45].

As discussed in Section 2.1.2, $\mathbb{L}z$ will always point along lines perpendicular to the constraint set. This means that $\bar{z}$ will be a point where $\frac{\partial f}{\partial z}(\bar{z})$ points along a line perpendicular to the constraint set. Now, let $\bar{z}' = z(t)$ as $t \longrightarrow \infty$ where $\dot{z} = -\mathbb{L}z$ and $z(0) = \bar{z}$. Since $\mathbb{L}z$ points directly to the constraint set, $\bar{z}'$ will be the point of intersection of the constraint set orthogonal to $\bar{z}$. Therefore, if $f(z)$ is such that the gradient will always point directly at the unconstrained optimal point, then the result of the optimization strategy proposed in [61] can converge arbitrarily close to the optimal value. An example of such a convex function is shown in Figure 1.

More important to our discussion is that a constantly weighted consensus term will not have enough control authority to pull the state of the system all of the way to the optimal point. However, it will help to guide the state to, and maintain it on, a line in which the

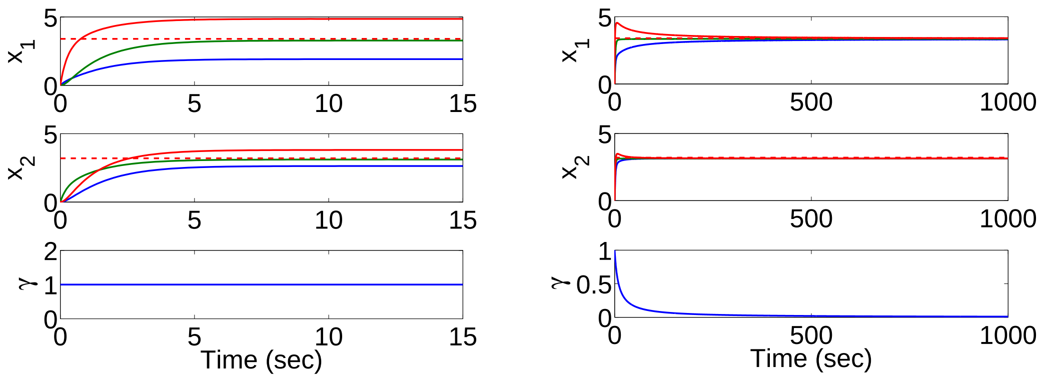


Figure 4: This figure shows the result of optimizing using consensus for the problem given in (27) for both a constant and fading value for k_G on the left and right respectively

only additional control effort need be in the direction of consensus. This further motivates the choice of adding an integral control term.

2.3.3 Example

We continue the example started in Section 2.2 using the consensus-based distributed optimization. Two scenarios are shown for the gain: $k_G = 1$ which violates the diminishing or adaptive gain requirement and $k_G = \frac{1}{1+.1t}$ which satisfies the requirement. The results are shown in Figure 4 and Table 1. The constant gain example exhibits the very desirable attribute of quick convergence, however suffers in performance as the values do not converge and the optimal value is not reached. On the other hand, the fading gain example shows that the optimal values can be achieved, but convergence suffers as expected. Both exhibit the desirable attribute of very little oscillation in the solution, however, the fading gain does show a significant increase in overshoot.

Remark 2. *In presenting examples throughout the remainder of the paper, the results from both a constant and a diminishing gain will be shown. We do this instead of trying to tune the "stopping" criteria given in [61]. The result of a constant gain will emphasize the possible convergence rate and a diminishing gain will emphasize the ability to reach optimality.*

2.4 PI Distributed Optimization

In Sections 2.2 and 2.3, dual decomposition and the consensus method for distributed optimization were introduced and the parallel to integral and proportional control laws was seen. In this section, we show that these two methods can be combined to create a new distributed optimization method which is guaranteed to converge to the collective minimum, much like integral control can be added to proportional control to achieve zero steady-state error with good convergence properties.

This section begins by developing the PI distributed optimization method and proving that it converges to the collective minimum. The relationship to PI control is then discussed and the example of the previous two sections is finished.

2.4.1 PI Distributed Optimization Algorithm

The PI distributed optimization algorithm is formed by noting that the dual-decomposition method discussed in Section 2.2 shares similar structure with the consensus method discussed in Section 2.3. Each has a gradient term along with an additional term added to enforce equality between agents. Dual-decomposition guarantees convergence to the goal, but has an undesirable transient, oscillatory behavior. On the other hand, the consensus method does not converge under constant gains, but has a much more damped transient response. Therefore, we join the two methods in a desire to achieve the benefits of each.

Combining equations (17) and (18) with (29), the aggregate dynamics can be expressed as

$$\begin{aligned} \dot{z} &= -k_G \frac{\partial f}{\partial z}^T - k_P \mathbb{L} z - k_I' \mathbb{D} \mu \\ \dot{\mu} &= k_I' \mathbb{D}^T z. \end{aligned} \tag{31}$$

Similarly, (25) and (26) can be combined with (28) to get a distributed implementation as follows:

$$\dot{x}_i = -k_G \frac{\partial f_i}{\partial x}^T (x_i) - k_P \sum_{j \in \mathcal{N}_i} (x_i - x_j) - k_I' \sum_{j \in \mathcal{N}_i} \mu_i^j. \tag{32}$$

$$\dot{\mu}_i^j = k_I'(x_i - x_j) \tag{33}$$

where we again define μ_i^j as in (26). As in Sections 2.2 and 2.3, *the only information exchange required between agents is the exchange of the state vectors between neighboring agents.*

To show convergence to the collective minimum, we give the following two theorems.

Theorem 3. *Given Assumptions 1, 2, and 3 as well as the dynamics in (31), the saddle point* $(\dot{z}, \dot{\mu}) = (0, 0)$ *is globally asymptotically stable.*

Proof. The same proof can be used as was used in Theorem 1 with two modifications.

1. $H(z) = k_G \frac{\partial^2 f}{\partial z^2} + k_P \mathbb{L}$, but $H(z) \succ 0$ still holds.

2. $\ddot{z} = -k_G \frac{\partial^2 f}{\partial z^2} \dot{z} - k_P \mathbb{L} \dot{z} - k_I' \mathbb{D} \dot{\mu}$ which when $\dot{z} = 0$ still simplifies to $\ddot{z} = -k_I' \mathbb{D} \dot{\mu}$

$\square$

Theorem 4. *Given Assumptions 1, 2, and 3 as well as the dynamics in (31), the saddle point* $(\dot{z}, \dot{\mu}) = (0, 0)$ *corresponds to the collective minimum.*

Proof. The same proof can be used as was used in Theorem 2 by noting for a feasible solution, $\mathbb{L}z = 0$. This will give the same equation for $\dot{z}$ as given in (22). $\square$

The proofs of Theorems 3 and 4 basically show that adding the consensus term does not break the convergence properties of the dual-decomposition method of Section 2.2, but do nothing to speak of the benefit of adding the consensus term. To see the benefit of the consensus term, consider the following problem:

$$\min_z k_G f(z) + \frac{k_P}{2} z^T \mathbb{L} z. \tag{34}$$

$$\text{s.t. } k_I' \mathbb{D}^T z = 0$$

This is the same problem as given in (10), but with the addition of a term proportional to the square of the constraint (recall $\mathbb{D}\mathbb{D}^T = \mathbb{L}$). Adding the square of the constraint is known as the augmented Lagrangian method, which has been shown to add dampening to the dual optimization problem, improving convergence, (see [6] for a discussion and analysis of the augmented Lagrangian).

Following the same method to develop dynamic update laws as in Section 2.2, the following dual optimization problem would be solved:

$$\max_{\mu} \min_{z} \Big(k_G f(z) + k_I' z^T \mathbb{D} \mu + \frac{k_P}{2} z^T \mathbb{L} z \Big), \tag{35}$$

with the resulting dynamics being the same as (31). Thus, adding in a consensus term corresponds to modifying the problem to solve the augmented Lagrangian, producing the desired dampening effect without modifying the guarantee of convergence.

2.4.2 Connections to PI Control

As with the previous two distributed optimization techniques, we note the similarity of this distributed optimization framework with a PI control framework. The Lagrange multiplier, μ, can be expressed in the same form as done in (23). Thus, the following expression for $\dot{z}$ can be obtained:

$$\dot{z}(t) = -k_G \frac{\partial f}{\partial z}^T - k_I \mathbb{L} \int_{t_0}^{t} z(\tau) d\tau - k_P \mathbb{L} z(t). \tag{36}$$

This is the same equation that was derived for a PI control law in Section 2.1.3. We can therefore expect to see properties of PI control such as increased overshoot resulting from decreased dampening of the proportional control, zero steady-state error due to the integral term (which has already been proved), and faster settling time than pure integral control, e.g. [25].

While there exist many distributed optimization techniques, e.g. [68, 90, 45, 88, 89, 85], it is important to note the similarity of the method in this section to that presented in [88] and extended in [89, 26]. While the development of the algorithm in [88] is different than the development in this paper, it can be expressed as using the augmented Lagrangian to solve the following problem:

$$\min_{z} f(z), \tag{37}$$

$$\text{s.t. } \mathbb{L}z = 0$$

where the resulting dynamics can be expressed as

$$\dot{z} = -\frac{\partial f}{\partial z}^T (z(t)) - \mathbb{L}z(t) - \mathbb{L}\mu(t), \tag{38}$$

$$\dot{\mu} = \mathbb{L}z, \tag{39}$$

and now $\mu \in \mathbb{R}^{Nn}$ as opposed to $\mu \in \mathbb{R}^{Mn}$ as before. The only difference between this method and the method we have developed is simply that the constraint is expressed in terms of the graph Laplacian instead of the incidence matrix. This would result in an equation similar to (36), except with an $\mathbb{L}^2$ term instead of an $\mathbb{L}$ term in front of the integral.

While this may seem like a small difference, due to the fact that we have utilized dual-decomposition in the development of the integral term, *we form a PI distributed optimization technique which requires half of the communication* that the technique developed in [88] requires. This can be seen from the fact that the incidence matrix, used in dual-decomposition, allows each agent to update the necessary values of μ using only local information. However, using the Laplacian matrix to express the constraint forms an $\mathbb{L}^2$ term which requires that either each agent knows their neighbors' neighbors' states or each neighbor must additionally communicate μ_i at each optimization step.

2.4.3 Example

We continue the example in (27) using the newly derived dynamics. In Figure 5, it is apparent that the PI optimization is able to achieve zero error while converging quickly and with little oscillation. Furthermore, Table 1 shows that settling time and overshoot are in between the values of pure proportional and pure integral control, as expected. These attributes will be emphasized in the examples in the following sections as more complex problems are presented.

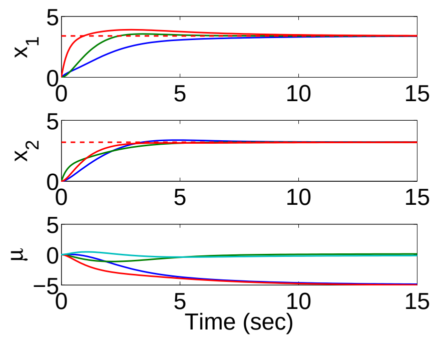


Figure 5: This figure shows the results from the convex optimization example using PI distributed optimization

2.4.4 Scalable Multi-Agent Formulation

Up until this point, we have presented the algorithms in terms of a framework where each agent keeps its own version of the entire state vector as done in previous works, e.g. [45, 88, 61, 85]. This is not necessary if some of the agents' individual costs do not depend upon all of the elements of the parameter vector being optimized. An example of this will be shown at the end of the section where each agent introduces more parameters to be optimized, typical in multi-robot scenarios, e.g. [39, 18, 75, 20]. However, each agents' cost depends solely on the parameters introduced by its neighbors. In such a situation, it is not necessary for each agent to keep track of the entire parameter vector and, in fact, doing so is not scalable to large numbers of agents.

In this section, we address this in a similar fashion to [89] and show that it fits quite naturally into the framework of the previous sections. First, it is shown that even with the reduction of parameters the previous theorems still hold. Then, the reduction of parameters will lead to a slight reformulation of the PI distributed optimization algorithm. Finally, we end this section with an example where drastic improvement in convergence is achieved by reducing the number of variables that each agent must maintain.

2.4.4.1 *Eliminating Unneeded Variables*

When each agent does not have an opinion about a parameter in the parameter vector, the problem can be simplified to eliminate redundancies. Similar to [89], let $I_j = \{i | f_i$ depends on the element $j\}$ be the set of agents which depend on element j with cardinality $N_j = |I_j|$. As agents no longer needs to keep track of the entire vector, the definition of z_j needs to be slightly modified to $z_j \triangleq vec[x_{ij}]_{i \in I_j} \in \mathbb{R}^{N_i}$, a subset of the elements originally contained in z_j. Now, the aggregate vector can be defined as $z = \begin{bmatrix} z_1^T & ... & z_n^T \end{bmatrix}^T \in \mathbb{R}^{N_1 + ... + N_n}$.

Let the induced subgraphs, $\mathcal{G}_i(\mathcal{V}_i, \mathcal{E}_i)$, be defined as $\mathcal{V}_i = \{v_j \in \mathcal{V} | j \in I_i\} \subseteq \mathcal{V}$ and $\mathcal{E}_i = \{(v_i, v_j) \in \mathcal{E} | v_i, v_j \in \mathcal{V}_i\}$. Finally, the following assumption is made to allow for convergence

Assumption 4. *$\mathcal{G}_i$ is connected* $\forall i \in \{1, ..., N\}$.

Note that, given Assumption 3, Assumption 4 is not limiting. If there exists i s.t. $\mathcal{G}_i$ is not connected, one needs only to extend $\mathcal{G}_i$ to contain nodes originally in $\mathcal{G}$ that will connect the different connected components of $\mathcal{G}_i$.

Along this same line of reasoning, we briefly touch upon a topic of study which is out of the scope of this paper, but worth mentioning. There may be simple cases in which choosing $\mathcal{G}_i$ such that it is connected with the smallest number of vertices possible will not result in the fastest convergence to the collective minimum. There has been much work done on the convergence of the consensus equation and the network topology plays a key role in determining the convergence rate [57, 66, 34]. Therefore, to achieve the fastest performance, selection of the sub-graph for each variable could be more complicated than simply choosing the minimally connected sub-graph.

In any case, given $\mathcal{G}_i$, the corresponding incidence matrix, $D_i \in \mathbb{R}^{N_i \times M_i}$, where $M_i = |\mathcal{E}_i|$, and graph Laplacian, $L_i \in \mathbb{R}^{N_i \times N_i}$, can be defined. This allows for the definition of the aggregate matrices $\mathbb{D} \triangleq diag(D_1, ..., D_n)$ and $\mathbb{L} = diag(L_1, ..., L_n)$. These aggregate matrices will continue to exhibit the same properties mentioned in Section 2.1.2 as they can still be expressed as n connected components of a graph. The only difference is that the connected components do not have the same structure. As these properties still hold, Theorems 1 through 4 will also hold using the newly defined augmented matrices and addition of Assumption 4.

2.4.4.2 *Distributed Implementation*

While the aggregate dynamics of the multi-agent system can be expressed without any change, the dynamics executed by each agent will change slightly due to the fact that each variable in the parameter vector will have a different set of agents that are maintaining a version of it. We express the dynamics of a single variable as follows:

$$\dot{x}_{ij} = -k_G \frac{\partial f_i}{\partial x_{ij}} - k_P \sum_{k \in \{\mathcal{N}_i \cap I_j\}} (x_{ij} - x_{kj}) - k_I' \sum_{k \in \{\mathcal{N}_i \cap I_j\}} \mu_{i,j}^k \tag{40}$$

$$\dot{\mu}_{i,j}^k = k_I'(x_{ij} - x_{kj}). \tag{41}$$

Note that the algorithms in Sections 2.2 and 2.3 can be achieved by setting $k_P = 0$ and $k_I' = 0$ respectively. Again, we see that *each agent is able to execute this algorithm using local information and only communicating its version of the parameters being optimized with its neighbors.*

2.4.4.3 *Ring Example*

We now present an example in which scaling down the number of parameters that each agent worries about drastically improves the performance of the system. Consider the "Ring" network depicted in Figure 6 where each agent can communicate with agents to each side. In this example, each agent has a variable that "belongs" to it and it wants to balance having its value be close to its neighbors' value as well as a nominal value. This can be expressed in the form of the following quadratic cost:

$$f_i = (x_{i,i-1} - x_{ii})^2 + (x_{ii} - x_{d_i})^2 + (x_{ii} - x_{i,i+1})^2 \tag{42}$$

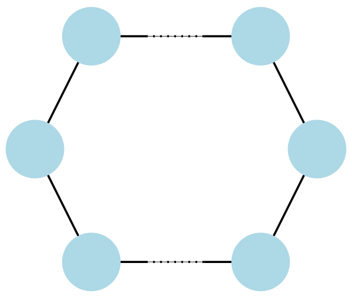


Figure 6: This figure depicts the “Ring” network structure used in Section 2.4.4

where $x_{d_i} = i$ is the desired value.

Note that for the formulation in Sections 2.2, 2.3, and 2.4, each agent would have had to keep track of $N = 20$ variables, corresponding to the aggregate state vector having 400 elements. However, this is greatly reduced by following the formulation in this section. Each agent will only need to keep track of 3 variables with a total of 60 variables in the aggregate state vector.

The results of both representations of the state can be seen in Figure 7 and Tables 2 and 3. Significant improvement can be seen across the board in terms of settling time for reducing the number of variables. Moreover, the overshoot is drastically improved for both the I and PI distributed optimization methods. Related to overshoot, it is seen in Figure 7 that the oscillation is drastically reduced for dual-decomposition.

One final observation about the performance of the PI distributed optimization technique is noteworthy. This example demonstrates the performance of the system when a larger number of variables is in question. We see in Table 2 that the PI distributed optimization significantly outperforms the other methods in terms of convergence. There is a drastic improvement over the dual-decomposition method in terms of overshoot and oscillation as well as an improvement over the consensus method in terms of steady-state error.

Again, we emphasize that this is an extreme example meant to demonstrate the possible utility of reducing the number of variables that each agent deals with. Conclusions should

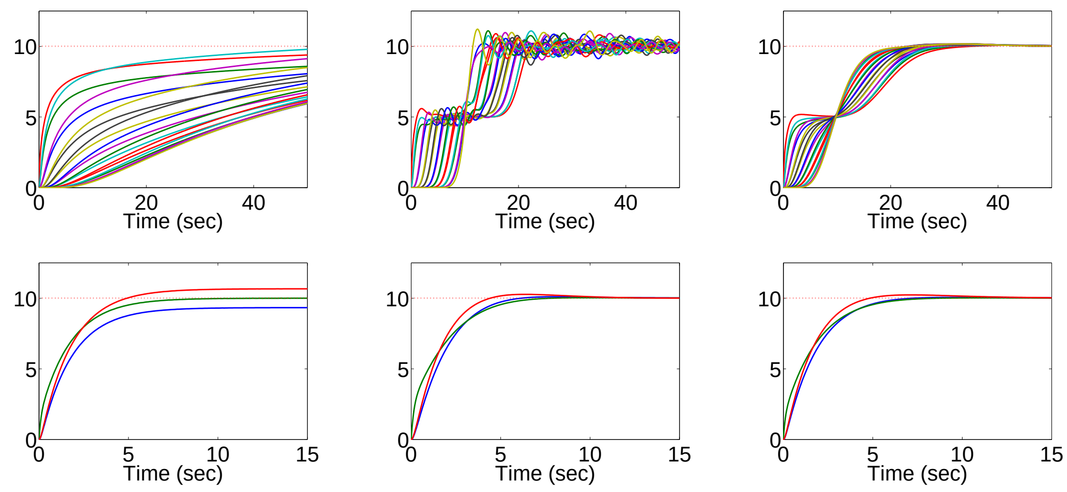


Figure 7: This figures shows the results of applying the formulation of Sections 2.2, 2.3, and 2.4 on the top row and 2.4.4 bottom row to solve the problem in (42). The left, middle, and right images of each row correspond to consensus, dual-decomposition, and PI distributed optimization techniques. The results shown are for variable 10. The solutions in the top row require 20 versions of this variable to converge to the optimal value where the solutions in the bottom row require only 3.

Table 2: The results of performing proportional, integral, and PI distributed optimization with each agent optimizing over the full state vector.

	P: $\gamma = 1$	P: $\gamma = \frac{1}{1+.1t}$	I	PI
M	0.1%	0.12%	37.5%	7.9%
t_{10}	120.8	659.42	115.28	29.78
t_1	226.58	4884.8	542.71	83.02
% error	55.4%	0.92%	0%	0%

not be drawn beyond the notion that this may be beneficial as there may be instances in which scaling back as much as possible would not be beneficial.

2.4.5 PI Distributed Optimization of a Non-Convex Function

In this section, we make one further contribution to PI distributed optimization. Up until this point, we have concerned ourselves solely with the optimization of a convex function, but it may well be the case that the desired function to be optimized is non-convex and a local minimum will suffice. We first show that, under an assumption of local-convexity, the PI distributed optimization method will converge to a local minimum. This is then followed

Table 3: The results of performing proportional, integral, and PI distributed optimization with each agent optimizing over a subset of the state vector.

	P: $\gamma = 1$	P: $\gamma = \frac{1}{1+.1t}$	I	PI
M	0.1%	35.15%	7.12%	4.51%
t_{10}	5.2	82.85	6.12	6.03
t_1	9.47	692.57	12.78	12.33
% error	57.48%	5.3%	0%	0%

with an example of such a minimization.

2.4.5.1 Non-Convex Optimization

While not always true, many non-convex functions are defined such that the cost is strictly-convex on many subsets of the parameter space. We refer to these subsets as local strictly-convex regions. An example of such a function is shown in Figure 8. Such functions are suitable for gradient-based methods where the methods will guide the parameters to a local minimum of the cost. The purpose of this section is to give conditions under which the gradient-based distributed optimization techniques will converge.

Two assumptions are now given:

Assumption 5. *The function being optimized, $\sum_{i=1}^{N} f_i(x)$, has local strictly-convex regions.*

Assumption 6. *There exists a time such that the parameter vector is in a local strictly-convex region of $f(z) = \sum_{i=1}^{N} f_i(x_i)$ while the Lagrange multiplier is simultaneously in the corresponding local concave region of the dual function.*

These assumptions basically tell us that the function must be well-suited to gradient techniques for optimization and that the initial guess for the parameters must be sufficiently good. There may, in fact, be a trade-off between these two criteria, i.e. a function that has smaller regions of local convexity may require a better initial guess for the parameters.

Assumption 6 is feasible for all locally strictly-convex regions of functions satisfying Assumption 5. This comes from the theory of local duality for equality constraints which

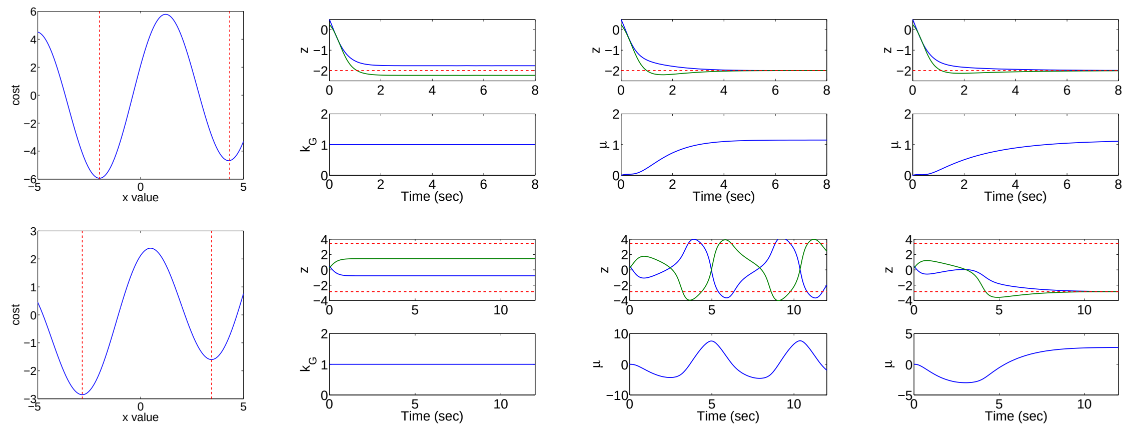


Figure 8: From left to right in each row, this figure shows the cost, consensus, dual-decomposition, and PI distributed optimization methods applied to the problem in (43). The top row corresponds to results when $\theta = \frac{\pi}{4}$ and the bottom row corresponds to $\theta = \frac{3\pi}{4}$. The axis label z corresponds to each agents version of the variable, k_G is the gain on the gradient for the consensus method, and μ is the Lagrange multiplier for the other methods.

guarantees that there is a local maximum in the dual function corresponding to each local minimum in the primal function. Moreover, there is zero duality gap between the local minimum and maximum, e.g. [53]. With this fact and assumptions in mind, we now present a theorem for the convergence of the PI distributed optimization algorithm.

Theorem 5. *Given Assumptions 3, 4, 5, and 6, a system executing the dynamics for the PI distributed optimization algorithm in (31) will converge to a local minimum.*

Proof. The proof of Theorem 3 will hold for local asymptotic stability due to the fact that $H(z) \succ 0$ in a region around the local minimum. Similarly, the proof for Theorem 4 will hold for a local minimum in a strictly-convex region. □

2.4.5.2 *Example*

We present two examples illustrating the ability of the PI distributed optimization framework to come to a local minimum. Both examples consider the case where there are two agents minimizing the summation of the following functions

$$\begin{aligned} f_1(x) &= .1x + 3\sin(x) \\ f_2(x) &= .1x + 3\sin(x+\theta). \end{aligned} \tag{43}$$

While the true minimum would be $x \longrightarrow -\infty$, there exists an abundance of periodic local minimum, to which the optimization could converge.

In the first example we set $\theta = \frac{\pi}{4}$ and show the results in Figure 8 and Table 4. The initial conditions are $\begin{bmatrix} x_1(0) & x_2(0) & \mu(0) \end{bmatrix} = \begin{bmatrix} .5 & .25 & 0 \end{bmatrix}$ and satisfy Assumption 6. The results are very similar with respect to those in the convex examples in that the same trends in oscillation, overshoot, percent error, and settling time are seen.

There is one result, not related to the non-convexity of the problem, which deserves mention. Although the PI optimization has a smaller value for t_{10} than dual-decomposition, the value for t_1 is greater. This shows that absolute convergence to the final value is not guaranteed to be faster in the PI method versus dual-decomposition. This can again be explained with a relation to PI control. Depending on the gains, k_G, k_I, k_P, The proportional term may cause the state to quickly approach a steady-state value. However, in doing so, it may not give as much time for the integral term to have built up the necessary summation in error and therefore the integral term may take longer to build up a large enough value to push the state closer to the desired steady-state.

In the second example we set $\theta = \frac{3\pi}{4}$ with the same initial conditions and show the results in Figure 8 and Table 5. In this case, the initial conditions do not satisfy Assumption 6, but provide some interesting results. The consensus method with constant gain converges to a much worse parameter vector than before. This is due to the fact that the parameters are pulled strongly in different directions as they were initialized close to two separate local minima. However, even more interesting is the result of dual-decomposition as Assumption 6 is never satisfied. In Figure 8, it can be seen that the values oscillate between two different local minima and the Lagrange multiplier is never able to make them converge. In contrast, PI distributed optimization is able to converge due to the increased dampening provided

Table 4: The results of performing proportional, integral, and PI distributed optimization with $\theta_2 = \frac{\pi}{4}$

	P: $\gamma = 1$	P: $\gamma = \frac{1}{1+.1t}$	I	PI
M	0%	10.08%	10.2%	6.57%
t_{10}	1.016	1.89	1.77	1.42
t_1	1.8	223.81	4.01	5.99
% error	10.3%	0%	0%	0%

Table 5: The results of performing proportional, integral, and PI distributed optimization with $\theta_2 = \frac{3\pi}{4}$

Non-convex $\frac{3\pi}{4}$	P: $\gamma = 1$	P: $\gamma = \frac{1}{1+.1t}$	I	PI
M	1.1%	4.97%	-	26.37%
t_{10}	0.72	51.51	-	7.2
t_1	1.34	475.64	-	10.48
% error	62.08%	0%	-	0%

by the proportional term. Thus, while initially Assumption 6 is not satisfied, it quickly becomes satisfied and the parameters converge to a local minimum.

We note that PI distributed optimization *will not guarantee convergence* to a local minimum if Assumption 6 is not initially satisfied. This can be seen by increasing the scalar multiplying the linear term in each function in (43). This effectively reduces the size of the local-convex regions and there is a point in which the PI distributed optimization method will no longer converge. However, it can be expected that the increased dampening, due to the addition of the proportional term, will help in situations such as this where oscillating between different solutions is a potential problem.

2.5 Conclusion

We have developed a new, PI distributed optimization method through the combination of dual decomposition and the consensus method for distributed optimization. This has been done by noting the similarity of the methods when considering the underlying constrained optimization problem. This new method is able to achieve desirable properties from both of the previous methods. Namely, faster convergence and dampening due to the proportional term, originating from the consensus based method, and zero steady-state error from the integral term, originating from dual-decomposition. It was also seen that, under local convexity assumptions, that the PI distributed optimization method is capable of converging to a local minimum in situations when dual-decomposition was not.

CHAPTER III

PROPORTIONAL-INTEGRAL DISTRIBUTED OPTIMIZATION FOR NETWORKED SYSTEMS WITH SWITCHING TOPOLOGIES

In the previous chapter, a PI distributed optimization technique was developed which captures the desirable transient response from consensus approach and the desirable convergence guarantees of the decomposition approach. This PI distributed optimization technique could be applied to provide a framework for agents to collaboratively solve a number of tasks. Consider, for example, the formation control problem depicted in Figure 9. To successfully solve the problem, agents must collaborate to determine the translation, rotation, and scaling of the formation while simultaneously moving into their respective positions.

The developed PI distributed optimization technique will allow for a damped response for the agents' motion, when compared to decomposition methods, with a guarantee of convergence. However, an essential aspect to consider is that while agents are moving, they may move into or out of communication range with different agents. To make the PI distributed optimization technique developed in the previous chapter applicable to a more

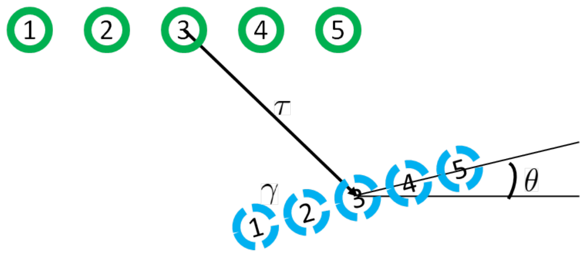


Figure 9: The task of moving into formation can be defined as finding a translation (τ), rotation (θ), and scaling (γ) from the nominal formation, shown in the upper left, to the desired position of the agents,shown in the bottom right.

general setting, this chapter will extend the approach to switching topologies.

This is done by first adding an element of time into the notation presented for multi-agent networked systems in 3.1. Then, using a newly defined incidence matrix, an alternative proof of convergence is given in Section 3.2 for static topologies which is readily extended to dynamic topologies in Section 3.3. However, the formulation of the algorithm will be seen to be undesirable as it will require each agent to maintain the contribution all other agents have ever made to the variables being optimized. Therefore, Section 3.4 will reformulate the algorithm to eliminate this undesirable aspect. In Section 3.5 we return to the formation control example depicted in Figure 9 and conclude the Chapter in Section 3.6.

3.1 Network Multi-agent Systems with Switching Topologies

The communication topology at time t can be represented as an undirected graph, $\mathcal{G}(\mathcal{V}, \mathcal{E}_{i(t)})$, where the node $v_l \in \mathcal{V}$ corresponds to agent l and the edge-set, $\mathcal{E}_{i(t)} \subseteq \mathcal{E}_a = \{\mathcal{V} \times \mathcal{V}\}$ corresponds to agents which can communicate. For sake of simplicity, $\mathcal{E}_{i(t)}$ will be denoted as $\mathcal{E}_i$. Let the index set of all possible graph topologies be denoted by I, such that $\mathcal{E}_i \subseteq \mathcal{E}_a, \forall i \in I$. In a similar fashion, let the index set of all possible connected topologies be denoted as $I_c \subset I$, such that the graph $\mathcal{G}(\mathcal{V}, \mathcal{E}_i)$ is connected $\forall i \in I_c$. The final assumption used to ensure convergence can now be stated:

Assumption 7. *Let the graph at time t be given as $G(\mathcal{V}, \mathcal{E}_{i(t)})$ where $i(t) \in I_c$ $\forall t$. In other words, the graph is always connected.*

We know redefine the incidence matrix to account for edges in the graph being added and removed. Allow the elements, $e_j \in \mathcal{E}_a$ $j \in J = 1, ..., |\mathcal{E}_a|$, to be indexed with the set

J. The incidence matrix, $\mathcal{D}_i \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}_a|}$, is then defined for graph $\mathcal{G}(\mathcal{V},\mathcal{E}_i)$, such that

$$\mathcal{D}_i = [d_{kj}] = \begin{cases} 1 & \text{if } e_j \in \mathcal{E}_i \text{ and } v_k \text{ is the head of} \\ & e_j, j \in J \\ -1 & \text{if } e_j \in \mathcal{E}_i \text{ and } v_k \text{ is the tail of} \\ & e_j, j \in J \\ 0 & \text{otherwise} \end{cases} \tag{44}$$

In other words, column j in $\mathbb{D}_i$ is dedicated to edge $e_j \in \mathcal{E}_a$ and is only non-zero if $e_j \in \mathcal{E}_i$. Denote the incidence matrix as defined in (9) for graph $\mathcal{G}_i$ as D_i.

Note that the new definition of the incidence matrix does not affect the Laplacian matrix. In other words, $L_i = D_i D_i^T = \mathcal{D}_i \mathcal{D}_i^T$. To show this, without loss of generality assume that the only non-zero entries in $\mathcal{D}_i$ are the first columns. $\mathcal{D}_i$ can then be written as $\mathcal{D}_i = [D_i \; 0]$. Formulating L_i we see that $L_i = \mathcal{D}_i \mathcal{D}_i^T = [D_i \; 0] \begin{bmatrix} D_i^T \\ 0 \end{bmatrix} = D_i D_i^T + 0^T 0 = D_i D_i^T$.

As before, we form aggregate matrices as $\mathbb{D}_i \triangleq I_n \otimes \mathcal{D}_i$ and $\mathbb{L}_i \triangleq I_n \otimes L_i$, where I_n is the $n \times n$ identity matrix. Note that, for static topologies, the development in Section 2.1 through 2.4 could be done with the redefined incidence matrix without changing any of the arguments.

3.2 Alternative Proof of Convergence for Static Topologies

Under static topologies, the proofs given for convergence in Theorems 3 and 4 hold, even with the newly defined incidence matrix. However, the Lyapunov function used in the proofs depends on the dynamics of the system which in turn depend on the incidence matrix. As the incidence matrix changes each time the topology changes, the proof of convergence does not easily extend to the switching topology scenario. Therefore, the proof to the following theorem uses a Lyaponov function independent of the dynamics and is extended to dynamic topologies in Section 3.3.

Theorem 6. *Given that $\mathcal{G}(\mathcal{V}, \mathcal{E}_i)$ forms a static, connected graph and Assumptions 1 and 2 hold, the dynamics given in (31) will cause (z, μ) to converge to the optimal values of the dual problem, (z^*, μ^*), defined by*

$$\max_{\mu} \min_{z} (f'(z) + \mu^T \mathbb{D}_i^T z) \tag{45}$$

Proof. We note that the notation and structure of the proof follow very closely the proof presented in [89]. It was shown in Theorem 4 that at equilibrium $\dot{z} = 0, \dot{\mu} = 0$ corresponds to the optimal values, (z^*, μ^*). Define $\tilde{z} = z - z^*$, $\tilde{\mu} = \mu - \mu^*$ and the function $f'(z) = k_G f(z) + \frac{k_P}{2} z^T \mathbb{L}_i z$. Note that due to Assumption 1 and the fact that $\mathbb{L}_i \succeq 0 \Rightarrow z^T \mathbb{L}_i z$ is convex, $f'(z)$ is strictly convex, (see [5] for properties of convex functions). The dynamics for the newly introduced variables, $(\tilde{z}, \tilde{\mu})$, can be expressed as

$$\begin{aligned}
\dot{\tilde{z}} &= \dot{z} - \dot{z}^* \\
&= -\frac{\partial f'}{\partial z}^T(z) - k_I' \mathbb{D}_i \mu + \frac{\partial f'}{\partial z}^T(z^*) + k_I' \mathbb{D}_i \mu^* \\
&= -\frac{\partial f'}{\partial z}^T(z) + \frac{\partial f'}{\partial z}^T(z^*) - k_I' \mathbb{D}_i \tilde{\mu} \\
\dot{\tilde{\mu}} &= \dot{\mu} - \dot{\mu}^* = k_I' \mathbb{D}_i^T z - k_I' \mathbb{D}_i^T z^* = k_I' \mathbb{D}_i^T \tilde{z}.
\end{aligned} \tag{46}$$

Now, consider the candidate Lyapunov function

$$V(\tilde{z}, \tilde{\mu}) = \frac{1}{2} \tilde{z}^T \tilde{z} + \frac{1}{2} \tilde{\mu}^T \tilde{\mu}, \tag{47}$$

which allows the time derivative to be written as

$$\begin{aligned}
\dot{V}(\tilde{z}, \tilde{\mu}) &= \tilde{z}^T \dot{\tilde{z}} + \tilde{\mu}^T \dot{\tilde{\mu}} \\
&= \tilde{z}^T \Big(\frac{\partial f'}{\partial z}^T(z) + \frac{\partial f'}{\partial z}^T(z^*) - k_I' \mathbb{D}_i \tilde{\mu} \Big) \\
&+ k_I' \tilde{\mu}^T \mathbb{D}_i^T \tilde{z} \\
&= -\tilde{z}^T \frac{\partial f'}{\partial z}^T(z) + \tilde{z}^T \frac{\partial f'}{\partial z}^T(z^*).
\end{aligned} \tag{48}$$

To examine $\dot{V}$, we pull upon the global under-estimator property of convex functions, e.g. [89, 5], which states:

$$g(y) \geq g(x) + \frac{\partial g}{\partial x}^T(x)(y - x) \; \forall x, y, \tag{49}$$

with strict inequality when g is strictly convex. From the global under-estimator property, the time derivative of the Lyapunov function can be written as:

$$\begin{aligned}\dot{V}(\tilde{z},\tilde{\mu}) &= -\frac{\partial f'}{\partial z}(z)\tilde{z} + \frac{\partial f'}{\partial z}(z^*)\tilde{z} \\ &< f'(z^*) - f'(z) + f'(z) - f'(z^*) \\ &= 0\end{aligned} \tag{50}$$

So, in other words $V(\tilde{z},\tilde{\mu})$ is always non-increasing. LaSalle's invariance theorem can then be invoked [40]. Denote the smallest invariant set, $V_0 = \{(\tilde{z},\tilde{\mu})|\dot{V} = 0\} = \{(0,\mu), \forall\mu\}$. In V_0, $z = z* \Rightarrow \dot{z} = \ddot{z} = 0$. Note that $\ddot{z} = -k_G \frac{\partial f}{\partial z}^T \dot{z} - k_P \mathbb{L}_i \dot{z} - k_I' \mathbb{D}_i \dot{\mu} = -k_I \mathbb{D}_i \mathbb{D}_i^T z = -k_I \mathbb{L}_i z = 0$ which implies $z = \alpha \otimes \mathbf{1}$ where $\alpha \in \mathbb{R}^n$ and $\dot{\mu} = \mathbb{D}_i^T z = 0$. Therefore, the control law converges.

□

3.3 Extending Convergence for Dynamic Topologies

The key factor that allows for a convergence proof for dynamic topologies is the redefinition of the incidence matrix in (44). It allows μ to remain the same size and be continuous across switch times, in turn allowing V to be continuous across switch times. This is due to the fact that the dimension and meaning of the elements of μ are directly dependent upon the dimension and ordering of the columns of the incidence matrix, as is evident in (31). Since (44) has a specific column dedicated to each possible edge, the elements in μ will always correspond to the integral of the error across the corresponding edge and the dimension of μ will never change, neither of which is true using the typical definition of the incidence matrix.

One final assumption concerning dwell time is required before giving a theroem about convergence. It is adapted from Assumption 3 in [31] to fit the multi-agent scenario:

Assumption 8. *There exists $\tau > 0$ such that for every $T \geq 0$ a positive integer i can be found for which $t_{i+1} - \tau \geq t_i \geq T$, where t_i denotes the i^{th} switch time. In other words,* *the system persistently encounters intervals of length at least $\tau > 0$ where the network topology remains unchanged.*

Theorem 7. *Given that Assumptions 1, 2, 7, and 8 hold, the dynamics given in (31) will cause (z, μ) to converge to the optimal values, (z^*, μ^*), defined by (45).*

Proof. To show convergence, we invoke the LaSalle invariance principle for hybrid systems stated in Theorem 7 of [31]. To do so, the system must satisfy four conditions. Theorem 6 satisfies the first two conditions which require each set of dynamics for the switched system (corresponding to different network topologies in our case) to have a weak Lyapunov function which can be shown to converge to the equilibrium. The third condition concerns dwell time and is satisfied by Assumption 8. The final condition concerns non-increasing values for the Lyapunov functions across switching, which is trivially satisfied as all topologies have a common Lyapunov function. □

3.4 Index-free PI Distributed Optimization

While the previous section proved that the dynamic update law given in (31) will converge to the optimal value, it is important to note that the individual dynamics given in (32) form an undesirable solution. The reason being is that the dynamic update law requires each agent to "remember" the individual contribution that every other agent has made to the integral of the error. To create an index free solution, we take a step back and evaluate the problem being solved in (10).

The structure of (10) is nothing more than a convex optimization with a linear constraint. It is well know, e.g. [53], that such a problem will have a solution satisfy (21). In taking a closer look at (21), the true value needing to be solved for is a vector which "offsets" the gradient, $\frac{\partial f}{\partial z}$, at the optimal value, as shown in Figure 10. In other words,

$$0 = \frac{\partial f}{\partial z}(z^*) + \nu^T, \tag{51}$$

where in actuality $\nu^T = \lambda^{*T} \frac{\partial h}{\partial z}$.

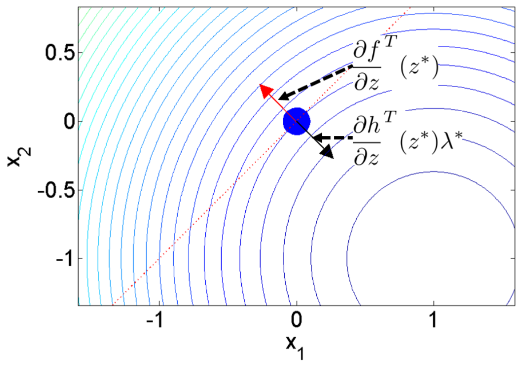


Figure 10: This figure shows the optimal point to the cost $f(z) = (x_{11} - 1)^2 + (x_{21} + 1)^2$ with constraint $x_{11} = x_{21}$. The unconstrained gradient, $\frac{\partial f}{\partial z}^T(z^*)$, is balanced by the vector $\frac{\partial h}{\partial z}^T(z^*)\lambda^*$

In some sense, the distributed optimization algorithm consists of the agents working together to share information in order to collectively learn the value of ν. Thus, this leads to the idea of re-formulating the distributed optimization algorithm in terms of ν as follows:

$$\dot{z} = -k_G \frac{\partial f}{\partial z}^T - k_P \mathbb{L}_i z - \nu \tag{52}$$

where the dynamics for ν can be written as

$$\dot{\nu} = k_I \mathbb{L}_i z. \tag{53}$$

In the same manner as done in (32), the aggregate dynamics can be split up between the agents in a distributed fashion as follows:

$$\begin{aligned} \dot{x}_i &= -k_G \frac{\partial f_i}{\partial x}^T(x_i) - k_P \sum_{j \in \mathcal{N}_i} (x_i - x_j) - \nu_i \\ \dot{\nu}_i &= k_I \sum_{j \in \mathcal{N}_i} (x_i - x_j). \end{aligned} \tag{54}$$

It is important to note that we have removed the agent indexing present in (32) *In (54), agent i no longer needs to keep track of agent j's contribution, rather agent i need only keep track of the aggregate contribution to the error by its neighbors.* A theorem is now given about the convergence of the newly formed dynamic update law:

Theorem 8. *Given that Assumptions 1, 2, 7, and 8 hold, the dynamics given in (53) and (54) will cause z to converge to the optimal value, z^*, defined by*

$$\min_z f(z)$$
$$\text{s.t. } x_i = x_j \forall i, j \in [1, ..., N]$$

Proof. The proof of the theorem hinges upon the fact that the state dynamics, $\dot{z}(t)$, for the aggregate state and $\dot{x}_i(t)$ for individual agents remain the same. Thus, given the same initial conditions and uniqueness of solutions the states will remain unchanged under the new dynamics.

Previously we had

$$\begin{aligned}\dot{\mu} &= k_I'\mathbb{D}_i^T z \\ \dot{z} &= -k_G\frac{\partial f}{\partial z}^T - k_P\mathbb{L}_i z - k_I'\mathbb{D}_i\mu \\ &= -k_G\frac{\partial f}{\partial z}^T - k_P\mathbb{L}_i z - k_I'\mathbb{D}_i\int_0^t k_I'\mathbb{D}_i^T z(s)ds \\ &= -k_G\frac{\partial f}{\partial z}^T - k_P\mathbb{L}_i z - k_I\mathbb{L}_i\int_0^t z(s)ds\end{aligned} \tag{55}$$

and corresponding agent state dynamics

$$\begin{aligned}\dot{\mu}_i^j =& k_I'(x_i - x_j) \\ \dot{x}_i =& -k_G\frac{\partial f_i}{\partial x}^T(x_i) - k_P\sum_{j\in\mathcal{N}_i}(x_i - x_j) - k_I'\sum_{j\in\mathcal{N}_i}\mu_i^j \\ =& -k_G\frac{\partial f_i}{\partial x}^T(x_i) - k_P\sum_{j\in\mathcal{N}_i}(x_i - x_j) - \\ & k_I\sum_{j\in\mathcal{N}_i}\int_0^t(x_i(s) - x_j(s))ds.\end{aligned} \tag{56}$$

The newly formed dynamics can be stated in a similar fashion:

$$\begin{aligned}\dot{\nu} &= k_I\mathbb{L}_i z \\ \dot{z} &= -k_G\frac{\partial f}{\partial z}^T - k_P\mathbb{L}_i z - \nu \\ &= -k_G\frac{\partial f}{\partial z}^T - k_P\mathbb{L}_i z - \int_0^t k_I\mathbb{L}_i z(s)ds \\ &= -k_G\frac{\partial f}{\partial z}^T - k_P\mathbb{L}_i z - k_I\mathbb{L}_i\int_0^t z(s)ds\end{aligned} \tag{57}$$

which is the same as (55). Similarly, the individual agent states can be written as

$$\begin{aligned}\dot{\nu}_i =& k_I \sum_{j\in\mathcal{N}_i}(x_i - x_j)\\ \dot{x}_i =& - k_G \frac{\partial f_i}{\partial x}^T(x_i) - k_P \sum_{j\in\mathcal{N}_i}(x_i - x_j) - \nu_i\\ =& - k_G \frac{\partial f_i}{\partial x}^T(x_i) - k_P \sum_{j\in\mathcal{N}_i}(x_i - x_j) -\\ & k_I \sum_{j\in\mathcal{N}_i}\int_0^t (x_i(s) - x_j(s))ds\end{aligned} \tag{58}$$

which is the same as (56). Therefore, because the dynamics for z remain unchanged, z will converge to z^*. Also, note that because z converges to z^*, $\dot{\nu} \longrightarrow 0$ from the fact that $\mathbb{L}_i z^* = 0$. $\square$

One final note worth making is that the arguments made in Section 2.4.4 for making the distributed optimization problem scalable still hold. As the arguments are the same, we simply state the modified dynamics for agent i's version of variable j as:

$$\begin{aligned}\dot{x}_{ij} &= -k_G \frac{\partial f_i}{\partial x_{ij}} - k_P \sum_{k\in\{\mathcal{N}_i\cap I_j\}}(x_{ij} - x_{kj}) - \nu_{i,j}\\ \dot{\nu}_{i,j} &= k_I \sum_{k\in\{\mathcal{N}_i\cap I_j\}}(x_{ij} - x_{kj}).\end{aligned} \tag{59}$$

Basically, this results in each variable only being maintained and updated by the agents which actually have an opinion about the variable.

3.5 Example: Formation Control

To demonstrate the ability for PI distributed optimization to achieve a collective objective utilizing local information, this section introduces an example of formation control. The method for formation control in this section is based upon a relative state formulation, e.g. [57]. The basic idea being that a formation control problem can be defined by a nominal position for each agent, $y_i \in \mathbb{R}^2$. The agents must come to an agreement upon a translation,

$\tau \in \mathbb{R}^2$, from the nominal position as well as a possible rotation, $\theta \in \mathbb{R}$, about the nominal origin.

While [57] then introduces methods based on feedback control on relative displacements between neighboring agents in the formation, we show that distributed optimization can be utilized to solve for the various parameters. Therefore, agents can choose a displacement, rotation, as well as a scaling ($\gamma \in \mathbb{R}^+$) (as depicted in Figure 9) using solely information available to each agent in the network.

To choose the parameters, $x = [\tau^T, \theta, \gamma]^T$, the agents perform PI distributed optimization. The cost assigned to each agent takes the form

$$f_i(x_i(t)) = \frac{1}{2}||q_i(t) - q_{d_i}(t)||^2 + k(\gamma_i(t) - 1)^2 \tag{60}$$

where $q_i(t) \in \mathbb{R}^2$ is agent i's position at time t, k is a weight on the scaling, and $q_{d_i}(t) \in \mathbb{R}^2$ is the desired position of agent i. Assuming that the nominal formation is defined with the center at the origin, $q_{d_i}(t)$ can be expressed as

$$q_{d_i}(t) = R(\theta_i(t))\gamma_i(t)y_i + \tau_i(t), \tag{61}$$

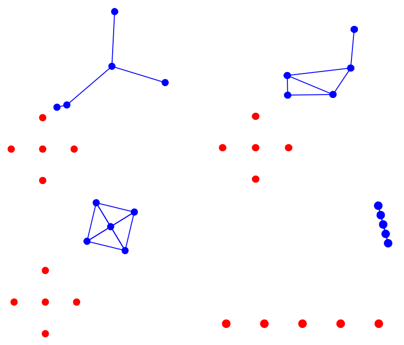
where $R \in \mathbb{R}^{2\times 2}$ is a rotation matrix. Each variable in (60) and (61) is written as a function of time to emphasize that the variables are continually being updated.

The cost defined in (60) has two terms to guide the selection of the parameters. The first term penalizes the distance between the current position and desired position. The second term penalizes deviation from unit scaling where k is only non-zero for the final example where proper scaling is important to see the spelling of the word. Note that (60) is only locally convex, so agents will converge to some local minima.

While agents are optimizing they are also moving towards their respective desired position. Using integrator dynamics, $\dot{q}_i(t) = u_i(t)$, for each agent, the feedback law

$$u_i(t) = q_{d_i}(t) - q_i(t), \tag{62}$$

is used to move towards the desired position. It is also assumed that the underlying graph



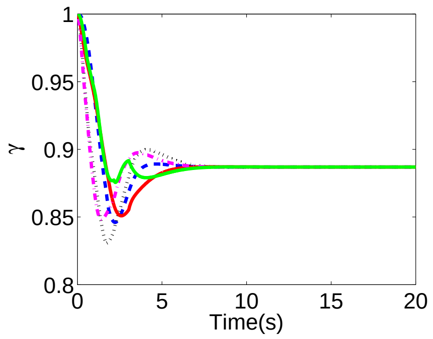


Figure 11: On the top left is shown snapshots of the agents while converging to the diamond formation. On the bottom left is shown the resulting diamond and line formations. The lines between agents show the communication topology and the bottom left of each figure is shown the nominal configuration. On the right is shown each agents' version of γ while converging to the diamond formation. Other variables are not shown as this plot is indicative of the convergence characteristics of the variables in each simulation.

Table 6: This table shows the resulting parameters as well as the average and standard deviation of distances for each simulation. The nominal distance, d_N, refers to the distance from agent's starting position to their nominal position. The travel distance, d_T, is how far the agents actually traveled to reach formation.

	τ	γ	θ	d_N Ave	d_N Std	d_T Ave	d_T Std
Line	(2.12, 2.60)	0.25	4.97	4.03	0.98	1.28	0.67
Diamond	(2.09, 2.39)	0.89	2.12	3.35	2.11	1.49	0.57
GRITS	(1.44, 2.27)	0.98	-1.53	3.12	1.46	2.07	1.00

topology is a δ-disk graph, e.g. [57], where agents i and j are only able to communicate at time t if $\|q_i(t) - q_j(t)\| \leq \delta$.

Three formations are shown to demonstrate the ability for the agents to come to an agreement by minimizing the collective cost using PI distributed optimization. The first two examples are shown in Figure 11 where agents form a diamond and a line formation. To demonstrate the ability to specify arbitrary formations, the third formation has the agents spell out GRITS (the acronym for Georgia Robotics and Intelligent Systems), as shown in Figure 12.

A comparison between the nominal formation and the optimized formation of the distance each agent was required to travel is shown in Table 6. All three examples show a

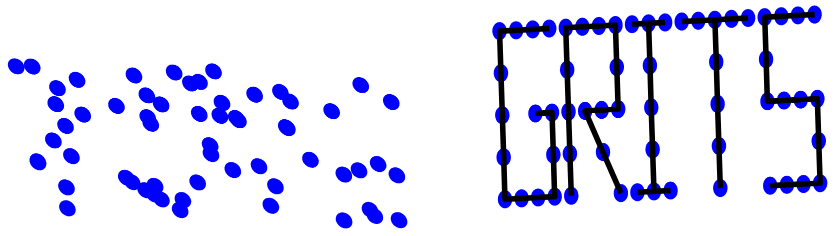


Figure 12: This figure shows 60 agents assigned to spell out the word ‘GRITS’. On the left is shown the initial positions and on the right is the final position.

significant decrease in both the average distance and standard deviation of the distance that each agent was required to travel.

3.6 Conclusion

We have extended the PI distributed optimization method presented in Chapter 2 to account for dynamic topologies. This has been accomplished by redefining the incidence matrix to have a column dedicated to each edge, making it possible to define a Lyapunov function that is continuous across switching topologies. However, the adjustment of the incidence matrix makes the actual implementation undesirable as it requires agents to remember the contribution every other agent has ever made to its error. By re-examining the underlying constrained optimization problem, it was shown that it was possible to reformulate the algorithm so agents solely keep track of the aggregate contribution of their neighbors.

We examined an example of formation control to demonstrate the ability of PI distributed optimization to cope with changing communication topologies while maintaining convergence properties. Agents simultaneously moved and optimized and were able to come to agreement on several parameters in order to determine where the formation would end up. On average each agent traveled less than they would have had to in order to get to the nominal position and the distance traveled by one agent was much closer to that traveled by another.

CHAPTER IV

BEHAVIOR-BASED MPC

We now shift our focus from distributed optimization to develop behavior-based MPC. It is logical that for the proposed method to work well for the motion control of multiple-agents, it must work well for the control of a single agent's motion. Thus, this chapter and the next focus on the development of the behavior-based MPC method for a single agent. The methods and concepts introduced in these chapters are then used in Chapters 6 and 7 for the development of a multi-agent behavior-based MPC framework.

As a motivating example for a behavior-based approach to MPC, consider the problem of having a non-holonomic robot settle to a circular orbit. It is entirely imaginable that a cost could be designed to allow a typical MPC algorithm to find the optimal trajectory at each time instant to allow an agent to fall into orbit. However, it is not always necessary for an optimization framework to reinvent the wheel. Control laws exist which can already produce desirable trajectories for the robot. Instead of having the optimization framework come up with an entire trajectory of control inputs, it can tweak a few parameters to get the desired result.

An example of such a control law for orbiting is shown in Figure 13. To come into orbit, the robot can follow a circular limit cycle, as developed in [63, 4, 41]. The convergence rate of the limit cycle can then be tuned online. When the robot is far away from the limit cycle, it is beneficial to head straight towards the limit cycle. When the robot comes close, it can switch to less aggressive parameters which allow it to converge nicely and avoid oscillations. Illustrated in Figure 13 is the depiction of both the aggressive and smooth vector fields, along with the results achieved by using a behavior-based MPC approach to tune the parameters on-line.

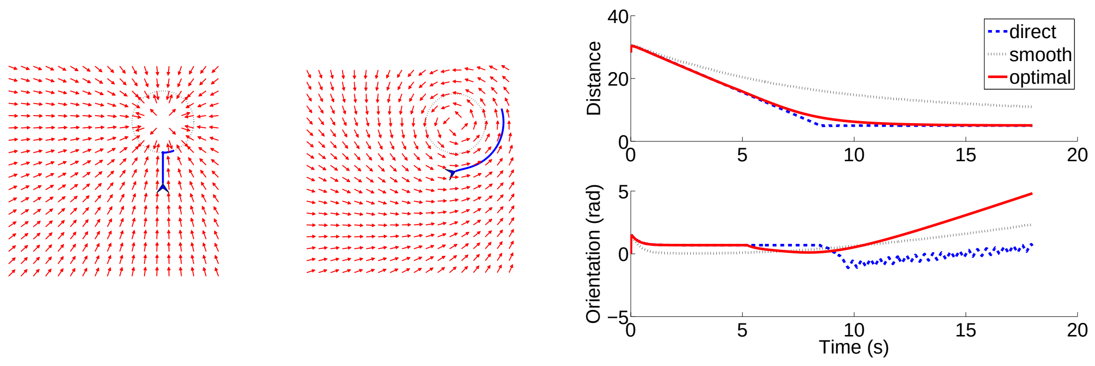


Figure 13: The images show the result of different gains on an orbiting vector field. The left image shows the result of gains that move the robot directly towards the orbit and the middle shows the result of gains that very smoothly transition into orbit. The right image shows the resulting distance and orientation of starting at the same point and executing the direct and smooth vector fields as well as the result of adapting the vector field using behavior-based MPC.

This chapter begins by detailing the behavior-based MPC framework for the general setting in Section 4.1. The example in Figure 13 is then expounded upon in Section 4.2. A more complex example detailing the use of multiple behaviors being used in series to navigate through an environment is then presented in Section 4.3. The method is applied to an inverted pendulum robot which must maintain balance while navigating. The chapter ends with some concluding remarks in Section 4.4.

4.1 Behavior-based MPC Formulation

As outlined in Sections 1.2.2 and 1.3, applying MPC to a robotic system may be difficult as MPC can be computational intensive. This section presents a framework which adds a level of abstraction by introducing parameterized feedback laws to generate the robot's state trajectory. This allows for the exchange of a possibly computationally burdensome optimization problem for a problem involving the optimization of a few parameters, less than ten in each of the examples in the following sections. This section introduces the MPC formulation and then gives the first order necessary conditions for optimality which can be used in gradient strategies to find the parameters at each time step.

4.1.1 MPC Framework

To reduce computational complexity inherent to MPC, tunable feedback control laws can be utilized to generate state trajectories and the parameters can be optimized to achieve the desired result. Moreover, in order to accomplish a desired task it may be desirable to have the robot execute a string of such control laws. This string can be written as $(\kappa_0, \tau_0), (\kappa_1, \tau_1), ..., (\kappa_N, \tau_N)$, where τ_i indicates the time when the robot will switch from executing the control law κ_{i-1} to the control law κ_i. This allows the robot dynamics to be expressed as[1]

$$\dot{x} = f\big(x(t;t_0), \kappa_i(x(t;t_0), \theta_i)\big) \text{ for } \tau_i \le t < \tau_{i+1}, \tag{63}$$

which we simplify as

$$\dot{x} = f_i(x(t;t_0), \theta_i) \text{ for } \tau_i \le t < \tau_{i+1}. \tag{64}$$

To choose both the parameters of each feedback law as well as the time instances to switch between feedback laws, we build upon results from switch time optimization (e.g., [22, 55]). Also, a key point to note is that the environment is not completely known at the time of optimization, denoted as t_0. To explicitly represent this fact, the known environmental data is denoted as, $O(t_0)$, is included as a term in the instantaneous cost. The cost to be minimized is written as:

$$J(\tau, \theta) = \int_{t_0}^{t_0+\Delta} L(x(t;t_0), \theta, O(t_0))dt + \Psi((x(\tau_{N+1};t_0), \theta) \tag{65}$$

$$\text{s.t. } \dot{x} = f_i(x(t;t_0), \theta_i) \text{ for } \tau_i \le t < \tau_{i+1},$$

where $\tau = [\tau_0, ..., \tau_{N+1}]^T$, $\theta = [\theta_1^T, ..., \theta_N^T]^T$, and Δ denotes the time horizon of optimization. Note that we actually do not optimize with respect to the first and last elements of τ, rather we fix them as $\tau_0 = t_0$ and $\tau_{N+1} = t_0 + \Delta$. To denote that the parameters only enter

[1]We again remind the reader that we change the notation to reflect the the notation present in much of the literature. In Chapters 2 and 3 where distributed optimization was discussed, x represented a parameter vector and f represented a cost. In the remainder of the work, x represents a state, f represents dynamics, J, L, and Ψ represent costs, and θ is used to represent a parameter vector.

the cost for the time period over which they are used, the instantaneous cost can be written as:

$$L\big(x(t;t_0),\theta,Ofree(t_0)\big)=L_i\big(x(t;t_0),\theta_i,B_{free}(t_0)\big) \text{ for } \tau_i\le t<\tau_{i+1}.$$

By formulating this cost, we can define our MPC strategy as in Algorithm 1. Step 1 of the Multi-Modal Parameterized MPC algorithm states the ideal case where (65) would be minimized. However, we have found (and a similar conclusion was reached in [32]) that taking a small number of gradient steps is sufficient; further reducing the computational burden on the robot. As a further matter of implementation, the optimization can not be done instantaneously. Step 2 explicitly allows a time period of $\delta_{execute}$ seconds to allow for an appropriate amount of time to be allocated to the optimization step.

Algorithm 1 Multi-Modal Parameterized MPC

```
1. Minimize (65) with respect to the parameters, $\theta$, and the switch times, $\tau$.
2. Apply the feedback laws for a period of $\delta_{execute}$ seconds.
3. Repeat.
```

A further note must be made on the optimization step of the algorithm. We consider each parameter to be optimized to have upper and lower bounds to allow for stability guarantees. As the resulting Khun-Tucker conditions are trivial (see [17] for an example of such Khun-Tucker conditions), these limits come with little computational cost and will be ignored in the derivation of the gradients. The conditions basically say that if the update of a gradient step leaves the parameter outside the bounds then the parameter should be set to the closest limit.

4.1.2 First Order Optimality Conditions

In order to minimize (65) with respect to the desired variables, the first order necessary conditions for optimality are now presented. These conditions can be used with gradient descent methods for optimization (see, for example [5]). In Section 4.2 and 4.3, these gradients will be used with an Armijo step size to allow for quick descent (see [70] for a detailed analysis of the Armijo step size). This will allow for favorable results to be achieved with relatively few gradient steps performed by the robot.

Theorem 9. *The first order necessary conditions of optimality of (65) with respect to the switch times, τ_i, and the parameter vectors, θ_i, are given by*

$$\frac{\partial J}{\partial \tau_i} = \Big(L_{i-1} - L_i + \lambda^T(f_{i-1} - f_i)\Big) = 0, \tag{66}$$

$$\frac{\partial J}{\partial \theta_i} = \xi_i(\tau_i) = 0, \tag{67}$$

where

$$\dot{\lambda} = -\frac{\partial L_i}{\partial x}^T - \frac{\partial f}{\partial x}^T \lambda, \textit{ for } \tau_i \le t < \tau_{i+1},\ i = 0, ..., N \tag{68}$$

$$\lambda(\tau_{N+1}) = \frac{\partial \Psi}{\partial x}(x(\tau_{N+1}))$$

$$\dot{\xi}_i = -\frac{\partial L_i}{\partial \theta_i}^T - \frac{\partial f}{\partial \theta_i}^T \lambda,\ \xi_i(\tau_{i+1}) = \frac{\partial \Psi}{\partial \theta_i} \tag{69}$$

Remark 3. *Due to the fact that the dynamics of the state, x, do not depend on the costates, λ and ξ_i, the gradients for all of variables can be calculated by simulating the state forward in time and then simulating the costates backward in time. This alleviates the main difficulty of solving a two-point boundary value problem where the state also depends on the costate, making the step of forward simulation difficult.*

Proof. The proof of Theorem 9 follows standard variational methods similar to those used for other switch time optimization problems, e.g., [22, 55]. Note that for a concise development, we remove all of the time indexing on the variables inside the integrals and use a single time indexing for the terminal cost. We first augment (65) with the dynamics:

$$\hat{J}(\tau, \theta) = \sum_{i=0}^{N} \int_{\tau_i}^{\tau_{i+1}} \Big(L_i(x, \theta_i, O(t_0)) + \lambda^T(f_i(x, \theta_i) - \dot{x})\Big)dt + \Psi(x(\tau_{N+1}), \theta) \tag{70}$$

Now, the switch times and parameter vectors are varied as $\tau \rightarrow \tau + \epsilon v$ and $\theta \rightarrow \theta + \epsilon\gamma$ which causes the state to vary as $x \rightarrow x + \epsilon\eta$. Including this variation as well as separating the integral term into three parts we can write

$$\hat{J}(\tau + \epsilon v, \theta + \epsilon\gamma) = \sum_{i=0}^{N} \Bigg[\int_{\tau_i}^{\tau_{i+1}} \Big(L_i(...) + \lambda^T (f_i(...) - \dot{x} - \epsilon\dot{\eta}) \Big) dt + \int_{\tau_{i+1}}^{\tau_{i+1}+\epsilon v_{i+1}} \Big(L_i(...) + \lambda^T (f_i(...) - \dot{x} - \epsilon\dot{\eta}) \Big) dt - \int_{\tau_i}^{\tau_i+\epsilon v_i} \Big(L_i(...) + \lambda^T (f_i(...) - \dot{x} - \epsilon\dot{\eta}) \Big) dt \Bigg] + \Psi(x(\tau_{N+1}) + \epsilon\eta(\tau_{N+1}), \theta + \epsilon\gamma) \tag{71}$$

where $(...) = (x + \epsilon\eta, \theta_i + \epsilon\gamma_i)$. Note that without the variation, the cost could have been written as

$$\hat{J}(\tau, \theta) = \sum_{i=0}^{N} \Bigg[\int_{\tau_i}^{\tau_{i+1}} \Big(L_i(x, \theta_i) + \lambda^T (f_i(x, \theta_i) - \dot{x}) \Big) dt + \int_{\tau_{i+1}}^{\tau_{i+1}+\epsilon v_{i+1}} \Big(L_{i+1}(x, \theta_{i+1}) + \lambda^T (f_{i+1}(x, \theta_{i+1}) - \dot{x}) \Big) dt - \int_{\tau_i}^{\tau_i+\epsilon v_i} \Big(L_i(x, \theta_i) + \lambda^T (f_i(x, \theta_i) - \dot{x}) \Big) dt \Bigg] + \Psi(x(\tau_{N+1}), \theta) \tag{72}$$

Now, subtract $\hat{J}(\tau + \epsilon v, \theta - \epsilon\gamma) - \hat{J}(\tau, \theta)$, take the Taylor expansion, invoke the mean value theorem on integrals where ϵ appears in the limit of integration, use integration by parts on the $\lambda^T\dot{\eta}$ term, and simplify algebraically to write

$$\frac{1}{\epsilon}\Big(\hat{J}(\tau + \epsilon v, \theta - \epsilon\gamma) - \hat{J}(\tau, \theta)\Big) = \sum_{i=0}^{N} \Bigg[\int_{\tau_i}^{\tau_{i+1}} \Big(\frac{\partial L_i}{\partial x} + \lambda^T \frac{\partial f}{\partial x} + \dot{\lambda}^T \Big) \eta dt + \int_{\tau_i}^{\tau_{i+1}} \Big(\frac{\partial L_i}{\partial \theta_i} + \lambda^T \frac{\partial f}{\partial \theta_i} \Big) dt \gamma_i + v_{i+1}\big(L_i - L_{i+1} + \lambda^T(f_i - f_{i+1})\big)|_{\tau_{i+1}} \Bigg] + \sum_{i=1}^{N} \frac{\partial \Psi}{\partial \theta_i}\gamma_i + \frac{\partial \Psi}{\partial x}\eta|_{\tau_{N+1}} \tag{73}$$

Allowing λ to be defined as in (68) and ξ_i to be defined as

$$\xi_i(t) = \int_t^{\tau_{i+1}} \Big(\frac{\partial L_i}{\partial \theta_i} + \lambda^T \frac{\partial f}{\partial \theta_i} \Big) ds + \frac{\partial \Psi}{\partial \theta_i}, \tag{74}$$

equation (73) can be simplified as

$$\frac{1}{\epsilon}\Big(\hat{J}(\tau+\epsilon v,\theta-\epsilon\gamma)-\hat{J}(\tau,\theta)\Big) = \tag{75}$$

$$= \sum_{i=0}^{N}\Big(\xi_i(\tau_i)\gamma_i + v_{i+1}\big(L_i - L_{i+1} + \lambda^T(f_i - f_{i+1})\big)|_{\tau_{i+1}}\Big)$$

which gives the partials in (66) and (67). The dynamics given in (69) can be obtained by differentiating (74) with respect to t. A very similar proof of the variation in the negative direction yields the same result.

□

4.2 Example: Vector-field Orbiting for Nonholonomic Vehicle

To illustrate the utility of the MPC approach presented in the previous section, we present a control method amenable to the proposed framework which will allow a nonholonomic mobile robot to follow a vector field. This has an array of applications as vector field approaches are the basis of many control schemes for mobile robots, e.g. [2, 63, 41, 48, 76]. More importantly, this provides for a good example for the MPC framework as the behavior is able to overcome the nonholonomic constraints and the MPC scheme is able to optimize over the parameters of the behaviors. We will proceed by outlining the control law, giving optimality conditions necessary for use with Theorem 9, and ending with an example utilizing the MPC framework for orbiting.

4.2.1 Non-Linear Unicycle Control

To account for the motion constraint present in mobile platforms, we utilize the unicycle motion model which is a common method used to model planar motion in mobile robotic platforms, e.g., [41, 48]. Figure 14 shows a diagram of a typical unicycle robot where the

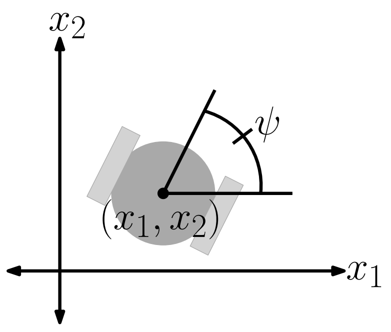


Figure 14: This figures shows a diagram of the states of a unicycle robot. (x_1, x_2) gives the position and ψ gives the orientation.

state dynamics are given as:[2]

$$\dot{x} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ \omega \end{bmatrix}, \tag{76}$$

and v and w correspond to the input translational and rotational velocities of the vehicle, respectively.

One common method of making a unicycle robot follow a vector field is to use a proportional-derivative (PD) control, e.g., [41]. However, due to the differential term, this type of control is difficult to use in optimization as the partial derivative of the control is needed. Therefore, we present a nonlinear unicycle control which is capable of following a vector field while being easily incorporated into our optimization framework.

To do so, we give an alternate expression for the unicycle dynamics which makes our controller very simple to express. The unicycle dynamics given in (76), with control input $u = \begin{bmatrix} v & \omega \end{bmatrix}^T$, can be rewritten in Cartesian coordinates as

$$\begin{aligned} \dot{p} &= vh \\ \dot{h} &= \omega J h \end{aligned} \tag{77}$$

[2]Note that for a concise development, when we are not speaking of MPC, we remove the time indexing or use solely a single time index when clarity is needed (i.e. $x(t) = x(t;t)$).

where $p = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$, $h = \begin{bmatrix} \cos(\psi) & \sin(\psi) \end{bmatrix}^T$, and

$$J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{78}$$

is the 90-degree rotation matrix. The state space of (77) is $X \triangleq \mathbb{R}^2 \times S_1$ – the plane (which represents positions), together with the circle (which represents orientations).

Given a compact workspace $\Omega \subset \mathbb{R}^2$ containing the origin, together with positive definite,[3] continuously-differentiable function $U : \Omega \to \mathbb{R}$, we define the controller,

$$\begin{aligned} \omega &= -k_\omega (\operatorname{grad} U)^T J h \\ v &= -k_v (\operatorname{grad} U)^T h \end{aligned} \tag{79}$$

where $k_\omega, k_v > 0$ are arbitrary constants.

The controller, (79), globally asymptotically stabilizes the robot to the origin of the workspace, without regard to the robot's orientation. This is stated formally by the next theorem:

Theorem 10 (Unicycle Stabilization)**.** *Let $\Omega \subset \mathbb{R}^2$ be a compact set (the workspace), and $U : \Omega \to \mathbb{R}$ a positive definite, continuously-differentiable scalar field. Then (79) globally stabilizes the dynamics in (77) to the set $X_0 \triangleq \{(p, h) \in \mathbb{R}^2 \times S_1 \mid p = \mathbf{0}\}$.*

Proof: The proof uses LaSalle's Theorem, and the candidate Lyapunov function,

$$X \ni (p, h) \underset{V}{\longrightarrow} U(p) \; ; \tag{80}$$

i.e., we treat U, which is a function defined only on the workspace Ω, as a function V on the entire state space $X = \Omega \times S_1$.

Differentiating V in time and substituting from (77) and (79), we obtain

$$\dot{V} = -k_v \left\langle \operatorname{grad} U, h \right\rangle^2 \leq 0 \; , \tag{81}$$

[3]This positive-definiteness requirement can be omitted, in which case stabilization to a local minima is guaranteed.

so the nonincreasing-Lyapunov-value condition of LaSalle's Theorem is satisfied. We will denote by E the set of states where (81) holds.

Moreover, $\dot{V} = 0$ only when $\operatorname{grad} U \perp h$, in which case (79) implies

$$|\omega| = k_\omega || \operatorname{grad} U || \tag{82}$$

and $\dot{x} \neq 0$ (so long as $|| \operatorname{grad} U|| \neq 0$). Consequently, X_0 is not just positively-invariant, but also the largest positively-invariant set in E, and by LaSalle's Theorem is the positive limit set of (77) under the controller (79).

This shows that the control law will follow a gradient field to a minima. For a general vector field, where $u \in \mathbb{R}^2$ is an element of that field, we can modify (79) to follow the vector field as

$$\begin{aligned} \omega &= -k_\omega \|u\| \sin(\phi) \\ v &= -k_v \|u\| \cos(\phi), \end{aligned} \tag{83}$$

where $\phi = \operatorname{atan2}(u_2, u_1) - \psi$. This can be found by noting that $Jh \perp h$ and the use of the definition of the inner product (i.e. $\langle a, b\rangle = \|a\|\|b\| \cos(\psi)$).

4.2.2 Partials for Cost Optimization

While the given unicycle control is able to follow a vector field, it is also important in this context for its ability to easily be incorporated into the optimization framework presented in Section 4.1. To make this clear, we set $k_v = k_\omega = 1$ and give the control in the form of (64) as

$$f(x,\theta) = \begin{bmatrix} \|u(x,\theta,O)\| \cos(\phi)\cos(\psi) \\ \|u(x,\theta,O)\| \cos(\phi)\sin(\psi) \\ \|u(x,\theta,O)\| \sin(\phi) \end{bmatrix}. \tag{84}$$

Since u is an element of a vector field, it can be a function of the state, x, the environmental data present to the robot, O, as well as a vector of parameters, θ.

Defining the control as such allows us to write the following theorem which can then be used to find the optimal parameters at each time step in conjunction with Theorem 9 and a definition of the vector field.

Theorem 11. *The partial of (84) with respect to a parameter γ, where γ can be x_i or an element of θ_i given in (65), is given as*

$$\frac{\partial f}{\partial \gamma} = \begin{bmatrix} \frac{\partial v}{\partial \gamma}\cos(\psi) - v\sin(\psi)\frac{\partial \psi}{\partial \gamma} \\ \frac{\partial v}{\partial \gamma}\sin(\psi) + v\cos(\psi)\frac{\partial \psi}{\partial \gamma} \\ \frac{\partial w}{\partial \gamma} \end{bmatrix}, \tag{85}$$

where

$$\frac{\partial v}{\partial \gamma} = \frac{1}{\|u\|}u^T R(\phi)\frac{\partial u}{\partial \gamma} + \|u\|\sin(\phi)\frac{\partial \psi}{\partial \gamma},$$
$$\frac{\partial w}{\partial \gamma} = \frac{1}{\|u\|}u^T R(\phi - \frac{\pi}{2})\frac{\partial u}{\partial \gamma} - \|u\|\cos(\phi)\frac{\partial \psi}{\partial \gamma},$$
$$R(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix},$$

and $v = \|u\|\cos(\phi)$.

Proof. The derivation comes directly from taking the partial derivative with respect to (84) and algebraic simplification using rotation matrices. □

4.2.3 Orbit Example

To demonstrate the ability of the MPC framework presented in Section 4.1 to optimize the parameters of the behavior, we continue the orbiting example starting in the introduction of this chapter. Orbiting is often accomplished by having the vehicle follow a vector field that creates a stable limit cycle [63, 41]. As such, we parameterize the control law given in [41] to allow our method to adapt the vector field online and express it as

$$u(x) = g_s \begin{bmatrix} \gamma & \omega_{orb} \\ -\omega_{orb} & \gamma \end{bmatrix} \hat{x}, \tag{86}$$

where

$$\gamma = g_{lc}\Big(r^2 - \|\hat{x}\|^2\Big),$$

$\hat{x} = x_p - c$, $x_p = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ is the two-dimensional position of the robot, $c \in \mathbb{R}^2$ is the center of the orbit, $g_s \in \mathbb{R}$ is a gain on the speed, $g_{lc} \in \mathbb{R}$ is a gain on convergence to the limit cycle, $\omega_{orb} \in \mathbb{R}$ is the desired frequency of the orbit, and $r \in \mathbb{R}$ is the radius of the orbit. To adapt the vector field using our MPC scheme we allow the parameter vector to be optimized to be $\theta = \begin{bmatrix} g_s & g_{lc} & \omega_{orb} & r \end{bmatrix}$.

The goal we set to accomplish is to approach a desired orbit while maintaining a given velocity, v_d, and with as little angular velocity as possible. Therefore we define our instantaneous cost as

$$L_i = \frac{\rho_1}{2}(v - v_d)^2 + \frac{\rho_2}{2}\omega^2, \tag{87}$$

and our terminal cost as

$$\Psi = \frac{\rho_3}{2}\big(\|x - c\| - r\big)^2, \tag{88}$$

and set $\Phi = 0$. To optimize the parameters, we use the gradients given in Section 4.1.2, along with an Armijo step-size for the gradient step. We found that we could take five steps in approximately 0.02 seconds, and used $\delta_{execute} = 0.02$.

Figures 13, 15, and 16 illustrate the result of using the MPC framework to adapt the parameters in order to minimize the cost. Figure 13 shows a comparison between two hand-tuned sets of parameters for the orbiting control and the parameters adapted using the behavior-based MPC approach. It is seen that the behavior-based approach is able to achieve quick convergence to the orbit as well as a smooth transition into a less aggressive set a parameters to avoid a jittery orbiting behavior.

We were also able to see improvement as we used a string of behaviors where each behavior was the orbiting behavior with separate parameters to be optimized. Figure 15 shows the results of using two behaviors. It illustrates the ability of the behavior-based approach to anticipate the need for a change of variables as the robot reached the point where it began circling. Figure 16 shows the resulting cost of increasing the number of behaviors. While intuitively it may seem that with each additional behavior the cost should

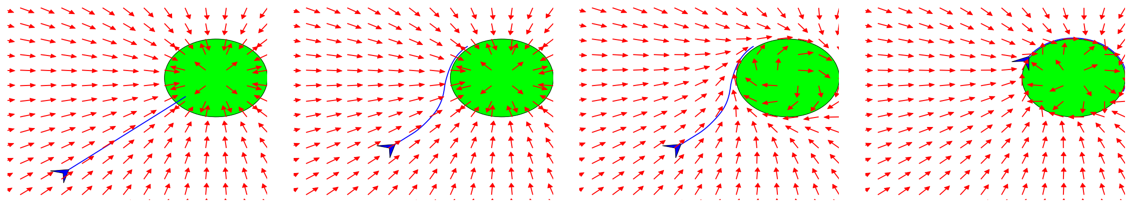


Figure 15: This figure shows different snapshots in time of the adaptation of the vector-field given in equation (86). A series of two behavior was executed at each time instant, each being an implementation of the control law in (84) with different parameters for the vector field. The robot is shown with its planned trajectory extending from it in each case. The middle two images are actually the same time instance where the middle-left image shows the vector field produced in the first time window and the middle-right image shows the vector field produced in the second time window.

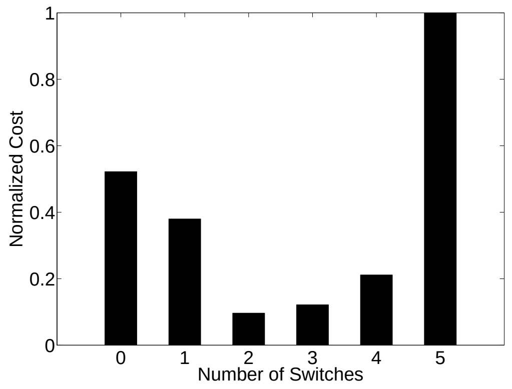


Figure 16: This figure shows the costs associated with different numbers of switches with each behavior executing the control law in (84) with different parameters. The costs are normalized so that the largest cost is scaled to one.

be reduced, this is not the case. Each additional switch time to be optimized introduces non-convexities in the cost, which causes the gradient descent strategy employed to get stuck in local minima. Overcoming these local minima is a contribution presented in Chapter 5.

4.3 Example: Vector-field Navigation for Inverted Pendulum Robot

We now expand on the capabilities of the behavior-based MPC scheme to present an example where a string of behaviors is used to perform the desired task on a robot with a complex motion model. The MPC scheme is illustrated through the control of a two-wheeled nonholonomic inverted pendulum robot. This provides an example where consideration of the

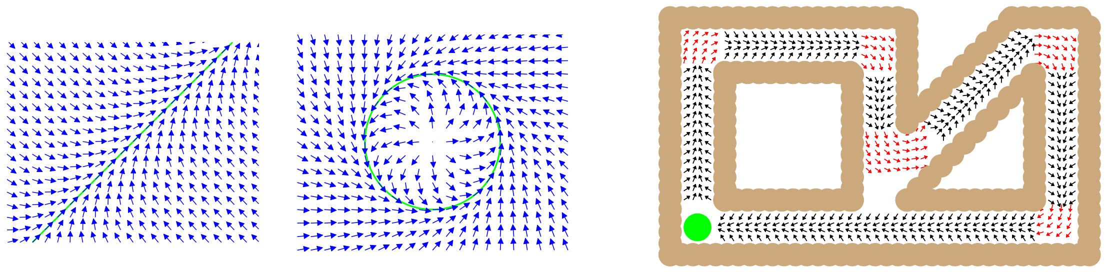


Figure 17: On the left is shown the line and orbit vector fields. On the right is shown an example of the vector fields being concatenated together to guide the robot on a path through the environment.

dynamics of the system is very important when planning for the action. Not only must the planner consider the nonholonomic constraints, which limit the robot's possible movement, it must also maintain balance and an awareness that the robot is unable to have instantaneous changes in velocity when avoiding obstacles. This example is representative of a large host of mobile platforms which face similar nonholonomic constraints and/or must consider stability when planning actions.

To control the inverted pendulum robot, we adapt a navigation scheme presented in [63, 4] in which a string of vector fields can be used to navigate a complicated environment, as shown in Figure 17. By using the behavior-based MPC formulation, both the parameters associated with a predefined sequence of vector fields as well as the time to switch between each can be optimized. This example alludes to a possible general control scheme (expanded upon in Chapter 5) in which deliberative planning is done on a lower dimensional space, ignoring the full dynamics of the system (which is often the case, e.g. [4, 48, 63, 64]), and the parameterized MPC adapts low level control laws to achieve a desirable result. Thus, the deliberative planner can influence the MPC scheme through the choice of cost and schedule of control laws, and the MPC scheme can adapt certain parameters while taking into account the full dynamic model of the system to ensure success.

The remainder of this section proceeds as follows: In Section 4.3.1.1, a model for the inverted pendulum is presented and a control law is designed to be able to control the robot through desired translational and rotational velocity commands. This allows for the vector field control in Section 4.2.1 to be employed. A line following vector field is then designed in Section 4.3.2 to compliment the orbiting vector field already designed. The section ends giving two examples employing the navigation approach combined with the behavior-based MPC algorithm.

4.3.1 Inverted Pendulum Robot

The dynamics and feedback control law for the two-wheeled inverted pendulum robot which will be used to illustrate the utility of the MPC-scheme in the following sections are now introduced. The reason for using this robot model as an example is that the difficulties and complexities associated with its control are representative of those associated with many mobile platforms. The nonholonomic constraints are similar to the constraints present when modeling planar motion in mobile robotic platforms such as cars, differential drive systems, and unmanned aerial vehicles (UAV) e.g., [41, 48]. Also, balancing considerations are similar to stability issues encountered in other vehicles such as UAVs [4]. Using a full dynamic model instead of a kinematic model allows us to incorporate the fact that the desired velocities cannot be instantaneously controlled. Thus, the methods of motion planning considered in this chapter could be applied to a large class of systems.

4.3.1.1 Dynamics of Two-Wheel Inverted Pendulum

To design a model that will incorporate the stability concerns of a two-wheel inverted pendulum robot while also being amenable to common navigation methods, we adjoin the kinematic unicycle motion model in (76), with the dynamics derived in [42]. The kinematic model of the unicycle captures the nonholonomic constraint introduced by the wheels and the dynamics presented in [42] consider the dynamic effects of the input torques on the orientation and balance of the robot. The dynamic models can be joined together as the

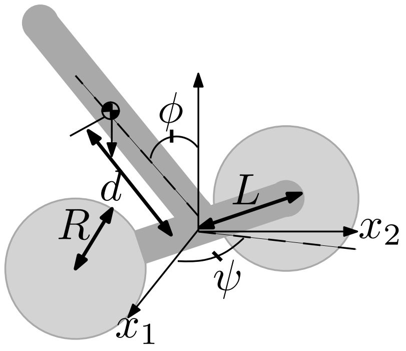


Figure 18: Shown is a diagram of the inverted pendulum robot with the symbols defined in Table 7.

translational and rotational velocities of the unicycle model are states of the inverted pendulum. The first state given in [42] can be removed as it corresponds to the distance traveled by the robot and does not appear in the dynamic equations. The full state of the robot can be expressed as:

$$x = \begin{bmatrix} x_1 & x_2 & v & \psi & \dot{\psi} & \phi & \dot{\phi} \end{bmatrix}^T, \tag{89}$$

where x_1, x_2, v, and ψ are defined as before and ϕ is the tilt angle from the vertical, as depicted in Figure 18. This allows the dynamics of the system to be expressed as

$$\dot{x} = f(x,u) = \begin{bmatrix} v\cos(\psi) & v\sin(\psi) & \dot{v} & \dot{\psi} & \ddot{\psi} & \dot{\phi} & \ddot{\phi} \end{bmatrix}^T \tag{90}$$

where $\dot{v}$, $\ddot{\psi}$, and $\ddot{\phi}$ are obtained from the following equations

$$3(m_c+m_s)\dot{v} - m_s d\cos(\phi)\ddot{\phi} + m_s d\sin(\phi)(\dot{\phi}^2+\dot{\psi}^2) = -\frac{1}{R}(\alpha+\beta), \tag{91}$$

$$\big((3L^2+\frac{1}{2R^2})m_c + m_s d^2\sin^2(\phi) + I_2\big)\ddot{\psi} + m_s d^2\sin(\phi)\cos(\phi)\dot{\psi}\dot{\phi} = \frac{L}{R}(\alpha-\beta), \tag{92}$$

$$m_s d\cos(\phi)\dot{v} + (-m_s d^2 - I_3)\ddot{\phi} + m_s d^2\sin(\phi)\cos(\phi)\dot{\phi}^2 + m_s g d\sin(\phi) = \alpha+\beta. \tag{93}$$

The symbols are defined in Table 7, and the input vector is defined as $u = \begin{bmatrix} \alpha & \beta \end{bmatrix}^T$.

4.3.1.2 Velocity Controller

As we now have a formulation of the dynamics of the system, we design a feedback control law that will allow us to control the robot with the same inputs as used for the control of

Table 7: This table defines the symbols used in the dynamics of the two-wheel inverted pendulum robot. The numeric values are given in [42].

Table of Symbols	
m_c	Mass of wheel
m_s	Mass of body
d	Distance from center of wheel axis to center of gravity
L	Half the distance between the wheels
R	Radius of wheels
I_2	Rotational inertia of the body about the x_3 axis
I_3	Rotational inertia of the body about the axel
$\alpha.\ \beta$	Wheel Torques

a robot executing the unicycle kinematics. This fits into the form of the dynamics given in (4), where the tunable parameters are the desired translational and rotation velocities. It also allows for control methodologies similar to those used for unicycle kinematics, but that are tuned on-line using the parameterized MPC to consider the full dynamics.

To create this feedback law, we use linear quadratic (LQ) control with infinite horizon, e.g. [9]. As this is a highly developed control methodology, the details are not presented here except to mention that it requires a linear system, which is now derived. When (90) is linearized about $x = 0$ and $u = 0$, it produces a system which is not completely controllable (see [8] for details on controllability of linear systems and linearization). Since the goal is to control the velocities, the system can be linearized around a subset of the states which are completely controllable. Namely, let z be defined as $z = \begin{bmatrix} v & \dot{\psi} & \phi & \dot{\phi} \end{bmatrix}^T$ and linearize about $z = 0$ and $u = 0$ to obtain

$$\dot{\delta z} = \begin{bmatrix} 0 & 0 & a_{13} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & a_{43} & 0 \end{bmatrix} \delta z + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ 0 & 0 \\ b_{41} & b_{42} \end{bmatrix} \delta u \qquad (94)$$

$$a_{13} = \frac{d^2 g m_s^2}{2d^2 m_s^2 + 3m_c d^2 m_s + 3I_3 m_s + 3I_3 m_c} = 2.1639,$$

$$a_{43} = \frac{3dgm_s^2 + 3dgm_c m_s}{(2d^2 m_s^2 + 3m_c d^2 m_s + 3I_3 m_s + 3I_3 m_c)} = 72.4858,$$

$$b_{11} = -\frac{m_s d^2 + R m_s d + I_3}{R(2d^2 m_s^2 + 3m_c d^2 m_s + 3I_3 m_s + 3I_3 m_c)} = -1.6687,$$
$$b_{21} = \frac{2LR}{6m_c L^2 R^2 + 2I_2 R^2 + m_c} = 0.0290,$$
$$b_{41} = -\frac{3Rm_c + 3Rm_s + dm_s}{R(2d^2 m_s^2 + 3m_c d^2 m_s + 3I_3 m_s + 3I_3 m_c)} = -24.1514,$$

$b_{12} = b_{11}$, $b_{22} = -b_{21}$, and $b_{42} = b_{41}$. We note that similar constructions were done in [42, 60] without the removal of the states χ and ψ.

To control the translational and rotational velocities, the following change of state can be made

$$\hat{\delta z} = \delta z - \begin{bmatrix} v_d & \omega_d & 0 & 0 \end{bmatrix}^T, \tag{95}$$

where v_d and ω_d are the desired translational and rotational velocities, respectively. After some algebra it can be seen that the dynamics of the new linear system can be expressed as $\dot{\hat{\delta z}} = A\hat{\delta z} + Bu$, where A and B are the matrices in (94).

As this system is completely controllable, an LQ feedback matrix can be used, e.g. [9], to design a control law which will be locally exponentially stable to the desired velocities, e.g. [40]. The linear control law on the nonlinear system is guaranteed to have a region of convergence around the equilibrium, [40]. Experimentally, it was found that if $\|v_d - v\| \leq .8\frac{m}{s}$, the system would always maintain balance.

4.3.2 Straight Path Following

An approach to path following using vector fields was presented in [63] and later modified in [4]. The basic idea being that trajectory tracking, which assigns a desired position to a given instance in time, can present problems when disturbances, such as wind, deter a UAV from being at the desired location. If the trajectory is not updated, then the UAV could over-correct in attempting to follow the predefined trajectory. In many instances, the time component is not a necessary element of the desired vehicle motion. In such a case, path planning, as opposed to trajectory planning, is more desirable. This drops the assignment of a specific time value to a position.

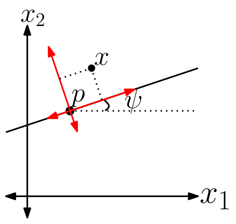


Figure 19: This figure shows the definition of a line through a point p and with a given angle ψ. It also shows the coordinate frame of the line used to create a line following vector field.

In [4, 63] a method was presented in which the path could be followed and disturbances overcome by defining vector fields for the robot to follow. Specifically, simple vector fields could be executed in a predefined sequence to produce the desired path. We now develop a vector field to allow the robot to follow a straight line in the plane to be used in conjunction with the orbiting vector field developed in Section 4.2.3 (as shown in Figure 17).

Note that a line in a plane can be defined by

$$\{x : x = p + k \begin{bmatrix} \sin\psi & \cos\psi \end{bmatrix}^T, \forall\, k \in \mathbb{R}\} \tag{96}$$

where $p \in \mathbb{R}^2$ is a point on the line and $\psi \in [-\pi, \pi]$ is the angle from the x_1 axis, as shown in Figure 19. To create a vector field that will stabilize a vehicle to the line, a coordinate transform can first be performed to define a frame where p is the origin and the x_1 axis points along the line, also illustrated in Figure 19. The desired angle of travel, ψ_d, can then be determined in the new coordinate frame using a sigmoid function, where at an infinite distance the vector field would point directly towards the line and at a close distance it points along the line. This desired angle can be defined as

$$\psi_d = \frac{\pi}{2}\Big(1 - \frac{2}{1 + e^{-a_d d}}\Big), \tag{97}$$

$$\text{s.t. } d = \begin{bmatrix} 0 & 1 \end{bmatrix} R(-\psi)(x - p),$$

$$R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix}$$

With the desired direction of travel in the transformed frame, a unit vector can be transformed back to the original frame. This vector is then scaled by the desired velocity and can be expressed as

$$u = v_d R(\psi) \begin{bmatrix} \cos(\psi_d) \\ \sin(\psi_d) \end{bmatrix}, \tag{98}$$

where u is the vector in the vector field and R is the rotation matrix defined in (97). The parameters that can be adapted in the parameterized MPC approach are a_d and v_d which affect the desired slope of the field and translational velocity respectively.

4.3.3 Navigation Examples

We now present examples where a pre-planned series of vector fields can be adapted on-line through the behavior-based MPC approach to account for the full dynamics of the inverted pendulum robot. To follow these vector fields, the vector field following controller presented in Section 4.2.1 is used to produce desired velocity values for the inverted pendulum control law presented in Section 4.3.1.2. The behavior-based MPC algorithm is used to optimize over the time to switch between vector fields as well as the parameters that will be best for the execution of the desired path. Two examples are provided to illustrate the application of this control method, namely executing a figure-eight and navigating through corridors.

The method we take to have the robot execute this problem is to work with two modes at a time. For example, in the figure-eight execution, the robot starts off following a straight line vector field. While executing this control, it optimizes over the time to switch to the orbiting field as well as the parameters associated with each controller. Once it has switched to the orbiting vector field, we then append the second straight path mode and the robot must optimize over the time to switch to the second straight path mode along with the parameters associated with both controllers. This process is repeated each time the robot switches to a new mode.

To optimize over the variables, a cost can be created which penalizes deviation from the desired vector field, maintains a given velocity, $\bar{v}$, avoids the obstacles, and penalizes large tilt angle. The instantaneous cost portion of this cost can be expressed as

$$L_i = \frac{\rho_1}{2} dist(x_p)^2 + \rho_2 \sum_{i=1}^{N_s} r(x_p, o_i) + \frac{\rho_3}{2}\phi^2 + \frac{\rho_4}{2}(v - \bar{v})^2, \tag{99}$$

where $dist(x)$ is defined as is d in (97) for the follow line mode and as the first term in (100) for the orbit mode. The terminal cost for the orbit mode is defined as

$$\Psi_o = \frac{\rho_5}{2}\big(\|x_p - c\| - r\big)^2 + \frac{\rho_6}{2}(v_{d_1} - v_{d_2})^2, \tag{100}$$

and the terminal cost for the line-follow mode is defined as

$$\Psi_l = \frac{\rho_5}{2}\|x_p - r_l\|^2 + \frac{\rho_6}{2}(v_{d_1} - v_{d_2})^2, \tag{101}$$

where v_{d_1} is the desired velocity from the first mode and v_{d_2} is that of the second mode. In the navigation example, we use the terminal cost associated with the second mode to encourage progress along the path. In the figure-eight example, we alternate between the final costs associated with each orbit once the robot passes the top or bottom of the figure-eight. This encourages the robot to always be trying to go into orbit around the obstacles and produces the figure-eight trajectory. The second term in the terminal costs helps maintain the same velocity between switches to avoid large tilt angles resulting from large deviations in the desired velocities.

The results of each simulation can be seen in Figures 20 and 21 which show that the robot was successfully able to traverse the desired paths while maintaining balance. This is shown as that the robot was able switch between and converge to the desired vector fields while navigating through the environments. Moreover, Figure 21 shows that the robot was again able to maintain balance by slowly varying the translational velocity in order to maintain a small tilt angle. This illustrates the ability of the MPC scheme to adapt parameters and follow a scheduled set of modes while considering the full dynamic model of the system.

The computational burden associated with this MPC approach can be seen in Figure 22 which shows the average time of performing the parameterized MPC optimization at each

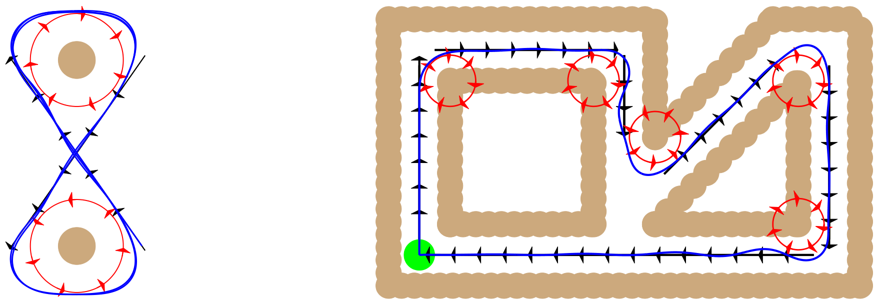


Figure 20: The trajectory and underlying orbits for the vector field navigation examples are shown above.

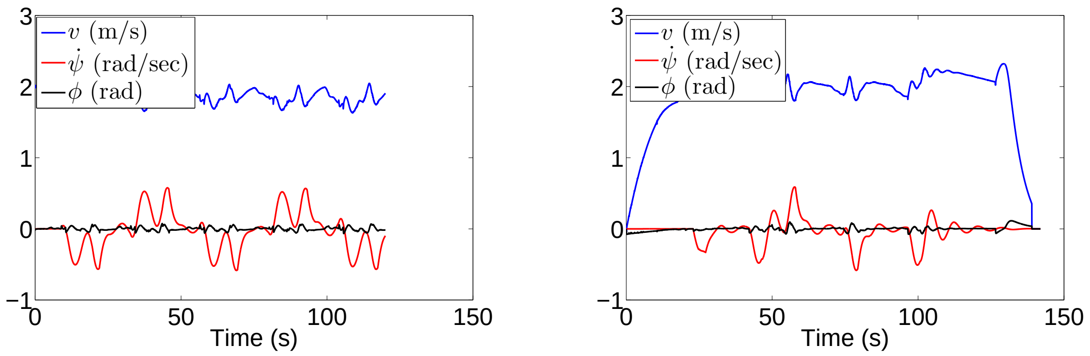


Figure 21: Above are shown three states during the vector field navigation of the environments shown in Figure 20.

time step for the figure-eight trajectory when a certain number of gradient steps is allowed. An appropriate value for $\delta_{execute}$ in step 2 of Algorithm 1 can be selected as 0.05 for two or three steps or 0.1 for up to twenty steps. An interesting note to make is that the optimization converges quite quickly. The optimization time for twenty steps is only slightly larger than the optimization time for five steps, meaning that the Armijo step algorithm did not require all the allowed optimization steps.

To see the utility of the parameterized MPC approach we show the cost associated with different numbers of gradient steps in Figure 22. The usefulness of the parameterized MPC

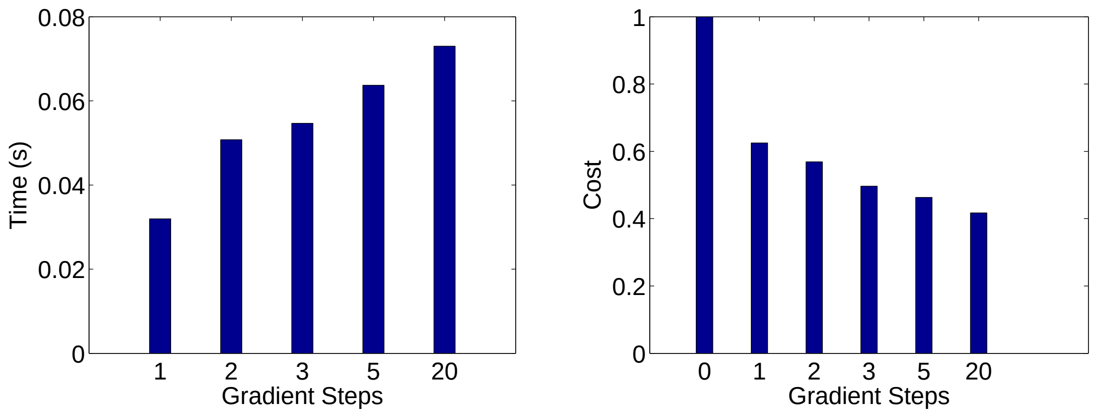


Figure 22: On the left is shown the average computation time per gradient step while executing the figure-eight shown in Figure 21. On the right is shown the normalized cost per gradient step of a figure-eight. The cost is normalized so that no optimization has a cost of 1.

approach is seen as it outperforms the case where no gradient steps are taken (ie hand-tuned control laws with fixed switch times are used). Using a single gradient step, the total cost is reduced by approximately 40%, with up to 60% reduction achieved with twenty steps.

Remark 4. ***A Note on Stability:*** *In the example of the previous section, stability was ensured through placing limitations on the allowable commanded velocities. Due to a finite execution time enforced by $\delta_{execute}$, and the fact that switching between control laws occurred infrequently, the inverted pendulum was able to maintain balance. To apply the behavior based approach to other situations or vehicles, stability may not be as easily maintained.*

As we are using an approach based on concatenating control laws, hybrid control techniques for finding appropriate Lyapunov functions, e.g. [49], could be employed, although this may not always be a trivial task. We can, however, use MPC techniques to ensure stability. In Chapter 5, an extensive example is given which uses a dual-mode MPC approach to ensure stability when navigating an unknown environment. Similar to the examples in this chapter, stability will come from the underlying control laws used for motion control, with additional conditions on both the costs and control laws being employed.

4.4 Conclusion

In this paper we have presented an MPC strategy which utilizes the ability of feedback control laws to create desirable trajectories, exchanging a two-point boundary value optimization problem for a parameter optimization problem. The versatility of this method was demonstrated through two different examples where robots adapted vector fields as a method of motion control. Both examples showed the ability of the robot to adapt the parameters of the control laws on-line to achieve the desired result.

CHAPTER V

DUAL-MODE DYNAMIC WINDOW APPROACH TO NAVIGATION

We build upon the behavior-based MPC formulation developed in the previous chapter to demonstrate its ability to be a viable component in a motion planning framework. Vehicle motion planning in unknown environments forms an integral part of robotics and is arguably a solved problem under certain conditions, e.g. [47, 2, 48, 29]. However, when stability becomes an issue, e.g. at high speeds, or when optimality considerations are to be taken into account, the problem is not yet solved. Even when the environment is completely known in advance, optimal solutions can be difficult to compute as dynamic constraints, such as acceleration and motion limitations, must be considered, especially as the speed of the robot increases, e.g. [29, 82].

In [29] it was noted that analytic solutions to the optimal motion planning problem are only computable for the most simple of cases, which leads to the need for approximation algorithms for pretty much any realistic scenario. The difficulty associated with incorporating dynamic constraints is compounded in unknown environments as a solution must be repeatedly computed to take new environmental data into account. In this chapter, we address this very issue by combining DWA with a fast, deliberative planner through a dual-mode behavior-based MPC construction.

As mentioned in Section 1.3.3, DWA provides a direct way of incorporating dynamic constraints for fast navigation through an unknown environment, but lacks general convergence guarantees, [24, 64, 7]. In a way, one can think of the modification for DWA presented in [64] as adding a deliberative component to DWA, albeit a very specific deliberative component. The contribution of this chapter is a generalization of this idea, and, similar to [64], we capatilize on existing theory of MPC to provide guarantees of convergence; instead of navigation functions, the deliberative component is allowed to be a generic path-planner. Thus, a “global planner” finds a path, giving guarantees such as completeness, e.g. [48], and the MPC framework takes into account the full dynamic model of the system to give guarantees of convergence to the goal location.

The remainder of the chapter will proceed as follows: The dual-mode arc-based MPC approach is detailed in Section 5.1. Implementation details are given in Sections 5.2 and 5.3 for an Irobot Magellan-pro. It is shown to successfully run at 80% of its maximum velocity, while traversing tight corridors in an unknown environment despite the robots severely limited computational resources and sluggish dynamics. A further demonstration of the ability of the framework to deal with complicated dynamics, while maintaining the convergence guarantees, is presented in Section 5.4 through a simulation of an inverted pendulum robot. The chapter ends with concluding remarks in Section 5.5.

5.1 Dual-mode Arc-based MPC

The proposed dual-mode arc-based MPC algorithm builds upon DWA by using a dual-mode MPC approach to incorporate a reference tracking controller to ensure that the robot converges to some goal position while incorporating dynamic constraints. This section develops the dual-mode approach by first presenting the algorithm, giving a convergence theorem, and then discussing how a behavior-based approach can be used as part of the optimization.

5.1.1 Dual-mode Arc-baseed MPC Algorithm

As the dual-mode arc-based MPC algorithim is an example of the behavior-based MPC framework presented in Chapter 4, many of the details are very similar. We review the basic setup here for sake of clarity. It is assumed that the robot will execute a given sequence of control laws, denoted as $(\kappa_0, \tau_0), (\kappa_1, \tau_1), ..., (\kappa_N, \tau_N)$, where τ_i indicates the time when the system will switch from executing the control law κ_{i-1} to the control law κ_i. Each control law is a function of the state and a tunable vector of parameters, written as $\kappa_i(x(t;t_0), \theta_i)$. This allows the system dynamics to be written as $\dot{x}(t;t_0) = f(x(t;t_0), \kappa_i(x(t;t_0), \theta_i)$ for $\tau_i \leq t < \tau_{i+1}$.

In the proposed algorithm, we assume that the unicycle motion model in (5) forms part of the state dynamics where v and ω are either the inputs, as in (5), or additional states of the system. The first N controllers in the sequence regulate the dynamics to desired constant velocities, with the parameter vector being the desired velocities on that interval, i.e. $\theta_i = [v_i,\ \omega_i]^T$ for $i = 0, ..., N-1$. The final control law is designed to track a reference trajectory, $y_d(t) \in \mathbb{R}^2$. There is no need for a parameter vector, so we deviate from the original notation and write the final controller as $\kappa_N(x(t), y_d(t))$, where the time index is included to denote that $y_d(t)$ is time varying. An example of a possible trajectory is illustrated in Figure 23 where three arc-based controllers are executed back to back with a reference tracking controller at the end.

The reference trajectory is produced by planning a path to the goal location, $y_{goal} \in \mathbb{R}^2$, and creating a continuous mapping from time to a position on the path. In the examples presented in Sections 5.3 and 5.4, the path planning is done using A^*. Mapping from time to position is done by respecting translational velocity constraints. However, we note that

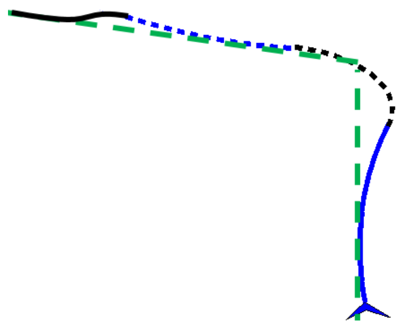


Figure 23: This figure shows an example of a dual-mode arc-based trajectory. The robot is shown as a triangle with a planned trajectory extending from it. The trajectory is created from three arc-based controllers appended back to back with a reference tracking controller at the end. The different portions of the trajectory are differentiated by color and line styles. A reference trajectory is also shown as a dashed line.

this is merely an example and not essential to the formulation of the algorithm.

To explicitly represent details specific to the arc-based MPC algorithm, the cost is slightly modified from (65). A key point to note is that the environment is not completely known at the time of optimization. So the set of free or unexplored positions, $B_{free}(t_0) \subset \mathbb{R}^{2}$[1], is included as a term in the instantaneous cost. The final change made is to explicitly represent the fact that the terminal cost depends upon the reference trajectory where $y(t; t_0) = h(x(t; t_0)) \in \mathbb{R}^2$ is the position of the robot that is expected to follow the reference trajectory. The optimization problem can be written to incorporate these changes as:

$$\min_{\tau,\theta} J(\tau, \theta) = \int_{t_0}^{t_0+\Delta} L\big(x(t; t_0), \theta, B_{free}(t_0)\big)dt + \Psi\big(y(\tau_{N+1}; t_0), y_d(\tau_{N+1})\big), \qquad (102)$$

$$\text{s.t. } \dot{x}(t; t_0) = \begin{cases} f\big(x(t; t_0), \kappa_i(x(t; t_0), \theta_i)\big) & \tau_i \leq t < \tau_{i+1}, i = 0, ..., N-1 \\ f\big(x(t; t_0), \kappa_i(x(t; t_0), y_d(t))\big) & \tau_N \leq t < \tau_{N+1} \end{cases}$$

where $\tau = [\tau_0, ..., \tau_{N+1}]^T$, and $\theta = [\theta_1^T, ..., \theta_{N-1}^T]^T$. Note that we actually do not optimize with respect to the first and last elements of τ, rather we fix them as $\tau_0 = t_0$ and $\tau_{N+1} = t_0 + \Delta$. To denote that the parameters only enter the cost for the time period over which they are used, the instantaneous cost can be written as:

$$L\big(x(t; t_0), \theta, B_{free}(t_0)\big) = \begin{cases} L_i\big(x(t; t_0), \theta_i, B_{free}(t_0)\big) & \tau_i \leq t < \tau_{i+1}, i = 0, ..., N-1 \\ L_N\big(x(t; t_0), B_{free}(t_0)\big) & \tau_N \leq t < \tau_{N+1} \end{cases}.$$

We make one final note on obstacle avoidance before stating the dual-mode arc-based MPC algorithm. Previously unseen obstacles may render the desired reference trajectory or previously found parameters invalid due to an unforseen collision. For the scenarios presented in Sections 5.3 and 5.4, it is conceivable that an obstacle avoidance controller, $\kappa_{avoid}(x)$, could consist of steering away from obstacles while slowing down as fast as

[1]We use $B_{free}(t_0)$ as opposed to $O(t_0)$ (used previously) to explicitly denote the fact that the information being incorporated about the environment is the two-dimensional set of free states.

possible. The dynamic models allow angular velocities or accelerations to be controlled independent of the control of the translational velocities or accelerations. The angular velocities or accelerations are also quite responsive.

However, we have found the design of such an avoidance control law unnecessary. A number of predefined parameters defining different arc-based motions can be utilized. As will be discussed in detail in Section 5.1.3, these parameters can be quickly modified to ensure collision avoidance. To incorporate either the design of an obstacle avoidance controller or a method of quickly searching the parameter space, the following assumption is given:

Assumption 9. *(Collision Avoidance) If a collision is detected at time t_0 to occur at time $t_c > t_0$ for either $y_d(t_c)$ or $y(t_c; t_0)$, there exists one of the following:*

1. *A control law, $\kappa_{avoid}(x)$, which will guarantee obstacle avoidance.*
2. *Parameters can be quickly found such that $y(t; t_0)$ is collision-free $\forall\, t \in [t_0,\ t_0 + \Delta]$.*

The dual-mode arc-based MPC algorithm is now stated in Algorithm 2.

5.1.2 Convergence

We now show that, under Algorithm 2, the robot will converge to the goal location, y_{goal}. An underlying assumption is made that the desired reference trajectory leads to the goal location in finite time, i.e. $\exists\, t_g$ s.t. $y_d(t) = y_{goal}\ \forall t \geq t_g$. Convergence is then established by ensuring that Algorithm 2 has certain tracking abilities for $t < t_g$ and making dual-mode MPC arguments for $t \geq t_g$. This section discusses these two aspects of convergence and then gives a theorem about the overall convergence of the algorithm.

5.1.2.1 Sufficient Tracking of Reference Trajectory

The idea of "sufficient tracking" is to ensure that after step 4 of Algorithm 2, the robot plans on getting within δ of the reference trajectory (i.e. $y(t_0 + \Delta; t_0) \in \mathcal{B}_\delta(y_d(t_0 + \Delta))$) and

Algorithm 2 Dual-mode Arc-based MPC

1. Initialize:
   (a) Plan path from $y(t_0; t_0)$ to y_{goal} and assign mapping from time to position to form $y_d(t)$.
   (b) Set $\tau_0 = ... = \tau_N = t_0$.
2. If collision is detected along $y_d(t)$ or $y(t; t_0)$ for $t \in [t_0, t_0 + \Delta]$:
   (a) Cost barriers in terminal cost are dropped.
   (b) Trivial parameters are set (or κ_{avoid} employed).
   (c) New parameters are executed until new path has been planned.
   (d) Assign mapping to form $y_d(t)$.
3. Initialize Parameters θ and τ:
   (a) Test previous values of θ and τ.
   (b) Test a variety of predefined values for θ and τ.
   (c) Choose parameters from steps 3a and 3b which result in lowest cost .
4. Minimize $J(\theta, \tau)$ with respect to θ and τ, using parameters from 3c as an initialization.
5. Execute control sequence for $\delta_{execute}$ seconds.
6. Repeat steps 2 through 6 (incrementing t_0 by $\delta_{execute}$).

collisions are avoided (i.e. $y(t;t_0) \in B_{free}(t_0)\ \forall t \in [t_0, t_0 + \Delta]$). To clearly express this concept, allow a solution (θ, τ, x_0) at time t_0 to consist of the parameters and switch times mentioned in (65) along with an initial condition $x(t_0;t_0) = x_0$. The idea of "sufficient tracking" is encoded through the definition of an admissible solution:

Definition 1. *(Admissible Solution) A solution, (τ, θ, x_0), is said to be admissible at time t_0 if simulating $\dot{x}(t;t_0) = f\big(x(t;t_0), \kappa_i(x(t;t_0), \theta_i)\big)$ for $\tau_i \leq t < \tau_{i+1}$ with initial condition $x(t_0;t_0) = x_0$ results in $y(t;t_0) \in B_{free}(t_0)\ \forall t \in [t_0, t_0+\Delta]$ and $y(t_0+\Delta;t_0) \in \mathcal{B}_\delta(y_d(t_0+\Delta))$.*

Similar to dual-mode MPC, conditions on both the cost and final control are employed to ensure that step 4 of Algorithm 2 is always initialized with an admissible solution and always results in an admissible solution. These conditions are given through the following assumptions:

Assumption 10. *(Collision Barrier) The instantaneous cost, $L : \mathbb{R}^n \times \mathbb{R}^M \times B_{free} \mapsto \mathbb{R}$, forms a cost barrier to collisions such that $L(x, \theta, B_{free}) \longrightarrow \infty$ as $dist\big(h(x), \mathcal{B}^c_{free}\big) \rightharpoonup 0$, $\forall\theta$, where $dist(y, B)$ denotes the distance from point y to the set B.*

Assumption 11. *(Terminal Barrier) The terminal cost, $\Psi : \mathbb{R}^2 \times \mathbb{R}^2 \mapsto \mathbb{R}$, forms a cost barrier around $y_d(t_0 + \Delta)$ such that $\Psi(y, y_d) \longrightarrow \infty$ as $dist\big(y, \mathcal{B}^c_\delta(y_d)\big) \rightharpoonup 0$.*

Assumption 12. *(Trajectory Tracking) If $y_d(t)$ is collision free $\forall\ t \geq t_0$ and $y(\tau_N;t_0) \in \mathcal{B}_\delta(y_d(\tau_N))$, then computing $x(t;t_0)$ using the dynamics $\dot{x}(t;t_0) = f\big(x, \kappa_N(x(t;t_0), y_d(t))\big),\ t \geq \tau_N$ will result in $y(t;t_0) \in B_{free}(t_0)\ \forall t \geq \tau_N$ and $y(t;t_0) \in \mathcal{B}_\delta(y_d(t))\ \forall t \geq \tau_N + \delta_{execute}$.*

Note that in Assumption 12, $\delta_{execute}$ can be used to give the final control law sufficient time to converge to a necessary region of the state-space before it is required to have excellent tracking abilities. Together, these three assumptions allow for a statement of sufficient tracking of the reference trajectory, as given in the following theorem:

Theorem 12. *Given Assumptions 10, 11, and 12, if Step 4 of Algorithm 2 is initialized with an admissible solution at time t_s and $y_d(t)$ is collision free $\forall t \geq t_s$, at each future iteration of Algorithm 2, step 4 will produce parameters θ and τ resulting in an admissible solution.*

Proof. Due to Assumptions 10 and 11, the minimization of (65) will produce an admissible solution if it is initialized with an admissible solution. The reason being that an admissible solution will result in a finite cost while an inadmissible solution will result in an infinte cost. As an admissible solution will result from the optimization step, at the following iteration, step 3a of Algorithm 2 will produce an admissible initialization to step 4. This can be seen to be the case as, after executing the solution from the previous iteration for $\delta_{execute}$, the robot will be planning on using the final control for the last $\delta_{execute}$ seconds of the new time horizon, resulting in an admissible solution due to Assumption 12. $\square$

5.1.2.2 Convergence to Goal Location

Assuming that $y_d(t) = y_{goal}, \forall t \geq t_g$ and $B_{free}(t)$ is constant $\forall t \geq t_g$, dual-mode MPC can be employed to ensure convergence to y_{goal}. The terminal region can be given as $\mathcal{B}_\delta(y_{goal})$ and the reference tracking control, $\kappa_N(x, y_d)$, can be used as the stabilizing controller. Together with the costs, $\kappa_N(x, y_d)$ can be designed to satisfy B1 through B4 once $y_d(t) = y_{goal}$. With an additional assumption on the instantaneous cost, a theorem on convergence can then be state:

Assumption 13. *(Zero Instantaneous Cost) The instantaneous cost is zero when $y \in \mathcal{B}_\delta(y_{goal})$ and greater than or equal to zero elsewhere.*

Theorem 13. *Assuming $y_d(t) = y_{goal}$, $B_{free}(t)$ is constant, and Conditions B1 through B4 are satisfied using $\kappa_f(x) = \kappa_N(x, y_{goal})$, $y(t)$ will converge asymptotically to y_{goal}.*

Proof. The development of stability in [12] can be closely followed to show asymptotic convergence. Equation (65) can be evaluated as a discrete time candidate Lyapunov function, i.e. $V(t_0) = J(\tau, \theta, t_0)$, where the time indexing is added to denote the cost as a function of time. The difference in V after one time step can be written as

$$\begin{aligned}\Delta V =& V(t+\delta_{execute}) - V(t)\\ =& \int_{t+\delta_{execute}}^{t+\Delta+\delta_{execute}} L(x(s;t+\delta_{execute}),\theta,B_{free})ds + \Psi(y(t+\Delta+\delta_{execute},y_{goal})),\\ &- \int_{t}^{t+\Delta} L(x(s;t),\theta,B_{free})ds - \Psi(y(t+\Delta,y_{goal}))\end{aligned} \tag{103}$$

where we have written B_{free} without time indexing to denote that it is constant. The integral terms can be simplified by examining several of the presented conditions. Condition B3 states that κ_N maintains $y \in \mathcal{B}_\delta(y_{goal})$ if y enters $\mathcal{B}_\delta(y_{goal})$, which is by definition a property of an admissible solution once $y_d(t) = y_{goal}$. Also, assuming no perturbations, $x(s;t) = x(s;t+\delta)\ \forall s \geq t+\delta$ when the parameters are maintained the same. Assumption 13 can then be employed to state that $L(x(s;t+\delta_{execute}),\theta,B_{free}) = 0$ for $s \in [t+\Delta,\ t+\Delta+\delta_{execute}]$. This allows the integral terms to mostly cancel out, leaving:

$$\Delta V = \Psi(y(t+\Delta+\delta_{execute},y_{goal})) - \int_{t}^{t+\delta_{execute}} L(x(s;t),\theta,B_{free})ds - \Psi(y(t+\Delta,y_{goal})) \tag{104}$$

Again employing Assumption 13, which states that $L \geq 0$, along with Condition B4 which states that Ψ is decreasing when using κ_N, the result can be simplified further as:

$$\Delta V \leq \Psi(y(t+\Delta+\delta_{execute},y_{goal})) - \Psi(y(t+\Delta,y_{goal})) \leq 0 \tag{105}$$

where strict inequality holds for all $y \neq y_{goal}$. Then, using a discrete time version of LaSalle's invariance principle, [33], the system will converge to the set $\{x | h(x) = y_{goal}\}$, showing asymptotic convergence to the goal location. $\square$

5.1.2.3 *Convergence of the Dual-mode Arc-based MPC Algorithm*

The previous two sections discuss the convergence of Algorithm 2 assuming that the reference trajectory is collision free. One final assumption is made which will allow us to state a theorem on the convergence of the dual-mode arc-based MPC algorithm.

Assumption 14. *(Known Admissible Solution) After re-planning $y_d(t)$ in step 2d of Algorithm 2, an admissible solution can be found.*

Due to the tracking assumption in Assumption 12 and the fact that we can use fast path planners, Assumption 14 is not limiting. An admissible solution can almost always be found by setting the switch times to equal t_0, as in step 1b. However, we note that with the initialization step discussed in detail in Section 5.1.3, better parameters can typically be found. We now state the convergence theorem:

Theorem 14. *Given Assumptions 9 through 14 and that conditions B1 through B4 hold once $y_d(t) = y_{goal}$, executing Algorithm 2 will cause $y(t) = y_{goal}$ without collision where asymptotic convergence is achieved once $y_d(t) = y_{goal}$.*

Proof. Assumption 9 ensures that if a collision is detected due to a previously unseen obstacle, the robot can replan a path while avoiding any obstacles. Due to Assumption 14, Theorem 12 will then hold true for the new trajectory, guiding the robot towards the goal location until either $y_d(t) = y_{goal}$ or another collision is detected. If $y_d(t) = y_{goal}$, then Theorem 13 can be applied. If a collision is detected then the process is repeated. If the path planner is complete, eventually the robot will explore the environment enough such that a collision free path is found, if it exists. □

5.1.3 Behavior-based Optimization

In Section 4.1.2, gradients were derived for the behavior-based approach. However, it is important to note that the cost to be minimized is subject to local minima caused by both the environment and the switched-time optimization. To avoid getting stuck in undesirable local minima, a behavior-based approach capitalizing on arc-based motion primitives can be used to find a good initialization for the gradient-based optimization. This behavior-based approach for initialization is also useful as a technique for avoiding obstacles in step 2b of Algorithm 2.

When introducing the DWA algorithm, it was noted in [24] that a single arc was used due to the complexity of searching the parameter space when more than one arc is used. While a grid-based discretized search of the parameter space, as done in [24, 82, 7], would lead to a significant increase in computation when adding additional arcs, arc-based motion can be exploited to reduce the search space. To exploit the arc-based motion, an analytic solution to the unicycle model in (5) is given in the following Theorem:

Theorem 15. *Assuming v and ω are constant, the solution to (5) can be expressed as*

$$\begin{aligned} x_1(t) &= \frac{v}{\omega}\Big(\sin(\psi(t)) - \sin(\psi_{30})\Big) + x_{10} & \qquad x_1(t) &= \cos(\psi_{30})vt + x_{10} \\ x_2(t) &= \frac{v}{\omega}\Big(\cos(\psi_{30}) - \cos(\psi(t))\Big) + x_{20} & \qquad x_2(t) &= \sin(\psi_{30})vt + x_{20} \\ \psi(t) &= \omega t + \psi_{30} & \qquad \psi(t) &= \psi_{30} \end{aligned}$$

for $\omega \neq 0$ and for $\omega = 0$ on the left and right respectively.

Proof. The proof follows from uniqueness of solutions to differential equations. Taking the time-derivative of the given equations for $\omega \neq 0$ and $\omega = 0$, respectively, will result in the original unicycle model. $\square$

Theorem 15 can be used to quickly compute collision free trajectories when a collision has been detected. This is detailed in the following theorem:

Theorem 16. *Assuming the following:*

1. *The unicycle model for the dynamics in (5) is used with initial condition $x(0) = x_0$.*
2. *The state trajectory, $x(t)$ is formed using velocities (v_i, ω_i) for $\tau_i \leq t < \tau_{i+1}$, where, without loss of generality, $\tau_0 = 0$.*
3. *The time of first collision is given by $t_c > 0$.*

If a scaled trajectory, $x_s(t)$ with $x_s(0) = x_0$, is found by using the velocities $(sv_i, s\omega_i)$ for $\frac{\tau_i}{s} < t < \frac{\tau_{i+1}}{s}$, where $s = \alpha\frac{t_c}{\Delta}$ and $0 < \alpha < 1$. The following holds true:

1. $x_s(t) = x(st)$

2. $x_s(t)$ *is collision free for* $t \in [0,\ \Delta]$

Proof. A proof can be obtained by applying Theorem 15 on each interval for the original and scaled variables, and seeing algebraically that the resulting scaled trajectory, $x_s(t) = x(st)$. Note that if t_c is the time of the first collision then $x(s\Delta) = x(\alpha t_c)$ will be a point on the original trajectory before the collision as $\alpha < 1$. Thus, $x_s(t)$ will be collision free $\forall$ $t \in [0\ \Delta]$. $\square$

Applying Theorem 16, a robot with unicycle dynamics can quickly avoid collisions by slowing down appropriately. This is seen through the illustrations shown in Figure 24. It shows the trajectories resulting from the arcs before and after scaling as well as the results of scaling on the cost contour map. It illustrates that each velocity pair that produces a collision can immediately be mapped into the collision free region by scaling. It is important to note that this method is effectively used on robots with more complicated dynamic models in Sections 5.3 and 5.4 as they use controllers designed to quickly converge to constant velocities.

The concept of scaling can be employed to perform a quick search of the parameter

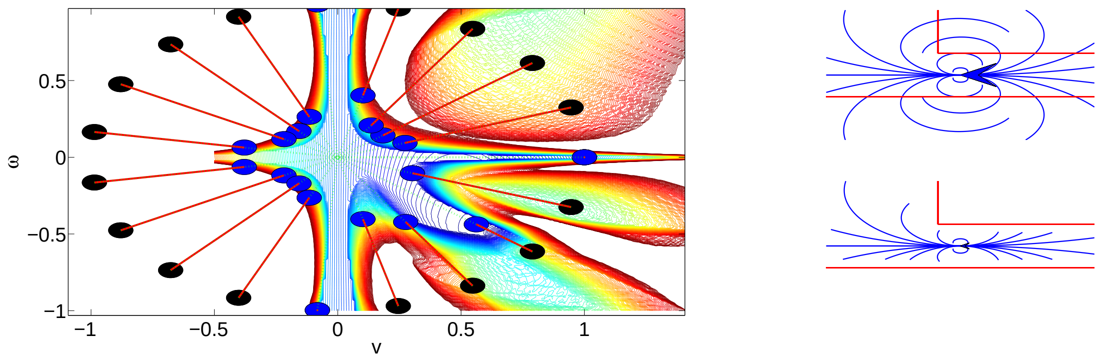


Figure 24: On the left is shown a contour plot of a cost when choosing a single (v, ω) pair. Shown on the contour plot are 20 points tested along a circle of radius 1 with a line connecting each point to a collision free pair of velocities executing the same arc, with a different speed. On the top-right is shown the initial arcs before scaling and on the bottom-right is shown the arcs after scaling with $\alpha = .8$. The straight lines denote walls.

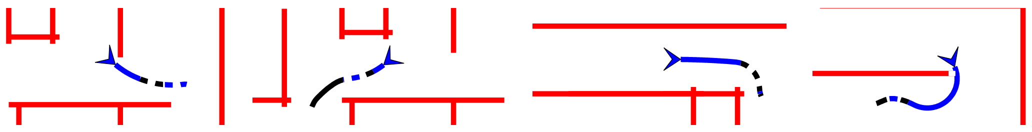


Figure 25: This figure demonstrates possible behaviors based on arc-based control. From left to right is shown a single arc, arc then reference follower, turning, and point-to-point behaviors. In each image the robot is represented as a triangle, with the resulting trajectory extending from it. The arcs are differentiated through color and line style, with the final reference tracking mode denoted as a solid line at the end of the trajectory (only visible in the arc-then-reference behavior). The straight lines denote walls.

space using a behavior-based approach. Multiple arcs can be placed back-to-back to create a desired behavior. Several examples of such behaviors that we found to be useful are illustrated in Figure 25. The illustrated examples consist of the following behaviors:

- Single-arc behavior: all arcs are given the same velocities and $\tau_N = \tau_{N+1}$.
- Arc-then-reference behavior: same as single-arc except $\tau_N < \tau_{N+1}$.
- Turn behavior: $\omega_0 = 0$ and the remaining arcs are designed to make a ninety-degree turn.
- Point-to-point behavior: using solutions to Theorem 15, arcs can be computed to navigate between any two points.

A quick search of the parameter space can then be accomplished by testing various iterations of each behavior. For the first two behaviors, a number of arcs with different curvature can be tested (we found 20 to work quite well). For the turn behavior, both clockwise and counter-clockwise turns can be tested as well as different lengths for the first, straight mode (we found 5 lengths to work well). Finally, the point-to-point behavior can be used by extracting desired points from the reference trajectory. As the behaviors are based on arcs, scaling can be used to quickly present a collision free trajectory as an option for initialization of the optimization step.

5.2 Control and Costs for Unicycle Motion Model

The unicycle model for motion in (5) provides a basic example where considering the nonholonomic constraints of a mobile robot becomes important. Simply planning in the position space is not sufficient as a robot with such a model is not capable of moving orthogonal to its direction of motion. It is also a useful model to examine as control laws can be designed for robots with more complicated dynamics which, after transients die out, cause the robot to move as if its dynamic model were the unicycle model. Results are not given in this section, but a reference trajectory tracking control law and costs are developed which are then used as a basis for the control and costs developed in Sections 5.3 and 5.4.

5.2.1 Reference Following Control

To define a valid control law to be used as the final mode, it is necessary that it satisfy the conditions imposed in Section 5.1. These conditions are the tracking condition in Assumption 12 as well as the dual-Mode MPC conditions in B3 and B4. Condition B1 is satisfied by choice of goal location, and B2 is trivially satisfied as no constraints have been put on the inputs.

To define a reference following control law satisfying these conditions, the concept of approximate ϵ-diffeomorphism presented in [67] is utilized. The idea being that, while the center of the robot has a nonholonomic constraint, a point directly in front of the robot does not. So, instead of controlling the actual position of the robot, the final control law will control

$$y = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \epsilon \begin{bmatrix} \cos(\psi) \\ \sin(\psi) \end{bmatrix},$$

where $\epsilon > 0$ is some pre-defined parameter as shown in Figure 26. Note that to simplify notation, we drop the time indexing on state, input, and reference variables throughout the remainder of the paper, except when required for sake of clarity.

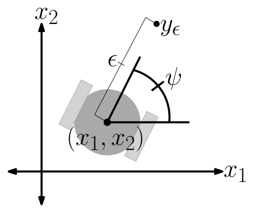


Figure 26: Shown is a diagram of a two-wheeled robot with the ϵ-point to be controlled.

A simplification of the development in [67] (i.e. controlling velocities instead of accelerations) leads to $\dot{y} = u_\epsilon$, where the inputs of the unicycle system can be solved for algebraically as

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\epsilon} \end{bmatrix} \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} u_\epsilon. \tag{106}$$

To control the unicycle to track a reference trajectory, the following controller can be used:

$$u_\epsilon(t) = \kappa_{N_\epsilon}(y(t), y_d(t)) = \dot{y}_d + k_p(y_d - y) \tag{107}$$

where $k_p > 0$ is constant and $\kappa_N(x, y_d)$ is given by combining (106) and (107) to obtain the commanded velocities. A theorem is now given on the satisfaction of the tracking condition of the control law.

Theorem 17. *The control law in (107) satisfies Assumption 12.*

Proof. The proof is accomplished by showing that $V(y(t)) = \frac{1}{2}||y_d(t) - y(t)||^2$ is a Lyapunov function. As such, $\dot{V}$ will be shown to always be decreasing. Therefore, if $\kappa_N(x(t), y_d(t))$ is executed starting at time τ_N and $y(\tau_N) \in \mathcal{B}_\delta(y_d(\tau_N))$, then $y(t) \in \mathcal{B}_\delta(y_d(t))$ $\forall t \geq \tau_N$ as the distance between $y_d(t)$ and $y(t)$ will always be decreasing as time increases.

$V(t)$ can be seen to be decreasing by evaluating the time derivative:

$$\dot{V} = (y_d - y)^T(\dot{y}_d - \dot{y}) = (y_d - y)^T(\dot{y}_d - u_\epsilon) = -k_p(y_d - y)^T(y_d - y) < 0 \; \forall y_d \neq y \tag{108}$$

□

To show that the control law will satisfy condition B3 and B4, an assumption on the terminal cost is made and a theorem is stated.

Assumption 15. *(Terminal Cost Convexity) The terminal cost is strictly convex in y with a unique minimum located at $y = y_{goal}$.*

Theorem 18. *Given Assumptions 13 and 15, $\Psi(y(t), y_{goal})$ satisfies B3 and B4 once $y_d(t) = y_{goal}$ when using the feedback control law κ_N.*

Proof. In the terminal region, $y_d(t) = y_{goal}$ is constant which allows (107) to be reduced to $\kappa_{N_\epsilon}(y(t)) = k_p(y_{goal} - y)$. The proof of Theorem 17 shows that $\mathcal{B}_\delta(y_{goal})$ is invariant under κ_{N_ϵ}, satisfying B3. To evaluate B4, $\Psi(y(t))$ needs to be evaluated as a candidate control-Lyapunov function under the control κ_{N_ϵ}. The time derivative can be evaluated as

$$\dot{\Psi} = \frac{\partial \Psi}{\partial y}(y)\dot{y} = k_p \frac{\partial \Psi}{\partial y}(y)(y_d - y) < k_p(\Psi(y) + \frac{\partial \Psi}{\partial y}(y_d - y)) \tag{109}$$

as $\Psi(y) \geq 0$ with strict inequality $\forall y \neq y_d$ due to Assumption 15. Using the global underestimator property of convex functions, we can further state:

$$\dot{\Psi} < k_p(\Psi(y_d)) = 0 \tag{110}$$

Therefore, $\dot{\Psi} < 0 \; \forall y \neq y_d$ and B4 is satisfied. □

5.2.2 Cost Definition

To define valid costs for the dual-mode arc-based MPC algorithm, the costs must satisfy several conditions in terms of the ϵ-point, y. The terminal cost must satisfy two conditions:

1. Assumption 11 gives a condition of a barrier around the reference position.
2. Assumption 15 and B4 give a condition on the convexity.

Similarly, the instantaneous costs must satisfy two conditions:

1. Assumption 10 requires the instantaneous cost to form a barrier to collisions.

2. Assumption 13 requires the instantaneous cost to approach zero as $y \longrightarrow \mathcal{B}_{\delta}(y_{goal})$.

The terminal cost is defined as:

$$\Psi(x(t), y_d(t)) = \frac{\rho_4}{2}||y(t) - y_d(t)||^2 + \rho_5(-\log(\frac{dist(y(t), \mathcal{B}_{\delta}(y_d(t))}{\delta})). \tag{111}$$

As both terms in $\Psi(x(t), y_d(t))$ are strictly convex with minimum at $y = y_d$, Assumptions 15 and B4 are satisfied. Moreover, Assumption 11 is satisfied as Ψ is defined as a log-barrier. Note that while the first term seems to be redundant, it is used to help find parameters in step 2 of Algorithm 2 when the terminal barrier cost is removed.

The instantaneous cost is used to ensure obstacle avoidance while moving at a desirable speed, v_d. To represent the obstacles we use a grid-based map where N_{obs} is the number of occupied grid points within a radius of d_{max} of the robot. It is assumed that the occupied grid points are available at time t as $o_1(t), ..., o_{N_{obs}}(t)$, where $o_i(t) \in \mathbb{R}^2$. This allows the cost to be written as

$$L(x(t), \theta_i, B_{free}(t_0)) = L_{goal}(||y(t) - y_{goal_\epsilon}||)\big(\frac{\rho_1}{2}(v_d - v(t))^2 + \frac{\rho_2}{2}\omega(t)^2\big) + \frac{\rho_3}{2}\sum_{i}^{N_{obs}} L_{avoid}(||y - o_i(t)||) \tag{112}$$

where

$$L_{goal}(d) = \begin{cases} \frac{2}{1+e^{-a(d-\delta)}} - 1 & d \geq \delta \\ 0 & d < \delta \end{cases}$$

$$L_{avoid}(d) = \begin{cases} -a\log(d - d_{min}) + a\log(d_{max} - d_{min}) & d_{min} < d \leq d_{max} \\ \infty & d \leq d_{min} \\ 0 & d > d_{max} \end{cases}$$

$\rho_i > 0$ is a weight, and d_{min} is the minimum allowed distance from an occupied grid cell to the robot.

The term L_{goal} is provided to smoothly reduce the influence of the instantaneous cost around the goal to satisfy Assumption 13 for the first two terms in the cost. Note, to completely satisfy these two assumptions, the goal position needs to be defined such that the goal is at least $d_{max} + \delta$ from the nearest obstacle so L_{avoid} is also zero in the terminal region. By inspection, the log-barrier cost L_{avoid} satisfies Assumption 10.

5.3 Case Study: Magellan Robot

The unicycle motion model and control presented in Section 5.2 can be used when the convergence time to the desired velocities is negligible. However, it is often the case that dynamic limitations cannot be ignored, especially as the speed of the robot increases. The Irobot Magellan-Pro, shown in Figure 27, presents an excellent example of such a scenario. We have observed the maximum translational acceleration to be approximately $.1\frac{m}{s}$, which becomes a significant factor in avoiding obstacles when traveling near the maximum velocity of $1\frac{m}{s}$.

The Magellan is also a prime candidate to demonstrate the dual-mode arc-based MPC algorithm due to its relatively limited computational capabilities. It has a Pentium III 850 Mhz processor, which is limited when compared to the state of the art. To develop the dual-mode arc-based MPC algorithm for the Magellan, this section begins with an explanation of the dynamic model to be used, develops a control law for reference tracking, and ends with experimental results.

5.3.1 First Order Model

While the kinematic constraints of the Magellan can be written as the unicycle model in (5), the velocities cannot be instantaneously realized. The Magellan-pro robot has an on-board

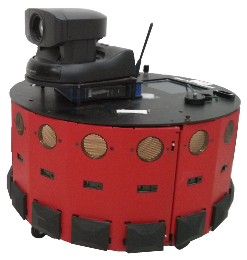


Figure 27: Shown is a picture of the IRobot Magellan-pro.

control which accepts velocity commands and limits accelerations. Thus, similar to [44], which modeled a UAV with a similar on-board control, we utilize a first order model on the velocity inputs written as follows:

$$f(x) = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\psi} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ \omega \\ a_1 v + b_1 u_1 \\ a_2\omega + b_2 u_2 \end{bmatrix}, \tag{113}$$

$$\text{s.t. } |\dot{v}| \leq a_{max}, |\dot{\omega}| \leq \alpha_{max}$$

where a_i, b_i, a_{max}, and α_{max} are constants.

5.3.2 Reference Following Control

To use the dual-mode arc-based MPC algorithm, two control laws must be defined: a control law to regulate the system to desired, constant velocities, and a controller to be used for reference tracking. The control law used to converge to desired, constant velocities is formed using an LQ approach, e.g. [9]. As this is a highly developed control methodology, the details are not presented. Rather, we focus our attention on the reference tracking control as it must satisfy the conditions imposed in Section 5.1. These conditions are given as the tracking condition in Assumption 12 as well as the dual-Mode MPC conditions given in B2, B3, and B4. Again, condition B1 is satisfied by choice of goal location.

To form a control law, we first ignore the acceleration constraints and control the velocities and then examine the control law to see when it can be employed. Let $e_{\bar{v}}$ be the error between $\bar{v} = [v, \omega]^T$ and the desired velocities, denoted as $\bar{v}_d$. Let the control be defined as follows

$$u_{\bar{v}} = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T = B^{-1}\big(\dot{\bar{v}}_d - A\bar{v}_d\big) - k_v e_{\bar{v}}, \tag{114}$$

$$\bar{v}_d = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\epsilon} \end{bmatrix} \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} u_\epsilon, \tag{115}$$

$$A = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix}, B = \begin{bmatrix} b_1 & 0 \\ 0 & b_2 \end{bmatrix},$$

$$\dot{\bar{v}}_d = \begin{bmatrix} -\omega \sin(\psi) & \omega \cos(\psi) \\ -\frac{\omega \cos(\psi)}{\epsilon} & -\frac{\omega \sin(\psi)}{\epsilon} \end{bmatrix} u_\epsilon + \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\frac{\sin(\psi)}{\epsilon} & \frac{\cos(\psi)}{\epsilon} \end{bmatrix} \dot{u}_\epsilon, \tag{116}$$

where u_ϵ is defined as in (107), and

$$\dot{u}_\epsilon = \ddot{x}_{d_\epsilon} + k(\dot{x}_{d_\epsilon} - y).$$

The time derivative of $e_{\bar{v}}$ can then be written as $\dot{e}_{\bar{v}} = (A - Bk_v)e_{\bar{v}} = \tilde{A}e_{\bar{v}}$ where k_v is a matrix of gains used for feedback. Thus, the error in velocities is exponentially stable to zero. As the desired velocities converge, the system behaves as the unicycle model of Section 5.2. To ensure that the required conditions are met, two additional assumptions are made:

Assumption 16. *The desired reference trajectory accelerations must be bounded below the acceleration limitations of the Magellan-pro robot.*

Assumption 17. *The velocities at time τ_N are sufficiently close to the desired velocities to allow convergence within $\delta_{execute}$ seconds .*

Assumption 16 corresponds to a condition that the time mapping must satisfy the translational acceleration constraint and a condition on the curvature of the path must satisfy the rotational acceleration constraint. Assumption 17 presents a relationship between the velocities at time τ_N, $\delta_{execute}$, and the convergence rate of the control, directly affected by choice of k_v in (114). Assumption 17 could be guaranteed through the introduction of additional cost barriers on the velocities. In practice, we have seen this to be unnecessary by choosing $v_d = ||\dot{y}_d(\tau_N)||$ for the final arc in the initialization step discussed in Section 5.1.3 and noting that ω converges quite quickly.

5.3.3 Results

To present the results for the implementation of the dual-mode arc-based MPC algorithm on the Magellan Pro robot, we perform a mix of simulation and actual implementation, where the desired velocity was set at .9 $\frac{m}{s}$, 90% of the Magellan's top speed. A simulated environment was projected onto the floor and the robot navigated through the environment as shown in Figure 28. To allow for high speeds, the environment continuously switched between two predefined environments. Along with a simulated environment, a simulated laser scanner is used as a sensor. The laser scanner takes 100 measurements ranging from $\psi - \frac{\pi}{2}$ to $\psi + \frac{\pi}{2}$. The robot is interfaced to a mapping algorithm through the ROS navigation package, using an A^* planning algorithm to find desired paths and the dual-mode arc-based MPC algorithm to follow the planned path.

To perform optimization, the techniques from Section 5.1.3 were employed. The NLopt optimization library, [37], together with the gradients in [19], were used to perform the gradient-based portion of the optimization. To test the efficacy of the different portions of the algorithm, several trials were run. First, the tracking control was run by itself, then the arc-based algorithm was performed without the gradient descent, and finally the gradient descent was incorporated and the remaining trials consisted of varying the alloted time for gradient descent (a parameter available in the NLopt library).

It is important to note that the tracking controller resulted in multiple collisions when running near the maximum velocity of the Magellan. This is quite possibly due to modeling errors, which are amplified at the top speeds. This was not a hindrance to the other trials as the tracking controller was never actually executed, due to the behavior-based initialization step to the optimization. Thus, for a fair comparison, the results for pure tracking control were obtained from simulation.

Results to the different trials are shown in Figure 29. Using solely the behavior-based portion of the optimization, the cost was reduced by 50% when compared to the reference

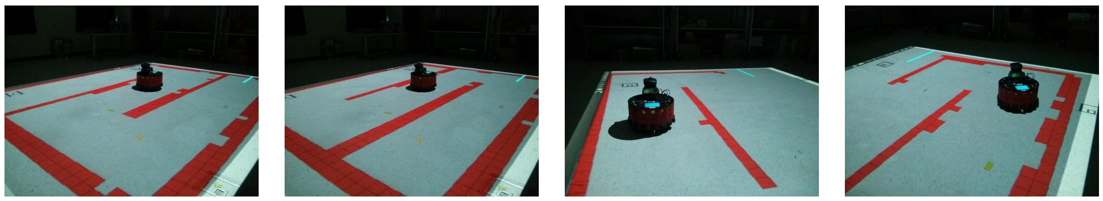


Figure 28: The left two images show the two environments completely mapped with the Magellan in the center. The right two images show the Magellan executing Algorithm 2. As the Magellan approaches the goal, the map is erased so that the Magellan can traverse the next environment as if it were unknown.

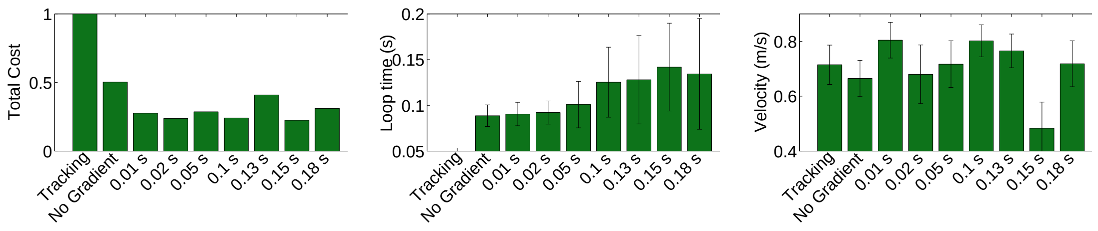


Figure 29: This figure shows results from the reference tracking control, optimization without gradient descent, and various times allowed for gradient-based optimization. From left to right is shown the total cost (normalized so that the reference tracking cost equals one), average execution time for a loop of Algorithm 2, and the average velocity for a window time excluding acceleration and deceleration periods. On both the middle and right plots is shown the standard deviation for each trial.

tracking control, with up to a 75% reduction in cost by including the gradient-based optimization. It can also be seen that there was no loss in average velocity when comparing the trajectory tracker with the arc-based MPC results. The highest average velocity seen being 92% of the desired $.9\frac{m}{s}$, despite the corridors being less than twice the width of the robot.

While one may expect to see the resulting overall cost to monotonically decrease with allowed optimization, this is not always the case. As the allowed gradient-descent time approaches $\delta_{execute} = .2$ seconds, the actual time to execute an iteration of Algorithm 2 increasingly exceeds $\delta_{execute}$, causing undesirable results. Also, while it is true that the cost at each time instant will decrease with increased optimization time, when considering the aggregate cost, this need not be the case. As the information available to the robot is limited, what may seem good at one point in time may prove to have been a poor choice at a future time. Thus, the costs shown in Figure 29 are those of local minima and cannot be truly compared beyond the point of noticing the trend that small time allowed for optimization appears to yield good results.

5.4 Case Study: Inverted Pendulum Robot

We now examine the problem of performing dual-mode arc-based MPC on the model of an inverted pendulum robot discussed in Section 4.3.1. This provides an example where consideration of the dynamics of the system is very important when planning for the action. Not only must the planner consider the nonholonomic constraints, it must also maintain balance. The arc-based MPC algorithm is applied to account for the complicated dynamics while maintaining stability and convergence guarantees. In this section we first develop a reference tracking control law for the inverted pendulum robot, discuss stability, and then present simulation results.

5.4.1 Control Laws

Two control laws must be defined to apply the dual-mode arc-based MPC algorithm. The first is a constant velocity control, which was designed in Section 4.3.1.2. The second is a reference following controller, which is the focus of this section.

As part of the development of the constant velocity control, a linearization of a subset of the states was performed to produce a controllable state vector with linear dynamics. From this linearization, the state vector, $\delta z = [v,\ \omega,\ \phi,\ \dot{\phi}]^T$ can be further divided into states to control the translational velocity, $z_v = [v,\ \phi,\ \dot{\phi}]^T$ and the rotational velocity, ω. The two state vectors can be controlled separately by allowing the control inputs to be:

$$u_1 = (\alpha + \beta),\ u_2 = (\alpha - \beta). \tag{117}$$

To control a time varying velocity, a potential problem arises as the same input must be used to both stabilize the pendulum and control the velocity. To overcome this, we control the tilt angle, which can in turn be used to control the velocity. This is made possible due to the fact that the tilt angle is much more responsive and that we can define the reference trajectory to maintain nearly constant translational acceleration. It is also worth noting that it is much easier to maintain stability of the pendulum by controlling the tilt angle to a desired value without any feedback on the velocity. The feedback on the velocity will come in the form of adjusting the desired tilt angle.

As controlling the desired tilt angle directly affects the translational acceleration, we slightly modify the approach for controlling y_ϵ given in Section 5.2. Instead of controlling $\dot{y}$, $\ddot{y}$ is controlled as $\ddot{y} = u_\epsilon$. This corresponds to the formulation for control presented in [67], namely

$$u_\epsilon(t) = \dot{y}_d + k_p(y_d - y) + k_d(\dot{y}_d - \dot{y}_\epsilon), \tag{118}$$

where k_d and k_p are constants. Similar to (116), the desired accelerations, $\dot{\bar{v}}_d = [\dot{v}_d \; \dot{\omega}_d]^T$, can be written as:

$$\dot{\bar{v}}_d = \begin{bmatrix} -\omega \sin(\psi) & \omega \cos(\psi) \\ -\frac{\omega \cos(\psi)}{\epsilon} & -\frac{\omega \sin(\psi)}{\epsilon} \end{bmatrix} \dot{y} + \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\frac{\sin(\psi)}{\epsilon} & \frac{\cos(\psi)}{\epsilon} \end{bmatrix} u_\epsilon. \tag{119}$$

Given the desired accelerations, we allow the inverted pendulum control to take the form

$$\begin{aligned} u_1 &= -k_\phi e_\phi - \frac{a_{43}}{b_{41}} \phi_d \\ u_2 &= \frac{\dot{\omega}_d}{b_{21}} \end{aligned} \tag{120}$$

where

$$\phi_d = \frac{b_{41} a_{13}}{a_{13}^2 b_{41} - b_{11} a_{43}} \dot{v}_d,$$

$$e_\phi = \begin{bmatrix} \phi - \phi_d \\ \dot{\phi} \end{bmatrix}$$

and k_ϕ is a feedback matrix. Assuming constant acceleration, the dynamics of the error for the linearized system becomes

$$\dot{e}_\phi = (\begin{bmatrix} 0 & 1 \\ a_{43} & 0 \end{bmatrix} - \begin{bmatrix} 0 \\ b_{41} \end{bmatrix} k_\phi) e_\phi,$$

which has exponential convergence rates to the desired tilt angle. Similar to the development in Section 5.3.2, Assumptions 16 and 17 can be made to ensure the convergence to the desired reference trajectory. Similarly, to ensure that the control law can converge to the necessary characteristics within $\delta_{execute}$ seconds, a cost barrier could be introduced. However, in the same fashion as mentioned in Section 5.3.2, we found that with the initialization step, this was not needed in practice.

5.4.2 Maintaining Balance

To ensure that the pendulum robot maintains balance, we analyze both the stability of the control law presented in the previous section as well as the ability for Algorithm 2 to maintain balance. While the previous section considered the linearized dynamics, this section considers the balancing on the full, nonlinear dynamics.

Under an assumption that $\dot{v}_d$ is constant, we can utilize the Lyapunov function $V = \frac{1}{2}e_\phi^T P e_\phi$ and evaluate numerically the space such that $\dot{V} = e_\phi^T P \dot{e}_\phi < 0$. Using the feedback gain matrix $k_\phi = [-8.5223 \ -.9922]$ and

$$P = \begin{bmatrix} 2.8929 & 0.0037 \\ 0.0037 & .0210 \end{bmatrix},$$

we found that $\dot{V} < 0 \; \forall \, e_\phi \neq 0$ for $|\phi| \leq .5$, $|\dot{\phi}| < 5$, and extreme initial conditions(i.e. values greater than we ever observed) of $v = 2.0$ and $\dot{\omega} = 2$ for all $|\phi_d| \leq 0.35$. To ensure stability, we place a bound on the desired tilt angle such that $|\phi_d| < .25$.

To ensure that the robot will maintain balance while executing Algorithm 2, a barrier cost is introduced as part of the instantaneous cost. The barrier cost takes the form:

$$-\rho_6 \log\left(\frac{\phi_{max}^2 - \phi(t)^2}{\phi_{max}^2}\right), \tag{121}$$

where $\phi_{max} = .5$ is the maximum allowable tilt angle and ρ_6 is a constant. This barrier cost will not permit step 4 of Algorithm 2 to produce a solution with a resulting $\phi(t) > \phi_{max}$ for some $t \in [t_0, t_0 + \Delta]$ if it is initialized with a solution that satisfies the constraint. Moreover,

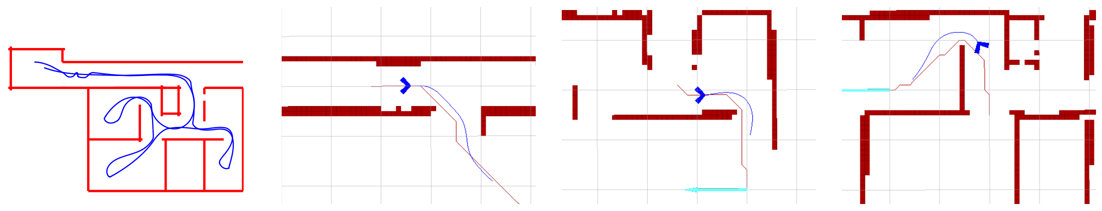


Figure 30: On the left is shown an executed path through the environment. The right three images show the robot and current map at different positions along the trajectory. The robot is shown as a triangle. The line extending from it is the trajectory created by the arc-based MPC framework.

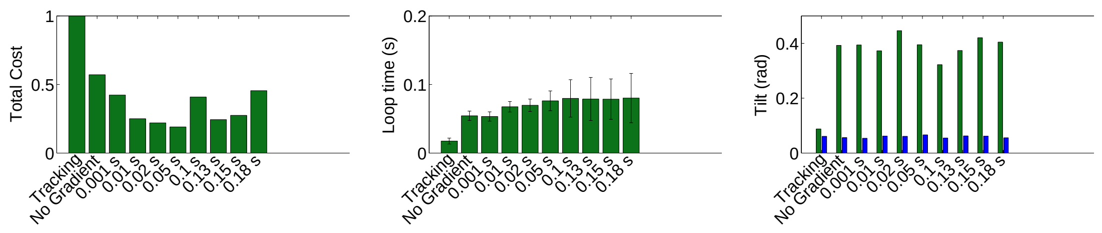


Figure 31: This figure shows results for the inverted pendulum robot using the reference tracking control, optimization without gradient descent, and various times allowed for gradient-based optimization. From left to right is shown the total cost (normalized so that the reference tracking cost equals one), average execution time for a loop of Algorithm 2, and average and maximum tilt angle for each trial.

the final control given in (120) will present an admissible solution with $\phi(t) < \phi_{max}\ \forall t \in [t_0, t_0 + \Delta]$ at the next iteration of the algorithm. All other costs discussed in Section 5.2.2 remain the same.

5.4.3 Results

We utilize the inverted pendulum robot to demonstrate the ability of dual-mode arc-based MPC to take a complicated dynamic model into account and ensure stability. Results were obtained on a simulation of the inverted pendulum robot through the environment shown in Figure 30 using a dual-core i7 2.67 GHz processor. The mapping and visualization was again performed using ROS, with a simulated laser range finder giving data to ROS to form the map. As before, several trials were performed to evaluate the performance of the dual-mode arc-based MPC algorithm.

Each trial consisted of navigating to a series of waypoints throughout the environment, with an A^* path planner being used to plan two dimensional paths between waypoints. The robot was given a desired speed of $1\frac{m}{s}$, with each trial having the robot approach that speed between waypoints (not shown for sake of brevity). The trials included using the reference tracking control, arc-based MPC without gradient descent, and using several limits on the allowed time for gradient descent. Results are shown in Figure 31.

Several trends can be observed on the results. Similar to before, the behavior-based portion of the optimization significantly reduces the cost when compared to the reference tracking control. The inclusion of a gradient-based optimization step provides even better results. By observing the loop time versus time allotted for gradient descent, it is apparent that convergence of the optimization at each time step is achieved rather quickly despite the complicated dynamic model. The average time to execute a loop of Algorithm 2 is less than $\frac{1}{2}\delta_{execute}$, even as the alloted time approaches $\delta_{execute}$. Most important to the development of this section, the robot maintained balance. The maximum tilt angle in each trial is below the tilt constraint, with the average being much smaller.

5.5 Conclusion

In this chapter we have developed a dual-mode arc-based MPC algorithm which ensures obstacle avoidance and guarantees convergence to a desired goal location despite complicated dynamic models. Dynamic motion constraints, limited accelerations, and stability concerns were considered in the examples without imposing unreasonable computational demands. The algorithm was applied to a decade old Irobot Magellan-pro, which maintained an average of 80% of its top speed while navigating through corridors less than twice the robot's width without collision. The ability of the algorithm to deal with more complicated dynamics, including stability concerns, was illustrated through the example of an inverted pendulum robot where it reached high speeds while traversing a complicated environment without exceeding a pre-specified maximum tilt angle.

CHAPTER VI

MULTI-AGENT DISTRIBUTED BEHAVIOR-BASED MPC

In this chapter, we combine concepts from distributed optimization with the behavior-based MPC to create a multi-agent behavior-based MPC formulation. The motion control framework allows a group of robots to work together to decide upon the motions by minimizing a cost. Agents are able to cooperatively optimize without any central computing component or any one agent performing a large portion of the computation.

As discussed in Section 1.4, two difficulties arise when extending MPC formulations to the multi-agent scenario. First, each agent must be able to either communicate or simulate neighboring agents actions. Second, a way must be formulated in which agents can influence the actions of neighboring agents in order to come to a collective minimum. It will be shown in this chapter that a behavior-based approach allows agents to simulate neighboring agents actions after communicating a small number of parameters. Also, agents will be able to influence each other by using parameter optimization techniques to optimize over these parameters.

In other words, the behavior-based MPC approach alleviates some of the main difficulties associated with distributed MPC by changing the problem into a distributed parameter optimization problem. However, a key element which makes multi-agent behavior-based MPC different from typical distributed parameter optimization is that a typical MPC cost not only depends on the current state, but on future states. Care must be taken to ensure that an agent is able to simulate other agents' state forward in time using solely information local to it in the network.

In this chapter we present the details of the multi-agent behavior-based MPC, addressing each of the mentioned areas of difficulty. In the next section we formulate preliminaries for multi-agent parameterized MPC and discuss how it allows agents to overcome communication difficulties. Section 6.2 will then detail how distributed optimization can be used to negotiate on future trajectories. In section 6.3, information requirements will be discussed to determine if a given network of communication is sufficient to perform distributed behavior-based MPC. The chapter will end with a simple example in Section 6.4 and concluding remarks in Section 6.5.

6.1 Preliminaries

This section will introduce the basic concept of multi-agent behavior-based MPC. This will be done by explicitly outlining the problem solved by the formulation. The ability for this method to overcome communication difficulties will then be addressed. Finally, notation used throughout the remainder of the chapter will be introduced.

6.1.1 Problem Definition

To explicitly represent the multi-agent scenario, assume that each agent's dynamics are of the form $\dot{x}_i(t) = f_i(x_i(t), u_i(t))$, where $x_i(t)$ is the state and $u_i(t)$ is the control input of agent i at time t. Also, assume that each agent will execute a feedback law of the form $\kappa_i(x_i(t), x^d_{-i}(t), \theta_i)$ where $x^d_{-i}(t)$ is a set of the states upon which agent i's dynamics depend and θ_i is a vector of tunable parameters. Without loss of generality, this will allow the dynamics to be expressed as:

$$\dot{x}_i(t) = f_i(x_i(t), x^d_{-i}(t), \theta_i). \tag{122}$$

In order to chose θ_i at each time instant, we consider the following general form for the collective cost to be minimized at each time step:

$$J = \sum_{i=1}^{N} J_i, \tag{123}$$

where J_i is the cost assigned to agent i and takes the form

$$J_i = \int_{t_0}^{t_f} L_i(x_i(t;t_0), x^c_{-i}(t;t_0), \theta^c_{-i})dt + \tag{124}$$
$$\Psi_i(x_i(t_f;t_0), x^c_{-i}(t_f;t_0`), \theta^c_{-i}),$$

subject to (122), where x^c_{-i} represents the set of agents that agent i is coupled to through its cost, θ^c_{-i} is the parameter vectors corresponding to those agents, t_0 is the initial time, $t_f = t_0 + \Delta$, and Δ is the time horizon being evaluated.

The division of the cost is left to the designer of the system. A method proposed in [81] recommends separating the cost into components which are not additively separable. However, the authors of [39] mention that adding extra terms in the cost can improve the results. For example, if the non-separable J_i was dependent on both x_j and x_k, then it may be good for convergence to also assign agent i any terms in J which deal exclusively with x_j and x_k. This will allow agent i to better model agents j and k.

The problem to be solved consists of a distributed optimization where agents must collaboratively minimize the summation of costs with respect to the parameter vectors and switch times. In other words, the problem takes the form:

$$\min_{\theta} J(\theta).$$

6.1.2 Communication of Trajectories

A key aspect of this formulation is the ability for agents to communicate entire trajectories by solely communicating a number of parameters. This is possible as the trajectory of each agent will be determined by its parameter vector, initial conditions, and the feedback control law it is executing. Therefore, to communicate a trajectory to a neighbor, an agent need only communicate these three pieces of information. The neighboring agent will then be able to calculate the trajectory of its neighbor by simulating it forward in time. Thus, there is an inherent tradeoff between communication and computation.

6.1.3 Notation

With the problem to be solved in mind, we outline the notation to be used throughout the chapter. First note that, as discussed in Chapter 2, distributed optimization requires each agent to have a local version of all the variables associated with the agents to which it is coupled. As such, we use the subscripts a_{ij} to denote agent i's version of agent j's variable a.

There are several groups of agents that each agent must keep track of. Denote $\mathcal{N}_i^d \equiv \{j : \frac{\partial f_i}{\partial x_j} \neq 0\}$ as the agents to which agent i is dynamically coupled. Similarly, $\mathcal{N}_{-i}^d \equiv \{j : \frac{\partial f_j}{\partial x_i} \neq 0\}$ denotes the agents that are dynamically coupled to agent i. The agents to which agent i is coupled to through cost is denoted as $\mathcal{N}_i^c \equiv \{j : \frac{\partial L_i}{\partial x_j} \neq 0 \text{ or } \frac{\partial \Psi_i}{\partial x_j} \neq 0\}$. The cost dependencies and dynamic dependencies are evaluated in Section 6.3 to determine set of agents from which agent i will require information, denoted as $\mathcal{N}_i^I$.

Furthermore, we can represent cost and dynamic dependencies through directed graph structures which will induce an undirected information dependency graph. We use $\mathcal{G}^c(V, E^c)$ to denote a graph where the node v_i corresponds to agent x_i and a directed edge $(v_j, v_i) \in E^c$ iff $j \in \mathcal{N}_i^c$. Similarly we use $\mathcal{G}^d(V, E^d)$ to denote the dynamic dependency graph where $(v_j, v_i) \in E^d$ iff $j \in \mathcal{N}_i^d$. In Section 6.3 we will then give conditions on $\mathcal{G}^c$ and $\mathcal{G}^d$ which will induce an undirected information graph $\mathcal{G}^I(V, E^I)$ where an edge $(v_i, v_j) \in E^I$ exists iff it is necessary for agents i and j to exchange information, thus building the set $\mathcal{N}_i^I$. An example of these three graphs is shown in Figure 32.

Each agent must also keep track of a number of state vectors. Allow $x_{ij}(t; t_0)$ to be the state of agent j at time t as simulated by agent i at time t_0. Denote agent i's versions of the states agent j depends on through agent j's dynamics as $x_{i,-j}^d(t; t_0) \equiv \{x_{ik}(t; t_0) : k \in \mathcal{N}_j^d\}$. Similarly, denote agent i's versions of the states agent j depends on through agent j's costs as $x_{i,-j}^c(t; t_0) \equiv \{x_{ik}(t; t_0) : k \in \mathcal{N}_j^c\}$. As each agent requires a vector of parameters in its control law, denote θ_{ij} as agent i's version of agent j's parameter vector. Finally, f_j denotes the dynamics used to compute x_{ij}.

For simplicity in presentation, we employ a number of abbreviations to the presented notation. x_{ij} or $x_{ij}(t)$ refers to $x_{ij}(t;t_0)$. $x^d_{i,-j}$ or $x^c_{i,-j}$ is used to refer to $x^d_{i,-j}(t;t_0)$ or $x^c_{i,-j}(t;t_0)$ respectively. f_{ij} is used to refer to $f_j(x_{ij}(t,t_0), x^d_{i,-j}(t;t_0), \theta_{ij})$. And also, $\bar{\theta}_i \equiv \{\theta_{ij} : j = 1, ..., N\}$: is used to denote all of the parameter vectors that agent i could depend on.

6.2 Multi-agent Distributed Parameterized MPC

The previous section presented problem formulation for multi-agent parameterized MPC and addressed how using parameterized feedback laws can facilitate the communication of trajectories between agents. In this section, it is shown that the use of these parameterized feedback laws also facilitates the negotiation between agents, allowing them to arrive at a collective minima. We begin by giving the gradients and end with a summary of the distributed parameterized MPC algorithm.

6.2.1 Gradients

As seen in the development of PI distributed optimization in Chapters 2 and 3, agents will be able to use the gradients of their individual costs to be able to collaborate with neighbors in the minimization of the collective cost. As such, we now provide a theorem giving the gradients used for optimization.

Theorem 19. *Given a cost J_i of the form in (124), the gradient of J_i with respect to θ_{ij} is given as*

$$\frac{\partial J_i}{\partial \theta_{ij}} = \xi^T_{ij}(t_0) \tag{125}$$

where

$$\dot{\xi} = \frac{\partial L_i}{\partial \theta_{ij}}^T - \frac{\partial f_{ij}}{\partial \theta_{ij}}^T \lambda_{ij};\ \xi(t_f) = \frac{\partial \Psi_i}{\partial \theta_{ij}}^T (t_f) \tag{126}$$

$$\dot{\lambda}_{ij} = -\frac{\partial L_i}{\partial x_{ij}}^T - \sum_{k \in \mathcal{N}^d_{-j}} \frac{\partial f_{ik}}{\partial x_{ij}}^T \lambda_{ik}; \tag{127}$$

$$\lambda_{ij}(t_f) = \frac{\partial \Psi_i}{\partial x_{ij}}^T (t_f)$$

Proof. We first augment (124) with the dynamics and write it as

$$\hat{J}_i(\bar{\theta}_i) = \int_{t_0}^{t_f} \Big(L_i(x_{ii}(t), x^c_{i,-i}(t), \theta^c_{i,-i}) + \tag{128}$$

$$\sum_{j=1}^{N} \lambda_{ij} \big(f_{ij}(x_{ij}(t), x^d_{i,-j}(t), \theta_{i,j}) - \dot{x}_{ij} \big) \Big) dt +$$

$$\Psi_i(x_{ii}(t_f), x^c_{i,-i}(t_f), \theta^c_{i,-i}).$$

Now, vary $\theta_{ij} \to \theta_{ij} + \epsilon\gamma_{ij}$ which causes the state to vary as $x_{ij} \to x_{ij} + \epsilon\eta_{ij}$. After applying traditional variational principles (e.g., [43]) and rearranging terms we can write

$$\frac{1}{\epsilon}\Big(J_i(\bar{\theta}_i + \epsilon\bar{\gamma}_i) - J_i(\bar{\theta}_i) \Big) = \tag{129}$$

$$= \sum_{j=1}^{N} \Bigg[\int_{t_0}^{t_f} \Bigg(\Big(\frac{\partial L_i}{\partial x_{ij}} + \sum_{k=1}^{N} \lambda_{ik}^T \frac{\partial f_{ik}}{\partial x_{ij}} + \dot{\lambda}_{ij}^T \Big) \eta_{ij} +$$

$$\Big(\frac{\partial L_i}{\partial \theta_{ij}} + \lambda_{ij}^T \frac{\partial f_{ij}}{\partial \theta_{ij}} \Big) \gamma_{ij} \Bigg) dt + \frac{\partial \Psi_i}{\partial \theta_{ij}}(t_f)\gamma_{ij}$$

$$- \lambda_{ij}^T \eta_{ij} \big|_{t_0}^{t_f} + \frac{\partial \Psi_i}{\partial x_{ij}}(t_f)\eta_{ij}(t_f) \Bigg] + o(\epsilon).$$

Note that $\frac{\partial f_{ij}}{\partial x_{ik}} = 0 \; \forall \, k \notin \mathcal{N}_j^d$ and $\eta_{ij}(t_0) = 0$. Then allow λ to be defined as (127). This permits the gradient for θ_{ij} to be expressed as

$$\frac{\partial J_i}{\partial \theta_{ij}} = \int_{t_0}^{t_f} \Big(\frac{\partial L_i}{\partial \theta_{ij}} + \lambda_{ij}^T \frac{\partial f_{ij}}{\theta_{ij}} \Big) ds + \frac{\partial \Psi_i}{\partial \theta_{ij}}(t_f). \tag{130}$$

Allowing ξ_{ij} to be defined as

$$\xi_{ij}^T(t) = \int_{t}^{t_f} \Big(\frac{\partial L_i}{\partial \theta_{ij}} + \lambda_{ij}^T \frac{\partial f_{ij}}{\theta_{ij}} \Big) ds + \frac{\partial \Psi_i}{\partial \theta_{ij}}(t_f). \tag{131}$$

We can differentiate (131) with respect to t to obtain the dynamics in (126) and the gradient in (125). □

6.2.2 MPC Framework

Having addressed distributed optimization in Chapters 2 and 3 as well presented the gradients in the previous section, we have two of the pieces needed in order to outline the multi-agent parameterized MPC algorithm below. The final piece, $\mathcal{N}_i^I$, will be given in the next section.

Before we introduce the algorithm, we mention a detail concerning the implementation. Agents will be simultaneously optimizing and moving. They actually optimize over values to be used in the future. Thus, when communicating, they communicate a future initial position and parameters they expect to be using beginning at a future point in time. The optimization occurs over a period of $\delta_{execute}$ seconds, at which point the agents update the parameters to execute. We denote the variables that agents are using for movement as $\bar{\theta}_{ii}$ and state the multi-agent behavior-based MPC algorithm in Algorithm 3.

Algorithm 3 Multi-agent Behavior-based MPC

```
1. Initialize:
    • Set $t_0$ to $t + \delta_{execute}$, where $t$ is current time.
2. Agent $i$ communicates following to each agent $j \in \mathcal{N}_i^I$:
    • Initial state $x_{ii}(t_0; t)$, where $t$ is current time.
    • Parameter vectors $\theta_{ij}$ and $\theta_{ii}$ .
3. Agents use distributed optimization to update variables and Lagrange multipliers.
4. Repeat steps 2 and 3 until $t = t_0$.
5. Agent $i$ updates parameters: $\bar{\theta}_{ii} = \theta_{ii}$.
6. Process repeated starting at step 1.
```

6.3 Induced Information Structure

While the previous sections have discussed the setup of the parameterized MPC scheme, this section will address whether or not a given network of communication will be capable of performing the distributed MPC scheme. Specifically, information requirements will be derived to determine whether or not the agents are capable of performing the algorithm with the information available to them in the network.

An information exchange between two agents will be necessary if either agent has an opinion about the other agent. This is expressed mathematically by determining whether or not each agent always has a non-zero gradient with respect to the variables associated with the other agent. Thus, the following theorem and corollary express the information required by each agent in the network.

Theorem 20. $\frac{\partial J_i}{\partial \theta_{ij}} = 0$ *for general costs and dynamics iff* $\forall\ k \in \mathcal{N}_i^c$ *there is no directed path in* $\mathcal{G}^d$ *pointing from* v_j *to* v_k.

Corollary 1. *Evaluating* $\mathcal{G}^I$ *presented in Section 6.1.3, one result of this theorem is that* $j \in \mathcal{N}_i^I$ *iff one of the following conditions hold:*

1. $\exists k \in \mathcal{N}_i^c$ *such that there is a path pointing from* v_j *to* v_k *in* $\mathcal{G}^d$.

2. $\exists l \in \mathcal{N}_j^c$ *such that there is a path pointing from* v_i *to* v_l *in* $\mathcal{G}^d$.

Proof. We expressly evaluate the costate $\lambda_{ij}(t)$ given in (127) as the arguments stating when $\lambda_{ij} = 0\ \forall t$ will likewise show that $\xi_{ij}(t) = 0\ \forall t$. First, note that the costate from (127) can be written as

$$\lambda_{ij}(t) = \Phi_{ij}(t, t_f)\lambda_{ij}(t_f) + \int_t^{t_f} \Phi_{ij}(t_f, s)B_j(s)ds, \tag{132}$$

where the state transition matrix, $\Phi_{ij}(t_f, t)$ holds the property

$$\frac{d}{dt}\Phi_{ij}(t_f, t) = A_j(t)\Phi_{ij}(t_f, t),$$

$$B_j(t) = -\frac{\partial L_i}{\partial x_{ij}}^T - \sum_{k \in \mathcal{N}_{-j}^d - \{j\}} \frac{\partial f_{ik}}{\partial x_{ij}}^T \lambda_{ik},$$

and $A(t) = \frac{\partial f_{ij}}{\partial x_{ij}}$. This can be verified using properties of the state transition matrix (e.g., [8]) and uniqueness of solutions to differential equations (assuming Lipshitz conditions hold), e.g, [40].

The proof comes in two parts. We first prove that if $\exists k \in \mathcal{N}_i^c$ such that a directed path pointing from v_j to v_k exists in $\mathcal{G}^d$ then $\frac{\partial J_i}{\partial \theta_{ij}} \neq 0$ for general dynamics and costs. We then prove that if such a path does not exist then $\frac{\partial J_i}{\partial \theta_{ij}} = 0$.

For the first part, we assume that $\exists k \in \mathcal{N}_i^c$ such that there exists a path pointing from v_j to v_k. We re-label the path of nodes as $\{v_{\sigma_1}, ..., v_{\sigma_m}\}$ such that $x_{i\sigma_1} = x_{ij}$ and $x_{i\sigma_m} = x_{ik}$. We also choose the path such that $\frac{\partial L_i}{\partial x_{i\sigma_l}} = 0 \ \forall l \neq m$. By definition of a path pointing from v_{σ_1} to v_{σ_m} in $\mathcal{G}^d$, we note that $\frac{\partial f_{i\sigma_{l+1}}}{\partial x_{i\sigma_l}} \neq 0$, for general dynamics.

We give a proof by induction: we show that if $\lambda_{i\sigma_{l+1}} \neq 0 \Rightarrow \lambda_{i\sigma_l} \neq 0$. We will then show that $\lambda_{i\sigma_m} \neq 0$ which completes the proof of the first part. By assumption, $\lambda_{i\sigma_{l+1}} \neq 0$ and $\frac{\partial f_{i\sigma_{l+1}}}{\partial x_{i\sigma_l}} \neq 0$ which means that $\frac{\partial f_{i\sigma_{l+1}}}{\partial x_{i\sigma_l}}^T \lambda_{i\sigma_{l+1}} \neq 0$. This implies that $B_l(t) \neq 0 \Rightarrow \lambda_{il} \neq 0 \Rightarrow \frac{\partial J_i}{\partial \theta_{il}} \neq 0$. We note that σ_m is defined such that $\frac{\partial L_i}{\partial x_{i\sigma_m}} \neq 0 \Rightarrow B_m(t) \neq 0 \Rightarrow \lambda_{i\sigma_m} \neq 0$ which completes the first part of the proof.

For the second part of the proof we assume that there does not exist a $k \in \mathcal{N}_i^c$ such that a path exists in $\mathcal{G}^d$ from v_j to v_k and prove that $\frac{\partial J_i}{\partial \theta_{ij}} = 0$. We define the set of indices that agent j connects to as $\mathbb{J} = \{l : \exists \text{ path from } v_j \text{ to } v_l \text{ in } \mathcal{G}^d\}$. For all $l \in \mathbb{J}$ we can simplify (132) as

$$\lambda_{ij}(t) = \int_t^{t_f} \Phi_{ij}(t_f, s) B_j(s) ds, \tag{133}$$

where

$$B_j(t) = -\sum_{k \in \mathcal{N}_{-j}^d - \{j\}} \frac{\partial f_{ik}}{\partial x_{ij}}^T \lambda_{ik}.$$

Therefore, the only way for $\lambda_{il}(t) \neq 0$ is if $\exists s > t$ such that $B_l(s) \neq 0$.

We now prove by contradiction that if $\lambda_{il}(t) = 0 \ \forall t \in (a, b)$ and $\forall l \in \mathbb{J}$ then $\lambda_{il} = 0$ $\forall t \leq a$ and $\forall l \in \mathbb{J}$. Assume that $\lambda_{il}(t) = 0 \ \forall t \in (a, b)$ and $\forall l \in \mathbb{J}$, but $\lambda_{im}(a) \neq 0$. This implies that $\exists \tau > a$ such that $B_m(\tau) \neq 0$. However, this is a contradiction as $B_m(\tau) \neq 0$ $\Rightarrow \exists k \in \mathcal{N}_{-m}^d \subset \mathbb{J}$ such that $\lambda_{ik}(\tau) \neq 0$. Now, note that we define $\lambda_{il} = 0 \ \forall l \in \mathbb{J}$ and $t \in [t_f, \infty) \Rightarrow \frac{\partial J_i}{\partial \theta_{ij}} = 0$ which completes the second part of the proof. $\square$

This shows the detrimental effect of dynamic dependence in multi-agent distributed

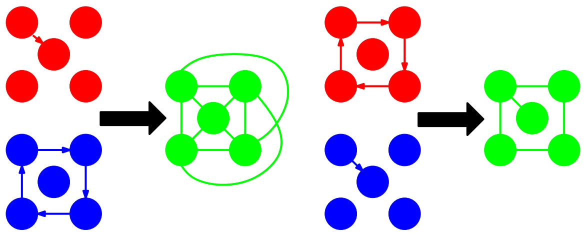


Figure 32: This figure shows how dynamic dependencies make the required communication increase as needed information propagates through the edges in dynamic dependency graph. Cost dependence ($\mathcal{G}^c$), dynamic dependence ($\mathcal{G}^d$), and the induced information graph ($\mathcal{G}^I$) are shown in red, blue, and green respectively

MPC. Dynamic dependencies induce the need for information from other agents. For example, if agent i depends dynamically on agent j and agent j depends dynamically on agent k, then agent i will have an opinion about agent k. Figure 32 shows a simple case where one agent requires information from the entire system despite having no dynamic dependence itself. On the other hand, when there are no dynamic dependencies, the information graph is simply the undirected cost graph. *Therefore, when coupling between agents is done only through the costs, it permits the amount of information required to be a function of the separability of the collective cost.*

6.4 Example

The task we set out to accomplish is to have the agents spread out on a circular orbit while maintaining a certain velocity. Assuming integrator dynamics, i.e. $\dot{x} = u$, the agents can achieve orbiting by following a vector field. We again utilized the orbiting vector field from Chapter 4 given in (86) and illustrated in Figure 13. We set the desired convergence rate in (86) as a constant and have agents solely optimize over each their commanded velocity.

To maximize distance between agents while maintaining a desired speed each agent is assigned the following cost:

$$J_i = \int_{t_0}^{t_f} \frac{\rho_1}{2}(\theta_i - v_d)^2 dt + \rho_2 \sum_{j \in \mathcal{N}^c} \sum_{k \in \mathcal{N}^c, k \neq j} \exp\Big(-\rho_3 (x_j - x_k)^T (x_j - x_k)\Big) \qquad (134)$$

where ρ_1 through ρ_3 are weights. This cost will allow each agent to adjust it's speed so that, at the end of the time window, it is as far away from its neighbors as possible.

This example was simulated under four different distributed optimization scenarios: dual-decomposition, consensus-based with a constant step-size, PI, and no negotiation (i.e. agents solely use their own cost for gradient descent). In each scenario, the agents effectively used $\delta_{execute} = 0$ in Algorithm 3 to illustrate the convergence qualities of the distributed optimization algorithms and to show that very little optimization time can be used to achieve desirable results.

The results of the simulations can be seen in Figures 33, 34 and 35. When not using a distributed optimization technique, the agents are not able to spread out effectively. They are essentially performing a greedy optimization, doing what seems best for them without any concern for the collective whole. Because of the communication topology, adjacent agents cannot see each other and thus act without knowing that they are getting close to each other.

Much better results are seen when using distributed optimization. Despite not being able to see or speak to neighboring agents, agents in the network help each other to avoid getting close to each other and effectively spread out around the circle. Similar convergence characteristics to the examples in Chapters 2 and 3 are observed. The consensus method has a damped convergence with the optimal not reached, dual-decomposition has an oscillatory

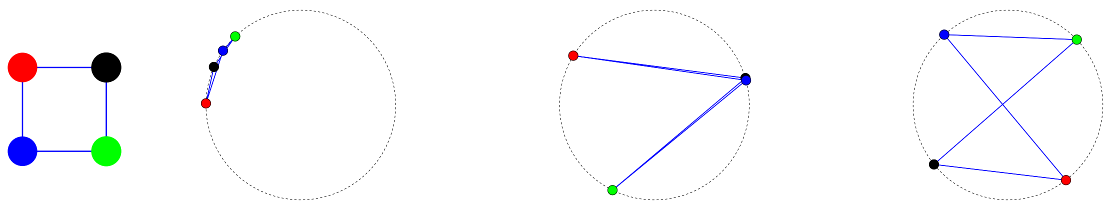


Figure 33: This illustrates the utility of the proposed MPC framework. From left to right is shown $\mathcal{G}^I$, the starting configuration, result of agents not negotiating, and the result of agents using dual-decomposition to negotiate. The result of consensus with constant step size is not shown as it is very similar to using dual-decomposition. The result of consensus with diminishing step-size is not shown as it does not converge.

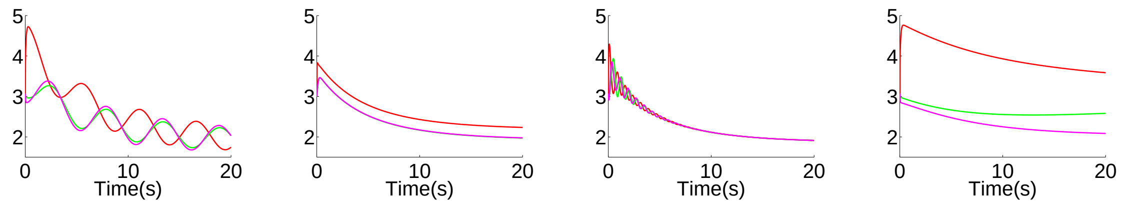


Figure 34: This figures shows each agents opinion of what the velocity should be of the blue colored agent. From left to right are the results from dual-decomposition, consensus with constant step-size, PI, and no distributed optimization.

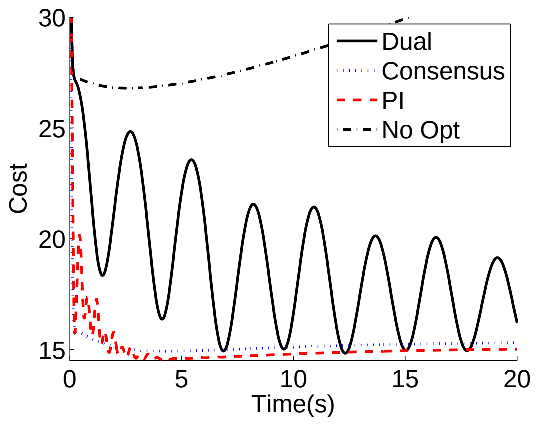


Figure 35: This figure shows the resulting cost versus time for each of the four simulations, namely, distributed optimization using dual-decomposition, the consensus based method with constant step-size, the consensus method with diminishing step size, and no distributed optimization.

response, and PI technique oscillates until agreement at which time its convergence looks very similar to the consensus method.

6.5 Conclusion

In this chapter we have presented a framework based on parameterized feedback control laws for performing distributed MPC of networked multi-agent systems. This allows for the communication and optimization of state trajectories in a distributed manner. To perform distributed MPC we characterized the information necessary for each agent to perform the optimization at each step and found that dynamic dependencies cause required information between agents to propagate down dynamic dependencies whereas cost dependencies did not. We presented an example using this framework which showed that the performance can be greatly improved versus a greedy approach without negotiation.

CHAPTER VII

DISTRIBUTED VIRTUAL LEADER FORMATION CONTROL

Formation control could be considered a canonical problem in multi-agent systems. Solving the formation control problem requires agents to work together in a distributed manner to cooperatively produce a desired, collective outcome. A successful moving formation control framework must consider the motion constraints of both the individual agents and the collective whole while respecting the communication constraints imposed by the network.

In this chapter we present a moving formation control example as the culmination of the concepts presented in previous chapters. In Chapter 3, formation control was discussed in terms of a distributed optimization problem. Agents optimized over different parameters that affect the structure of the formation allowing for a desirable collective outcome for a stationary formation. In Chapter 6, a behavior-based MPC framework was established to allow agents to consider their future states and adapt parameters appropriately. In Chapter 5, arc-based motions were discussed as a possible motion primitive for natural movement of a wheeled robot.

We combine these aspects to develop a virtual-leader formation control example. The virtual-leader executes an arc-based motion which can be exploited by follower agents to stay in position while the formation moves. Agents in the network negotiate over both the motion of the virtual leader and the parameters affecting the structure of the formation. Thus, the movement and structure of the network is maintained in a completely decentralized fashion. Network communication restraints are respected and no single agent performs a significant amount of the computation. Using the behavior-based multi-agent MPC framework, agents simulate their own and neighboring agents' actions into the future to better maintain the structure of the formation while avoiding collisions with obstacles.

To present the distributed virtual leader approach to formation control, the remainder of the Chapter will proceed as follows: After a brief introduction to leader-based formation control in Section 7.1, the behavior to be executed by the individual agents will be developed in Section 7.2. Then, in Section 7.3, the interactions between agents and the control of the collective formation will be discussed. The chapter will end with an example in Section 7.4 and concluding remarks in Section 7.5.

7.1 Virtual Leader Formation Control Background

As the formation control problem includes many intriguing difficulties, many approaches have been developed. In this section we discuss three classes of formation control problems which directly affect our virtual-leader approach.

The first class to be considered consists of local or relative approaches. The formation can be defined in terms of local interactions between agents, such as inter-agent distances (see, for example, [57, 59, 30] and the references therein). Once the agents know where they should be relative to neighboring agents, control laws can be designed to ensure that the agents settle into the correct position. The approach has the advantage that agents need not know where they are. They solely rely on information local to them, such as distance to neighboring agents. However, an apriori analysis of the network must be made to ensure the desired structure of the formation is maintained. Basically, each agent must have enough interactions with neighboring agents to maintain its position rigid inside the structure of the formation.

Relaxing the conditions on the underlying graph structure often comes at the cost of requiring global position measurement (see, for example [50] and the references therein), which is the approach taken in this chapter. Methods of course exist for relaxing global position requirements. In [50], knowledge of a leader's motion along with the relative position of the leader with respect to an agent is enough for that agent to maintain its correct position in the formation. Generalizing this idea, it is also conceivable that distributed estimation, e.g. [78, 73], could be utilized to have only a number of informed agents, although this is out of the scope of the presented work.

The second approach we consider as a background for our work is that of leader based control. One or more agents can be informed with global position information and act accordingly, e.g. [57]. Other agents then use controllers designed to follow the leader agents. To be able to incorporate the performance of the agents in maintaining the formation structure, both [65] and [91] developed constructions where a virtual leader and structure is introduced to the system. The virtual leader is basically a central computing component capable of communicating with all agents in the network. The virtual structure is not unlike the construction in Chapter 3 where knowledge of the virtual leader position and orientation allows agents to move into a desired position in the formation. In this fashion, feedback can be introduced into the moving formation as the virtual leader can design its motion to accommodate each individual agent.

The final approach we consider is that of an MPC framework (see for example [39] and the references therein). In [39], agents cooperatively perform MPC to minimize the error between relative distances and desired displacements. The MPC formulation allows for a wide array of motions by the agents and the collective whole. Guarantees, such as collision avoidance, are provided through constraints in the optimization problem. Adaptability was a key factor emphasized in [39] as they argued that local control laws are difficult to update on the fly.

We incorporate aspects from each of these approaches to design a behavior-based MPC framework. First, control laws are designed to ensure agents converge to their respective position in the network relative to a leader agent. To avoid a central computing component, we use a virtual leader controlled through distributed optimization. Behavior-based MPC is employed to control the motions of the agents through parameters affecting the structure of the formation as well as parameters controlling the motion of the virtual leader.

7.2 *Arc-based Leader-follower Control*

A behavior-based design allows the system to act collectively as a whole without a need for excessive communication. A key element in the behavior-based MPC formulation is the underlying behavior that agents will be executing. To implement a virtual leader behavior-based MPC formulation, this section focuses on the design of the behavior to be used by each agent to maintain its desired position in the network.

To design a leader-follower control, we take concepts from Chapter 5 in which arc-based motion is exploited to simplify the motion problem. In this section, a controller will be designed to allow a follower agent to stay in formation with a leader that is executing a constant arc motion. The control law will be developed using the diffeomorphism control approach presented in Chapter 5 for reference following. This immediately allows for a proof of ϵ-tracking of the desired position. We then show that the designed control can actually achieve perfect tracking. As a final aspect to the behavior-based motion control design, a term is included for obstacle avoidance.

7.2.1 Epsilon-tracking Formation Control

To define the desired ϵ-tracking controller for a follower agent, we first examine the motion of the leader agent. The desired follower motion can then be derived. From the follower agent's desired motion, the desired motion of the ϵ-point can be defined which will lead to the definition of an ϵ-tracking controller. A diagram is provided in Figure 36 for the different positions and motions being considered.

7.2.1.1 Leader Motion

The leader is defined such that it has a position, q_l[1], and orientation, ψ_l. It is assumed that the leader agent executes an arc motion with the unicycle motion model in (5). We

[1]Note that we use a slight change of notation in this chapter due to the fact that there are various positions to keep track of. Thus, we use q to denote a position with subscripts to distinguish between positions. We continue to use ψ to denote orientation, with the subscripts again used to clarify which orientation.

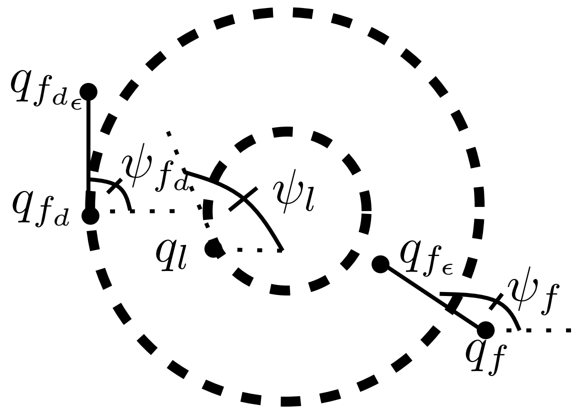


Figure 36: This diagram shows the leader position, q_l, and orientation, ψ_l, the actual and desired follower positions, q_f and q_{f_d}, and orientations, ψ_f and ψ_{f_d} as well as the actual and desired follower ϵ-points, q_{f_ϵ} and $q_{f_{d_\epsilon}}$. The leader position and desired follower position are located on circles as the motion of each will be on circles with the same center, but different radii.

redefine the unicycle motion model using a unit vector $h_\psi = [\cos(\psi),\ \sin(\psi]^T$, to denote the direction of motion as this will permit a concise development of the follower motion. This allows the motion of the leader to be defined as

$$\begin{bmatrix} \dot{q}_l \\ \dot{\psi}_l \end{bmatrix} = \begin{bmatrix} v_l h_{\psi_l} \\ \omega_l \end{bmatrix}, \tag{135}$$

where $v_l > 0$ and ω_l denote the translational and rotational velocities of the leader agent.

As depicted in Figure 36, executing constant v_l and ω_l corresponds to the leader executing a circular motion. The radius of the circle can be written as $\rho_l = \frac{v_l}{|\omega_l|}$, with a center located at

$$c = \frac{v_l}{\omega_l} J h_{\theta_l} + q_l, \tag{136}$$

$$J = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix},$$

where J corresponds to the ninety degree rotation matrix. The center, c, can be directly extracted using the solution to the unicycle motion model in Theorem 15 of Chapter 5.

7.2.1.2 Desired Follower Motion

Associated with the follower agent is both its actual and desired positions, written as q_f and q_{f_d}, respectively, with corresponding orientations ψ_f and ψ_{f_d}. It is assumed that the follower agent also has a unicycle motion model with velocities v_f and ω_f. In this section we focus on the desired motion of the follower agent, with a discussion of the actual motion in Section 7.2.1.4.

A relationship between the leader position and the desired follower position can be defined using a number of parameters to represent the structure of the formation. As discussed in Chapter 3, the formation can be defined in terms of a rotation, scaling, nominal follower position, and translation. To incorporate the virtual leader into the formation, its orientation is considered part of the rotation and its position is considered as the translation. This allows the desired follower position to be defined as

$$q_{d_f} = R(\psi_l + \Psi)\gamma\tau_f + q_l, \tag{137}$$

where Ψ is a scalar affecting the rotation of the formation with respect to the leader, γ is a scalar representing the scaling of the formation, and τ_f is the nominal position of follower agent when the leader is at the origin and $\psi_l = 0$.

Assuming that τ_f, γ, Ψ, v_l, and ω_l are all constant, the desired position of the follower agent will always be located at a constant offset from the leaders point of view. This corresponds to the desired position making one circular loop around the center of the leader's circle each time the leader makes one loop, although it may loop at a different radius. As the desired position is moving around a circle, the desired orientation can immediately be computed. Define the angle off the x-axis pointing to the center of the circle from q_{f_d} as $\phi_{f_d} = atan2(c_2 - q_{f_{d2}}, c_1 - q_{f_{d1}})$. The desired orientation of the follower can be extracted by rotating ϕ_{f_d} by $\pm\frac{\pi}{2}$, depending on which way the leader is moving around the circle.

This allows the orientation to be expressed as

$$\psi_{d_f} = \begin{cases} \psi_l & \omega_l = 0 \\ -sign(\omega_l)\frac{\pi}{2} + \phi_{f_d} & \text{otherwise} \end{cases}. \tag{138}$$

Note that the $sign(\omega_l)$ term in the calculation of ψ_{d_f} accounts for the direction of orbit around the center point, as depicted in Figure 37.

As the desired position of the follower agent is also moving around a circle, the motion of the desired follower position can be defined as executing a unicycle motion model with constant translational and rotation velocities, denoted as v_{f_d} and ω_{f_d}. The value for ω_{f_d} can immediately be deduced from the fact that the desired position is moving around a circle at the same frequency as the leader agent. As the orientation is always perpendicular to the circle, the desired orientation of the follower changes at the same rate of the leader, i.e. $\omega_{f_d} = \omega_l$. The translational velocity of the follower agent can then be extracted from the radius of curvature and the rotational velocity as

$$v_{f_d} = \begin{cases} v_l & \omega_l = 0 \\ \rho_{f_d}|\omega_l| & \text{otherwise} \end{cases}, \tag{139}$$

where ρ_{f_d} denotes the radius of the circle and can be calculated as $\rho_{d_f} = ||q_{d_f} - c||$, which is constant for constant ω_l and v_l. Similar to the leader motion defined in (135), the desired

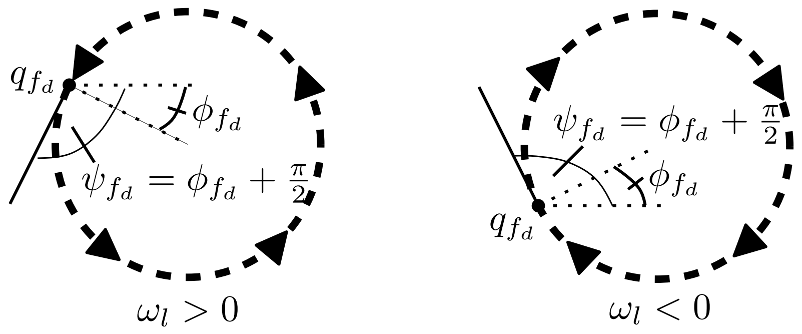

q_{f_d}
ϕ_{f_d}
$\psi_{f_d} = \phi_{f_d} + \frac{\pi}{2}$
$\omega_l > 0$
$\psi_{f_d} = \phi_{f_d} + \frac{\pi}{2}$
ϕ_{f_d}
q_{f_d}
$\omega_l < 0$

Figure 37: The left and right images show the desired orientation of the follower for $\omega < 0$ and $\omega > 0$ respectively. The arrow-heads on the circles denote the direction of motion around the circle.

follower motion can be written as:

$$\begin{bmatrix} \dot{q}_{f_d} \\ \dot{\psi}_{f_d} \end{bmatrix} = \begin{bmatrix} v_{f_d} h_{\psi_{f_d}} \\ \omega_{f_d} \end{bmatrix}. \tag{140}$$

7.2.1.3 *Desired ϵ-point Motion*

Recalling the discussion in Section 5.2.1, the center of the robot has a non-holonomic constraint, but a point directly in front of the center of the robot has no such constraint (refer to Figure 26 of Chapter 5 for an illustration of the relationship between the actual point and ϵ-point). For this reason, it is important to define the motion of a point directly in front of the desired position, referred to as the desired ϵ-point.

The positions of the desired ϵ-point is denoted $q_{f_{d_\epsilon}}$. It can be expressed in terms of q_{f_d} and ψ_{f_d} as:

$$q_{f_{d_\epsilon}} = q_{f_d} + \epsilon h_{\psi_{f_d}}.$$

The motion of the desired ϵ-point is denoted as $\dot{q}_{f_{d_\epsilon}}$. Noting that $\dot{h}_\psi = \omega J h_\psi$, the desired motion of the ϵ-point can be written as:

$$\dot{q}_{d_{f_\epsilon}} = v_{d_f} h_{f_d} + \epsilon \omega_{f_d} J h_{f_d}. \tag{141}$$

Also, note that $\ddot{q}_{d_{f_\epsilon}}$ is used in a proof of convergence, so (141) can be differentiated to obtain:

$$\ddot{q}_{d_{f_\epsilon}} = v_{d_f} \omega_{d_f} J h_{f_d} - \epsilon \omega_{f_d}^2 h_{f_d}. \tag{142}$$

7.2.1.4 *ϵ-Point Control*

We are now ready to define the desired control law to be used as the leader-follower behavior. The same control scheme in Chapter 5 is followed to control the desired ϵ-point. Denote the position of the ϵ-point as q_{f_ϵ}. It can be calculated from the position and orientation of the follower as:

$$q_{f_\epsilon} = q_f + \epsilon h_{\psi_f}.$$

As discussed in Section 5.2.1, assuming a unicycle motion model allows the time derivative of the ϵ-point to be directly controlled. A proportional control law can then be used to ensure ϵ-tracking of the desired trajectory. As a review, this means that the ϵ-point will converge at an exponential rate to its desired trajectory, causing the robot to converge to a distance of ϵ away from the trajectory. We give the control law again using the notation presented in this chapter:

$$\dot{q}_{f_\epsilon} = \dot{q}_{d_{f_\epsilon}} + k_p(q_{d_{f_\epsilon}} - q_{f_\epsilon}), \tag{143}$$

The commanded velocities for the follower agent can be calculated directly from $\dot{q}_{f_\epsilon}$ as:

$$\begin{bmatrix} v_f \\ \omega_f \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\epsilon} \end{bmatrix} \begin{bmatrix} \cos(\psi_f) & \sin(\psi_f) \\ -\sin(\psi_f) & \cos(\psi_f) \end{bmatrix} \dot{q}_{f_\epsilon}, \tag{144}$$

where v_f and ω_f are the commanded translational and rotational velocities of the follower agent.

7.2.2 Perfect Tracking Using Approximate Control

As previously mentioned, the controller in (143) will achieve ϵ-tracking of $q_{f_{d_\epsilon}}(t)$. This results in the actually position of the robot, $q_f(t)$, to converge at an exponential rate to a distance of ϵ away from $q_{f_{d_\epsilon}}(t)$. Due to the fact that we have designed $q_{d_\epsilon}(t)$ to be a distance of ϵ away from the desired point of the robot, this gives us a guarantee that the robot will converge to within 2ϵ of its desired position on the reference trajectory, still achieving ϵ-tracking, albeit at a distance of 2ϵ. We now show that, despite the fact that the robot is using an approximate-diffeomorphism controller, it can actually achieve perfect tracking.

As the robot achieves ϵ-tracking, the set of possible positions of the robot at time t forms a circle of radius ϵ around the point $q_{f_\epsilon}(t)$, as depicted in Figure 38. As mentioned, the worst possible position at any given time for the robot would be a distance of 2ϵ from the desired point, $q_{f_d}(t)$. But $q_{f_d}(t)$ also exists on the circle, making it possible that $q_f(t) = q_{f_d}(t)$.

Since we know that the position of the robot will converge to the circle with its orientation pointed towards the center, we need only to evaluate the orientation of the robot in

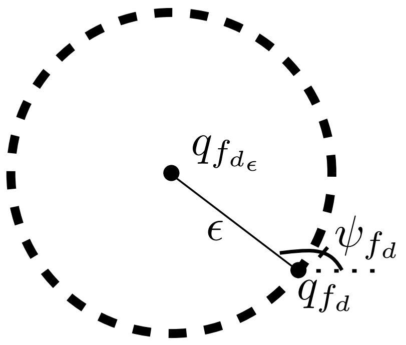


Figure 38: This figure shows the set of all positions that could satisfy $q_{f_\epsilon} = q_{f_{d_\epsilon}}$. The set forms a circle of radius ϵ, with q_{f_d} lying on the circle and ψ_{f_d} pointing towards the center of the circle.

the constraint set. The reason being is that each value of $\psi_f \in [-\pi,\, \pi)$ will correspond to a unique point on the circle. So if $\psi_f(t) = \psi_{f_d}(t)$ then we know that $q_f(t) = q_{f_d}(t)$. Therefore, we evaluate the following candidate Lyapunov function:

$$V = -h'_{\psi_f} h_{\psi_{f_d}} + 1, \tag{145}$$

which is zero for $\psi_f(t) = \psi_{f_d}(t)$ and positive for $\psi_f(t) - \psi_{f_d}(t) \in (-\frac{\pi}{2},\, \frac{\pi}{2})$. Taking the time derivative of V we obtain:

$$\begin{aligned}
\dot{V} &= -\dot{h}'_{\psi_f} h_{\psi_{f_d}} - h'_{\psi_f} \dot{h}_{\psi_{f_d}} \\
&= -\omega_f (J h_{\psi_f})' h_{\psi_{f_d}} - \omega_{f_d} h'_{\psi_f} J h_{\psi_{f_d}} \\
&= -\omega_f h'_{\psi_f} J' h_{\psi_{f_d}} - \omega_{f_d} h'_{\psi_f} J h_{\psi_{f_d}} \\
&= -\frac{1}{\epsilon} (J h_{\psi_f})' q_{d_\epsilon} h'_{\psi_f} J' h_{\psi_{f_d}} - \omega_{f_d} h'_{\psi_f} J h_{\psi_{f_d}} \\
&= -\frac{1}{\epsilon} (J h_{\psi_f})' (v_{f_d} h_{\psi_{f_d}} + \epsilon \omega_{f_d} J h_{\psi_{f_d}}) h'_{\psi_f} J' h_{\psi_{f_d}} - \omega_{f_d} h'_{\psi_f} J h_{\psi_{f_d}} \\
&= -\frac{v_{f_d}}{\epsilon} (h'_{\psi_f} J' h_{\psi_{f_d}})^2 + (1 - h'_{\psi_f} h_{\psi_{f_d}}) \omega_{f_d} h'_{\psi_f} J' h_{\psi_{f_d}}
\end{aligned} \tag{146}$$

The first term is always negative, but the second term is not. However, we can capitalize on the fact that $h'_{\psi_f} h_{\psi_{f_d}} \approx 1$ around $\psi_f = \psi_{f_d}$. So, for some region around $\psi_f = \psi_{f_d}$, $V \leq 0$ with strict inequality for $\psi_f \neq \psi_{f_d}$. Thus, the $\frac{v_{f_d}}{\epsilon}$ term will dominate for some region around the origin and the bigger the value of $\frac{v_{f_d}}{\epsilon}$ in comparison to ω_{f_d}, the larger the region of convergence.

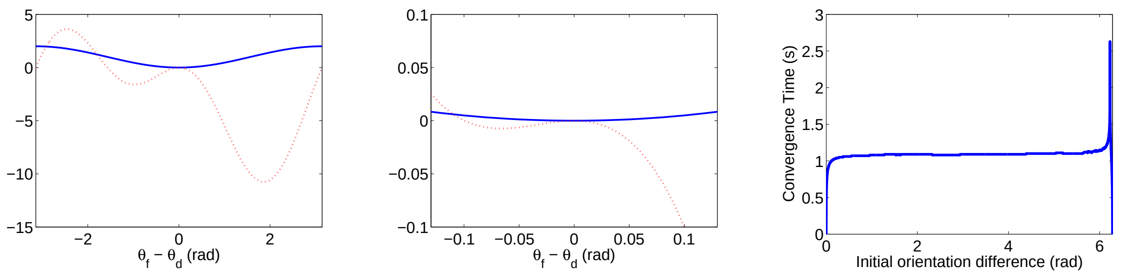


Figure 39: The left and middle images show the Lyapunov function, V, given in (145), as well as its derivative, $\dot{V}$. V is plotted as a solid line and $\dot{V}$ is plotted as a dotted line. The left image shows the results for $v_{f_d} = .5$, $\epsilon = .1$, and $\omega_{f_d} = 5$ and the middle shows results for $v_{f_d} = .5$, $\epsilon = .1$, and $\omega_{f_d} = 100$. The right image shows convergence times for the values in the middle image for all possible deviations in orientation, showing that V may be conservative.

Depicted in Figure 39 is the Lyapunov function V and its time derivative $\dot{V}$ for values of $v_{f_d} = .5$, $\epsilon = .1$, and $\omega_{f_d} = 5$. Note that $\omega_{f_d} = 5$ is quite extreme compared to what was seen in the experiments. The region of convergence derived from V for the mentioned values is $\psi_f - \psi_{f_d} \in [-1.79,\ \pi)$. We note that the region of convergence determined by V also seems quite conservative. For the very extreme case of $\omega_{f_d} = 100$, $v_{f_d} = .5$, and $\epsilon = .1$, Figure 39 also depicts the time for convergence to within 1% of the desired value for $\psi_f - \psi_{f_d} \in [-\pi,\ \pi]$. Convergence is always achieved.

7.2.3 Obstacle Avoidance Control

Much like we have capitalized on behavior-based design to ensure an agent can follow another, we can also use behavior-based design to ensure that the agents avoid obstacles. The overall obstacle avoidance scheme will include a term in the cost for agents to avoid getting near obstacles, but we include a term in the controller as somewhat of an "emergency" measure for when the gradient-based optimization does not have all agents completely avoid the obstacles.

To avoid obstacles, we use the vector-field approach given in [2] in conjunction with the ϵ-point controller discussed in the previous sections. We assume there are N_{obs} circular obstacles of radius R, where the two-dimensional center points are available to each agent as $o_1, ..., o_{N_{obs}}$. The controller assumes a sphere of influence, S, around the obstacle where the avoidance behavior will influence the motion of the robot. The avoidance controller for the follower agent can then be stated as:

$$u_{avoid} = \sum_{j=1}^{N_{obs}} A(||q_f - o_j||)(q_f - o_j), \tag{147}$$

where

$$A(d) = \begin{cases} 0 & d > S \\ \frac{S-d}{S-R} & d \leq S \end{cases}.$$

With the obstacle avoidance behavior in hand, we can define the overall behavior of each robot in the network. The obstacle avoidance behavior is combined with the follower behavior developed in previous sections. This allows the overall behavior which each follower robot will execute to be written as:

$$\dot{q}_{f_\epsilon} = \dot{q}_{d_{f_\epsilon}} + k_p(q_{d_{f_\epsilon}} - q_{f_\epsilon}) + \sum_{j=1}^{N_{obs}} A(||q_f - o_j||)(q_f - o_j), \tag{148}$$

where the input velocity commands can be obtained from $\dot{q}_{f_\epsilon}$ using (144).

7.3 Virtual Leader Behavior-based MPC

With the behavior-based approach to leader tracking in the previous section, we are now ready to discuss the interactions between the agents in the network. We first discuss the different parameters that each agent will need to keep track of, which will lead to a statement of the algorithm to be employed. We then discuss two details important for determining the agents' interactions: the cost to be minimized and the communication topology.

7.3.1 Virtual Leader Algorithm

To implement the virtual leader approach, agents need to communicate and negotiate over various parameters. The parameters to negotiate upon are those that define the structure and movement of the formation. Namely, agent i must maintain a vector of parameters,

$\psi_i = [\Psi_i, \gamma_i, v_{il}, \omega_{il}]^T$, where the subscript i denotes that the variable is maintained by agent i, Ψ and γ are the rotation and scaling of the formation, and (v_l, ω_l) is the velocity pair that defines the motion of the virtual leader.

We note that, while the network of agents can find desirable values for v_l and ω_l in the absence of obstacles, in the presence of obstacles, a very undesirable local minimum exists. Namely, to avoid obstacles, the agents can simply stop moving. Thus, in the examples in Section 7.4.2, v_l is held constant and the agents solely steer the virtual leader using ω_l.

Beyond the distributed optimization of the mentioned parameters, agents must also come to agreement about the initial position and orientation of the leader, q_{l0} and ψ_{l0} respectively. While these could potentially be variables to be optimized, we found that simply performing consensus (i.e. the proportional component of PI distributed optimization) over the initial conditions provided better results . Performing consensus requires each agent to also maintain its own version of the leader's initial conditions, denoted as q_{l0_i} and ψ_{l0_i}. Beyond parameters to be negotiated, each agent must also communicate its initial state, which we denote as x_{0_i}.

One key note to make is that the optimization and execution is performed simultanously. In other words, agents optimize for $\delta_{execute}$ seconds on future values. If the time that optimization is started is denoted as t_0, agents are optimizing over the parameters that will begin to be executed at time $t_0 + \delta_{execute}$. This allows $\delta_{execute}$ to be chosen such that the optimization has a chance to converge before the parameters are employed. *We emphasize that for agents to move in unison, they must be in strict agreement on the virtual leader motion and the parameters influencing the structure of the formation.*

To account for the difference between variables being optimized and variables being executed, we again use the added notation $\bar{\psi}_i, \bar{v}_{il}, \bar{\omega}_{il}, \bar{q}_{il}, \bar{\psi}_{il}$ to denote the values actually being executed by the agent. The behavior-based virtual leader MPC algorithm can now be stated in Algorithm 4. While the algorithm is simply a specific implementation of Algorithm 3, we present it here for added clarity, giving the notation specific to the virtual leader approach.

Algorithm 4 Behavior-based Virtual Leader MPC

```
1. Initialize $t_0 = t + \delta_{execute}$ where $t$ is current time.
2. Agents communicate parameters with neighboring agents:
    (a) Formation structure and movement: $\psi_i, \gamma_i, \omega_{il}$ (and optionally $v_{il}$).
    (b) Virtual leader initial conditions: $q_{il}$, $\psi_{il}$.
    (c) Agent initial conditions: $x_{0_i} = x_i(t_0; t)$.
3. Update parameters:
    (a) Update $\psi_i, \gamma_i, \omega_{il}$ using distributed optimization.
    (b) Update $q_{il}$, $\psi_{il}$ using consensus.
4. Repeat steps 2 and 3 until $t = t_0$.
5. Set $\bar{\psi}_i = \psi_i, \bar{v}_{il} = v_{il}, \bar{\omega}_{il} = \omega_{il}, \bar{q}_{il} = q_{il}, \bar{\psi}_{il} = \psi_{il}$.
6. Repeat steps 1 through 5.
```

7.3.2 A Note on Information Topology

One of the essential factors to consider in behavior-based MPC for multi-agent systems is the information that each agent requires. In Section 6.3, required information in the network was discussed in terms of cost and dynamic dependencies. Issues arise as undesirable information dependencies can occur from having agent i's cost depend on an agent j and then agent j's dynamics depend on agent k, with whom agent i cannot communicate.

The only dynamic dependency that exists in the proposed virtual leader approach is a dependency between each agent and the virtual leader. If the virtual leader were a real agent, all agents would require a communication link between themselves and the leader. However, since the virtual leader is an artifact of distributed optimization, each agent need only communicate with the agents that form part of its cost.

Since the virtual leader has no dynamic dependencies, it then stops any “flow” of additional required information. This allows us to use the set of indices for which agent i has information, i.e. $\mathcal{N}_i^I$ from Chapter 6, as the typical definition of a neighborhood set, i.e. [57]. In other words, $j \in \mathcal{N}_i^I$ iff agent j and agent i are able to communicate.

7.3.3 Cost Definition

An essential aspect to maintaining the structural integrity and desirable movement of the formation comes through the definition of the cost. Similar to the DWA discussed in Section 1.3.3 and developed in [24], we include terms in the cost to avoid obstacles (in our case for all agents) while encouraging the orientation of the virtual leader to point towards the goal location at the end of the optimization horizon. We also include a term in the terminal cost to penalize deviation from the desired structure of the formation.

One further note is necessary before introducing the costs. Cost barriers, as presented in Chapter 5 for collision avoidance and reference tracking, are not suitable for the gradient-based distributed optimization. The reason being that, while presented in continuous time in Chapters 2 and 3, actual implementation must be discretized. To discretize a similar distributed optimization algorithm, [88] used a small gain theorem which requires the gradient to be bounded. Cost barriers, on the other hand, have an unbounded gradient, so they must be avoided.

To encourage keeping distance away from obstacles, the instantaneous portion of the cost consists of an exponential cost on proximity to obstacles. For faster convergence of the optimization, agents also include their neighboring agents in the cost. Thus, the instantaneous cost is defined as:

$$L_i = -\frac{\rho_1}{2} \sum_{j \in \mathcal{N}_i^I} \sum_{k=1}^{N_{obs}} (q_{ij}(t;t_0) - o_k)^T \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} (q_{ij}(t) - o_k) \tag{149}$$

where ρ_1 is a weight, a is a scalar which directly affects the radius of influence of the cost, and $q_{ij}(t;t_0)$ is agent i's version of agent j's position at time t computed at time t_0. Note that this cost allows the influence of the obstacle to be accounted for in the cost before the agent gets so close that the obstacle avoidance behavior kicks in. Thus it encourages obstacle avoidance while the obstacle avoid behavior ensures avoidance.

The terminal cost is designed to both maintain the structure of the formation as well as encourage progression towards the goal. The first term penalizes deviation from a desired scaling, the second penalizes deviation from agents desired position, and the third encourages the leader to point towards the goal location. The cost can be written as:

$$\begin{aligned}\Psi_i =& \frac{\rho_2}{2}(\gamma_i - 1)^2 + \frac{\rho_3}{2}\sum_{j\in\mathcal{N}_i^I} ||q_{ij}(t_0+\Delta;t_0) - q_{ij_d}(t_0+\Delta;t_0)||^2 \\ &+ \frac{\rho_4}{2}(\psi_{il}(t_0+\Delta;t_0) - \psi_{il_d}(t_0+\Delta;t_0))^2,\end{aligned} \tag{150}$$

where $q_{ij_d}(t_0+\Delta;t_0)$ is agent i's version of what agent j's desired position should be using agent i's version of the leader's variables and $\psi_{il_d}(t_0+\Delta;t_0)$ is the orientation that would point from $q_{il}(t_0+\Delta;t_0)$ to the goal position. Care must be taken to ensure that $(\psi_{il}(t_0+\Delta;t_0) - \psi_{il_d}(t_0+\Delta;t_0)) \in [-\pi,\ \pi)$.

Many more costs are conceivable, such as costs on velocities, and positions, etc. In particular, in the absence of obstacles, we found that the cost $\frac{1}{2}||q_{il} - q_{goal}||^2$ was particular useful in helping to regulate v_l so that the leader agent converged to the goal location, q_{goal}. In the presence of obstacles, we found a constant leader velocity and the mentioned orientation cost to work quite well in guiding the virtual leader.

7.4 Results

7.4.1 Measuring Formation Performance

To compare results between different trials, we evaluate the distance of each agent from its position in the formation. However, the measure of distance from an agent's position to its desired position in the formation is not well-defined. It is complicated by the fact that the agents use the rotation and scaling of the formation to adapt their motion.

Figure 40 shows an example of the current position of each agent plotted along side the positions resulting from the global variables. A naive comparison would be to sum the distances of each agent to its desired position. However, despite the fact that the agents

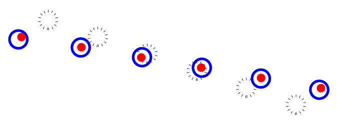


Figure 40: This figure shows the actual positions of the agents plotted as circles with solid lines, the desired positions using the global variables as circles with dotted lines, and the desired positions using the process in Section 7.4.1 as solid circles.

are not in their "desired positions" they are obviously very close to executing a line, as they are supposed to be doing. Therefore, to measure the deviation of the agents from the formation, we post-process the data to find values for the scaling, rotation, and center of the formation which better fit the actual positions of the agents.

The scaling, rotation, and center of the formation can be computed using post-processing by examining three agents. First, a value for the scaling can be found by examining the distance between two agents and comparing it with the nominal distance. In other words, let the nominal distance between agents i and j be denoted as $\bar{d}_{ij} = ||y_i - y_j||$ and the actual distance as $d_{ij} = ||x_i - x_j||$. An estimate for the scaling, as seen between i and j, can be found as

$$\hat{\gamma}_{ij} = \frac{d_{ij}}{\bar{d}_{ij}}.$$

The estimate for the scaling, $\hat{\gamma}$, as seen between three agents is just taken to be the average of the scaling between each pair of agents.

The estimate of the position of the origin of the formation, $\hat{q}_l$, can be found by computing the point closest to the three circles with centers at x_i, x_j, and x_k and radii $\hat{\gamma}||y_i||$, $\hat{\gamma}||y_j||$, and $\hat{\gamma}||y_j||$. Ideally there would be a unique intersection point between the three circles, however, error in $\hat{\gamma}$ and agent position makes this not the case.

From the center position of the formation and the scaling, the rotation of the formation can then be found. Allow ϕ_{nom_i} to denote the angle from the x-axis to y_i, and ϕ_i to be the angle off the x-axis to x_i with $\hat{q}_l$ being the origin. The estimate of the rotation of the formation according to agent i can be calculated as $\hat{\psi}_i = \phi_i - \phi_{nom_i}$. The estimate,

$\hat{\psi}$, according the three agents is then the average, with care being taken to ensure $\hat{\psi}_i \in [-\pi,\ \pi)$.

At each point in time, we evaluate all combinations of three agents. For each set of three agents, the estimated parameters are extracted and the distance from all agents to the desired position in the formation is computed using the estimates of the parameters. The parameters selected for that particular point in time are chosen to be the parameters which result in the lowest median distance from agents to their computed desired position. The median is used, as opposed to the average, in an effort to remove the affect from agents that are actually out of formation. As a measure of distance to formation, we then use the average and maximum distance of any agent to its position in the formation.

The results of using the outlined process to find the desired positions of the agents is shown in Figure 40. The desired positions match the actual positions of the agents much better than simply using the global variables as a measure. This allows a truer measure of whether or not the agents are in formation.

7.4.2 Virtual Leader Formation Results

To present an example of the virtual-leader behavior-based MPC framework, we present two sets of trials. The two sets consist of agents forming a line formation and a GT formation (similar to the Georgia Tech logo). In each trial, agents used distributed optimization to solve for ω_l, γ, and ψ. The only centralized aspect of the trials was the occasional broadcast of a new goal position for the leader, enabling the agents to traverse the entire environment through a waypoint navigation approach.

We emphasize that the most important aspect for the agents to maintain the structure of the formation is that within $\delta_{execute}$ the disagreement between agents must be small. In each trial, the gains for distributed optimization were set as $k_g = .1$, $k_p = 2.5$, and $k_I = 1.5$. Together with the costs in Section 7.3.3 and a value of $\delta_{execute} = 0.4$, the distributed optimization had sufficient amount of time to achieve sufficient agreement.

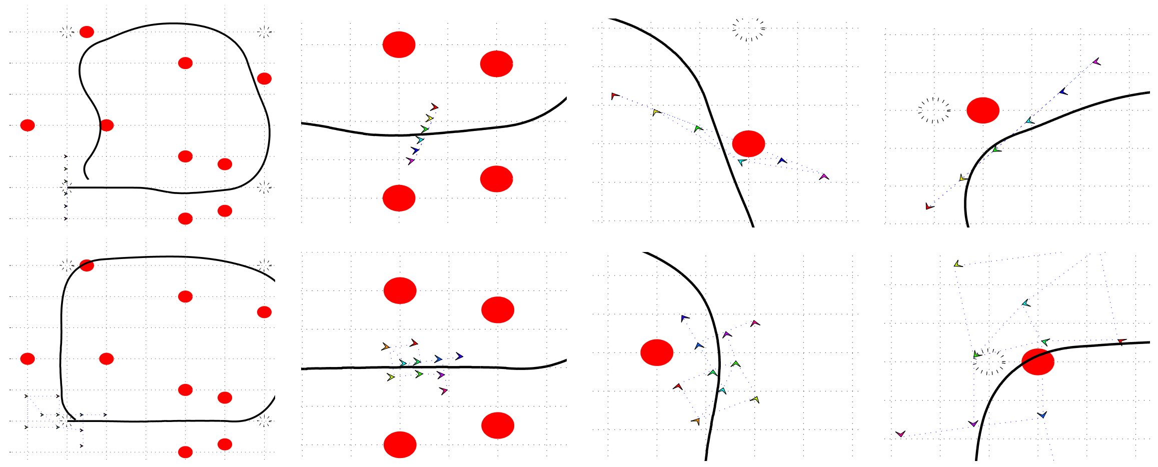


Figure 41: This figure shows the results from the equal gain trial (i.e. $\rho_1 = \rho_2 = \rho_3 = \rho_4 = 1$) for the line formation on the top row and GT formation on the bottom row. The left most image shows the initial environment and nominal positions of the agents. The right three images show zoomed in images of the agents at different points in the environment. The agents are shown as triangles, the obstacles are shown as solid circles, the waypoints are shown as dotted circles, and the trajectory of the leader agent around the environment is plotted as a solid line.

Each formation was run for two trials. The first trial consisted of the agents controlling only the leader velocity, without regard to the structure of the formation. This was achieved by setting $\rho_1 = \rho_2 = \rho_3 = 0$ and $\rho_4 = 1$. The second trial consisted of setting $\rho_1 = \rho_2 = \rho_3 = 1$. Basically a trial without tuning the costs.

We present the line as it shows the ability for agents to maintain a formation without an underlying rigid structure. It is also highly rotationally variant, meaning as the line is rotated it can move very differently between obstacles. In Figure 41, it is seen that the line is able to use both the rotation and scaling parameters to navigate around obstacles while maintaining the structure of the formation. Figure 42 shows that without tuning the costs, a large reduction in both the maximum distance and average distance is attained.

The GT formation presents an example showing the ability for agents to maintain arbitrary structures. It is different from the line formation in that the structure of the formation renders rotation much less useful. Figure 41 shows a resulting path of the formation as well as multiple instances. The GT structure mainly used the scaling to avoid obstacles and ω_l

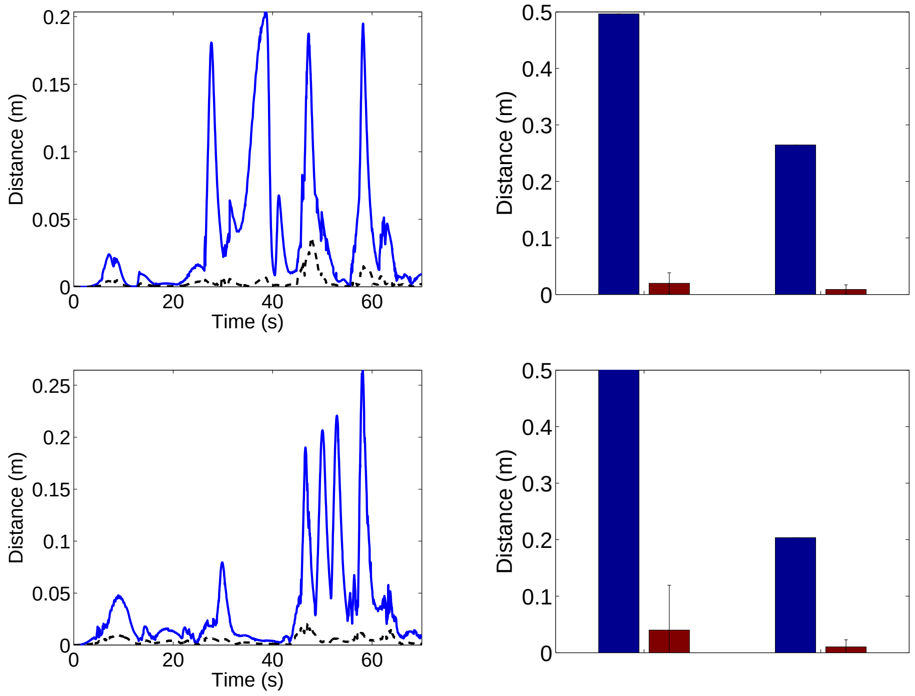


Figure 42: This figure shows the formation error for the line formation on the top row and the GT formation on the bottom row. On the left is shown the maximum and average distance for each agent for $\rho_1 = \rho_2 = \rho_3 = \rho_4 = 1$. On the right is shown a bar graph with the maximum and average distance for both trials. The left set of bars corresponds to the results from leader-only control (i.e. $\rho_1 = \rho_2 = \rho_3 = 0$, $\rho_4 = 1$). The right set of bars corresponds to the results from using $\rho_1 = \rho_2 = \rho_3 = \rho_4 = 1$. The value for the maximum distance in the leader-only control trial was 1.23 for the line formation and 0.497 for the GT formation.

to steer around them. Figure 42 again shows a large reduction in the error without tuning the costs.

Both examples show the ability of the controller discussed in Section 7.2 to keep agents in formation. The trials used a value of $\epsilon = 0.1$, but the average distance from agents to the desired position is far less than ϵ.

7.5 Conclusion

In this Chapter, a distributed virtual-leader formation control was developed as a culmination of many of the concepts presented in previous chapters. In particular it provided a detailed example of the multi-agent behavior-based MPC approach presented in Chapter 6. The basic elements of the approach, an underlying behavior, costs, and an evaluation of the communication dependencies were examined. Through the introduction of the optimization framework presented with behavior-based MPC, the agents were able to maintain formation, moving throughout an environment while keeping the structure of the formation in tact.

REFERENCES

[1] ANGELI, D., CASAVOLA, A., and MOSCA, E., "Constrained predictive control of nonlinear plants via polytopic linear system embedding," International Journal of Robust and Nonlinear Control, vol. 10, no. 13, pp. 1091–1103, 2000.

[2] ARKIN, R., Behavior-based robotics. The MIT Press, 1998.

[3] ARROW, K., HURWICZ, L., and UZAWA, H., Studies in Nonlinear Programming. Stanford University Press, Stanford, CA, 1958.

[4] BEARD, R. and MCLAIN, T., Small unmanned aircraft: Theory and practice. Princeton University Press, 2012.

[5] BOYD, S., Convex Optimization. Cambridge University Press, 2004.

[6] BOYD, S., PARIKH, N., CHU, E., PELEATO, B., and ECKSTEIN, J., "Distributed optimization and statistical learning via the alternating direction method of multipliers," Foundations and Trends in Machine Learning, vol. 3, no. 1, pp. 1–122, 2011.

[7] BROCK, O. and KHATIB, O., "High-speed navigation using the global dynamic window approach," in International Conference on Robotics and Automation, Proceedings, vol. 1, pp. 341–346, IEEE, 1999.

[8] BROGAN, W. L., Modern Control Theory. New York: Quantum Publishers, 1974.

[9] BRYSON, A. and HO, Y., Applied optimal control: optimization, estimation, and control. Hemisphere Publications, 1975.

[10] CAMPONOGARA, E., JIA, D., KROGH, B. H., and TALUKDAR, S., "Distributed model predictive control," Control Systems, vol. 22, no. 1, pp. 44–52, 2002.

[11] CANNON, M., "Efficient nonlinear model predictive control algorithms," Annual Reviews in Control, vol. 28, no. 2, pp. 229–237, 2004.

[12] CHEN, H. and ALLGÖWER, F., "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability," Automatica, vol. 34, no. 10, pp. 1205–1217, 1998.

[13] CHISCI, L., FALUGI, P., and ZAPPA, G., "Gain-scheduling mpc of nonlinear systems," International Journal of Robust and Nonlinear Control, vol. 13, no. 3-4, pp. 295–308, 2003.

[14] CHOSET, H., LYNCH, K., HUTCHINSON, S., KANTOR, G., BURGARD, W., KAVRAKI, L., and THRUN, S., Principles of robot motion: theory, algorithms, and implementations. MIT press, 2005.

[15] CONTE, C., VOELLMY, N. R., ZEILINGER, M. N., MORARI, M., and JONES, C. N., "Distributed synthesis and control of constrained linear systems," in American Control Conference (ACC), pp. 6017–6022, IEEE, 2012.

[16] DIEHL, M., FERREAU, H., and HAVERBEKE, N., "Efficient numerical methods for nonlinear mpc and moving horizon estimation," Nonlinear Model Predictive Control, pp. 391–417, 2009.

[17] DROGE, G. and EGERSTEDT, M., "Adaptive time horizon optimization in model predictive control," in American Control Conference (ACC), pp. 1843–1848, IEEE, 2011.

[18] DROGE, G. and EGERSTEDT, M., "Distributed parameterized model predictive control of networked multi-agent systems," American Control Conference (ACC), 2013.

[19] DROGE, G., KINGSTON, P., and EGERSTEDT, M., "Behavior-based switch-time mpc for mobile robots," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2012.

[20] DUNBAR, W. and MURRAY, R., "Distributed receding horizon control for multi-vehicle formation stabilization," Automatica, vol. 42, no. 4, pp. 549–558, 2006.

[21] DUNBAR, W. B., "Distributed receding horizon control of dynamically coupled nonlinear systems," Transactions on Automatic Control, vol. 52, no. 7, pp. 1249–1263, 2007.

[22] EGERSTEDT, M., WARDI, Y., and DELMOTTE, F., "Optimal control of switching times in switched dynamical systems," in Proceedings. 42nd IEEE Conference on Decision and Control, vol. 3, pp. 2138 – 2143 Vol.3, dec. 2003.

[23] FEIJER, D. and PAGANINI, F., "Stability of primal-dual gradient dynamics and applications to network optimization," Automatica, vol. 46, no. 12, pp. 1974–1981, 2010.

[24] FOX, D., BURGARD, W., and THRUN, S., "The dynamic window approach to collision avoidance," Robotics & Automation Magazine, vol. 4, no. 1, pp. 23–33, 1997.

[25] G F, F., POWELL, J., and EMAMI-NAEINI, A., "Feedback control of dynamic systems," 2001.

[26] GHARESIFARD, B. and CORTÉS, J., "Continuous-time distributed convex optimization on weight-balanced digraphs," in IEEE 51st Annual Conference on Decision and Control (CDC), pp. 7451–7456, IEEE, 2012.

[27] GHARESIFARD, B. and CORTÉS, J., "Distributed continuous-time convex optimization on weight-balanced digraphs," IEEE Transactions on Automatic Control, To appear. Preprint available at http://arxiv.org/abs/1204.0304.

[28] GISELSSON, P. and RANTZER, A., "Distributed model predictive control with suboptimality and stability guarantees," in 49th IEEE Conference on Decision and Control, pp. 7272–7277, 2010.

[29] GOERZEN, C., KONG, Z., and METTLER, B., "A survey of motion planning algorithms from the perspective of autonomous uav guidance," Journal of Intelligent and Robotic Systems, vol. 57, no. 1-4, pp. 65–100, 2010.

[30] GOUVEA, J. A., LIZARRALDE, F., and HSU, L., "Formation control of dynamic nonholonomic mobile robots with curvature constraints via potential functions," in American Control Conference (ACC), pp. 3039–3044, IEEE, 2013.

[31] HESPANHA, J. P., LIBERZON, D., ANGELI, D., and SONTAG, E. D., "Nonlinear norm-observability notions and stability of switched systems," IEEE Transactions on Automatic Control, vol. 50, no. 2, pp. 154–168, 2005.

[32] HOWARD, T., GREEN, C., and KELLY, A., "Receding horizon model-predictive control for mobile robot navigation of intricate paths," in Field and Service Robotics, pp. 69–78, Springer, 2010.

[33] HURT, J., "Some stability theorems for ordinary difference equations," SIAM Journal on Numerical Analysis, vol. 4, no. 4, pp. 582–596, 1967.

[34] JADBABAIE, A., LIN, J., and MORSE, A., "Coordination of groups of mobile autonomous agents using nearest neighbor rules," IEEE Transactions on Automatic Control, vol. 48, no. 6, pp. 988–1001, 2003.

[35] JADBABAIE, A., YU, J., and HAUSER, J., "Unconstrained receding-horizon control of nonlinear systems," IEEE Transactions on Automatic Control, vol. 46, no. 5, pp. 776–783, 2001.

[36] JAN, A. and IJSPEERT, "Central pattern generators for locomotion control in animals and robots: A review," Neural Networks, vol. 21, no. 4, pp. 642 – 653, 2008.

[37] JOHNSON, S. G., "The nlopt nonlinear-optimization package."

[38] JONES, C., PU, Y., RIVERSO, S., FERRARI-TRECATE, G., ZEILINGER, M. N., and OTHERS, "Plug and play distributed model predictive control based on distributed invariance and optimization," in The 52nd Conference on Decision and Control, 2013.

[39] KEVICZKY, T., BORRELLI, F., and BALAS, G., "Decentralized receding horizon control for large scale dynamically decoupled systems," Automatica, vol. 42, no. 12, pp. 2105–2115, 2006.

[40] KHALIL, H., Nonlinear systems. 3rd ed. Prentice hall, 2002.

[41] KIM, D. and KIM, J., "A real-time limit-cycle navigation method for fast mobile robots and its application to robot soccer," Robotics and Autonomous Systems, vol. 42, no. 1, pp. 17–30, 2003.

[42] KIM, Y., KIM, S., and KWAK, Y., "Dynamic analysis of a nonholonomic two-wheeled inverted pendulum robot," Journal of Intelligent & Robotic Systems, vol. 44, no. 1, pp. 25–46, 2005.

[43] KIRK, D., Optimal control theory: an introduction. Dover Publications, 2004.

[44] KRAJNÍK, T., VONÁSEK, V., FIŠER, D., and FAIGL, J., "Ar-drone as a platform for robotic research and education," in Research and Education in Robotics-EUROBOT 2011, pp. 172–186, Springer, 2011.

[45] KVATERNIK, K. and PAVEL, L., "A continuous-time decentralized optimization scheme with positivity constraints," in IEEE Conference on Decision and Control, 2012.

[46] KVATERNIK, K. and PAVEL, L., "Lyapunov analysis of a distributed optimization scheme," in 5th International Conference on Network Games, Control and Optimization (NetGCooP), IEEE, 2011.

[47] LATOMBE, J., Robot motion planning. Springer, 1990.

[48] LAVELL, S. M., Planning Algorithms. Cambridge: Cambridge University Press, 2006.

[49] LIBERZON, D., Switching in systems and control. Springer, 2003.

[50] LIU, T. and JIANG, Z.-P., "A nonlinear small-gain approach to distributed formation control of nonholonomic mobile robots," in American Control Conference (ACC), pp. 3051–3056, IEEE, 2013.

[51] LOBEL, I. and OZDAGLAR, A., "Distributed subgradient methods over random networks," in Proceedings of Allerton Conference on Communication, Control, Computation, 2008.

[52] LOBEL, I. and OZDAGLAR, A., "Distributed subgradient methods for convex optimization over random networks," Automatic Control, IEEE Transactions on, vol. 56, no. 6, pp. 1291–1306, 2011.

[53] LUENBERGER, D. and YE, Y., Linear and nonlinear programming (3rd ed.), vol. 116. Springer, 2008.

[54] MANIATOPOULOS, S., PANAGOU, D., and KYRIAKOPOULOS, K. J., "Model predictive control for the navigation of a nonholonomic vehicle with field-of-view constraints," in American Control Conference (ACC), pp. 3967–3972, IEEE, 2013.

[55] MARTIN, P. and EGERSTEDT, M., "Optimization of multi-agent motion programs with applications to robotic marionettes," Hybrid Systems: Computation and Control, pp. 262–275, 2009.

[56] MAYNE, D., RAWLINGS, J., RAO, C., and SCOKAERT, P., "Constrained model predictive control: Stability and optimality," Automatica, vol. 36, no. 6, pp. 789–814, 2000.

[57] MESBAHI, M. and EGERSTEDT, M., Graph theoretic methods in multiagent networks. Princeton Univerisity Press, 2010.

[58] MICHALSKA, H. and MAYNE, D., "Robust receding horizon control of constrained nonlinear systems," IEEE Transactions on Automatic Control, vol. 38, no. 11, pp. 1623–1633, 1993.

[59] MURRAY, R. M., "Recent research in cooperative control of multivehicle systems," Transactions-American Society of Mechanical Engineerers Journal of Dynamic Systems Measurement and Control, vol. 129, no. 5, p. 571, 2007.

[60] NAWAWI, S., OSMAN, J., and JOHARI, H., "Control of two-wheels inverted pendulum mobile robot using full order sliding mode control," in Proc. of the Conf. on Man-Machine Systems, 2006.

[61] NEDIC, A. and OZDAGLAR, A., "Distributed subgradient methods for multi-agent optimization," IEEE Transactions on Automatic Control, vol. 54, no. 1, pp. 48–61, 2009.

[62] NEDIC, A. and OZDAGLAR, A., "Subgradient methods for saddle-point problems," Journal of optimization theory and applications, vol. 142, no. 1, pp. 205–228, 2009.

[63] NELSON, D., BARBER, D., MCLAIN, T., and BEARD, R., "Vector field path following for miniature air vehicles," IEEE Transactions on Robotics, vol. 23, no. 3, pp. 519–529, 2007.

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

[65] OGREN, P., FIORELLI, E., and LEONARD, N. E., "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," Automatic Control, IEEE Transactions on, vol. 49, no. 8, pp. 1292–1302, 2004.

[66] OLFATI-SABER, R. and MURRAY, R., "Consensus problems in networks of agents with switching topology and time-delays," IEEE Transactions on Automatic Control, vol. 49, no. 9, pp. 1520–1533, 2004.

[67] OLFATISABER, R., "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, pp. 4690–4695, IEEE, 2002.

[68] PALOMAR, D. and ELDAR, Y., Convex optimization in signal processing and communications. Cambridge University Press, 2010.

[69] PARK, J. J., JOHNSON, C., and KUIPERS, B., "Robot navigation with model predictive equilibrium point control," in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4945–4952, IEEE, 2012.

[70] POLAK, E., Optimization: algorithms and consistent approximations, vol. 7. Springer New York, 1997.

[71] PRIMBS, J. A., NEVISTIĆ, V., and DOYLE, J. C., "Nonlinear optimal control: A control lyapunov function and receding horizon perspective," Asian Journal of Control, vol. 1, no. 1, pp. 14–24, 1999.

[72] QUIGLEY, M., CONLEY, K., GERKEY, B., FAUST, J., FOOTE, T., LEIBS, J., WHEELER, R., and NG, A. Y., "Ros: an open-source robot operating system," in ICRA workshop on open source software, vol. 3, 2009.

[73] RAM, S. S., VEERAVALLI, V. V., and NEDIC, A., "Distributed and recursive parameter estimation in parametrized linear state-space models," Automatic Control, IEEE Transactions on, vol. 55, no. 2, pp. 488–492, 2010.

[74] RANTZER, A., "On prize mechanisms in linear quadratic team theory," in 46th IEEE Conference on Decision and Control, pp. 1112–1116, 2007.

[75] RANTZER, A., "Dynamic dual decomposition for distributed control," in American Control Conference, pp. 884–888, 2009.

[76] RIMON, E. and KODITSCHEK, D., "Exact robot navigation using artificial potential functions," IEEE Transactions on Robotics and Automation, vol. 8, no. 5, pp. 501–518, 1992.

[77] SAUNDERS, J. and BEARD, R., "Reactive vision based obstacle avoidance with camera field of view constraints," in Guidance, Navigation, and Control Conference, 2008.

[78] SAYED, A. H., TU, S.-Y., CHEN, J., ZHAO, X., and TOWFIC, Z. J., "Diffusion strategies for adaptation and learning over networks: An examination of distributed strategies and network behavior," Signal Processing Magazine, IEEE, vol. 30, no. 3, pp. 155–171, 2013.

[79] SCATTOLINI, R., "Architectures for distributed and hierarchical model predictive control–a review," Journal of Process Control, vol. 19, no. 5, pp. 723–731, 2009.

[80] SHAH, P. and PARRILO, P., "An optimal controller architecture for poset-causal systems," Arxiv preprint arXiv:1111.7221, 2011.

[81] SHAMMA, J., Cooperative control of distributed multi-agent systems. Wiley Online Library, 2007.

[82] STACHNISS, C. and BURGARD, W., "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," in International Conference on Intelligent Robots and Systems, vol. 1, pp. 508–513, IEEE, 2002.

[83] STEWART, B. T., VENKAT, A. N., RAWLINGS, J. B., WRIGHT, S. J., and PANNOCCHIA, G., "Cooperative distributed model predictive control," Systems and Control Letters, vol. 59, no. 8, pp. 460–469, 2010.

[84] SUNDHAR RAM, S., NEDIĆ, A., and VEERAVALLI, V., "Distributed stochastic subgradient projection algorithms for convex optimization," Journal of optimization theory and applications, vol. 147, no. 3, pp. 516–545, 2010.

[85] TERELIUS, H., TOPCU, U., and MURRAY, R., "Decentralized multi-agent optimization via dual decomposition," in KTH, Automatic Control, IFAC, 2011.

[86] TESCH, M., LIPKIN, K., BROWN, I., HATTON, R., PECK, A., REMBISZ, J., and CHOSET, H., "Parameterized and scripted gaits for modular snake robots," Advanced Robotics, vol. 23, no. 9, pp. 1131–1158, 2009.

[87] TU, K.-H. and SHAMMA, J. S., "Nonlinear gain-scheduled control design using set-valued methods," in American Control Conference, 1998. Proceedings of the 1998, vol. 2, pp. 1195–1199, IEEE, 1998.

[88] WANG, J. and ELIA, N., "Control approach to distributed optimization," in 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton), pp. 557–561, 2010.

[89] WANG, J. and ELIA, N., "A control perspective for centralized and distributed convex optimization," in 50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC), 2011.

[90] WEI, E. and OZDAGLAR, A., "Distributed alternating direction method of multipliers," in IEEE Conference on Decision and Control, 2012.

[91] YOUNG, B. J., BEARD, R. W., and KELSEY, J. M., "A control scheme for improving multi-vehicle formation maneuvers," in American Control Conference, 2001. Proceedings of the 2001, vol. 2, pp. 704–709, IEEE, 2001.

[92] ZEGEYE, S., DE SCHUTTER, B., HELLENDOORN, J., and BREUNESSE, E., "Parameterized mpc to reduce dispersion of road traffic emissions," in American Control Conference (ACC), pp. 4428–4433, IEEE, 2011.