A MODEL PREDICTIVE CONTROL APPROACH TO A CLASS OF MULTIPLAYER MINMAX DIFFERENTIAL GAMES

BY

SEUNGHO LEE

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

ABSTRACT

In this dissertation, we consider a class of two-team adversarial differential games in which there are multiple mobile dynamic agents on each team. We describe such games in terms of semi-infinite minmax Model Predictive Control (MPC) problems, and present a numerical optimization technique for efficiently solving them. We also describe the implementation of the solution method in both indoor and outdoor robotic testbeds.

Our solution method requires one to solve a sequence of Quadratic Programs (QPs), which together efficiently solve the original semi-infinite minmax MPC problem. The solution method separates the problem into two subproblems called the inner and outer subproblems, respectively. The inner subproblem is based on a constrained nonlinear numerical optimization technique called the Phase I-Phase II method, and we develop a customized version of this method. The outer subproblem is about judiciously initializing the inner subproblems to achieve overall convergence; our method guarantees exponential convergence.

We focus on a specific semi-infinite minmax MPC problem called the harbor defense problem. First, we present foundational work on this problem in a formulation containing a single defender and single intruder. We next extend the basic formulation to various advanced scenarios that include cases in which there are multiple defenders and intruders, and also ones that include varying assumptions about intruder strategies.

Another main contribution is that we implemented our solution method for the harbor defense problem on both real-time indoor and outdoor testbeds, and demonstrated its computational effectiveness. The indoor testbed is a custom-built robotic testbed named HoTDeC (Hovercraft Testbed for Decentralized Control). The outdoor testbed involved full-sized US Naval Academy patrol ships, and the experiment was conducted in Chesapeake Bay in collaboration with the US Naval Academy. The scenario used involved one ship(the intruder) being commanded by a human pilot, and the defender ship being controlled automatically by our semi-infinite minmax MPC algorithm.The results of several experiments are presented.

Finally, we present an efficient algorithm for solving a class of matrix games, and show how this approach can be directly used to effectively solve our original continuous space semi-infinite minmax problem using an adaptive approximation.

To my family, for their love, support, and patience.

ACKNOWLEDGMENTS

I would like to thank Professors Tamer Başar, Professr Srinivasa Salapaka, and Professor Prashant Mehta, for serving on my committee, and for their useful feedback, advice and comments. I also would like to thank Professor Elijah Polak at University of California, Berkeley, for helpful discussions and comments on numerical optimization. I would like to thank my adviser, Professor Geir E. Dullerud, for the constant support and guidance he has gladly been giving me. He has been a great motivator and inspirer for my studies. I am indebted to him for his advice and for the academic freedom I received to pursue research, both theoretical and application-oriented, in many subjects of my interest. I could not have hoped for a better adviser. Last but foremost, I am grateful to my family for their support, patience, sacrifice, and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

N Prediction horizon of MPC.

t Discrete time sample number for MPC.

i Number of iterations in the Method of outer approximation.

m Number of iterations for solving the Phase I-Phase II problem.

k Index of the cost functions.

j Index of the constraint functions.

$x_+ = \max(x, 0)$.

$S_a(f) = \{x \mid f(x) \leq a\}$ Sub level set of function f.

$df(x; h)$ Directional derivative of function f in h direction.

$df_2(z, x; h)$ One-sided directional derivative of function f with respect to second argument in h direction.

$\Sigma_q = \{\mu \mid \mu \in \mathbf{R}^q_+, \Sigma^q_{i=0}\mu^i = 1\}$ Simplex set.

$\Sigma_{q+1} = \{(\mu^{0,}\mu) \mid \mu^0 \in \mathbf{R}, \mu \in \mathbf{R}^q_+, \Sigma^q_{i=1}\mu^i = 1\}$.

$\partial f(x)$ Subgradient of function f at x.

$B(x, \rho)$ Ball centered at x with a radius ρ.

$A(k)$ k^{th} An element of set A.

$\hat{q}(x) = \{j \in q \mid f^j(x) = \psi(x)\}$ Active index set.

$\nabla^2 f(x)$ Hessian matrix.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Model Predictive Control (MPC) is a form of feedback control in which the current control action is obtained by solving online, at each sample time, an open-loop optimal control problem over a fixed time window with the current system state as the initial condition. The solution of this optimization problem yields a finite sequence of control actions over the optimization window; only the first value is applied to the system, and then the window is advanced one sample time and the optimization process repeated [1]. The Fig.1.1 conceptually illustrates the framework of MPC.

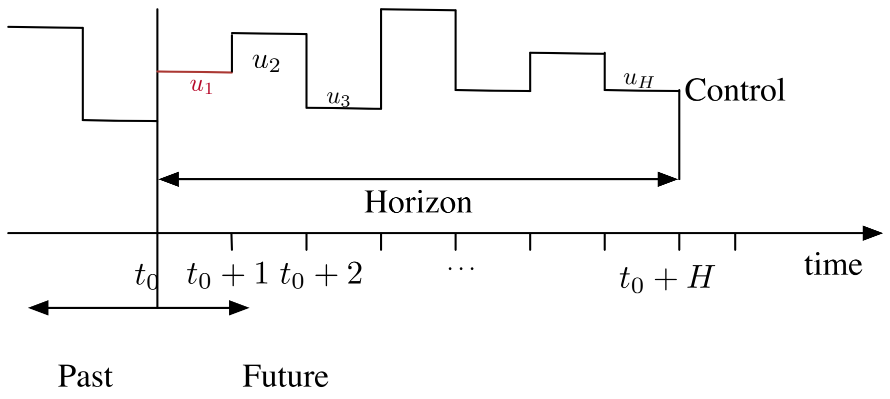


Figure 1.1: Model Predictive Control. The current time is t_0. Horizon length is H. The first element of the sequence of control u_1 is applied.

One of the most attractive aspects of MPC is that state or control bounds can be incorporated as either hard or soft constraints in the optimal control formulation [2]. This allows MPC to be broadly applicable to many domains, such as chemical process and industrial control [3], [4], supply chain management [5], [6], economics and finance [7], [8], dynamic hedging [9] and revenue management [10], [11].

However, most of the MPC approaches are limited to solving the *Bolza* problem [12] at each discrete time:

$$\min_{u} \sum_{t=t_0}^{t_H} L(t, x(t), u(t)) + K(t_H, x_H) \tag{1.1}$$

Here t_H and $x(t_H)$ are the final time and state, L is the running cost, and K is the terminal cost. Very few MPC approaches allow the use of a minmax performance metric, which is the topic of this thesis. There are some exceptions. In [13], the concept of minimax MPC extended to uncertain linear systems. The analysis of the robustness is performed in a worst case sense in [14]. Both fixed and variable horizon minimax MPC are discussed in [15]. However, the system is limited to the linear, and the cost is quadratic [13–15]. In some special cases, the cost function is relaxed to convex function [16], or the problem is a finite minmax problem [17–19]. None of those aforementioned work consider the computing time in terms of real-time applicable.

Among the class of minmax problems, we are particularly focused on the semi-infinite minmax problem as such problems arise naturally in engineering design where it is necessary to maintain the response of dynamical systems within prescribed performance [20]. The semi-infinite minmax problem is the problem with the following form.

$$\min_{x \in R^n} \max_{j \in \mathbf{q}} \Psi^j(x), \tag{1.2}$$

where the functions $\Psi^j : R^n \to R$, $j = 0, 1, \cdots, q$ are of the form

$$\Psi^j(x) = \max_{y_j \in Y_j} \phi^j(x, y_j), \tag{1.3}$$

with the function $\phi^j : R^n \times R^{m_j} \to R$ and the sets $Y_j \subset R^{m_j}$, $\mathbf{q} := \{1, 2, \cdots, q\}$. Such problems are called semi-infinite because the design vector x is finite dimensional, and there are infinitely many functions $\phi^j(\cdot, y)$, determined by all the $y \in Y_j$ in their specification.

One of the well-known drawbacks of MPC is the computational cost [1]. This has limited MPC applications to systems with slow dynamics such as chemical process control [21], [22]. When it comes to the semi-infinite or finite minmax problems, this limitation becomes even more significant. Hence, current minmax MPC applications are limited to simple problems [13–15].

Another drawback of the MPC is its recursive feasibility [23], [24], which will be explained in detail in the next chapter, is also well addressed in [25]. Let the set of admissible states $\mathbb{X}$ and admissible control sequences of length N be $U^N(x)$. Suppose the feasible set is $\mathbb{X}_N := \{x \in \mathbb{X} \mid U^N(x) \neq \emptyset\}$. Then the system is recursively feasible if and only if for all $x \in A \subset \mathbb{X}_N$ and $u^* \in U^N(x)$, $x^+ = f(x, u^*) \in A$ is satisfied. In other words, once the system is feasible and it is updated via the MPC control law, the solution in the next time window is also feasible. Almost all work on MPC assumes a priori that this property holds. The typical approach to remedy this issue is to append the optimization problem with additional, somewhat artificial constraints, which will guarantee that loss of feasibility cannot occur [1].

Some level of remedy for all the aforementioned drawbacks of MPC was expected by adopting Phase I-Phase II method [26], [27], and the method of outer approximation [28], [29].

The Phase I-Phase II method was introduced in 60's and 70's [30–33]. Phase I refers to the situation when the current state is infeasible region and the Phase II refers that is in feasible region. The main concern is about how to bring the state in infeasible region to the feasible region efficiently. This seems to be related to the recursive feasibility issue in MPC. The method of outer approximation, which will be discussed in the next chapter, is a process of successive minimizations of the finite max functions. It produces approximate solutions for semi-infinite problems. This approximation scheme is applicable to the semi-infinite MPC problem to reduce the computation without sacrificing the quality of the solution.

1.2 Contribution

We developed method for solving a class of semi-infinite mimnax problems efficiently enough to be implemented in real-time while satisfying recursive feasibility. Our method is based on the Phase I- Phase II method and the method of the outer approximation.

In this study, we consider the adversarial game between two teams (X and Y teams). The dynamic model of two teams is generally nonlinear with control u and v, respectively.

$$\dot{x} = \bar{f}_x(x, u), \quad \dot{y} = \bar{f}_y(y, v) \tag{1.4}$$

Many games of this type can be well described as a minmax MPC problem. To incorporate the notion of MPC, we introduce the finite sequence of controls. At discrete time sample number t, let the sequence of control for the X and Y teams be $\mathbf{u_t}$ and $\mathbf{v_t}$, respectively.

$$\begin{aligned} \mathbf{u_t} &= \{u_{t+1}, u_{t+2}, \cdots, u_{t+H}\} \\ \mathbf{v_t} &= \{v_{t+1}, v_{t+2}, \cdots, v_{t+H}\}. \end{aligned} \tag{1.5}$$

Throughout this study, the basic form of the problem we are interested in is the following min max problem. For a given x_0 and y_0, solve

$$\begin{aligned} &\min_{\mathbf{u_t} \in \mathbf{U}} \max_{\substack{\mathbf{v_t} \in \mathbf{V}(\mathbf{u_t}) \\ k \in \{t+1, t+2, \cdots, t+H\}}} c(x_k, y_k) \\ &s.t. \; f^j(x_k, u_k) \le 0, \quad j \in \mathbf{q} \\ &\quad\; x_{k+1} = f_x(x_k, u_k), \\ &\quad\; y_{k+1} = f_y(y_k, v_k), \end{aligned} \tag{1.6}$$

where $c : R \times R \to R$ is a cost function, $\mathbf{q} = \{1, 2, 3, \cdots, q\}$ is a finite index set for the X team inequality constraints, H is a horizon length. The set $\mathbf{V}(\mathbf{u_t})$ contains only feasible control sequence of Y team.

$$\mathbf{V}(\mathbf{u_t}) := \{\mathbf{v_t} \mid g^j(x_k, y_k, u_k, v_k) \le 0\}, \tag{1.7}$$

where the element wise inequality $g^j(\cdot, \cdot, \cdot, \cdot)$ are indexed by $j \in \{1, 2, \cdots J\}$, with $k \in \{1, 2, \cdots H\}$. In Y team's point of view, a finite number of inequality constraints could be a function of X team's control and state because we assume that information of the X team is correctly known to the Y team, and hence Y team's control is worst case to X team.

The solution of (1.6) is a best-worst case sequence of controls in x player's point of view. The only initial fraction of the solution is taken as a control for the current time of the player x. Such a sequence of control is fed to the player dynamics and results in the trajectory. Therefore, this problem is considered as a class of path planning. There seems to be no known method solving (1.6) directly. The solution for this class of problem can be obtained by bi-level set programming or optimality function approach. However, as this emerges as a subproblem in MPC formulation, computational complexity becomes a more significant issue.

We set a specific problem called the harbor defense problem as the main target problem to demonstrate the effectiveness of our solution method. The harbor defense problem that we describe as a semi-infinite minmax MPC problem is an adversarial game between two teams: intruder and defender teams. The goal of the intruder team is to destroy a the high-value unit that is located in the harbor by outmaneuvering the defender team. The goal of the defender team is to prevent all the intruders from doing so. An intruder is destroyed by the defender if the intruder comes within a pre-defined distance of the defender. As an additional demonstration, we formulate and present a mobile network jamming problem. This problem demonstrates that our solution method can be easily applied to other semi-infinite minmax problems.

Besides theoretical development, a contribution is also the implementation and demonstration of the solution method to real-time robotic testbeds. The main robotic testbed is the HoTDeC (Hovercraft Testbed for Decentralized Control) which is developed from the scratch at the Networked autonomous vehicle laboratory. Development of the HoTDeC includes a well-structured network environment because an effective network is crucial to implementing our minmax MPC scheme. Another testbed involves Yard Patrol ships at the U.S. Naval Academy. The experiments with HoTDeC are conducted at the laboratory scale with a multiple number of the agents per a team, whereas the experiments with Yard Patrol ships are conducted at the Chesapeake Bay near the U.S. Naval Academy.

1.3 Outline

Our approach to solving the problem (1.6) is to use numerical optimization techniques and separate the semi-infinite problem into a sequence of two smaller problems: finite minmax problem and maximization problem. We refer to the finite minmax problem as the inner problem and the maximization problem as the outer problem. Then the question is how to solve the subproblems efficiently and whether the sequence of the solutions of the subproblems converge to the original semi-infinite minmax problem. In Chapter

3, the solution method to the subproblems and the convergence of the sequence of the problem is analyzed. Illustrative examples are presented.

In Chapters 4 and 5, we consider the harbor defense problem and the implementation of our method. In Chapter 4, the basic harbor defense problem, where there are a single intruder and a single defender, is discussed. Advanced scenarios of the harbor defense problems are discussed in Chapter 5. Advanced scenarios include the case when there are multiple defenders and intruders are present, and the case when one team outnumbers the other team, and the case when the assumed strategy for the other team is incorrect.

The development of HoTDeC and the implementation of the solution of the harbor defense problem is presented in Chapter 6. The development of HoTDeC includes various sub-topics such as circuit design, network programming, and controller development. Chapter 7 elaborates on the implementation step and modification of the algorithm so that it can be implemented in real-time systems. In Chapter 8, results are presented for the sea experiment that was conducted at the Chesapeake Bay, near the US Naval Academy. The aim of this experiment is to demonstrate the effectiveness of our solution method in a real world situation.

Chapter 9 presents the application of our solution method to the mobile network jamming problem, and this shows that our solution method is applicable to other semi-infinite minmax MPC problems.

It is well known that finding a global solution to our optimization problem efficiently is a hard problem. In Chapter 10, we present an application of outer approximation to the discretized semi-infinite minmax problem for finding a global solution efficiently. We demonstrate the effectiveness of the algorithm by presenting three examples.

CHAPTER 2

BACKGROUND

2.1 Model Predictive Control

There is huge volume of MPC literature. This MPC literature is categorized into several main topics. Most common and well-understood is the MPC of deterministic systems. The basic form is as follows.

Suppose the current discrete time is k and let the discrete time system with state x and control u be described by

$$x_{k+1} = f(x_k, u_k). \tag{2.1}$$

Here, the dynamics $f : X \times U \to X$ assigns the state $x_k \in X$ at the next time instant to each pair of state $x \in X$ and control value $u \in U$.

$$\min_{u \in U} \sum_{i=0}^{N-1} l(x_{k+i}, u_{k+i}) + V(x_{k+N}, u_{k+N}), \tag{2.2}$$

where the u is a sequence of control, $u = \{u_{k+1} \cdots u_{k+N}\}$.

If the cost function is quadratic, it is normally expressed as follows

$$l(x_{k+i}, u_{k+i}) = \sum_{i=0}^{N-1} x_{k+i}^T Q x_{k+i} + \sum_{i=0}^{N-1} u_{k+i}^T Q u_{k+i} + x_{k+N}^T P x_{k+N} \tag{2.3}$$

It is common to have “stability constraints” to guarantee the stability or feasibility.

2.1.1 Recursive feasibility

The formulation of MPC is quite straight forward in many applications. MPC requires to repeatedly solve optimization problems online in order to decide the current input to the system. Although MPC opens up for general and advanced control scheme, it comes with a serious drawback. In contrast to the linear quadratic control that stabilizes by construction, the MPC is not guaranteed to be stable; see [1] for detail. In planning application of the MPC, MPC solution is interpreted as an optimal trajectory. In these applications, stability corresponds to the recursive feasibility problem. The recursive feasibility is well addressed in [25].

Definition 2.1.1. (Recursive feasibility). The MPC is recursively feasible if and only if for all initially feasible x_0 and for all optimal sequences of control inputs the MPC optimization problem remains feasible for all time.

One of the effort to archive recursive feasibility is to use bilevel programming [34]. The key feature which sets bilevel programming apart from standard optimization problems is that the inner variables z are constrained to be optimal with respect to an inner optimization problem which may depend on the outer variable x. Suppose we have following bilevel formulation.

$$\begin{aligned} &\min f(x, z^*) \\ s.t\ &(x, z^*) \in C \\ &z^* = \arg\min_z h(x, z) \\ &s.t.(x, z) \in D \end{aligned} \tag{2.4}$$

The bi-level problems are notoriously hard to solve, even in the case when both the inner and outer problems are linear programs, which pretty much is the simplest bilevel problem possible [35], [36]. In case of quadratic cost function,

$$\begin{aligned} &\min_{y, x_k, u_k} y^T(b - E(Ax_k + Bu^*)) \\ s.t.\ &y \geq 0, \quad y^T F = 0 \\ &U_k^* = \arg\min_{U_k} \frac{1}{2} U_k^T H U_k + U_k^T G x_k \\ &s.t E x_k + F U_k \leq b. \end{aligned} \tag{2.5}$$

The inner problem is a convex quadratic program, so we can replace the optimality condition with the corresponding KKT condition. Introduce a non-negative dual variable λ and append the outer problem with the stationarity, and feasibility and complementarity constraints of the inner problem.

$$\begin{aligned} &\min_{y,x_k,u_k,\lambda} y^T(b - E(Ax_k + Bu^*)) \\ s.t.\ &y \geq 0, \quad y^T F = 0 \\ &HU_k + Gx_k + F^T\lambda = 0 \\ &0 \leq \lambda \perp b - Ex_k - FU_k \geq 0. \end{aligned} \tag{2.6}$$

The above problem is hard to solve. Not only does it involve the complementarity constraints $\lambda \perp b - Ex_k - FU_k$, but it also involve a bilinear objective function. Numerically, this problem is severely ill-conditioned.

2.2 Semi-infinite optimization

Among semi-infinite optimization problems presented [27], [26], and [20], we consider the case when the problem is $\min\max$. Semi-infinite optimization minmax problem is of the following form

$$\min_{x \in R^n} \max_{j \in \mathbf{q}} \Psi^j(x), \tag{2.7}$$

where the functions $\Psi^j : R^n \to R$, $j = 0, 1, \cdots q$ are of the form

$$\Psi^j(x) = \max_{y_j \in Y_j} \phi^j(x, y_j), \tag{2.8}$$

with the function $\phi^j : R^n \times R^{m_j} \to R$ and the sets $Y_j \subset R^{m_j}$, $\mathbf{q} := \{1, 2, \cdots, q\}$. The reason such problems are called semi-infinite is that the design vector x is finite dimensional, but there are infinitely many functions $\phi^j(\cdot, y)$, determined by all the $y \in Y_j$ in the specification of these problem.

2.2.1 Method of outer approximations

The method of outer approximation has been used for the nonlinear optimization problems [37], [29], mixed-integer programming [38, 39], and nonconvex mixed-integer nonlinear programming [40]. The aim of these works is to solve a complex problem by appropriately discretizing it. One of the core methods in this dissertation to solve semi-infinite minmax optimization problem is the method of outer approximations. The role of the outer approximation in this dissertation is not only relaxing the problem but also solving it fast enough. We present its concept in this subsection. The origin of the method of outer approximation can be traced to cutting-plane methods for convex problems. To obtain an intuitive understanding, consider the simpler case of (2.7).

$$\Psi(x) = \max_{y \in Y} \phi(x, y), \tag{2.9}$$

where $\phi : R^n \times R^m \to R$ is continuous and $Y \subset R^m$ is compact. For any compact set $\Omega \subset Y$, let

$$\Psi_\Omega(x) = \max_{y \in \Omega} \phi(x, y). \tag{2.10}$$

Then for any $x \in R^n$, $\Psi_\Omega(x) \leq \Psi(x) = \Psi_Y(x)$, and hence the sub level sets of $\Psi(\cdot)$ are contained in the sub level sets of $\Psi_\Omega(\cdot)$. This fact is responsible for the name "outer approximation".

The simplest example of a conceptual method of outer approximations for solving the problem

$$\min_{x \in R^n} \max_{y \in Y} \phi(x, y). \tag{2.11}$$

Let us define maximizer for given x

$$\hat{Y}(x) := \arg\max_{y \in Y} \phi(x, y). \tag{2.12}$$

Following conceptual algorithm describes the method of outer approximations for solving (2.9).

Method of Outer Approximations (Conceptual Form)

Data. $x_0 \in \mathbb{R}^n$, $\hat{Y} \subset Y$.

Step 0. Set $k = 0$, choose a $y_0 \in \hat{Y}(x_0)$, set $Y_0 = \{y_0\} \cup \hat{Y}$.

Step 1. Compute

$$x_{k+1} \in \arg\min_{x \in X} \Psi_{Y_k}(x) \tag{2.13}$$

Step 2. Compute a $y_{k+1} \in \hat{Y}(x_{k+1})$, and set

$$Y_{k+1} = Y_k \ \cup \{y_{k+1}\}. \tag{2.14}$$

Step 3. Replace k by $k+1$, and go to Step 1.

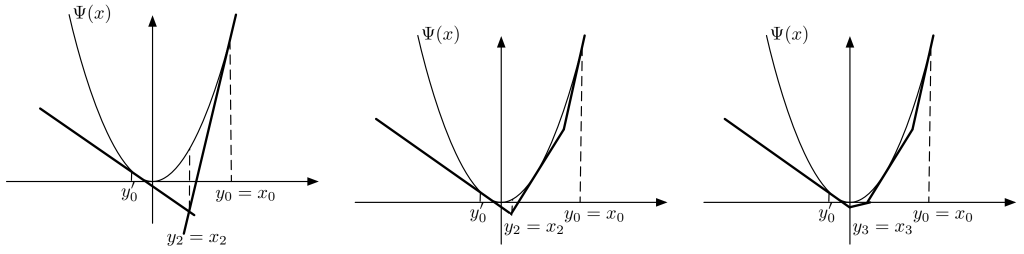
Example : To illustrate the behavior of the algorithm, suppose

$$\Psi(x) = x^2 \tag{2.13}$$

Then it is easy to see that the function $\Psi(\cdot)$ can be defined by its tangents, as follows

$$\Psi(x) = \max_{y \in R} y(2x - y) \tag{2.14}$$

Starting with an arbitrarily given x_0, we get $y_0 = x_0$ and add another point y_0' to form the set Ω_0. Three iterations of the algorithm applied to the problem (2.14) are shown in Fig. 2.1. We see that the algorithm converges rapidly to the solution $x = 0$.



(a) First iteration (b) Second iteration (c) Third iteration

Figure 2.1: Illustration of the iteration of the example.

2.3 Numerical optimization

In this section, we presents some important but non-standard numerical optimization concepts that are used throughout this dissertation.

2.3.1 Outer and inner semicontinuity

Definition 2.3.1. A set-valued function $f : R^n \times 2^{R^m}$ is said to be outer semicontinuous (o.s.c) at $\hat{x}$ if $f(\hat{x})$ is closed and, for every compact set S such that $f(\hat{x}) \cap S = \emptyset$, there exists a $\hat{\rho} > 0$ such that $f(x) \cap S = \emptyset$ for all $x \in B(\hat{x}, \hat{\rho})$, where $\emptyset$ denotes the empty set and $B(x, \rho)$ is a closed ball with a center at x, radius ρ.

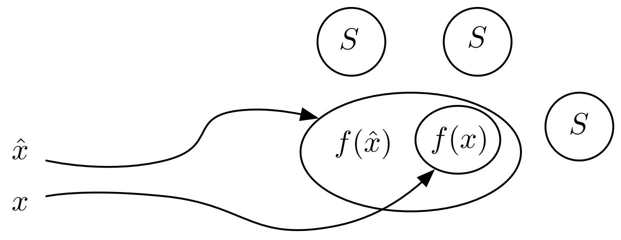


Figure 2.2: Outer semicontinuity

Definition 2.3.2. A set-valued function $f : R^n \times 2^{R^m}$ is said to be inner semicontinuous (i.s.c) at $\hat{x}$ if for every open set G such that $f(\hat{x}) \cap G = \emptyset$, there exists a $\hat{\rho} > 0$ such that $f(x) \cap G = \emptyset$ for all $x \in B(\hat{x}, \hat{\rho})$.

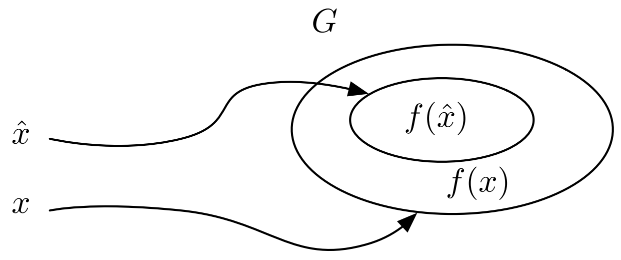


Figure 2.3: Inner semicontinuity

Fig 2.2 and 2.3 illustrates the concept of the outer and inner semicontinuity.

2.3.2 Directional derivatives and subgradients

It is clear from Fig 2.4 that max functions are not generally differentiable everywhere due to the cusp.

However, as we will now show, the max functions are locally Lipschitz continuous (lLc) and hence, by the Rademacher Theorem ([41]), they are differentiable almost everywhere. Also it is strongly suggested by Fig. 2.4

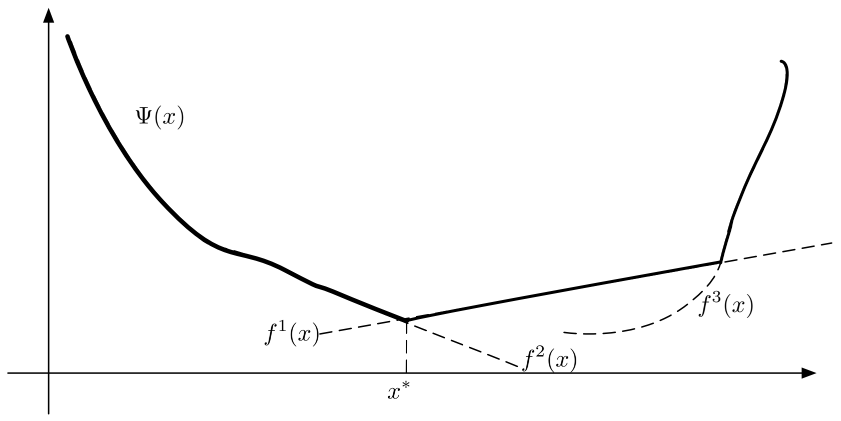


Figure 2.4: Max function with a non-differentiable points

that the directional derivative $d\Psi(x^*;h)$, where h is a direction vector, is equal to the largest of the directional derivatives of the functions that are "active" at x. The following result is from Danskin [42] and Demyanov [43], [44].

Theorem 1. *Consider the function*

$$\Psi(x) = \max_{j \in \mathbf{q}} f^j(x), \tag{2.15}$$

where the functions $f^j : R^n \to R$, $j \in \mathbf{q}$ are lLc and have directional derivatives $df^j(x;h)$ for all $x, h \in R^n$. Then
(a) $\Psi(x)$ is lLc, and
(b) the directional derivative $d\Psi(x;h)$ exists for all $x, h \in R^n$ and is given by

$$d\Psi(x;h) = \max_{j \in \hat{\mathbf{q}}(\mathbf{x})} df^j(x;h). \tag{2.16}$$

For the proof, see [20].

Lemma 1. *Consider the function $\Psi(x) = \max_{j \in \mathbf{q}} f^j(x)$, with $f^j : R^n \to R$, $j \in \mathbf{q}$, continuously differentiable. Then,*
(a) The generalized directional derivative $d^0\Psi(x;h)$ and the directional derivative $d\Psi(x;h)$ exist for all $x, h \in R^n$ and are given by

$$d^0\Psi(x;h) = d\Psi(x;h) = \max_{j \in \hat{\mathbf{q}}(\mathbf{x})} \left\langle \nabla f^j(x), h \right\rangle \tag{2.17}$$

(b) The directional derivative $d\Psi(\cdot;\cdot)$ is upper semicontinuous, and for every

$x \in R^n$, the function $d\Psi(x,;)$ is positively homogeneous, subadditive, and Lipschitz continuous. (c) The subgradient $\partial\Psi(x)$ of $\Psi(x)$ at $x \in R^n$ is given by

$$\partial\Psi(x) = co_{j\in\hat{\mathbf{q}}(\mathbf{x})}\{\nabla f^j(x)\}, \tag{2.18}$$

where coA is a convex hull of A.

2.3.3 PPP minmax algorithm

In this subsection, we discuss an algorithm that can be viewed as a natural extension of the Armijo Gradient Algorithm. Different versions of this algorithm were proposed by Pshenichnyi in 70's [45]. And later by Pironneau and Polak [46]. These two somewhat similar algorithms are combined and presented in [20]. This method can solve finite minmax problem which is a subclass of the semi-infinite problem.

Suppose the finite minmax problem is given as follows.

$$\min_{x\in R^n} \Psi(x), \tag{2.19}$$

with

$$\Psi(x) = \max_{j\in Q} f^j(x). \tag{2.20}$$

and the functions $f^j : R^n \to R$ continuously differentiable. Then the most fundamental first-order optimality condition for the local minimizer is that $d\Psi(\hat{x} : h) \geq 0$ for $\forall h \in R^n$. If $d\Psi(\hat{x} : h) > 0$ for $\forall h \in R^n$, $\hat{x}$ is a strict local minimizer.

To check the first-order optimality condition, we define the nonpositive-valued function $\theta : R^n \to R$ by

$$\theta(x) := \min_{\|h\|\leq 1} \max_{j\in\hat{q}(x)} \left\langle \nabla f^j(x), h \right\rangle \tag{2.21}$$

We see that the first-order optimality condition holds if and only if $\theta(x) = 0$. Rewriting (2.21) can be recasted as a linear program and hence, can be evaluated in a finite number of operations. Using the convex-relaxation and approximation which will be discussed in the next chapter, we find following optimality function.

$$\theta(x) = \min_{h\in R^n} \max_{j\in Q} f^j(x) - \Psi(x) + \langle \nabla f^j(x), h\rangle + \frac{1}{2}\delta\|h\|^2, \tag{2.22}$$

and

$$h(x) = \arg\min_{h\in R^n} \max_{j\in Q} f^j(x) - \Psi(x) + \langle \nabla f^j(x), h\rangle + \frac{1}{2}\delta\|h\|^2. \tag{2.23}$$

As appears in [27], (2.22) can be equivalently expressed as

$$\theta(x) = -\min_{\mu\in\Sigma_q} \Sigma_{j=1}^q \mu^j[\Psi(x) - f^j] + \frac{1}{2\delta}\|\Sigma_{j=1}^q \mu^j \nabla f^j(x)\|^2 \tag{2.24}$$

and

$$h(x) = -\frac{1}{\delta}\Sigma_{j=1}^q \mu_x^j \nabla f^j(x). \tag{2.25}$$

Pshenichnyi-Pironneau-Polak (PPP) Algorithm

Paramters. $\alpha \in (0,1]$, $\beta \in (0,1)$, $\delta > 0$.

Data. $x_0 \in R^n$.

Step 0. Set $i = 0$.

Step 1. Compute the optimality function $\theta_i := \theta(x_i)$ and search direction $h_i = h(x_i)$, i.e.,

$$\theta_i = -\min_{\mu\in\Sigma_q} \Sigma_{j=1}^q \mu^j[\Psi(x_i) - f^j] + \frac{1}{2\delta}\|\Sigma_{j=1}^q \mu^j \nabla f^j(x_i)\|^2$$

and

$$h_i = -\frac{1}{\delta}\Sigma_{j=1}^q \mu_{x_i}^j \nabla f^j(x_i),$$

where μ_x is any solution of θ_i.

Step 2. If $\theta_i = 0$, stop. Else, compute the step size.

$$\lambda_i = \lambda(x_i) := \arg\max_{k\in N}\{\beta^k \mid \Psi(x_i + \beta_k h_i - \Psi(x_i) - \beta^k\alpha\theta_i \le 0)\}. \tag{2.26}$$

Step 3. Set

$$x_{i+1} = x_i + \lambda_i h_i, \tag{2.27}$$

replace i by $i+1$ and go to Step 1.

As we will see in the next chapter, we will find the natural extension of PPP algorithm to the semi-infinite minmax problem and this method will be applied to the inner problem of the MPC formulation.

CHAPTER 3

SOLUTION METHODOLOGY

In this chapter, we present the numerical method to solve the problem (1.6) and its convergence. Since our approach is based on the numerical optimization techniques, we need to find searching direction, step size, and terminate condition. We use slightly simplified problem formulation from 1.6. The reason is that we are interested in the solution in certain discrete time t. Hence, there is no reason to include the variable t in analyzing a static situation. The modified formulation is the following. For given x_0 and y_0,

$$\begin{aligned}
&\min_{\mathbf{u}\in\mathbf{U}} \max_{\substack{\mathbf{v}\in\mathbf{V}(\mathbf{u})\\ k\in\{1,2,\cdots,H\}}} c(x_k, y_k) \\
&s.t.\ f^j(x_k,u_k) \leq 0, \quad j \in \mathbf{q} \\
&\qquad x_{k+1} = f_x(x_k,u_k), \\
&\qquad y_{k+1} = f_y(y_k,v_k),
\end{aligned} \tag{3.1}$$

where

$$\begin{aligned}
\mathbf{u} &= \{u_1, u_2, \cdots, u_H\} \\
\mathbf{v} &= \{v_1, v_2, \cdots, v_H\}.
\end{aligned} \tag{3.2}$$

and $\mathbf{q} = \{1,2,3,\cdots,q\}$ is a finite set. The Y team's constraint set

$$\mathbf{V}(\mathbf{u}) := \{\mathbf{v} \mid g^j(x_k,y_k,u_k,v_k) \leq 0\}, \tag{3.3}$$

is similarly defined as in chapter 1. Element wise inequality $g^j(\cdot,\cdot,\cdot,\cdot)$ are indexed by j, $j \in \{1,2,\cdots I\}$, with $k \in \{1,2,\cdots H\}$.

3.1 Problem separation

For concise notation let us define index set $\mathbf{H} = \{1, 2, \cdots, H\}$. We separate problem (3.1) into two problems: finite minmax problem and maximization problem. The concept of the method of outer approximation allows us to separate the semi-infinite problem. Below algorithm explains the concept of the outer approximation applied to the problem (3.1).

```
**Data**: $\mathbf{P}^1 = \{\hat{y}^0\}, \mathbf{I}_1 = \{1\}, x_0$
**Result**: $\hat{x}$
Set $i = 1$.
**if** *Terminate condition = false* **then**
    Step 1. Obtain $\hat{x}^i$ such that $\{x_m\}_{m=0}^{\infty} \to \hat{x}^i$ by solving finite
        minmax problem (3.4).
    Step 2. Solve $Y$ team maximization problem (3.5).
    Step 3. Update $\mathbf{P}^{i+1} = \mathbf{P}^i \cup \{\hat{y}^i\}$, $\mathbf{I}^{i+1} = \mathbf{I}^i \cup \{i+1\}$,
        Replace $i = i + 1$.
    Step 4. Evaluate terminate condition
**else**
    $\hat{x} = \hat{x}^i$
**end**
```

Algorithm 1: Problem separation algorithm

In the Algorithm 1, an index i is used to specify the iteration number of the algorithm. In each iteration, both team X and Y solves their problem and find solution $\hat{u}^i$ and $\hat{v}^i$. The trajectories $\hat{x}^i$ and $\hat{y}^i$ are obtained by feeding $\hat{u}^i$ and $\hat{v}^i$ to X and Y team's dynamics. Team X solves following finite minmax problem.

$$\begin{aligned} \hat{u}^i = &\arg \min_{\mathbf{u} \in \mathbf{U}} \max_{j \in \mathbf{I}^i} \quad c^j(x_k) \\ &s.t.\ f^l(u_k, x_k) \le 0,\ l \in \{1, 2, \cdots, q\}, \end{aligned} \tag{3.4}$$

where $c^j(x_k) := c(x_k, \hat{y}_k)$, with $\hat{y}_k \in \mathbf{P}^i$.

Team X assumes that team Y solves the following maximization problem.

$$\begin{aligned} \hat{v}^i = \arg &\max_{\substack{\mathbf{v} \in \mathbf{V}(\hat{\mathbf{u}}^{\mathbf{i}}) \\ k \in \mathbf{H}}} \quad c(\hat{x}_k^i, y_k) \\ s.t.\ &g^j(u_k, v_k, x_k, y_k) \le 0,\ j \in \{1, 2, \cdots, I\}, \end{aligned} \tag{3.5}$$

with a finite set $\mathbf{P}^i = \{\hat{y}^k\}_{k=0}^i$, where each $\hat{y}^k$ is a maximizer of team y. The set $\mathbf{P}^i$ is created such that $\mathbf{P}^{i-1} \subset \mathbf{P}^i$. $\mathbf{I}^i$ is an index set of $\mathbf{P}^i$. As algorithm iterates, set $\mathbf{P}^i$ monotonically increased. The goal of this chapter is to show that the sequence of the solution of (3.4), $\{x^i\}_{i=0}^{\infty}$ converges to the solution of (3.1). We also show that for any i, feasibility of the problem (3.1) is guaranteed.

Note that subscript is used to indicate inner sequence and superscript is used to indicate outer sequence.

Assumptions

Throughout this research, we assume that following two conditions are always satisfied.

1. For all $j \in \mathbf{q}$, f^j is twice differentiable and for some m and M such that $0 < m \leq M$, $m\|h\|^2 \leq \langle h, \nabla^2 f^j(x)h\rangle \leq M\|h\|^2$, for all h and $|f^j(x_{m+1}) - f^j(x_m)| \leq L\|x_{m+1} - x_m\|$ with $L > 0$.
2. For all $k \in \mathbf{I}$, c^k is twice differentiable and for some m and M such that $0 < m \leq M$, $m\|h\|^2 \leq \langle h, \nabla^2 \phi(x)h\rangle \leq M\|h\|^2$, for all h and satisfies $|c^k(x_{m+1}) - c^k(x_m)| \leq L\|x_{m+1} - x_m\|$ with $L > 0$.

These assumptions are about the existence of the hessians for the constraint functions and the cost functions, respectively. As we will see in the next sub chapter, the existence of the hessian allows us to find upper and lower bounds on optimality function.

3.2 Inner problem

In this section, we focus on Step 1 in Algorithm 1. Specifically, we consider the problem (3.4), in any fixed index i. Therefore, we omit i. We use index k to describe the iteration of the Phase I-Phase II method. Note that subscript is numerical optimization sequence in this chapter. For concise notation we represent (3.4) as follows in this section.

$$\begin{aligned} \min_{\mathbf{u}\in\mathbf{U}} \max_{j\in\mathbf{I}} \quad & c^j(x) \\ s.t.\ & f^l(u,x) \le 0, \quad \forall l \in \{1,2,\cdots,q\} \end{aligned} \tag{3.6}$$

The goal of this sections is to show that the sequence $\{x_m\}_{m=0}^{\infty}$ created by the Algorithm 1 to solve (3.4) converges to the solution of (3.1), $\hat{x}^i$, regardless of the feasibility of the initial point. First, we find searching direction.

3.2.1 Searching direction

In this section, we find searching direction of the optimization problem in (3.6).

Lemma 2. *The direction*

$$\hat{h}(x) = -\frac{1}{1+\gamma\mu_2^0}\left(\sum_{i=1}^{k}\mu_1^i \nabla c^i(x) + \gamma\mu_2^0 \sum_{j=1}^{q}\mu_2^j \nabla f^j(x)\right) \tag{3.7}$$

is the descent direction of the cost in (3.6) while satisfying feasibility, where $\gamma > 0$, $\mu_1 \in \Sigma_k$, *and* $\mu_2 \in \Sigma_{q+1}$.

The evaluation of the optimality function provides a constructive way of determining whether the necessary condition for the optimization problem is satisfied or not. If the starting point does not satisfy the feasibility, optimality function provides the direction toward the feasibility is satisfied while decreasing the cost. Let $\epsilon > 0$ small, for a given x_m, consider the following conceptual form of the optimality function.

$$\begin{aligned} \tilde{\theta}(x) := \min_{h\in B(0,\epsilon)} & \ \phi(x') - \phi(x) \\ s.t.\ & \Psi(x') \le 0 \end{aligned} \tag{3.8}$$

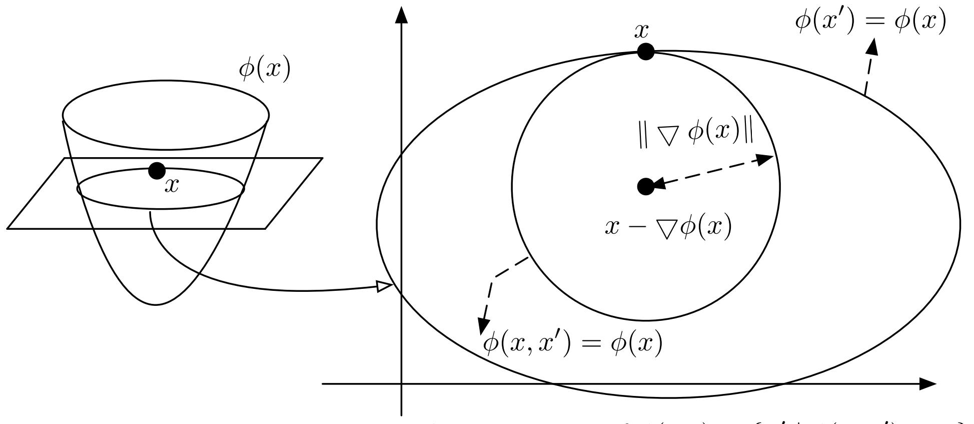
where $\phi(x) = \max_{k\in\mathbf{I}} c^k(x)$, $\Psi(x') = \max_{l\in\mathbf{q}} f^l(x')$, $x' = x + h$.

The solution (3.4) guarantees the feasibility while h_m that yields $\tilde{\theta}(x_m) < 0$ is a descent direction. $\tilde{\theta}(x_m)$ is well defined because $B(0,\epsilon)$ is compact.

Consider the following first order strictly convex approximation in h_m.

$$\begin{aligned}\phi(x') \approx \phi(x, x') :=& \phi(x) + \langle \nabla\phi(x), h\rangle + \frac{1}{2}\|h\|^2 \\ =& \max_{k\in\mathbf{I}} c^k(x) + \max_{k\in\mathbf{I}} \left\langle \nabla c^k(x), h\right\rangle + \frac{1}{2}\|h\|^2 \\ =& \max_{k\in\mathbf{I}} \left[c^k(x) + \left\langle \nabla c^k(x), h\right\rangle\right] + \frac{1}{2}\|h\|^2\end{aligned} \tag{3.9}$$

$$\{x' \mid \phi(x') = \phi(x)\} \text{ and } \{x' \mid \phi(x, x') = \alpha\}$$


$\phi(x)$
x
x
$\phi(x') = \phi(x)$
$\| \bigtriangledown \phi(x)\|$
$x - \bigtriangledown\phi(x)$
$\phi(x, x') = \phi(x)$

Equal cost contour of $\phi(x, \cdot) = \{x' \mid \phi(x, x') = \alpha\}$

$\rightarrow (x' - [x - \bigtriangledown\phi(x)])^2 = 2(\alpha - \phi(x)) + \| \bigtriangledown \phi(x)\|^2$

$\xrightarrow{\alpha=\phi(x)} (x' - [x - \bigtriangledown\phi(x)])^2 = \| \bigtriangledown \phi(x)\|^2$

Figure 3.1: Convex approximation of $\phi(x')$.

Fig. 3.1 presents the geometric interpretation of the convex approximation (3.9). Consider a current point x and an equal cost contour of the function $\phi(x) = \alpha$. We see that the set of x' such that $\phi(x) = \phi(x')$ is obviously the boarder of $\phi(x) = \alpha$. Since $x' - x = h$, we can rewrite (3.9) as follows.

$$\phi(x) + \nabla\phi(x)(x' - x) + \frac{1}{2}(x' - x)^2 = \alpha \tag{3.10}$$

This can be rearranged as follows

$$\begin{aligned}
&\phi(x) + \nabla\phi(x)x' - \nabla\phi(x)x + \frac{1}{2}(x'^2 - 2xx' + x^2) = \alpha \\
&x'^2 - 2x'(x - \nabla\phi(x)) + (x - \nabla\phi(x))^2 = 2\alpha - 2\phi(x) + 2\nabla\phi(x)x - x^2 + \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad (x - \nabla\phi(x))^2 \\
&(x' - [x - \nabla\phi(x)])^2 = 2(\alpha - \phi(x)) + \|\nabla\phi(x)\|^2 \\
&(x' - [x - \nabla\phi(x)])^2 = \|\nabla\phi(x)\|^2 \quad \textit{(when } \phi(x) = \alpha\textit{)}.
\end{aligned} \tag{3.11}$$

This shows that the convex approximation result in circular equal cost contour for the arbitrary shape of the equal cost contour. As the shape of the cross section of the $\phi(x)$ is close to the circle, this approximation works better.

We apply the above approximation to the optimality function (3.8). Following optimality function is a first order convex approximation of (3.8) which does not requires the ball.

$$\begin{aligned}
\theta(x) &= \min_{h \in R^{n\times N}} \phi(x, x') - \phi(x) \\
&= \min_{h \in R^{n\times N}} \left(\max_{k \in \mathbf{I}} \left[c^k(x) + \left\langle \nabla c^k(x), h \right\rangle \right] + \frac{1}{2}\|h\|^2 - \phi(x) \right) \\
&s.t.\ \Psi(x') \le 0
\end{aligned} \tag{3.12}$$

Using exact l_1 penalty function $\gamma\psi(x_{m+1})_+$, following optimality function is obtained.

$$\begin{aligned}
\theta(x) = \min_{h \in R^{n\times N}} \max_{k \in \mathbf{I}} &\left[c^k(x) + \left\langle \nabla c^k(x), h \right\rangle + \frac{1}{2}\|h\|^2 - \phi(x) \right] + \\
&\gamma \max \left[\max_{j \in \mathbf{q}} \left(f^j(x) + \left\langle \nabla f^j(x), h \right\rangle \right) + \frac{1}{2}\|h\|^2, 0 \right]
\end{aligned} \tag{3.13}$$

We can also conceptually represent optimality function as follows.

$$\theta(x) = \min_{h \in R^{n\times N}} F(x, x'), \tag{3.14}$$

where

$$F(x, x') = \phi(x, x') - \phi(x) + \gamma\Psi(x')_+. \tag{3.15}$$

The idea about optimality function is to add penalty term that prescribes a high cost to infeasible points. Note that this is only conceptual form because $F(x_m, x_{m+1})$ does not explicitly depends on h_m. We apply following convex combinations.

$$\max_{k\in\mathbf{I}} \left[c^k(x) + \left\langle \nabla c^k(x), h \right\rangle\right] = \max_{\mu_1\in\Sigma_k} \sum_{i=1}^{k} \mu_1^i \left[c^i(x) + \left\langle \nabla c^i(x), h\right\rangle\right] \tag{3.16}$$

and

$$\max_{j\in\mathbf{q}} f^j(x) + \left\langle \nabla f^j(x), h\right\rangle = \max_{\mu_2\in\Sigma_q} \sum_{j=1}^{q} \mu_2^j \left(f^j(x) + \left\langle \nabla f^j(x), h\right\rangle\right) \tag{3.17}$$

Note that for scalar a, $\max[a, 0] = \max_{\mu^0\in[0,1]} \mu^0 a$. Using the similar approximation to the (3.9),

$$\begin{aligned}\theta(x) = \min_{h\in R^{n\times N}} \max_{\mu_1\in\Sigma_k} \sum_{i=1}^{k} \mu_1^i \left[c^i(x) + \left\langle \nabla c^i(x), h\right\rangle + \frac{1}{2}\|h\|^2 - \phi(x)\right] \\ + \gamma \max_{\mu_2^0\in[0,1]} \left[\max_{\mu_2\in\Sigma_q} \mu_2^0 \sum_{j=1}^{q} \mu_2^j \left(f^j(x) + \left\langle \nabla f^j(x), h\right\rangle\right) + \frac{\mu_2^0}{2}\|h\|^2\right]\end{aligned} \tag{3.18}$$

Let

$$\begin{aligned}w(h) := \sum_{i=1}^{k} \mu_1^i \left[c^i(x) + \left\langle \nabla c^i(x), h\right\rangle + \frac{1}{2}\|h\|^2 - \phi(x)\right] \\ + \gamma\mu_2^0 \left[\sum_{j=1}^{q} \mu_2^j \left(f^j(x) + \left\langle \nabla f^j(x), h\right\rangle\right) + \frac{1}{2}\|h\|^2\right]\end{aligned} \tag{3.19}$$

Since $w(h)$ is convex in h and concave in μ_1 and μ_2,

$$\theta(x) = \max_{\substack{\mu_1\in\Sigma_k \\ \mu_2\in\Sigma_{q+1}}} \min_{h\in R^{n\times N}} w(h) \tag{3.20}$$

Taking $\frac{\partial w(h)}{\partial h} = 0$ yields

$$\hat{h}(x) = -\frac{1}{1+\gamma\mu_2^0}\left(\sum_{i=1}^{k}\mu_1^i\nabla c^i(x) + \gamma\mu_2^0\sum_{j=1}^{q}\mu_2^j\nabla f^j(x)\right) \tag{3.21}$$

$$\begin{aligned}\theta(x) = -\min_{\substack{\mu_1\in\Sigma_k\\ \mu_2\in\Sigma_{q+1}}}\sum_{i=1}^{k}\mu_1^i(\phi(x)-c^i(x)) - \gamma\mu_2^0\sum_{j=1}^{q}\mu_2^j f^j(x) + \\ \frac{1}{1+\gamma\mu_2^0}\|\sum_{i=1}^{k}\mu_1^i\nabla c^i(x) + \gamma\mu_2^0\sum_{j=1}^{q}\mu_2^j\nabla f^j(x)\|^2.\end{aligned} \tag{3.22}$$

Lemma 3. *Following holds.*
1. $\theta(x) \le 0$ if starting point is feasible, $\theta(x) \le \gamma\Psi(x)$ for non-feasible starting point.
2. There exists $\lambda_m > 0$ and $\alpha > 0$ such that $F(x, x+\lambda_m\hat{h}(x)) \le \lambda_m\alpha\theta(x)$.

Proof. 1. Consider equation (3.18). We observe that $w(0) = \gamma\Psi(x)_+$. Since $\theta(x)$ is smaller than any h, $\theta(x) \le \gamma\Psi(x)_+$.

2. Let us consider the minimizer $\hat{h}(x)$ in (3.21) and the situation when $\hat{h}(x)$ is applied in (3.13).

$$\theta(x) \ge \max_{k\in\hat{I}(x)}\left\langle\nabla c^k(x), \hat{h}(x)\right\rangle + \frac{1}{2}\|\hat{h}(x)\|^2 + \gamma\left[\max_{j\in\hat{q}(x')} f^j(x')\right]_+ \tag{3.23}$$

and

$$\theta(\hat{x}) \ge \max_{k\in\hat{I}(\hat{x})}\left\langle\nabla c^k(\hat{x}), \hat{h}(\hat{x})\right\rangle + \frac{1}{2}\|\hat{h}(\hat{x})\|^2 \tag{3.24}$$

Taking directional derivative of (3.15) to direction vector $\hat{h}(x)$,

$$d_2F(x, x'; \hat{h}(x)) = \max_{k\in\hat{I}(x')}\left\langle\nabla c^k(x'), \hat{h}(x)\right\rangle + \gamma\left[d\phi(x'; \hat{h}(x))\right]_+ \tag{3.25}$$

When $x = \hat{x}$,

$$d_2F(\hat{x}, \hat{x}; \hat{h}(\hat{x})) \le \theta(\hat{x}) - \frac{1}{2}\|\hat{h}(\hat{x})\|^2 \le 0 \tag{3.26}$$

For a given $0 < \beta < 1$ and for any $\epsilon > 0$ such that $0 < \alpha - \epsilon < 1$, $\exists k_e \in \mathbf{N}$ such that

$$\begin{aligned} F(\hat{x}, \hat{x} + \beta^{k_\epsilon} \hat{h}(\hat{x})) - F(\hat{x}, \hat{x}) &\leq \beta^{k_\epsilon}(\alpha - \epsilon) d_2 F(\hat{x}, \hat{x}; \hat{h}(\hat{x})) \\ &\leq \beta^{k_\epsilon}(\alpha - \epsilon) \left[\theta(\hat{x}) - \frac{1}{2}\|\hat{h}(\hat{x})\|^2 \right] \end{aligned} \tag{3.27}$$

Since $F(\hat{x}, \hat{x}) = 0$,

$$F(\hat{x}, \hat{x} + \beta^{k_\epsilon} \hat{h}(x)) - \beta^{k_\epsilon} \alpha \theta(\hat{x}) \leq -\beta^{k_\epsilon} \left[\epsilon \theta(\hat{x}) + \frac{1}{2}(\alpha - \epsilon)\|\hat{h}(x)\|^2 \right] \tag{3.28}$$

$\epsilon\theta(\hat{x}) + \frac{1}{2}(\alpha - \epsilon)\|\hat{h}(x)\|^2 \geq 0$ for all $\epsilon > 0$ such that

$$\frac{2\epsilon}{\alpha - \epsilon} \leq -\frac{\|\hat{h}(x)\|^2}{\theta(\hat{x})} := w^* \tag{3.29}$$

Then we see the following.

$$F(x_m, x_m + \beta^{k_{\hat{\epsilon}}} \hat{h}(x_m)) - \beta^{k_{\hat{\epsilon}}} \alpha \theta(x_m) \leq 0 \tag{3.30}$$

$\square$

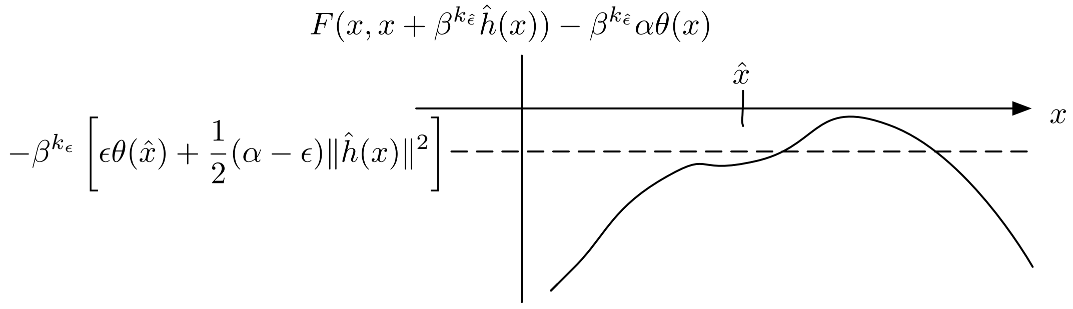

$F(x, x + \beta^{k_{\hat{\epsilon}}} \hat{h}(x)) - \beta^{k_{\hat{\epsilon}}} \alpha \theta(x)$
$\hat{x}$
x
$-\beta^{k_\epsilon} \left[\epsilon \theta(\hat{x}) + \frac{1}{2}(\alpha - \epsilon)\|\hat{h}(x)\|^2 \right]$

Figure 3.2: Existence of $B(\hat{x}, \rho)$

Fig.3.2 shows the inequality of (3.28) and (3.30).

3.2.2 Step size rule

Armijo type step size rule:

$$\hat{\lambda}_m = \max_{\lambda_m \in R_+} \{\lambda_m \mid F(x_m, x_m + \lambda_m \hat{h}(x_m)) \leq \lambda_m \alpha \theta(x_m)\} \tag{3.31}$$

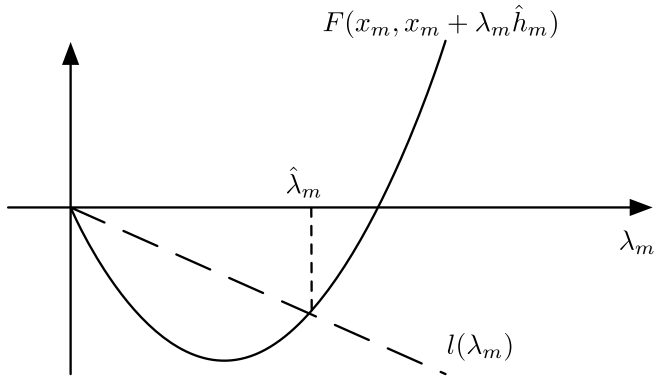


Figure 3.3: Armijo Step size rule

3.2.3 Convergence rate

Lemma 4. *Let* $\{x_m\}_{m=1}^{\infty}$ *be a sequence constructed using (3.21) and (3.31). Let* $\hat{x}$ *be any accumulation point. Then* $\theta(\hat{x}) = 0$.

Proof. From (3.15) and (3.30) $\phi(x, x') - \phi(x) \leq F(x, x + \lambda_m \hat{h}(x)) \leq \lambda_m \alpha \theta(x_m)$. We can observe that $\theta(x_m) \leq \theta(\hat{x})$ as follows.

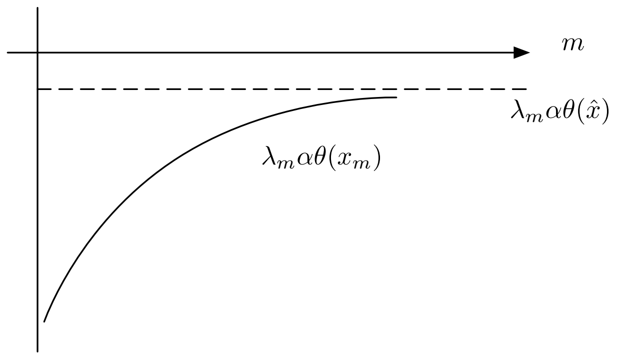

m
$\lambda_m \alpha \theta(\hat{x})$
$\lambda_m \alpha \theta(x_m)$

Figure 3.4: Upper bound of F and its convergence.

Because $\theta(x_m) \leq \theta(\hat{x})$. Together with (3.30), $\phi(x, x') - \phi(x) \leq \lambda_m \alpha \theta(\hat{x})$. From Lemma 2.1, $\theta(\hat{x}) \leq 0$. Suppose $\theta(\hat{x}) < 0$, then since $\phi(x)$ is a convex with a unique, finite minimizer and $x \to \hat{x}$, we conclude $\phi(x) \to -\infty$ as $m \to \infty$. Since ϕ is continuous, it contradicts the fact that $\phi(x) \to \phi(\hat{x})$. Therefore $\theta(\hat{x}) = 0$. $\square$

```
**Data**: $x_0$, $\gamma > 0$, $0 < \alpha < 1$
**while** $\theta(x_m) \neq 0$ **do**
    Compute $h(x_m)$
    Compute $\theta(x_m)$
    Compute $\lambda(x_m)$
    Update $x_{m+1} = x_m + \lambda_m h_m$
    Replace $m \leftarrow m + 1$
**end**
```

Algorithm 2: Inner algorithm

Theorem 2. *Suppose $\hat{x}$ is a unique solution of (3.6), then from the Algorithm 1, $\{x_m\}_{m=0}^{\infty}$ converges to $\hat{x}$.*

Proof. Algorithm guarantees the existence of k such that $\{x_m\}_{m=k}^{\infty} \subset S_0(\psi)$. From assumption.3, $S_0(\psi)$ is compact. Therefore, $\{x_m\}$ is bounded. Also, $\hat{x}$ is the unique zero of $\theta(.)$ and $\{h_m\}_{m=0}^{\infty}$ is the direction that results in $\theta(x_m) \to 0$. Therefore, $\{x_m\}_{m=0}^{\infty} \to \hat{x}$. $\square$

Following lemma presents the exponential convergence of the inner problem The approximation technique and assumptions that is used here is found in [47]. In this book, the condition that is called "Self-concordance" is the following.

$$mI \preceq \nabla^2 f(x) \preceq MI, \quad \|\nabla^2 f(x) - \nabla^2 f(y)\|_2 \leq L\|x - y\|_2$$

The reason for introducing this assumption is to compensate the classical convergence analysis of Netwon's method. The main reason is that the unknown parameters, m, M, L are generally impossible to obtain. Another drawback is that Newton's analysis method is heavily depends on the coordinate system used. If the coordinate is changed, m, M, L should all be changed accordingly. To overcome this, self-concordance condition is introduced and it is important in three reasons.

First, self-concordance functions include many of logarithmic barrier functions that is important approximation to the exact penalty functions. Second, self-concordance functions do not depend on the unknown parameters when the Newton's analysis method is applied. Three, they are affine-invariant. Therefore, is is still self-concordance after the linear transform.

Lemma 5. *Suppose $\psi(x_0) \leq 0$ and $\{x_m\}_{m=0}^{\infty} \to \hat{x}$ is constructed using Algorithm 1. Then $\phi(x_{m+n}) - \phi(\hat{x}) \leq C^n[\phi(x_m) - \phi(\hat{x})]$, where $C = \frac{\lambda^2 M^2}{2} \leq 1$.*

Proof. From Taylor's theorem,

$$\phi(x_{m+1}) - \phi(x_m) = \\ \langle \nabla\phi(x_m), x_{m+1} - x_m\rangle + \frac{1}{2}\left\langle x_{m+1} - x_m, \nabla^2\phi(x_m + s(x_{m+1} - x_m))(x_{m+1} - x_m)\right\rangle \tag{3.32}$$

for some $s \in [0, 1]$. From assumption 2,

$$\begin{aligned}\phi(x_{m+1}) - \phi(x_m) &\leq \langle \nabla\phi(x_m), x_{m+1} - x_m\rangle + \frac{1}{2}M\|x_{m+1} - x_m\|^2 \\ &= \frac{1}{M}\left[\langle \nabla\phi(x_m), M(x_{m+1} - x_m)\rangle + \frac{1}{2}\|M(x_{m+1} - x_m)\|^2\right].\end{aligned} \tag{3.33}$$

Taking minimum on both side yields

$$\phi(\hat{x}) - \phi(x_m) \leq \frac{1}{M}\theta(x_m). \tag{3.34}$$

Since $\psi(x_0) \leq 0$, switching parameter $\mu_2^0 = 0$. Therefore, from (3.21) and (3.22), $\theta(x_m) = -\min_{\mu_1\in\Sigma_k}\sum_{i=1}^k \mu_1^i(\phi(x_m) - c^i(x_m)) + \|h_m\|^2$. Let minimizer $\hat{\mu}_1$ be applied, then $\theta(x_m) = \sum_{i=1}^k \hat{\mu}_1^i(c^i(x_m) - \phi(x_m)) - \|h_m\|^2$. Since $c^i(x_m) - \phi(x_m) \leq 0$ for all i, $\theta(x_m) \leq -\|h_m\|^2$. Therefore, from (3.34)

$$\phi(x_m) - \phi(\hat{x}) \geq \frac{1}{M}\|h_m\|^2. \tag{3.35}$$

Next, from (3.30), $F(x_m, x_m + \lambda_m\hat{h}_m) \leq \lambda_m\alpha\theta(x_m)$. Then from (3.15), $\phi(x_{m+1}) - \phi(x_m) - \lambda_m\alpha\theta(x_m) \leq 0$. Then again from the Taylor's theorem,

$$\begin{aligned}&\phi(x_{m+1}) - \phi(x_m) - \lambda_m\alpha\theta(x_m) \\ &= \langle \nabla\phi(x_m), \lambda_m h_m\rangle + \frac{1}{2}\left\langle \lambda_m h_m, \nabla^2\phi(x_m + s(x_{m+1} - x_m))\lambda_m h_m\right\rangle - \lambda_m\alpha\theta(x_m) \\ &\leq \langle \nabla\phi(x_m), \lambda_m h_m\rangle + \frac{M}{2}\|\lambda_m h_m\|^2 - \lambda_m\alpha\theta(x_m)\end{aligned} \tag{3.36}$$

When $x_m = \hat{x}$, $\theta(\hat{x}) = 0$ and $\nabla\phi(\hat{x}) = 0$. Therefore,

$$\phi(x_{m+1}) - \phi(\hat{x}) \leq \frac{M}{2}\lambda_m^2\|h_m\|^2 \tag{3.37}$$

From (3.35) and (3.37),

$$\phi(x_{m+1}) - \phi(\hat{x}) \leq C[\phi(x_m) - \phi(\hat{x})] \tag{3.38}$$

, where $C = \frac{\lambda^2 M^2}{2}$. Note that $C < 1$ when $\lambda M < \sqrt{2}$. By induction,

$$\phi(x_{m+n}) - \phi(\hat{x}) \leq C^n[\phi(x_m) - \phi(\hat{x})] \tag{3.39}$$

This shows exponential convergence. □

3.3 Outer problem

The aim of this section is to show that the sequence of solution of the outer problem, (3.5) converges to the global solution. Recall the following Y team problem. Recall that X player assumes that Y player solves below maximization problem.

$$\begin{aligned} &\hat{v}^i = \arg \max_{\substack{v \in Y(\hat{u}^i) \\ k \in \mathbf{N}}} \quad c(\hat{x}^i_k, y_k) \\ &s.t.\ f^l(u_k, v_k, x_k, y_k) \leq 0,\ l \in \{1, 2, \cdots, q\}, \end{aligned} \tag{3.40}$$

Since the maximum in a finite set of scalar values are equal to the maximum in the convex combinations of these scalar values, following is true.

$$\max_{k \in \mathbf{N}} c(\hat{x}^i_k, y_k) = \max_{\mu \in \Sigma_N} \mu^k c(\hat{x}^i_k, y_k). \tag{3.41}$$

If we augment control and multipliers for the convex combinations $v' := (v, \mu)$, $c'(\hat{x}^i_k, y_k) := \mu^k c(\hat{x}^i_k, y_k)$, we can rewrite (3.40) in the following equivalent form.

$$\begin{aligned} &\hat{v}^i = \arg \max_{v' \in Y(\hat{u}^i)} \quad c'(\hat{x}^i_k, y_k) \\ &s.t.\ f^l(u_k, v_k, x_k, y_k) \leq 0,\ l \in \{1, 2, \cdots, q\} \end{aligned} \tag{3.42}$$

with a finite set $\mathbf{P}^i = \{y^k\}^i_{k=0}$, where each y^k is a maximizer of team y. The set $\mathbf{P}^i$ is created such that $\mathbf{P}^{i-1} \subset \mathbf{P}^i$. $\mathbf{I}^i$ is an index set of $\mathbf{P}^i$.

Let us define following max function.

$$\phi(x) := \max_{v' \in Y(u)} \quad c'(x, y) \tag{3.43}$$

and

$$\phi_{\mathbf{P^i}}(x) := \max_{v' \in \mathbf{P^i}} \quad c'(x, y) \tag{3.44}$$

```
**Data**: $P_0, I_0, x_0$, $\gamma > 0$, $0 < \alpha < 1$
**Result**: $\hat{x}$
Set $i = 0$.
**if** $t_i \geq \epsilon_t$ **then**
    Step 1. Obtain $\hat{x}^i$ such that $\{x_m\}_{m=0}^{\infty} \to \hat{x}^i$ by solving inner problem:
        $$\min_{x \in R^{n \times N}} \max_{k \in \mathbf{I_i}} \quad c^k(x) \qquad s.t.\ f^j(x) \leq 0, \quad j \in \mathbf{q}$$
    Step 2. Solve team y maximization problem:
        $$\hat{y}^i = \arg \max_{y^i \in Y(\hat{x}^i)} \quad c(\hat{x}^i, y^i)$$
    Step 3. Update $\mathbf{P}^{i+1} = \mathbf{P}^i \cup \{\hat{y}^i\}$, $\mathbf{I}_{i+1} = \mathbf{I}_i \cup \{i+1\}$, Replace $i = i + 1$.
    Step 4. Evaluate terminal condition: $t_i = \|\hat{x}^i - \hat{x}^{i-1}\|$
    Step 5. goto Step 1.
**else**
    $\hat{x} = \hat{x}^i$
**end**
```

Algorithm 3: Outer algorithm

Lemma 6. *Suppose that the Algorithm 1 has constructed an infinite sequence* $\{\hat{x}^i\}_{i=1}^{\infty}$, *in solving (3.5). If* $\{\hat{x}^i\}_{i=1}^{\infty} \to \hat{x}$, *then* $\phi_{\mathbf{P}_i}(\hat{x}^i) \to \phi(\hat{x})$

Proof.

$$\phi(\hat{x}^i) \geq \phi_{\mathbf{P}_i}(\hat{x}^i) \geq c'(\hat{x}^i, \hat{y}^{i-1}) \tag{3.45}$$

Note that $\hat{y}^{i-1}$ is a maximizer in $i - 1^{th}$ iteration in Algorithm 1. Since $\phi(\cdot)$ is a continuous function,

$$\phi(\hat{x}^i) \to \phi(\hat{x}) \tag{3.46}$$

Since $c'(\cdot)$ is a continuous function,

$$|c'(\hat{x}^i, \hat{y}^{i-1}) - c'(\hat{x}^{i-1}, \hat{y}^{i-1})| \to 0 \quad \text{as} \quad i \to \infty. \tag{3.47}$$

Because $c'(\hat{x}^{i-1}, \hat{y}^{i-1}) = \phi(\hat{x}^{i-1})$,

$$c'(\hat{x}^i, \hat{y}^{i-1}) \to \phi(\hat{x}^{i-1}) \to \phi(\hat{x}) \quad \text{as} \quad i \to \infty. \tag{3.48}$$

Therefore,

$$\phi_{\mathbf{P}_i}(\hat{x}^i) \to \phi(\hat{x}) \tag{3.49}$$

□

Theorem 3. *Suppose Algorithm 1 constructed* $\{\hat{x}^i\}_{i=1}^{\infty}$. *If* $\hat{x}$ *is an accumulation point, then* $\hat{x}$ *is a minimizer of (3.5).*

Proof. From *Lemma*.5,

$$\phi_{\mathbf{P}_i}(\hat{x}^i) \to \phi(\hat{x}) \tag{3.50}$$

as $\{\hat{x}^i\}_{i=1}^{\infty} \to \hat{x}$. Let $\hat{v} = \min_{x \in R^{n\times N}} \phi(x)$. Now, suppose $\hat{x}$ is not a minimizer of $\phi(x)$. Then we see that there exists $i' \in [1, \infty]$ such that $\phi_{\mathbf{P}_{i'}}(x) > \phi(x)$ for some x. This contradict $\phi_{\mathbf{P}_{i'}}(x) \leq \phi(x)$. Therefore, $\hat{x}$ is a minimizer of $\phi(x)$. Proof is illustrated in Fig.3.5. □

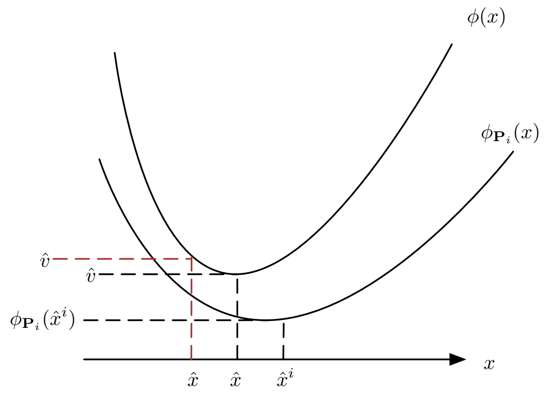


Figure 3.5: Illustration of proof: $\hat{x}$ is an unique minimizer.

3.4 Examples

Consider X player and Y player with the following dynamics.

$$\begin{aligned} \dot{x}(t) &= u(t), \\ \dot{y}(t) &= v(t), \end{aligned} \tag{3.51}$$

where x and y are position of the player X and Y, u and v are their controls, respectively. Discretized dynamics are $x_{k+1} = x_k + \Delta u_k$ and $y_{k+1} = y_k + \Delta v_k$.

Let horizon length be N and consider the following X player problem at certain sampling time of MPC.

$$\begin{aligned} &\min_{U \in R^N} \max_{\substack{V \in Y(x_k) \\ k \in \mathbf{N}}} y_k \{2x_k - y_k\} \\ &s.t. \quad x_k \leq 1, \end{aligned} \tag{3.52}$$

where $\mathbf{N} = \{1, 2, \cdots, N\}$. Let the set of the sequence of the feasible control of Y player be $Y(x_k) := \{V = \{v_1, v_2, \cdots, v_N\} \mid x_k \leq y_{k+1}, \forall k \in \mathbf{N}\}$. Sequence of controls for X players is $U = \{u_1, u_2 \cdots, u_N\}$.

We observe that the best sequence of control for Y player is the one that matches the position of X player, *i.e.* $y_k \to x_k$. Noticing this, X player's best controls are the one that yields $x_k \to 0$.

Fig. 3.6 shows the trajectories for the X and Y players.

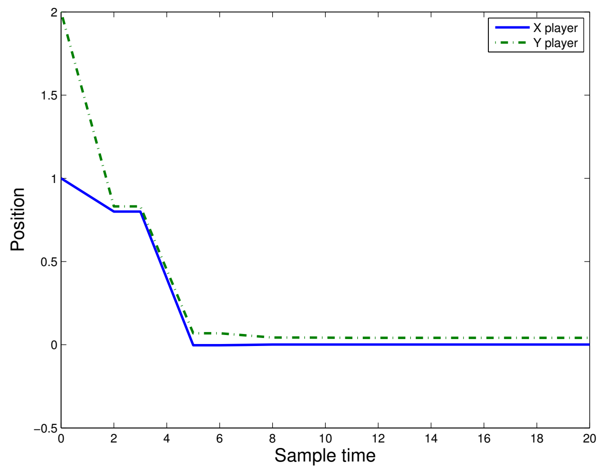


Figure 3.6: Trajectories of X and Y player.

3.5 Comparison to Polak-He method

In this subchapter, we compare our Phase I-Phase II algorithm to the state-of-the-art methodology called PH(Polak-He) algorithm [20]. PH algorithm is developed for the robustness by sacrificing the accuracy and the computational speed. This is the primary reason for developing our own Phase I-Phase II method as described in this chapter.

PH algorithm starts by constructing the following max functions, $F(z, x$, that the algorithm wishes to minimize.

$$F(z,x) := \max\left\{f^0(x) - f^0(z) - \gamma\Psi(z)_+, \Psi(x) - \Psi(z)_+\right\}, \tag{3.53}$$

where z is a current state and x is a next state. When current state is in a feasible region, $F(z,x)$ becomes

$$F(z,x) = \max\left\{f^0(x) - f^0(z), \Psi(x)\right\}. \tag{3.54}$$

The following situation is possible. Suppose that the initial steepest descent direction of $F(z,x)$ is more toward the $\Psi(x)$ than $f^0(x)$. Then initial searching direction is not desirable in a sense of fast descent of $f^0(x)$.

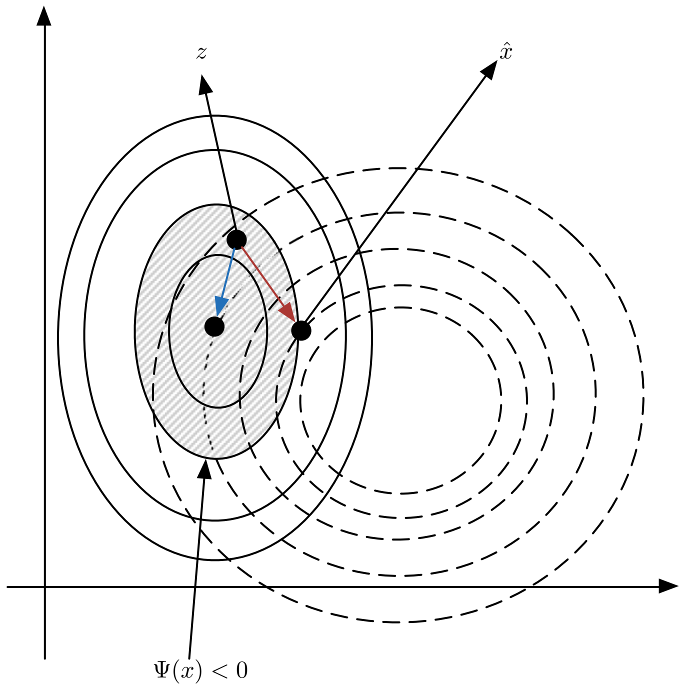
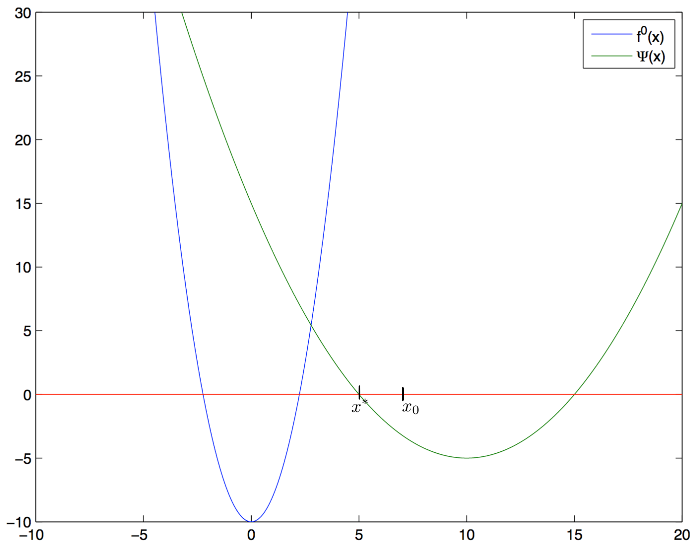


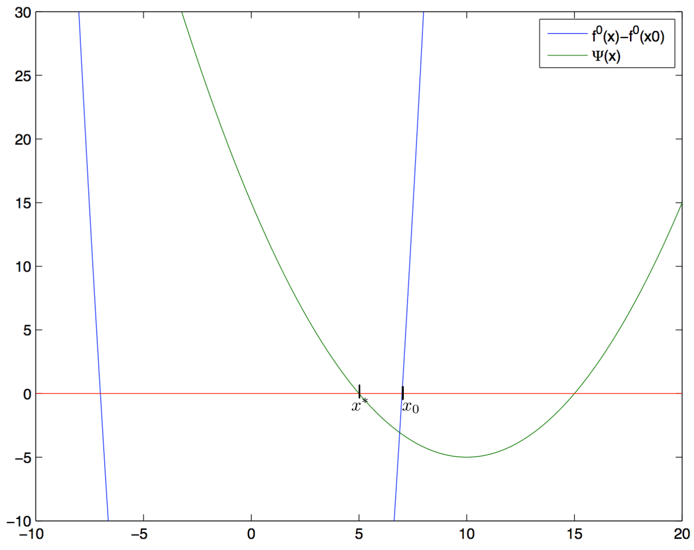
Figure 3.7: Contours for the $f^0(x)$ and violation function $\Psi(x)$ when $\Psi(z) \leq 0$: Solid curves and dotted curves are equal cost contours for the constraint violation function $\Psi(x)$ and the cost function $f^0(x)$, respectively.

Fig.3.7 illustrates the situation. Shaded area is the feasible region. Solid curves and dotted curves are equal cost contours for the constraint violation function $\Psi(x)$ and the cost function $f^0(x)$, respectively. Red line decreases the cost more than the blue line. Below example describes the sequence of finding the solution of PH algorithm.

The Fig.3.8 shows the first two steps of minimizing $F(z,x)$ when $z = x_0$. It minimizes $F(x_0,x) = \max\left\{f^0(x) - f^0(x_0), \Psi(x)\right\}$. Fig.3.8 (a) is the plot of the



(a) Cost and constraint functions, initial state

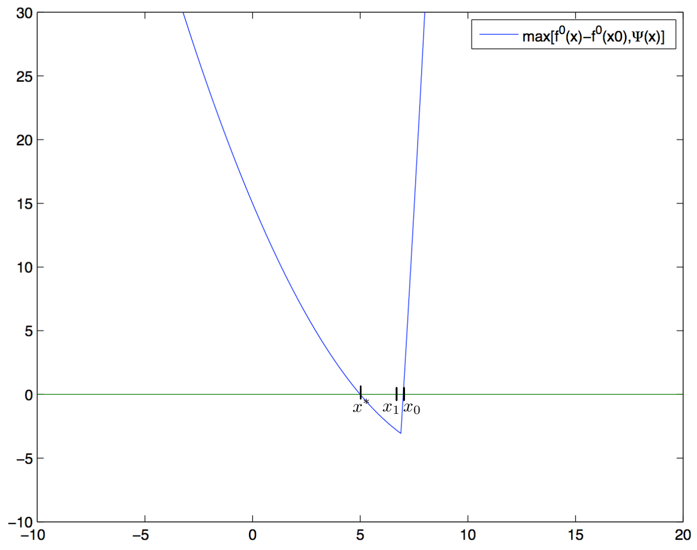


(b) Shifted cost function

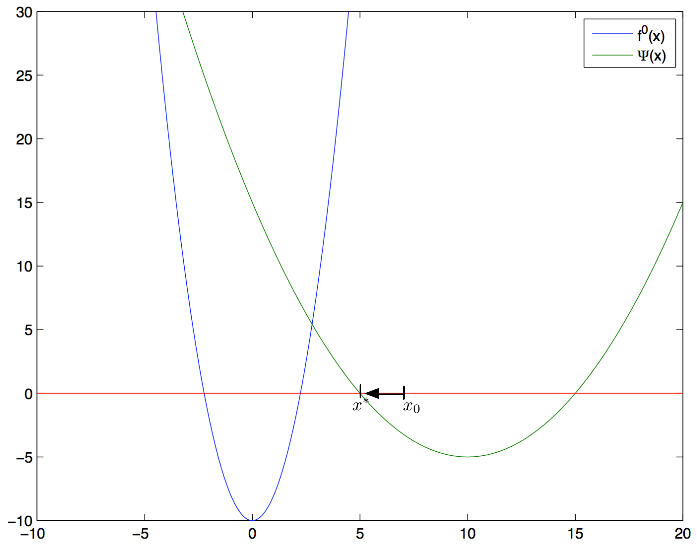
Figure 3.8: First two steps of $\min\max\{f^0(x) - f^0(x_0), \Psi(x)\}$

cost function, $f^{0(x)}$ and the max constraint function $\Psi(x)$. We observe that the solution x^* is on the boundary of the $\Psi(x)$. Fig.3.8 (b) shows the plots of $f^{0(x)} - f^0(x_0)$ and $\Psi(x)$.

The Fig.3.9 (a) shows the final step of the PH method for finding next solution point x_1. Since it minimizes the max function $F(x_0, x)$, it finds the next point



(a) max function and its minimizer



(b) Alternative approach

Figure 3.9: Final step of $\min\max\{f^0(x) - f^0(x_0), \Psi(x)\}$ and alternative method

x_1 that is not directly towards x^*. However, it is possible to steer the next step directly toward x^* as shown in The Fig.3.9 (b) using the our method that directly minimizes $f^0(x)$, s.t. $\Psi(x_1) \leq 0$.

Now we present three examples of comparison between the proposed method noted as FPH(Fast PH) method and PH method. All parameters in PH method are as in [27].

Example 1:

The cost function and the constraint function are both convex functions as given below.

$$\begin{aligned} f^0(x) &= 2x^2 - 1 \\ f^1(x) &= 0.5(x-5)^2 - 3 \end{aligned} \tag{3.55}$$

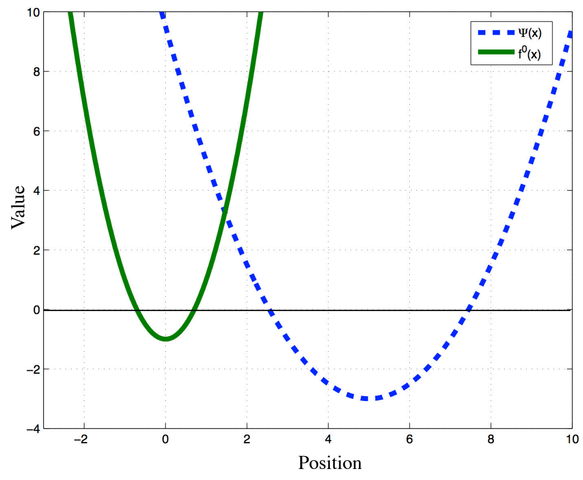
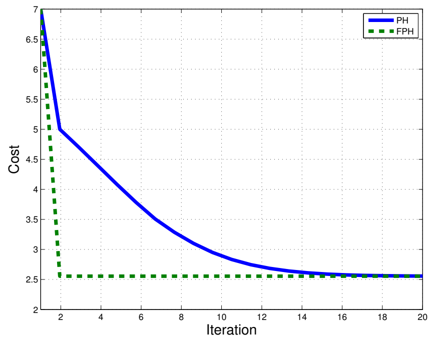
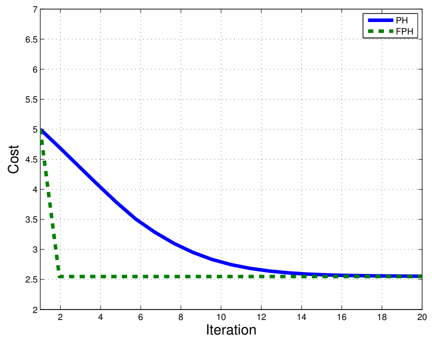


Figure 3.10: Example 1: Cost function and constraints

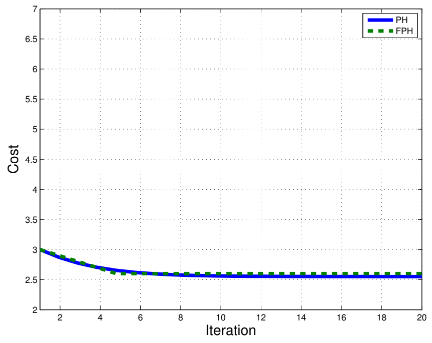
We observe that $\Psi(x) = 0.5(x-5)^2 - 3$ and the feasible region is $[2.55, 7.45]]$. The Solution is on the boundary of the feasible region. We present the three cases with different initial values in a feasible region $x_0 = 7, 5,$ and 3.



(a) $x_0 = 7$



(b) $x_0 = 5$

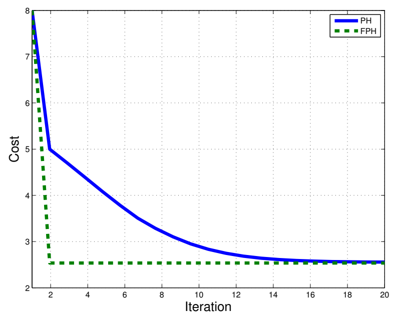


(c) $x_0 = 3$

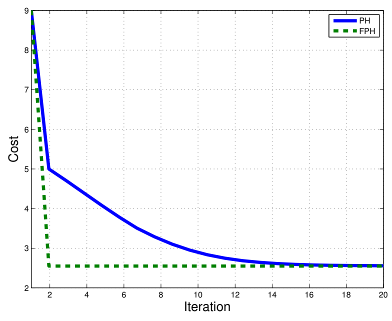
Figure 3.11: Example 1: Feasible starting points $x_0 = 7, 5$, and 3.

As we see from Fig. 3.11 (a) to (c), the proposed method result in faster convergence to the solution. Throughout the experiment, we observe that alternative method notated as FPH (Fast PH) outperforms PH method.

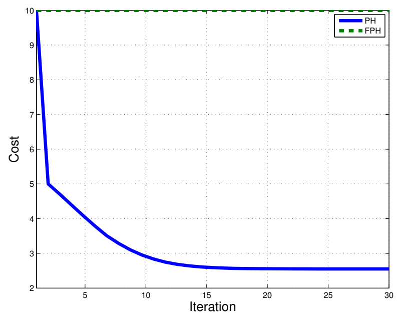
In next experiment, initial values are not in feasible region $x_0 = 8, 9$, and 10. We observe from Fig. 3.12 that FPH outperforms PH in case of (a) and (b). However, as the initial value gets farther away from the feasible region, $x_0 = 10$, FPH method cannot handle the feasibility. PH method still able to bring the state to the feasible region as in (c).



(a) $x_0 = 8$



(b) $x_0 = 9$



(c) $x_0 = 10$

Figure 3.12: Example 1: Infeasible starting points $x_0 = 8, 9$, and 10.

Example 2:

Second example to show the efficiency of the proposed method over the PH algorithm is following convex cost function with upper and lower bounded hessian $f^0(x)$ and non differentiable max constraint function as follows.

$$\begin{aligned} f^0(x) &= 10(x-0.5)^2 - 20 \\ f^1(x) &= -15e^{-x} + 2 \\ f^2(x) &= -15e^{x} + 2 \end{aligned} \tag{3.56}$$

Max constraint function $\Psi(x) = \max[-15e^{-x} + 2, 15e^{x} + 2]$ and the feasible region is $[-2.0149, 2.0149]$. The shape of the cost and the max constraint function is presented in Fig. 3.13.

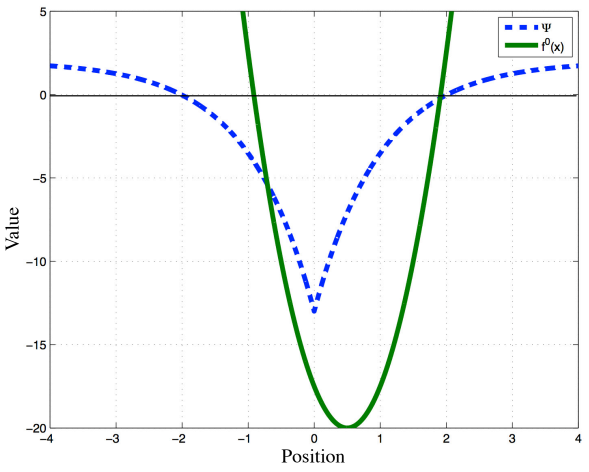
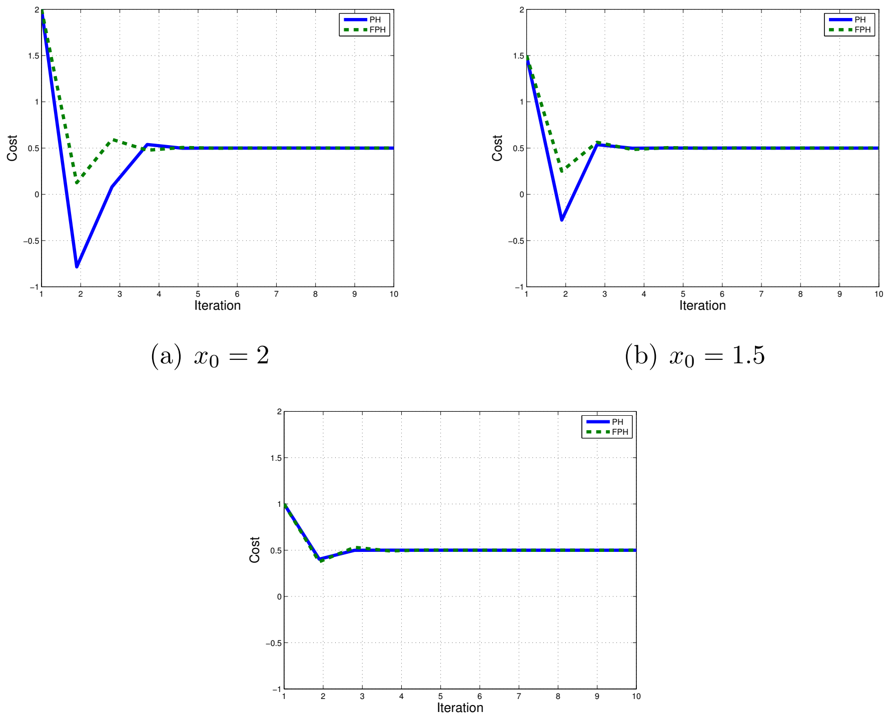


Figure 3.13: Example 2: Shape of the cost function $f^0(x)$ and the max constraint function $\Psi(x)$

Fig. 3.14 shows examples with different feasible initial values $x_0 = 2, 1.5$ and 1. Similar to the example 1, we observe that the FPH outperforms PH.



(a) $x_0 = 2$

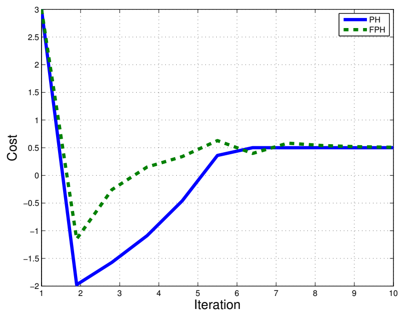
(b) $x_0 = 1.5$

(c) $x_0 = 1$

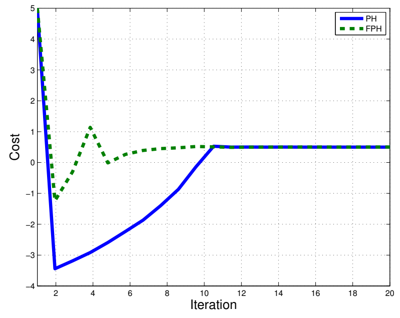
Figure 3.14: Example 2: Feasible starting points $x_0 = 2, 1.5$, and 1.

Next, Fig. 3.15 shows the case when the initial values are not in feasible region

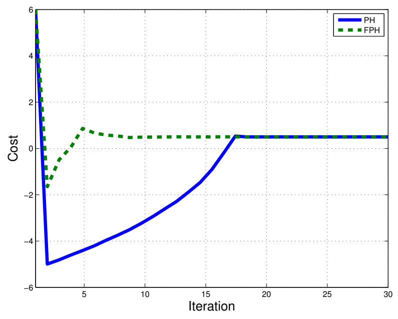
$x_0 = 3, 5, 6$, and 7.



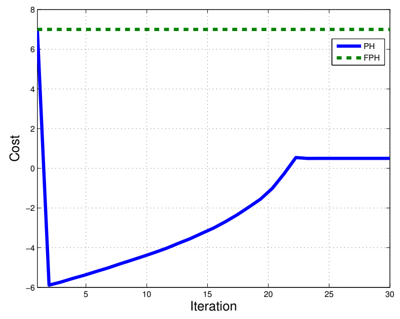
(a) $x_0 = 3$



(b) $x_0 = 5$



(c) $x_0 = 6$



(d) $x_0 = 7$

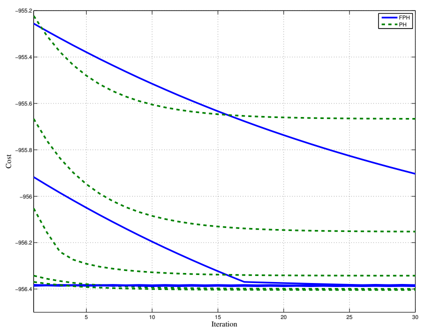
Figure 3.15: Example 2: Infeasible starting points $x_0 = 3, 5, 6$, and 7.

We observe the similar result as in Example 1. FPH result in faster convergence but less robustness with respect to the feasibility.

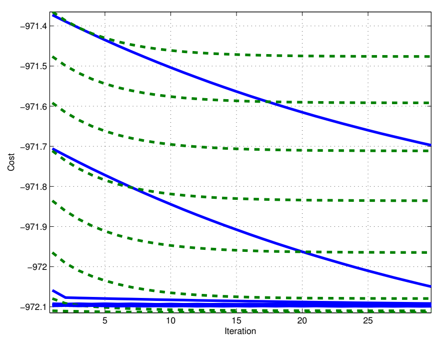
Example 3:

In this example we present convergence rate of the harbor defense problem that will be mainly discussed in the next chapter. In Fig. 3.16, both the inner problem convergence and the outer problem convergence are presented. Overall solution is obtained in the most iterated outer problem.

Fig.3.17 presents the number of iterations required to obtain the solution of the harbor defense problem. Experiments are performed for 50 consecutive sample times. The terminate condition for the iteration is that the difference of the new cost function and the current cost function is $< 0.1\%$. Three of such experiments are performed and the average data is presented. The average number of required iterations for PH=6.54 and FPH=2.88.



(a) Outer iterations: case 1



(b) Outer iterations: case 2

Figure 3.16: Example 3: Convergence of the outer iterations of the harbor defense problem.

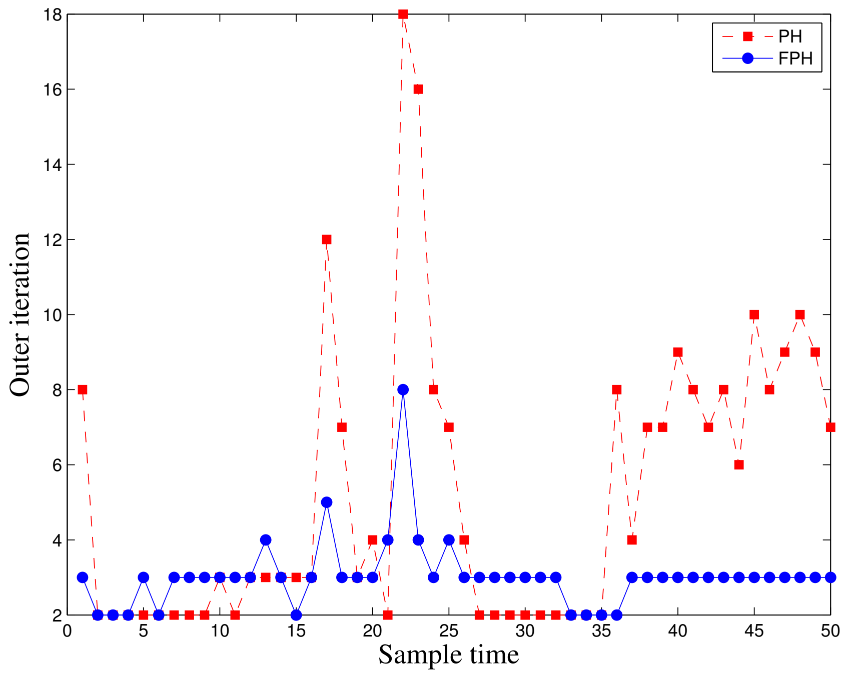


Figure 3.17: Example 3: Comparison of the number of iterations of the outer problem.

CHAPTER 4

THE HARBOR DEFENSE PROBLEM

4.1 Introduction

In recent years, small unmanned vehicles have become inexpensive and deadly weapons. It is easy to imagine a scenario where a small unmanned explosives-packed submarine is launched by terrorists from a freighter, at a safe distance from the entrance to a harbor, with the mission of destroying a large cruise ship carrying many thousands of passengers. The effect could be as devastating as the 9/11, 2001 attack on the World Trade Center in New York.

The thwarting of such an attack can be viewed as a pursuit-evasion game, but not one with a set of pre-specified mathematical rules, since it is not a gentlemen's game. For the purpose of this work, we assume that the floor of the harbor is seeded with sensors that enable the defending team to determine continuously the position of the intruder, that the intruder can be destroyed with very high probability if it comes within a distance δ of a defending vehicle, which can also be an unmanned submarine, or unmanned hovercraft, or drone. Within this scenario, the intruder aims to achieve its goal by outmaneuvering the defender. Just to be safe, we assume that the intruder can determine the location of the defender.

Clearly, if we rename the defender "pursuer" and the intruder "evader", we see that this is a modern day version of the pursuit-evasion games that have been studied extensively since the pioneering work of Isaacs [48]. See [49] for a presentation of the theory of those games. In a classical pursuit-evasion game, a defender tries to capture an intruder while the intruder tries to reach a target while avoiding being captured. For example, consider a game, in a two dimensional plane, in which the intruder wins if he manages to reach a target set $\mathcal{T}$ while maintaining a distance larger than δ from the defender; otherwise, the defender wins.

There are two important features in which our study of the harbor defense problem differs from that of a classical pursuit-evasion game. The first is that we assume that the trajectories of the defender and intruder are constrained to remain in a rectangular region, and that there are hard bounds on the strategies (accelerations) that they can employ. Neither are a part of a classical pursuit-evasion game. The second is in the type of result that one is trying to obtain. The questions that we are asking now are only approachable because of the enormous progress made in digital computers, in optimization algorithms, and dynamic system control methods since the middle 1960's.

The classical study of a pursuit-evasion game consists of examining the set $\mathcal{E}$ of initial states from which the intruder can win and of calculating the boundary of that set, called the *barrier*. The barrier is characterized by differential equations that express the fact that the intruder cannot move from outside of $\mathcal{E}$ into $\mathcal{E}$ if the defender acts adequately. For some games, this method enables one to determine $\mathcal{E}$. One can then refine the analysis by considering an additive cost functional on the intruder trajectory, such as the time to reach the target set, and by assigning an amount to the intruder that depends on the point where he hits the target set $\mathcal{T}$. This refinement regularizes the game by making the cost functional smooth. Starting inside $\mathcal{E}$, the intruder tries to minimize the cost and the defender tries to maximize it. Under suitable assumptions on the dynamics, the resulting upper and lower value of the cost satisfy PDEs know as Isaacs equations. For some games, these equations can be solved, at least numerically (see [50]).

The aim of this chapter is to construct a model predictive feedback control law (see [1]) for the defender that is designed to prevent the intruder reaching the high value targets in the harbor. A model predictive control law is a type of nonlinear sample-data feedback control law in which the control to be applied at each iteration is determined by the solution of a finite or infinite horizon optimal control problem. Since, ideally, the required solution of the optimal control problem needs to be computed in less than 1/10 of the sample time, it is obvious that computing time is a very serious issue in the design of practical model predictive control laws. Since we are dealing with a pursuit-evasion game, the optimal control problem that the defender needs to solve at each sample time turns out to be a generalized max-min problem that does not have an intruder min-max counter part. This type of generalized max-min problem can only be solved using the outer approximation algorithm [20] in conjunction with exact penalties [51]. Since each iteration of the outer approximations algorithm involves the solution of an optimal control problem, and one might need at least 10 iterations to get a reasonable approximation to a solution of the max-min problem, it is clear that computing time is critical in determining the real world implementability of a receding horizon control law.

In our 2011 paper [52] we presented our first results dealing with harbor defense. There we concentrated on getting theoretical bounds on the defensibility of a harbor using a single defender modeled by simplified unicycle dynamics, as well as making an attempt at constructing a model predictive control law for the defender, based on a max-min optimal control model. In the course of the research for this paper, we discovered that the commonly used nonlinear form of unicycle dynamics leads to severe ill-conditioning of the resulting max-min problem, which affected adversely even such reputably ill-conditioning resistant algorithms as SNOPT [53], resulting in unacceptably long computing times as well as occasional failures to compute a result at all. The ill-conditioning may have been aggravated by the use of a nested square root formula [54] for smoothing a min function that is part of the max-min problem formulation.

In the next section, we show that by augmenting the state and control spaces, the unicycle dynamics can be converted to linear dynamics, albeit at the expense of the addition of a large number of convex inequalities. We have also abandoned the GAMS formula [54] for smoothing the min function in the max-min problem and replaced it with a convex hull representation, which also added more design variables, inequalities, and an equality constraint. Nevertheless, the experimental results presented in this paper show that this transformation leads to well conditioned max-min optimal control problems, with a reduction of computing times by a factor of 30. In fact, our computing times are now sufficiently short for using the algorithm presented in this work for controlling water craft moving at speeds in excess of 20MPH in a harbor channel.

One is rather limited by the physical goals of the pursuit-evasion game to using either a deterministic cost function or a probabilistic one. In this work, we use a deterministic one. In either case, the mechanics of the outer approximations algorithm rule out the use of free time optimal control problems.

4.2 Dynamic Models and Model Predictive Control

We consider a harbor that can be reached via a rectangular channel of width W. An intruder tries to reach the harbor and a defender tries to prevent him from doing so, as illustrated in Fig. 4.1.

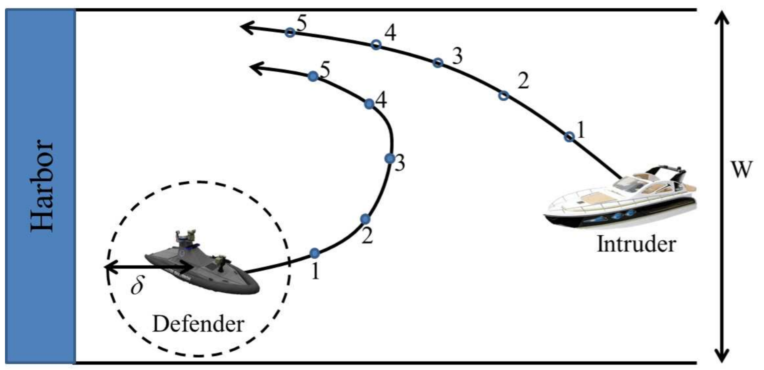


Figure 4.1: The Harbor, Access Channel, Intruder and Defender

4.2.1 Model for the Vehicle Dynamics

For vehicles moving in a plane, as in our case, it is common to assume that their dynamics have are of the three state "unicycle" form (see [55, 56]):

$$\begin{bmatrix} \dot{x}^1(t) \\ \dot{x}^2(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \sigma \cos\theta(t) \\ \sigma \sin\theta(t) \\ u(t) \end{bmatrix}, \tag{4.1}$$

where $x^1(t), x^2(t)$ are the physical coordinates of the vehicle, σ is the constant speed at which it is moving, and $u(t)$ is a control which governs the rate at which the vehicle can change its travel direction. Obviously, at some point, one must consider bounds on the control.

The nice thing about the form of the dynamics in (4.1) is that they use the smallest number of state variables possible and capture simply the fact that the vehicle moves at constant velocity. Unfortunately, they are also nonlinear, and in our earlier numerical experiments have caused severe ill-conditioning in the optimal control problems that one needs to solve within a moving horizon control scheme, resulting in unacceptably long computing times. We therefore propose to replace them with the following four state, two input equivalent *linear* dynamic model:

$$\dot{z}(t) = \bar{A}z(t) + \bar{B}u(t), \tag{4.2}$$

where $z(t) = (x^1(t), x^2(t), v^1(t), v^2(t))^T$, with $x^1(t), x^2(t)$ the vehicle coordinates in the plane, $v^1(t), v^2(t)$ the components of the vehicle velocity in the coordinate directions, and $u(t) = (u^1(t), u^2(t))^T$ the control. The matrices $\bar{A}, \bar{B}$ have the form

$$\bar{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \bar{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.3}$$

Note that the dynamics in (4.2) do not ensure that the vehicle moves at a constant speed σ. Hence we must augment (4.2) with the inequality

$$\|v(t)\| \leq \sigma, \quad \forall t, \tag{4.4a}$$

where $v(t) = (v^1(t), v^2(t))^T$. The reason for using the inequality (4.4) rather than the equality $\|v(t)\| = \sigma, \quad \forall t$ is technical. The equality is incompatible with the Polak-He Phase I - Phase II algorithm (see [20]) that we use for solving the discrete-time optimal control problems in the model predictive control scheme. However, it so happens that the algorithm keeps the inequality tight at a solution, so the constant speed requirement is satisfied.

Finally, if in the original formulation of the dynamics, there was a constraint $|u(t)| \leq \alpha$, then in the new dynamics this constraint becomes

$$\|u(t)\|^2 \leq \alpha^2, \quad \forall t. \tag{4.4b}$$

Next, let $\Delta > 0$ be the sampling time associated with the model predictive control scheme. Then, because the dynamics are linear and time invariant, we obtain the following discrete-time dynamics that are to be used in the model predictive control scheme, under the assumption that for $t \in (k\Delta, (k+1)\Delta, u(t)] = u(k\Delta)$, $k = 1, 2, 3, \ldots$:

$$z((k+1)\Delta) = Az(k\Delta) + Bu(k\Delta), \quad k = 1, 2, 3, \ldots, \tag{4.5}$$

where

$$A = exp(\Delta\bar{A}), \quad B = \int_0^\Delta exp((\Delta - t)\bar{A})dt, \tag{4.6}$$

The inequality (4.4a) now leads to the constraint

$$\|v(k\Delta)\|^2 \leq \sigma^2, \quad \forall k \tag{4.7}$$

Note that (4.7) is a system of convex inequality constraints and that this discretization is exact. Had we used the original form (4.1) of the dynamics and used Euler discretization to obtain the discrete-time system needed for model predictive control, the resulting difference equation would be time varying and not necessarily a good approximation to the actual behavior of the dynamical system.

We must admit that the development of the substitute linear model, which pays the price of a large number of convex inequalities for linearity and time invariance, was based on a hunch rather than analysis. The fact that this hunch was sound is born out by the fact that computing times for the solution of the associated optimal control problems have dropped from hours to seconds.

4.2.2 Model Predictive Control Law Formulation

The aim of the intruder is to destroy a high value target (e.g., a cruise ship with 5000 people on board) moored in a harbor at the end of a rectangular channel. If we assume that, almost certainly, the defender can destroy the intruder when the intruder comes within a range of less than δ units of the defender, it is clear that whatever strategy the intruder adopts for achieving its goal, to be successful, it must maintain a distance of at least δ from the defender during its attack.

We assume that both the defender and intruder motions are governed by the unicycle dynamics (4.5), with appropriate different constraints determining their speeds and accelerations.

Given the uncertainty of the intruder's strategy and possible inaccuracies of determination of its dynamics and state, the defense strategy has to be based on a feedback law. As we have already stated, a sample-data model predictive feedback law seems to be about the only choice available at this time. Hence, the only remaining issue is what kind of optimization problem should be solved to determine the defender control inputs. In view of the discussion above, we propose to adopt a worst case approach and use the following max-min problem for this purpose.

We begin by defining the horizon to be $N\Delta$, where $N > 0$ is an integer and Δ is the sampling time for the discrete-time dynamics (4.5). Next, we assume that the initial states of the defender and intruder $z_d(0), z_i(0)$ are obtained by measurement. Note that because we are dealing with time invariant systems, the initial time can always be taken to be 0, rather than actual time, for the purposes of setting up the model predictive optimization problem.

For the defender, we define the control constraint set by

$$\begin{aligned} \mathbf{U}_d = \{\mathbf{u}_d =& (u_d(0), \ldots, u_d(N\Delta)) \in \mathbb{R}^2 \times \mathbb{R}^N \\ s.t.\ & \|u_d(k\Delta)\|^2 \leq \alpha_d^2, \\ & \|v_d(k\Delta, \mathbf{u}_d)\|^2 \leq \sigma_d^2, \\ & 0 \leq x_d^1(k\Delta, \mathbf{u}_d), \\ & 0 \leq x_d^2(k\Delta, \mathbf{u}_d) \leq W, \quad k = 0, \ldots, N-1\} \end{aligned} \tag{4.8}$$

where $\sigma_d > 0$ is the speed limit for the defender and W is the width of the channel.

For the intruder, the control constraint depends on the choice of a defender input $\mathbf{u}_d$ via the resulting defender trajectory $\mathbf{x}_d = (x_d(0, \mathbf{u_d}), x_d(\Delta, \mathbf{u}_d), \ldots, x_d(N\Delta, \mathbf{u}_d)$, determined by (4.5). Hence,

$$\begin{aligned} \mathbf{U}_i(\mathbf{u}_d) = \{\mathbf{u}_i =& (u_i(0), \ldots, u_i(N\Delta)) \in \mathbb{R}^2 \times \mathbb{R}^N \\ s.t.\ & \|u_i(k\Delta)\|^2 \leq \alpha_i^2, \\ & \|v_i(k\Delta, \mathbf{u}_i)\|^2 \leq \sigma_i^2, \\ & \|x_i(k\Delta, \mathbf{u}_i) - x_d(k\Delta, \mathbf{u}_d)\|^2 \geq \delta^2, \\ & 0 \leq x_i^1(k\Delta, \mathbf{u}_i), \\ & 0 \leq x_i^2(k\Delta, \mathbf{u}_i) \leq W, \quad k = 0, \ldots, N-1\}, \end{aligned} \tag{4.9}$$

where σ_i is the speed limit for the intruder.

Since the intruder may succeed in destroying its target in fewer than N sample intervals, we have to take into account that it may then choose to try to escape. Hence we propose the following max-min optimization problem for the model predictive control law:

$$\max_{\mathbf{u}_d \in \mathbf{U}_d} \min_{\mathbf{u}_i \in \mathbf{U_i(u_d)}, k \in \mathbf{N}} \{x_i^1(k\Delta, \mathbf{u}_i)\}, \tag{4.10}$$

where $\mathbf{N} = \{1, 2, \ldots, N\}$.

Note that (4.10) is a *generalized* max-min problem, because the constraint set of the intruder depends on the strategy $\mathbf{u}_d$ of the defender. Because of this, one cannot formulate a corresponding min-max problem and any consideration of a duality gap is meaningless.

To avoid abusing notation, we will denote the actual states of the defender and intruder by $\bar{z}_d(k\Delta)$ and $\bar{z}_i(k\Delta)$, $k = 0, 1, \ldots$ to distinguish them from the states $z_d(k\Delta)$, $z_i(k\Delta)$, $k = 0, 1, \ldots, N$, which are used in solving problem (4.10). We are finally ready to state the receding horizon control law as an algorithm:

Defender model predictive Control Algorithm

Data: Sampling Time = Δ, horizon = $N\Delta$, initial defender and intruder states $\bar{z}_d(0), \bar{z}_i(0)$, parameters $\alpha_d, \alpha_i, \sigma_d, \sigma_i$, and matrices A, B.

Step 1: Set $k = 0$.

Step 2: Set $z(0) = \bar{z}_d(k\Delta)$ and $z_i(0) = \bar{z}_i(k\Delta)$.

Step 3: Solve (4.10) for an optimal defender control $\mathbf{u}_d^*$.

Step 4: Apply the control $u_d^*(0)$ to the defender for Δ units of time.

Step 5: Measure the states $\bar{z}_d((k+1)\Delta)$ and $\bar{z}_i((k+1)\Delta)$.

Step 6: Replace k by $k+1$ and go to Step 2.

4.3 Method of Outer Approximations

It remains to discuss the details of solution of the generalized, semi-infinite max-min problem (4.10). We begin by observing that there appears to be only one practical method, *the Method of Outer Approximations* see [20], for solving semi-infinite max-min problems of the form

$$\max_{\xi \in \Xi} \min_{\eta \in H} \phi(\xi, \eta), \tag{4.11}$$

where $\Xi \subset \mathbb{R}^n$ and $H \subset \mathbb{R}^m$ are dense sets and the function $\phi(\xi, \eta)$ is continuously differentiable. For a detailed discussion of the Method of Outer Approximations and proof of its convergence, see [20]. There seem to be no known methods for solving generalized max-min problems directly. Hence we resort to a technique first proposed in [51], which consists of replacing the constraint set $\mathbf{U}_i(\mathbf{u}_d)$ by a constraint $\mathbf{U}_i$ that does not depend on a defender input, and dealing with the contribution of the defender input using exact penalty functions. This transforms problem (4.10) into the problem

$$\max_{\mathbf{u}_d \in \mathbf{U}_d} \min_{\mathbf{u}_i \in \mathbf{U}_i, k \in \mathbf{N}} \{x_i^1(k\Delta, \mathbf{u}_i) + \pi \max_{k \in \mathbf{N}} (\delta^2 - \|x_i(k\Delta, \mathbf{u}_i) - x_d(k\Delta, \mathbf{u}_d)\|^2)_+\} \tag{4.12}$$

where $\pi > 0$ is an exact penalty parameter, $a_+ := \max\{0, a\}$, and

$$\begin{aligned}\mathbf{U}_i = \{\mathbf{u}_i =&(u_i(0), \dots, u_i(N\Delta)) \in \mathbb{R}^N \times \mathbb{R}^N \\ & s.t.\ \|u_i(k\Delta)\|^2 \le \alpha_i^2, \\ & \|v_i(k\Delta, \mathbf{u}_i)\|^2 \le \sigma_i^2, \\ & 0 \le x_i^1(k\Delta, \mathbf{u}_i), \\ & 0 \le x_i^2(k\Delta, \mathbf{u}_i) \le W, \quad k = 0, \dots, N-1\}.\end{aligned} \tag{4.13}$$

We observe that (4.12) is a standard max-min problem, except for the fact that the term $\min_{k\in\mathbf{N}} x_i^1(k\Delta, \mathbf{u}_i)$ is not smooth. To deal with this issue we make use of the fact that

$$\min_{k\in\mathbf{N}} x_i^1(k\Delta, \mathbf{u}_i) = \min_{\mu\in\mathbf{\Sigma}} \sum_{\mu\in\mathbf{\Sigma}} \mu^k x_i^1(k\Delta, \mathbf{u}_i), \tag{4.14}$$

where $\mathbf{\Sigma}$ is the unit simplex in $\mathbb{R}^N$, i.e.,

$$\mathbf{\Sigma} := \{\mu \in \mathbb{R}^N \mid \sum_{k=1}^{N} \mu^k = 1, \quad \mu^k \ge 0,\ \forall k \in \mathbf{N}\}. \tag{4.15}$$

Thus, by introducing an additional design variable μ and a set of linear inequalities and one linear equation, we are finally able to transform (4.10) into the tractable form

$$\begin{aligned}\max_{\mathbf{u}_d\in\mathbf{U}_d}\ \min_{\mathbf{u}_i\in\mathbf{U}_i,\ \mu\in\mathbf{\Sigma}} \{ \sum_{k=1}^{N} \mu^k x_i^1(k\Delta, \mathbf{u}_i) \\ +\pi \max_{k\in\mathbf{N}} (\delta^2 - \|x_i(k\Delta, \mathbf{u}_i) - x_d(k\Delta, \mathbf{u}_d)\|^2)_+\}\end{aligned} \tag{4.16}$$

At this point, we can define a (dual) min-max problem corresponding to (4.16). However, because the cost function is not convex-concave, there is most likely a duality gap. The significance of this fact is hard to interpret.

In terms of the abstract semi-infinite max-min problem form (4.11), the Method of Outer Approximations consists of the successive minimization of the *finite* max functions $\psi_{H_i} : \mathbb{R}^n \to \mathbb{R}$, defined by

$$\psi_{H_i}(\xi) = \min_{\eta\in H_i} \phi(\xi, \eta), \tag{4.17}$$

with the sets $H_i \subset H$ of finite cardinality, which result in progressively better and better local approximations to the function $\psi_H(\cdot)$ near an optimizer of $\psi_{H_i}(\cdot)$. The cardinality of the sets $H_i \subset H$ grows monotonically, with $H_i \subset H_{i+1}$. Finite cardinality max-min problems can be solved directly by means of algorithms described in [20], or, by the addition of a slack variable, transcribed into standard nonlinear programming problems that can be solved by an array of algorithms.

To shorten expressions, we define

$$\hat{H}(\xi) := \arg\min_{\eta \in H} \phi(\xi, \eta). \tag{4.18}$$

Method of Outer Approximations (Conceptual Form)

Data. $\xi_0 \in \mathbb{R}^n$, $H^* = \{\eta_{01}, \ldots, \eta_{0k}\} \subset H$.

Step 0. Set $k = 0$, compute a $\eta_0 \in \hat{H}(\xi_0)$, set $H_0 = \{\eta_0\} \cup H^*$.

Step 1. Compute

$$\xi_{k+1} \in \arg\max_{\xi \in \Xi} \ \psi_{H_k}(\xi) \tag{4.19}$$

Step 2. Compute a $\eta_{k+1} \in \hat{H}(\xi_{k+1})$, and set

$$H_{k+1} = H_k \ \cup \{\eta_{k+1}\}. \tag{4.20}$$

Step 3. Replace i by $k+1$, and go to Step 1.

Note that the algorithm above requires that the minimizer η_{i+1}, in Step 2, be computed exactly, which is usually impossible in practice. Because of that we refer to this algorithm as a *conceptual* algorithm. In practice, we use sequentially better and better approximations to such minimizers (see [20]).

Hence, in terms of our pursuit-evasion problem, the sequence of computations at $k - th$ iteration of the Method of Outer Approximations are as follows: the defender solves the *finite*, discrete-time max-min optimal control problem

$$\begin{aligned} \max_{\mathbf{u}_d \in \mathbf{U}_d} \min_{\mathbf{u}_i \in \mathbf{U}_{i,k}, \mu \in \mathbf{\Sigma}} &\{\sum_{k=1}^{N} \mu^k x_i^1(k\Delta, \mathbf{u}_i) \\ &+ \pi \max_{k \in \mathbf{N}} \delta^2 - \|x_i(k\Delta, \mathbf{u}_i) - x_d(k\Delta, \mathbf{u}_d)\|^2\}, \end{aligned} \tag{4.21}$$

where $\mathbf{U}_{i,k}$ is the set of intruder controls accumulated up to this point. We denote the approximate solution to this problem by $\mathbf{u}_{d,k+1}$. Note the omission of $(\ldots)_+$ in (4.21). When there is a $\mathbf{u}_{d,k+1}$ such that the intruder inequality constraints are violated, the operation $)_+$ is redundant. When the term $(\ldots)_+ = 0$ for all admissible $\mathbf{u}_d$, then any $\mathbf{u}_{d,k+1}$ can be used, since it does not change the value of the value of the cost, i.e., it is a stationary control.

Then the intruder solves the discrete-time optimal control problem

$$\begin{aligned} \min_{\mathbf{u}_i \in \mathbf{U}_i, \mu \in \mathbf{\Sigma}} &\{\sum_{k=1}^{N} \mu^k x_i^1(k\Delta, \mathbf{u}_i) \\ &+ \pi \max_{k \in \mathbf{N}} (\delta^2 - \|x_i(k\Delta, \mathbf{u}_i) - x_d(k\Delta, \mathbf{u}_{d,k+1})\|^2)_+\}, \end{aligned} \tag{4.22}$$

to obtain a solution $\mathbf{u}_{i,k+1}$. Then this solution is added to the set $\mathbf{U}_{i,k}$ to form the new set $\mathbf{U}_{i,k+1}$.

The property of exact penalty functions ensures that provided $\pi \geq \pi^*$, a specific minimum value, the defender solution does not depend on π. However, if for the given initial positions of the defender and intruder, there is no admissible defender control that makes the penalty term positive, then the cost does not depend on the defender control, i.e., we are at a stationary point. To avoid the need for inventing a good "tunneling" heuristic that would get us out of this situation, we use instead an interior penalty function when solving the defender problem. Our experimental results show that this is a better approach than the "tunneling" heuristics of minimizing the maximum distance to the accumulated intruder trajectories.

Next, as far as the intruder is concerned, the use of exact penalty functions is just one way of dealing with some of its constraints. The intruder problem is actually more easily solved by leaving the constraints in their original form. So, again, we do not need a specific value for the penalty π. It is also clear on an intuitive level that the exact penalty function can be replaced with an interior penalty function with little deterioration in the quality of solution.

For all of our computations we have used the Polak-He Minimax Algorithm 2.6.1 in [20] p. 260. This is a very robust first-order algorithm which quickly obtains a reasonable approximation to a solution. Our choice of algorithm is definitely counter intuitive, since there are many potentially superlinearly convergent algorithms that one could have used. However, in our experience, we had frequent failures on our problems with the excellent algorithms in the TOMLAB library [53] as well as will some other open source algorithms.

4.4 Experimental Results

We will now present several of the experiments that we have conducted. The static figures below do not easily convey the evolution in time of the defender and intruder trajectories. We have therefore deposited animations of these trajectories in http://publish.illinois.edu/lee822/. In the labels of these files, w = channel width, s = $\sigma_r = \sigma_d/\sigma_i$ (defender/intruder speed ratio), and x = initial "horizontal" separation distance.

4.4.1 Head-to-Head Starting Position Experiment

The goal of this experiment is to obtain the maximum channel width that a single defender can defend. We assume that initially the defender is located on the center line of the channel, as illustrated in Fig. 4.2, where X is the initial distance between intruder and defender. Again, we denote the speed ratio between defender and intruder as $\sigma_r = \frac{\sigma_d}{\sigma_i}$.

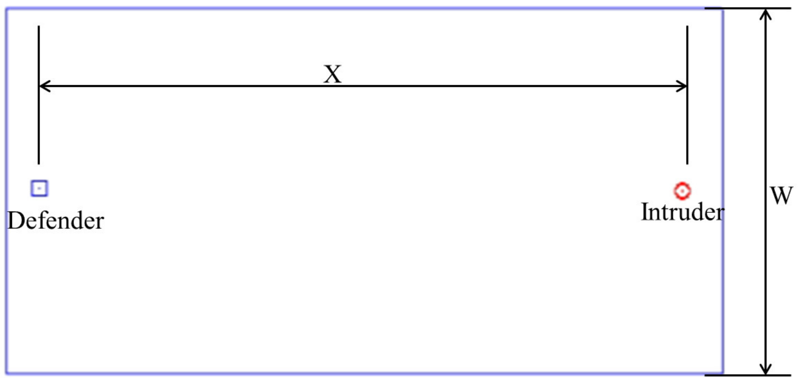


Figure 4.2: Experiment setup: Head-to-Head Initial Positions

The outcome of each experiment is either the defender wards off the intruder (defender wins), or the the intruder reaches to the harbor (intruder wins). In Table I and the subsequent tables, 'D' indicates defender wins, 'I' indicates intruder wins. An entry of the form wαsβxγ means that the channel width $W = \alpha$, $\sigma_r = \beta$, and $X = \gamma$. For example, 'w9s1x10' indicates the experiment with channel width 9, relative speed limit 1 (equal speeds), and initial separation distance 10. In all of our experiments, the initial distance from from the defender to the harbor is six units.

To obtain the results in Table I, by varying the initial "vertical" (x_i^2) position of the intruder. The worst case for the defender was when the intruder was located at the channel boundary. The results in Table I show that a large X favors the defender. Not surprisingly, for a fixed X, the defender can cover a larger area as its speed ratio is increased. Also, even when speed of the defender is larger than that of the intruder, the defender may fail. However, when the speed of the defender is about 40% larger than that of the intruder, and the initial distance from the

Table 4.1: Head-to-Head Initial Positions Experiment Results

X=10				
σ_r	W	Winner	Maximum W	Reference
1	$\leq$9	D		w9s1x10
1	10	I	9	w10s1x10
1.2	$\leq$16	D		w16s12x10
1.2	17	I	16	w17s12x10
1.4		D		w27s14x10

X=13				
σ_r	W	Winner	Maximum W	Reference
1	$\leq$14	D		w14s1x13
1	15	I	14	w15s1x13
1.2	$\leq$23	D		w23s12x13
1.2	24	I	23	w24s12x13
1.4		D		w26s14x13

X=16				
σ_r	W	Winner	Maximum W	Reference
1	$\leq$25	D		w25s1x16
1	26	I	25	w26s1x16
1.2	$\leq$27	D		w27s12x16
1.2	28	I	27	w28s12x16
1.4		D		w29s14x16

defender to the harbor is 6 units, the defender succeeds regardless of the channel width, for any for any $X > 0$.

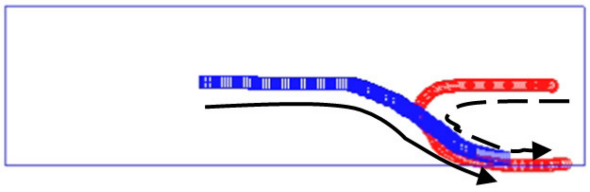


Figure 4.3: Defender Succeeds (w9s1x10).

Fig. 4.3 shows a typical result when the defender wins in a head-to-head position experiment. The solid arrow curve indicates the direction of the defender's path and the dotted arrow curve indicates the direction of the intruder's path. The defender starts heading towards the intruder (away from the harbor) and the intruder starts heading directly towards to the harbor. After an initial attempt at direct penetration, the intruder turns towards the channel wall to improve its chances, but is nevertheless thwarted by the defender and it flees.

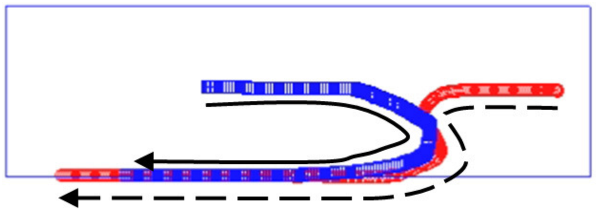
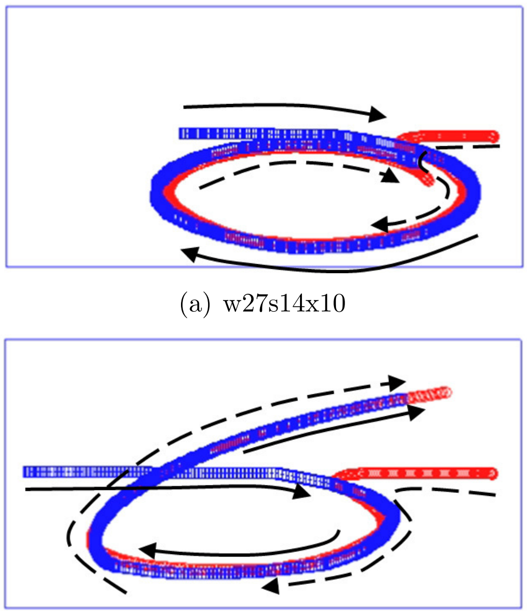


Figure 4.4: Intruder Succeeds (w10s1x10).

When the channel is a little wider (W=10), the intruder manages to penetrate, as we see in Fig. 4.4.



(a) w27s14x10

(b) w29s14x16

Figure 4.5: Defender Succeeds.

Fig. 4.5 shows a case when the speed of the defender is 40% larger than intruder's, with the defender starting 6 units from the harbor.

4.4.2 Cross Position Experiment

In this experiment, the intruder approaches the harbor by following the channel wall. The defender is in the middle of channel as illustrated in Fig. 4.6.

Comparing Table 7.1 with Table 4.2, we see that it is advantageous for the defender to have a narrow channel and that the most advantageous position for

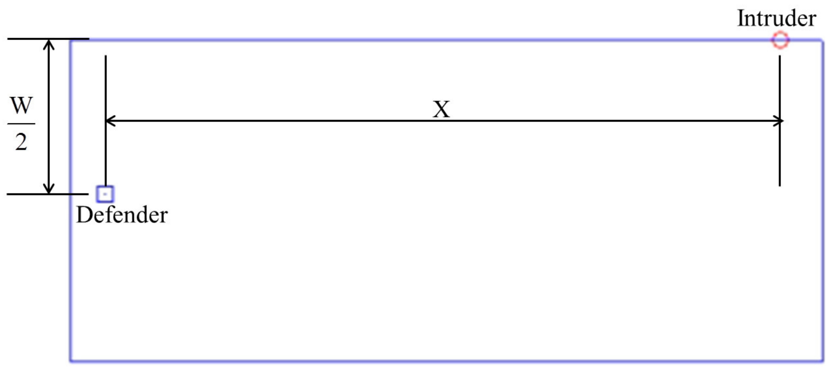


Figure 4.6: Experiment Setup: Cross Position

Table 4.2: Cross Position Experiment Results

X=13				
σ_r	W	Winner	Maximum W	Reference
0.4	≤7	D		w7s04x13
0.4	8	I	7	w8s04x13
0.6	≤9	D		w9s06x13
0.6	10	I	9	w10s06x13
0.8	≤10	D		w10s08x13
0.8	11	I	10	w11s08x13
X=16				
σ_r	W	Winner	Maximum W	Reference
0.8	≤14	D		w15s08x16
0.8	15	I	14	w15s08x16
1	≤15	D		w15s1x16
1	16	I	15	w15s1x16
1.4	≤ 18	D		w18s14x16
1.4	19	I	18	w19s14x13

the intruder to enter the channel is along a wall. Our experiments can be used in deciding whether a single defender is sufficient to deter a single intruder in a particular channel.

Fig. 4.7 illustrates the case when the channel is narrow (W=7). The defender successfully wards off the intruder who is approaching the harbor by following the channel wall, forcing the intruder to retreat to avoid being destroyed.

However, as we see in Fig. 4.8, when the channel is little wider (W=10), the defender fails to stop the intruder, which moves straight to the harbor.

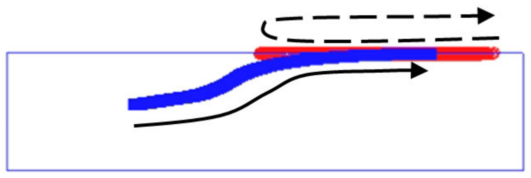


Figure 4.7: Defender Succeeds (w7s04x13).

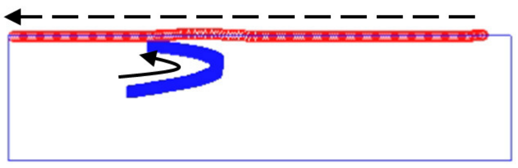
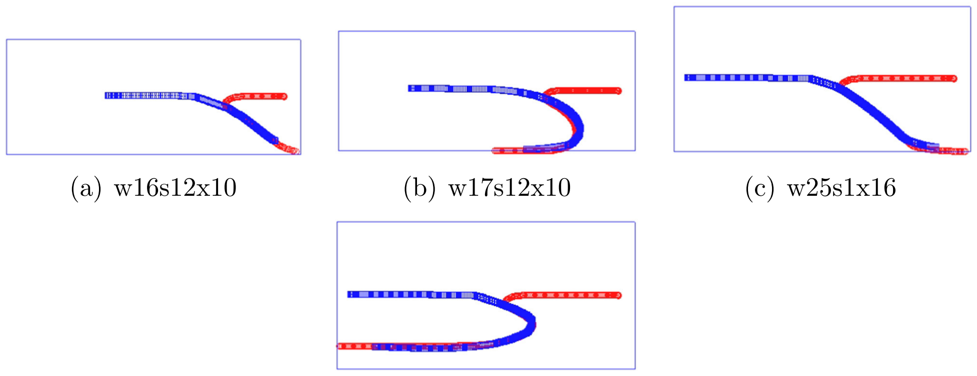


Figure 4.8: Intruder Succeeds (w8s04x13).

4.4.3 Additional Results

In this subsection, some additional scenarios are presented.



(a) w16s12x10

(b) w17s12x10

(c) w25s1x16

(d) w26s1x16

Figure 4.9: Various Scenarios of Head-to-head Experiment.

Throughout the experiment, one can determine the upper bound of channel with known defender and intruder's initial position and maximum speed limit.

4.4.4 Two Uncoordinated Defenders

In principle, it would be best to use two defenders whose actions are coordinated. This would require the use of offshore command center. Here we explore the effectiveness of two uncoordinated defenders, each governed by the same model predictive control law, in stopping an intruder when the channel is too wide for a single defender to defend successfully.

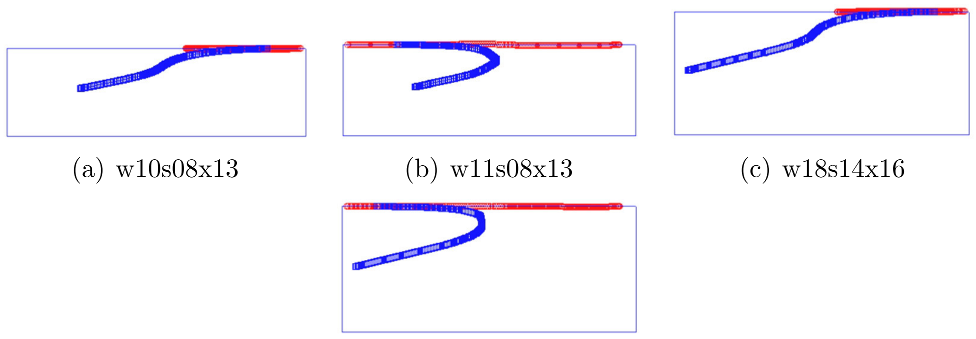

(a) w10s08x13 (b) w11s08x13 (c) w18s14x16 (d) w19s14x16

Figure 4.10: Various Scenarios of Cross Position Experiment.

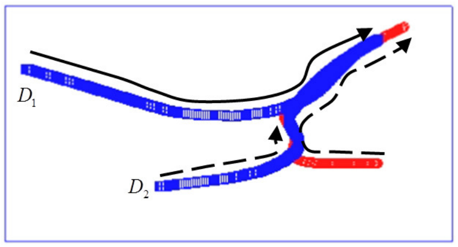


Figure 4.11: Two Uncoordinated Defenders Succeed (w30s1x13).

In Fig. 7.5, as the intruder approaches, the closer defender(D_2) tries to block the intruder. However, since the channel is wide (W=30), D_2 cannot defend successfully, as seen from from Fig. 4.4. However, with the help of defender (D_1), the two defenders successfully deter the intruder.

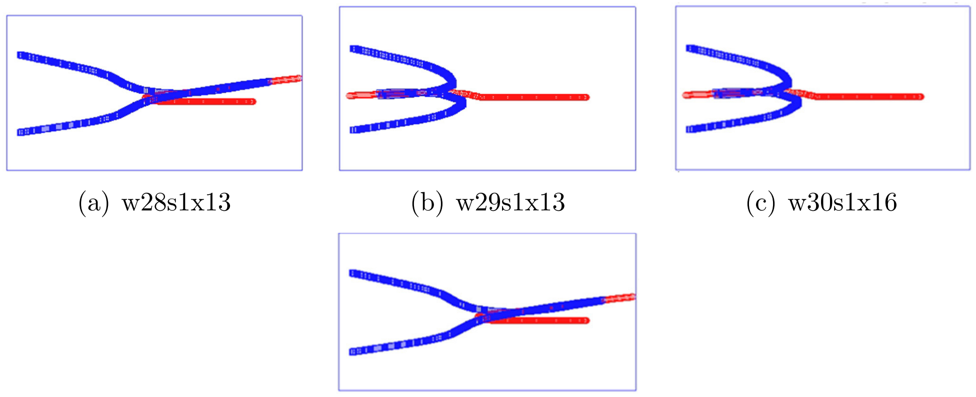

(a) w28s1x13 (b) w29s1x13 (c) w30s1x16 (d) w31s1x16

Figure 4.12: Various Scenarios of Two Uncoordinated Defenders Experiment.

Fig. 4.12 shows various scenarios of two uncoordinated defenders experiment result. Coverage of the two defenders is twice larger than the single defender's. Results are organized in Table 4.3.

Table 4.3: Two uncoordinated Defender Experiment Results

X=13				
σ_r	W	Winner	Maximum W	Reference
1	$\leq$28	D		w28s1x13
1	29	I	28	w29s1x13
X=16				
σ_r	W	Winner	Maximum W	Reference
1	$\leq$30	D		w30s1x16
1	31	I	30	w31s1x16

4.4.5 Computing time

The impact on the computing time resulting from the reformulation of the model dynamics and smoothing techniques for dealing with the min function in (4.10), is shown in Table 4.4. It compares computing times, for $\Delta = 0.1$, for solving the max-min optimal control problem (4.10) using "classical" unicycle dynamics and smoothing of the min function using the GAMS method [54], as in [52], with the approach taken in this work. In this table, 'OAA Iters' denotes the number of outer approximation iterations used to compute the control by the outer approximations algorithm.

Table 4.4: Computing Time Comparison

	2011	2013
Dynamics model	Bicycle: Nonlinear	Bicycle: Linear
model predictive	16Δ	16Δ
OAA Iters	10	10
Solver	SNOPT	PH
Comp. Time/OAA Iter	33.8 sec	1.0 sec
Comp. Time/sample	338 sec	10 sec

In this table, PH is the Polak-He min-max Algorithm 2.6.1 in [20]. In [52], SNOPT is a fast algorithm in the TOMLAB [53] optimization library. The various trajectories in the figures in this paper, are 120 samples long, and took 20 minutes to generate. The trajectories in [52] are only 32 samples long and took about 3 hours to obtain.

When implemented in C++ and refined through the use of adaptive approximations, as in [57], we can expect computing times to drop by a factor of at least 10, which means we can use a sample time Δ of 1 second. This is compatible with the control of a water craft moving at 20 mph in a channel.

4.5 Conclusions

We have presented model predictive control law that can be used by one or multiple uncoordinated defenders to ward off an intruder that is trying to attack a target in a harbor.

CHAPTER 5

VARIATIONS OF THE HARBOR DEFENSE PROBLEM

5.1 Introduction

In this chapter, we consider variations of the original harbor defense problem that is presented in the previous chapter. Variations include not only the multiple number of defenders and intruders, but also various intruder's strategies. Particularly, improved discretized dynamic model (5.5) of the intruder and the defender are used. New model is an exact integration of the continuous model. Detailed discussion is presented in Appendix A.

Similar to the previous chapter, we assume that the defending vehicles are manned or unmanned submarines, manned or unmanned hovercraft, or drones. The purpose of this paper is to construct a feedback model predictive control (MPC) law (see [1]) for the defenders, based on max-min optimal control problems which, we believe, capture the essence of the intruders' goal of at least one of them getting within striking distance of their target, as well as the intruders' perception of how they can be destroyed by the defenders.

The idea behind MPC is quite old, going back to the 1950's, and is based on the following observation. Suppose that we have a dynamical system that is modeled by a differential equation of the form

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_0, \ t \geq 0 \tag{5.1}$$

where $x(t)$ is the state of the system and $u(t)$ is the control. Now suppose that one wants to optimize its behavior by solving an optimal control problem of the form

$$\min_{u(t) \in U, x(t) \in X} \int_0^\infty f^0(x(t), u(t)) dt, \tag{5.2}$$

subject to the differential equation constraint (5.1). Let $\hat{u}(t)$ be solution of this optimal control problem and $\hat{x}(t) = x(t, \hat{u}(t))$, $t \geq 0$ the resulting trajectory. Now suppose that the model (5.1) is not perfect, so the actual trajectory resulting from the control $\hat{u}(t)$, $x^*(t, \hat{u}(t))$, is quite different from $\hat{x}(t)$. To remedy this situation, it was proposed that every Δ time units, the actual state be measured and the problem (5.1) re-solved, to obtain a corrected control, effectively creating a feedback mechanism. Experiments have shown that this is an excellent idea. For an excellent survey of MPC, see [1]. For previous attempts of using MPC in pursuit-evasion situations, see [52, 55, 58].

Our situation is more complicated than the one above. We have multiple defender dynamic systems and multiple intruder dynamic systems and the optimal control problem (5.2) must be replaced by a problem that reflect this fact as well as the fact that the defenders do not know the intruders strategies. Hence we propose a worst case approach and, since continuous time optimal control problems require a great deal of time for their solution, we will assume that the controls are constant during the sample times, which results in the replacement of dynamics described by differential equations by derived dynamics described by difference equations.

5.2 Model Predictive Control Formulation

5.2.1 Assumptions

First, it does not seem to be possible to solve the type of problem we propose over an infinite horizon, as in (5.2). Hence we introduce a *finite* horizon $N\Delta > 0$, with N a positive integer. The *sample time* $\Delta > 0$ has to be chosen taking into account the speed with which the craft are moving and the time it takes to solve the MPC determining min-max optimal control problem. For example, if, as in our case, with a 10 sample time horizon, it takes 0.2 seconds to solve the problem, the sample time could be 0.5-1.0 seconds, which is equivalent to adjusting the control every 0.01-0.02 miles for a torpedo travelling at 80 MPH. Since the longer the horizon the longer the computing time, the length of the horizon is largely determined by the computing power available for the defenders.

Second, we assume that the intruders cannot risk engaging the defenders in battle and we consider three possible scenarios. In the first, which is deterministic, the intruders assume that they are safe as long as they avoid coming within striking distance of the defenders [58]. In the next two scenarios, which are probabilistic as well as more realistic, the intruders assume that the probability of their destruction is a function of their distances from the defenders. In the first probabilistic formulation, the intruders attempt to survive over the entire horizon. In the second one, first the intruders attempt to destroy their target within the horizon time, and only if successful do they attempt to survive to the end of the horizon.

Third, we assume that the defenders are able to determine the dynamics of the intruders and that the floor of the harbor is seeded with sensors that enable the defending team to determine continuously the position, velocity, and direction of travel of the intruder. Just to be safe, we also assume that the intruders have access to similar information about the defenders.

Fourth, we assume that the actions of the defenders are coordinated and that the actions of the intruders are also coordinated. Coordination of the defenders (intruders) can be achieved either by using an offshore or mother ship computer to solve the MPC optimal control problem and then transmitting the required controls to each individual craft, or by each craft solving the same MPC optimal control problem and using the appropriate resulting control.

5.2.2 Dynamics

Assuming that both the defenders and intruders are unmanned underwater vehicles (UUVs) and that they are confined to a rectangular channel of width W at the end of which is the harbor with the high value target, their dynamics have the form

$$\begin{aligned}\dot{x}^1(t) &= v(t)\cos\theta(t)\\ \dot{x}^2(t) &= v(t)\sin\theta(t)\\ \dot{\theta}(t) &= \sigma(t)\\ \dot{v}(t) &= \alpha(t),\end{aligned} \tag{5.3}$$

where x^1 is the positional coordinate of the UUV along the channel, x^2 is the positional coordinate of the UUV perpendicular to the channel, and θ is the heading, i.e., the angle between the direction of motion of the UUV and the x^1 axis in our positional coordinate system. We assume that the channel is sufficiently shallow that a depth coordinate is not needed. We assume that there is a steering input $\sigma(t)$ and a propulsion input $\alpha(t)$, which are subject to constraints of the form:

$$\begin{aligned}&0 \le v(t) \le \bar{v}\\ &|\alpha(t)| \le \bar{\alpha}\\ &|\sigma(t)| \le \bar{\sigma}\\ &\sigma(t)v(t) \le k_f.\end{aligned} \tag{5.4}$$

The constraint $\sigma(t)v(t) \leq k_f$ captures the relationship between centripetal force and velocity.

We assume that the model predictive control law uses a sample time Δ, so that for any integer $k \geq 0$ and $t \in [k\Delta, (k+1)\Delta)$ the controls are constant, i.e., for $t \in [k\Delta, (k+1)\Delta)$, $v(t) = v(k\Delta)$ and $\sigma(t) = \sigma(k\Delta)$. We can integrate the differential equation (5.2) for $t \in [k\Delta, (k+1)\Delta)$, to obtain the difference equations:

$$\begin{aligned}
x^1_{k+1} &= x^1_k + \Delta v_k \cos\theta_k + \Delta^2 \alpha_k \cos\theta_k \\
x^2_{k+1} &= x^2_k + \Delta v_k \sin\theta_k \Delta^2 \alpha_k \sin\theta_k \\
\theta_{k+1} &= \theta_k + \Delta\sigma_k \\
v_{k+1} &= v_k + \Delta\alpha_k,
\end{aligned} \tag{5.5}$$

where $x^1_k = x^1(k\Delta)$, $x^2_k = x^2(k\Delta)$, $\theta_k = \theta(k\Delta)$, $v_k = v(k\Delta)$, $\sigma_k = \sigma(k\Delta)$, and $\alpha_k = \alpha(k\Delta)$. The derivation of (5.5) is presented in Appendix A.

5.2.3 Defender model predictive control law

Let $\bar{z}_{d,1}(k\Delta) = (\bar{z}_{d,1}(k\Delta), \ldots, \bar{z}_{d,N_d}(k\Delta))$ and $\bar{z}_{i,1}(k\Delta) = (\bar{z}_{i,1}(k\Delta), \ldots, \bar{z}_{d,i_{N_i}}(k\Delta))$, where $\bar{z}_{i,1}(k\Delta) = (x^1_{i,1}(k\Delta), x^2_{i,1}(k\Delta), \theta_{i,1}(k\Delta), v_{i,1}(k\Delta))$, and so forth.

Defender Model Predictive Control law Algorithm

Data: Sampling Time = Δ, computing time $\delta < \Delta$, horizon = $N\Delta$, initial defender and intruder states $\bar{z}_d(0), \bar{z}_i(0)$,

Step 1: Set $k = 0$.

Step 2: Set $z(0) = \bar{z}_d(k\Delta)$ and $z_i(0) = \bar{z}_i(k\Delta)$.

Step 3: Solve one of the defender min-max problems, below, for an optimal coordinated defender control $\mathbf{u}^*_d$.

Step 4: Apply the control $u^*_d(0)$ to the defender for Δ units of time.

Step 5: At time $\delta + k\Delta$, measure the states $\bar{z}_d(\delta + k\Delta)$ and $\bar{z}_i(\delta + k\Delta)$.

Step 6: Estimate the states $\bar{z}_d((1+k)\Delta)$ and $\bar{z}_i((1+k)\Delta)$ using the differential equation (5.3).

Step 7: Replace k by $k+1$ and go to **Step 2**.

Note that the min-max problem in Step 4, above, can be changed at each sampling time, and so can the sample time Δ. It makes sense to use a large Δ when the adversaries are far apart and decrease it as they get nearer to each other.

5.3 Min-max problem formulations

The next element that we must introduce is the min-max optimal control problem, reflecting a worst case scenario, that must be solved at each sample time within the MPC law. We will consider three possible scenarios: (a) where the intruders are risk averse, (b) where the intruders are willing to take risks, and (c) where the intruders are willing to sacrifice themselves to achieve their goal.

To distinguish between the intruder and defender, we will add a subscript i, j to indicate the j-th intruder states, controls, and constraints, a subscript d, k to indicate k-the defender states, controls, and and constraints.

Suppose that there are N_d defenders and N_i intruders and that the horizon length is $N\Delta$. For $k = 0, \ldots, N-1$, let $u_{d,j}(k\Delta) = (\sigma_{d,j}(k\Delta), \sigma_{d,j}(k\Delta))^T$ and $u_{i,l}(k\Delta) = (\sigma_{i,l}(k\Delta), \sigma_{i,l}(k\Delta))^T$, with $j = 1, \ldots, N_d$ and $l = 1, \ldots, N_i$.

5.3.1 Risk averse intruders

Assuming that the intruders are risk averse and hence will not venture within torpedo striking distance $\tau > 0$ of the defenders, we propose the following max-min optimal control problem for the receding horizon control law:

$$\max_{\mathbf{u}_d \in \mathbf{U}_d} \quad \min_{\mathbf{u}_i \in \mathbf{U}_i(\mathbf{u}_d), k \in \mathbf{N}} \quad \min_{j \in \mathbf{N}_i} \{x^1_{i,j}(k\Delta, \mathbf{u}_i)\}, \tag{5.6}$$

where $\mathbf{N} = \{1, 2, \ldots, N\}$ is the length of the receding horizon control horizon and $\mathbf{N}_i = \{1, 2, \ldots, N_i\}$. The constraint set $\mathbf{U_d} = \mathbf{U}_{d,1} \times \ldots \times \mathbf{U}_{d,N_d}$, for the coordinated defenders, is defined by

$$\begin{aligned}
\mathbf{U}_{d,j} = \{\mathbf{u}_{d,j} =& (u_{d,j}(0), \ldots, u_{d,j}((N-1)\Delta)) \\
& s.t.\ 0 \le v_{d,j}(k\Delta) \le \bar{v}_{d,j}, \\
& \quad |\sigma_{d,j}(k\Delta)| \le \bar{\sigma}_{d,j}, \\
& \quad |\alpha_{d,j}(k\Delta)| \le \bar{\alpha}_{d,j}, \\
& \quad \sigma_{d,j}(k\Delta) v_{d,j}(k\Delta) \le \kappa_{d,j}, \\
& \quad 0 \le x^1_{d,j}(k\Delta, \mathbf{u}_{d,j}), \\
& \quad 0 \le x^2_{d,j}(k\Delta, \mathbf{u}_{d,j}) \le W\},
\end{aligned} \tag{5.7}$$

where $k = \{0, \ldots, N-1\}$, $\bar{v}_{d,j} > 0$, $j = 1, \ldots, N_d$, are the speed limits for the defenders, $\bar{\sigma}_{d,j}$, $j = 1, \ldots, N_d$, are the limits on the steering inputs for the defenders, and W is the width of the channel.

For the intruder, the control constraint depends on the choice of a defender input

$\mathbf{u}_d$ via the resulting defender trajectory $\mathbf{x}_d = (x_d(0, \mathbf{u_d}), x_d(\Delta, \mathbf{u}_d), \ldots, x_d(N\Delta, \mathbf{u}_d))$, determined by (5.5). Hence,

$$
\begin{aligned}
\mathbf{U}_i(\mathbf{u}_d) = \{&\mathbf{u}_i = (u_i(0), \ldots, u_i((N-1)\Delta)) \\
s.t.\ & 0 \leq v_{i,j}(k\Delta) \leq \bar{v}_{i,j}, \\
& |\sigma_{i,j}(k\Delta)| \leq \bar{\sigma}_{i,j}, \\
& \|x_{i,j}(k\Delta, \mathbf{u}_i) - x_{d,l}(k\Delta, \mathbf{u}_{d,l})\|^2 \geq \tau^2, \\
& 0 \leq x^1_{i,j}(k\Delta, \mathbf{u}_{i,j}), \\
& 0 \leq x^2_{i,j}(k\Delta, \mathbf{u}_{i,j}) \leq W, k = 0, \ldots, N-1, \\
& \sigma_{i,j}(k\Delta) v_{i,j}(k\Delta) \leq k_{i,j}\},
\end{aligned}
\tag{5.8}
$$

where $j = 1, \ldots, N_i$, $l = 1, \ldots, N_d$, and τ is a torpedo distance.

Note that the defenders' actions do not affect the cost function. Defense is achieved by interference as expressed by the constraints imposed on the intruder.

There are three issues that must be dealt with in solving problem (5.6). The first two are obvious, the last one is subtle.

First, (5.6) is a type of *generalized* max-min problem [51], because the constraint set of the intruders depend on the strategy $\mathbf{u}_d$ of the defenders and hence cannot be solved by standard max-min algorithms, such as outer approximations. In fact, it is a type of bilevel problem that can be converted to a "standard" max-min problem by adding the defender dependent constraints to the cost function using exact penalty functions, as was done in [51]. The exact penalty functions need not be used when evaluating the min part of the max-min problem for a given set of defender controls, but they must be used in the maximization process. The introduction of exact penalty functions transforms problem (5.6) into

$$
\begin{aligned}
&\max_{\mathbf{u}_d \in \mathbf{U}_d} \min_{\mathbf{u}_i \in \mathbf{U'}_d, k \in \mathbf{N}} \min_{j \in \mathbf{N}_i} \{x^1_{i,j}(k\Delta, \mathbf{u}_i) + \pi \max\{ \\
&0, -\|x_{i,j}(k\Delta, \mathbf{u}_i) - x_{d,l}(k\Delta, \mathbf{u}_{d,l})\|^2 + \tau^2\}\},
\end{aligned}
\tag{5.9}
$$

where $\pi > 0$ is the value of the exact penalty function and

$$\begin{aligned}\mathbf{U'}_i =& \{\mathbf{u}_i = (u_i(0), \ldots, u_i((N-1)\Delta)) \\ & s.t.\ 0 \leq v_{i,j}(k\Delta) \leq \bar{v}_{i,j}, \\ & \quad |\sigma_{i,j}(k\Delta)| \leq \bar{\sigma}_{i,j}, \\ & \quad 0 \leq x^1_{i,j}(k\Delta, \mathbf{u}_{i,j}), \\ & \quad 0 \leq x^2_{i,j}(k\Delta, \mathbf{u}_{i,j}) \leq W, k = 0, \ldots, N-1 \\ & \quad \sigma_{i,j}(k\Delta) v_{i,j}(k\Delta) \leq k_{i,j}\}.\end{aligned} \tag{5.10}$$

Second, the cost function

$$\min_{j \in \mathbf{N}_i} \{x^1_{i,j}(k\Delta, \mathbf{u}_i)\} \tag{5.11}$$

is not differentiable. This can be dealt with by smoothing (see [54]), as was done in [52], and found to cause serious ill-conditioning, or, as we do now, by making use of the fact that the minimum over a set is equal to the minimum over its convex hull. This requires the addition of decision variables $\mu_{j,k} \geq 0$, $j = 1, \ldots, N_i$, $k = 0, \ldots, N-1$ such that

$$\sum_{j \in \mathbf{N}_i, k \in \mathbf{N}_N} \mu_{j,k} = 1, \tag{5.12}$$

i.e., we add $N_i \times N$ variables, with positivity constraints and one equality constraint, which can be eliminated explicitly. The cost function now becomes

$$\min_{j \in \mathbf{N}_i} \{ \sum_{i \in \mathbf{N}, k-1 \in \mathbf{N}} \mu_{j,k-1} x^1_{i,j}(k\Delta, \mathbf{u}_i)\}. \tag{5.13}$$

The third issue stems from the fact that when the separation between defenders and intruders is sufficiently large, the solution of the min part does not require that the constraints be active. Hence, at such situations, the value of the min function is independent of the value of the defender controls, and hence is a stationary point. At such points, solving the min-max problem (5.9) does not produce a meaningful result from the defenders' point of view. Hence, we can replace the problem (5.9) with the problem

$$\begin{aligned}&\max_{\mathbf{u}_d \in \mathbf{U}_d} \min_{\mathbf{u}_i \in \mathbf{U'}_d), k \in \mathbf{N}} \min_{j \in \mathbf{N}_i} \{x^1_{i,j}(k\Delta, \mathbf{u}_i) + \\ &\pi\{-\|x_{i,j}(k\Delta, \mathbf{u}_i) - x_{d,l}(k\Delta, \mathbf{u}_{d,l})\|^2 + \tau^2\}\},\end{aligned} \tag{5.14}$$

which results in the defenders always pursuing the intruders.

5.3.2 Risk taking intruders

In this case, the defenders assume that the intruders are willing to take a chance of coming within striking distance of a defender, on the belief that the defender may miss him with a certain probability. For the case of a single defender and single intruder, this results in the following min-max optimal control problem that the defender must solve at each sample time.

$$\min_{\mathbf{u}_d \in \mathbf{U}_d} \max_{\mathbf{u}_i \in \mathbf{U'}_i(\mathbf{u}_d), k \in \mathbf{N}} \phi_1(x_i(k\Delta, \mathbf{u}_i))\phi_2(\mathbf{u}_i, \mathbf{u_d}, N), \tag{5.15}$$

where the probability of a successful strike by the intruder at time $k\Delta$ is

$$\phi_1(x_i(k\Delta, \mathbf{u}_i)) = \frac{\exp(g_1(x_i(k\Delta, \mathbf{u}_i)))}{1 + \exp(g_1(x_i(k\Delta, \mathbf{u}_i)))} \tag{5.16}$$

and the probability of the intruder surviving for horizon of $N\Delta$ sample times is

$$\begin{aligned} &\phi_2(\mathbf{u}_i, \mathbf{u_d}, N) \\ &= \exp\left\{ -\sum_{k=1}^{N} \frac{\lambda \exp(g_2(u_i, u_d, k\Delta))}{1 + \exp(g_2(u_i, u_d, k\Delta))} \Delta \right\} \end{aligned} \tag{5.17}$$

with

$$\begin{aligned} &g_1(x_i(k\Delta, \mathbf{u}_i)) = \\ &- \alpha_1 \left(\|P(z_i(k\Delta, u_i)) - \tau\|^2 - s_1^2 \right) \end{aligned} \tag{5.18}$$

$$\begin{aligned} &g_2(\mathbf{u}_i, \mathbf{u_d}, N) = \\ &- \alpha_2 \left(\|P(z_i(k\Delta, u_i) - z_d(k\Delta, u_d))\|^2 - s_2^2 \right) \end{aligned} \tag{5.19}$$

where $\lambda, \alpha_1, \alpha_2$ are parameters.

5.3.3 Suicidal intruders

This case differs from the preceding one in that no intruder places any value on surviving after a successful attack, but, should his attack be successful will take evasive action. Hence, for the case of a single defender and single intruder, we get the following variant of (5.15)

$$\min_{\mathbf{u}_d \in \mathbf{U}_d} \max_{\mathbf{u}_i \in \mathbf{U'}_i(\mathbf{u}_d), k \in \mathbf{N}} \phi_1(x_i(k\Delta, \mathbf{u}_i))\phi_2(\mathbf{u}_i, \mathbf{u_d}, k). \tag{5.20}$$

where, $\phi_1(x_i(k\Delta, \mathbf{u}_i))$ and $\phi_2(\mathbf{u}_i, \mathbf{u_d}, k)$ are defined as in (5.16) and (5.17), respectively.

When the solution time of (5.20) $k^*\Delta < N\Delta$ i.e., the intruder may have succeeded in destroying his target, then the defender assumes that the intruder switches cost functions at time $k^*\Delta$ and concentrates on escape. In that case, we get the following secondary problem for the defender

$$\min_{\mathbf{u}_d \in \mathbf{U}_d} \max_{\mathbf{u}_i \in \mathbf{U'}_i(\mathbf{u}_d), k \in \{k*, \ldots, N\}} \phi_2(\mathbf{u}_i, \mathbf{u_d}, k). \tag{5.21}$$

Again, the expressions for multiple defenders and intruders are considerably more complicated and are omitted because of lack of space.

5.4 Simulation results

We only present results for the risk averse intruder, based on (5.9). We used a horizon of 10 sample times and used we used the Method Of Outer approximations (MOA) (see [20]) with the Polak-He unified method (PH) (see [20]) as a subroutine.

The approximate solution of (5.9) required 3 iterations of MOA and a total of of 40 iterations of PH. Programmed in JAVA, the solution of (5.9) required 0.18 seconds, while programmed in in MATLAB with TOMLAB [53], it took 1.8 seconds.

Although the Polak-He unified method is only a first-order method, it computes a good approximate solution to an inequality constrained optimization problem very rapidly. Given that we always had very good starting points for the MOA and the speed of the PH method, even using a laptop, we were able to compute controls at a rate that is compatible with real time implementation in craft moving up to 80 knots.

Since static figures do no convey the evolution in time of the defender and intruder trajectories, we have deposited videos of our experiments in
https://sites.google.com/site/walrandberkeley/research/harbor.

5.4.1 Single defender and intruder

Fig. 5.1 illustrates an experimental setup for the case of a single intruder and single defender. The intruder and defender are located in the rectangular channel with a channel width W. The harbor is depicted as a thick line which is located behind the defender. Small red and blue circles indicate the locations of intruder

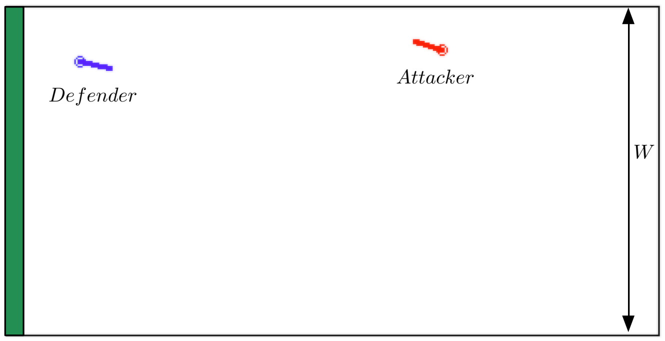
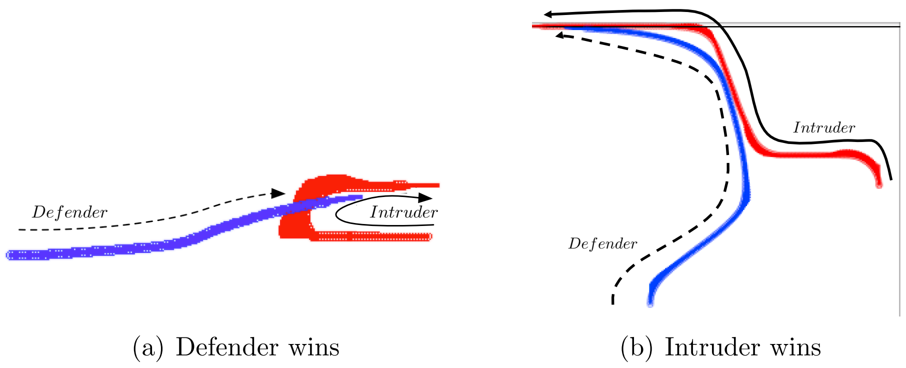


Figure 5.1: Experiment setup

and defender, respectively. The small bar attached to the circles indicates their orientation.

Fig. 5.2 (a) illustrates the case when the defender successfully defends the harbor. The dotted line are added as a trajectory guidance. Fig. 5.2 (b) depicts the case when the intruder successfully outmaneuver the defender, and reaches to the harbor.



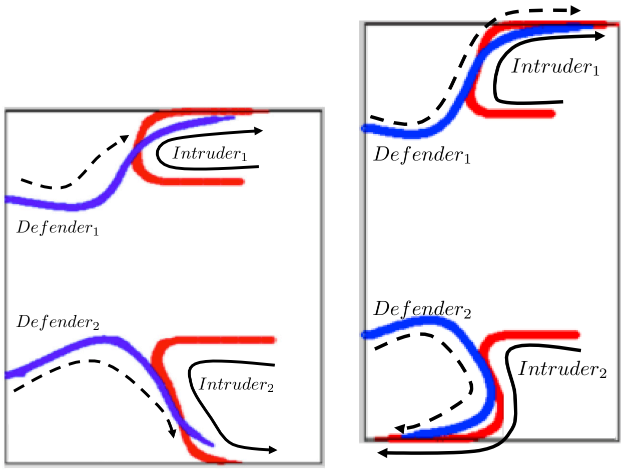
(a) Defender wins (b) Intruder wins

Figure 5.2: Example trajectories: single intruder and defender.

A set of experiments was performed of a type that can be used to determine the maximum channel width that a single defender can protect, assuming that the parameters of the intruder are known. Initially, the intruder and the defender are facing each other: initial intruder and defender orientations are π and 0, respectively. Their controls are bounded by identical limits.

5.4.2 Two defenders and intruders

Fig. 5.3 shows the simulation result with two intruders and two defenders. Initially, they are facing each other. Fig. 5.3 (a) is the case when the defender team wins as the channel is narrow with $W = 20$. Fig. 5.3 (b) is the case when the intruder team wins as the channel is wide with $W = 25$. Since one of the intruder team member successfully reached the harbor, the intruder team wins the game.

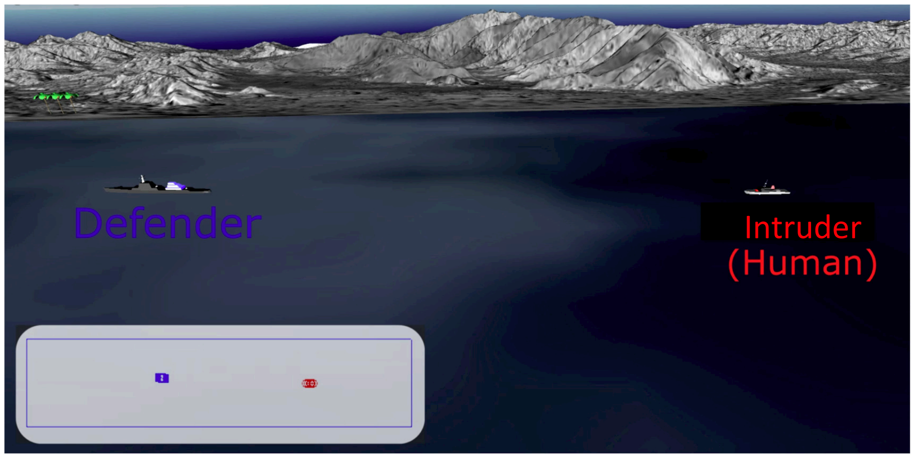


(a) Defender wins (b) Intruder wins

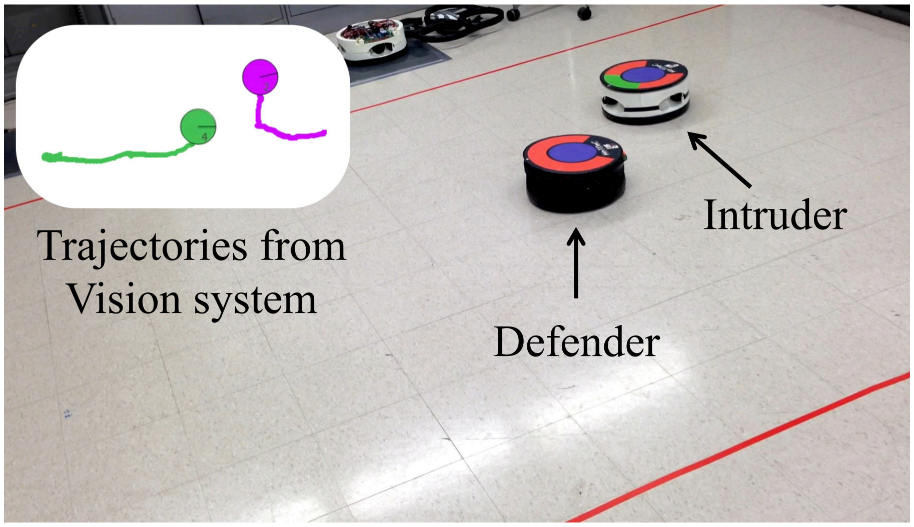
Figure 5.3: Example trajectories : multiple intruders and defenders.

5.4.3 Human-machine interaction

Fig. 5.4 (a) is a screen capture of a real time 3D simulation. We assume the harbor is located behind of the defender (left side of the screen). The human controls the intruder. Fig. 5.4 (b) is a photo of a laboratory experiment involving a human intruder and an autonomous defender. Both in the 3D simulation and in the experiment, the human intruder uses a joystick to activate the intruder. In the experiment, HoTDeC (HOvercraft Testbed for DEcentralized Control) vehicles developed at the University of Illinois at Urbana-Champaign (UIUC) were used as players.



(a) 3D simulation



(b) Static photo of experiment

Figure 5.4: Human-computer interactive real-time simulation and experiment

CHAPTER 6

TESTBED DEVELOPMENT

6.1 Introduction

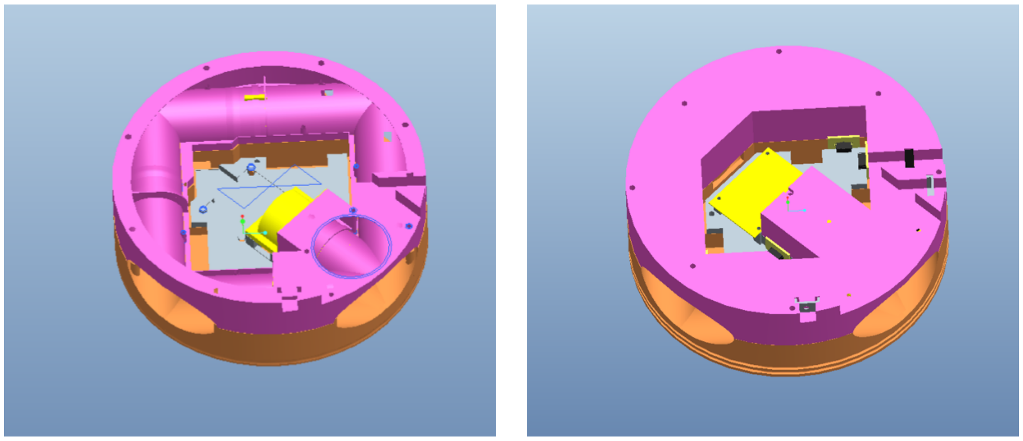
In this section, we present the development of the testbed in hardware and software. The hardware side is most about HoTDeC (Hovercraft Testbed for Decentralized Control), the software side includes network, control, and vision programming. The new generation of HoTDeC (Hovercraft Testbed for Decentralized Control) vehicle is developed at the University of Illinois at Urbana-Champaign (UIUC) and it is used as a robotic testbed. Detailed information about earlier versions of HoTDeC can be found in the dissertation [59].

There are two processors in each HoTDeC. The main processor is in the Gumstix Overo. It supports a fully featured real-time embedded Linux operating system called Linaro. Gumstix Overo has been widely used for robotic applications. In [60], small ground vehicles are built as a swarm. In [61] and [62], it is used for the quad-roter implementation. In this work, a Linear-Quadratic Regulator (LQR) with a standard Kalman filter is running in Gumstix as a position and orientation controller. Gumstix also handles the subscription of the message from the vision server or other agents. The other processor is a Texas (TMS320F28335) Instrument digital signal processor (DSP) [63]. This processor controls angular velocity of the five thrusters that run at over 10,000 RPM. Angular speed is sensed using a Hall-effect sensor. To control the thrusters angular velocity at high RPM, registers in the processor are directly handled. A simple network layer is established between Gumstix and DSP using serial communication.

6.2 Hardware

6.2.1 HoTDeC body

There are two different types of HoTDeC body: 3D printed and precision machined. The material of the machined body and the 3D printed body are dense



(a) 3D Printed body (b) Precision machined body

Figure 6.1: Two different types of the HoTDeC body.

styrofoam and ABS-M30, respectively. The weights of he machined body and the 3D printed body are 325g and 565g, respectively.

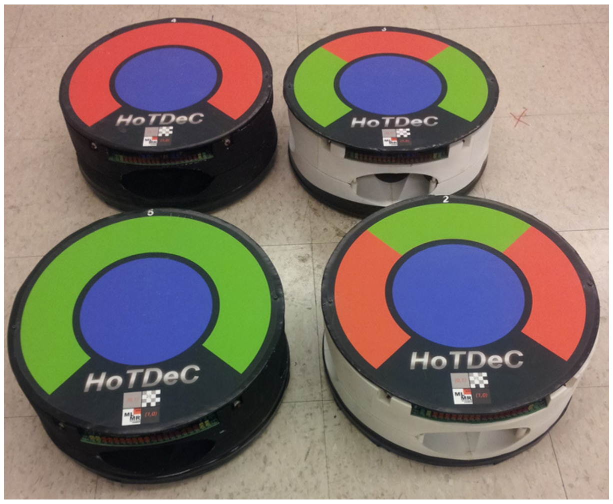


Figure 6.2: Different patterns of top

Fig. 6.2 shows HoTDeC vehicles. Each top is a visual vehicle identifier. HoTDeC vehicles have four lateral thrusters and it uses a micro controller to generate commanded forces and moments; an additional thruster is used for lift. There are two types of HoTDeC vehicles, both produced by the rapid prototyping laboratory in UIUC (3D-printed, precision machined).

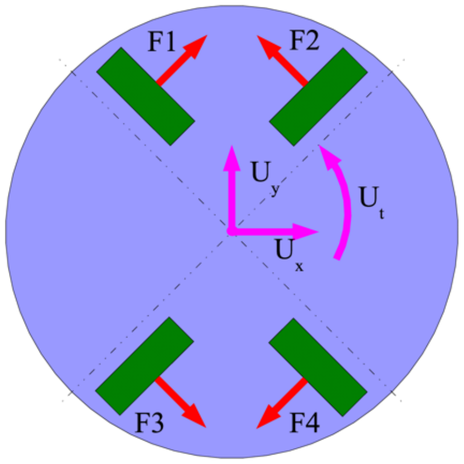


Figure 6.3: Thruster configuration

There are five thrusters in each HoTDeC. One is for the hovering, the other four is for the positioning and the orientation. Fig. 6.3 shows the configuration of the four thrusters. They generates thrust forces $F1, F2, F3$ and $F4$ in a body fixed frame (U_x, U_y axis). When $F2, F4$ are on, HotDeC generates thrust force to the positive U_x direction. When $F3, F4$ are on, it generates thrust force to the positive U_y direction. Similarly, $F1, F4$ generates moment to the positive U_t direction.

6.2.2 Main Board

The schematic of the main board is presented in the Fig. 6.4. The main board is in charge of integration of the peripherals such as Gumstix, DSP, and Powerboard. The current version of the main board has two slots for Gumstix and DSP board respectively. They can be turned on and off by the toggle switches, and they are connected through the serial port. The serial port can be selected by switching the jumper. The Gumstix can receive user commands such as reference thruster speed and directly passes them to the DSP. The DSP's interfaces contain an input, output pairs for each thruster. Five of them are currently used to get the the Hall effect sensor input and give PWM signal out. Logic converters are used to convert the DSP output to 5V. We also have jtag in the right and RS232 in the left side of the board. There are two analog input ports, which are currently unused.

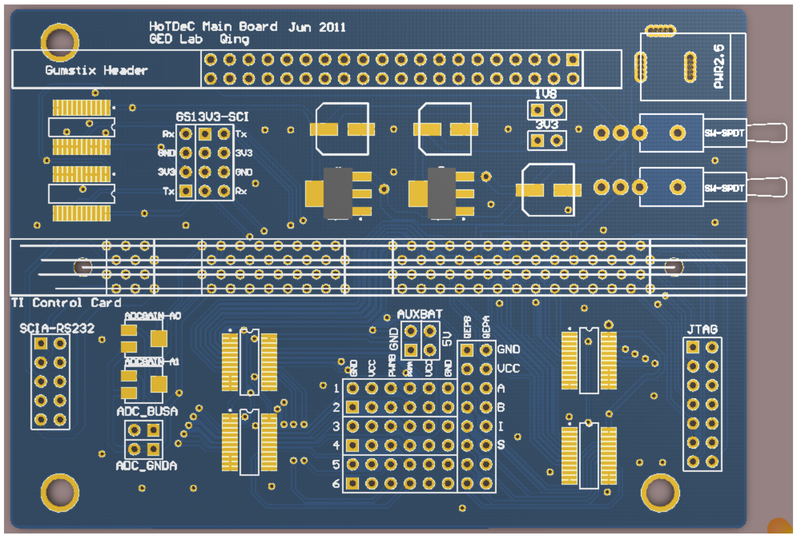


Figure 6.4: Main Board

6.2.3 Isolator Board

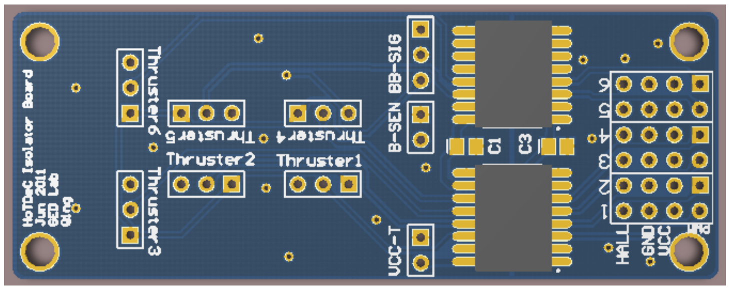
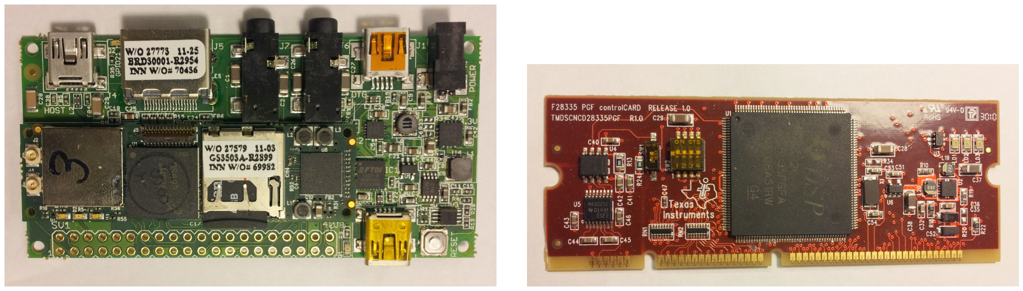


Figure 6.5: Isolator Board

The isolator board is designed to isolate the digital circuit from the noise. The signal from the thruster and DSP are separated by this board. We have six set of interfaces and five of them are used for each HoTDeC. From the front view of the board, left to right, three pins should be connected to : hall effect sensor, PWM signal and kill signal, respectively. Note that both side of the powers of the isolator should connect to 5V. As shown in Fig. 6.5, the right part is powered from the main board and the left side gets the power from the power board (VCC-T, and the pin closer to the center is GND). The isolator can also be used to measure the battery voltage and gives the analog output. The BB-SIG connector should be connected with the battery and the pin closer to the center of the board is kill signal, which is connected to the power board.

6.2.4 Control boards

There are two commercial board in the HoTDeC. The Gumstix Overo fire and Texas Instrument DSP (TMSF28335). The Gumstix is full-featured linux computer that enables the HoTDeC to work as an embedded linux machine. Key features of the Gumstix are the Wifi module and the ARM 8 Core processor. Each thruster in the HoTDeC runs more than 10,000 RPM in normal operation. To satisfy the fast computation requirement, an additional DSP is introduced. The DSP is in charge of the speed control of five thrusters.



(a) Gumstix Overo fire (b) TMSF28335 DSP

Figure 6.6: Gumstix Overo fire and TMS320F

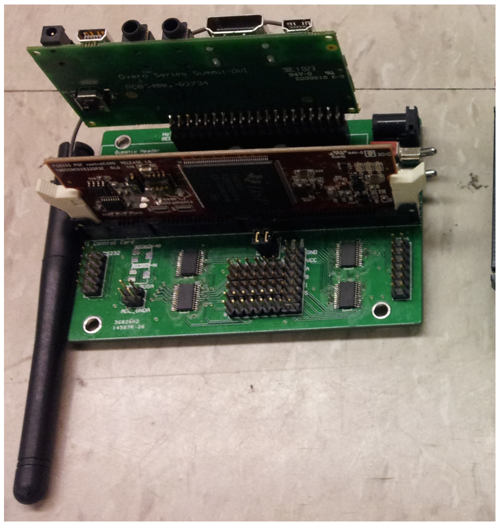


Figure 6.7: Gumstix and DSP on the main board

Fig. 6.7 shows the Gumstix and DSP integrated to the main board.

6.3 Software

6.3.1 Network

One of the important feature of the testbed we developed is a network and the aim of the network is to allow whole system a dynamic message exchange. The following figure shows this concept.

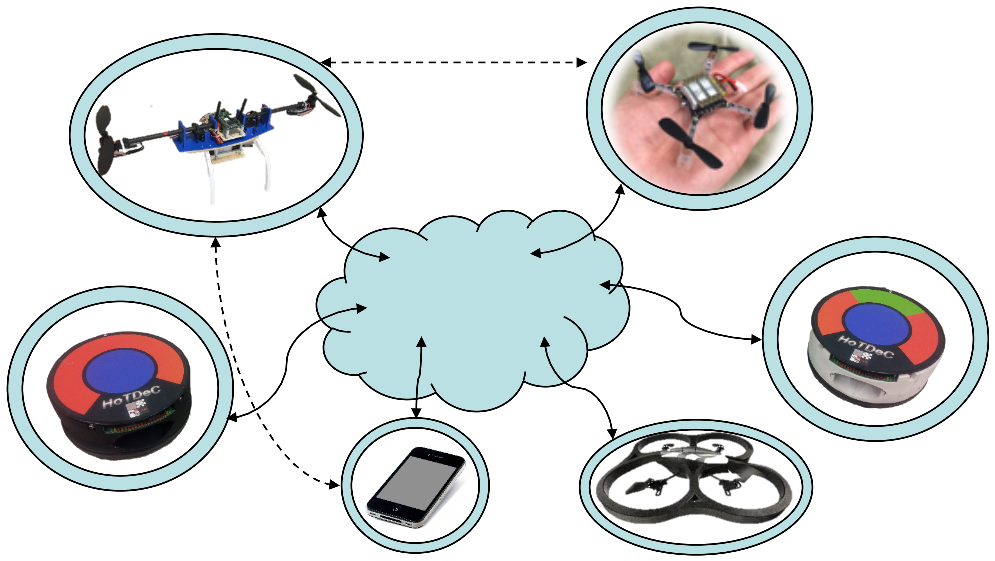


Figure 6.8: Concept of the remote testbed.

The circular layer encapsulates the individuals such as HoTDeC, Drone. This constructs the abstract layer around the individual agents. Because the agents are encapsulated by the layer, and network modules can be accessed only through the layer, the specific type of the individual agent is not required for the message exchange. This allows dynamic registration and removal of the agents in the local network. Another feature is the discovery service. Each agent has knowledge about the available resources around them. This brings efficiency to the network since each individual do not have to know the entire network.

There are two different concepts for the network programming. One is flexibility, and the other is efficiency. In this research, the desirable network architecture depends on the application. For example, when we operate HoTDeC manually, one directional channel is sufficient. However, when we implement centralized defense strategy to the multiple HoTDeCs, the publish-subscribe structure is more suitable. To incorporate various applications, the flexible network architecture that is based on the ZeroMQ is developed. Another requirement is the efficiency. This is particularly important in inter-process communication between Gumstix and DSP as thruster speed controller runs in fast frequency, 1KHz. In for this requirement, one of the lightest weighted protocol, serial communication is used.

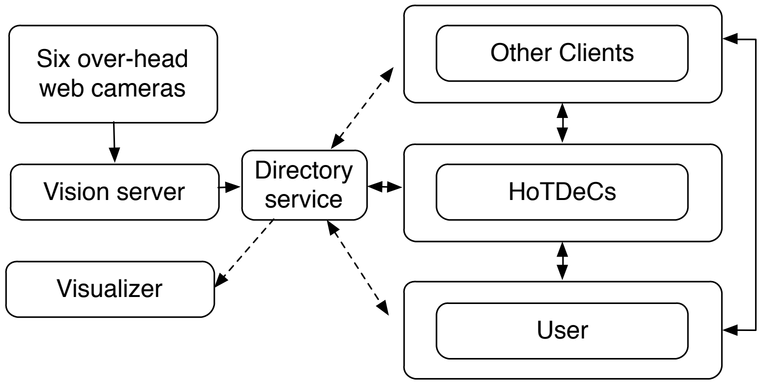


Figure 6.9: Network environment: solid line is for the direct communication and dotted line is through the discovery service.

Fig. 6.9 schematically depicts the network environment. There are six overhead cameras connected to the vision server. The vision server broadcasts position and orientation of each agent through the Directory service in the local network. We have developed an abstract network layer for each agent (HoTDeC, Drones, PC, or phones). This environment allows a reliable network under dynamic message exchanging. Each agent can either directly exchange the message(solid line) or through the Directory service. As agents dynamically exchange the messages in our local network, with a unique message format, an abstract network layer for each agent is developed. Fig. 6.10 shows network layer that encapsulates the agent. It consists of three parts: transport, serialization, and service discovery. The complete discussion of our network is in the thesis [64].

Transport layer represents the software layer responsible for defining rules that govern the transfer of data from one location to another. We use ZeroMQ in this layer. Our transport layer supports flexible structure in this layer in a sense that both Request-Reply, and Publish-Subscribe structure can be switched easily.

Fig.6.11 shows two different basic message patterns in the transport layer. Request-reply pattern is the simplest model that allows the confirmation of the receiving the message. In other words, that the socket pair is in lockstep. The other pattern, publish-subscribe is to push the updates to a set of subscribers. In this pattern, the sender is not responsible for receiver to receive the message correctly, and hence the network packet might be lost. This character is sometimes called, “Fire and Forget”.

Fig.6.13 shows the concept of the serialization. Once the network program

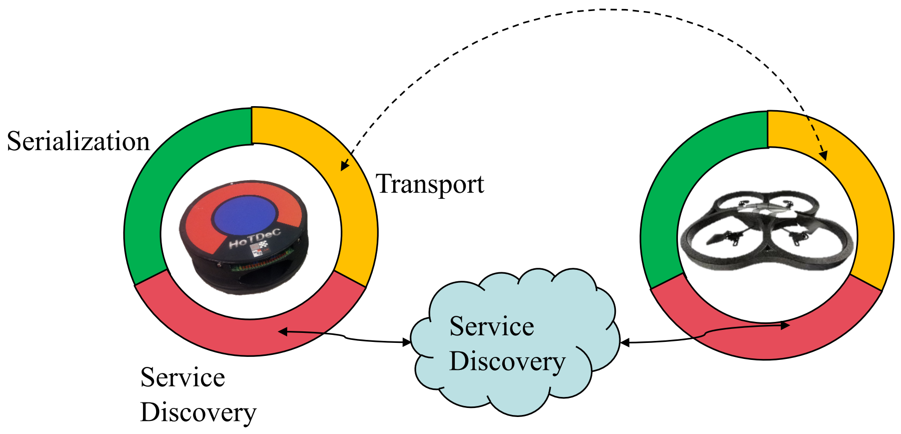
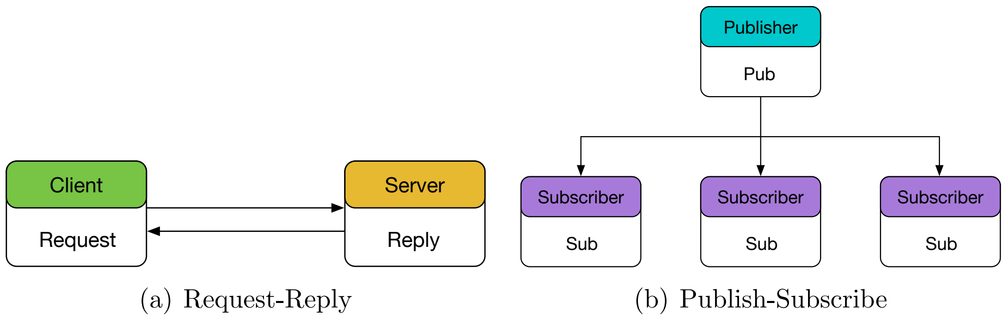


Figure 6.10: Encapsulation of the agents: each agent is encapsulated with an abstract network layer and the layer consists of three parts.



(a) Request-Reply (b) Publish-Subscribe

Figure 6.11: Two basic message patterns

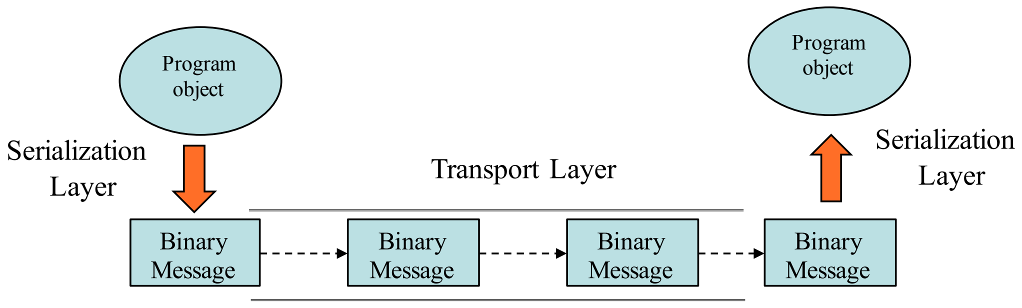


Figure 6.12: Serialization: packet is created, transmitted through the transport channel, received and parsed in order.

creates the objective that needs to be sent, the message packet is created according to the predefined rule. Then the message is transmitted through the transport layer. Receiver receives the message and parse the data according to the parsing rule. These action takes place in serial. Serialization defines how an object is converted into a format that can be stored or transmuted over the network through the transport layer. In this layer, we use JavaScript Object Notation (JASON) and Protocol Buffers. As JASON is simple to use as it is text-based. It is used in inter-agent message exchange. Protocol Buffers are an efficient way of encoding data into messages. It specifies an binary format for encoding data as message, whereas JASON messages are free-form. Therefore, it is used in inter-process message exchange (between Gumstix and DSP).

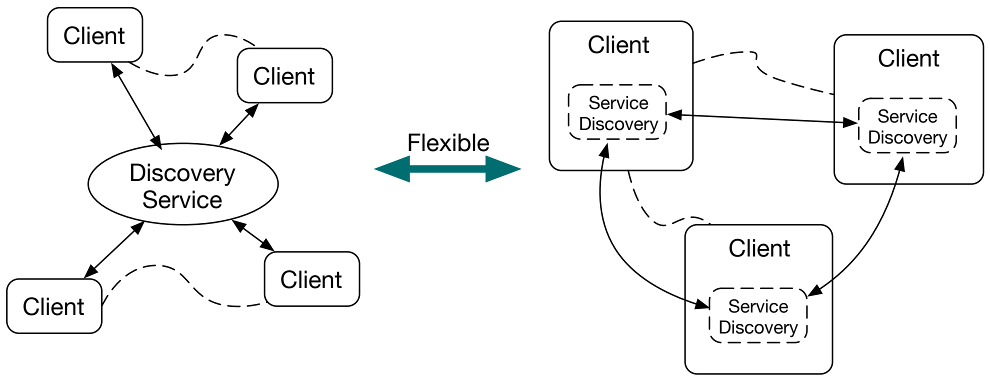


Figure 6.13: Service discovery: our discovery service provides both centralized and peer-to-peer ad hoc structure.

Service discovery defines functionality by which an application can find other applications on the network which provide the services that it needs access to. As our lab has unique type of message and they are dynamically interfacing each other, custom built software called "Directoryd" is developed in this layer. It is a software package that provides service directory support for the HoTDeC is developed. It is deposited in GitHub [65]. There should be one instance of Directoryd running on each agent.

6.3.2 Control

There are two control layer in the HoTDeC. The higher level of the control is about the position and orientation control and the lower level of the control is to control the thruster speed. In high level control, Linear-Quadratic Regulator (LQR) with a standard Kalman filter is implemented in the Gumstix. In the lower level thruster speed control, PD controller is implemented in the DSP.

The dynamic model of the HoTDeC is linear model with two inputs.

$$\begin{aligned} \dot{X} &= AX + BU + GW \\ Y &= CX + DV, \end{aligned} \tag{6.1}$$

where the states are positions and velocities in cartesian coordinate, $X =$

$[x, \dot{x}, y, \dot{y}, \theta, \dot{\theta}]$. with following state, input, output matrices

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{\beta_x}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{\beta_y}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -\frac{\beta_\theta}{J} \end{bmatrix} \tag{6.2}$$

$$B = G = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{J} \end{bmatrix} \tag{6.3}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \ V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{6.4}$$

The states for the control inputs are $U = [u_x, u_y, \tau]$, actuator disturbance input $W = [w_x, w_y, w_\theta]$, measurement noise $V = [v_x, v_y, v_\theta]$. Detailed information about the disturbance and noise are presented in [59]. The mechanical parameters of the HoTDeC is presented in the following table.

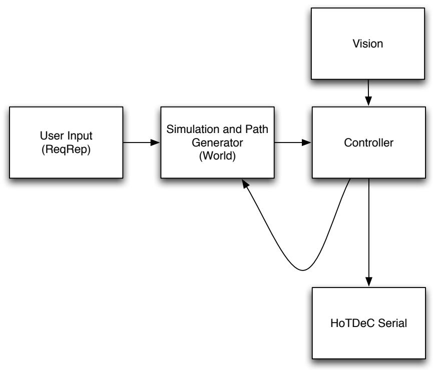
Table 6.1: Mechanical parameters of the HoTDeC

	Mass $(m, [kg])$	Moment of inertia $(J, [kg \cdot m^2])$
Form Body	1.827	0.021
RP Body	2.26	0.028

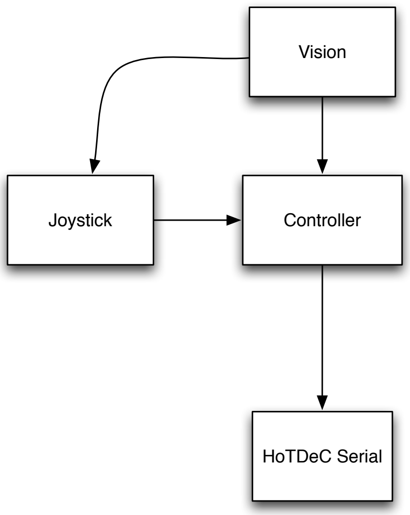
The schematic presented in Fig 6.14 (a) illustrates the required softwares to run HoTDeC autonomous mode.

The vision system contains one vision server and three clients. Each client processes the images taken from the web camera and send it to the vision server. The server merges and blends the images and creates result image.

The role of the simulation and the path generator program called "World" is to generate real-time reference trajectory according to the user input. It geometrically computes the reference trajectory so that HoTDeC can reach to the target position. One of the main features of the real-time reference generator is a smooth transition between a linear path and circular path. By combining line and circle, it allows



(a) Autonomous mode



(b) Joystick input manual mode

Figure 6.14: Autonomous and manual mode operation

dynamic trajectory generation. This operation can be done using “Request-Reply” structure.

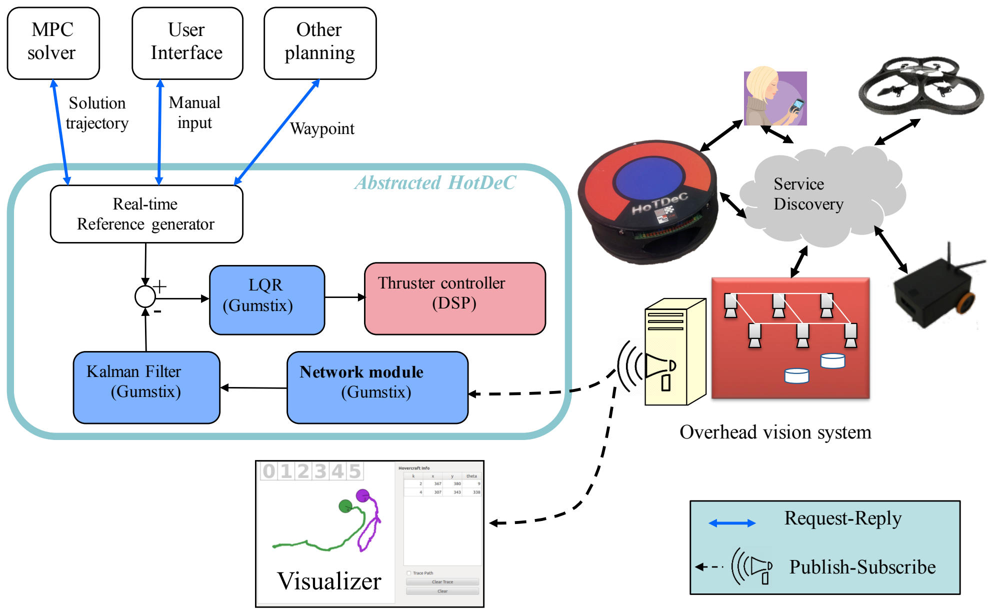
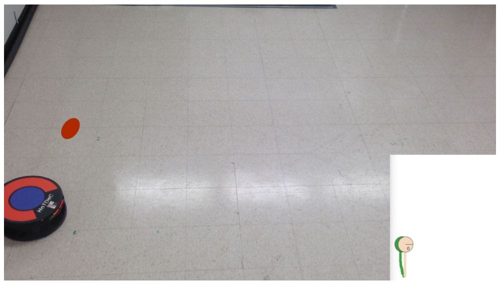
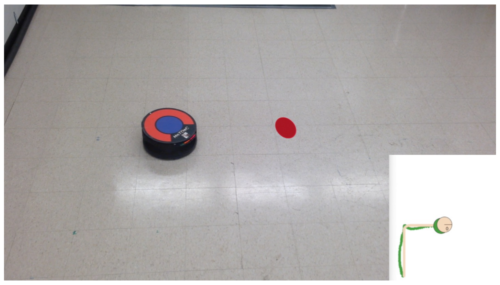
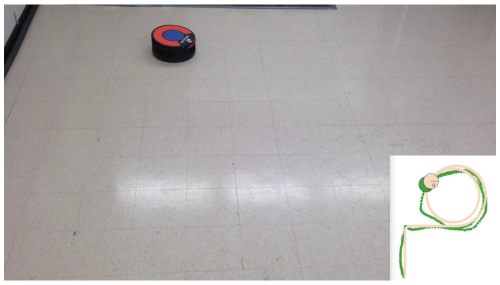


Figure 6.15: Control and network signal flow diagram.

Fig. 6.15 presents the control and network flow diagram of HoTDeC. The solution for the MPC problem, user manual waypoint input, or the solution from the other approach is given as a input. The real-time reference generator generates the optimal position and velocity profile of the HoTDeC from the current position to the target point. It also generates smooth transition between current waypoint trajectory to the new waypoint trajectory provided that the user gives new command input before the HoTDeC is not finished the current task. The LQR controller and Kalman filter that are embedded in the Gumstix generates control input. Control input is translated to the thruster inputs and transmitted to the thruster controller that is implemented in the DSP. While the HoTDeC is operating, the Network module which is in Gumstix is responsible for subscribing the position and orientation for the required HoTDeCs. The loop is closed in this fashion.


(a) Waypoint 1.


(b) Waypoint 1 to 2.


(c) Waypoint to circles.

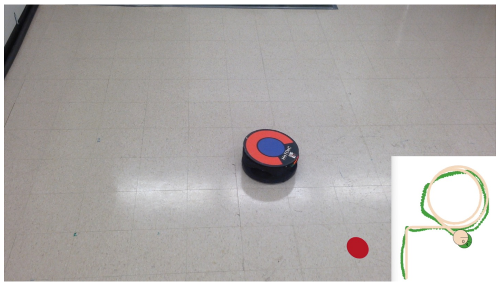

(d) Circle to waypoint.

Figure 6.16: Real-time reference generation and following: reference consists of lines and circles

Fig. 6.16 shows screen captures of a HoTDeC trajectory following experiment. Green path is a trajectory from the experiment and light-red trajectory is a reference, which is generated in real-time at user command. HoTDeC starts from the left bottom corner at user's first waypoint command (Fig. 6.16(a)). When the HoTDeC reaches to it, the second way point (Fig. 6.16(b)) is given by the user. In the middle of transition, user commended a circle reference input with a large radius. To comply with a new user commend, HoTDeC smoothly switches reference from line to the circle reference (Fig. 6.16(c)). Followed by another transition from large circle to the smaller circle, another waypoint is commended. HoTDeC smoothly switches his reference from circle to waypoint (Fig. 6.16(d)).

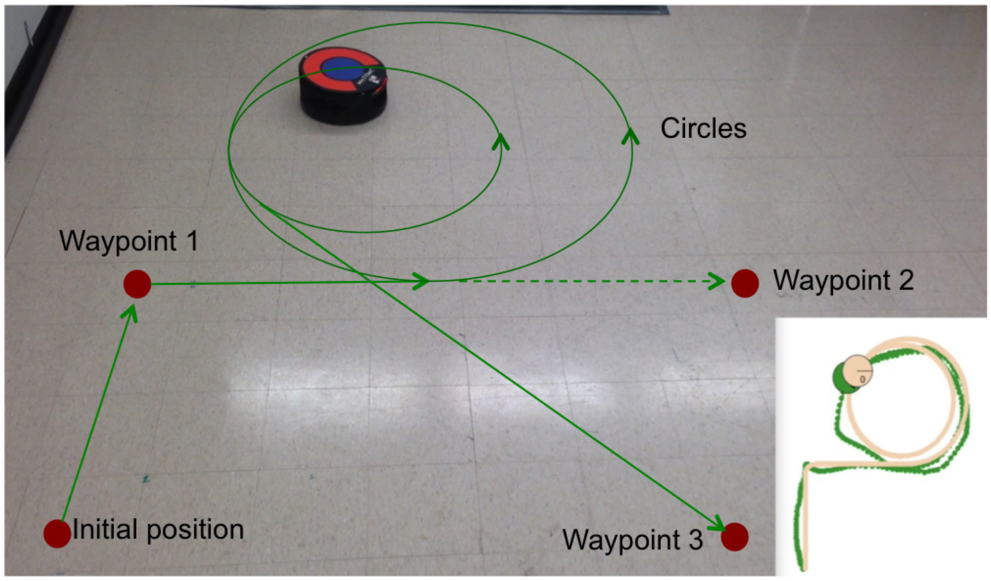


Figure 6.17: Real-time reference generation and following: in a single figure

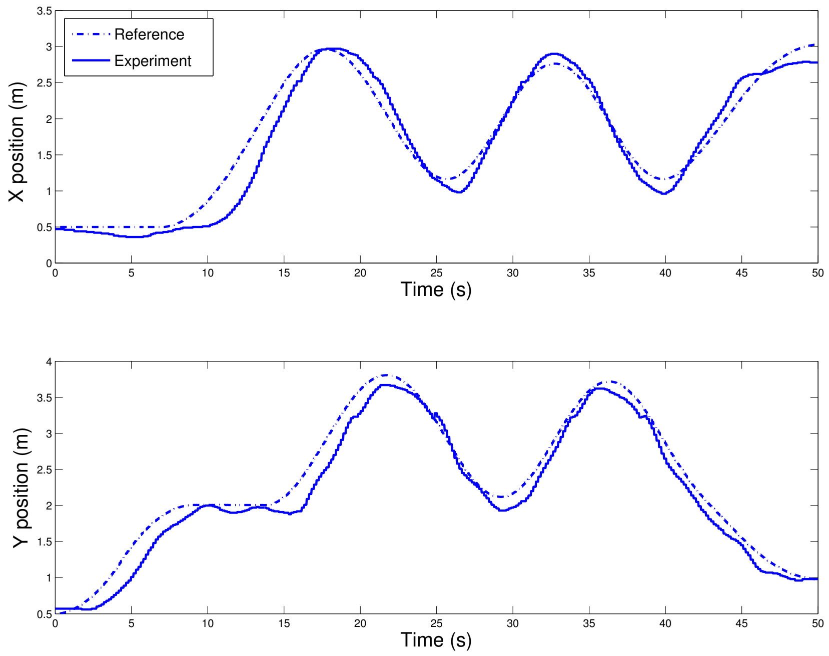
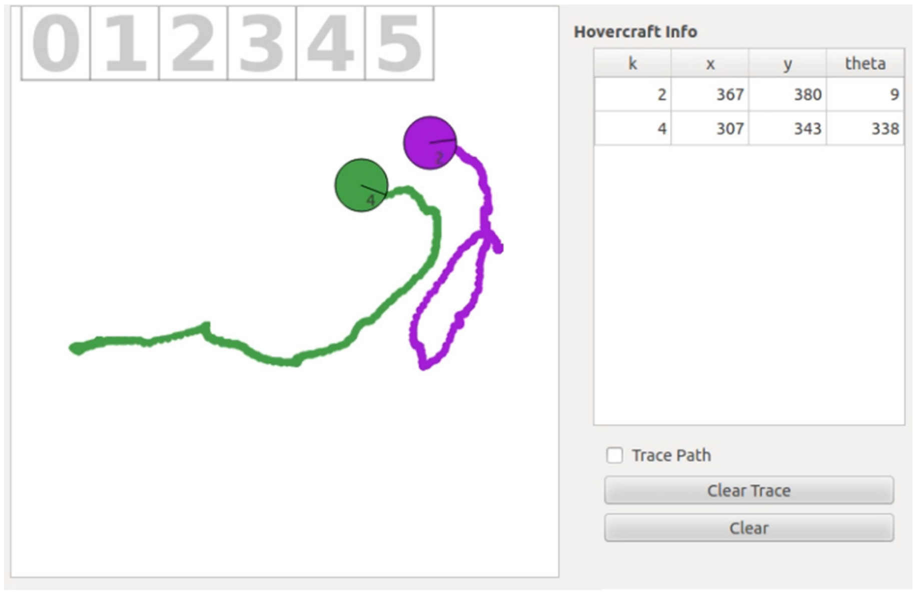


Figure 6.18: Reference following result: position error is less than 5%

Fig. 6.17 shows overall trajectory in a single figure. Dotted line indicates that planned trajectory but canceled due to user's new input.

Fig. 6.18 shows the reference following experiment result. The position error are less than 5%. Average speed of HoTDeC is approximately 0.7m/s.



k	x	y	theta
2	367	380	9
4	307	343	338

Figure 6.19: Visualizer

The Fig 6.19 is a screen capture of the Visualizer written in Python. The visualizer provides the information about the position and orientation real-time as well as reference trajectory as an option.

The manual input mode via joystick is similar to the autonomous mode. The difference is the the joystick server. Joystick server generates the references for the position and velocity. To generates the references, it requires vision information as presented in Fig. 6.14 (b). There are two versions of joystick server, the difference is only a language. One is in Java, the other one is in C++.

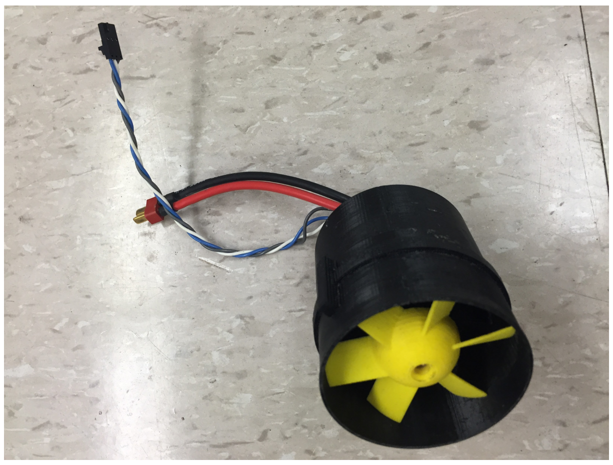
6.3.3 DSP Code

Once the LQR controller generates control inputs u_x, u_y, u_θ, desired RPM of each thrusters are found by the look up table [67]. Fig. 6.20 are the photos of the thruster assembly. Yellow propeller is connected to the DC motor. Black colored housing is 3D printed so that it fits to the HoTDeC. The Fig. 6.20 (b) is the back side of the assembly. Small circuit that is attached to the motor is a hall effect sensor circuit. The PI controllers are running in the DSP to control the thruster speed. The reference signal that the DSP requires is the force from the thruster. The detailed explanation regarding the dynamic model and the controller is presented [67]. It is assumed that the relationship between the force F and the thruster angular velocity ω is linear: $\omega^2 = 2560000 \times F$. Following is the thruster dynamic model:

$$\begin{aligned} \dot{x} &= -8.6x + u \\ \omega^2 &= 20868760x, \end{aligned} \tag{6.5}$$

with $F = 8.1518x - 0.02$, if $x > 0.00246$. $F = 0$, otherwise. The speed controller is the following

$$u = K_P(r - \omega) + K_I \sum_{i=0}^{t} (r - \omega), \quad K_P = 5.12, K_I = 60. \tag{6.6}$$


(a) Thruster assembly

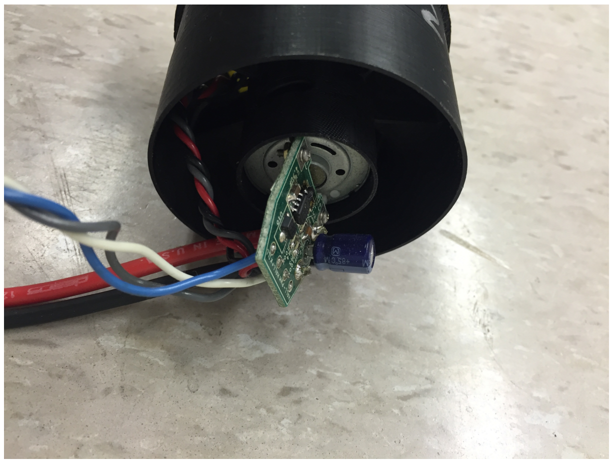

(b) Back side of the thruster assembly

Figure 6.20: Thruster assembly

CHAPTER 7

HARDWARE IMPLEMENTATION

7.1 Introduction

In this chapter, we mainly discuss about implementation strategy of minmax MPC scheme. There has been an increasing interest in the development of fast MPC implementations. One promising method, called *explicit MPC*, uses a look-up table to access the explicit precomputed solution [68], citetondel2003algorithm. The potential drawback of this method is that the number of entries in the table can grow exponentially with the horizon length, state, and input dimensions. Therefore, explicit MPC can only be applied reliably to situations with "small" problems [69].

Until recently, the real-time application of MPC in robotics has been limited to simple tasks. In [70], an explicit MPC solution is developed and implemented for mobile robot trajectory tracking. In [71], an MPC based obstacle avoidance control law is developed and implemented in a mobile robot which is operated in a master-slave teleoperation configuration, not fully autonomously. In [72], an autonomous bicycle robot with an MPC law for balancing is described. The reported simulation results are promising, however, hardware experiments have not yet been successful. In [73], a vibration attenuation problem in linked, linearized robot dynamics is addressed with an MPC law. In [74], MPC is used in trajectory following for an under-actuated radio controlled model hovercraft, which has two thrusters each able to generate three discrete control values: positive force, negative force, and zero force.

In our previous work [58], a MPC law was constructed for a defender based on a max-min optimal control problem which we believed captured the essence of the intruder goals. In this paper, we describe the implementation of the results of [58] in a real-time hardware situation, in which the defender and intruder are custom-built hovercraft, with the defender controlled by a computer and the intruder by a human.

To obtain a numerical method for solving the required max-min problems, that is compatible with real-time implementation, we combined the outer approximations algorithm with a customized optimization code implementing the Polak-He [20] first-order minimax algorithm. For obtaining medium precision solutions, this approach is much more efficient than using most optimization algorithms available in commercial or free libraries. Furthermore, it resulted in a transparent code, which provided us with an open debugging environment, and thus allowed us to remove unnecessary error-checking routines once code was verified offline. Finally, we developed an abstract network layer which encapsulates each agent in a local network. This layer allows reliable and effective dynamic inter-agent and inter-process message exchange.

7.2 Dynamic Models and model predictive Control

7.2.1 Problem statement

We model the intruder by a nonlinear dynamic model of the form

$$\dot{z}_i(t) = f(z_i(t), u_i(t)). \tag{7.1}$$

The vectors $z_i(t)$ and $u_i(t)$ are state and input vectors, respectively. The state vector $z_i(t)$ includes horizontal and vertical positions $x_i(t)$ and $y_i(t)$. In order to use a model predictive control scheme, we discretize (7.1) using a sampling time Δ, with results in discrete-time dynamics of the form

$$z_i((k+1)\Delta) = \bar{f}(z_i(k\Delta), u_i(k\Delta)), k = 1, 2, \cdots, N \tag{7.2}$$

We define the horizon length to be $N\Delta$, with strictly positive integer N. We assume that the defender dynamics is in similar form.

Considering the intruder's objective, it is natural to formulate the intruder minimization problem as in below. This is the problem that the defender *assumes* that the intruder solves. Thus, in our formulation, the defender explicitly assumes that the intruder also uses an MPC control law, based on minimizing its distance from the HVUs, as follows:

$$\min_{\mathbf{u_i} \in \mathbf{U_i}(\mathbf{u_d}), k \in \mathbf{N}} \{x_i(k\Delta)\} \tag{7.3}$$

with

$$\begin{aligned}\mathbf{U}_i(\mathbf{u}_d) = \{\mathbf{u}_i =& (u_i(\Delta), \ldots, u_i(N\Delta)) \in \mathbb{R}^N \times \mathbb{R}^N \\ & s.t.\ \|u_i(k\Delta)\|^2 \leq \alpha_i^2, \\ & \quad \|P(z_i(k\Delta) - z_d(k\Delta))\|^2 \geq \delta^2, \\ & \quad 0 \leq x_i(k\Delta), \\ & \quad 0 \leq y_i(k\Delta) \leq W, \quad k = 1, \ldots, N, \\ & \quad z_i(k\Delta), z_d(k\Delta) \text{ satisfying (7.2)}\},\end{aligned} \tag{7.4}$$

where W is the harbor width (channel), α_i is the limits for the control of the intruder and a function P extracts the position vector from the state vector.

$$[x(t), y(t)]^T = P(z(t)) \tag{7.5}$$

Similarly, the feasible control set of the defender is

$$\begin{aligned}\mathbf{U}_d = \{\mathbf{u}_d =& (u_d(\Delta), \ldots, u_d(N\Delta)) \in \mathbb{R}^N \times \mathbb{R}^N \\ & s.t.\ \|u_d(k\Delta)\|^2 \leq \alpha_d^2, \\ & \quad 0 \leq x_d(k\Delta), \\ & \quad 0 \leq y_d(k\Delta) \leq W, \quad k = 1, \ldots, N\}\end{aligned} \tag{7.6}$$

where δ is a torpedo distance. The set of intruder controls U_i depends on u_d because of the existence of the hard constraint in (7.3): $\|P(z_i(k\Delta) - z_d(k\Delta)\|^2 \geq \delta^2$. The Defender problem is based on a worst case scenario defined by the solution of (7.3). Thus, at each sample time $k\Delta$, the defender solves the following max-min optimization problem

$$\begin{aligned}&\max_{\mathbf{u}_d \in \mathbf{U}_d} \min_{\mathbf{u}_i \in \mathbf{U}_i(\mathbf{u_d}), k \in \mathbf{N}} \{x_i(k\Delta)\} \\ &s.t. (7.4)\end{aligned} \tag{7.7}$$

and uses the first element of the optimal solution $\hat{u}_d$ as its control for the time interval $[k\Delta, (k+1)\Delta)$. As there is no known method to solve (7.7) directly, the solution for (7.7) is elaborated on [58] and Chapter 5.

7.2.2 Implementation algorithm

We present a new algorithm based on [58]. The new algorithm consists of explicit terminal condition, penalty function which yields numerically better condition. We also describe the algorithm in terms of custom-written code, open source libraries and commercial software in Algorithm 4 below.

In [58], an exact penalty function was used for the distance between the pursuer and evader. Since it is not differentiable, we replaced by barrier function ((7.8), (7.9)), which is computationally less demanding while capturing the essence of the penalty. A potential degradation of the precision of the solution is insignificant in our application.

$$p(z_i(k\Delta), z_d(k\Delta)) := \pi \sum_{k=1}^{N} D(z_i(k\Delta), z_d(k\Delta)), \tag{7.8}$$

where

$$D(z_i(k\Delta), z_d(k\Delta)) := \log(\|P(z_i(k\Delta) - z_d(k\Delta))\|^2 - \delta^2). \tag{7.9}$$

The algorithm below includes a termination condition based on a comparison of of controls $\hat{u}_d^i$ and $\hat{u}_d^{i-1}$ obtained in two consecutive iterations. It returns a boolean value: true if they are similar enough, false otherwise. Hence, the execution time of the algorithm varies at each sample time and this feature results in a dynamic algorithm.

In Step 1, defender solves finite minmax problem using Polak-He algorithm [20] as an inner algorithm. All gradient information is pre-computed. Solution for the inner algorithm consists of solving quadratic programming. This is done by using JOptimizer library [66]. The approach for solving the minimization problem in Step 2 is similar to the Step 1. Necessary sets are updated in Step 3, so that solution for Step 1 is updated in the next outer iteration. As terminal condition of the outer loop is satisfied, optimal control is obtained.

Once the $\hat{u}_d$ is obtained, reference trajectory is computed using (7.2). In the reference trajectory, initial fraction is fed to the defender. Defender generates reference and follow it using embedded position controller. The performance of the reference following is presented in the next section.

```
Result: Defender control $\hat{u}_d$
while Running the experiment do
    Subscribe position data from local network.
    Set $i = 0$.
    while terminal condition: $T(\hat{u}_d^i, \hat{u}_d^{i-1})$ = false do
        Step 1. Obtain $\hat{x}^i$ such that $\{u_{d,m}\}_{m=0}^{\infty} \to \hat{u}_d^i$ by solving inner
        problem:
            $$\left.\begin{aligned} \max_{\mathbf{u}_d \in \mathbf{U}_d} \min_{\substack{\mathbf{u}_i \in \mathbf{U}_{i,k} \\ \mu \in \boldsymbol{\Sigma}}} \{ & \sum_{k=1}^{N} \mu^k x_i^1(k\Delta, \mathbf{u}_i) \\ & - p(z_i(k\Delta), z_d(k\Delta))\} \end{aligned}\right\} \textit{Inner loop}$$
        Step 2. Solve intruder problem:
            $$\min_{\substack{\mathbf{u}_i \in \mathbf{U}_i \\ \mu \in \boldsymbol{\Sigma}}} \{\sum_{k=1}^{N} \mu^k x_i^1(k\Delta, \mathbf{u}_i)\}$$
        Step 3. Update $\mathbf{U}_{i+1} = \mathbf{U}_i \cup \{u_i\}$, $\mathbf{I}_{i+1} = \mathbf{I}_i \cup \{i+1\}$, Replace
        $i = i + 1$.
        Step 4. Evaluate terminal condition: $T(\hat{u}_d^i, \hat{u}_d^{i-1})$ of the outer
        loop
    end
    $\hat{u}_d = \hat{u}_d^i$
    Generate reference.
    Take initial part of it as a current reference.
end
```

Algorithm 4: Implementation algorithm

7.3 Simulation

Two human-computer interactive real-time simulators were developed. One with the linear dynamics:

$$\dot{z}(t) = \bar{A}z(t) + \bar{B}u(t), \tag{7.10}$$

with

$$\bar{A} = \begin{bmatrix} 0\ 0\ 1\ 0 \\ 0\ 0\ 0\ 1 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \end{bmatrix} \quad \bar{B} = \begin{bmatrix} 0\ 0 \\ 0\ 0 \\ 1\ 0 \\ 0\ 1 \end{bmatrix}. \tag{7.11}$$

,where $z(t) = [x(t), y(t).\dot{x}(t), \dot{y}(t)]$ and $u(t) = [u_x(t), u_y(t)]$. HoTDeC is modeled using this linear model. Input is x and y thrust force. The other one with nonlinear ship dynamics:

$$\begin{aligned} &\dot{x}(t) = v(t)\cos\theta(t), \quad \dot{y}(t) = v(t)\sin\theta(t) \\ &\dot{\theta}(t) = u_\omega(t), \quad \dot{v}(t) = u_a(t) \end{aligned} \tag{7.12}$$

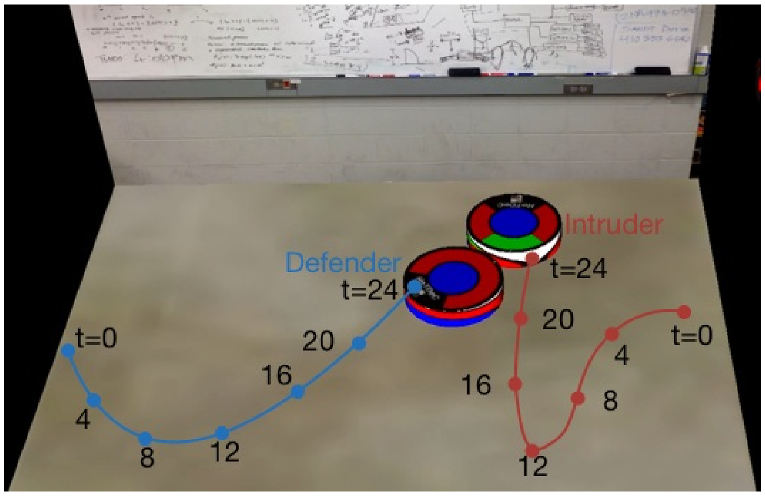
. We state the non-linear ship dynamics with the angular velocity input $u_\omega(t)$ and force input $u_a(t)$. These can be seen as modified Dubins-car dynamics. The human plays the role of the intruder via joystick input.

Fig. 7.1 is a screen capture of the simulation. Fig. 7.1 (a) is a simulation with the linear dynamics (7.8). Fig. 7.1 (b) is a simulation with the non-linear dynamics (7.10). We assume the harbor is located behind of the defender (left side of the screen). The human plays the part of the intruder in both simulations. Simulation is written in Matlab with VRML (Virtual Reality Modelling Language). Horizon length $N = 10$. In both cases human intruder tries to outmaneuver the defender by moving up and down. However, the defender successfully intercepts the intruder.

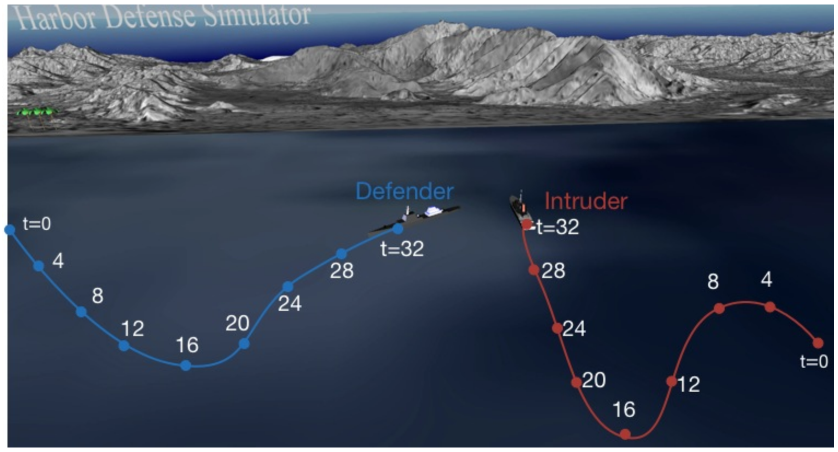
7.4 Implementation

7.4.1 Algorithm

Fig. 7.2 shows the evolution of the defender's trajectory as it iterates the Algorithm 4, given that the intruder is in upper-right position from the defender. Fig. 7.2 (a) to (d) shows the defender trajectories obtained from Step 1 in algorithm 4. Initially, defender roughly towards the intruder. As it iterates algorithm 4, trajectory converges (iteration 3 and 4). To facilitate the comparison of each figures,



(a) Simulation with HoTDeC dynamics



(b) Simulation with ship dynamics.

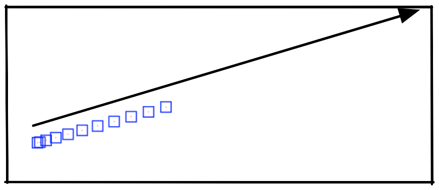
Figure 7.1: Two kinds of human-computer interactive real-time simulator: in both cases, the human player fails to reach to the harbor.

identical arrows indicating heading directions are inserted. Fig. 7.2(e) visualizes the evolution of the trajectories((a) to (d)) simultaneously with separated x and y coordinates.

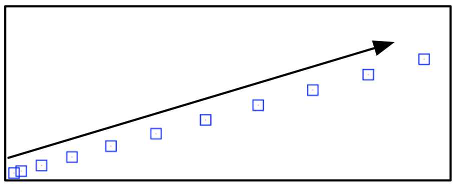
7.4.2 Experiment

We now present experiments involving an autonomous defender and a human intruder. The human intruder uses a joystick to activate the white-colored HoTDeC. We have deposited the videos and relevant material in the following repository: http://publish.illinois.edu/lee822.

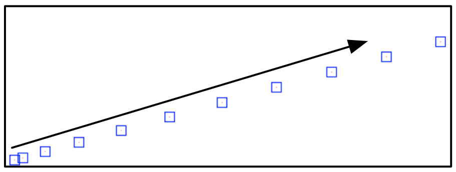
Fig. 7.3 is a static photo of the experiment between human intruder and au-



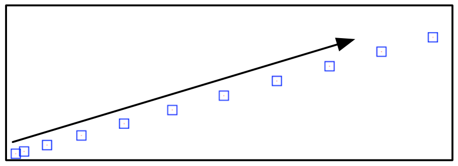
(a) Iteration 1.



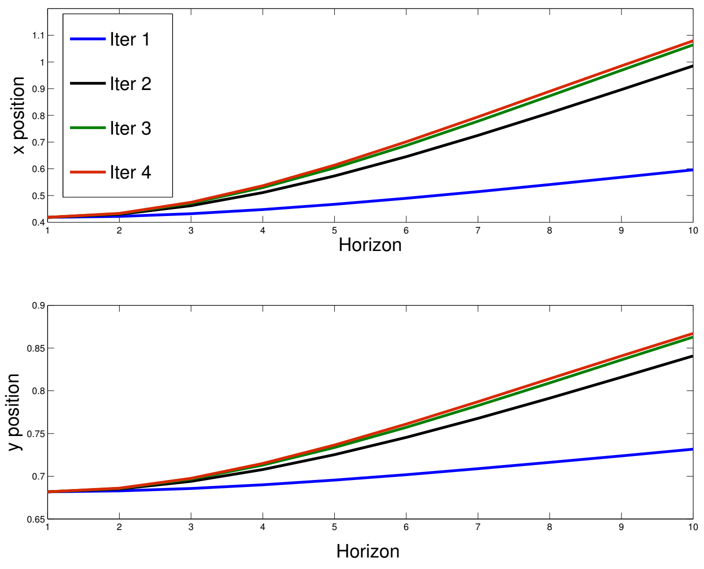
(b) Iteration 2.



(c) Iteration 3.



(d) Iteration 4.



(e) Positions of the defender in terms of horizon

Figure 7.2: Evolution of the defender trajectory in Outer approximation: as iterates the outer loop, the defender trajectory converges to the optimal trajectory. (a)-(d) is in $x - y$ plane, (e) is in position-N(horizon) plane.

tonomous defender. Trajectories are obtained from the vision server, shown in top left corner. Computing for the experiment and simulation is compared in Table.1. A laptop was used for the simulation. An ARM8(600MHz) processor is used in Gumstix. 'PH Iters' indicate the number of iterations in the outer approximation method and the PH algorithm. Computing time required per one sample time is 1.8 seconds in simulation and only 0.18 second in the experiment. As the longer horizon is beneficial to the optimality of the problem but it requires more com-

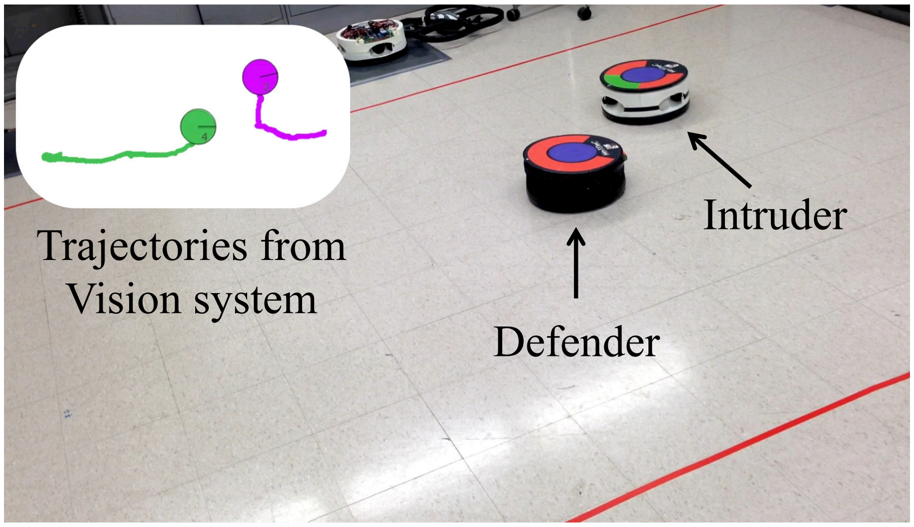


Figure 7.3: Static photo of experiment: the intruder is operated by human player and the defender is computer player

puting time, the horizon length 10Δ is chosen as it is the longest implementable horizon to our system.

Table 7.1: Computing Comparison

	Simulation	Experiment
Horizon	10Δ	10Δ
PH Iters	40	40
Solver	PH+Tomlab	PH+JOptimizer
Processor	Intel Core2 Duo 2.4GHz 4GB RAM	ARM8+DSP(TMSF28335) 600MHz 512MB RAM,flash,CMOS
Program Language	Matlab,VRML	Java,C
Comp. Time/OA Iter	0.6 sec	0.06 sec
Comp. Time/sample	1.8 sec	0.18 sec

Fig. 7.4 shows the trajectories of the intruder and the defender from the experiment. Elapsed times are labeled next to corresponding positions. The visualizer, written in Python, captures the trajectories of the HoTDeCs, their position and orientation along with their identification number in real-time. Blue and red paths are the trajectories generated in the experiment for the defender and intruder, respectively. We observe that the human intruder tried to trick the defender by moving down and up, but the defender successfully prevented the intruder from reaching the harbor located in the left side of the screen. We also observe that

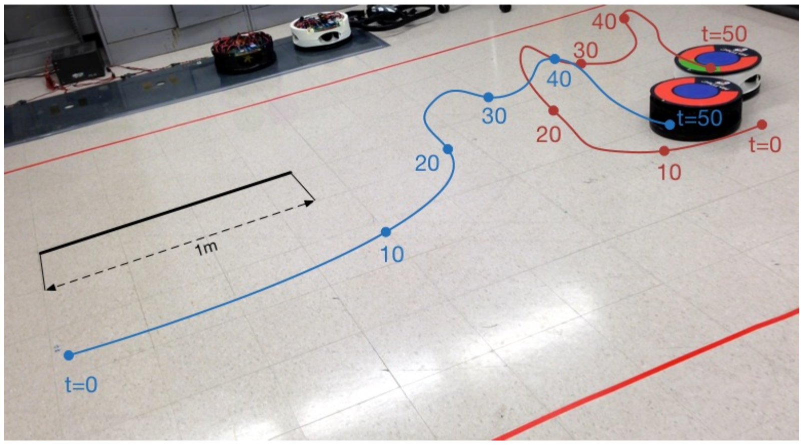


Figure 7.4: Trajectories from single-defender experiment: the running time of the experiment is about 50 seconds. Computer defender successfully prevents the human intruder from reaching to the left side of the lab.

defender traveled longer distance from $t = 0$ to $t = 10$ range than other ranges. Primary reason is that the defender requires less outer iterations in $t = 0$ to $t = 10$ range because its optimal path is not very different from its initialized trajectory (horizontal straight line). However, defender needs more outer iterations as the intruder changes direction after $t = 10$. When more computation time is required, defender maintain current position until iteration is finished (Step 4 in Algorithm.1) and obtain next reference. As human player is cautious, he does not operate the intruder with full speed. This result in shorter travel distance than defender's throughout the experiment.

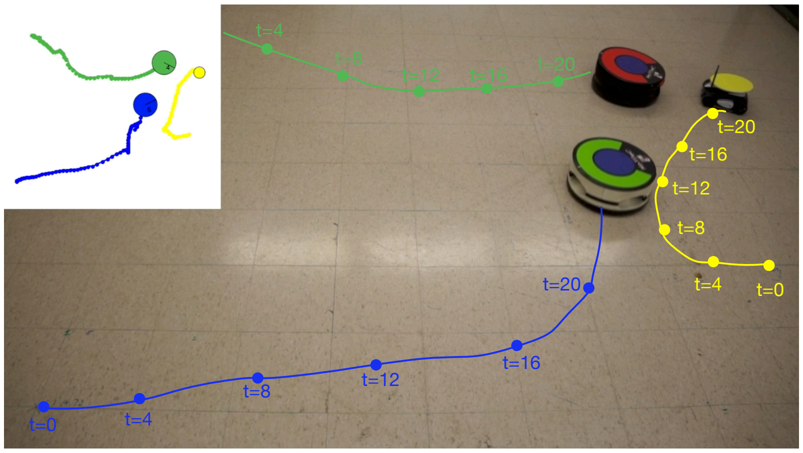


Figure 7.5: Trajectories from multi-defender experiment: the running time of the experiment is about 20 seconds. Team of the computer defenders successfully prevents the human intruder from reaching to the left side of the lab.

Fig. 7.4 shows the trajectories of the team of the defenders and human operated intruder. Human intruder tries to outmaneuver the white-colored defender by moving upward. However, realizing the black-colored defender approaches, he had to turn around and move backward. Throughout the experiments, defender team successfully cooperated to prevent the intruder from reaching to the left side of the lab.

7.5 Conclusion

A MPC scheme controller was successfully implemented in a real-time harbor defense scheme. Our approach was based on the newly developed implementation algorithm. Special optimization code was also developed to realize the algorithm. Our own abstract network layer enabled the effective communication between server and clients. We note that due to the physical constraint of the lab space, there were constraints on our ability to fully explore the effectiveness of the algorithm in all situations. Our previous results indicate that a narrow channel is advantageous for the defender [58].

CHAPTER 8

SEA EXPERIMENT: IMPLEMENTATION ON NAVAL SHIPS

8.1 Introduction

In order to demonstrate that our methodology results in solutions that are useful in real-world situations, we conducted three tests on the Chesapeake Bay in the vicinity of the United States Naval Academy. During these tests, Naval Academy Yard Patrol craft (YPs) served as the intruder and defender. The YPs were from the 676 class, and their hull numbers were 692 and 695. Fig. 8.1 provides a photograph of YP692 on the day of the test.

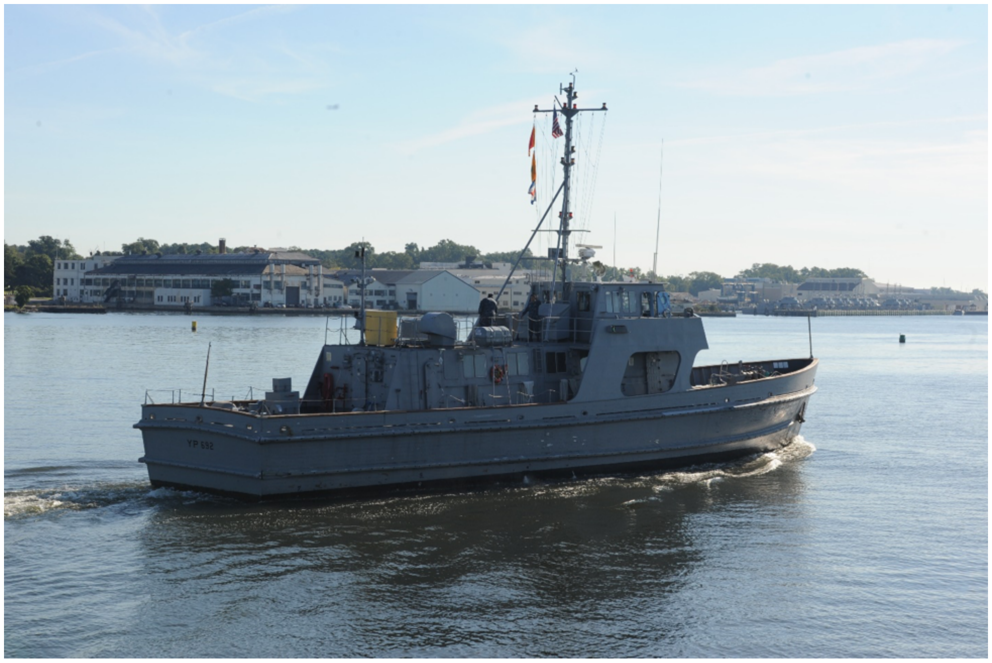


Figure 8.1: YP692 getting underway from the sea wall at the Naval Academy on the day of the test

The YPs are vessels normally used to train midshipmen in basic seamanship and navigation. The YPs are 108 feet long, have a beam of 23 feet 4 inches, a draft of 10 feet, and a fully loaded displacement of 172 tons. They have a maximum speed of 13.2 knots and a turning radius of approximately 150 feet. For all three tests, the intruder's heading was selected by the human craft master. The defender's heading was determined by the harbor defense algorithm in real-time and was relayed verbally to the human craft master as a course to steer. The harbor defense algorithm was implemented in Matlab R2015a with TOMLAB 7.9 running on a MacBook Pro with OS X Yosemite and a 2.3 GHz Intel Core i7 and 16GB of 1600 MHz DDR3 memory. The intruder program was implemented in Matlab R2015a running on a MacBook Air with OS X Yosemite and a 1.7 GHz Intel Core i5 and 4 GB of 1333 MHz DDR3 memory. To obtain the heading and position information needed by the algorithm, iPhone 6s were connected to the two laptops via universal serial bus (USB) cables. The phones provided heading and position information from their Global Positioning System (GPS) sensors to the laptops using the Matlab app (Version 5.2). The phones also shared their cellular internet connections with the laptops over the USB cables. In order for the data to be exchanged between the phones and the laptops, all of the devices had to appear as if they were on the same local area network. To establish this type of connection on the water, we connected all of the devices to the University of Illinois at Urbana-Champaign (UIUC) local network via the Cisco AnyConnect Virtual Private Network (VPN) client (version 4.0.03004 on the phones and version 3.1.10010 on the laptops). Fig. 8.2 provides a graphical depiction of the network configuration.

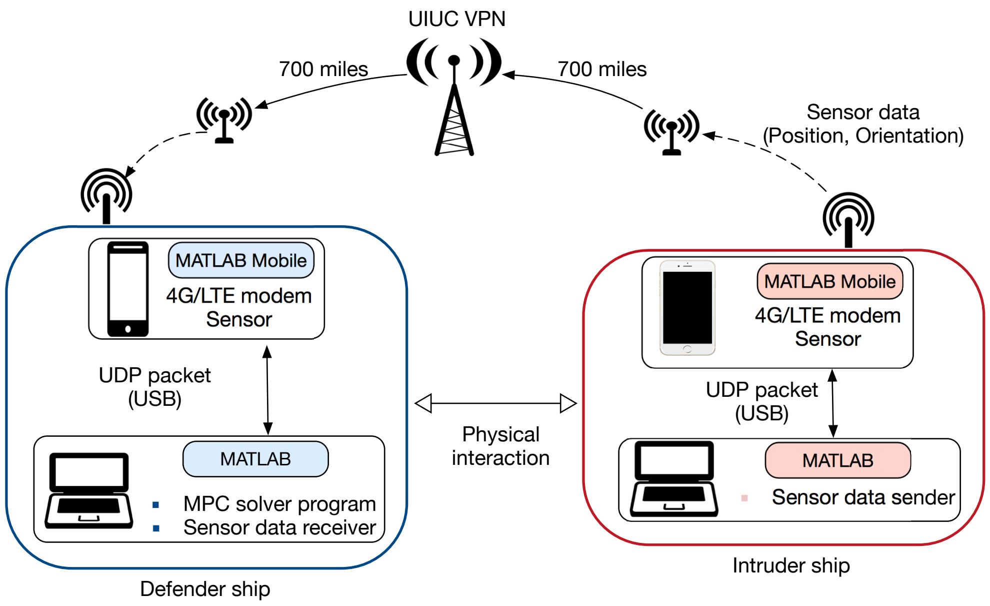


Figure 8.2: Schematic for the network setup

Once this network configuration was established, the Matlab program running on the intruder laptop would periodically send the sensor data (GPS heading and position) to the laptop on the defending vessel via user datagram protocol (UDP) packets. This data was then used by the MPC implementation of the harbor defense problem running on the defender laptop to provide an output command pair which included ship speed and heading angle. The command pair was relayed verbally to the craft master of the defending vessel for implementation every 10 to 15 seconds, depending on how significant the change was to the command pair. We conducted three experiments to simulate different possible intruder behavior. All of the experiments were run in the vicinity of the naval anchorage located near 38Ph.D.57' N 076Ph.D.26' W (see Fig. 8.3 below or NOAA Chart 12282 available from http://www.charts.noaa.gov/OnLineViewer/12282.shtml). The simulated harbor entrance for the experiments was an imaginary line from day marker "A" at the top left corner of the naval anchorage to red nun buoy "2" at the bottom right. The starting position for the intruder was near green can buoy "1" located just north of the naval anchorage.

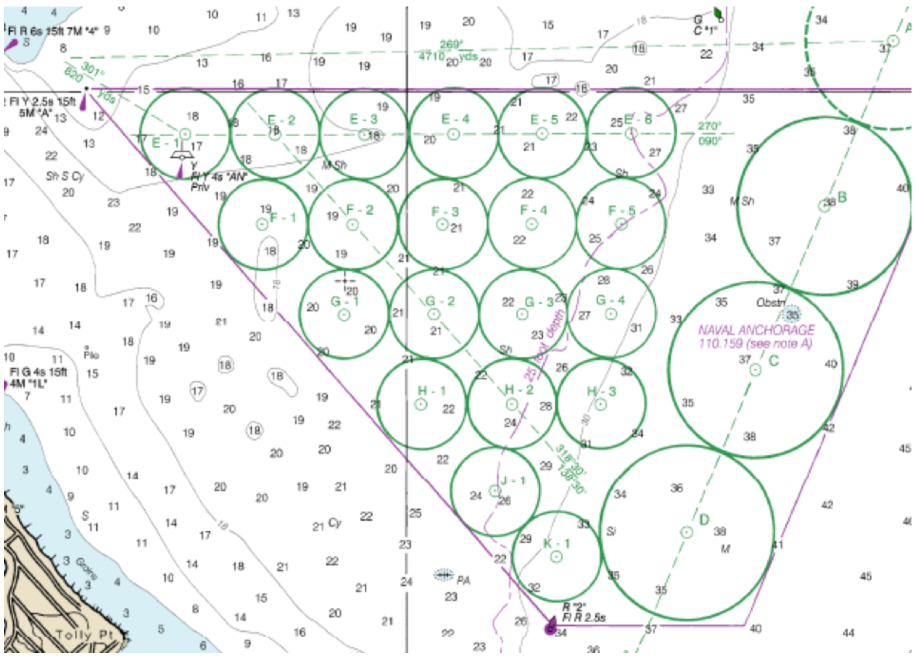


Figure 8.3: NOAA Chart 12282

8.2 Experiment 1

For this experiment, the maximum speed of both the intruder and defender was set to 5 knots. Fig. 8.4 shows the trajectories of the vessels for this experiment overlaid on imagery from Google Earth. Blue and red dots indicate the initial positions of the defender and intruder, respectively. The blue and red curves depict the trajectories for the two YP's, while the black dashed curves with arrows show the direction of movement of the vessels.

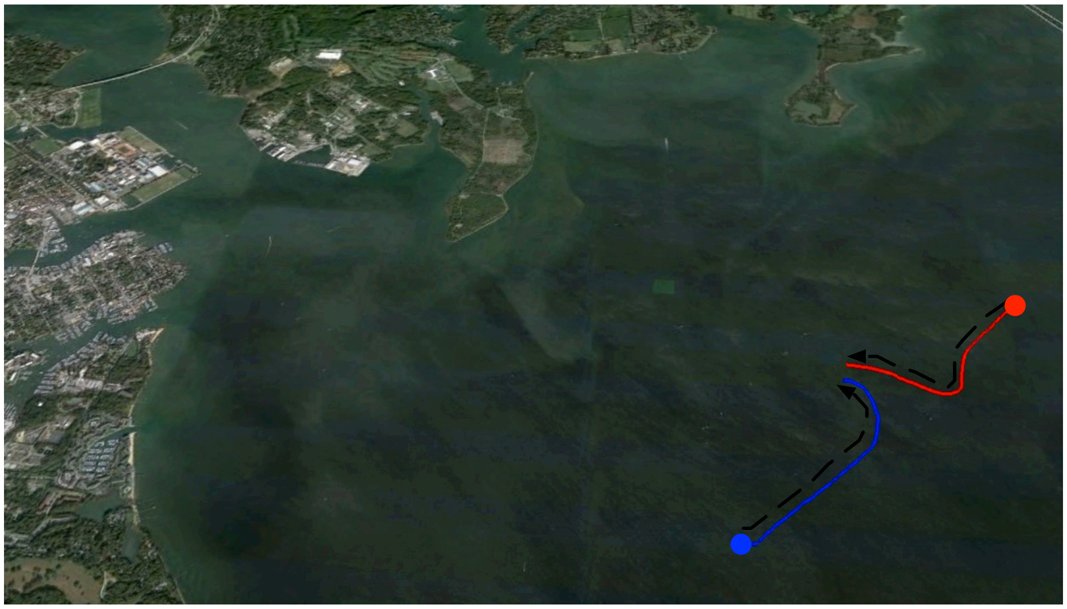
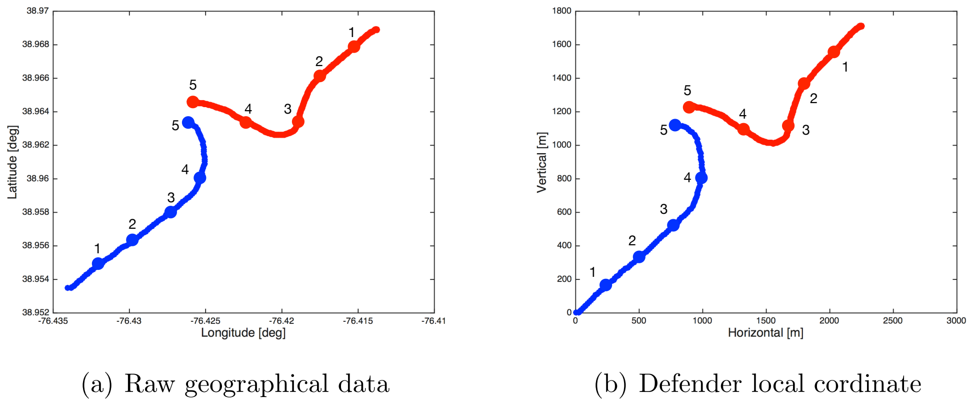


Figure 8.4: Trajectories on the Google Earth 1

Fig. 8.5 provides more detailed trajectories for the YPs. Fig. 8.5(a) shows the raw geographical data obtained from the GPS, while Fig. 8.5 (b) shows the trajectories in the local Cartesian coordinate system. The local Cartesian coordinate system was used to facilitate numerical



(a) Raw geographical data

(b) Defender local cordinate

Figure 8.5: Detailed trajectories for the experiment 1

computations to solve the defender's MPC problem. In the defender's local coordinate system, the origin is defined to be the defender's initial position and the horizontal and vertical positions are approximated as the arc length on the earth's surface derived from the changes in latitude and longitude. For the first experiment, the defender's initial heading is parallel to 38.9538 degrees north latitude, and therefore the trajectories in Fig. 8.5 (a) and (b) are qualitatively the same.

8.3 Experiment 2

For this experiment, the maximum speed of both the intruder and defender was set to 6 knots. The intruder attempted to outmaneuver the defender by performing a 360 degree turn. This large turn resulted in the speed of the intruder being slightly less than its maximum speed. Fig. 8.6 shows the trajectories of the vessels for this experiment overlaid on imagery from Google Earth. Blue and red dots again indicate the initial positions of the defender and intruder, respectively. As before, the blue and red curves depict the trajectories for the two YP's, while the black dashed curves with arrows show the direction of movement of the vessels.

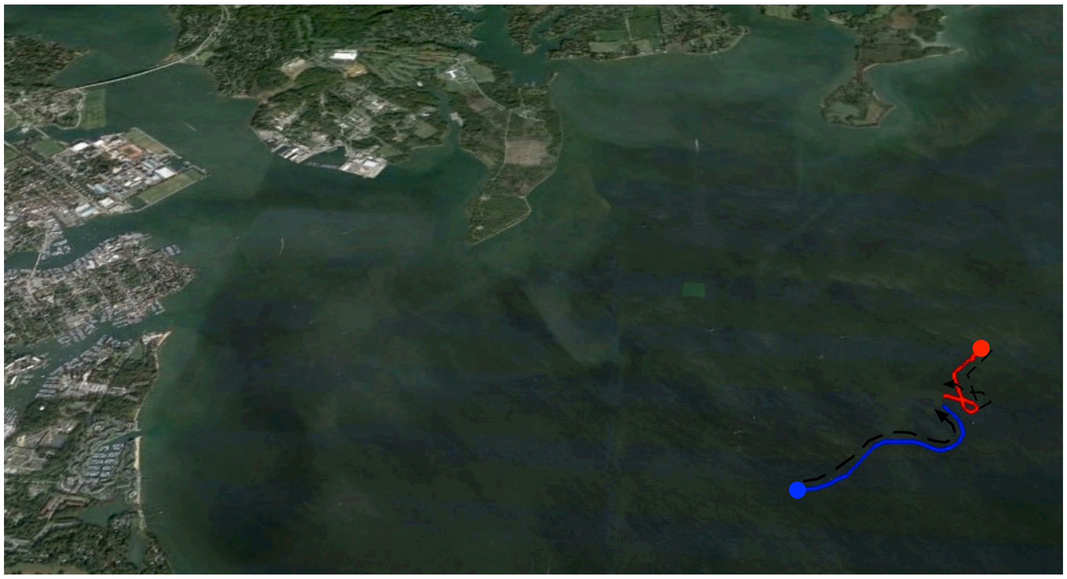
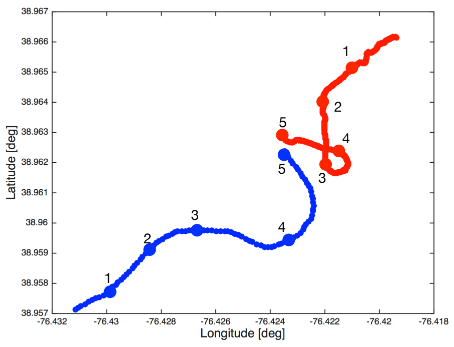


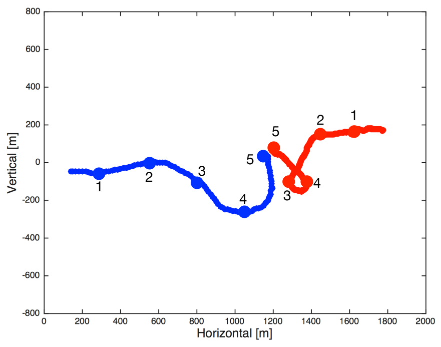
Figure 8.6: Trajectories on the Google Earth 2

Detailed trajectories in both raw geographical coordination and the local cartesian coordination is presented in Fig. 8.7.

Detailed trajectories are given in Fig. 8.7 for both the raw geographic coordinate system and the local Cartesian coordinate system. For this experiment, the initial heading of the defender is not parallel to a line of latitude. Hence the defender's local Cartesian coordinate system is rotated about 43 degrees clockwise.



(a) Raw geographical data



(b) Defender local coordinate

Figure 8.7: Detailed trajectories for the experiment 2

8.4 Experiment 3

For this experiment, the maximum speed of both the intruder and defender was set to 7 knots. The intruder attempted to outmaneuver the defender by heading directly for the simulated harbor entrance. Fig. 8.8 shows the trajectories of the vessels for this experiment overlaid on imagery from Google Earth. Blue and red dots again indicate the initial positions of the defender and intruder, respectively. As before, the blue and red curves depict the trajectories for the two YP's, while the black dashed curves with arrows show the direction of movement of the vessels.

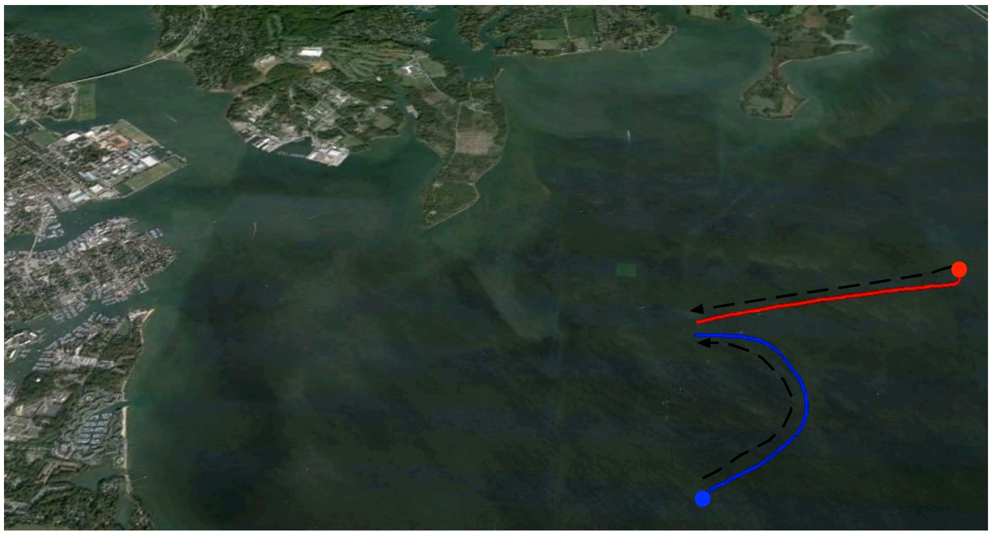
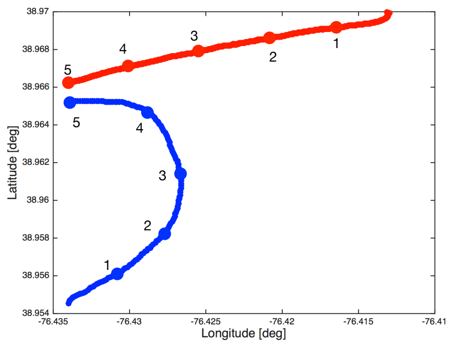
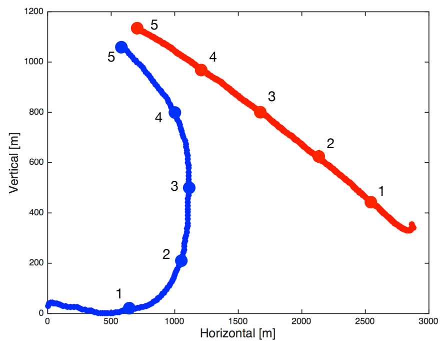


Figure 8.8: Trajectories on the Google Earth 3

Detailed trajectories are given in Fig. 8.9 for both the raw geographic coordinate system and the local Cartesian coordinate system. For this experiment, the initial heading of the defender is not parallel to a line of latitude. Hence the defender's



(a) Raw geographical data



(b) Defender local coordinate

Figure 8.9: Detailed trajectories for the experiment 3

local Cartesian coordinate system is rotated about 45 degrees clockwise.

8.5 Conclusion

We have demonstrated that our minmax MPC solution is effectively defend the harbor from the approaching intruder. The computation speed was fast enough even if the implementation language was not a low level language. Matlab environment approximately works up to 1 sec sample time, which corresponds to the about 10 knots. For a fast moving ship, it is encouraged to use low level implementation such as Java or C, as we have implemented in the lab scale experiment. We have also showed that the network setup via UDP packet works reliably through the 4G/LTE network connected to the University of Illinois VPN (Virtual Private Network). This allowed the intruder and the defender ship to be considered in the same local network and hence provide convenient message exchange.

CHAPTER 9

MOBILE NETWORK JAMMING PROBLEM

9.1 Introduction

This problem is addressed in [75]. HJI (Hamilton-Jacobi-Isaac) equation based approach results in very expensive computational load. The goal of this section is to solve this problem with the more realistic dynamic model, in a significantly shorter time.

Consider the situation with two UAVs (U_1 and U_2) and one jammer (U_J). Let $D_k^{1,2}$, $D_k^{1,J}$ and $D_k^{2,J}$ be a Euclidean distance between U_1 and U_2, U_1 and U_J, and between U_2 and U_J at time k. If $\eta \min[D_k^{1,J}, D_k^{2,J}] \leq D_k^{1,2}$ for some positive constant η, then the communication channel between U_1 and U_2 is considered to be jammed. In other words, communication is jammed if the U_J is closer to the UAV than the other UAV. Consider a situation in which U_1 and U_2 are initially jammed in the presence of a jammer U_J. The objective of the jammer is to maximize the time for which it can jam the communication between U_1 and U_2. The objective of U_1 and U_2 is to minimize the time for which communication remains jammed. The game terminates at the first instant at which U_1 and U_2 are in a position to communicate. We assume that U_1 and U_2 has a complete knowledge about the state of the system.

Original problem formulation is a time optimization problem, which one can expect bang-bang type of optimal control. The formulation is the form of zero sum game. Approach to the solution is to seek a saddle point using the HJI equation, which is computationally very expensive and hence far from the real-time application. Original problem does not include minimum separation between U_1 and U_2. Lastly, the speed of UVAs are fixed in the original problem, and control is a yaw rate.

Consider a situation in which U_1 and U_2 are initially jammed in the presence of a jammer U_J. The objective of the jammer is to maximize the time for which it can jam the communication between U_1 and U_2. The objective of U_1 and U_2 is to minimize the time for which communication remains jammed. Also, U_1 and

U_2 need to maintain the minimum separation. The dynamic model of UAVs and the jammer is nonlinear with two inputs: thrust and yaw rate is used. We assume that U_1 and U_2 has a complete knowledge about the state of the system. Also, we assume U_1 and U_2 can move only inside of the constrained space that is modeled as a rectangular shaped space.

9.2 Nomenclature

Δ is a sampling time.
H is a length of the planning horizon.
$\bar{v}_u$ and $\bar{v}_J$ are maximum speed of UVAs and jammer, respectively.
$\underline{v}_u$ and $\underline{v}_J$ are minimum speed of UVAs and jammer, respectively.
$\bar{\omega}_u$ and $\bar{\omega}_J$ are maximum yaw rate inputs of UVAs and jammer, respectively.
$\bar{a}_u$ and $\bar{a}_J$ are maximum acceleration inputs of UVAs and jammer, respectively.
$zu_{1,k} = [x_{1,k}, y_{1,k}]^T$ be a position of the UAV1.
$zu_{2,k} = [x_{2,k}, y_{2,k}]^T$ be a position of the UAV2.
$zu_k = [zu_{1,k}^T, zu_{2,k}^T]^T$ be a position of the UAV1 and UAV2.
$zj_k = [x_{J,k}, y_{J,k}]^T$ be a position of the jammer.
σ_u is a minimum separation between UAVs.
σ_J is a minimum separation between UAVs and jammer.

9.3 Problem formulation

The kinematic model for both jammer and UAVs are described as the following unicycle model.

$$\begin{aligned}\dot{x}(t) &= v\cos\theta(t)\\ \dot{y}(t) &= v\sin\theta(t)\\ \dot{\theta}(t) &= \omega(t)\end{aligned} \tag{9.1}$$

We can integrate the differential equation (9.1) for $t \in [k\Delta, (k+1)\Delta)$, to obtain the difference equations:

$$\begin{aligned}x_{k+1} &= x_k + \Delta v\cos\theta_k + \Delta^2 a_k \cos\theta_k\\ y_{k+1} &= y_k + \Delta v\sin\theta_k + \Delta^2 a_k \sin\theta_k\\ \theta_{k+1} &= \theta_k + \Delta\omega_k\end{aligned} \tag{9.2}$$

where $x_k = x(k\Delta)$, $y_k = y(k\Delta)$, $\theta_k = \theta(k\Delta)$, and $\omega_k = \omega(k\Delta)$. The discretized model is from exact integration of (9.1). Detail is presented in Appendix A.

Following describes the MPC formulation for the mobile network jamming problem at a given time t and given current positions of UAVs and jammer.

$$\begin{aligned} &\min_{u\in U} \max_{\substack{v\in Y(u_k)\\ k\in\mathbf{H}}} \left\{ D_k^{1,2} - \min[D_k^{1,J}, D_k^{2,J}] \right\} \\ s.t.\quad & D_k^{1,2} \geq \sigma_u, \\ & \underline{x} \leq x_k \leq \bar{x}, \quad \underline{y} \leq y_k \leq y, \textit{for all players}, \\ & \textit{Dynamics } (9.2). \end{aligned} \tag{9.3}$$

where $\mathbf{H} = \{t+1, t+2, \ldots, t+H\}$.

Sequence of augmented control space for the UAV is

$$U = \left\{ \begin{bmatrix} \omega_{1,1}, \cdots, \omega_{1,H} \\ \omega_{2,1}, \cdots, \omega_{2,H} \end{bmatrix} \in R^{2\times H} \mid -\bar{\omega}_u \leq \omega_{i,k} \leq \bar{\omega}_u, \ \forall k \in \mathbf{H}, \quad i = 1,2 \right\}. \tag{9.4}$$

Control constraint set for the jammer is

$$v = \left\{ \begin{bmatrix} \omega_{J,1}, \cdots, \omega_{J,H} \end{bmatrix} \in Y(u_k) \mid -\bar{\omega}_J \leq \omega_{J,k} \leq \bar{\omega}_J, \ \forall k \in \mathbf{H} \right\}. \tag{9.5}$$

, where

$$Y(u_k) := \left\{ \begin{bmatrix} \omega_{J,1}, \cdots, \omega_{J,H} \end{bmatrix} \in R^H \mid \min[D_k^{1,J}, D_k^{2,J}] \geq \sigma_J, k \in \mathbf{H} \right\}. \tag{9.6}$$

We observe that at some k, if the value of (9.3) is negative, network is established and hence UAVs tries to decrease the value. However, the jamming occurs when the value if positive and the jammer tries to increase the value.

9.4 Formulation conditioning

1. $D_k^{1,2} - \min[D_k^{1,J}, D_k^{2,J}]$ is replaced by $(D_k^{1,2})^2 - (\min[D_k^{1,J}, D_k^{2,J}])^2$. This facilitates the gradient conditioning as quadratic function is generally a favorable form in numerical optimization.
2. Since $\max_{k\in\mathbf{H}}\{(D_k^{1,2})^2 - (\min[D_k^{1,J}, D_k^{2,J}])^2\}$ is not smooth, and by the fact that

maximum of set of scalar is equivalent to the maximum of their convex hull,

$$\begin{aligned}
&\max_{k\in\mathbf{H}}\{(D_k^{1,2})^2-(\min[D_k^{1,J},D_k^{2,J}])^2\}=\max_{k\in\mathbf{H}}\{(D_k^{1,2})^2-(-\max[-D_k^{1,J},-D_k^{2,J}])^2\}\\
&=\max_{\mu\in\Sigma}\sum_{k=1}^{H}\mu_k\{(D_k^{1,2})^2-(\max_{\nu\in[0,1]}[(\nu-1)D_k^{2,J}-\nu D_k^{1,J}])^2\},\\
&=\max_{\mu^+\in\Sigma^+}\sum_{k=1}^{H}\mu_k^+\{(D_k^{1,2})^2-[(\nu-1)D_k^{2,J}-\nu D_k^{1,J}])^2\},
\end{aligned} \tag{9.7}$$

where $\Sigma^+=[\nu,\Sigma]^T$. Hence, (9.3) becomes the following.

$$\begin{aligned}
&\min_{u\in U}\max_{v'\in Y'(u_k)}\sum_{k=1}^{H}\mu_k^+\{(D_k^{1,2})^2-[(\nu-1)D_k^{2,J}-\nu D_k^{1,J}])^2\}\\
s.t.\quad &D_k^{1,2}\geq\sigma_u,\\
&\underline{x}\leq x_k\leq\bar{x},\quad \underline{y}\leq y_k\leq y, \textit{for all players},\\
&\textit{Dynamics } (9.2).
\end{aligned} \tag{9.8}$$

where $\mathbf{H}=\{1,2,\dots,H\}$, and U as in (9.4).

$$Y'(u_k):=\left\{\begin{bmatrix}\omega_{J,1},\cdots,\omega_{J,H}\\ \mu_1^+,\cdots,\mu_H^+\end{bmatrix}\in R^H\times\Sigma^+\mid\min[D_k^{1,J},D_k^{2,J}]\geq\sigma_J,k\in H\right\}. \tag{9.9}$$

9.5 Simulation results

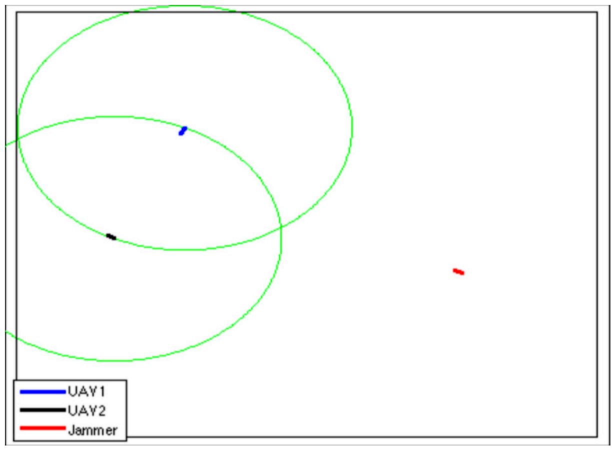


Figure 9.1: Experiment set up.

Fig 9.1 show a set up for the simulation. UAV 1 and 2 are colored blue and black, respectively. Jammer is colored red. There are two circles with a radius of $D_k^{1,2}$, and a center as current position of UAVs. When the network is established, The two circle indicates the area of network. If the jammer is located inside of any circles, the network is jammed and the color of circles turn to red.

Various simulation videos are deposited in http://publish.illinois.edu/lee822 for the various initial positions.

9.6 Necessary and Sufficient condition

In this section, we are interested in finding necessary and sufficient condition for the jamming to occur.

9.6.1 Sufficient condition

First, we find sufficient condition for the jamming. At a discrete time t, reachable sets for an UAV ($R_{u,t}$) and the jammer ($R_{J,t}$) are defined in their own coordinate (see [48]) as follows.

$$R_{u,t} := \left\{ \begin{bmatrix} x_u \\ y_u \end{bmatrix} \mid x_u = sgn(u) r_u(u)(1-\cos ut), \quad y_u = r_u(u)|\sin(ut)|, -\bar{\omega}_u \le u \le \bar{\omega}_u \right\}$$
$$R_{J,t} := \left\{ \begin{bmatrix} x_J \\ y_J \end{bmatrix} \mid x_J = sgn(v) r_J(v)(1-\cos vt), \quad y_J = r_J(v)|\sin(vt)|, -\bar{\omega}_J \le v \le \bar{\omega}_J \right\}, \tag{9.10}$$

where $r_u(u)$ is the radius of turning ($r_u(u) = \frac{v_u}{|u|}$), $0 \le t \le \frac{\pi}{2}\min[\frac{1}{\bar{w}_u}, \frac{1}{\bar{w}_J}]$. It is assumed that control is constant for $[0, t]$. This is particularly reasonable because of the space constraint for both UAV and the jammer. As they located close to the limit of x and y, they need nonzero control inputs. By the definition of the $R_{J,t}$, there exists a unique mapping from the origin, O, to any $p \in R_{J,t}$. Also from the definition of the reduced space ([48]), initial headings of UAV and jammer are both $\frac{\pi}{2}$,*i.e.*, coincide with positive y axis. Therefore, tuple $(O, p, \frac{\pi}{2})$ exists for any $p \in R_{J,t}$. Obviously, tuple $(p, O, -\frac{\pi}{2})$ also exists for any $p \in R_{J,t}$.

Let us define transformed reachable set of the jammer, $t_{t,\Phi}(r)$ which is constructed by rotating the jammer coordinate by Φ and shift the origin to $r \in R_{u,t}$.

$$t_{t,\Phi}(r) = T_\Phi R_{J,t} + r, \quad r \in R_{u,t}, \tag{9.11}$$

where T_Φ is a 2×2 rotational matrix. Then by the definition of $t_{t,\Phi}(r)$,for any given r, there exists triplet $(r, q, \frac{\pi}{2} + \Phi)$, $\forall q \in t_{t,\Phi}(r)$. Then again obviously, there exists tuple $(q, r, -\frac{\pi}{2} + \Phi)$, $\forall q \in t_{t,\Phi}(r)$.

Suppose there exists nonempty set $\zeta_{t,\Phi} = \bigcap_{r \in R_{u,t}} t_{t,\Phi}(r)$. Then any $\eta \in \zeta_{t,\Phi}$ guarantees the existence of tuple $(\eta, r, -\frac{\pi}{2} + \Phi)$, $\forall r$. This means η is a jamming point without considering the radius of jamming.

Now, let us consider the jammer with a radius of jamming $c = \|UAV_1 - UAV_2\|$. Transformed reachable set is defined as the jamming radius is introduced. For a given $r \in R_{u,t}$,

$$TR_{t,\Phi}(r) := \{z \mid \|(T_\Phi R_{J,t} + r) - z\| \leq c\} \tag{9.12}$$

Then at time t, the jamming region, JR_t is obtained as

$$JR_{t,\Phi} = \left\{ \eta \mid \eta \in \bigcap_{\forall r \in R_{u,t}} TR_{t,\Phi}(r) \right\} \tag{9.13}$$

At time t, if the jammer is located inside of JR_t with a heading angle $\Phi - \frac{\pi}{2}$, jamming occurs.

9.6.2 Necessary condition

Minimum requirement for the jamming to be occur is the existence of nonempty set

$$TR_{t,\Phi}(r_1) \bigcap TR_{t,\Phi}(r_2), \tag{9.14}$$

where any two different $r_1, r_2 \in R_{u,t}$.

9.6.3 Necessary and sufficient condition

Necessary condition is the existence of at least one element of intersection of $TR_{t,\Phi}(r)$ for two different r. Sufficient condition is the existence of element of intersection of $TR_{t,\Phi}(r)$ for all r. Hence, the necessary and sufficient condition is to find an element such that if $\eta \in \{TR_{t,\Phi}(r_1) \bigcap TR_{t,\Phi}(r_2)\}$ for some $r_1, r_2 \in R_{u,t}$ then $\eta \in JR_t$.

First, let us define extreme points for both UAV and jammer. Extreme points are obtained by applying upper and lower bound of control. For the UAV,

$$\begin{aligned} r_l &= [r_u(u)(\cos\bar{\omega}_u t - 1), r_u(u)|\sin\bar{\omega}_u t|] \\ r_r &= [r_u(u)(1 - \cos\bar{\omega}_u t), r_u(u)|\sin\bar{\omega}_u t|]. \end{aligned} \tag{9.15}$$

Similarly, for the jammer,

$$\begin{aligned} p_l &= [r_v(v)(\cos\bar{\omega}_v t - 1), r_v(v)|\sin\bar{\omega}_v t|], \\ p_r &= [r_v(v)(1 - \cos\bar{\omega}_v t), r_v(v)|\sin\bar{\omega}_v t|]. \end{aligned} \tag{9.16}$$

Now, let us consider the intersection $TR_{t,\Phi}(r_1)\bigcap TR_{t,\Phi}(r_2)$ by fixing $r_1 = r_l$, and varying r_2. We can observe that if $r_l \approx r_2$, intersection is trivially nonempty. As r_2 gets further from r_l and closer to r_r, the intersection becomes smaller. Hence, the smallest set is obtained from the following intersection.

$$T_\Phi P_r + r_l \bigcap T_\Phi P_l + r_r, \tag{9.17}$$

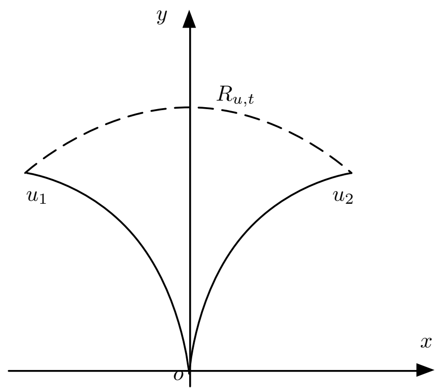
where $P_r = \{z \mid \|z - p_r\| \leq c\}$, $P_l = \{z \mid \|z - p_l\| \leq c\}$. The necessary and sufficient condition is the existence of η such that

$$\eta \in \{T_\Phi P_r + r_l \bigcap T_\Phi P_l + r_r\}, \tag{9.18}$$

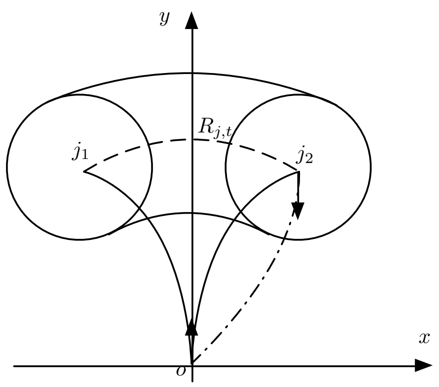
where $P_r = \{z \mid \|z - p_r\| \leq c\}$, $P_l = \{z \mid \|z - p_l\| \leq c\}$.

9.6.4 Geometric interpretation

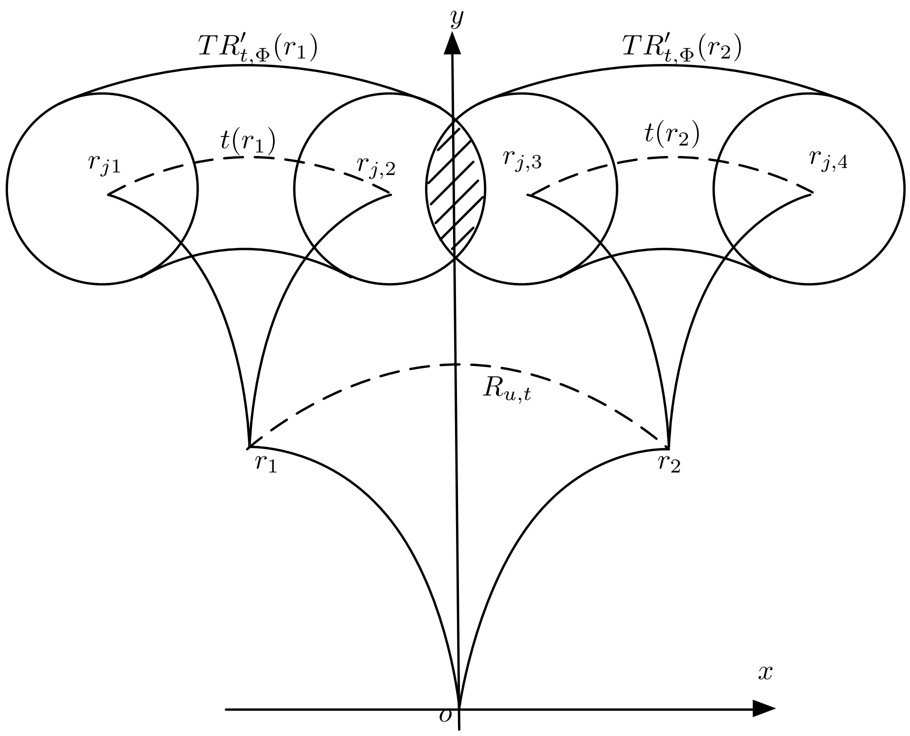
In this section, we present the example of geometric jamming regions. The Fig. 9.2 shows the example when $\Phi = 0$. The dotted arc in Fig. 9.2(a) and (b) are reachable sets of an UAV and the jammer, respectively. The circle in 9.2(b) indicates the jamming radius c. 9.2(c) presents the transformed reachable sets of the jammer, $TR_{t,\Phi}(\cdot)$. The shaded area is the intersection of $TR_{t,\Phi}(r_1)$ and $TR_{t,\Phi}(r_2)$. The Fig. 9.3 presents an example when $\Phi = \pi$.



(a) UAV reachable set (dashed arc)



(b) Jammer reachable set (dashed arc)



(c) Transformed reachable set

Figure 9.2: Jamming region($\Phi = 0$).

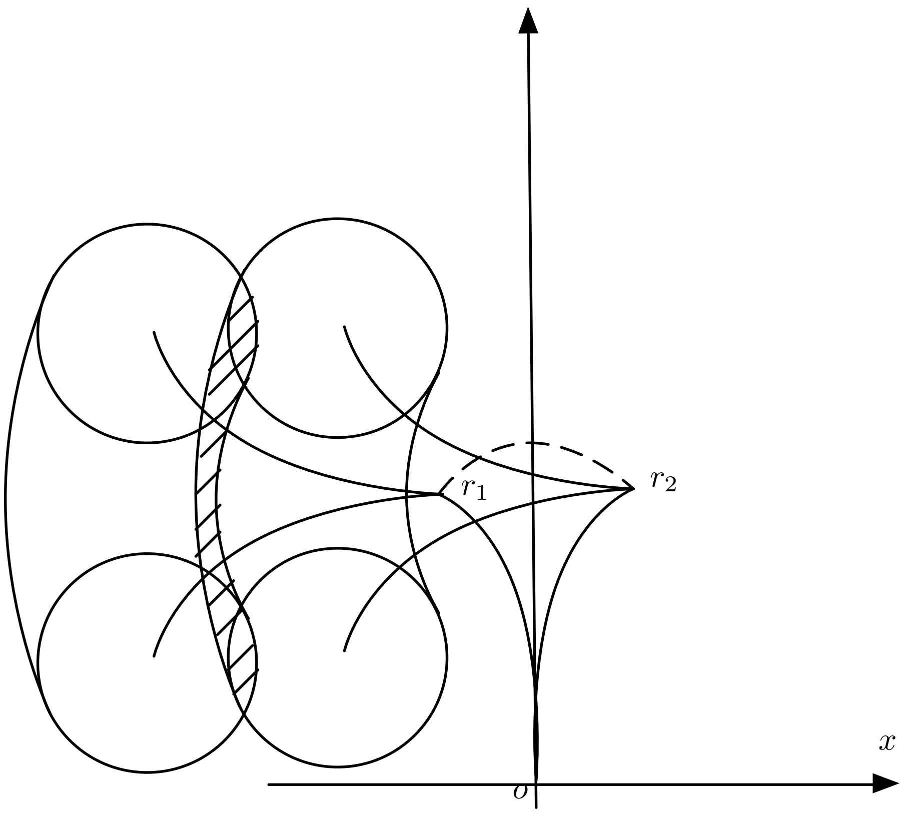
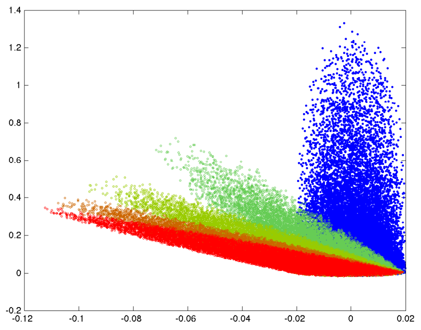


Figure 9.3: Example of the jamming regions with $\Phi = \pi$.

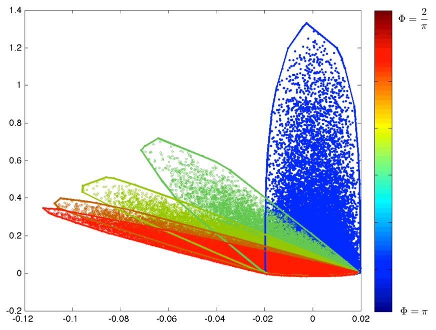
9.7 Visualization

In this section, the jamming region is visualized for several different Φ and radius of jamming c. The visualization is based on the Monte-Carlos method.

The number of 10^6 points are randomly scattered in a plane and evaluated if the point satisfies the condition (9.18).



(a) Monte-Carlos method



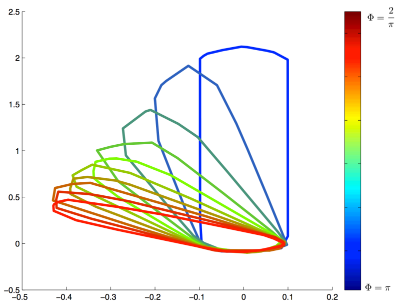
(b) Convex hull of (a)

Figure 9.4: Example of the jamming regions: Monte Carlo method.

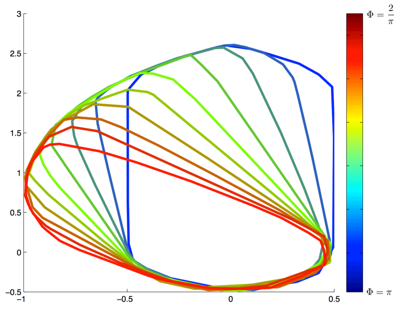
Fig. 9.4 shows an example of the visualization. Fig. 9.4 (a) is the result of Monte-Carlos method and Fig. 9.4 (b) is a convex hull of each jamming regions.

Fig. 9.5 shows jamming regions for different values of capture radius $c =$ 0.1,0.5,and 1. We can observe that as the capture radius grow, the overall jamming region grows as well.

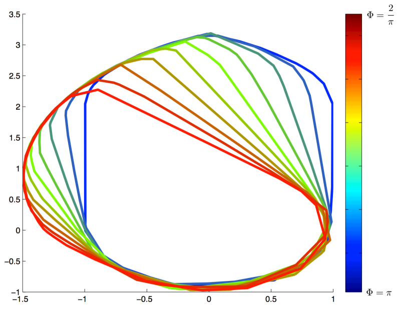
Fig. 9.6 and Fig. 9.7 are two different view of the comparison of the jamming regions with a different radius of capture. We see that the region is monotonically grows to the radius of capture.



(a) $c = 0.1$



(b) $c = 0.5$



(c) $c = 1$

Figure 9.5: Example of the jamming regions: convex hulls.

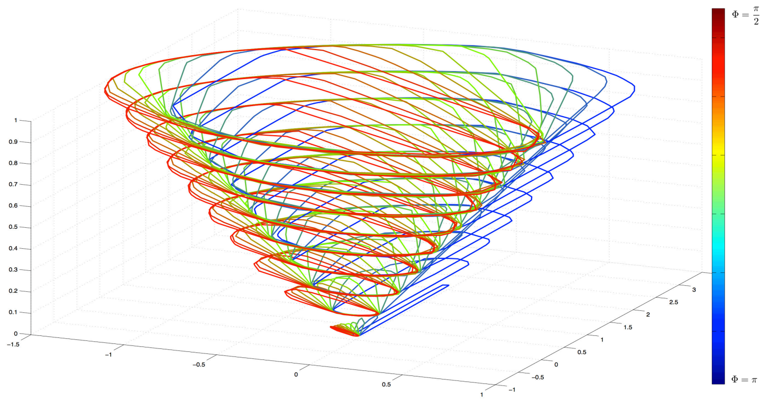


Figure 9.6: Jamming region view 1

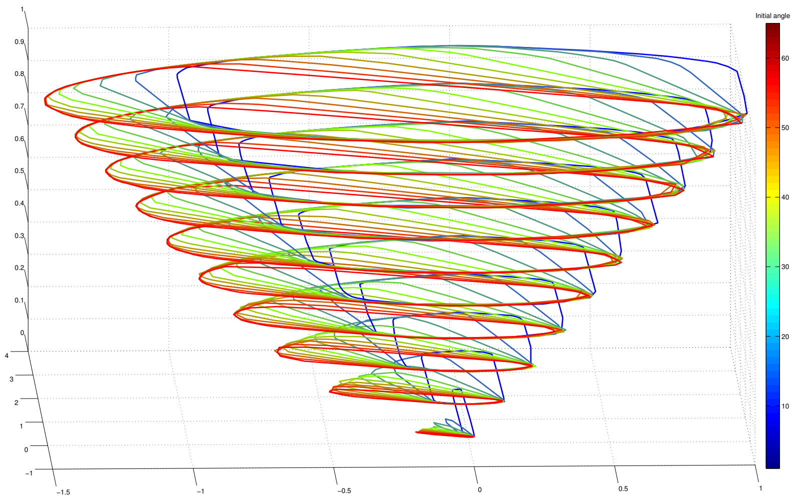


Figure 9.7: Jamming region view 2

CHAPTER 10

METHOD OF OUTER APPROXIMATIONS AND ADAPTIVE APPROXIMATIONS FOR A CLASS OF MATRIX GAMES

10.1 Introduction

Very large matrix games of the form $\min_{x\in X}\max_{y\in Y} A(x,y)$, where the elements $A_{i,j}(x,y)$ of the matrix A are functions of variables x,y, arise naturally in the formulation of pursuit-evasion games (see [48]) involving unmanned aerial vehicles (UAVs) commonly referred to as drones, as well as in pursuit evasion games involving unmanned surface or underwater ships, in which the controls are restricted to discrete values. In these problems the matrix A is often larger than the $3^{16} \times 3^{16}$, as in our example to be presented later. Another area in which very large matrix games arise is global solution of min-max problems of the form $\min_{x\in X}\max_{y\in Y} \phi(x,y)$, with $\phi(x,y)$ continuous and x,y one dimensional, and a solution of specified accuracy is required. As we will see in our examples, a straightforward approach to the solution of such min-max problems requires an inordinate amount of computing time, making it unusable in real time applications. Since the common factor of all such problems is that they can be viewed as discretization of continuous min-max problems, we explore in this paper the possibility of using tools that are an adaptation of tools used for the continuous min-max case: the construction of consistent approximations and the Method of Outer Approximations (see Secs. 3.3 and 3.4.5 in [20]). The result is a scheme which combines sequential precision refinement with an adaptation of the classical Method of Outer Approximations. The scheme computes fast enough for use in real time applications, such as navigation in pursuit-evasion situations.

We were unable to find much in the literature that addresses matrix form min-max problems and could serve as an alternative to the algorithm that we present. One possibility would be to transform the matrix min-max problem into a decision tree and use some form of the $\alpha-\beta$ algorithm [76], such as *SCOUT* [77], *Principal variation* [78], or *MTD-f* [79]. Unfortunately, when a matrix game is converted to a decision tree, we find that the depth of the tree is one, and hence pruning offers no advantages.

Another possibility would be to apply the PSO (Particle Swarm Optimization) technique [80], which is intended for solving the finite min-max problems. PSO is a population based stochastic optimization technique that requires the construction of large populations and is not proven to always converge to a solution. Hence, it is not really a competitor to the algorithm proposed in this paper.

A global optimization algorithm for solving semi-infinite optimization problems is presented in [81]. It shares with us the use of a version of outer approximations and complements it with a branch and bound technique. Since a min-max problem can be converted into a optimization problem by the addition of a variable, the algorithm described in [81], can be used for solving the second set of example problems in our paper. However, it does not appear to be competitive on the narrow class of examples that we consider, since, to quote from the conclusions in [81], "the main drawback of the method, as it stands, is the rapid increase in the size of the lower and upper-bounding problems as nodes of increasing depth are visited by the B&B procedure", a problem from which our algorithm does not suffer.

Consequently, our experimental results compare the computing times obtained using our method with those obtained using a basic algorithm that requires the evaluation of all the elements of the A matrix. As we will see from our examples, the computational time reductions with respect to any min-max algorithm, which requires the evaluation of all the elements $A_{i,j}(x,y)$ of the game matrix, become spectacular as the size of the problems increases.

10.2 Matrix Method of Outer Approximations

The method of outer approximations (see Sec. 3.4.5 in [20]) is a standard technique for solving *continuous* nonlinear programming problems with semi-infinite constraints and semi-infinite min-max problems. For semi-infinite min-max problems of the form

$$\min_{x \in X} \max_{y \in Y} \phi(x,y) \tag{10.1}$$

where $\phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is continuous and the sets $X \subset \mathbb{R}^n, Y \subset \mathbb{R}^m$ are compact, the conceptual method of outer approximations [1] has the following form (see [20] page 436):

[1]As stated, this method is conceptual because it requires global optimization evaluations.

Conceptual Method of Outer Approximations.

Data: $x_0 \in \mathbb{R}^n$.

Step 0: *Set* $i = 0$*, compute*

$$y_0 \in \arg\max_{y \in Y} \phi(x_0, y), \tag{10.2}$$

and set $Y_0 = \{y_0\}$.

Step 1: *Compute*

$$x_{i+1} = \arg\min_{x \in X} \max_{y \in Y_i} \phi(x, y). \tag{10.3}$$

Step 2: *Compute a* $y_{i+1} \in \arg\max_{y \in Y} \phi(x_{i+1}, y)$ *and set* $Y_{i+1} = Y_i \cup \{y_{i+1}\}$.

Step 3: *Replace* i *by* $i + 1$ *and go to* **Step 1**.

The Conceptual Method of Outer Approximations decomposes the original semi-infinite min-max problem (10.1) into an infinite sequence of *finite* min-max problems of the form (10.3), i.e.,

$$\min_{x \in X} \max_{y \in Y_i} \phi(x, y), \tag{10.4}$$

where the sets Y_i are finite, but growing monotonically in size as the computation proceeds. The algorithm to be used for solving (10.4) is not specified in the Method of Outer Approximations and the user is free to choose one.

In [20] page 438, we find the following result:

Theorem 1. *Every accumulation point* $\hat{x}$ *of a sequence* $\{x_i\}_{i=0}^{\infty}$*, constructed by the Conceptual Method of Outer Approximations, is a global minimizer for problem (10.1).*

When the sets X, Y in (10.1) are discrete, i.e., $X = \{x_1, \ldots, x_p\}$, $Y = \{y_1, \ldots, y_q\}$, the values $\phi(x_i, y_j)$ define a $p \times q$ matrix A, with elements $A_{i,j}$ defined by

$$A_{i,j} = \phi(x_i, y_j), i = 1, \ldots, p, j = 1, \ldots, q \tag{10.5}$$

and the min-max problem (10.1) can be viewed as a matrix game:

$$\min_{i \in \mathbf{P}} \max_{j \in \mathbf{Q}} A_{i,j}, \tag{10.6}$$

where

$$\mathbf{P} = \{1, \ldots, p\}, \quad \mathbf{Q} = \{1, \ldots, q\}. \tag{10.7}$$

In the applications section, where we will introduce a successive approximations technique, we will make use of min-max problems defined on submatrices of the matrix A, i.e., on submatrices defined by the elements $A_{i,j}$ of A, with $i \in \mathbf{I}$ and $j \in \mathbf{J}$, where $\mathbf{I} \subset \mathbf{P}$ and $\mathbf{J} \subset \mathbf{Q}$. These problems have the form

$$\min_{i \in \mathbf{I}} \max_{j \in \mathbf{J}} A_{i,j} \tag{10.8}$$

The obvious interpretation of the Method of Outer Approximations, for solving matrix games of the form (10.8), is as follows:

Matrix Method of Outer Approximations 1.

Data: *A $p \times q$ Matrix A, with elements $A_{i,j}$, and a row index $i_0 \in \mathbf{I}$.*

Step 0: *Set $k = 0$, and compute the column index*

$$j_0 = \arg\max_{j \in \mathbf{J}} A_{i_0,j} \tag{10.9}$$

and set $\mathbf{J}_0 = \{j_0\}$.

Step 1: *Solve the problem*

$$\min_{i \in \mathbf{I}} \max_{j \in \mathbf{J_k}} A_{i,j} \tag{10.10}$$

to obtain its value m_k and corresponding row index i_k of A.

Step 2: *Find the largest element A_{i_k,j_k} in the i_k row of the matrix A and the corresponding column index j_k of A.*

Step 3: *If $A_{i_k,j_k} = m_k$, stop (since the triplet $\{m^*, i^*, j^*\} := \{m_k, i_k, j_k\}$ is a solution of (10.8)). Else, set*

$$\mathbf{J}_{k+1} = \mathbf{J}_k \cup \{j_k\}, \tag{10.11}$$

replace k by $k+1$ and go to **Step 1**.

Theorem 2. *The Matrix Method of Outer Approximations solves the problem (10.8).*

Proof *Let $\mathbf{J}_k$ be any subset of $\mathbf{J}$. Then for any $i \in I$,*

$$\max_{j \in \mathbf{J}_k} A_{i,j} \leq \max_{j \in \mathbf{J}} A_{i,j} \tag{10.12}$$

and hence

$$\min_{i \in \mathbf{I}} \max_{j \in \mathbf{J_k}} A_{i,j} \leq \min_{i \in \mathbf{I}} \max_{j \in \mathbf{J}} A_{i,j}. \tag{10.13}$$

Let $i_k \in \mathbf{I}$, $j_k \in \mathbf{J_k}$ be such that

$$A_{i_k,j_k} = \min_{i\in\mathbf{I}} \max_{j\in\mathbf{J}_k} A_{i,j} \tag{10.14}$$

and suppose that

$$A_{i_k,j_k} = \max_{j\in\mathbf{J}} A_{i_k,j}, \tag{10.15}$$

then we have that

$$A_{i_k,j_k} = \max_{j\in\mathbf{J}} A_{i_k,j} \leq \min_{i\in\mathbf{I}} \max_{j\in\mathbf{J}} A_{i,j}, \tag{10.16}$$

i.e., $\{A_{i_k,j_k}, i_k, j_k\}$ is also the triplet solution for the full min-max problem (10.8), and hence it is clear that the Matrix Method of Outer Approximations solves the problem (10.8).

As stated, the Matrix Method of Outer Approximations involves a lot of costly duplicate function evaluations, which can be eliminated as in the following streamlined version of the Matrix Method of Outer Approximations. In this version, the maximum values computed by the Matrix Method of Outer Approximations using the index set $\mathbf{J}_k$ are stored in a vector M. Hence the construction of the index set $\mathbf{J}_{k+1}$ requires only the computation of $\mu_k := \max_{j\in\mathbf{J}\setminus\mathbf{J}_k}$ followed by finding the larger of μ_k and M_{i_k}. These two operations can be combined notationally, as we will see.

Matrix Method of Outer Approximations 2

Data: *A $p \times q$ matrix A, index sets $\mathbf{I} \subset \mathbf{P}$, $\mathbf{J} \subset \mathbf{Q}$, $j_0 \in \mathbf{J}$.*

Step 0: *Set $k = 0$, $M \in \mathbb{R}^p$, with $M_i = -\infty$ for all $i \in \mathbf{Q}$, $\mathbf{J}_0 = \{j_0\}$.*

Step 1: *For all $i \in \mathbf{I}$, Set $M_i = \max\{M_i, A_{i,j_k}\}$.*

Step 2: *Set*

$$m_k = \min_{i\in\mathbf{I}} M_i, \tag{10.17}$$

$$i_k = \arg\min_{i\in\mathbf{I}} M_i. \tag{10.18}$$

Step 3: *Set*

$$m_k^* = \max_{j\in\mathbf{J}\setminus\mathbf{J}_k} \{M_{i_k}, A_{i_k,j}\} \tag{10.19}$$

$$j_k = \arg\max_{j\in\mathbf{J}\setminus\mathbf{J}_k} \{M_{i_k}, A_{i_k,j}\} \tag{10.20}$$

Step 4: *If $m_k^* = M_{i_k}$, set $m^* = m_k$, $i^* = i_k$, $j^* = j_k$ to be the solution of (10.8) and stop. Else, set $\mathbf{J}_{k+1} = \mathbf{J}_k \cup \{j_k\}$, $k = k+1$, and go to* **Step 1**.

In our numerical experiments, we will compare the performance of the Method of Outer Approximations 2 with that of the Basic Algorithm below, which evaluates all the elements of the matrix A, but stores only three numbers, m^*, i^*, j^*, during its execution.

Basic Algorithm.

Data: *A $p \times q$ matrix A, with elements $A_{i,j}$, and two sets $\mathbf{I} \subset \{1, 2, \dots, p\}$, $\mathbf{J} \subset \{1, 2, \dots, q\}$.*

Step 0: *Set $m^* = \infty$, $i^* = p$, $j^* = q$.*

Step 1 *For $i \in \mathbf{I}$ compute*

$$j(i) = \arg\max_{j \in \mathbf{J}} A_{i,j} \tag{10.21}$$

If $A_{i,j(i)} < m^*$, set $m^* = A_{i,j(i)}$, $i^* = i$ and $j^* = j(i)$.

The following result is obvious.

Proposition 3. *If m^*, i^*, j^* are computed by the Basic Algorithm, then m^* is the optimal value of the matrix game (10.8), restricted to the submatrix defined by the sets $\mathbf{I}$ and $\mathbf{J}$, and i^*, j^* define the corresponding solution row and column of A.*

It should be obvious, that Matrix Method of Outer Approximations 2 is never less efficient than the Basic Algorithm in solving arbitrary matrix games of the form (10.8). However, as we will see in the following two sections, that when combined with sequential precision refinement to obtain good starting points, it produces spectacular improvements over the Basic Algorithm on a class of important min-max problems.

10.3 A Harbor Defense Problem

In recent years, small unmanned vehicles have become inexpensive and deadly weapons. It is easy to imagine a scenario where a small unmanned explosives-packed submarine is launched by terrorists from a freighter, at a safe distance from the entrance to a harbor, with the mission of destroying a large cruise ship carrying many thousands of passengers. The effect could be as devastating as the 9/11, 2001 attack on the World Trade Center in New York. Idealizing a real world situation, we consider a harbor that can be reached via a rectangular channel of width W and assume that an intruder tries to reach the target ships anchored at the end of the harbor. The task of the defender craft is to prevent the intruders from reaching its target, as illustrated in Fig. 4.1. The defender can achieve its goal either by destroying the intruder or causing the intruder to flee.

We assume that the defending vehicle is a unmanned submarine, hovercraft, or drone, whose behavior is determined by a feedback model predictive control (MPC) law (see [24]) based on a min-max optimal control problem which, we believe, captures the essence of the intruder's goal of getting within striking distance, δ of its target, as well as the intruder's perception of how it can be destroyed by the defender.

The idea behind Model Predictive Control (MPC) is quite old, going back to the 1950's. It consists of generating a feedback law for a dynamical system by recomputing an optimal trajectory for the dynamical system every Δ time units, where Δ is called the sampling time. This mechanism makes it possible to compensate for changing conditions as time proceeds. For an excellent survey of classical uses of MPC, see [24]. For previous attempts of using MPC in pursuit-evasion situations, see [55], [82].

The situation we will describe is more complicated than the one in the classical case in that there is not one but two dynamical systems involved with opposing objectives. In addition, we assume that the steering settings are not continuously variable, but can only take a few discrete values, such as stop, full speed ahead, full speed 60 degrees to the right, and full speed 60 degrees to the left. As a result, the optimal control problem that the defender needs to solve is a min-max optimal control problem, with discrete variables. In our scenario, the defender is able to determine the dynamics of the intruder as well as its position, velocity, and direction of travel.

10.3.1 Dynamics

Assuming that both the defender and intruder are unmanned underwater vehicles (UUVs), and that they are confined to a rectangular channel of width W (at the end of which is the harbor with a high value target), their dynamics have the form

$$\begin{aligned}
\dot{x}^1(t) &= v(t)\cos\theta(t),\\
\dot{x}^2(t) &= v(t)\sin\theta(t),\\
\dot{\theta}(t) &= \sigma(t),\\
\dot{v}(t) &= \alpha(t),
\end{aligned} \tag{10.22}$$

where x^1 is the positional coordinate of the UUV along the channel, x^2 is the positional coordinate of the UUV perpendicular to the channel, and θ is the heading, i.e., the angle between the direction of motion of the UUV and the x^1 axis in our positional coordinate system. We assume that the channel is sufficiently shallow that a depth coordinate is not needed. We assume that there is a steering input $\sigma(t)$ and a propulsion input $\alpha(t)$, which are subject to constraints of the form:

$$\begin{aligned} &0 \leq v(t) \leq \bar{v}, \\ &|\alpha(t)| \leq \bar{\alpha}, \\ &|\sigma(t)| \leq \bar{\sigma}, \\ &\sigma(t)v(t) \leq k_f. \end{aligned} \tag{10.23}$$

The constraint $\sigma(t)v(t) \leq k_f$ captures the relationship between centripetal force and velocity.

We assume that the model predictive control law uses a sample time Δ, so that for any integer $k \geq 0$ and $t \in [k\Delta, (k+1)\Delta)$ the controls are constant, i.e., for $t \in [k\Delta, (k+1)\Delta)$, $v(t) = v(k\Delta)$ and $\sigma(t) = \sigma(k\Delta)$. We can integrate the differential equation (10.22) analytically for $t \in [k\Delta, (k+1)\Delta)$, to obtain the difference equations:

$$\begin{aligned} x_{k+1}^1 &= x_k^1 + \Delta v_k \cos\theta_k + \Delta^2 \alpha_k \cos\theta_k, \\ x_{k+1}^2 &= x_k^2 + \Delta v_k \sin\theta_k \Delta^2 \alpha_k \sin\theta_k, \\ \theta_{k+1} &= \theta_k + \Delta\sigma_k, \\ v_{k+1} &= v_k + \Delta\alpha_k, \end{aligned} \tag{9c}$$

where $x_k^1 = x^1(k\Delta)$, $x_k^2 = x^2(k\Delta)$, $\theta_k = \theta(k\Delta)$, $v_k = v(k\Delta)$, $\sigma_k = \sigma(k\Delta)$, and $\alpha_k = \alpha(k\Delta)$.

10.3.2 Defender model predictive Control Law

To distinguish between the intruder and defender, we will add a subscript i to indicate the intruder states, controls, and constraints, and a subscript d to indicate the defender states, controls, and and constraints. Let

$$z_d(k\Delta) = (x_d^1(k\Delta), x_d^2(k\Delta), \theta_d(k\Delta), v_d(k\Delta))^T$$

denote the state of the defender at time $t = k\Delta$ and let

$$z_i(k\Delta) = (x_i^1(k\Delta), x_i^2(k\Delta), \theta_i(k\Delta), v_i(k\Delta))^T$$

denote the state of the intruder at time $t = k\Delta$. Similarly, let

$$u_d(k\Delta) = (\sigma_d(k\Delta), \alpha_d(k\Delta))^T$$

denote the control for the defender at time $t = k\Delta$ and let

$$u_i(k\Delta) = (\sigma_i(k\Delta), \alpha_i(k\Delta))^T$$

denote the control for the intruder at time $t = k\Delta$.

Defender model predictive Control Algorithm

Data: *Sampling Time* $= \Delta$, *horizon* $= N\Delta$, *initial defender and intruder states* $z_d(0), z_i(0)$.

Step 1: *Set* $k = 0$.

Step 2: *Set* $z(0) = z_d(k\Delta)$ *in (9c) to compute the defender trajectory and* $z_i(0) = z_i(k\Delta)$ *in (9c) to compute the intruder trajectory.*

Step 3: *Solve the defender min-max problem (10c), below, for an optimal defender control sequence* $\mathbf{u}_d^* = (u_d^*(0), u_d^*(\Delta), \ldots, u_d^*((N-1)\Delta))$.

Step 4: *Apply the control* $u_d^*(0)$ *to the defender for* Δ *units of time.*

Step 5: *At time* $(k+1)\Delta$, *measure the states* $z_d((k+1)\Delta)$ *and* $z_i((k+1)\Delta)$.[2]

Step 6: *Replace* k *by* $k+1$ *and go to* **Step 2**.

Note that the min-max problem in Step 4, above, can be changed at each sampling time, and so can the sample time Δ. It makes sense to use a large Δ when the adversaries are far apart and decrease it as they get nearer to each other.

10.3.3 Min-Max Problem Formulation

Next, we introduce a min-max optimal control problem, reflecting a worst case scenario, that must be solved at each sample time within the MPC law.

Suppose that the horizon length is $N\Delta$, with $N = 2^s$ and $s \geq 2$ a positive integer. For $k = 0, \ldots, N-1$, let $u_d(k\Delta) = (\sigma_d(k\Delta), \alpha_d(k\Delta))^T$ and $u_i(k\Delta) = (\sigma_i(k\Delta), \alpha_i(k\Delta))^T$.

Assuming that the intruder is risk averse and will not venture within torpedo striking distance $\tau > 0$ of the defenders, we propose the following max-min optimal

[2]A more realistic statement of the model predictive control algorithm includes provisions for the time it takes to solve the appropriate min-max problem of the form (10.8)

control problem for the receding horizon control law:

$$\begin{aligned}\max_{\mathbf{u}_d\in\mathbf{U}_d}\min_{\mathbf{u}_i\in\mathbf{U}_i}\big\{&\min_{k\in\mathbf{N}}\{x_i^1(k\Delta,\mathbf{u}_i)+\\ &\pi\max_{k\in\mathbf{N}}\{-\|x_i(k\Delta,\mathbf{u}_i)-x_d(k\Delta,\mathbf{u}_d)\|^2+\tau^2\}_+\}\big\},\end{aligned}\tag{10.24}$$

where $\{x\}_+=\max\{0,x\}$, $\mathbf{N}=\{1,2,\ldots,N\}$ and $\pi>0$ is a sufficiently large penalty to ensure that the intruder keeps his distance from the defender.

Making use of the fact that

$$\max_{x\in X}\min_{y\in Y}\phi(x,y)=\max_{x\in X}(-\max_{y\in Y}-\phi(x,y))=-\min_{x\in X}\max_{y\in Y}(-\phi(x,y)),\tag{10.25}$$

We can rewrite (10.24) as

$$\begin{aligned}-\min_{\mathbf{u}_d\in\mathbf{U}_d}\max_{\mathbf{u}_i\in\mathbf{U}_i}-\big\{&\min_{k\in\mathbf{N}}\{x_i^1(k\Delta,\mathbf{u}_i)+\\ &\pi\max_{k\in\mathbf{N}}\{-\|x_i(k\Delta,\mathbf{u}_i)-x_d(k\Delta,\mathbf{u}_d)\|^2+\tau^2\}_+\}\big\},\end{aligned}\tag{10.26}$$

The constraint set $\mathbf{U_d}$ for the defender N-sample control string $\mathbf{u}_d=(u_d(0),\ldots,u_d((N-1)\Delta))$ is defined by

$$\begin{aligned}\mathbf{U_d}=\{\mathbf{u_d}=&(u_d(0),\ldots,u_d((N-1)\Delta))\\ s.t.\ &0\le v_d(k\Delta)\le\bar{v}_d,\\ &|\sigma_d(k\Delta)|\le\bar{\sigma}_d,\\ &|\alpha_d(k\Delta)|\le\bar{\alpha}_d,\\ &\sigma_d(k\Delta)v_d(k\Delta)\le\kappa_d,\\ &0\le x_d^1(k\Delta,\mathbf{u}_d),\\ &0\le x_d^2(k\Delta,\mathbf{u}_d)\le W\},\end{aligned}\tag{10.27}$$

where $k=\{0,\ldots,N-1\}$, $\bar{v}_d>0$ is the speed limit for the defender, $\bar{\sigma}_d$, is the limit on the steering inputs for the defender, and W is the width of the channel.

For the intruder control string $\mathbf{u}_i=(u_i(0),\ldots,u_i((N-1)\Delta))$ we get a similar result:

$$\begin{aligned}\mathbf{U_i} = \{&\mathbf{u_i} = (u_i(0), \dots, u_i((N-1)\Delta)) \\ s.t.\ &0 \le v_i(k\Delta) \le \bar{v}_i, \\ &|\sigma_i(k\Delta)| \le \bar{\sigma}_i, \\ &0 \le x_i^1(k\Delta, \mathbf{u}_i), \\ &0 \le x_i^2(k\Delta, \mathbf{u}_i) \le W, k = 0, \dots, N-1, \\ &\sigma_i(k\Delta)v_i(k\Delta) \le k_i\},\end{aligned} \tag{10.28}$$

where τ is the distance at which intruder torpedoes become effective. At this point, we assume that the inputs for the defender and intruder can be adjusted in discrete increments only. In particular, we assume that σ_d can only take the values $0, \bar{\sigma}_d, -\bar{\sigma}_d$ and α_d can only take the values $0, \bar{\alpha}_d, -\bar{\alpha}_d$, and similarly for the intruder. Hence we see that the control $u_d(k\Delta)$ can assume no more than 9 possible values, and the same holds for the control $u_i(k\Delta)$, $k = 0, 1, \dots, N-1$.

If we associate these control combinations with the integers $0, 1, 2, \dots, 8$, e.g., $(\sigma_d = 0, \alpha_d = 0)$ with the number 0, $(\sigma_d = \bar{\sigma}_d, \alpha_d = 0)$ with the number 1, etc, we can represent any control sequence $\mathbf{u_d}$ as a N digit number to base 9, where the first digit defines the control at time $t = 0$, the second digit defines the control at time $t = \Delta$, the third digit defines the control at time $t = 2\Delta$, etc. Clearly, there are 9^N possible defender and intruder control sequences.

We adopt the convention that every number to base 9, in the range of 0 to 9^N, is always written as an N digit number, so the number 1 is written as $000\dots01$, the number 9 is written as $000\dots010$, etc.

We can number the possible defender controls as $\mathbf{u_{d,i}}$, $i = 1, 2, \dots, 9^N$ and the intruder controls as $\mathbf{u_{i,j}}$, $j = 1, 2, \dots, 9^N$, which result in a $9^N \times 9^N$ cost matrix A, with elements

$$A_{i,j} = -\min_{k\in\mathbf{N}}\{x_i^1(k\Delta, \mathbf{u}_{i,j}) + \pi \max_{k\in\mathbf{N}}\{-\|x_i(k\Delta, \mathbf{u}_{i,j}) - x_d(k\Delta, \mathbf{u}_{d,i})\|^2\tau^2\}_+\}. \tag{10.29}$$

Now, the indices i and j in the matrix elements $A_{i,j}$ are expressed as numbers to base 10. To determine which defender and intruder control sequences are to be used in computing $A_{i,j}$, we convert i and j to N digit numbers to base 9, and then deduce the control sequences from these numbers.

If we assume that the speeds of the defender and of the intruder are constant, then $\sigma_d(k\Delta) = \sigma_i(k\Delta) = 0$ for $k = 0, 1, \dots, N-1$, and the number of possible control values at each sample time reduces to 3. In this case, the control sequences can be represented by an N digit number to base 3, and the resulting matrix A

has dimensions $3^N \times 3^N$.

It is simpler to take a general approach and assume that the number of possible control values at each sample time is some positive integer k, so that the the resulting A matrix has dimensions $k^N \times k^N$. Let $\mathbf{K} = \{1, 2, \ldots, k^N\}$ and let $\mathbf{F}_d \subset \mathbf{K}$ be the set of all the elements in $\mathbf{K}$ such that when expressed as an N digit number to the base of k they represent a feasible control for the defender (i.e., one in $\mathbf{U_d}$), and similarly, let $\mathbf{F}_i \subset \mathbf{K}$ be the set of all the elements in $\mathbf{K}$ such that when expresses as a N digit number to the base of k they represent a feasible control (i.e., one in $\mathbf{U}_i$) for the Intruder.

Then we see that the problem (10.26) can be rewritten as the matrix game

$$-\min_{i \in \mathbf{F}_d} \max_{j \in \mathbf{F}_i} A_{i,j}, \tag{10.30}$$

where A is the matrix whose elements are defined by (10.29).

10.3.4 Successive Precision Refinement

In the case of the continuous min-max problem, the time needed to solve the problem using the Method of Outer Approximations is very strongly dependent on the quality of the starting point. Not surprisingly, the same is true for the Matrix Method of Outer Approximations 2 applied to the discretized harbor defense problem. To obtain an excellent starting point for the $k^N \times k^N$ matrix game, we proceed as follows. For the sake of simplicity, let us assume that $N = 16$ and that $k = 3$, that would result in a 43046721×43046721 A matrix in the discretized game. So, instead, we proceed as follows. We begin by imposing the restriction that the defender and intruder controls are kept the same for the first 8 samples and then, again, for the next 8 samples, i.e., we assume that there are only two samples which are 8 times as long as the actual samples. This results in a min-max game (ignoring the - sign in (10.29)) with a $3^2 \times 3^2$, i.e., a 9×9 A matrix. The solution of the resulting game, obtained using the Basic Algorithm, is of the form of a triplet $A_{i_0^* j_0^*}, i_0^*, j_0^*$, where $i_0^*, j_0^* \in \{1, 2, ..., 9\}$. Next, we proceed to solve a game with 4 samples. Writing i_0^* and j_0^* as integers to base 4, $i_0 = ab$, $j_0 = cd$, with $a, b, c, d \in \{0, 1, 2, 3\}$. The matrix game corresponding to 4 samples, has a $3^4 \times 3^4$ matrix $A_{i,j}$. We initialize its solution process by finding the largest element $A_{i_0^*, j^*}$ in the $i_0^* = aabb$ row (read as a number to base 4) of the new matrix $A_{i,j}$, and store the j^* column of A. We use this column to initialize the matrix outer approximations algorithm to obtain the next solution A_{i_1,j_1}, i_1, j_1. We then continue to 8-sample and finally to 16-sample controls. We can restate the above description as a general procedure for constructing the sets $\mathbf{I}, \mathbf{J}$, for the case of $N = 2^w$ sample controls, which can assume k values at each sample time. We assume that the horizon length $N = 2^w$, with $w \geq 1$, a positive integer, is the same for the defender and intruder as well as the number of control values at each sample time.

In the algorithm below, which computes the index sets to be used at each successive optimization stage in the algorithm that follows, N denotes the number of samples in the horizon, k denotes the number of control values that are possible at each sample time, and l denotes the number of samples for which the controls must be kept the same. To simplify exposition, we assume that the parameters N, k, and l are the same for the intruder and the defender. These assumptions are carried over to the algorithm after the one below.

Precision Refining Index Definition Algorithm 1.

Data: *Positive integers $N = 2^w$, where $w \geq 1$ is an integer, k, and $l = 2^v$, where $v < w$.*

Step 0: *Set $\mathbf{I} = \mathbf{J} = \{1, 2, \ldots, N\}$.*

Step 1:
```
for i=1 to N,
    express $i$ as a base $k$ string $(i_1, i_2, \ldots, i_N)$.
        for $j = 1$ to $N/l$,
            If $i_{(j-1)l+1} = i_{(j-1)l+2} = \ldots = i_{(j-1)l+l}$ is false, remove $i$
                from $\mathbf{I}$ and $\mathbf{J}$.
```

Successive Precision Refining Algorithm 1.

Data: *Positive integer $N = 2^w$, where $w \geq 1$ is an integer, integers k, and $l = 2^v$, where $v < w$; Initial "two-sample" control for the defender: $\mathbf{u}_d^*$ with $(u_d^*(0) = u_i(10.1) = \cdots = u_d^*(N/2-1)$, and $u_i^*(N/2) = u_i^*(N/2+1) = \cdots = u_i^*(N-1)$, and the corresponding row index i^* for the matrix A defined by (10.29).*

Step 0: *Set $l = N/2$.*

Step 1: *Use the Precision Refining Index Definition Algorithm 1 to compute the index sets $\mathbf{I}$ and $\mathbf{J}$.*

Step 2: *Initialize the Matrix Method of Outer Approximations 2 by setting the new values $\mathbf{I} = \mathbf{I} \cap \mathbf{F}_d$, $\mathbf{J} = \mathbf{J} \cap \mathbf{F}_i$, and $i_0 = i^*$, and use it to solve the min-max problem (10.29), to obtain its solution m^*, i^*, j^*. Convert the index i^**

into the defender control sequence $\mathbf{u}_d$ *with*
$u_d^*(kN/l) = u_d^*(kN/l+1) = u_d^*(kN/l+2) = \cdots = u_d^*((k+1)N/l-1)$,
for $k = 0, 1, \ldots, N/l - 1$, *and convert the index* j^* *into the intruder control sequence* $\mathbf{u}_d$ *with* $u_i^*(kN/l) = u_i^*(kN/l+1) = u_i^*(kN/l+2) = \cdots = u_i^*((k+1)N/l-1)$, *for* $k = 0, 1, \ldots, N/l - 1$.

Step 3: *Set* $l = l/2$. *If* $l < 1$ *Stop, the last computed control sequences are optimal for the problem (10.29). Else go to* **Step 1**.

In practice, it is simpler and more efficient to proceed as follows. Replace Δ by $\Delta = N/2$, which automatically results in a 2 sample optimal control problem and hence in a $k^2 \times k^2$ matrix game which can be solved by the Matrix Method of Outer Approximations. Next, replace Δ by $\Delta = N/4$, which results in a 4 sample optimal control problem, and hence in a $k^4 \times k^4$ matrix game. Interpret the solution of the previous game as an initial point for the new game and use the Matrix Method of Outer Approximations to solve it, etc.

10.3.5 Numerical Results: Optimal Control

We considered two scenarios. In the first scenario, the speed was kept constant for both the intruder and the defender at their maximum value $\bar{v}_i = \bar{v}_d = 1$. The other parameters were set as follows: $\bar{\alpha_i} = \bar{\alpha_d} = 1, \bar{\sigma}_i = \bar{\sigma}_d = 0.1$, and $k_f = 1$. The initial state for the intruder was $z_i(0) = \{40, 10, \pi, 0\}$ and the initial state for the defender was $z_d(0) = \{10, 10, 0, 0\}$. The initial control for the defender was $u_d(k\Delta) = (\sigma_d(k\Delta), \alpha_d(k\Delta))^T = (0, 1)^T$ for $k = 0, \cdots, N-1$. The numerical experiment were carried out in MATLAB on a MacBook Pro with an Intel i7 processor, 2.3 GHz speed, and 16 GB random access memory.

Numerical results obtained using the Successive Precision Refining Algorithm 1, which incorporates the Matrix Method of Outer Approximations 2, are presented in Table 10.10.1 . In Table 10.1 and the subsequent tables, MOA and BA stand for "Method of Outer Approximations" and "Basic Algorithm" respectively. The unit of computation time is the second, unless specified otherwise. The numerical solution of the min-max problem (10.24), transcribed into the form (10.1), using the BA is presented for the comparison purposes. The 'Horizon Samples' column shows the number of samples in the horizon. The number of function evaluations of MOA is $2 \times$ Iters $\times$ number of rows of A plus the initialization cost. The factor 2 is needed because in each MOA iteration, the BA has to search one row and one column. The initialization cost is due to the fact that the data needs to be computed in the first stage of the Successive Precision Refining Algorithm 1, which requires N function evaluations. Obviously, The number of function evaluations of BA is number of rows of $A \times$ number of rows of A.

Note that Table 10.1 gives the computing times at each stage of the Precision Refining Algorithm 1. To obtain the total computing time, the reader needs to add the results for each stage, which are dominated by the computing time in the last stage.

Table 10.1: Numerical experiment result: Single input case.

Horizon Samples	A size	MOA		BA	$\frac{\text{BA}}{\text{MOA}}$
		Iters	Comp.Time	Comp.Time	
2	$3^2 \times 3^2$	5	0.025	0.02	0.8
4	$3^4 \times 3^4$	3	0.075	0.92	12.27
8	$3^8 \times 3^8$	6	5.64	2847	504.7
16	$3^{16} \times 3^{16}$	10	20 hrs	4982 yrs*	1.7×10^6

* are estimated values

Throughout the numerical experiment, the computing time difference between the results obtained using the Successive Precision Refining Algorithm 1 and those obtained using the BA gets exponentially larger and larger as the number of stages grows. There are two reasons for this. The first reason is that the MOA constructs a column such that every updates of it contains a maximizer of the previous iterations. Since this columns are "close" to a maximizing column, the MOA can reduce the number of function evaluations needed to find a minimizer. The second reason is that the the Successive Precision Refining Algorithm 1 produces good starting points for the MOA. The reason for this is that when the objective function $\phi(\cdot,\cdot)$ is continuous, the Successive Precision Refining Algorithm 1 constructs finer and finer approximations to a continuous min-max problem of the form (10.1), whose solutions converge to the solution of the continuous min-max. Hence, not surprisingly, the solution of a coarser problem is a good approximation to the solution of the next, finer problem, with the result that although we use many stages, in each the number of iterations remains quite small and hence the overall computing time is spectacularly reduced, compared to a one stage solution of the finest discretization that one needs to deal with. In the next experiment, we use two inputs: angular velocity and thrust. Due to limitations imposed by MATLAB, the number of samples in the horizon was limited to 8. Results are presented in Table 10.2.

We note that, just as in the single input case, the computing time difference between the two algorithms gets exponentially larger and larger as the discretization

Table 10.2: Numerical experiment result: Two inputs case.

Horizon Samples	A size	MOA		BA	$\frac{\text{BA}}{\text{MOA}}$
		Iters	Comp.Time	Comp.Time	
2	2	7×9^2	0.11	0.97	8.81
4	5	8×9^4	6.3	5166	8210.12
8	6	10×9^8	13 hrs	6476 yrs*	4.3×10^6

* estimated values

gets finer and finer.

10.4 Global Optimization

In this section, we demonstrate the efficiency of the Matrix Method of Outer Approximations 2 in finding approximations to global solutions of min-max problems of the form (10.1), where $x, y \in \mathbb{R}$, so that the sets $X = [a_x, b_x]$ and $Y = [a_y, b_y]$ are intervals. When we discretize the intervals into equi-spaced grids of $N + 1$ points, we get a matrix game defined by an $(N + 1) \times (N + 1)$ matrix A with elements $A_{i,j} = \phi(a_x+(i-1)(b_x-a_x)/N, a_y+(j-1)(b_y-a_y))/N$, with $i, j \in \{1, 2, \cdots, N+1\}$. Note that the end points of the intervals $[a_x, b_x]$ and $[a_y, b_y]$ are included in the grids.

Definition 1. Let $\epsilon \geq 0$ define the precision with which the min-max problem (10.1) must to be solved. Let $l_x = b_x - a_x$ and $l_y = b_y - a_y$. Let $s \in \mathbb{N}$ be the number stages in which the min-max problem (10.1) is to be solved, and let $r \in \mathbb{N}$ be the smallest integer such that $\max\{l_x/r^s, l_y/r^s\} \leq \epsilon$. Let $N = r^s$.

Given the decision to solve the discretized min-max problem in s stages, the following algorithm computes k, the number of grid points in the intervals $[a_x, b_x, a_y, b_y]$ to be used in stage $q \leq s$, and the indices, accumulated in the sets $\mathbf{I}, \mathbf{J}$, of the k rows and columns of the A matrix that must be used in stage q[3].

Precision Refining Index Definition Algorithm 2.

Data: *Positive integers s = the total number of stages to be used, r, such that $N = r^s$ is the number of discretization intervals in X and Y to be used for the construction of the $N+1 \times N+1$[4] matrix A, and the stage number $q \leq s$*
.

[3]The cardinality of $\mathbf{I}$ and $\mathbf{J}$ is obviously k.

[4]The number of points in X and Y respectively is $N + 1$

```
Step 0: Set $k = r^q$ and $l = N/k$;
Step 1: Set $i(10.1) = 1$.
Step 2: for $j = 1$ to $k$,
          Set $i(j+1) = i(j) + l$.
Step 3 Set $\mathbf{I} = \mathbf{J} = i$
```

It may be helpful to examine a couple of examples of the use the Precision Refining Index Definition Algorithm 2.

Example 1. Suppose that we want to break up the search intervals for both x and y into $3^3 = 27$ sections, so that $r = 3$. That results in 28 search points and hence a matrix game with A a 28×28 matrix. Now suppose that we propose to solve this game in $s = 3$ stages. In stage 1, we set $q = 1$. Then in Step 0, we compute $k = r^q = 3$ and hence $l = 9$. Thus, for stage 1, the Precision Refining Index Definition Algorithm 2 computes the index sets

$$\mathbf{I} = \mathbf{J} = \{1, 10, 19, 28\},$$

which define a 4×4 matrix A.

In stage 2, $q = 2$, $k = 9$, and hence $l = 3$. Hence the Precision Refining Index Definition Algorithm 2 computes the index sets

$$\mathbf{I} = \mathbf{J} = \{1, 4, 7, 10, 13, 16, 19, 22, 25, 28\},$$

which define a 10×10 matrix A. Note that the index sets for stage 2 contain the index sets for stage 1, hence the matrix game solution computed in stage 1 can be used to initialize the Matrix Method of Outer Approximations in stage 2.

Finally, we see that in stage 3, the index sets are

$$\mathbf{I} = \mathbf{J} = \{1, 2, 3, \ldots, 26, 27, 28\},$$

Example 2. Now suppose that we want to break up the search intervals for both x and y into $10^3 = 1000$ sections, so that $r = 10$. That results in 1001 search points and hence a matrix game with a 1001×1001 matrix.

Again, suppose that we propose to solve this game in $s = 3$ stages. In stage 1, we set $q = 1$. Then in Step 0, we compute $k = r^q = 10$ and hence $l = 100$. Thus, for stage 1, the Precision Refining Index Definition Algorithm 2 computes the index sets

$$\mathbf{I} = \mathbf{J} = \{1, 101, 201, 301, 401, 501, 601, 701, 801, 901, 1001\},$$

which define an 11×11 matrix A.

In stage 2, $q = 2$, $k = 100$, and hence $l = 10$. Therefore the Precision Refining Index Definition Algorithm 2 computes the index sets

$$\mathbf{I} = \mathbf{J} = \{1, 11, 21, \ldots, 101, 111, 121, \ldots, 201, 211, \ldots, 981, 991, 1001\},$$

which define a 101×101 matrix A. Again note that the index sets for stage 2 contain the index sets for stage 1, hence the matrix game solution computed in stage 1 can be used to initialize the Matrix Method of Outer Approximations in stage 2.

Finally, we see that in stage 3, the index sets are

$$\mathbf{I} = \mathbf{J} = \{1, 2, 3, \ldots, 999, 1000, 1001\},$$

which define the entire matrix A, and it is ovbious that the solution computed in stage 2 can be used to initialize the Matrix Method of Outer Approximations in stage 3.

Successive Precision Refining Algorithm 2.

Data: *Positive integers s = the number of stages to be used, r, such that $N = r^s$ is the number of discretization intervals in X and Y to be used for the construction of the $(N+1) \times (N+1)$ matrix A. Also, a formula $\phi(x, y)$ for computing the elements of the $(N+1) \times (N+1)$ matrix A in (10.8) and a column index $j^* \in \{1, 2, 3, \ldots, N+1\}$.*

Step 0: *Set $q = 1$.*

Step 1: *Use the Precision Refining Index Definition Algorithm 2 to compute the index sets* $\mathbf{I}$ *and* $\mathbf{J}$.

Step 2: *Initialize the Matrix Method of Outer Approximations 2 with the column $A_{\cdot, j^*}$ and use it to solve the min-max problem (10.8), to obtain its solution: new values for the indices i^*, j^* and the corresponding value A_{i^*, j^*}, of the matrix game defined by the current index sets* $\mathbf{I}$ *and* $\mathbf{J}$.

Step 3: *If $q < s$, set $q = q + 1$ and* **go to Step 1** *, Else stop, A_{i^*, j^*}, i^*, j^* is the solution of the problem (10.8).*

We will now present three sets of numerical examples. The objective functions in examples are continuous and have many local extrema in the constraint intervals. Therefore standard, gradient based semi-infinite optimization algorithms, such as those presented in Section 3.4 of [20], and the smoothing method based approach [83] will most likely fail to find a global minimizer unless the starting point is in the vicinity of a global minimizer.

To solve the example problems to six decimal places, using the Successive Precision Refining Algorithm 2, we discretize the objective functions, as follows. In first two examples, we set $a_x = a_y = -8$, $b_x = b_y = 8$ and used two sets of values of (r, s), $(r = 10, s = 6)$, $(r = 2, s = 17)$, which results in $N = 16 \times 10^6$ and $N = 160 \times 2^{17}$ points, respectively. In the third example, we set $a_x = a_y = -300$, $b_x = b_y = 300$ and it results in $N = 600 \times 10^6$ and $N = 6000 \times 2^{17}$ points, respectively.

In Tables 3, 4 and 5 we present computational results obtained using the Successive Precision Refining Algorithm 2 and the BA. In the tables, the columns labeled 'Interval' contain the values of l used by the Successive Precision Refining Algorithm 2. The initial value of the interval is set to 0.1 for both cases. The 'Iter' column shows the number of iterations of the Modified Basic Algorithm. The 'Eval' column shows the number of evaluations of the elements $A_{i,j}$.

10.4.1 Minimization of The McCormick function

The McCormick function is given by

$$\phi(x, y) = \sin(x + y) + (x - y)^2 - 1.5x + 2.5y + 1 \qquad (10.31)$$

and is plotted in Fig 10.1, where we see that it has multiple extrema.

We solve (10.1) with $X = [-8, 8]$,and $Y = [-8, 8]$. The solution is approximately known to be $(x^*, y^*) = (1.27, 8)$. Numerical results associated with solving the min-max problem, with the McCormick function, is presented in Table 3, at the end of this section. We observe that the MOA requres a very small number of iterations. The time required to obtain the prescribed precision of solution was slightly faster with $r = 10$ than with $r = 2$. The initial value for the column is chosen as the first column of the A matrix from the Matrix Method of Outer Approximations.

The reason for this is that in most stages, for both values of r, the MOA sub-procedure, of the Successive Precision Refining Algorithm 2, required only one iteration to obtain a solution and hence the smaller number of stages associated with $r = 10$ resulted in faster overall solution time. However, $r = 2$ allows a finer

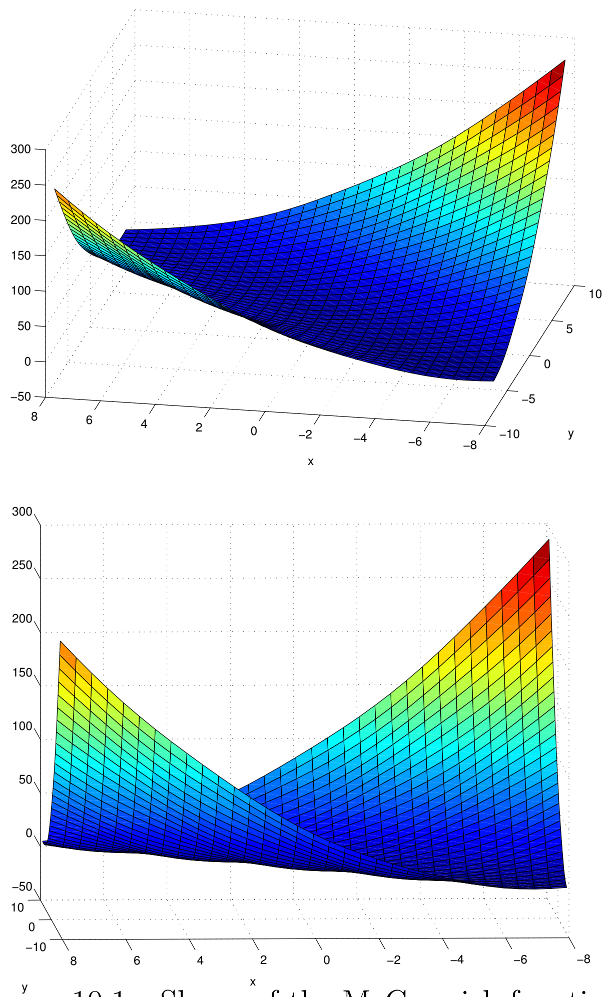


Figure 10.1: Shape of the McCormick function

choice of desired precision.

10.4.2 A Second Example

For our second test in finding global solutions of problems of the form (10.1), we chose the function $\phi(x,y)$ below, which, like the McCormack function, has multiple extrema in the constraint set.

$$\phi(x,y) = \frac{\sin\sqrt{0.5x^2+y^2}}{\epsilon+\sqrt{x^2+y^2}} - \frac{\sin\sqrt{(x-1)^2+y^2}}{\epsilon+\sqrt{(x-1)^2+y^2}}. \tag{10.32}$$

The plot of this function is given in Fig. 10.2.

We solve (10.1) with this $\phi(x,y)$ and with $X = [-8,8]$,and $Y = [-8,8]$.The solution was found approximately to be $(x^*,y^*) = (3.41,-5.23)$. The initial column

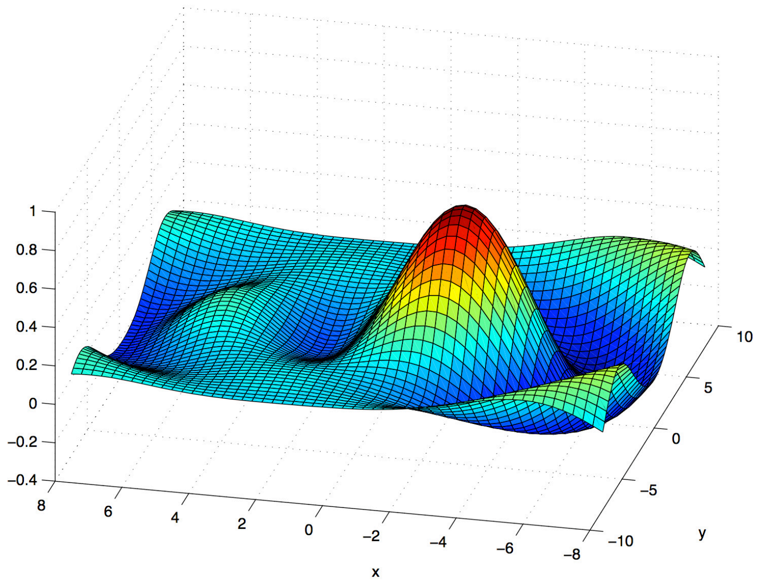


Figure 10.2: Shape of the testing function

is chosen as the first column of the A matrix from the Matrix Method of Outer Approximations.

Experimental results are presented in Table 10.4, at the end of this section, for both $r = 10$ and $r = 2$. In this experiment, the total time required to obtain similar precision results was slightly faster with $r = 10$ than with $r = 2$, for the same reasons as in the experiment with the McCormack function. Computational times for both cases for each level of discretization are presented in Fig 10.3. Note that both axes are to log scale.

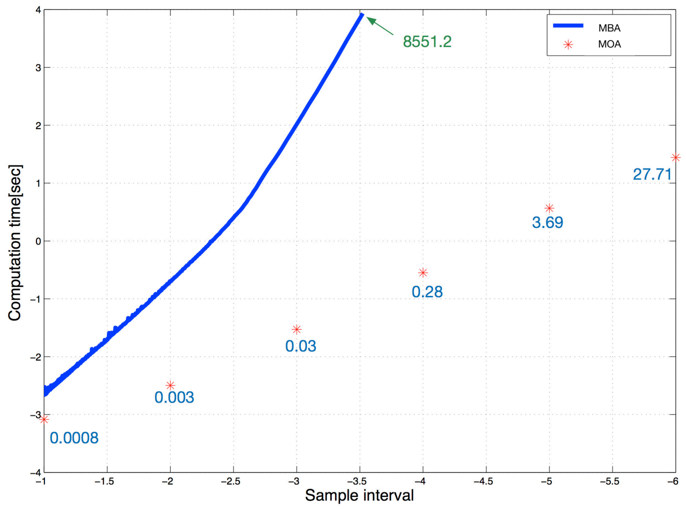


Figure 10.3: Time comparison of MOA and BA of the testing function

10.4.3 A Third Example: Egg holder function

For our third test in finding global solutions of problems of the form (10.1), we chose the function $\phi(x, y)$ below, called egg holder function [84], [85].

$$\phi(x, y) = -(y + 47)\sin\left(\sqrt{\left|y + \frac{x}{2} + 47\right|} - x\sin(\sqrt{|x - (y + 47)|}\right). \qquad (10.33)$$

This function has very large number of multiple extrema, as shown in Figure 10.4.

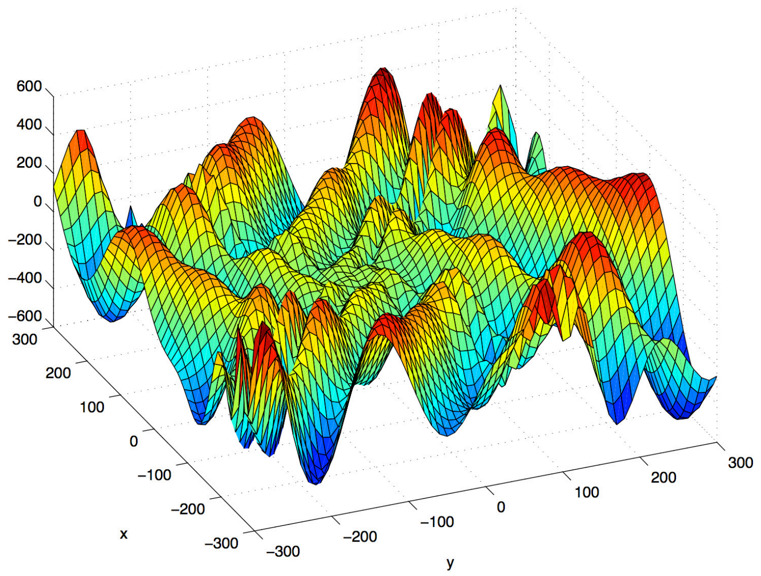


Figure 10.4: Shape of the egg holder function

We solve (10.1) with this $\phi(x, y)$ and with $X = [-300, 300]$,and $Y = [-300, 300]$. The solution was approximately found to be $(x^*, y^*) = (-156.04, 164.16)$. The initial column is chosen as the first column of the A matrix from the Matrix Method of Outer Approximations. Experimental results are presented in Table 10.5, for both $r = 10$ and $r = 2$. In this experiment, the total time required to obtain similar precision results was slightly faster with $r = 10$ than with $r = 2$, for the same reasons as in the previous experiments.

10.4.4 Computational Results

Table 10.3: Numerical experiment result: McCormick function

$r = 2$

Interval	MOA			BA	$\frac{\text{BA}}{\text{MOA}}$
	Iters	Comp.Time	Solutions	Comp.Time	
0.1	2	0.0004	(1.3, -8)	0.002	5
0.1×0.5	2	0.0007	(1.2, -8)	0.07	100
0.1×0.5^2	1	0.001	(1.27, 8)	0.1	100
0.1×0.5^3	1	0.001	(1.26, 8)	0.14	140
0.1×0.5^4	1	0.002	(1.268, 8)	0.5	250
0.1×0.5^5	1	0.004	(1.267, 8)	2.3	575
0.1×0.5^6	1	0.008	(1.268, 8)	9.3	1162
0.1×0.5^7	1	0.01	(1.268, 8)	37.3	3750
0.1×0.5^8	1	0.03	(1.268, 8)	149.4	4980
0.1×0.5^9	1	0.05	(1.268, 8)	540*	10800
0.1×0.5^{10}	1	0.07	(1.2680, 8)	39 mins*	3.3×10^4
0.1×0.5^{11}	1	0.15	(1.2680, 8)	2 hrs*	4.8×10^4
0.1×0.5^{12}	1	0.29	(1.26841, 8)	10 hrs*	1.2×10^5
0.1×0.5^{13}	1	0.54	(1.26842, 8)	1.87 days*	3×10^5
0.1×0.5^{14}	1	1.23	(1.26841, 8)	7 days*	5×10^5
0.1×0.5^{15}	1	2.27	(1.26841, 8)	23.6 days*	9×10^5
0.1×0.5^{16}	1	5.23	(1.268414, 8)	3.7 months*	1.8×10^6
0.1×0.5^{17}	1	18.27	(1.2684142, 8)	1.3 yrs*	2.2×10^6

$r = 10$

Interval	MOA			BA	$\frac{\text{BA}}{\text{MOA}}$
	Iters	Comp.Time	Solutions	Comp.Time	
0.1	2	0.0014	(1.3, -8)	0.002	1.42
0.1^2	2	0.0045	(1.27, -8)	0.05	11.1
0.1^3	1	0.0094	(1.268, 8)	42.5	4.5×10^3
0.1^4	1	0.0754	(1.2684, 8)	38 mins	3×10^4
0.1^5	1	0.65	(1.26841, 8)	72 hrs*	4×10^5
0.1^6	1	6.47	(1.268414, 8)	8.8 months*	3.57×10^6

* estimated values

Table 10.4: Numerical experiment result: testing function

$r = 2$

Interval	MOA			BA	$\frac{\text{BA}}{\text{MOA}}$
	Iters	Comp.Time	Solutions	Comp.Time	
0.1	5	0.0009	(3.4, -5.7)	0.002	2.22
0.1×0.5	4	0.002	(3.4, -5.2)	0.1	18.5
0.1×0.5^2	4	0.001	(3.41, -5.23)	0.2	100
0.1×0.5^3	4	0.001	(3.41, -5.23)	0.3	192
0.1×0.5^4	4	0.001	(3.412, -5.232)	1.36	412
0.1×0.5^5	4	0.003	(3.414, -5.233)	8.46	918
0.1×0.5^6	4	0.005	(3.414, -5.233)	28.73	1686
0.1×0.5^7	4	0.008	(3.4141, -5.2332)	117	3588
0.1×0.5^8	4	0.021	(3.4141, -5.2334)	445	7062
0.1×0.5^9	4	0.075	(3.4142, -5.2334)	1386 *	1×10^4
0.1×0.5^{10}	4	0.11	(3.4142, -5.2334)	12400*	2.7×10^4
0.1×0.5^{11}	4	0.24	(3.41420, -5.23348)	13 hrs*	3×10^4
0.1×0.5^{12}	4	1.23	(3.41421, -5.23346)	1.5 days*	5.7×10^4
0.1×0.5^{13}	4	2.98	(3.41421, -5.23346)	4.8 days*	8.6×10^4
0.1×0.5^{14}	4	6.34	(3.41421, -5.23347)	21 days*	2×10^5
0.1×0.5^{15}	4	7.2	(3.414213, -5.233466)	4.4 months*	3×10^5
0.1×0.5^{16}	4	10.34	(3.414213, -5.233466)	11.7 months*	8×10^5
0.1×0.5^{17}	4	19.72	(3.4142137, -5.2334662)	4.1 yrs*	1.1×10^6

$r = 10$

Interval	MOA			BA	$\frac{\text{BA}}{\text{MOA}}$
	Iters	Comp.Time	Solutions	Comp.Time	
0.1	5	0.0008	(3.4, -5.7)	0.002	2.2
0.1^2	4	0.003	(3.41, -5.24)	0.191	100
0.1^3	4	0.02	(3.414, -5.233)	87.34	918
0.1^4	4	0.17	(3.4142, -5.2334)	1.68 hrs*	1×10^4
0.1^5	4	2.69	(3.41421, -5.23346)	1.8 days*	8.6×10^4
0.1^6	4	18.11	(3.414213, -5.233466)	3.3 yrs*	1×10^6

* estimated values

Table 10.5: Numerical experiment result: Eggholder function

$r = 2$

Interval	MOA			BA	$\frac{\text{BA}}{\text{MOA}}$
	Iters	Evals	Solutions	Evals	
0.1	8	0.05	(-156, -200.3)	0.17	3.57
0.1×0.5	8	0.365	(-156.05, 164.1)	0.16	22.5
0.1×0.5^2	9	5.4	(-156.05, 164.15)	0.16	160
0.1×0.5^3	13	53	(-156.05, 164.175)	0.66	220
0.1×0.5^4	17	0.005	(-156.037, 164.175)	2.66	532
0.1×0.5^5	18	0.009	(-156.037, 164.175)	10.56	1173
0.1×0.5^6	18	0.019	(-156.037, 164.175)	42.69	2246
0.1×0.5^7	18	0.038	(-156.037, 164.175)	168	4421
0.1×0.5^8	18	0.096	(-156.037, 164.175)	678	7062
0.1×0.5^9	18	0.17	(-156.0371, 164.1711)	2580 *	1.5×10^4
0.1×0.5^{10}	18	0.29	(-156.0373,164.1652)	10800*	3.7×10^4
0.1×0.5^{11}	18	1.17	(-156.0375,164.1656)	16 hrs*	5×10^4
0.1×0.5^{12}	18	2.32	(-156.03984, 164.16386)	2 days*	7.4×10^4
0.1×0.5^{13}	18	4.54	(-156.03984, 164.16386)	6.3 days*	1.2×10^5
0.1×0.5^{14}	18	9.19	(-156.03984, 164.16386)	32 days*	3×10^5
0.1×0.5^{15}	18	18.3	(-156.039843, 164.163867)	5.64 months*	8×10^5
0.1×0.5^{16}	18	25.19	(-156.039843, 164.163867)	1.4 yrs*	1.7×10^6
0.1×0.5^{17}	18	38.37	(-156.0398437, 164.1638671)	5.7 yrs*	4.6×10^6

$r = 10$

Interval	MOA			BA	$\frac{\text{BA}}{\text{MOA}}$
	Iters	Comp.Time	Solutions	Comp.Time	
0.1	5	0.0008	(-156, -200.3)	0.002	2.5
0.1^2	4	0.003	(-156.04, 164.16)	0.184	61.3
0.1^3	4	0.03	(-156.04, 164.164)	104.23	3.4×10^4
0.1^4	4	0.28	(-156.0398, 164.1638)	2.89 hrs*	8.6×10^4
0.1^5	4	3.69	(-156.03982, 164.16384)	2.58 days*	2.8×10^5
0.1^6	4	27.71	(-156.03981, 164.163848)	2.9 yrs*	3.7×10^6

* estimated values

10.5 Conclusion

We have presented a new scheme for solving matrix games that arise in pursuit-evation, global optimization, and, likely, in other problems as well. As is evident from the numerical results, when the games are large, the use of our algorithm results in many orders of magnitude reduction in computational time over any method that requires the evaluation of all the elements of the A matrix.

CHAPTER 11

CONCLUSION

We have shown following in previous chapters.

1. The minmax MPC problem can be efficiently solved by separating the problem into two sequences of subproblems: inner and outer subproblems. The inner subproblem utilizes the Phase I-Phase II method approach and the method of the outer approximation is adopted in the other subproblem. We specially design an optimality function so that our solution method satisfies recursive feasibility. Also, our method is based on quadratic programming and is therefore computationally efficient enough to be implemented in real-time.

2. A minmax MPC problem called harbor defense problem is solved using the method that we developed. The basic harbor defense problem which includes a single intruder and defender was presented. Both 2D and 3D human-interactive simulations are developed.

3. The basic harbor defense problem was extended to various advanced problems. We presented the harbor defense problem with multiple numbers of defenders and intruders. Various strategies of the intruder such as suicidal, risk-averse, and max-min are considered as well. Simulation-based analyses are presented.

4. We implemented our minmax MPC scheme on a real-time testbeds and demonstrated the effectiveness of the solution. The first testbed is the custom-built robotic testbed called HoTDeC (Hovercraft Testbed for Decentralized Control). The development of the testbed includes not only the HoTDeC itself, but also network architecture in the laboratory. Each development step is elaborated in this dissertation. Next, we test our minmax MPC scheme on full-sized US Naval Academy patrol ships. An experiment was conducted in the Chesapeake Bay in collaboration with the US Naval Academy. One Navy patrol is a human-driven intruder and the other patrol ship is autonomously commanded by our minmax MPC algorithm. The

results of several experiments were presented.

5. We presented another example of the minmax MPC problem called mobile network jamming problem. We presented the solution, simulation as well as the necessary and sufficient conditions for the jamming to occur.
6. We showed that the numerical optimization technique that we adopt to solve the minmax MPC problem constructs a very efficient method for finding approximations to the global solutions of minmax problems in the matrix form as well. We presented examples that show the efficiency of the method.

APPENDIX A

EXACT INTEGRATION OF THE DYNAMIC MODEL

Suppose that we are interested in following dynamics:

$$\begin{aligned} \dot{x}^1(t) &= v(t)\cos\theta(t) \\ \dot{x}^2(t) &= v(t)\sin\theta(t) \\ \dot{\theta}(t) &= \sigma(t) \\ \dot{v}(t) &= \alpha(t) \end{aligned} \tag{A.1}$$

with input $\sigma(t)$ and input $v(t)$,

$$\begin{aligned} 0 \le v(t) &\le \bar{v} \\ |\alpha(t)| &\le \bar{\alpha} \\ |\sigma(t)| &\le \bar{u} \end{aligned} \tag{A.2}$$

$x^1(t)$,$x^2(t)$ are the position and $\theta(t)$ is a heading angle of the water vehicle in global coordination. (A.1) is viewed as a Dubins car with a varying speed.

Exact discretization of nonlinear model

In this sub section, let us notate $k\Delta$ as subscript k for the conciseness. For example, $x(k\Delta)$ is denoted as x_k. For $k\Delta \le t < (k+1)\Delta$, $\sigma(t) = \sigma_k$ and $\alpha(t) = \alpha_k$. Hence, for $k\Delta \le t < (k+1)\Delta$, $\theta(t) = \theta_k + \Delta\sigma_k$ and $v(t) = v_k + \Delta\alpha_k$. Now,

$$\begin{aligned} x^1_{k+1} &= x^1_k + \int_{k\Delta}^{(k+1)\Delta} v(t)\cos\theta(t)dt \\ x^2_{k+1} &= x^2_k + \int_{k\Delta}^{(k+1)\Delta} v(t)\sin\theta(t)dt \end{aligned} \tag{A.3}$$

Since $\theta(t) = \theta_k + t\sigma_k$ and $v(t) = v_k + t\alpha_k$ for $k\Delta \le t \le (k+1)\Delta$, $d\theta(t) = \sigma_k dt$ and $dv = \alpha_k dt$. Also,

$$\begin{aligned}\int_{k\Delta}^{(k+1)\Delta} \cos\theta(t)dt &= \int_{k\Delta}^{(k+1)\Delta} \cos(\theta_k + t\sigma_k)dt = \int_{\theta_k}^{\theta_{k+1}} \cos\theta(t)\frac{1}{\sigma_k}d\theta(t) \\ &= \frac{1}{\sigma_k}\left\{\sin\theta_{k+1} - \sin\theta_k\right\}\end{aligned} \tag{A.4}$$

Similarly,

$$\begin{aligned}\int_{k\Delta}^{(k+1)\Delta} \sin\theta(t)dt &= \int_{k\Delta}^{(k+1)\Delta} \sin(\theta_k + t\sigma_k)dt = \int_{\theta_k}^{\theta_{k+1}} \sin\theta(t)\frac{1}{\sigma_k}d\theta(t) \\ &= -\frac{1}{\sigma_k}\left\{\cos\theta_{k+1} - \cos\theta_k\right\}\end{aligned} \tag{A.5}$$

From the integration by part,

$$\int_{k\Delta}^{(k+1)\Delta} v(t)\cos\theta(t)dt = \left[v(t)\int \cos\theta(t)dt\right]_{k\Delta}^{(k+1)\Delta} - \alpha_k \int_{k\Delta}^{(k+1)\Delta}\left(\int \cos\theta(t)dt\right)dt \tag{A.6}$$

Since

$$\int \cos\theta(t)dt = \frac{1}{\sigma_k}\int \cos\theta(t)d\theta = \frac{1}{\sigma_k}\sin\theta(t) + c \tag{A.7}$$

(A.6) becomes

$$\begin{aligned}&\int_{k\Delta}^{(k+1)\Delta} v(t)\cos\theta(t)dt = \left[v(t)\frac{1}{\sigma_k}\sin\theta(t)\right]_{k\Delta}^{(k+1)\Delta} - \alpha_k\int_{k\Delta}^{(k+1)\Delta}\left(\frac{1}{\sigma_k}\sin\theta(t)\right)dt \\ &= v_{k+1}\left(\frac{1}{\sigma_k}\sin\theta_{k+1}\right) - v_k\left(\frac{1}{\sigma_k}\sin\theta_k\right) - \alpha_k\int_{k\Delta}^{(k+1)\Delta}\frac{1}{\sigma_k}\sin\theta(t)dt \\ &= \frac{1}{\sigma_k}\left\{v_{k+1}\sin\theta_{k+1} - v_k\sin\theta_k\right\} + \frac{\alpha_k}{\sigma_k^2}\left\{\cos\theta_{k+1} - \cos\theta_k\right\}\end{aligned} \tag{A.8}$$

Therefore,

$$x_{k+1}^1 = x_k^1 + \frac{1}{\sigma_k}\left\{v_{k+1}\sin\theta_{k+1} - v_k\sin\theta_k\right\} + \frac{\alpha_k}{\sigma_k^2}\left\{\cos\theta_{k+1} - \cos\theta_k\right\} \tag{A.9}$$

Similarly,

$$x_{k+1}^2 = x_k^2 - \frac{1}{\sigma_k}\left\{v_{k+1}\cos\theta_{k+1} - v_k\cos\theta_k\right\} + \frac{\alpha_k}{\sigma_k^2}\left\{\sin\theta_{k+1} - \sin\theta_k\right\} \tag{A.10}$$

Hence, we have

$$\begin{aligned}
\theta_{k+1} &= \theta_k + \Delta\sigma_k \\
v_{k+1} &= v_k + \Delta\alpha_k \\
x^1_{k+1} &= x^1_k + \frac{1}{\sigma_k}\left\{v_{k+1}\sin\theta_{k+1} - v_k\sin\theta_k\right\} + \frac{\alpha_k}{\sigma_k^2}\left\{\cos\theta_{k+1} - \cos\theta_k\right\} \\
x^2_{k+1} &= x^2_k - \frac{1}{\sigma_k}\left\{v_{k+1}\cos\theta_{k+1} - v_k\cos\theta_k\right\} + \frac{\alpha_k}{\sigma_k^2}\left\{\sin\theta_{k+1} - \sin\theta_k\right\}
\end{aligned} \tag{A.11}$$

(A.11) has singularity when $\sigma_k = 0$. Therefore, it needs to be transformed. First,

$$\begin{aligned}
\frac{1}{\sigma_k}\left\{v_{k+1}\sin\theta_{k+1} - v_k\sin\theta_k\right\} &= \frac{1}{\sigma_k}\left\{(v_k + \Delta\alpha_k)\sin(\theta_k + \Delta\sigma_k) - v_k\sin\theta_k\right\} \\
&= \frac{1}{\sigma_k}\left\{v_k\left[\sin(\theta_k + \Delta\sigma_k) - \sin\theta_k\right] + \Delta\alpha_k\sin(\theta_k + \Delta\sigma_k)\right\} \\
&= \frac{v_k}{\sigma_k}\left[\sin(\theta_k + \Delta\sigma_k) - \sin\theta_k\right] + \frac{\Delta\alpha_k}{\sigma_k}\sin(\theta_k + \Delta\sigma_k)
\end{aligned} \tag{A.12}$$

For small Δ,

$$\begin{aligned}
\frac{v_k}{\sigma_k}\left[\sin(\theta_k + \Delta\sigma_k) - \sin\theta_k\right] &= \frac{v_k\Delta\sigma_k}{\sigma_k}\frac{\left[\sin(\theta_k + \Delta\sigma_k) - \sin\theta_k\right]}{\Delta\sigma_k} \\
&= \frac{v_k\Delta\sigma_k}{\sigma_k}\cos\theta_k \\
&= v_k\Delta\cos\theta_k
\end{aligned} \tag{A.13}$$

Therefore, (A.12) becomes

$$\begin{aligned}
\frac{1}{\sigma_k}\left\{v_{k+1}\sin\theta_{k+1} - v_k\sin\theta_k\right\} &= \frac{v_k}{\sigma_k}\left[\sin(\theta_k + \Delta\sigma_k) - \sin\theta_k\right] + \frac{\Delta\alpha_k}{\sigma_k}\sin(\theta_k + \Delta\sigma_k) \\
&= v_k\Delta\cos\theta_k + \frac{\Delta\alpha_k}{\sigma_k}\sin(\theta_k + \Delta\sigma_k)
\end{aligned} \tag{A.14}$$

Also, in (A.11)

$$\begin{aligned}
\frac{\alpha_k}{\sigma_k^2}\left\{\cos\theta_{k+1} - \cos\theta_k\right\} &= \frac{\alpha_k}{\sigma_k^2}\frac{\left\{\cos\theta_{k+1} - \cos\theta_k\right\}}{\Delta\sigma_k}\Delta\sigma_k \\
&= -\frac{\Delta\alpha_k}{\sigma_k}\sin\theta_k
\end{aligned} \tag{A.15}$$

Plug (A.14) and (A.15) in (A.11) yields,

$$\begin{aligned}
x_{k+1}^1 &= x_k^1 + \frac{1}{\sigma_k}\left\{v_{k+1}\sin\theta_{k+1} - v_k\sin\theta_k\right\} + \frac{\alpha_k}{\sigma_k^2}\left\{\cos\theta_{k+1} - \cos\theta_k\right\} \\
&= x_k^1 + v_k\Delta\cos\theta_k + \frac{\Delta\alpha_k}{\sigma_k}\sin(\theta_k + \Delta\sigma_k) - \frac{\Delta\alpha_k}{\sigma_k}\sin\theta_k \\
&= x_k^1 + v_k\Delta\cos\theta_k + \frac{\Delta\alpha_k}{\sigma_k}\frac{[\sin(\theta_k + \Delta\sigma_k) - \sin(\theta_k)]}{\Delta\sigma_k}\Delta\sigma_k \\
&= x_k^1 + v_k\Delta\cos\theta_k + \Delta^2\alpha_k\cos\theta_k
\end{aligned} \tag{A.16}$$

Similarly,

$$\begin{aligned}
x_{k+1}^2 &= x_k^2 - \frac{1}{\sigma_k}\left\{v_{k+1}\cos\theta_{k+1} - v_k\cos\theta_k\right\} + \frac{\alpha_k}{\sigma_k^2}\left\{\sin\theta_{k+1} - \sin\theta_k\right\} \\
&= x_k^2 + v_k\Delta\sin\theta_k + \Delta^2\alpha_k\sin\theta_k
\end{aligned} \tag{A.17}$$

Therefore, (A.11) is transformed to

$$\begin{aligned}
\theta_{k+1} &= \theta_k + \Delta\sigma_k \\
v_{k+1} &= v_k + \Delta\alpha_k \\
x_{k+1}^1 &= x_k^1 + v_k\Delta\cos\theta_k + \Delta^2\alpha_k\cos\theta_k \\
x_{k+1}^2 &= x_k^2 + v_k\Delta\sin\theta_k + \Delta^2\alpha_k\sin\theta_k
\end{aligned} \tag{A.18}$$

APPENDIX B

GRADIENT COMPUTATION

First we need to discretize the dynamics

$$\dot{z} = Az + Bu \tag{B.1}$$

Apply the input that changes only at discrete sampling intervals. It is known that the solution of (B.1) is

$$z(t) = e^{At}z(0) + e^{At}\int_0^t e^{-A\tau}Bu(\tau)d\tau \tag{B.2}$$

Therefore, if we replace t to $k\Delta$, we obtain

$$z(k\Delta) = e^{Ak\Delta}z(0) + e^{Ak\Delta}\int_0^{k\Delta} e^{-A\tau}Bu(\tau)d\tau \tag{B.3}$$

Similarly, we have

$$z((k+1)\Delta) = e^{A(k+1)\Delta}z(0) + e^{A(k+1)\Delta}\int_0^{(k+1)\Delta} e^{-A\tau}Bu(\tau)d\tau \tag{B.4}$$

Since we want to write difference equation, we need to express $z((k+1)\Delta)$ in terms of $z(k\Delta)$. Hence multiply $e^{A\Delta}$ to (B.3) and solve for $e^{A(k+1)\Delta}z(0)$. Then substitute $e^{A(k+1)\Delta}z(0)$ in (B.4), we obtain

$$\begin{aligned} z((k+1)\Delta) &= e^{A\Delta}z(k\Delta) + e^{A(k+1)\Delta}\left[\int_0^{(k+1)\Delta} e^{-A\tau}Bu(\tau)d\tau - \int_0^{k\Delta} e^{-A\tau}Bu(\tau)d\tau\right] \\ &= e^{A\Delta}z(k\Delta) + e^{A(k+1)\Delta}\int_{k\Delta}^{k(\Delta+1)} e^{-A\tau}Bu(\tau)d\tau \end{aligned} \tag{B.5}$$

Since $u(t) = u(k\Delta)$ is a constant for $t \in [k\Delta, (k+1)\Delta)$,

$$\begin{aligned} z((k+1)\Delta) &= e^{A\Delta} z(k\Delta) + e^{A(k+1)\Delta} \int_{k\Delta}^{k(\Delta+1)} e^{-A\tau} d\tau B u(k\Delta) \\ &= e^{A\Delta} z(k\Delta) + \int_{k\Delta}^{k(\Delta+1)} e^{A[(k+1)\Delta - \tau]} d\tau B u(k\Delta) \\ \tau &\in [k\Delta, (k+1)\Delta) \end{aligned} \tag{B.6}$$

Let $\lambda = (k+1)\Delta - \tau$ then we obtain

$$\begin{aligned} z((k+1)\Delta) = e^{A\Delta} z(k\Delta) + \int_{0}^{\Delta} e^{A\lambda} d\lambda B u(k\Delta) \\ \lambda \in [0, k\Delta) \end{aligned} \tag{B.7}$$

Now, (B.7) is a difference equation in a form of

$$z((k+1)\Delta) = \bar{A}(\Delta)(\Delta) z(k\Delta) + \bar{B}(\Delta) u(k\Delta) \tag{B.8}$$

We see that $\bar{A}$ and $\bar{B}$ are function of Δ but we abuse the notation by simply $\bar{A}$ and $\bar{B}$.

Let us consider the gradient. z is a 4 by 1 column matrix. such that

$$z = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} \tag{B.9}$$

If we consider N steps of dynamics, z is 4 by N matrix. Let us define design vector $\mathbf{w}$ with $u = [u_1, u_2]^T$.

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{3N-1} \end{bmatrix} = \begin{bmatrix} u_1(B.1) \\ u_2(B.1) \\ u_1(B.2) \\ u_2(B.2) \\ u_1(B.3) \\ u_2(B.3) \\ \vdots \\ u_1(N) \\ u_2(N) \\ \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \\ \mu_{N-1} \end{bmatrix} \tag{B.10}$$

This is an attacker case with μ. In defender case, size of $\mathbf{w}$ is $2N$ by 1. Let us take derivative of (B.8) with respect to $\mathbf{w}$.

$$\frac{\partial z((k+1)\Delta)}{\partial w} = \bar{A}\frac{\partial z(k\Delta)}{\partial w} + \bar{B}\frac{\partial u(k\Delta)}{\partial w} \tag{B.11}$$

Let Jacobian matrix GZ. It is determined by iteration. When $k = 1$,

$$\frac{\partial z(2\Delta)}{\partial w} = \bar{A}\frac{\partial z(\Delta)}{\partial w} + \bar{B}\frac{\partial u(\Delta)}{\partial w} \tag{B.12}$$

Assuming the initial gradient of z is zero,

$$\frac{\partial z(2\Delta)}{\partial w} = \bar{B}\frac{\partial u(\Delta)}{\partial w} \tag{B.13}$$

$$\frac{\partial u(\Delta)}{\partial w} = \begin{bmatrix} \frac{\partial u_1(\Delta)}{\partial w_1}, \frac{\partial u_1(\Delta)}{\partial w_2}, \frac{\partial u_1(\Delta)}{\partial w_3}, \cdots, \frac{\partial u_1(\Delta)}{\partial w_{3N-1}} \\ \frac{\partial u_2(\Delta)}{\partial w_1}, \frac{\partial u_2(\Delta)}{\partial w_2}, \frac{\partial u_2(\Delta)}{\partial w_3}, \cdots, \frac{\partial u_2(\Delta)}{\partial w_{3N-1}} \end{bmatrix} \tag{B.14}$$

Since

$$\begin{aligned} \frac{\partial u_1(\Delta)}{\partial w_1} &= \frac{\partial u_1(\Delta)}{\partial u_1(\Delta)} = 1 \\ \frac{\partial u_2(\Delta)}{\partial w_2} &= \frac{\partial u_2(\Delta)}{\partial u_2(\Delta)} = 1 \end{aligned} \tag{B.15}$$

(B.14) becomes $[I_{2,2}, 0_{2,3N-3}]$. Process is iterated until $k = N$.

APPENDIX C

CIRCUIT SCHEMATICS

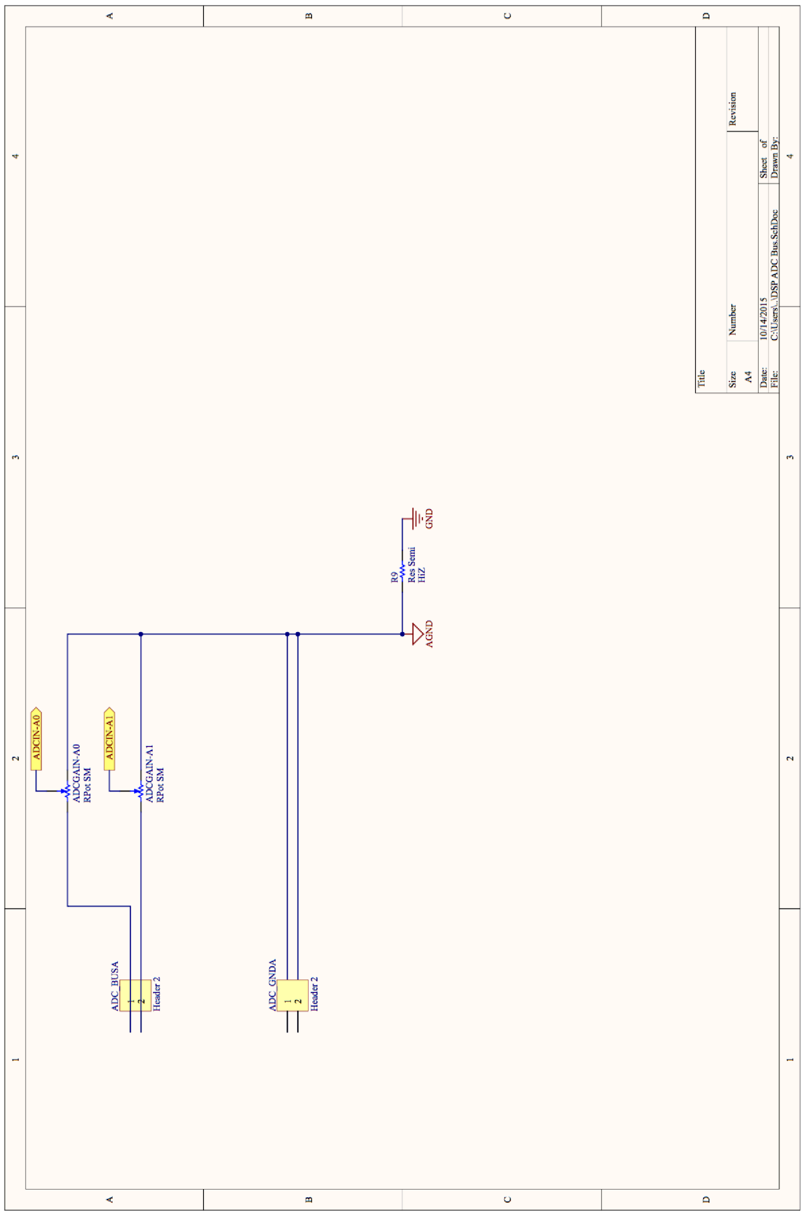


Figure C.1: ADC bus for DSP

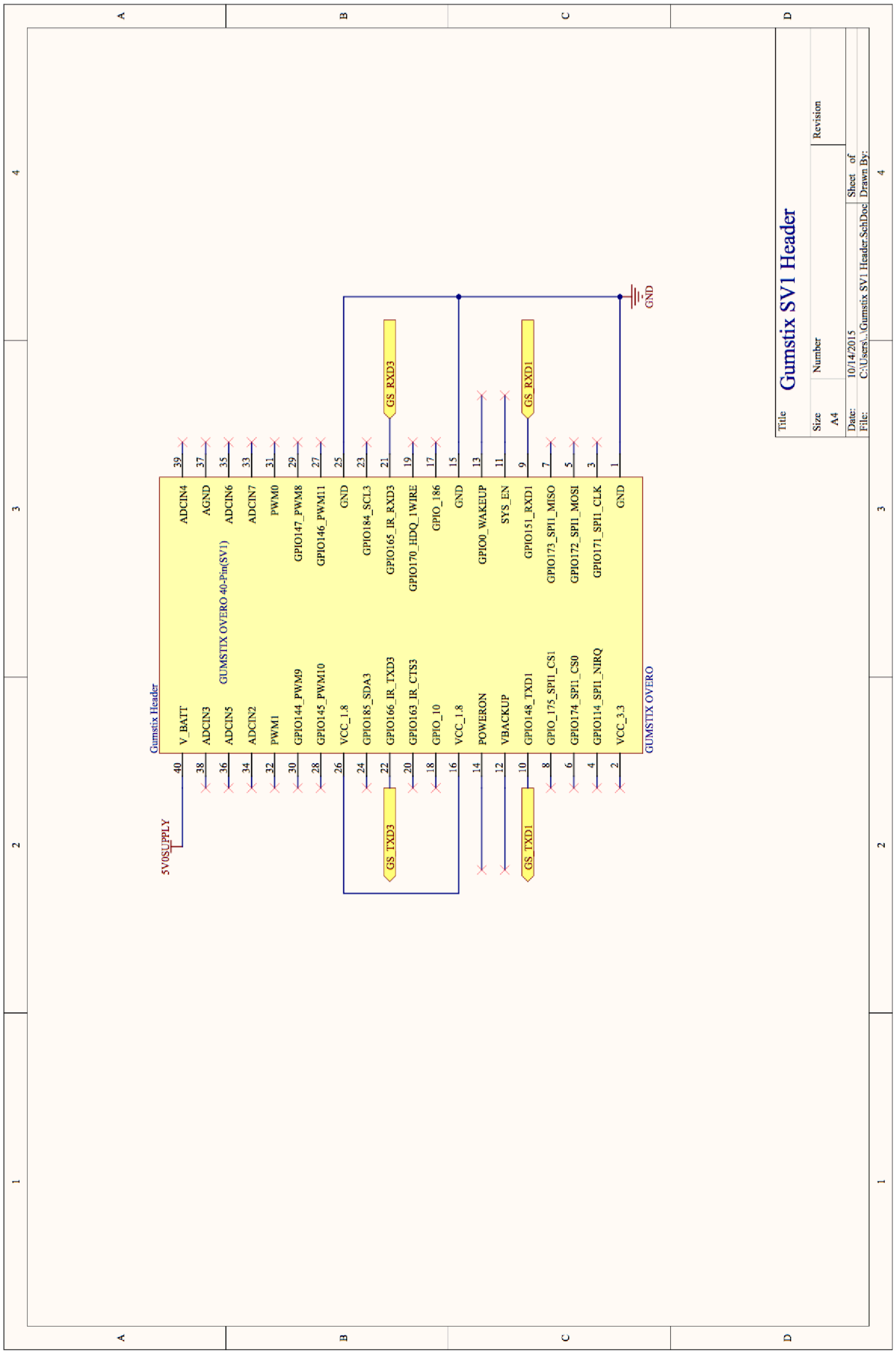


Figure C.2: Gumsitx connection to the main board

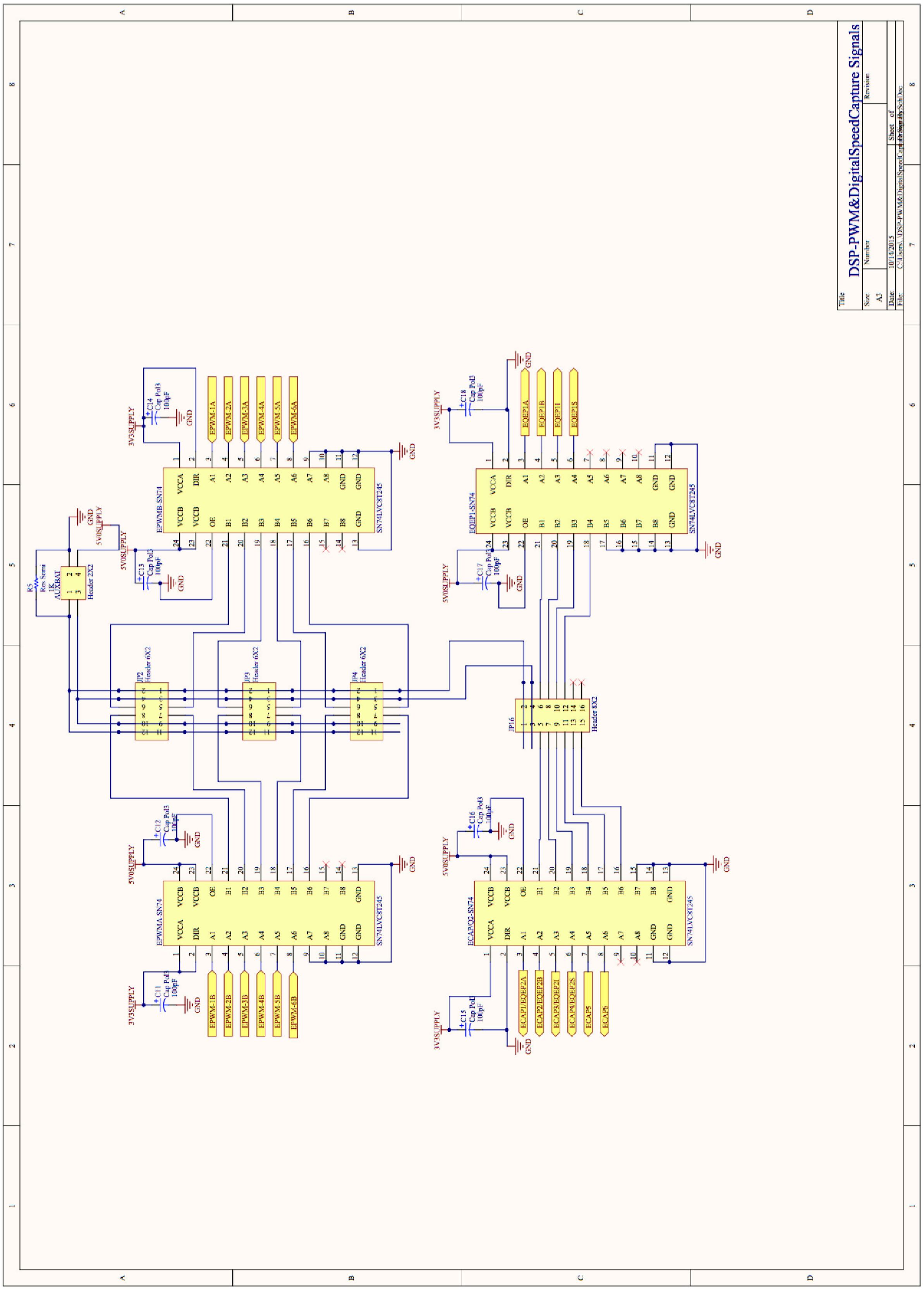


Figure C.3: DSP PWM signal to the Thrusters

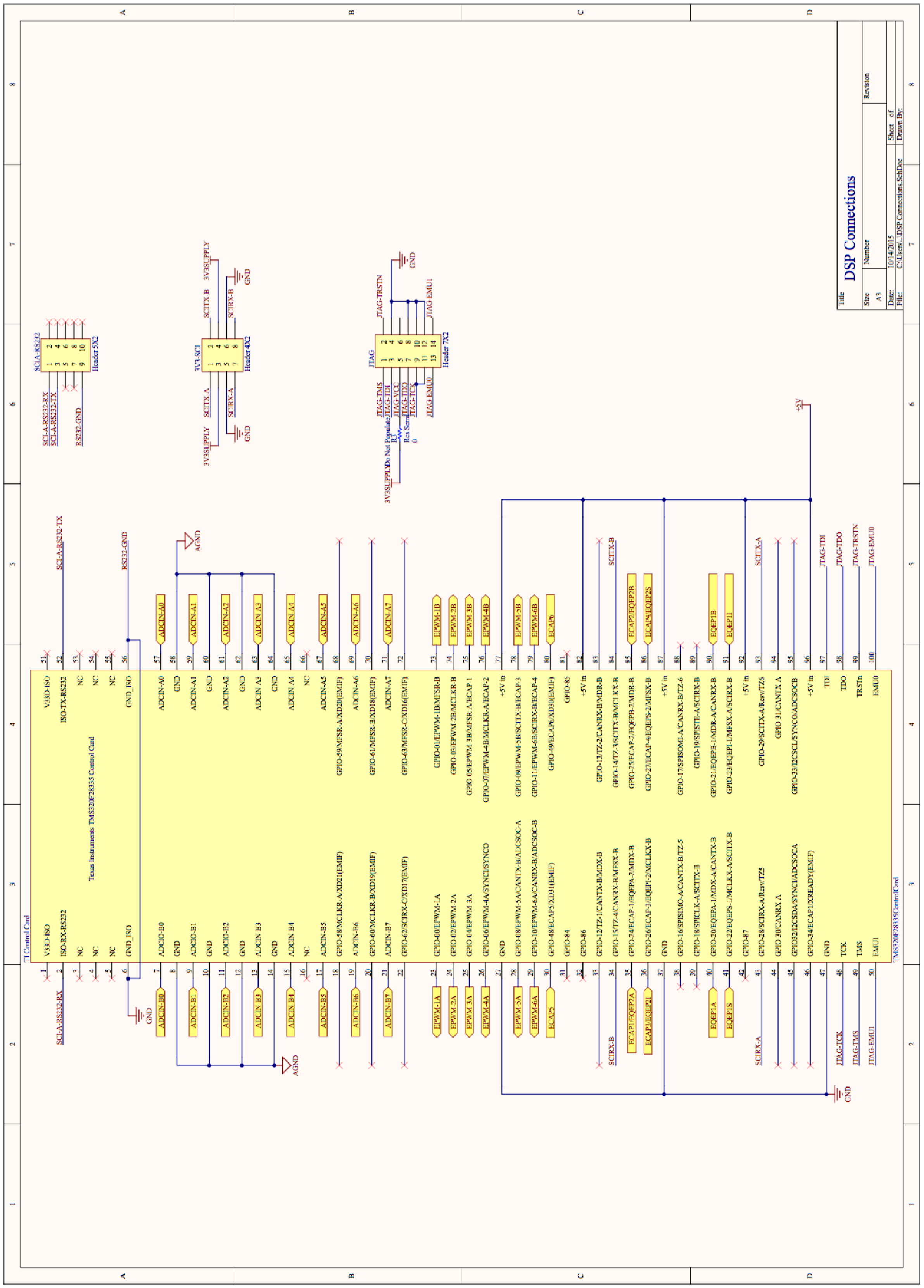


Figure C.4: DSP connection to the main board

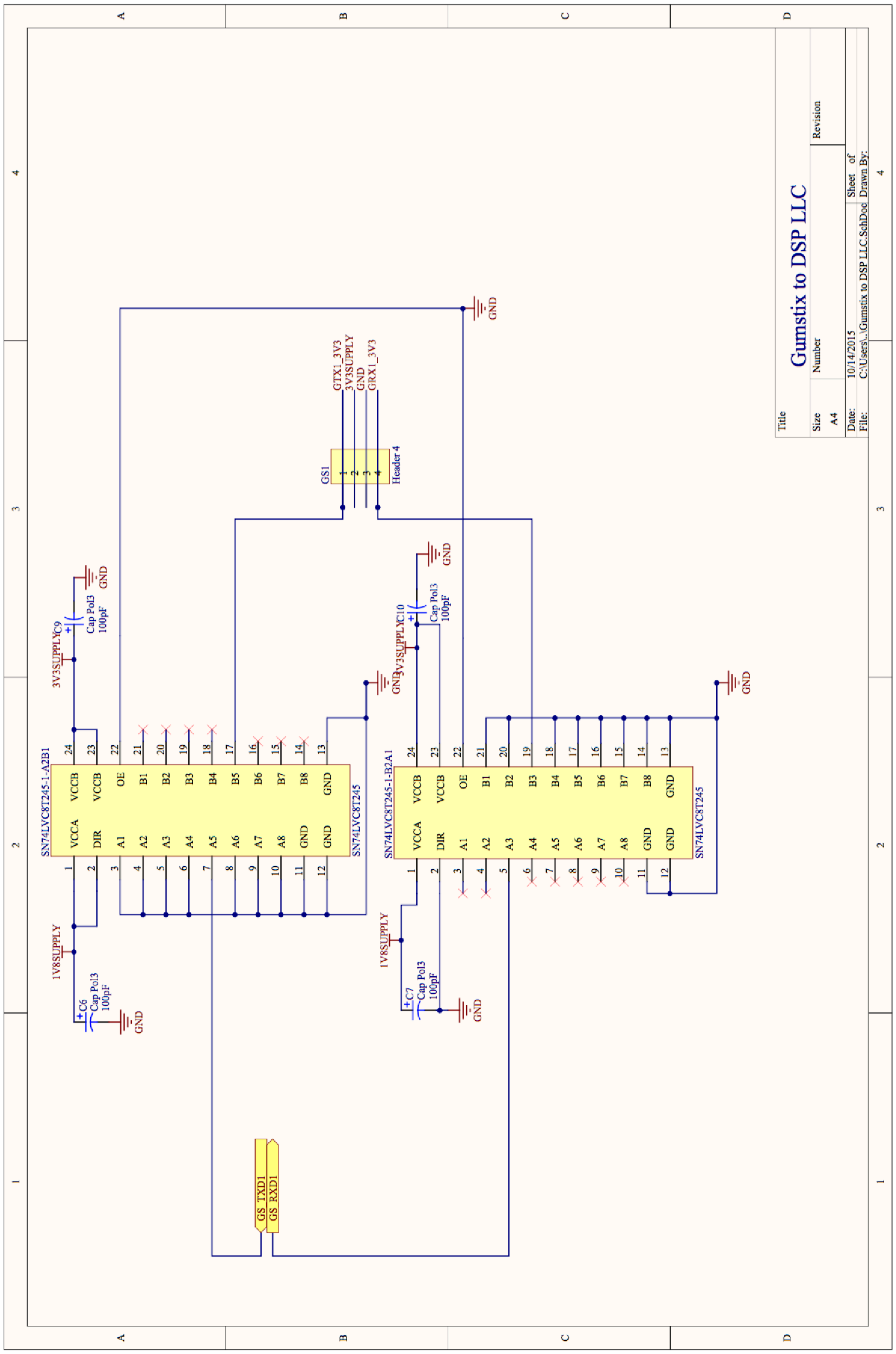


Figure C.5: Connection between Gumstix and DSP

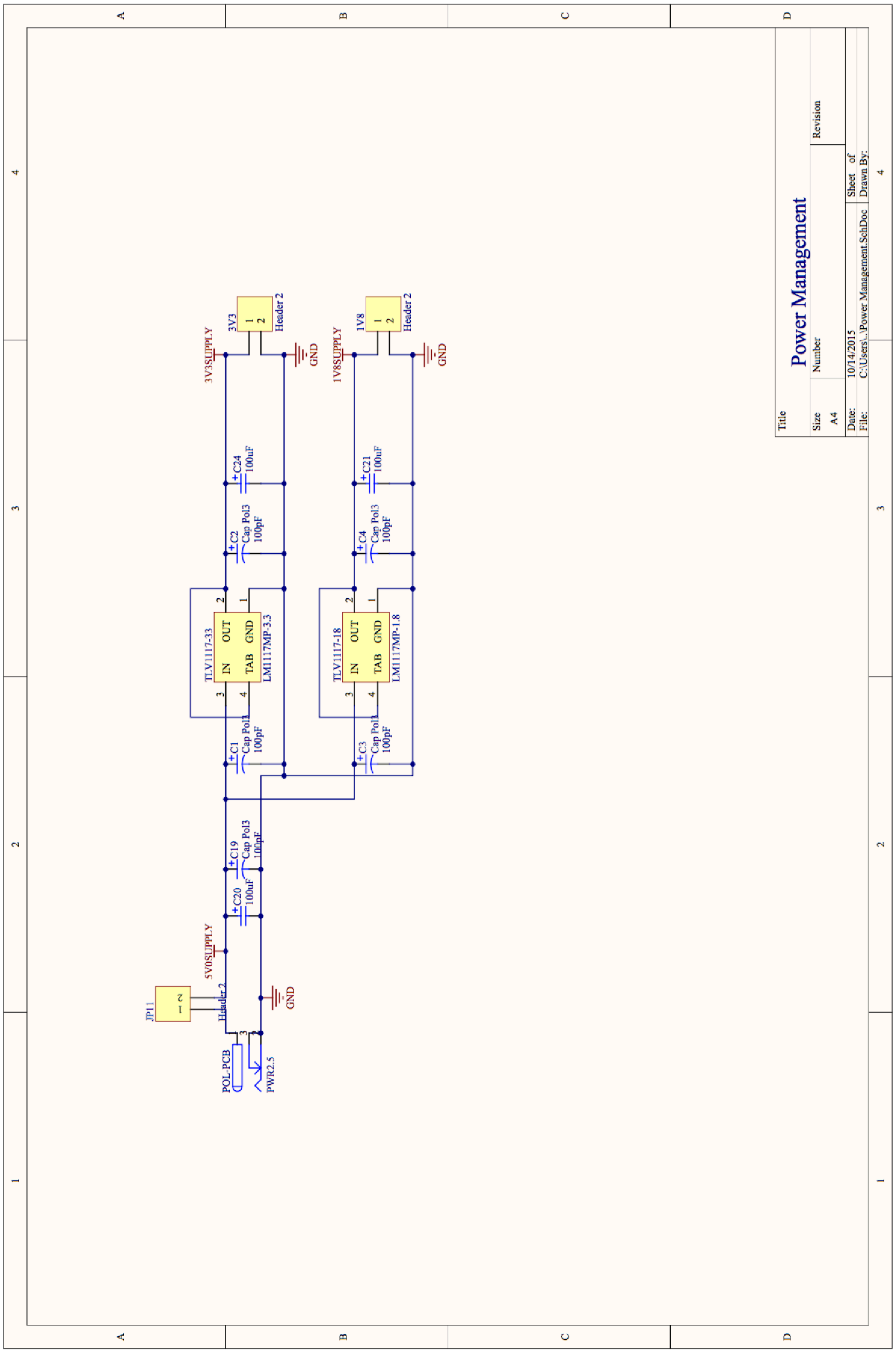


Figure C.6: Main board power

APPENDIX D

TESTBED MANUAL

In this appendix, the procedure for running the HoTDeC and related systems and programs are explained.

D.1 Vision system

The vision system must be run properly before operating the HoTDeC. The IP for vision clients are reserved to 192.168.0.101 to 192.168.0.103. Vision server IP is 192.168.0.3. To run vision clients, SSH into vision client and run following command.

```
/home/rotap2/projects/testbed/vision/vision_client_reloaded$ ./client #
```

The symbol # is the number of vision clients (1,2, or 3). Once all three vision clients are running, run the vision server by SSH into 192.168.0.3 and run the following commands

```
~/testbed/trunk/vision$python vision.py
~/testbed/trunk/vision$ruby vision.rb
```

There are two kinds of vision server programs: one written in python, the other one written in ruby. The python program is reserved to the port 6968, and the ruby one is reserved to the port 6969. Ruby server runs in slightly faster frequency. Either of these program output should display the current HoTDeC position and orientation on the laboratory floor. One way to check if all vision programs are properly running is to check the output of the vision server program.

The visualizer that displays the information about HoTDeC can be run as follow.

```
~/$python vision-ui.py
```

Note that the visualizer requires ZMQ, PyZMQ, and QT. Identifiers for the HoTDeC are reserved from 1 to 9. The number 10 is reserved for the yellow blob.

D.2 The operation of the HoTDeC

To run the HoTDeC, first log on to the target HoTDeC. We assume this is the one with the IP of 192.168.0.7. Open the terminal and SSH into it, go to the serial directory and run the following serial server program.

```
user@overo: /gumstix/serial$ ruby serial-serv.rb 192.168.0.7:6968
```

Now, HoTDeC is ready to send the message to the thrusters through the serial communication.

In the computer where the application programs are, run the following simulator program using JAVA.

```
java -Djava.library.path=/usr/local/lib -classpath zmq.jar:jackson-all-1.9.5.jar:./ World
```

This is the program that contains HoTDeC model and the most upper class of the JAVA programs. Once this runs properly, run the controller which contains LQR controller and Kalman filter.

```
java -Djava.library.path=/usr/local/lib -classpath zmq.jar:jackson-all-1.9.5.jar:./ Controller
```

Now you are ready to run the applications such as Harbor defense algorithm or manual way point input command. Manual way point input program can be run as follows.

```
java -Djava.library.path=/usr/local/lib -classpath zmq.jar:jackson-all-1.9.5.jar:./ ReqRep
```

Defender program can be run as follow.

```
java -Djava.library.path=/usr/local/lib -classpath zmq.jar:jackson-all-1.9.5.jar:./ Seungho
```

Following is the program that allows us to test the thrusters manually

```
GSTest$ ./ Thuster (IP should changed accordingly)
```

Following is the visualizer that shows the position and orientation of the HoTDeC real-time.

```
testbed/vision$ python vision-ui.py
```

You can also visualize the references on the top of the actual trajectory by setting the port to 6688, which is reserved for the real time reference generator.

```
req-rep visualizer: myIp:6688
```

The joystick input program can be run as follows. First, log on to the target HoTDeC by SSH. We assume this HoTDeC has the static IP of 192.168.0.8. Due to the ruby version compatibility issue, it is safe to run "use" command to change the ruby version.

```
user@overo: /gumstix/serial$ rvm use 1.9.2
```

Next, run the serial communication using the ruby command which takes joystick input.

```
user@overo: /gumstix/serial$ ruby seriald.rb 192.168.0.8:6968
```

Now, you are ready to run joystick application in the computer where the joystick is attached. Go to the proper directory and run the joystick server program as follows.

```
ztest/testbed/trunk/joystick$ ruby joyserve.rb
```

Then run the joystick program that sends the joystick signal to the HoTDeC.

```
/ztest/gumstix/trunk/joystick$ ruby joystick.rb
```

Also, run the joystick controller that includes LQR and Kalman filter.

```
/ztest/gumstix/trunk/controller$ python controller.py
```

D.3 Software structure

The software structure for solving the harbor defense problem using the Matlab is presented in the following figure. There are many custom-written m files involved in harbor defense problem.

The nodes of the tree in Fig. D.1 are names of m files. The file "batch_moam" is the root program that calls the subprograms "GLOBAL","init", "moam", and "re_init". The file "GLOBAL" contains definitions of global variables. The file "init" includes the initialization part. The role of "re_init" is to update the current state information in discrete time.

The file "moam" is an implementation of the method of the outer approximations. This includes intruder and defender sides separately. The core program of the intruder side is "ph_lsh_i". This program is an implementation of the Phase I-Phase II method in the intruder side. The files "fe" and "grad_fe" are the objective function and its gradient of the intruder. The files "const_f" and "const_g" contains the inequality constraints and their gradients. Subfiles "fecxy","fecu", and "fechn" are the constraints for the minimum separation, control bounds, and channel constraints, respectively. The files "z" and "grad_z" include the model of the intruder and its gradient.

File names of the defender side are similar to the intruder side. The file "ph_lsh_p" is an implementation of the Phase I-Phase II method on the defender side. The matrix "gXYET" contains the trajectories of the intruder. The trajectory of the intruder is inserted into this matrix in each iteration of "moam".

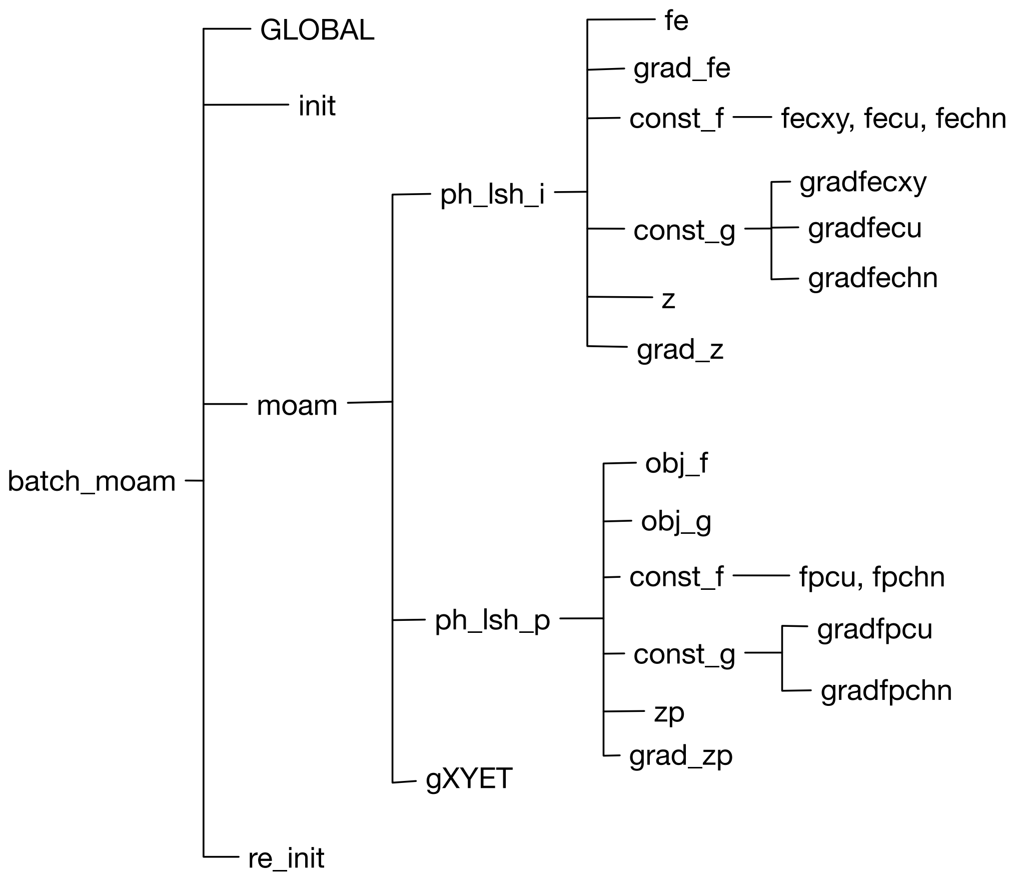


Figure D.1: Harbor defense program structure

REFERENCES

[1] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, “Constrained model predictive control: Stability and optimality,” *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[2] R. Bhattacharya, G. J. Balas, M. A. Kaya, and A. Packard, “Nonlinear receding horizon control of an F-16 aircraft,” *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 5, pp. 924–931, 2002.

[3] J. M. Maciejowski, *Predictive control: with constraints.* Pearson education, 2002.

[4] W. Wang, D. E. Rivera, and K. G. Kempf, “Model predictive control strategies for supply chain management in semiconductor manufacturing,” *International Journal of Production Economics*, vol. 107, no. 1, pp. 56–77, 2007.

[5] E. G. Cho, K. Thoney, T. J. Hodgson, R. E. King, *et al.*, “Rolling horizon scheduling of multi-factory supply chains,” in *Simulation Conference, 2003. Proceedings of the 2003 Winter*, vol. 2, pp. 1409–1416, IEEE, 2003.

[6] W. B. Powell, *Approximate Dynamic Programming: Solving the curses of dimensionality*, vol. 703. John Wiley & Sons, 2007.

[7] H. Dawid, “Long horizon versus short horizon planning in dynamic optimization problems with incomplete information,” *Economic Theory*, vol. 25, no. 3, pp. 575–597, 2005.

[8] F. Herzog, *Strategic portfolio management for long-term investments.* Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 16137, 2005.

[9] J. A. Primbs, “Dynamic hedging of basket options under proportional transaction costs using receding horizon control,” *International Journal of Control*, vol. 82, no. 10, pp. 1841–1855, 2009.

[10] K. T. Talluri and G. J. Van Ryzin, *The theory and practice of revenue management*, vol. 68. Springer Science & Business Media, 2006.

[11] D. Bertsimas and I. Popescu, “Revenue management in a dynamic network environment,” *Transportation science*, vol. 37, no. 3, pp. 257–277, 2003.

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

[13] P. J. Campo and M. Morari, "Robust model predictive control," in *American Control Conference, 1987*, pp. 1021–1026, IEEE, 1987.

[14] J. Löfberg, "Approximations of closed-loop minimax MPC," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 2, pp. 1438–1442, IEEE, 2003.

[15] P. Scokaert and D. Mayne, "Min-max feedback model predictive control for constrained linear systems," *Automatic Control, IEEE Transactions on*, vol. 43, no. 8, pp. 1136–1142, 1998.

[16] J. Löfberg, "A convex relaxation of a minimax MPC controller," Linkping University Electronic Press, 2001.

[17] M. Mirzaei, N. K. Poulsen, and H. H. Niemann, "Robust model predictive control of a wind turbine," in *American Control Conference (ACC), 2012*, pp. 4393–4398, IEEE, 2012.

[18] J. Löfberg, *Minimax approaches to robust model predictive control*, vol. 812. Linköping University Electronic Press, 2003.

[19] J. Lofberg, "Minimax MPC for systems with uncertain gain," in *World Congress*, vol. 15, pp. 614–614, 2002.

[20] E. Polak, *Optimization: algorithms and consistent approximations.* Springer-Verlag New York, Inc., 1997.

[21] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: theory and practicea survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[22] X. Chen, M. Heidarinejad, J. Liu, and P. D. Christofides, "Distributed economic MPC: Application to a nonlinear chemical process network," *Journal of Process Control*, vol. 22, no. 4, pp. 689–699, 2012.

[23] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*, pp. 207–226, Springer, 1999.

[24] D. Q. Mayne, "Model predictive control: Recent developments and future promise," *Automatica*, 2014.

[25] J. Löfberg, "Oops! I cannot do it again: Testing for recursive feasibility in MPC," *Automatica*, vol. 48, no. 3, pp. 550–555, 2012.

[26] E. Polak, R. Trahan, and D. Q. Mayne, "Combined Phase I-Phase II methods of feasible directions," *Mathematical Programming*, vol. 17, no. 1, pp. 61–73, 1979.

[27] E. Polak and L. He, "Unified steerable Phase I-Phase II method of feasible directions for semi-infinite optimization," *Journal of Optimization Theory and Applications*, vol. 69, no. 1, pp. 83–107, 1991.

[28] C. Gonzaga and E. Polak, “On constraint dropping schemes and optimality functions for a class of outer approximations algorithms,” *SIAM Journal on Control and Optimization*, vol. 17, no. 4, pp. 477–493, 1979.

[29] M. A. Duran and I. E. Grossmann, “An outer-approximation algorithm for a class of mixed-integer nonlinear programs,” *Mathematical programming*, vol. 36, no. 3, pp. 307–339, 1986.

[30] G. Zoutendijk, “Methods of feasible directions: a study in linear and non-linear programming,” 1960.

[31] S. Zukhovitskii, R. POLIAK, and M. Primak, “Algorithm for solution of convex programming problem,” *Doklady Akademii Nauk SSSR*, vol. 153, no. 5, p. 991, 1963.

[32] D. M. Topkis and A. F. Veinott, Jr, “On the convergence of some feasible direction algorithms for nonlinear programming,” *SIAM Journal on Control*, vol. 5, no. 2, pp. 268–279, 1967.

[33] E. Polak, *Computational methods in optimization.* Academic press, 1971.

[34] R. Hettich and K. O. Kortanek, “Semi-infinite programming: theory, methods, and applications,” *SIAM review*, vol. 35, no. 3, pp. 380–429, 1993.

[35] J. T. Moore and J. F. Bard, “The mixed integer linear bilevel programming problem,” *Operations research*, vol. 38, no. 5, pp. 911–921, 1990.

[36] B. Colson, P. Marcotte, and G. Savard, “Bilevel programming: A survey,” *4OR*, vol. 3, no. 2, pp. 87–107, 2005.

[37] J. Viswanathan and I. E. Grossmann, “A combined penalty function and outer-approximation method for minlp optimization,” *Computers & Chemical Engineering*, vol. 14, no. 7, pp. 769–782, 1990.

[38] R. Fletcher and S. Leyffer, “Solving mixed integer nonlinear programs by outer approximation,” *Mathematical programming*, vol. 66, no. 1-3, pp. 327–349, 1994.

[39] H. P. Benson, “An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem,” *Journal of Global Optimization*, vol. 13, no. 1, pp. 1–24, 1998.

[40] P. Kesavan, R. J. Allgor, E. P. Gatzke, and P. I. Barton, “Outer approximation algorithms for separable nonconvex mixed-integer nonlinear programs,” *Mathematical Programming*, vol. 100, no. 3, pp. 517–535, 2004.

[41] E. M. Stein, *Singular integrals and differentiability properties of functions*, vol. 2. Princeton university press, 1970.

[42] J. Danskin, “The theory of min-max and its application to weapon allocations problems,” Springer, New York, 1967.

[43] V. F. Demjanov, “Algorithms for some minimax problems,” *Journal of Computer and System Sciences*, vol. 2, no. 4, pp. 342–380, 1968.

[44] V. Demanov, “On the solution of several minimax problems. i,” *Cybernetics and Systems Analysis*, vol. 2, no. 6, pp. 47–53, 1966.

[45] B. N. Pshenichny and Y. M. Danilin, *Numerical methods in extremal problems.* Mir Moscow, 1978.

[46] O. Pironneau and E. Polak, “On the rate of convergence of certain methods of centers,” *Mathematical Programming*, vol. 2, no. 1, pp. 230–257, 1972.

[47] S. Boyd and L. Vandenberghe, *Convex optimization.* Cambridge university press, 2004.

[48] R. Isaacs, *Differential games*, Rand Corporation, 1954.

[49] T. Basar and G. J. Olsder, *Dynamic noncooperative game theory*, vol. 200, SIAM, 1995.

[50] L. C. Evans and P. E. Souganidis, “Differential games and representation formulas for solutions of hamilton-jacobi-isaacs equations.,” tech. rep., DTIC Document, 1983.

[51] E. Polak and J. Royset, “On the use of augmented lagrangians in the solution of generalized semi-infinite min-max problems,” *Computational Optimization and Applications*, vol. 31, no. 2, pp. 173–192, 2005.

[52] J. Walrand, E. Polak, and H. Chung, “Harbor attack: A pursuit-evasion game,” in *Communication, Control, and Computing, 2011 49th Annual Allerton Conference on*, pp. 1584–1591, IEEE, 2011.

[53] A. G. K. Holmstrom and M. Edvall, “User guide for tomlab,” in *TOMLAB Optimization*, 2010.

[54] S. N. Wood and N. H. Augustin, “Gams with integrated model selection using penalized regression splines and applications to environmental modelling,” *Ecological modelling*, vol. 157, no. 2, pp. 157–177, 2002.

[55] J. Sprinkle, J. M. Eklund, H. J. Kim, and S. Sastry, “Encoding aerial pursuit/evasion games with fixed wing aircraft into a nonlinear model predictive tracking controller,” in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 3, pp. 2609–2614, IEEE, 2004.

[56] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino, “Closed loop steering of unicycle like vehicles via lyapunov techniques,” *Robotics Automation Magazine, IEEE*, vol. 2, pp. 27–35, Mar 1995.

[57] H. Chung, E. Polak, and S. Sastry, “An external active-set strategy for solving optimal control problems,” *Automatic Control, IEEE Transactions on*, vol. 54, no. 5, pp. 1129–1133, 2009.

[58] S. Lee, E. Polak, and J. Walrand, "A receding horizon control law for harbor defense," in *Proc. 51th annual Allerton Conference on Communication, Control, and Computing*, October 2013.

[59] A. Stubbs, V. Vladimerou, A. Fulford, D. King, J. Strick, and G. Dullerud, "Multivehicle systems control over networks: a hovercraft testbed for networked and decentralized control," *Control Systems, IEEE*, vol. 26, pp. 56–69, June 2006.

[60] J. Faigl, T. Krajnik, J. Chudoba, L. Preucil, and M. Saska, "Low-cost embedded system for relative localization in robotic swarms," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 993–998, IEEE, 2013.

[61] J. Stowers, A. Bainbridge-Smith, M. Hayes, and S. Mills, "Optical flow for heading estimation of a quadrotor helicopter," *International Journal of Micro Air Vehicles*, vol. 1, no. 4, pp. 229–239, 2009.

[62] S. K. Phang, J. J. Ong, R. T. Yeo, B. M. Chen, and T. H. Lee, "Autonomous mini-UAV for indoor flight with embedded on-board vision processing as navigation system," in *Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8 International Conference on*, pp. 722–727, IEEE, 2010.

[63] P. E. Papamichalis, *Digital signal processing applications with the TMS320 family (vol. 2).* Prentice-Hall, Inc., 1991.

[64] R. Otap, "Development of a robotic testbed infrastructure with dynamic service discovery," *Univ.of Illinois at Urbana-Champaign Master's thesis*, 2013.

[65] "https://github.com/mhaberler/directoryd"

[66] "http://www.joptimizer.com"

[67] J. Rubel, "Design and control of hovercraft over a network," *Univ.of Illinois at Urbana-Champaign Master's thesis*, 2004.

[68] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[69] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *Control Systems Technology, IEEE Transactions on*, vol. 18, no. 2, pp. 267–278, 2010.

[70] I. Maurovic, M. Baotic, and I. Petrovic, "Explicit model predictive control for trajectory tracking with mobile robots," in *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, pp. 712–717, IEEE, 2011.

[71] S. Salmanipour and S. Sirouspour, "Teleoperation of a mobile robot with model-predictive obstacle avoidance control," in *Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE*, pp. 4270–4275, IEEE, 2013.

[72] Y. Noda, T. Sumioka, and M. Yamakita, "An application of fast MPC for bike robot," in *SICE Annual Conference (SICE), 2012 Proceedings of*, pp. 540–545, IEEE, 2012.

[73] K. Springer, F. J. Kilian, and H. Gattringer, "Active vibration control of a flexible link robot with MPC," in *Nonlinear Model Predictive Control*, vol. 4, pp. 163–168, 2012.

[74] H. Seguchi and T. Ohtsuka, "Nonlinear receding horizon control of an underactuated hovercraft," *International journal of robust and nonlinear control*, vol. 13, no. 3-4, pp. 381–398, 2003.

[75] S. Bhattacharya, A. Gupta, and T. Basar, "Jamming in mobile networks: A game-theoretic approach," *Numerical Algebra, Control and Optimization*, vol. 3, pp. 1–30, March 2013.

[76] A. Newell and H. A. Simon, "Computer science as empirical inquiry: Symbols and search," *Communications of the ACM*, vol. 19, no. 3, pp. 113–126, 1976.

[77] J. Pearl, "Scout: A simple game-searching algorithm with proven optimal properties.," in *AAAI*, pp. 143–145, 1980.

[78] J. Pearl, "Asymptotic properties of minimax trees and game-searching procedures," *Artificial Intelligence*, vol. 14, no. 2, pp. 113–138, 1980.

[79] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin, "Best-first fixed-depth minimax algorithms," *Artificial Intelligence*, vol. 87, no. 1, pp. 255–293, 1996.

[80] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Natural computing*, vol. 1, no. 2-3, pp. 235–306, 2002.

[81] B. Bhattacharjee, P. Lemonidis, W. H. Green Jr, and P. I. Barton, "Global solution of semi-infinite programs," *Mathematical programming*, vol. 103, no. 2, pp. 283–307, 2005.

[82] S. Lee, E. Polak, and J. Walrand, "On the use of min-max algorithms in receding horizon control laws for harbor defense," *Engineering Optimization 2014*, pp. 211–217, 2014.

[83] S. Xu, "Smoothing method for minimax problems," *Computational Optimization and Applications*, vol. 20, no. 3, pp. 267–279, 2001.

[84] M. Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[85] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias, “Evaluating evolutionary algorithms,” *Artificial intelligence*, vol. 85, no. 1, pp. 245–276, 1996.