Towards Real-Time Distributed Planning in Multi-Robot Systems

Dissertation by

Mohamed Abdelkader

In Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

King Abdullah University of Science and Technology

Thuwal, Kingdom of Saudi Arabia

April, 2018

EXAMINATION COMMITTEE PAGE

The dissertation of Mohamed Abdelkader is approved by the examination committee

Committee Chairperson: Prof. Jeff S. Shamma

Committee Members: Prof. Mohammad Younis, Prof. Taous-Meriem Laleg-Kirati, Dr. Georgios Chasparis (External)

ABSTRACT

Towards Real-Time Distributed Planning in Multi-Robot Systems

Mohamed Abdelkader

Recently, there has been an increasing interest in robotics related to multi-robot applications. Such systems can be involved in several tasks such as collaborative search and rescue, aerial transportation, surveillance, and monitoring, to name a few. There are two possible architectures for the autonomous control of multi-robot systems. In the centralized architecture, a master controller communicates with all the robots to collect information. It uses this information to make decisions for the entire system and then sends commands to each robot. In contrast, in the distributed architecture, each robot makes its own decision independent from a central authority. While distributed architecture is a more portable solution, it comes at the expense of extensive information exchange (communication). The extensive communication between robots can result in decision delays because of which distributed architecture is often impractical for systems with strict real-time constraints, e.g. when decisions have to be taken in the order of milliseconds.

In this thesis, we propose a distributed framework that strikes a balance between limited communicated information and reasonable system-wide performance while running in real-time. We implement the proposed approach in a game setting of two competing teams of drones, defenders and attackers. Defending drones execute a proposed linear program algorithm (using only onboard computing modules) to obstruct attackers from infiltrating a defense zone while having minimal local message passing. Another main contribution is that we developed a realistic simulation environment as well as lab and outdoor hardware setups of customized drones for testing the system

in realistic scenarios. Our software is completely open-source and fully integrated with the well-known Robot Operating System (ROS) in hopes to make our work easily reproducible and for rapid future improvements.

ACKNOWLEDGEMENTS

During my time at KAUST, I have had a unique experience. I have learned a lot and worked on several challenging and fun projects. I am thankful to everyone who has been part of this journey.

First, I would like to thank my advisor, Prof. Jeff S. Shamma, for his continuous support and encouragement during my PhD time. Thank you for helping me to go through the struggles of my PhD, guiding me to keep my focus whenever I diverge, and finally to complete my PhD.

I also would like to thank my committee members, Prof. Mohammad Younis, Prof. Taous-Meriem Laleg-Kirati, Dr. Georgios Chasparis, for taking the time to serve in my PhD committee.

A round of applause goes to my colleagues and friends who have helped me in several ways throughout my PhD. Thanks to them for sharing all the fruitful time during our daily works or fun moments. Thanks to Dr. Hassan for being an effective partner in my research work and for your guidance an support in our discussions. Thank you my lab colleagues and friends, Shaqura, Dr. Mohammad Mabrok, Dr. Jiming, Dr. Doha, Yimeng Lu, Dr. Samet, Alfiya, Khalid, Haya, Algarni, Alrashed, Amjad, Fadl.

A special thanks goes to my beloved mother. Without your continuous support, love, and prayers, I would not be able to survive life struggles. Also, thanks to my father for providing your golden wisdom throughout my life.

I also would like to thank my better half, my wife, Sally, for her continuous support, patience, and love. Finally, I can never forget the indescribable joy that my little 3-year old daughter has brought to my life. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The objective of this work is to present a set of tools for an efficient distributed coordination in a team of entities (or *agents*) to collaborate in order to fulfill an assigned objective under certain practical system's constraints. Our work is motivated by practical challenges that arise in certain adversarial game setups where agents (e.g. *robots*) need to coordinate their decisions in real-time under certain system constraints in order to achieve system-wide objectives. We apply certain optimization techniques to allow individual agents to coordinate their behavior in a distributed way to accomplish certain system objective in real time. We also validate our proposed approaches through realistic simulations and hardware implementation. Furthermore, our implementation is provided as open-source package that can run inside the *robot operating system* (ROS).

1.1 Multi-Agent System

There are many definitions of the term *agent* in different communities and we would like to elaborate on certain aspects that are in line with the concepts in this thesis. An agent is not limited to a computer system, but can generally be a biological entity such as bees, ants, birds, or human. However, in this thesis, we consider agents that of computer system type. More specifically, we consider agents that can sense, decide and take actions in certain environment. You can think of it as a robot that has sensors to perceive the environment, actuators to manipulate the environment, and computers to generate decisions.

We can summarize the required features in an agent we consider as follows.

- We consider *computer agents* such as robots that has sensors and actuators
- Agents need to be *autonomous*. They should be able to take decision on their own with out human intervention
- Agents should be able to *reason* about their environment in order to achieve certain objectives. For example, a group of robots need to navigate through an unknown environment while avoiding collision and with obstacle and with each other.

Designing an single agent to react in an environment might be a simple task. However, designing multiple agents to operate simultaneously in an environment to achieve certain objective is more challenging as the disturbance, not only from environment but also from the results of other agents actions, becomes harder to handle. So, an agent need to be able to react to changes in the environment and interact with other agents in order to achieve a specific objective.

Having multiple agents working together can help achieve tasks that are harder to do with a single agent. For example, a group of robots that carry a single heavy payload that cannot be handled by a single robot. However, that does not mean controlling them is easier. As the number and complexity of these agents increase, controlling them in a way that they cooperate with each other becomes very challenging. In *multi-agent* systems, cooperation becomes more difficult due to the fact that the environment is unpredictable and the information received about the world or other agents (through sensing or communication) are generally incomplete and imperfect. Designing agents controllers under such conditions becomes extremely challenging, as we will see later in the thesis, specially if certain optimal performance is required. In many real world applications, there are other important limitations that has to be addressed in designing controllers in *multi-agent* systems. More specifically, real-time implementability, limited computation power, and/or limited communication resources are of a paramount interest. Designing optimal controllers in the presence of such limitations becomes challenging as it may require more computation time and/or communications that may violate the real-time constraints of the system, and/or extensive communication (for information exchange). Generating goal-oriented decisions in multi-agent systems can be done in centralized or decentralized way. The difference is explained as follows.

1.2 Centralized vs. Distributed Coordination

A conventional way of generating optimal decisions in*multi-agent* systems is to design a *centralized* controller. A *centralized* controller receives full information about the world and agents and generates decisions that are sent to individual agents to execute. Although a *centralized* controller (or decision maker) can generate optimal decisions, it suffers from well known issues such as scalability in the number of agents and single-point-of-failure. Furthermore, It assumes full knowledge about the environment and agents states at all times which is impractical due to the aforementioned limitations.

Another way of generating decisions in *multi-agent* systems is *decentralized* control. This can be more practical in many systems such as wireless sensor networks [1, 2, 3], distributed parameters estimation [4, 5], formation control [6], and collaborative object transportation [7], to name a few. In *decentralized* control, each agent generates its own decision based on its own limited knowledge of the world and other agents. With such setup, decision making approaches can generally be scalable, single-point of failure is eliminated, the previous limitations on incomplete information and limited computations can be addressed. Several existing works provide methods for *decentralized* decision making in *multi-agent* systems for different system setups. However, they rarely address the real-time implementability of such methods and verifications on an actual system.

In this thesis, we try to design distributed planning algorithms that respect practical implementation constraints such as limited computation and communication resources. We also provide a hardware realization of the proposed approaches for system performance validation and assessment under real testbed.

1.3 Motivation

The work in this thesis is motivated by the challenges that arise in certain adversarial game settings. In particular, we consider an adversarial game setting between two teams with conflicting objectives. One team is called defenders which is a set of robots that are supposed to protect their defined defense zone against the other team. The other team is the attackers which is also a set of robots that try to infiltrate the defenders defense zone. Looking from the defenders perspective, we seek to design algorithms that allow each defender to take individual actions without a central coordinator and by interacting (e.g. communicating) with other defenders in the team. This game setting is a simplified version of the capture the flag game which is popular because it is a complex game with a variety of challenges that must be addressed in a practical multi-agent system (see for example [8, 9, 10, 11]).

In such systems consisting of mobile robots that are battery powered, have limited processing capabilities, and communicate over noisy and shared channels, real time implementability of an algorithm has higher priority than optimal performance specially when robots travel at high speeds and have to react promptly. In particular, for systems like quadrotors that have higher-order complex dynamics, it is extremely important to compute a feasible control action in real time. Optimality of control actions is always desirable but time complexity for computing these actions can make them undesirable for such complex systems under practical situations. It is imperative that the update actions are computed efficiently in real time because a small delay in the computation or execution of a feasible action may result in the defeat of the entire team.

In the literature, numerous approaches have been proposed for path planning problems. One approach is to formulate the problem as a dynamic program as in [12], which can guarantee optimal solution. However, except for special cases like LQR problems that have analytical solutions, dynamic programs suffer from the curse of dimensionality and cannot be solved even for a moderate size system. A computationally efficient approach is to formulate the problem as a mixed integer linear program or a linear program. However, only centralized solutions have been proposed based on these approaches [9, 10]. Another popular approach is based on model predictive control (MPC) in which each agent assumes a model for all the other dynamic agents in its environment over a finite prediction horizon and solves an optimization problem (see for example [13, 14, 15]). However, these solutions typically require communication of entire trajectories among neighboring agents, which can result in excessive communication cost and latency in decision making.

In this work, we present a distributed, energy efficient, and real time implementable algorithm for path planning for the attackers-defenders game. We start with reviewing the roots of the problem in a centralized setup and the corresponding challenges. Namely, the problem can be formulated as a centralized dynamic program which is computationally prohibitive to solve. Then, a centralized linear program formulation is adopted which is presented in [10] that allows for online solutions. Next, we propose an approximate algorithm for its distributed implementation. For the distributed implementation, each defender solves a local version of the linear program based on the current locations of its neighboring agents, and of the attackers. Since the optimization problem has to be solved over a fixed prediction horizon, each defender requires a model for the evolution of the attackers. We assume that the attackers are updating their locations based on a feedback law that moves them towards the flag.

We also evaluate the performance of the proposed algorithm in realistic simulations using quadrotors and provide a hardware implementation that shows the real-time applicability of the approach. Moreover, an open-source software package for the simulation and hardware implementation is available for public testing and future improvements/extensions.

1.4 Thesis Organization

In Chapter 2, a background on the related literature is presented. The motivating problem setup is presented in Chapter 3 followed by the centralized problem formulation and solutions in chapter 4. The proposed distributed approach is discussed in Chapter 5, and the simulation and the hardware implementations are discussed in Chapter 6. Finally, the concluding remarks and future work are discussed in the last chapter, Chapter 8.

1.5 Objectives and Contributions

The main objective of this thesis is to design distributed algorithms for online decision making that respects practical requirements in multi-agents such as limited computational resources and limited information exchange for real-time applications.

The contributions of this thesis folds in the following streams:

- Designing a real-time and online distributed path planning algorithm in the context of an adversarial game based on linear program formulation.
- Validating the proposed algorithms in terms of real-time performance in a realistic simulation as well as a hardware setup composed of two teams of quadrotors in competing in a certain game setup.
- Providing an open-source software package that is used in the simulation and hardware testing for easier future extensions.

Chapter 2

Background

There are several frameworks to solve distributed decision making problems according to the the problem structure. In this chapter, numerous works in this area are summarized under 3 main clusters: (1) distributed dynamic programming and reinforcement learning (2) distributed optimization algorithms, (3) distributed model predictive control (DMPC).

2.1 Distributed Dynamic Program and Reinforcement Learning

In order to introduce distributed approaches based on dynamic program, a brief introduction on a dynamic program problem is presented.

2.1.1 Dynamic Program

Dynamic programming provides a systematic theoretical framework to solve any sequential decision making problem [16]. Example of problems that can be solved using dynamic programming are optimal control, Markovian decision problems, planning, to name a few.

First, the dynamic program formulation of a centralized problem is presented. Then, different techniques for solving the centralized problem in a decentralized setup are presented in the next section.

In this thesis, we will work with problems of discrete and finite state space $\mathcal{X}$ and actions $\mathcal{U}$, and discrete time.

The elements of a decision making problem are described as follows.

- **States**: Representation of the information evolution over time e.g. position of a robot over time. A state at a specific time t is denoted by $x(t) \in \mathcal{X}$, where $\mathcal{X}$ is the space of all possible states. For brevity, when the time index t is dropped, $x = x(t)$ and $y = x(t+1)$.

- **Controls**: Actions that are committed to, which can take the system to a new state. A control variable at a specific time t is denoted by $u(t) \in \mathcal{U}$, where $\mathcal{U}$ is the set of all admissible controls. For brevity, when the time index t is dropped $u = u(t)$

- **State transition**: state evolutions are governed by generally non-linear transition function. $x(t+1) = f(x(t), u(t))$.

- **Objective**: A performance metric which can be a reward to maximize or a cost to minimize. In this thesis, without loss of generalities, we will work with cost functions. A stage cost at time t is denoted by $g(x(t), u(t))$. The objective is described by Eq. 2.1.

$$\mathcal{J}^* = \min_{u(0),u(1),\cdots,u(T-1)} \sum_{t=0}^{T} g(x(t), u(t)) \tag{2.1}$$

The objective is to minimize the total cost which is equal to the sum of stage costs $g(x, u)$ over a finite time horizon $\mathbf{T}$. The optimal actions are the minimizers that correspond to the optimal cost $\mathcal{J}^*$ are denoted by $u^*(0), \cdots, u^*(T-1)$.

A naive way to find u^* is to perform an exhaustive search over the actions $u \in \mathcal{U}$ for each state x. However, this is clearly a computationally prohibitive approach when the action space $\mathcal{U}$ is large. Dynamic programing uses principle of optimality [17] to solve 2.1 in an intelligent way that reduces the search space significantly compared to an exhaustive search. In simple words, the principle of optimality states that starting at any point x_t on an optimal path $x(0), \cdots, x(T)$, the path $x(t), \cdots, x(T)$ is itself an optimal path.

It can be shown that finding the optimal controls corresponds to finding the solution to *Bellman's equation* [16].

$$\mathcal{J}^*(x) = \min_u g(x,u) + \mathcal{J}^*(f(x,u)) \tag{2.2}$$

Problem 2.1 can be further generalized for stochastic systems where the probability of transitioning from x to y after committing action u is $P_u(x,y)$. The corresponding *Bellman's equation* for a discounted cost over an infinite horizon becomes,

$$\mathcal{J}^*(x) = \min_u g(x,u) + \alpha \sum_y P_u(x,y)\mathcal{J}^*(y) \tag{2.3}$$

Where $\alpha \in (0,1)$ is a *discount factor* that captures time preference i.e $\alpha \to 0$ corresponds to short term gain while $\alpha \to 1$ corresponds to long term gain.

Dynamic programing offers a number of approaches to solve *bellman's equation.* One approach that is relevant in later discussions of decentralized approximate dynamic program make use of linear programming shown below. It can be shown that the solution to *Bellman's equation* 2.3 is equivalent to the solution of a linear program, see [18]. Although there are very efficient tools to solve linear programs, the size of this linear programs (mainly the constrains) is a function of the state and inputs space size. For large size problems, it becomes computationally prohibitive to solve.

Another quantity of interest is the Q-function, which will be used later in the decentralized approaches. The optimal Q-function is:

$$Q^*(x,u) = g(x,u) + \alpha \sum_y P_u(x,y)\mathcal{J}^*(y) \tag{2.4}$$

It also can be written as:

$$Q^*(x,u) = g(x,u) + \alpha \sum_y P_u(x,y) \min_{v \in \mathcal{U}} Q^*(x,v) \tag{2.5}$$

Eq. 2.5 represents *Bellman's equation* in terms of Q-function.

The optimal policy $\mu^*(x)$ can be found by:

$$\mu^*(x) = \operatorname*{argmin}_u Q^*(x,u) \tag{2.6}$$

2.1.2 Distributed approaches

Now, let's consider a system of n agents. The system state is $x = (x_1, \cdots, x_p)$ where $x_j \in X_j$ is some partial state. Similarly, the system controls are $u = (u_1, u_2, \cdots, u_n)$ where $u_i \in U_i$ is the control of agent i. In general, policies obtained by dynamic programming produce actions that depend on the entire state $x = (x_1, \cdots, x_p)$. However, in a decentralized setup it is desired to have policies that produce controls u_i that depend only on a subset of the state variable x. We seek to solve for $Q^*(x,u)$ in a decentralized manner such that the obtained polices provide controls u_i that depend only on a subset of x.

In the literature, solving for Q^* in a decentralized way is done using different approaches. Some approaches try to solve for Q^* or an approximation for it in an offline setting. In other words, Q^* is first obtained and then implemented. Some other approaches solve for Q^* in an online way. In such approaches, usually the system model is not know, and Q^* is learned by experiencing different policies and optimizing them based on certain rewards. This usually discussed as *reinforcement learning*. Usually , Q^* is approximated by a function $\hat{Q}$ with a special structure that renders itself to decentralized implementation. This approximation is done either, to generate a structure that is appealing to distribute implementation, or because Q^* is actually complex and large to compute. Different works presented different methods of offline and online estimation of Q^* which are summarized below.

2.1.2.1 Exact offline approximation of Q-function

In [19], the authors present a linear programming method to find a decentralized policy from a function with special structure that approximates the centralized $Q^*(x,u)$. Firstly, $Q^*(x,u)$ is approximated by $\hat{Q}(x,u) = \sum_i^n Q_i(x,u_i)$. As mentioned, u_i depends only on some x_i according to an information structure. In particular, let

$$\mathcal{I}_i = \{j |\ u_i \text{ depends on } x_j\}$$

In particular, $\mathcal{I}_i$ is the set of indices of the state variables that action u_i depends on. The Cartesian product of the spaces of the variables used to decide action u_i is denoted as

$$\mathcal{X}_i = \bigtimes_{j\in\mathcal{I}_i} X_j$$

With this notation, a decentralized policy can be defined as $\mu_i : \mathcal{X}_i \to U_i$ for $i = 1, \cdots, n$. One should note two pieces of information about $Q_i(x,u_i) : \mathcal{X}_i \times U_i \to \mathbb{R}$. $\sum_i \underset{u_i}{\operatorname{argmin}}\ Q_i(x,u_i) = \underset{u}{\operatorname{argmin}}\ \hat{Q}(x,u)$, since u_i appears only in $Q_i(x,u_i)$. Also, the choice of u_i depends only on the states appearing in $\mathcal{X}_i$, because these are the only states appearing in $Q_i(x,u_i)$. Therefore, μ is decentralized since each decision u_i depends only on subset of the states $\mathcal{X}_i$.

Given this setup, a linear program algorithm can be formulated to generate an approximation $\hat{Q}(x,u) = \sum_i Q_i(x,u_i)$ to the optimal Q-funcitons, $Q^*(x,u)$. The reader is referred to [19] for more details on the linear program the corresponding theoretical analysis.

The solution to this linear program is equivalent to the problem of minimizing the error between the optimal Q-function Q^* and its approximation $\hat{Q}$. Therefore, The quality of the distributed solution is a function of the approximation error between Q^* and $\hat{Q}$.

Now, we present few notes on the previous approach. The previous approach seeks to find a local approximation architectures for decentralized control. However, it requires to solve a centralized linear program to generate an approximation to the Q-function. Furthermore, computing and storing the Q-function for practical scale of large scale is intractable, and using a centralized policy may lead to prohibitive requirements on communication structure between agents. It is also noted that the previous algorithm provides an offline solution i.e. decentralized policies need to be generated offline before they can be used. Another note is that the algorithm works only once the information structure is fixed and, hence, does not work with networks of dynamic topologies.

To overcome the high dimensionality in the previous approach, and to adapt to dynamic network topologies a different approach that adopts a different approximation of the Q-function based on *basis functions*is discussed in the next section.

2.1.2.2 Inexact offline approximation of Q-function

In [20] the authors presented another linear program approach to provide different approximations of Q-function that simultaneously address the curse of dimensionality and the need for decentralized control strategies. They propose a method of generating local and linear approximation of Q-function which is based on linear programming approximation of dynamic programs [21], [22].

The following equation shows the approximation of the Q-function.

$$Q^*(x,u) \approx \sum_i \tilde{Q}_i(h_i(x), u_i, r_i) = \sum_{i=1}^{n} \sum_{j=1}^{K} \phi_{i,j}(h_i(x), u_i) r_{i,j} \tag{2.7}$$

In 2.7, Q-function is approximated by a linear combination of local basis functions $\phi_{i,j}$. Each agent i makes observations governed by $h_i : \mathcal{X} \rightarrow \mathcal{O}$. When the state of the system is x, the agents observes $h_i(x)$. $\phi_{i,j}$ depends on the state x only through the information, provided by $h_i(x)$, available for agent i.

The optimal policy for the approximation in 2.7 is

$$\mu(x) = \operatorname*{argmin}_{u} \{\sum_{i=1}^{n}\sum_{j=1}^{K} \phi_{i,j}(h_i(x), u_i) r_{i,j}\} = (\operatorname*{argmin}_{u_i} \{\sum_{i=1}^{n}\sum_{j=1}^{K} \phi_{i,j}(h_i(x), u_i) r_{i,j}\})_{i=1}^{n} \tag{2.8}$$

Based on 2.8, each agent can make decisions $\mu_i(x)$ based on local information $h_i(x)$ without coordinating action choices with other agents. However, coordination can be rendered through the structure of the state space and information functions $h_i(x)$.

The reader is referred to [20] for more details on the approximate linear program algorithm and the error bounds between the approximation and the optimal Q-function Q^*.

This algorithm provides an approximation of the Q function in terms of basis function that reduces that dimensionality since the basis is a function of $h(x)$ and not directly a function of x. However, in the linear program that is solved in order to solve for the weights $\phi_{i,j}$, the constraint size is still linear in the number of states x, which makes computationally prohibitive for large scale problems.

For problems where the system model (e.g. that transition probabilities) is not know, online reinforcement learning methods can be used. This is described as follows.

2.1.2.3 Distributed reinforcement learning

Q function can be obtained iteratively in an online setting as follows. Define r as the immediate (current) reward of being at state x and committing action u. Q can be *learned* (or updated) according to the following rule,

$$Q(x, u) = r + \gamma max_{u'} Q(x', u'), \quad 0 < \gamma < 1 \tag{2.9}$$

Using 2.9, Q converges to Q^* given that each (x, u) is visited infinitely many times.

In a distributed setting, learning Q^* can done by sharing information about Q values from each agent. This can be done using simple averaging [23] as follows,

$$Q_i(x, u) = \frac{1}{n} \sum_j Q_j(x, u) \tag{2.10}$$

This simply means that each agent i shares its current Q value with every other agents j and computes a simple average. In this case, all agents share the same Q-table. However, this approach showed a reduced convergence to Q^* [23]. Another approach is to do expertness based cooperative Q-learning [24]. In this approach agents value more the Q values from expert agents. Specifically, each agent i updates its Q_i value as follows.

$$Q_i(x, u) = \sum_j W_{ij} \times Q_j(x, u) \tag{2.11}$$

Where W_{ij} is an importance weight to the Q data held by agent j. The idea is to value more highly the knowledge of agents who are *experts.* Using proper measures for expertness and when agents have highly different expertness values, it can be shown that it outperforms the simple averaging method in 2.10 in terms of faster convergence [24]

2.2 Distributed Optimization

A decision making problem can generally be formulated as an optimization problem with certain objective and constraints structure. Distributed optimization algorithms decompose the original problem to smaller subproblems, one for each agent, that can be solved iteratively either in sequential or parallel manner.In many problems such as networked robotics, collaborative control, network resource allocation, estimation and identification problems and others, it is desired that the distributed optimization method relies only on local information between the subproblems without the need for a central coordinator.

Distributed optimization algorithms usually exploit certain problems structure such as separability in the cost function and/or constraints in order to derive distributed algorithms. In this section, a set of distributed optimization algorithms are presented in order to give the reader a quick overview on the different methods used in this domain.

We will categorize the presented methods as follows.

- **Unconstrained convex problems**. In this category, distributed methods that solve problems with only objective functions and no constraints on the optimization vector, are presented. There are methods that can work with *time varying* and *time invariant* communication topology (graphs)

- **Constrained convex problems**. When constrains on the optimization variables are present, there are different methods that can handle *global convex constraints* and *local convex constraints*. Also, for this category there are methods for *time invariant* and *time varying* communication topologies.

In what follows, different algorithms that fall under the previous two categories are briefly presented in terms of their advantages as well as their shortcomings. For more details on the algorithms, the reader is referred to the mentioned references. Finally, a summary of different distributed optimization setups are presented in Table 2.1

2.2.1 Unconstrained Problems

Consider the following problem,

$$\begin{gathered}\min_x \sum_{i=1}^{N} f_i(x) \\ x \in \mathbb{R}^m\end{gathered} \tag{2.12}$$

In problem 2.12, the cost function is a sum of local convex functions $f_i(x) : \mathbb{R}^m \to \mathbb{R}$ which are functions of the whole optimization vector $x \in \mathbb{R}^m$

In [25], the author presents an algorithm that solves problem 2.12 in a distributed way using a combination of sub-gradient [26] and consensus steps [27]. The method works with a dynamic graph topology $G(V, E_k)$, where V is set of nodes or agents, and E_k is the set of edges at time k. Each agent i computes and maintains estimates about the optimal decision vector x based on sub-gradient information of his own cost function f_i. The communication is local and can be asynchronous. Each agent, obtains its own estimates on x iteratively over time, using the following rule.

$$x^i(k+1) = \sum_{j=1}^{N} a^i_j(k)x^j(k) - \alpha^i(k)d_i(k) \tag{2.13}$$

$x^i(k)$ is the estimate obtained by agent i at time step k. $a^i_j(k)$ is the weight that agent i puts on the estimate $x^j(k)$ provided by agent j at time k. $a^i_j(k) > 0$ if agent i is connected to agent j at time k, $a^i_j(k) = 0$ otherwise. $d_i(k)$ is the gradient of the local function f_i at time k. If f_i is not smooth then, $d_i(k)$ is the sub-gradient of f_i. The algorithm has convergence guarantees to approximate optimal global solution as well as convergence rates of estimates maintained by each agent.

The advantages of the previous algorithm is that it provides computationally appealing approach for distributed unconstrained optimization with asynchronous local communication, dynamic graph topology, and with convergence guarantees. However, it can be not practical to use if the (sub)gradient of f_i is complex or not easily obtainable. Although the required communication is local, it need to be iterative and can be practically not desirable in real systems specially when the size of x is large.

2.2.2 Constrained Problems

For constrained optimization problems, there are different algorithms for different constraint types i.e. *global* vs. *local* convex constraints.

First, let's consider the following problem of optimizing a sum of agent's local convex function over a global convex set of constrains.

$$\begin{aligned} &\min_x \sum_{i=1}^{N} f_i(x) \\ &x \in \mathcal{X} \end{aligned} \tag{2.14}$$

In [28], the author presented an algorithm that can solve problem in 2.14 in an asynchronous way for convex objective functions f_i with a common convex constraint set $x \in \mathcal{X}$. The algorithm provides guarantees of convergence to optimal solution with the presence of link failures for a static graph topology. The algorithm uses information (estimate of x from an agent) broadcast and only one agent is allowed to broadcast its estimate of x at every time step k.

Next, let's consider the following problem with *local* convex constraint sets.

$$\begin{aligned} &\min_x \sum_{i=1}^{N} f_i(x) \\ &x \in \bigcap_{i=1}^{N} X_i \end{aligned} \tag{2.15}$$

In 2.15, each agent i computes its estimate of $x(k)$, denoted by $x^i(k)$, at time k that has to lie in a local convex set X_i.

In [29], authors presented a distributed *projected* sub-gradient algorithm that can solve problems of the form 2.15 over a static graph topology. For a dynamic topology, an error on the optimal solution is established. The algorithm uses a combination of sub-gradient and consensus steps, similar to 2.13, but with projection on the local convex set X_i. Each agent updates its own estimate of $x(k)$, denoted by $x^i(k)$, at time k according to the following rule.

$$\begin{aligned} v^i(k) &= \sum_{j=1}^{N} a^i_j(k) x^j(k) \\ x^i(k+1) &= P_{X_i}\left[vi(k) - \alpha_k d_i(k)\right] \end{aligned} \tag{2.16}$$

In 2.16, there are three steps. In the first step, agent i performs a consensus step to obtain a weighted average $v^i(k)$ using its current local estimate and the estimates from neighbors j. Then, a gradient step is taken using the (sub)gradient $d_i(k)$ of the local function f_i. Finally, a projection is performed to get an estimate $x^i(k+1)$ that lies in the local convex set X_i.

Using a step size α_k converging fast enough to zero, with time-invariant topology (i.e. $a^i_j(k)$ are constant), and all local sets X_i are the same (i.e. $\bigcap_{i=1}^{N} X_i = X_i = X$), the estimates $x^i(k)$ converge to the optimal solution. The algorithm is not suitable of problems where a mixture of local and global constraints are present.

2.2.3 Summary

We presented some distributed optimization algorithms that demonstrate the foundational ideas for for solving multi-agent decision making problem where the objective function and the constraints are convex. Those are leading references to several other similar variants with differences on the technical details such as optimality, convergence rates and connectivity. Table 2.1 summarizes approaches concerning distributed optimization problems.

Methods for distributed optimization for non-convex problems are not as mature as the works on convex problems and is also not of interest of the work in this thesis. However, we refer the reader to works in [41], [42], and [43] for further readings on that topic.

Table 2.1: Summary of distributed optimization methods for convex problems.

Constraint Type	Communication graph	Approach	References
Unconstrained	Static	sub-gradient + consensus steps	[30], [31]
		augmented Lagrangian + consensus step	[32]
		randomized incremental sub-gradient	[33]
		ADMM	[27], [34], [35]
	Dynamic	sub-gradient + consensus steps	[36],[37]
		sub-gradient + consensus steps	[38]
Global set	Static	projected sub-gradient + consensus steps	[28], [39]
		dual decomposition	[40]
Local sets	Static	projected sub-gradient + consensus steps	[29]
		augmented Lagrangian method	[32]

2.3 Distributed Model Predictive Control

Another line of works related to online computation of control strategies utilize *model predictive control* (MPC) based approaches. MPC in simple words is a way of online periodic re-planning of decisions based on certain performance criterion to be optimized and a model for predicting system's response in time. It also can handle systems with multi-variabl and input and state constraints. For a rigorous explanation of receding horizon control concepts, see [44, 45, 46].

For distributed control applications, there is number of approaches on *distributed model predictive control* (DMPC). As the mathematical and problem descriptions can vary between the different approaches, only a brief description of the different approaches is provided according the proposed categorization below. DMPC can be categorized from the following perspectives.

- Process properties

- Control architecture
- Theoretical properties

Process properties perspective is related to the *system dynamics* type, i.e. linear, non-linear, hybrid. Also, it describes *coupling* sources in the system or how the optimization is non-separable in inputs, state, outputs, constraints, or objective. For example, in [47, 48, 49, 50], authors considers DMPC problems for linear systems with coupling in linear constraints, inputs, objective, states, respectively. For non-linear systems, works in [51, 52, 53, 54] provide specific approaches for problems with coupling in input, constraints, objective, state and dynamics, respectively.

Control architecture perspective is related to how the subsystems or agents interact (i.e. *distributed* with communication, *decentralized* with no communication). It is also related to how the computation and communication schemes. In other words, if the *computations* are iterative or not, and if the *communications* are serial or parallel. For example, works in [55, 56, 57] provide DMPC approaches that require iterative computations to generate the control actions. There is also some works that propose DMPC approaches with non-iterative computations, see [58, 59, 60]. From communication scheme perspective, most of the proposed DMPC approaches work with parallel communication scheme where each all agents can compute their controls at each sample time, see [61, 62, 55]. A less number of works discuss DMPC approaches where serial communications is used, see [63, 64].

Theoretical properties perspective is related to optimality, suboptimality bounds, stability and robustness. The reader is referred to [65] for more details on such approaches and on a review of several application specific DMPC approaches.

2.4 Summary

In this chapter we provided a general review of the main clusters of works we believe are related to the distributed decision making in multi-agent systems. In summary, there are two main clusters of works that provide general frameworks to distributed decision making: *distributed dynamic programing* and *distributed optimization.* There is another cluster of works that provide application specific approaches in the context of *distributed model predictive control* (DMPC). The work in this thesis starts building a distributed algorithm in the context of a specific setting discussed in Chapter 3, for online real-time decision making. Furthermore, the message of this work is to also provide a way of generalizing the proposed application-specific approach to quantify the trade-off quality of the distributed solution against the centralized one, for example, through (approximate) dynamic program techniques.

Chapter 3

Problem Setup

3.1 Introduction

In this chapter, we will introduce the motivating problem that is used as a guiding case throughout the work presented in this thesis. The motivating problem is a basic variant of a game setup called *capture the flag* (see for example [8, 9, 10, 11, 66]). In this game, two teams of players battle against each other. Each team has a zone that has some flags. Each team's objective is to infiltrate to the other team's home zone, grab the flag, and bring it back to their home zone while avoiding obstacles and collision with their teammates. The game is a mix of offensive and defensive behaviors: each team should capture the opponent's flag while at the same time blocking them from capturing their own flag. The game is depicted in Fig 3.1.

In a general setup of this game, the system consists of a large number of vehicles that are either self-operated or controlled by a human operator. Each team must achieve a common objective: search and rescue and home monitoring and surveillance. The vehicles can be controlled by a central coordinator or have some sort of distributed control architecture. All units may communicate with each other through a communication setup which can be subject to bandwidth and latency limitations and dropouts. There also can be either global sensing setup e.g. GPS, or local ones via the individual agent's sensors.

This game brings a set of complex challenges and research questions. They range from the individual vehicle control, the global planning and coordination of multi-

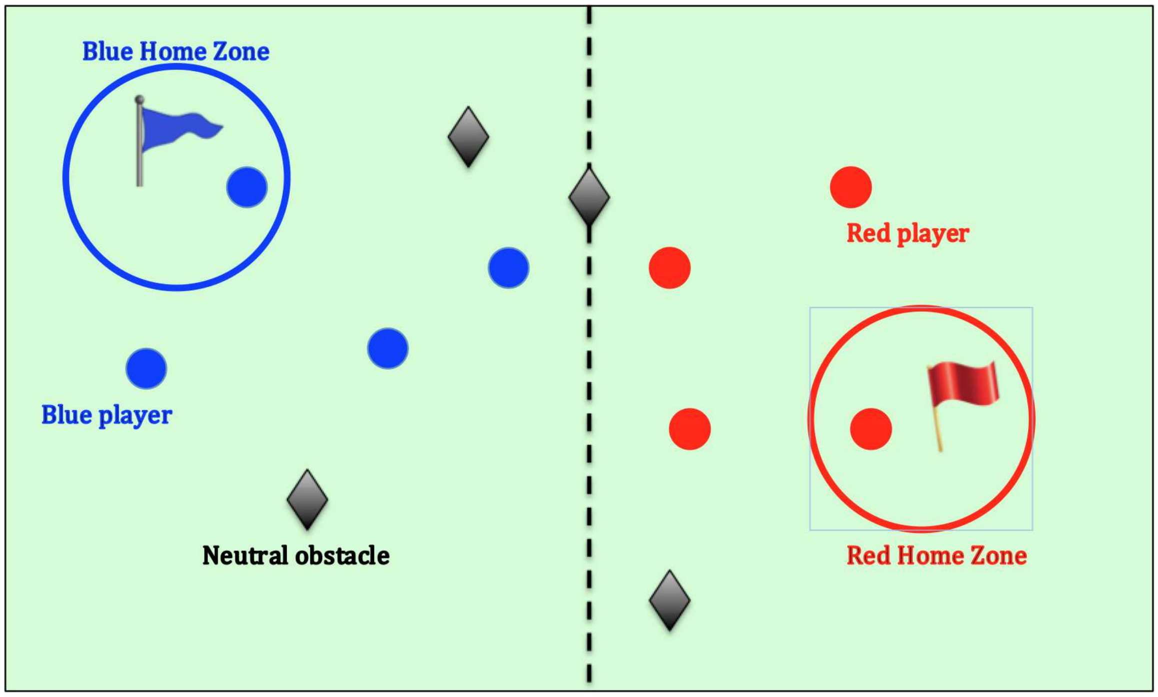


Figure 3.1: The RoboFlag playing field.

agent systems, distributed/decentralized planing and coordination, to hardware implementation and validation. These challenges must be achieved in an uncertain, and most likely adversarial, environment.

In this thesis, we work with a simplified version of this game in order to focus on particular research questions addressed in this thesis. In the simplified setup, the teams are named defenders (blue team) and attackers (red team). There are no flags that need to be captured. Instead, the attackers objective is to just infiltrate into the defenders' home zone (can be called *defense zone* later on). The defenders objective is to block attackers from infiltrating the home zone. The new simplified setup is depicted in Fig 3.2.

The objective in this work is to design a distributed decision making algorithm to allow defenders take meaningful local decisions (no central controller) subject to real-time constraints (e.g. fast decisions in milliseconds), limited information exchange, and probably with communication bandwidth limitations. In particular, the focus is on how the defenders generate meaningful paths locally in real-time in the presence of environmental uncertainties e.g. moving attackers. The attackers are allowed to commit to arbitrary decision strategies either automated or controlled by a human operator.

Next, related works on multi-vehicle path planning is briefly summarized. An optimization method for multi-vehicle path planning is based on the notion of coordination variables and coordination function [67]. In this approach, the information that must be shared to for cooperation is collected in single vector quantity called the coordination variable. The coordination function quantifies how the change in coordination variable affects the individual's myopic objectives. Trajectories are generated such that obstacle avoidance and timing constraints are satisfied. However, the locations of the considered threats are known.

Other papers on mission planning of unmanned aerial vehicles construct Voronoi-based paths from the currently known locations of the threats. Several combinations of paths from a start point to a target are provided by the Voronoi edges. Among those, one may select either the lowest-cost flyable path [68], or the path that minimizes exposure to radar [69]. A stochastic approach may also be applied, where a probability of a threat or target is assumed to be known [70]. In such approach, a chosen path minimizes the probability of getting captured or disabled by a threat. Using this method, global strategies may be computationally inefficient, while the path generated by the strategy can get in a limit cycle [70].

There are several classes of multi-robot systems that can be considered as hybrid systems with linear dynamics subject to linear inequalities involving real and integer variables. One approach called *model predictive control* (MPC) can be used to sole such problems, where an optimization problem is solved using based on a prediction of the system behavior over certain time horizon [71].This approach is useful when the prediction of the future system's behavior cannot be accurate due to, for example, future disturbances. In this case, the optimization is re-executed based on a new observation or measurements, and a new solution is used, which benefits from the feedback action.

Other multi-vehicle classes have their optimization as a *mixed integer linear programming* (MILP) problem. The optimization may combine task assignment and path planning subjected to UAV dynamics constraints, collision or obstacle avoidance and timing constraints [72, 73, 74]. Collision avoidance constraints guarantee that vehicles do not collide with each other or with obstacles, which can be static or moving according to a predefined motion model [73, 74].

Mixed integer linear program has been tested in the *RoboFlag* competition similar to the *capture-the-flag* competition which is described earlier in this introduction [75, 76]. The objective is to compute a set of control inputs that minimizes the number of opponents that enter their protected zone. The adversaries are considered UAVs with deterministic motion described by a linear model. In addition, the approach is computationally prohibitive, since the problem is not scalable in the number of agents.

In the following section, we provide the main mathematical modeling of the game setup that will be used in this thesis. Further modification in the modeling part will be presented as needed in later chapters.

3.2 Modeling

In this section, we provide definitions of the elements of the adopted problem setup. The following definitions will be general with no specific implementation details. In later chapters, the exact implementation of those definitions are presented as needed. For example, a control input u that belongs to a certain input set $\mathcal{U}$ is defined, but the structure of the $\mathcal{U}$ is not specifically mentioned yet. In later chapters, a more specific details on how, for example, u and $\mathcal{U}$ are implemented is presented.

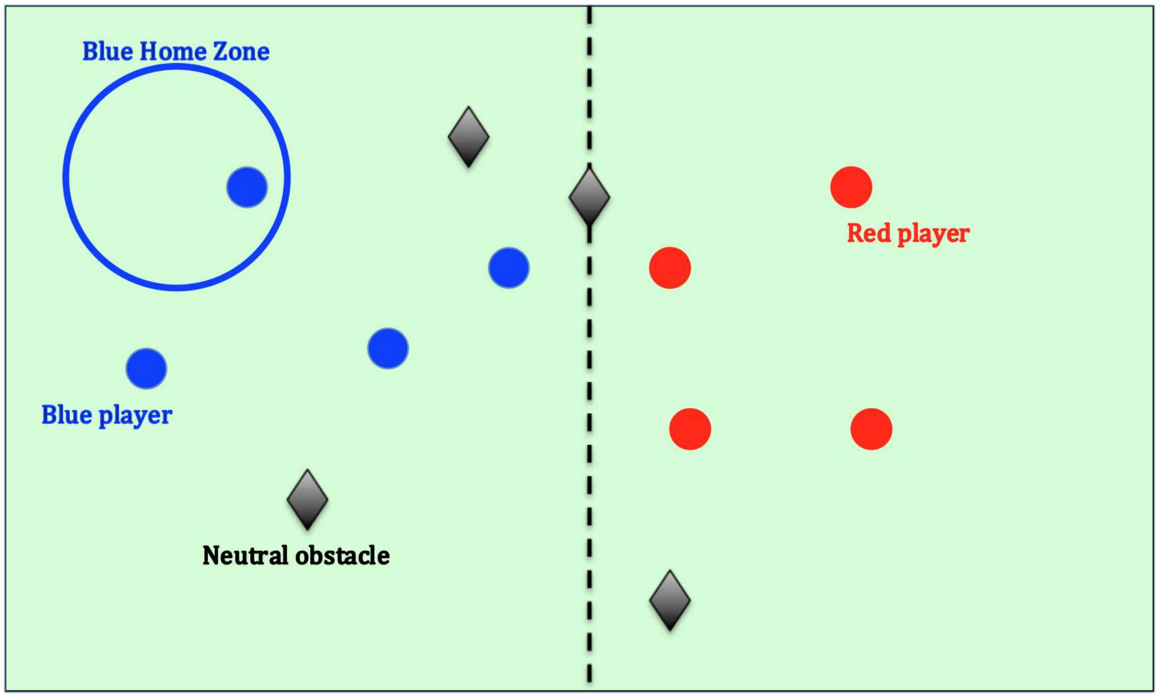


Figure 3.2: The simplified adversarial game setup used in this thesis.

Now, let's introduce the problem setup. As mentioned, the game is composed of two teams called *defenders* and *attackers* that battle against each other. *Agents* have certain *states* to describe their locations in *environment* over time. An agent's state evolves according to certain *dynamics* which is a function of the current state x and *action* u (or control input). The game runs according to some *rules*. These elements are explained as follows.

- **Environment**: This is the space where agents navigate, and is assumed to be bounded. The spacial model of the environment is represented by a *grid world* where a bounded continuous space, for example in $\mathbb{R}^2$, is discretized into a finite number of regions called *sectors*. So, the spacial model of the environment (or the map) is defined by a collection of sectors denoted by $\mathcal{S} = \{s_1, \cdots, s_{n_s}\}$ where n_s is the total number of sectors. With this model, there are special sectors defined as follows.

  - **Base**: Base sectors define the locations that correspond to the defender's base zone, which should be protected against attackers team. So the base is defined by a collection of sectors denoted by $\mathcal{S}_{\text{base}}(t) \subset \mathcal{S}$. If base locations are static over time, the time index is dropped, $\mathcal{S}_{\text{base}}$.

  – **Obstacles**: Those are locations that should be avoided by all agents at all times to avoid collision. Sectors that represent obstacle locations are denoted by $\mathcal{S}_{\text{obstacle}}(t) \subset \mathcal{S}$. If the obstacles set is static over time, the time index is dropped, $\mathcal{S}_{\text{obstacle}}$.

- **Agents**: Firstly, *agents* and *players* will be used interchangeably. Two teams are considered, *defenders*, and *attackers*. Each team consists of a collection of players (agents) denoted by $\mathcal{P}^r = \{p_1^r, \cdots, p_{n_r}^r\}$, where $r \in \{d, a\}$ is the team type, and n_r is the total number of players in the respective team r.

- **States**: States in this game setup provide a representation of the location of players in the grid S over time. The state of agent i of team r at time t is denoted by $x_i^r(t)$. The state of an agent at time t may be constrained, for example, to respect some dynamics,

$$x_i^r(t) \in \mathcal{X}_i^r(t) \tag{3.1}$$

where $\mathcal{X}_i^r(t)$ is a set of possible states at t. One representation of $x_i^r(t)$ is the sector number that agent i of team r occupies at time t. We will another representation of the state in Section 4.3. $\mathbf{x}^r(t) = (x_1^r, \cdots, x_{n_r}^r)$ is a vector that represents the joint state of team r.

- **Actions**: Firstly, we will use *actions* and *control* inputs interchangeably. Actions describe how players are transfered between sectors in $\mathcal{S}$. For a player i, we define $u_i^r(t)$ which is the action of agent i in team r at time t. $\mathbf{u}^r = (u_i^r, \cdots, u_{n_r}^r)$ defines a vector of the team's joint action. Each agent's action may belong to a set of admissible controls

$$u_i^r(t) \in \mathcal{U} \tag{3.2}$$

- **Dynamics**: Describes how the agents states evolve over time. In general, we can model each team's state $\mathbf{x}^r$ as follows:

$$(\mathbf{x}^r)^+ = f^r(\mathbf{x}^r, (u)^r) + f^{\text{attr}}(\mathbf{x}^r, \mathbf{x}^{-r}) \tag{3.3}$$

  where $(\mathbf{x}^i)^+$ is the updated state, $f^r((x)^r, \mathbf{u}^r)$ is a transition function, generally non-linear, $f^{\text{attr}}(\mathbf{x}^r, \mathbf{x}^{-r})$ is an attrition function that would describe how the state is affected in the case an agent is captured which is generally non-linear function. $-r$ represents the opposite team.

- **Rules**: The evolution of agents across the environment should follow the dynamics (3.3) and satisfy states and inputs constraints (3.1) and (3.2), respectively. An enemy is considered captured if it is within a predefined distance from a defender, *i.e* $d(x^e, x^d) \leqslant d_{\text{capture}}$ for some distance measure $d(\cdot)$.

- **Objectives**: The defenders objective is to protect the base zone $\mathcal{S}_{base}$ against the intrusion of the attackers, e.g. $\{x_i^a\} \cap \mathcal{S}_{base} = \phi$. The attackers objective is to infiltrate the base zone of the defenders, e.g. $\{x_i^a\} \cap \mathcal{S}_{base} \neq \phi$.

- **Time**: In this thesis, we will work with discrete time scale $t = 0, 1, \cdots$ for the case of an infinite time horizon. For a finite time horizon, $t = 0, 1, \cdots, T$, for some $T < \infty$.

3.3 Summary

The motivating problem that is used as a guiding case in this work is an adversarial game between two teams of players (or agents) called *defenders* and *attackers*. Defenders protects their base zone against attackers infiltration while avoiding obstacles and inter-collision. This setup brings several research question, for example, generating fast local planning decisions within limited computation and communication resources.

The objective in this thesis is to design algorithms that allow defenders to take meaningful local decisions (no central coordinator) in real-time (e.g. milliseconds) within limited computation and communication resources.

In this chapter, a general modeling of the problem setup is presented, and specific details and model implementations are discussed in next chapters. In the next chapters, we will start investigating how to solve this problem, i.e. generate optimal decisions for defenders to block attackers from infiltrating into the base zone. Centralized solutions are firstly discussed in Chapter 4 followed by the proposed distributed approaches in Chapter 5.

Chapter 4

Centralized Solutions

4.1 Introduction

Let's recall that we are working with an adversarial game between two teams of robots, defenders and attackers. On the defender's side, the objective is to find optimal policies to block attackers from infiltrating the base zone, see figure 3.2. This also can be considered as path planning problem where we seek to find the optimal paths (sequence of locations) of defenders that minimizes the infiltrations of attackers into the base zone over certain time horizon. The final target is to let the defenders generate local decisions on their own without relying on a central coordinator and based on limited information exchange. However, we start discussing how we can generate optimal policies using centralized approaches in the first place. The limitations of centralized approaches are, then, presented before we start the discussion on designing distributed real-time approaches.

In this chapter, two centralized approaches are discussed. First, the problem is formulated as a dynamic program (DP) which is a well established general framework for solving any sequential decision making problem. The dynamic program approach provides a framework to compute the optimal policy for the defenders decisions. The practical limitations of such approach are then presented. Next, an alternative centralized formulation that is appealing for on-line and real-time implementation is presented. This is an incentive approach that is based on linear program which provides generally suboptimal, but meaningful, policies compared to the dynamic program approach. In the next chapter, the distributed approach is presented.

4.2 Dynamic Programming Approach

The problem of interest is a sequential decision problem as defenders locatoins has to be decided at each time step over certain time horizon. Dynamic program can find the optimal policy for such problems [16]. In other words, we seek to find an optimal policy $\mu^d(\mathbf{x}^d)$ for the defenders to maximize time before an enemy infiltrates the base zone, $\mathcal{S}_{\text{base}}$. If such optimal policy exists, defenders can generate their decisions at time t by committing to action $\mathbf{u}^d(t) = \mu^d(\mathbf{x}^d(t))$. Several works used DP to solve similar path planning problems for cooperative autonomous vehicles, for example see [77, 12, 78]. To use DP to solve this problem, we need to define the *states*, *state transition*, *actions*, and *cost* (or *reward*).

- **States**: Each agent i in team $r \in \{d, a\}$ has a state $x_i^r(t)$ that represents its sector location in $\mathcal{S}$. The defenders team state is defined as $\mathbf{x}^d(t) = (x_1^d, \cdots, x_{n_d}^d)$ represents the sectors locations which the defenders team occupies. Similarly, $\mathbf{x}^a(t) = (x_1^a, \cdots, x_{n_a}^a)$ represents the sector locations which the attackers team occupies. For example, if there are two defenders at sectors $\{1, 10\}$ and one attacker at sector $\{20\}$, then $\mathbf{x}^d(t) = (1, 10)$, and $\mathbf{x}^a(t) = (20)$. The augmented state is denoted by $\mathbf{x}(t) = (\mathbf{x^d(t)}, \mathbf{x^a(t)}) \in \mathcal{S}^{n_d+n_a}$.

- **Actions**: Actions represent direction that an agent can take at any time. Namely, let the set of admissible actions at any time be $\mathcal{U} = \{$East, West, North, South, North-East, North-West, South-East, South-West, Stay$\}$. An agent i in team $r \in \{d, a\}$ can commit an action at time t which is denoted by $u_i^r(t) \in \mathcal{U}$. The join action of a team is denoted by $\mathbf{u}^r(t) = (u_1^r, \cdots, u_{n_r}^r)$ and the augmented action vector which combines both teams actions is denoted by $\mathbf{u}(t) = (\mathbf{u}^d(t), \mathbf{u}^a(t)) \in \mathcal{U}^{n_d+n_a}$.

- **Dynamics**: This describes how agent transition from stat $x^r(t)$ to state $x^r(t+1)$ after committing action $u^r(t)$. The dynamics for each time are generally captured by Eq. 3.3.

- **Stage reward**: This is the reward that defenders get as long as attackers have not infiltrated the base zone. We are interested in maximizing time before an attacker infiltrates the base zone. With that objective, a reasonable stage reward $g(\cdot)$ can be defined to receive a reward as long as attackers are not in the base zone. An example stage reward is defined as follows.

$$g(\mathbf{x}^d(t), \mathbf{u}^d(t), \mathbf{x}^a(t), \mathbf{u}^a(t)) = \begin{cases} 1 \ \text{ if } \mathbf{x}^a \cap \mathcal{S}_{\text{base}} = \phi \\ 0 \ \text{ if } \mathbf{x}^a \cap \mathcal{S} \neq \phi \end{cases} \tag{4.1}$$

The objective is to maximize time before an attacker gets into the base zone. We also assume that attackers will commit to a *worst-case* strategy in order to minimize the cumulative rewards of defenders. In other words, let defenders be at state $\mathbf{x}^d$, attackers will commit to a strategy that minimizes the cumulative reward of defenders. This can be interpreted as the maximizing the infinite discounted sum of the stage rewards as follows.

$$\max_{\mu^d} \min_{\mu^a} \sum_{t=0}^{\infty} \alpha^t g(\mathbf{x}^d(t), \mathbf{u}^d(t), \mathbf{x}^a(t), \mathbf{u}^a(t)) \tag{4.2}$$

Now let's define $J(\mathbf{x})$ as the cost-to-go and $J^*(\mathbf{x})$ as the optimal cost-to-go evaluated at the augmented state $\mathbf{x}$. In DP, a relationship of interest is the *Bellman's* equation. It provides a way for computing the optimal cost-to-go based on the *principal of optimality*, see [16]. *Bellman's* equation is described as follows.

$$J^*(\mathbf{x}(t)) = \max_{\mu^d} \min_{\mu^a} \{ g(\mathbf{x}^d(t), \mathbf{u}^d(t), \mathbf{x}^a(t), \mathbf{u}^a(t)) + \alpha J^*(\mathbf{x}(t+1)) \} \tag{4.3}$$

Eq 4.3 can be solved using a linear program over the whole state space $\mathbf{x} \in \mathcal{S}^{n_d+n_a}$ and input space $\mathbf{u} \in \mathcal{U}^{n_d+n_a}$, see [16]. The optimal policy can be obtained using $\mu^*(\cdot) = \text{argmax } J^*(\cdot)$.

4.2.1 Computational Example

From (3.1) and (3.2), we notice that the complexity of the state space scales exponentially in the number of agents, and, therefore, becomes intractable for large number of agents. In the following numerical example, the DP algorithm is tested to evaluate the optimal policy and the computational effort.

The DP algorithms is tested on the following game,

- **Environment**: 5×5 grid world, with $n_s = 25$. The sectors collection $\mathcal{S} = \{s_1, \cdots, s_{25}\}$.
- **Agents**: Two defenders $n_d = 2$, and one attacker $n_a = 1$.
- **Actions/controls**: Defenders control actions are defined as $\mathbf{u}^d = (u_1^d, u_2^d) \in \mathcal{U}^d = \{1, 2, . \cdots, 9\}^2$ which maps to {East, West, North, South, North-East, North-West, South-East, South-West, Stay} with cardinality of $|\mathcal{U}^d| = 81$. Each agent is assumed to move maximum of one square in any direction at one time step. The enemy's control action $u^e = (u^{e_1})$ is defined similarly.
- **States**: The augmented state (for two defenders and one enemy) $\mathbf{x} = (x_1^d, x_2^d, x_1^a) \in \mathcal{X}$, where $\mathcal{X} = \{1, 2, . \cdots, 25\}^3$ is the state space of cardinality $|\mathcal{X}| = 15625$.
- **Rules**: Agents can move at most one sector ahead per time step k. An enemy is captured if $d(x_i^d, x_j^a) < 1$ where $d(\cdot)$ is the number of sectors attacker j is away from defender i.
- **Base zone**: Defender has to protect the base sectors defined as $\mathcal{S}_{\text{base}} = \{s_{21}, s_{22}\}$.

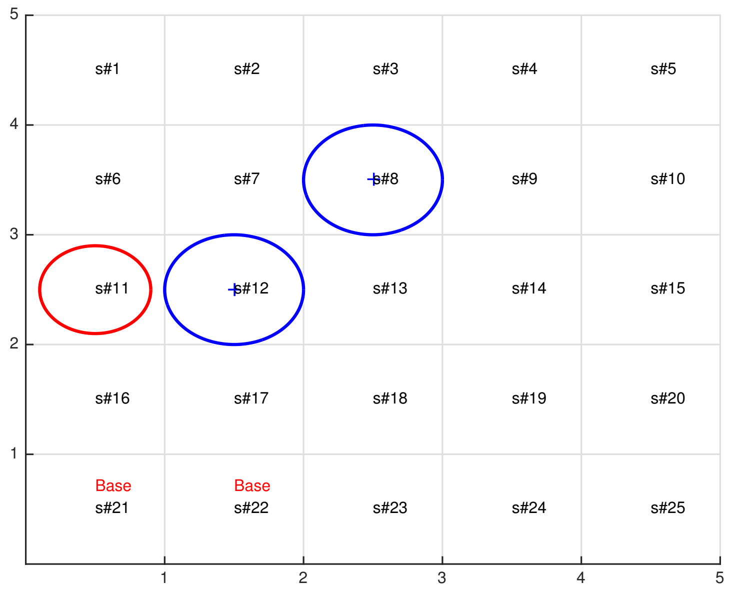


Figure 4.1: 25-sector grid. Blue circles are defenders in sectors s_8, s_{12}, and red circle is the attacker in sector s_{11}. Base sectors marked as "Base" in red are s_{21}, s_{22}. This snapshots shows a captured enemy since the enemy is one sector away from a defender.

- **Dynamics**: Agents are assumed to move on cell away in each time step, in any direction $u \in \mathcal{U}$

Figure 4.2.1 depicts this example.

The optimal policy $(\mu^d(\mathbf{x}))^*$ was computed offline using `2.5 GHz Mac laptop with 4 cores, 16GB RAM`, and took approximately 4 hours. Then, the policy is stored in a look-up table. During the game, defenders observe the game state (their sectors locations as well as attackers) $\mathbf{x} = (\mathbf{x}^d, \mathbf{x}^a)$, and they commit to the corresponding optimal action in the look-up table, $\mathbf{u}^d = \mu^d(\mathbf{x})$. Although the optimal policy can be stored and used in real-time, any modification in the grid size, defense sectors, and/or number of agents in both teams, would require recalculation of the optimal policy μ^d, which obviously not practical to be computed in real-time. The presented example is also a relatively small, and for larger setups, the computations get exponential in the number of agents and, hence, prohibitive.

Because of the curse of dimensionality of the path planning problem using dynamic program, we are forced (in practice) to accept reasonable and meaningful suboptimal routing policies. To overcome the curse of dimensionality in the DP approach, usually approximate methods are used. There are some works related to path planing in multi-vehicle settings. For example, in [79, 12], the authors use approximated dynamic programing approach for application of UAVs to search certain bounded area. The approach uses limited look-ahead horizon that is assumed to be much smaller than the actual time horizon, in order to reduce the the search complexity. It also adds a utility function to the stage gain in order to reduce the possible interference (duplicate visits of locations) of UAVs. In [80], the authors propose an approximate dynamic programing approach for multi-platform path planning. The approach simplifies a stochastic DP problem to a deterministic one that reduces the search space.

For an online calculation of optimal actions/controls and more scalable against changes in the number of agents, a linear program approach for multi-vehicle path planning with adversaries is proposed in [10]. The work in this thesis is build upon this approach which is discussed in the following section.

4.3 Linear Program Approach

In this section a different formulation of the adversarial game is presented. The approach assumes linear formulation of the objective function and the dynamics of agents motion in the world grid associated with some linear constraints. This leads to a linear program formulation which dramatically boosts the computation speeds and make it more appealing for real-time and online execution even on micro-controllers with limited computation power. In the following section, the game setup is described and the linear program solution is presented. Keep in mind that this is still a centralized approach. In the next chapter, a distributed version of this linear program approach is presented.

4.3.1 State Space Model

The model description of the game setup presented here is for the same environment setup that is discussed before. Agents in both teams (defenders and attackers) evolve in grid world represented by a collection of sectors $\mathcal{S}$ in discrete time. There is a base zone (or defense zone) to be protected by defenders and defined as $\mathcal{S}_{base} \in \mathcal{S}$. A collection of reference sectors $\mathcal{S}_{ref}$ represents sectors that possibly surround the base zone for surveillance purposes.

4.3.1.1 State and Actions

The state in this model represents the level of occupancy of each sector in the grid. The state of each sector s_i is $x_{s_i} \in \mathbb{Z}_+$, which is the number of its occupants, where $\mathbb{Z}_+$ is the set of non-negative integers. The state of the grid (all sectors) with respect to an agent i in team $r \in \{d, a\}$is

$$x_i^r = [x_{s_1,i}^r \dots x_{s_{n_s},i}^r]^T \in \mathcal{B}^{n_s},$$

where $\mathcal{B} \triangleq \{0, 1\}$ and

$$x_{s_j,i} = \begin{cases} 1 & \text{agent} i \text{ occupies } s_j \\ 0 & \text{otherwise.} \end{cases}$$

The state is restricted to be in $\{0, 1\}$ in order to have one agent per sector at any time, for example for collision avoidance purposes.

The state of the grid with respect to all the agents in team $r \in \{d, a\}$ is

$$\mathbf{x}^r = \sum x_i^r$$

Actions (or controls) describe the transfer of agents (or quantity of occupancy) between sectors in $\mathcal{S}$. For an agent i, we define an input vector that describes all transfers from sector s_j to sector s_k as follows.

$$u_{s_j \to s_k, i} = \begin{cases} 1 & x_{s_i,i} = 1 \text{ and } i \text{ transfers to } s_k \\ 0 & \text{otherwise.} \end{cases}$$

An augmented input vector for agent i is

$$\mathbf{u}_{s_j,i} = [u_{s_j \to s_1,i} \dots u_{s_j \to s_{j-1},i} u_{s_j \to s_{j+1},i} \dots \\ u_{s_j \to s_{n_s},i}]^T. \quad (4.4)$$

If agent i occupies sector s_j, then $\mathbf{u}_{s_j,i} \in \mathcal{B}^{(n_s-1)}$ represents its transfer to some other sector in the grid. The complete input vector of agent i in team $r \in \{d, a\}$ with respect to all the sectors in the grid is

$$\mathbf{u}_i^r = [(\mathbf{u}_{s_1,i}^r)^T \ (\mathbf{u}_{s_2,i}^r)^T \dots (\mathbf{u}_{s_{n_s},i}^r)^T]^T \in \mathcal{B}^{n_u}.$$

where $n_u = n_s(n_s - 1)$. Similar to states, the input vectors for defenders and attackers are

$$\mathbf{u}^d = \sum \mathbf{u}_i^d \quad \mathbf{u}^a = \sum \mathbf{u}_i^a$$

4.3.1.2 Dynamical Model and Constraints

The evolution of the state of s_j with respect to agent i is modeled by the following linear model.

$$x_{s_j,i}^+ = x_{s_j,i} + \sum_{s_k \in \mathcal{N}(s_j)} u_{s_k \to s_j,i} - \sum_{s_k \in \mathcal{N}(s_j)} u_{s_j \to s_k,i} \quad (4.5)$$

where $x_{s_j,i}^+$ is the updated state of sector s_j with respect to agent i. The first and the second terms in (4.5) represent the flow of agent i into and out of sector s_j,

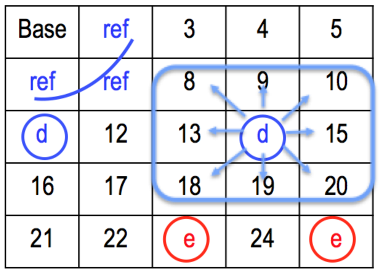


Figure 4.2: Representation of the battlefield. The defense zone is $\mathcal{S}_{\text{base}} = \{s_1\}$ labeled 'Base' and $\mathcal{S}_{\text{ref}} = \{s_2, s_6, s_7\}$ labeled 'ref'. There are two defenders that are located at sectors s_{11} and s_{14}. Neighborhood of the defender at sector s_{14} is represented by the arrows pointing to sectors $\{s_8, s_9, s_{10}, s_{13}, s_{15}, s_{18}, s_{19}, s_{20}\}$. Attackers are labeled as 'e' and are located at sectors s_{23} and s_{25}.

respectively, with respect to the neighborhood set of s_j defined by $\mathcal{N}(s_j) \in \mathcal{S}$. The neighborhood set of a sector s_j is the set of all sectors that surrounds it (see Figure 4.2). The input $u_{s_j \to s_j}$ is omitted as it coincides with the new state $x^+_{s_j,i}$.

The state of a sector is constrained to be non-negative (only positive resources). This constraint is defined as

$$x_{s_j,i} \in \mathcal{B} \triangleq \{0,1\},$$

$$0 \leqslant \sum_{s_k \in \mathcal{N}(s_j)} u_{s_j \to s_k,i} \leqslant x_{s_j,i} \tag{4.6}$$

The previous inequality constraint simply means that all out-transfers from a sector s_j should not exceed the current quantity available in that sector.

The dynamical model (4.5) and the constraints (4.6) for a team $r \in \{d, a\}$ can be written in a compact form as

$$\begin{cases} (\mathbf{x}^r)^+ = \mathbf{x}^r + \mathbf{B}_{\text{in}}\mathbf{u}^r - \mathbf{B}_{\text{out}}\mathbf{u}^r = \mathbf{x}^r + \mathbf{B}\mathbf{u}^r \\ \mathbf{0} \leqslant \mathbf{B}_{\text{out}}\mathbf{u}^r \leqslant \mathbf{x}^r \\ \mathbf{x}^r \in \mathcal{B}^{n_s}, \ \mathbf{u}^r \in \mathcal{B}^{n_u} \end{cases} \tag{4.7}$$

for appropriate $\mathbf{B}_{\text{in}}$ and $\mathbf{B}_{\text{out}}$ ($\mathbf{B} = \mathbf{B}_{\text{in}} - \mathbf{B}_{\text{out}}$). We refer the reader to [10] for further details on the problem setup.

4.3.1.3 Adversarial Model

The adversarial behavior in the game is modeled as attrition. Attrition can be interpreted as the result of collision with opponent team. The objective of each team can be maximization of such attrition to their opponents. Attackers motion are modeled similar to defenders' model in 4.7. By adding attrition, the model can be updated for both teams as follows.

$$\begin{cases} (\mathbf{x}^r)^+ = \mathbf{x}^r + \mathbf{B}_{\text{in}}\mathbf{u}^r - \mathbf{B}_{\text{out}}\mathbf{u}^r = \mathbf{x}^r + \mathbf{B}\mathbf{u}^r - \mathbf{d}^r(\mathbf{x}^r, \mathbf{x}^{-r}) \\ \mathbf{0} \leqslant \mathbf{B}_{\text{out}}\mathbf{u}^r \leqslant \mathbf{x}^r \\ \mathbf{x}^r \in \mathcal{B}^{n_s},\ \mathbf{u}^r \in \mathcal{B}^{n_u} \end{cases} \tag{4.8}$$

Where $-r$ is the opponent of r.

4.3.1.4 Model Simplifications

There are few obstacles that need to be handled before a linear program formulation can be done based on 4.8. These are

- The attrition function is generally nonlinear.
- the control vector of attackers $\mathbf{u}^a$ is not known.
- the states and controls are integers, namely, binaries, which qualifies for linear *mixed integer* program, but not a linear program.

In [81], a similar model was proposed with linear model approximation based on the following simplifications which lead to a linear program formulation.

- First, the attrition function is removed from 4.8 and the model becomes

$$\begin{cases} (\mathbf{x}^r)^+ = \mathbf{x}^r + \mathbf{B}_{\text{in}}\mathbf{u}^r - \mathbf{B}_{\text{out}}\mathbf{u}^r = \mathbf{x}^r + \mathbf{B}\mathbf{u}^r \\ \mathbf{0} \leqslant \mathbf{B}_{\text{out}}\mathbf{u}^r \leqslant \mathbf{x}^r \\ \mathbf{x}^r \in \mathcal{B}^{n_s}, \ \mathbf{u}^r \in \mathcal{B}^{n_u} \end{cases} \tag{4.9}$$

- Next, it is assumed that the attackers implement an *assumed* feedback control policy such as

$$\mathbf{u}^a = \mathbf{G}^a\mathbf{x}^a \tag{4.10}$$

  Where $\mathbf{G}^a \in \mathcal{B}^{n_u \times n_s}$

Due to these simplifications, the simplified model in 4.9 is expected to be largely different from the actual model in 4.8. For this, a *receding horizon* strategy [71] is implemented in order to overcome this discrepancy. This is demonstrated in more details in the following sections.

4.3.1.5 Enemy Modeling

The attackers' feedback matrix $\mathbf{G}^a$ contains assumed information about the anticipated behavior of attackers. It is generally unknown, but introduced for sake of prediction in the optimization problem which will be defined later. $\mathbf{G}^a$ can represent the following behavior.

- an assumed anticipated paths of attackers
- attackers behavior as diffusion over the grid
- probability map of attackers evolution

More specifically, a control input that represents a transfer from sector s_j to sector s_k can be written as

$$u_{s_j \to s_k} = g_{s_j \to s_k} x_{s_j} \tag{4.11}$$

Where $g_{s_j \to s_k} x_{s_j}$ is the assumed feedback. By setting $g_{s_j \to s_k} x_{s_j} \in \{0, 1\}$, it represents the next anticipated location of attackers. To represent diffusion behavior, i.e. an agent at sector s_j can split into more than one sector in the next step, the feedback can be set as $g_{s_j \to s_k} x_{s_j} < 1$. It also can be interpreted as a probability that of moving from sector s_j to sector s_k. In all cases, it can be represented as follows.

$$\begin{cases} g^a_{s_j \to s_k} \in [0, 1] \\ \sum_{s_k \in \mathcal{N}(s_j)} g^a_{s_j \to s_k} \leqslant 1 \end{cases} \tag{4.12}$$

A numerical example of $\mathbf{G}$ be found in [10].

4.3.2 Optimization Setup

The simplified system of agents (vehicles) is described by a system of linear equations and constraints described by 4.9 and 4.10. Next, a linear objective function is introduced in order to allow the formulation and use of linear program in order to derive optimal paths agents may follow. The optimal paths are described by a sequence of states over certain finite time horizon. In the following, the linear program elements such as the objective function and constraints are derived.

4.3.2.1 Objective Function

For each team $r \in \{d, a\}$, an objective vector is defined over a finite time horizon T_p as follows.

$$\mathbf{X}^r = \begin{pmatrix} \mathbf{x}^r[1]^T & \mathbf{x}^r[2]^T & \cdots & \mathbf{x}^r[T_p]^T \end{pmatrix}^T$$

where $\mathbf{x}^r[t] \in \mathcal{B}^{n_s}$ is the state vector at future time t

Considering an adversarial environment, possible objective can be:

- **Evasion**: By minimization of intercepted defenders vehicles
- **Pursuit**: By maximization of intercepted attackers vehicles.
- **Surveillance**: By tracking a reference state vector.

Now, a linear objective function can be represented in the following form.

$$\min_{\mathbf{x}^d[t]} \left(\sum_{t=1}^{T_p} \left[\alpha^d \cdot \mathbf{x}^a[t] + \beta^d \cdot \mathbf{x}^d_{\text{ref}}[t] \right]^T \cdot \mathbf{x}^d[t] \right)$$

which can be formulated in a compact form as follows,

$$\min_{\mathbf{X}^d} \left[\alpha^d \cdot \mathbf{X}^a + \beta^d \cdot \mathbf{X}^d_{\text{ref}} \right]^T \cdot \mathbf{X}^d \tag{4.13}$$

where $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}$ are constants. When $\alpha^d > 0$, the inner product $< \mathbf{X}^a, \mathbf{X}^d >$ induces an *evasion* behavior i.e. defenders will avoid attackers. On the other hand, if $\alpha^d < 0$, the inner product $< \mathbf{X}^a, \mathbf{X}^d >$ increases for an optimized state $\mathbf{X}^d$ and induces an *pursuit* behavior i.e. defenders will chase (or capture) attackers. If a *surveillance* behavior is desired, then $\beta^d < 0$ in order to increase $< \mathbf{X}^d_{\text{ref}}, \mathbf{X}^d >$. A higher magnitude of β forces the defenders to remain in $\mathcal{S}_{\text{ref}}$ and protect $\mathcal{S}_{\text{base}}$ from there. The coefficients α and β are selected to assign the desired weights to each of the behavior and they can be chosen to satisfy $|\alpha|, |\beta| \in [0, 1]$ and $|\alpha| + |\beta| = 1$.

4.3.2.2 Constraints

The linear objective function in 4.13 should satisfy the dynamics and constraints defined in 4.9 and 4.10. More specifically, the following equations must be satisfied for all $t \in \{0, 1, \cdots, T_p\}$.

$$\mathbf{x}^d[t+1] = \mathbf{x}^d[t] + \mathbf{B}^d \cdot \mathbf{u}^d[t]$$

Define a control vector over the time horizon $\{0, 1, \cdots, T_p - 1\}$,

$$\mathbf{U}^d = \begin{pmatrix} \mathbf{u}^d[0]^T & \mathbf{u}^d[1]^T & \cdots & \mathbf{u}^d[T_p - 1]^T \end{pmatrix}^T$$

For appropriate $\mathbf{T}_{xx_0}$ and $\mathbf{T}_{xu}$, the above dynamics can be equivalently written as,

$$\mathbf{X}^d = \mathbf{T}_{xx_0} \cdot \mathbf{x}^d[0] + \mathbf{T}_{xu} \cdot \mathbf{U}^d \tag{4.14}$$

The control vector $\mathbf{u}^d[t]$ must satisfy,

$$\mathbf{B}_{\text{out}} \cdot \mathbf{u}^d[t] \leqslant \mathbf{x}^d[t] \tag{4.15}$$

For appropriate $\mathbf{T}_{xu,c}, \mathbf{T}_{xx_0,c}$, the input constraints can be written as,

$$\mathbf{T}_{xu,c} \cdot \mathbf{U}^d \leqslant \mathbf{T}_{xx_0,c} \cdot \mathbf{x}^d[0] \tag{4.16}$$

In addition, obstacle avoidance can be represented as a linear constraint as follows,

$$\mathbf{X}_{\text{obs}}^T \cdot \mathbf{X}^d = 0 \tag{4.17}$$

where $\mathbf{X}_{\text{obs}} \in \mathcal{B}^{T_p n_s}$ is a vector where the entries corresponding to obstacle locations are $1'$s and 0 otherwise.

The details of the compact matrices mentioned above are discussed in Appendix A

4.3.2.3 Mixed-Integer Linear Program

The linear objective function in 4.13 and the constraints in 4.14, 4.16, and 4.17 form a *mixed integer linear program* optimization from defender's perspective and can be written as follows.

$$\begin{array}{ll} \min & \left[\alpha^d \mathbf{X}^a + \beta^d \mathbf{X}_{\text{ref}}\right]^T \mathbf{X}^d \\ \text{s.t.} & \mathbf{X}^d - \mathbf{T}_{xu}\mathbf{U}^d = \mathbf{T}_{xx_0}\mathbf{x}^d[0] \\ & \mathbf{T}_{xu,c}\mathbf{U}^d \leqslant \mathbf{T}_{xx_0,c}\mathbf{x}^d[0] \\ & \mathbf{X}_{\text{obs}}^T \mathbf{X}^d = 0 \\ \text{Variables} & \mathbf{X}^d \in \mathcal{B}^{T_p n_s},\ \mathbf{U}^d \in \mathcal{B}^{T_p n_u} \end{array} \tag{4.18}$$

The vector of attackers state $\mathbf{X}^a$ is not known. However, as previously discussed, an assumed feedback law $\mathbf{u}^a[t] = \mathbf{G}^a \cdot \mathbf{x}^a[t]$ is used to complete the linear program formulation. More specifically,

$$\mathbf{x}^a[t+1] = (\mathbf{I} + \mathbf{B}^a\mathbf{G}^a)^{t+1} \cdot \mathbf{x}^a[0] \tag{4.19}$$

For an appropriate $\mathbf{T}_{xx_0,G}$, the attackers state vector can be written in the following compact form.

$$\mathbf{X}^a = \mathbf{T}_{xx_0,G} \cdot x^a[0] \tag{4.20}$$

The details of constructing the compact matrices are discussed in Appendix A.

4.3.3 Linear Program Problem

The problem in 4.18 is a mixed-integer linear program problem where states and inputs take values on $\mathcal{B} = \{0, 1\}$. Although there are methods that solve such problems such as cutting plane, and branch and bound [82], a linear programing formulation is preferred as it provides faster computations. Therefore, problem 4.18 is transformed into a linear program after some relaxation steps explained as follows.

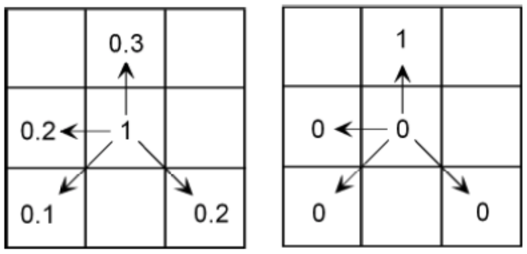
4.3.3.1 Relaxations

To formulate a linear program problem, the states and inputs are assumed to be $\mathbf{0} \leqslant \mathbf{X}^d, \mathbf{U}^d \leqslant \mathbf{1}$ instead of taking only binary values as in problem 4.18. If an optimal solution of the relaxed problem is feasible with respect to the mixed-integer problem, then it is also an optimal solution to the latter [82]. However, the relaxed problem can produce non-integer solutions which are not feasible to problem 4.18. Therefore, a suboptimal solution of the relaxed problem is constructed that is feasible to 4.18.

$$\begin{aligned} \min \quad & \left[\alpha^d \mathbf{X}^a + \beta^d \mathbf{X}^{\text{ref}}\right]^T \mathbf{X}^d \\ \text{s.t.} \quad & \mathbf{X}^d - \mathbf{T}_{xu}\mathbf{U}^d = \mathbf{T}_{xx_0}\mathbf{x}^d[0] \\ & \mathbf{T}_{xu,c}\mathbf{U}^d \leqslant \mathbf{T}_{xx_0,c}\mathbf{x}^d[0] \\ & \mathbf{X}_{\text{obs}}^T \mathbf{X}^d = 0 \\ & \mathbf{0} \leqslant \mathbf{X}^d \leqslant \mathbf{1} \\ & \mathbf{0} \leqslant \mathbf{U}^d \leqslant \mathbf{1} \end{aligned} \tag{4.21}$$

4.3.3.2 Suboptimal Solution

Let $(\mathbf{u}^*)^d[t] \in \bar{\mathcal{B}} = [0,1]$ be the relaxed optimal solution to the relaxed problem (linear program) for all $t \in \{0, 1, \cdots, T_p - 1\}$, which will be generally non-integer vector between $\mathbf{0}$ and $\mathbf{1}$. This implies that an agent in a sector can be split to several portions in the on-step reachable neighbor sectors. An example of such case is shown in Figure 4.3(a). To get a feasible solution, the control variable corresponding to the maximum split (portion) is assigned the value 1 and the rest are 0, see Figure 4.3(b) for an illustration. The suboptimal feasible solution is denoted by $\tilde{\mathbf{u}}^d[t]$



(a) (b)

Figure 4.3: (a) A possible optimal solution to the linear programming relaxation. (b) The corresponding integer suboptimal solution to the mixed integer programming problem.

4.3.3.3 Receding Horizon Implementation

Due to the modeling differences between the prediction and optimization models in 4.9, 4.10 and the actual linear model (or the controlled 'plant') in 4.8, significant discrepancies are expected. To compensate for that, the optimization result will be implemented in a receding horizon manner described in the following steps.

1. Defenders and attackers current states $\mathbf{x}^d$ and $\mathbf{x}^a$ are measured
2. The relaxed LP in 4.21 is solved and the first optimal control input $(\mathbf{u}^*)^d[0]$ is obtained
3. A feasible control input $\tilde{\mathbf{u}}^d[0]$ of the mixed-integer linear program is constructed
4. Apply $\tilde{\mathbf{u}}^d[0]$ and repeat.

4.4 Summary

In this chapter, the centralized modeling and solution to the adversarial game of interest is presented in two different setups. The objective is to derive optimal policy for the defenders to maximize the time before an attacker infiltrates to the defender's base zone. First, a dynamic programing approach is presented which provides the global optimal solution of the problem. However, it suffers from the curse of dimensionality in the number of agents, hence, the computations of the optimal policy even for moderate problem size (grid size and number of agents) become prohibitive. Second, a linear program approach is presented. Although it provides a suboptimal solution compared to the dynamic programing approach, it provides a fast online approach that is more practical for real-time implementation. In the next chapter, a distributed approach is proposed which allows to eliminate the central coordinator in the centralized setup while respecting some practical requirements such as limited computations and local information exchange.

Chapter 5

Distributed Solutions

5.1 Introduction

This chapters presents the proposed distributed approaches in this thesis. This is derived under the setting of the attackers-defenders adversarial game setup discussed earlier and utilizes the centralized linear program formulation presented in the previous chapter. The work in this chapter is also published in [83].

Distributed planing is desired in systems with multiple entities e.g. robots for number of reasons. First, the computation burden might be huge on a single central controller for large systems, and distributing the computation over multiple controllers can dramatically reduce the computations. Also, a centralized system is prone to what is called *single point of failure* which means that the system will completely fail once the central controller fails. On the other hand, using a distributed setup, allows the system to go under *graceful degradation.*

There are several distributed approaches that provide theoretical solutions for various problems setup as mentioned earlier in this thesis. However, we have observed that even for algorithms that are computationally light, e.g. gradient based algorithms, the implicit communication requirements are huge. In other words, each agent is required to exchange information, e.g trajectories of the states, iteratively many times, with its neighbors in order to converge to an optimal solution. The transfer of state trajectories iteratively many times between agents in order to take a single decision for a single time step imposes a lot of burden and trust on the

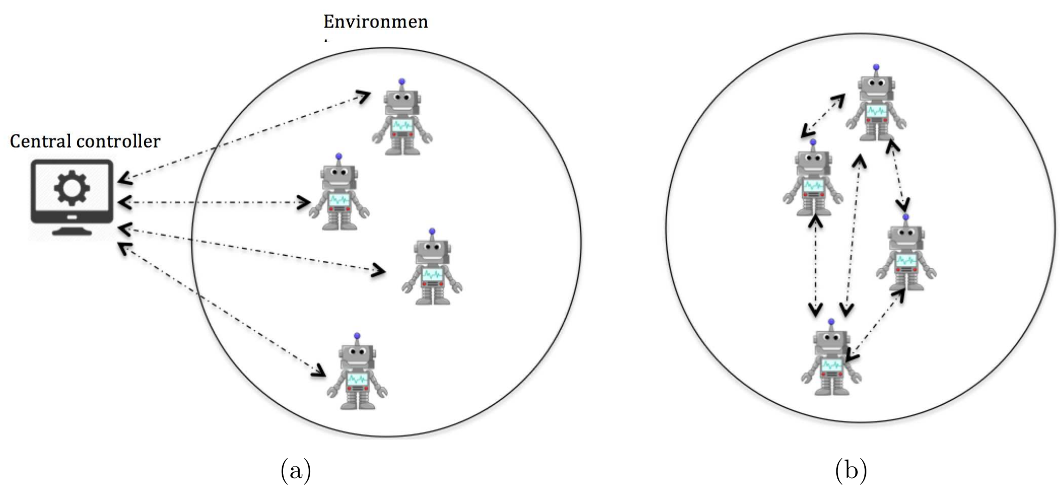


Figure 5.1: (a) depicts a centralized control setup. If the central controller fails, all system fails. (b) shows an example of distributed control setup where each robot has its own controller and communicates with its neighbors if needed. If a robot fails the system still functions, but with probably reasonable degraded performance.

communication resources which also is proportional to energy consumption. It also can introduce delays for a real-time decision making. For example, in our defenders-attackers game, each robot needs to compute its decision (its next location) based on its current location and its neighbors location, in a fraction of a second (e.g. maximum of 50 millisecond), denoted by T_c. The allowed maximum time step T_c decreases when the robot travels at higher speed. For a robot to collect its neighbors location, it may use communication for that. Using traditional distributed optimization algorithms, agents are required to communicate state trajectories during the allowed computation time T_c, until they converge. Since such algorithms are iterative, agents communicate a trajectory, then they perform some computations, then they communicate again until they converge. So, communicated trajectories cannot just be communicated once, but iteratively, as there are computations between each communicated trajectories. Figure 5.1 shows an example of such scenario.

We are interested in designing an algorithm that can be deployed in a distributed manner such that the information exchange (e.g. by communication) is reduced while producing reasonable decisions. In general, approximated solution is not avoidable

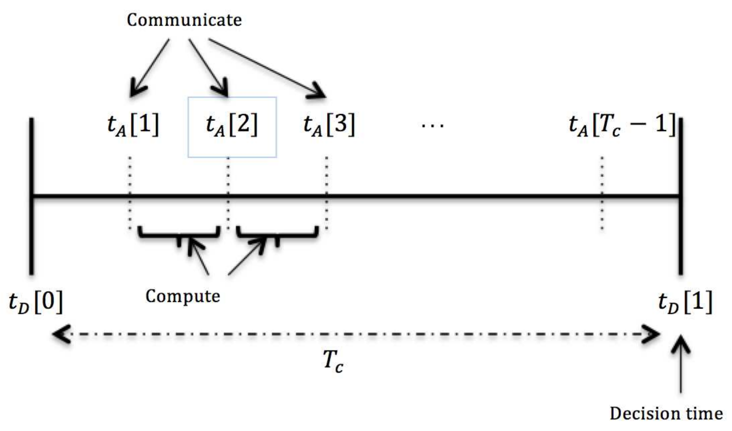


Figure 5.2: There are two time scales, an outer time scale and an inner time scale. Each agent has to take an action during each time step in the outer time scale, $[t_D[0], t_D[1]]$ where $t_D[1] - t_D[0] \leqslant T_c$. The step inside the outer time scale is represented by the two vertical solid lines at $t_D[0]$ and $t_D[1]$. Agents communications happen on an inner (faster) time scale (inside the outer time scale step T_c) represented by the vertical dotted lines. At each inner time tick t_A communication happens. Between two consecutive inner time ticks computation happens to eventually converge to a solution. The decision has eventually to be taken at $t_D[1]$

as a result of global information reduction. However, a suboptimal, but meaningful, decision can be good enough as a trade off. Furthermore, computationally light algorithms are desired for real-time execution and to be run on a robot's micro-controller in which usually has limited computation resources.

In this chapter, we show a proposed algorithm that address desired requirements, namely, real-time execution with reduced information exchange while providing reasonable decisions. In the defenders-attackers setting, we exploit the linear program framework discussed in previous chapter as it can be solved efficiently faster compared to other convex optimization problems, and is validated to run in a real-time setting on embedded micro-controllers.

5.2 Distributed Approach

In this section the proposed distributed approach is introduced. Recall that the objective is not only to introduce one more alternative distributed approach, but to actually address certain practical requirements. The following requirements need to be met.

- The approach can be easily implemented in a distributed way. In other words, the degree of autonomy is increased as much as possible, and centralized supervision is only needed for emergency or recovery purposes. Otherwise, the agents (e.g. robots) should be able to act stand alone all the time and generate meaningful local decisions.
- The distributed approach can generate useful decisions with out exhaustive exchanged information (i.e. exhaustive communication burden)
- The distributed approach should be implementable in real-time on computing modules with limited computation resources. In other words, most small scale robots (specially UAVs) can carry computing modules that have limited power and may not be able to execute complex and computation demanding algorithms in real-time (e.g. milliseconds).

With these requirements in mind, the following distributed approach is proposed.

5.2.1 Modeling

We consider the game of attackers-defenders described in Chapter 3. We recall the problem elements as follows.

- Optimize certain performance measure $J(\cdot)$ over a finite time horizon T. In particular, attackers infiltration into the base zone is to be minimized.
- The optimization is subject to state and control constraints. For example, the state has to respect certain dynamics $\mathbf{x}(t+1) = f(\mathbf{x}(t), \mathbf{u}(t))$ which may describe the reachable cells (grid sectors) in one time step, and $\mathbf{x} \in \mathcal{X}$. Similarly, $\mathbf{u} \in \mathcal{U}$,

which may describe certain possible directions of motion. We assume that $\mathcal{X}$ and $\mathcal{U}$ are discrete sets.

The problem can be generally written as

$$\begin{aligned} \text{min} \quad & \sum_t J(\mathbf{x}(t)) \\ \text{subject to} \quad & \mathbf{x}(t+1) = f(\mathbf{x}(t), \mathbf{u}(t)) \\ & \mathbf{x}(t) \in \mathcal{X}, \quad \mathbf{u}(t) \in \mathcal{U} \end{aligned} \tag{5.1}$$

We consider a cooperative multi-agent task where agents optimize a common performance cooperatively. In other words, each agent knows the objective function structure, the assumed system dynamics, and the system state and input constraints. However, only local information are available to each agent a time t. Specifically, each agent has partial information of the total system state $\mathbf{x}(t)$ and can only execute its own part of the total system control input $\mathbf{u}(t)$.

In the setting of the attackers-defenders game, if a centralized controller were to be used, and the total state $\mathbf{x}(t)$ is observed (positions of all attackers and defenders), an optimal solution for the defenders $\mathbf{u}^d(t)$ can be computed. However, in a distributed setup, we assume that each agent can generally observe part of the total state $\mathbf{x}$, and is responsible for generating its control only, which is part of the total system control input $\mathbf{u}^d(t)$. Therefore, a suboptimal solution can be generated. As a step to possible improvement of the solution, we propose that each agent uses the assumed dynamics of the whole system, which we assume to be known to all agents, to plan its own state, its neighbors state, and possibly the state of unobserved neighbors. More specifically, each agent executes the following algorithm.

1. Each agent i observes its current position as well as its neighbors positions. We

assume that neighbors positions are perfectly observed.

2. Agent i constructs a local copy of the system state $\mathbf{x}^d(t)$ denoted by $\mathbf{x}_i^d(t)$ based on its local observations

3. Agent i produces a local optimal solution $(\mathbf{u}_i^d)^*$ by solving

$$\begin{aligned} \text{min} \quad & \sum_t J(\mathbf{x}_i^d(t)) \\ \text{subject to} \quad & \mathbf{x}_i^d(t+1) = f(\mathbf{x}_i^d(t), \mathbf{u}_i^d(t)) \\ & \mathbf{x}_i^d(t) \in \mathcal{X}, \quad \mathbf{u}_i^d(t) \in \mathcal{U} \end{aligned} \tag{5.2}$$

   The local solution $(\mathbf{u}_i^d)^*$ contains a local solution for agent i and its neighbors, from the perspective of agent i

4. Agent i extracts its solution from $(\mathbf{u}_i^d)^*$ denoted by $[(\mathbf{u}_i^d)^*]_i$ and implements it. It does not share its local solution of other agents.

5. Repeat

Using the above algorithm, each agents acts like a local centralized planner that computes solutions for itself and its neighbors (or agents that it is connected to at time t) in its local problem, see Figure 5.2.1. However, the computed solutions of the neighbors are not communicated to them (as the usual case in distributed algorithms) which reduces communication requirements. The quality of the solution produced using this approach and compared to the centralized solution, which is considered as the ground truth, depends on the connectivity of the network, or the graph. In general, as the connectivity between each agent to every other agent increases, the solution quality increases. If the network is fully connected, i.e. each agent is connected to all other agents, the centralized solution is obtained. However, this comes at the expense of congested network due to the increased communication which we try to avoid in

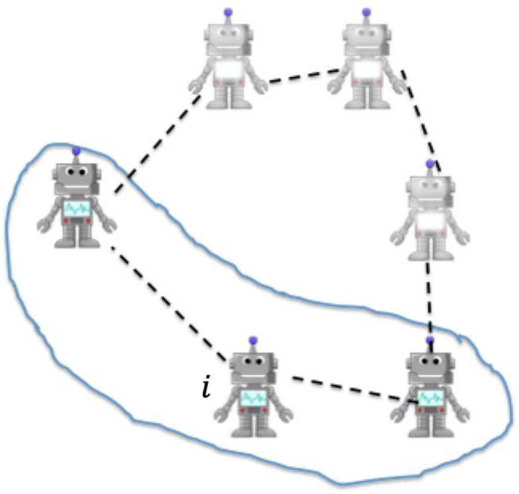


Figure 5.3: Example of a local perspective of a robot in a distributed planning instance. Agent i plans for itself and its neighbors only and does not count for other non-neighbor agents.

this work. So, apparently there is an unavoidable trade off between optimality and connectivity in distributed optimization algorithms. A meaningful suboptimal solution can be good enough and it will be shown through extensive numerical simulation that the proposed approach can provide near-optimal solution under the investigated simulation scenarios. An alternative distributed approach that can provide approximate centralized solution with analytical bounds on the optimal solution, and with generally reduced communication is discussed later.

5.2.2 Local LP Problem

In order to make online real-time local decisions, we exploit the linear program formulation in 4.21. We now write algorithm in 5.2 in the form of a local linear program as follows.

Define $\mathbf{X}_i^d = \big((\mathbf{x}_i^d(1))^T \quad (\mathbf{x}_i^d(T))^T\big)$ to be agent i estimate of defenders trajectory over a prediction horizon T. Define $\mathbf{U}_i^d = \big((\mathbf{u}_i^d(0) \quad (\mathbf{u}_i^d(T-1))^T)^T\big)$ to be agent's i estimate of the defenders trajectory over T. Define $\mathbf{x}_i^d(0)$ to be the initial state vector (with 1's at the initial sector locations, 0's otherwise) considering agent i and its neighbors only. $\mathbf{X}^a$, $\mathbf{X}^{\text{ref}}$, α, β and the optimization matrices are defined the same as in the centralized version in 4.21. If only local attackers are perfectly observed, then we define a local estimate of attackers trajectory as $\mathbf{X}_i^a$ Now, the local linear program problem is presented.

$$\begin{aligned}
\min \quad & \left[\alpha^d \mathbf{X}_i^a + \beta^d \mathbf{X}_{\text{ref}}\right]^T \mathbf{X}_i^d \\
\text{s.t.} \quad & \mathbf{X}_i^d - \mathbf{T}_{xu}\mathbf{U}_i^d = \mathbf{T}_{xx_0}\mathbf{x}_i^d(0) \\
& \mathbf{T}_{xu,c}\mathbf{U}_i^d \leqslant \mathbf{T}_{xx_0,c}\mathbf{x}_i^d(0) \\
& \mathbf{X}_{\text{obs}}^T \mathbf{X}_i^d = 0 \\
& \mathbf{0} \leqslant \mathbf{X}_i^d \leqslant \mathbf{1} \\
& \mathbf{0} \leqslant \mathbf{U}_i^d \leqslant \mathbf{1}
\end{aligned} \tag{5.3}$$

5.2.2.1 Collision Avoidance Guarantees

In the centralized LP problem in 4.21, collision of agents of the same team could be avoided by the constraints $\mathbf{0} \leqslant \mathbf{X}^d \leqslant \mathbf{1}$. This means that a maximum of one agent can occupy any sector at any time t. However, in the local version in 5.3, each agent has only local perspective of neighbor agents or agents connected to, hence, $\mathbf{x}^d(0)$ is not fully known to each agent. Therefore, collision avoidance is not guaranteed anymore. To overcome this problem, an extra condition is imposed on the local problem. We require the assumption that each agent can see (or share position information with) agents that are at least 2 hops away. More specifically, agent i at sector s can observe agents positions within reachable sectors that are two-step away in time. The set of reachable sectors of agent$i$ in sector s in k time step is denoted by $\mathcal{N}_k(s_i)$. So, each agent i in sector s can observe each agent j in $\mathcal{N}_2(s_i)$. To make sure that no collision happens in a 1-step ahead move, we can impose the condition $\mathcal{N}_1(s_i) \cap \mathcal{N}_1(s_j) = \phi$.

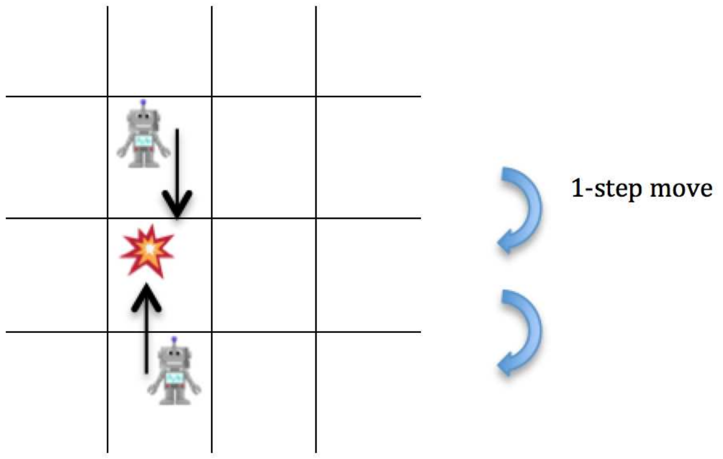


Figure 5.4: A collision example if robots that are 2-step away do not account for each other's position.

We also assume that agents follow the same linear dynamics in 4.9 while transferring between sectors.

A possible collision in case an agent does not account for agents that are 2-step away is demonstrated in Figure 5.2.2.1.

The previous approach is assessed through several simulations and showed reasonable performance in real-time scenarios. It was also implemented in a hardware setup and was able to provide responsive reasonable defender actions in real-time under the investigated setups. The details of the simulation and hardware setup is discussed in Chapter 6.

The previous local LP approach provides a promising reasonable heuristic solution for a real-time application of the attackers-defenders setup. However, more analysis on the performance guarantees are currently missing which is a potential future direction of this work. However, in the following, we introduce an alternative distributed approach that can provide a guaranteed approximate solution.

5.3 One-step Lookahead Distributed Approach

In this approach the optimal centralized cost $J^*(\cdot)$ is approximated by an approximate cost $\tilde{J}(\cdot)$, that is easier to compute. This can be seen in dynamics program problems as the usually the optimal *cost-to-go* function J^* is difficult to compute, for example due to the large size of the state and input spaces. It can be shown that that optimal cost obtained by using the approximation function $\tilde{J}$ is bounded by some value which is a function of the approximation accuracy [84].

We define the problem of interest as follows. Recall the infinite horizon deterministic cost with discount factor $0 < \gamma < 1$.

$$J = \sum_{t=0}^{\infty} \gamma^t g(x(t), u(t)) \tag{5.4}$$

Also, recall the bellman equation which can be used to solve for the optimal cost J^* for an infinite horizon, deterministic problem with discounted factor $\gamma < 1$.

$$J^* = \min\{g(x(t), u(t)) + \gamma J^*(f(x(t), u(t)))\} \tag{5.5}$$

As mentioned, J^* can be hard to compute, so an approximation $\tilde{J}$ is used such that,

$$||J^* - \tilde{J}|| = \epsilon \tag{5.6}$$

5.6 represents the approximation error.

Assuming that μ is a greedy policy that corresponds to $\tilde{J}$. Then, the corresponding optimal cost J^μ satisfies,

$$||J^\mu - J^*|| \leqslant \frac{2\gamma\epsilon}{1-\gamma} \tag{5.7}$$

Proof. See Sec. 6.1.1 in [84] □

Now let's rewrite the cost in 5.4 as two terms, the current stage cost at time $t = 0$ and the remaining cost over the remaining horizon.

$$J = g(x(0), u(0)) + \sum_{t=1}^{\infty} \gamma^t g(x(t), u(t)) \tag{5.8}$$

We can write the optimal cost-to-go J^* as

$$J^* = \min_{u(0)} \{ \; g(x(0), u(0)) + \gamma J_1^* \; \} \tag{5.9}$$

J_1^* is the optimal cost-to-go from time $t = 1$.

In fact, by comparing 5.5 and 5.9, we conclude that $J_1^* = J^*$.

In 5.9, once $J^* = J_1^*$ is known, one needs only to optimize over $u(0)$ only. Usually, J^* is hard to compute for large systems, and an approximation $\tilde{J}$ is adopted as in 5.6 and the bound on the approximate optimal cost is equivalent to the one in 5.7.

Up to now, we have obtained an approximate optimal solution that is as good as the approximation accuracy *for the centralized problem.* The advantage is that $\tilde{J}$ is supposed to be easier to compute. Now, one can solve the approximate problem in distributed way using a distributed optimization algorithm, for example using dual decomposition as in [29]. That way, agents will need to communicate less information instead of entire long trajectories as it is usually the case in such algorithms. However, agents still need to communicate *iteratively* in order to converge to the optimal solution and the convergence is asymptotic.

In next chapter, extensive simulations and hardware testing are discussed to test the proposed local LP algorithm under different settings.

Chapter 6

Simulation and Hardware Implementations

6.1 Introduction

This chapter presents another main contribution of this work. It provides several simulations and hardware testing results for the proposed local LP algorithm in 5.3 under realistic real-time simulations and hardware settings. The simulations are done in the context of the attackers-defenders game. The work discussed in this chapter is accepted in the ICRA2018 conference [85] at the time of writing this thesis.

The software that is used in order to perform the simulation and hardware testing in this work is available as an open-source package (with ROS integration) at [86].

We discuss the following simulations and hardware settings.

- First, we provide MATLAB simulations which heuristically compare between the quality of the local LP solutions and the centralized ones under large number of random attackers-defenders game configurations. This simulation does not include a real-time scenario with actual robot's dynamics simulation. However, it is meant to assess how the local algorithm behaves under individual game configuration without propagating the solution in time.

- Next, we evaluate the local algorithm in a realistic simulation environment with realistically simulated drones. The algorithm system-wide performance is evaluated in a complete game time under different settings. One setting is to allow a human operator, which has full view of the game, to control an attacker. Another setting is considered where multiple attackers have automated attacking strategy using the centralized LP in 4.21.

- Finally, the local LP algorithm is implemented and tested on real hardware using quadrotors equipped with low-power embedded computer which executes the algorithm in real-time, in both indoors and outdoors settings.

6.2 Simulations

6.2.1 Local vs. Centralized Solution Quality

The solution quality of the local LP algorithm is numerically compared to the centralized solution using several random configuration of a certain game setup. The game setup is as follows

- **Agents**: 4 defenders and 4 attackers are considered.
- **Grid**: 11 rows and 13 columns grid with 143 sectors. $\mathcal{S} = \{s_1, \cdots, s_{143}\}$. Base (defense) zone is defined as $\mathcal{S}_{\text{base}} = \{s_{72}\}$. Reference sectors which surrounds $\mathcal{S}_{\text{base}}$ is defined as $\mathcal{S}_{\text{ref}} = \{s_{58}, s_{59}, s_{60}, s_{73}, s_{86}, s_{85}, s_{84}, s_{71}\}$
- **Rules**. Each agent can move at most one sector away in any direction per time step. In this simulation, defenders can see all attackers at all time. An attacker is captured once it is one sector away from a defender.
- **Receding Horizon Length**: The prediction horizon $T_p = 3$ time steps.
- **Objective Parameters**: The attacking weight is $\alpha = -0.99$, and the defensive weight is $\beta = -0.01$. This induces an attacking strategy.

The algorithms implementation and simulation is done in MATLAB on using a single machine. A 1000 configurations were randomly sampled, i.e. different initial locations of attackers and defenders. For each configuration sample, the centralized and local LP algorithms are executed, and the defenders next optimal locations are

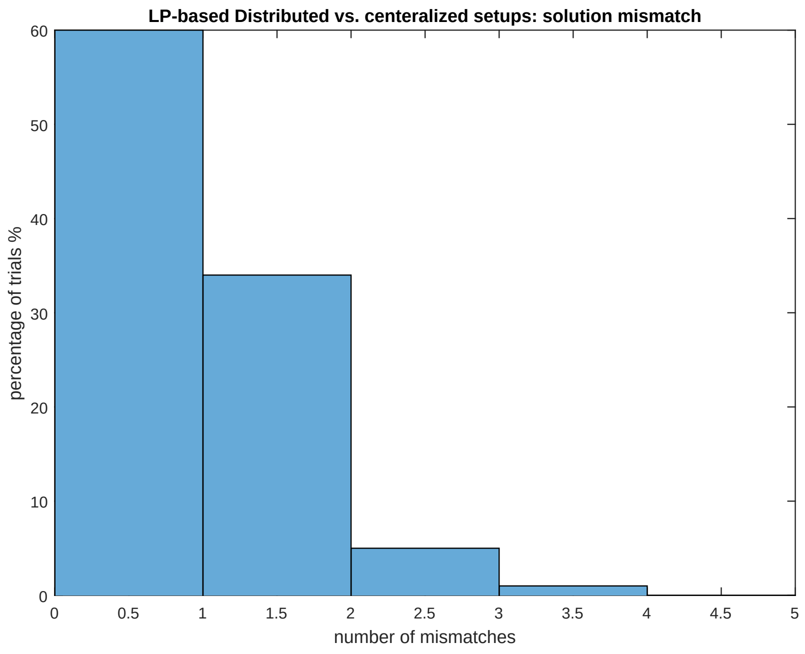
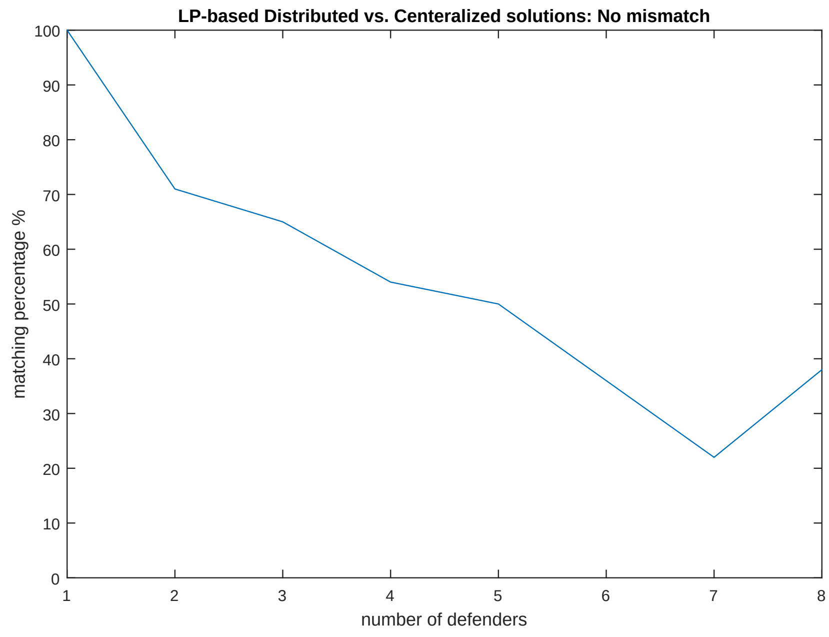


Figure 6.1: LP-based Distributed vs. centralized setups: solution mismatch

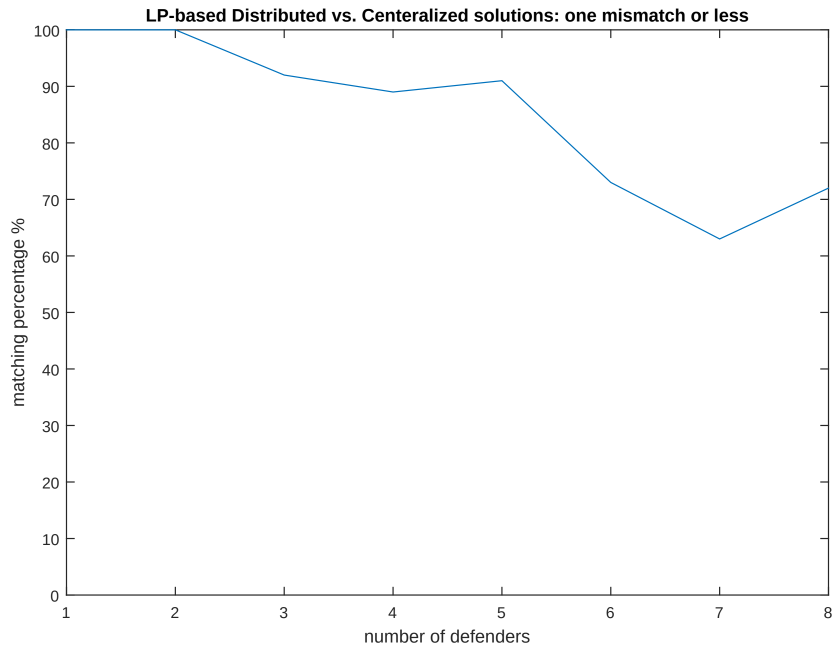
extracted. The number of matching solutions are then compared. A complete match is when all computed local solutions match the centralized one. For example, let the centralized solution (the next optimal location of a defender i) be d_i, defender i local solution be $\hat{d}_i$. If $d_i = \hat{d}_i$ for all i, then it's a complete match. Another comparison is also made when most of the solutions match. For example, a one-mismatch means that $d_i = \hat{d}_i$ for all i except for only one agent j. Similarly, a n-mismatch can be defined.

A histogram in Figure 6.1 shows the local solution quality compared the centralized one in terms of solution mismatches. It shows that 60% of the random scenarios provided the exact centralized solution, and 94% provided up to only one mismatch in the optimal defenders positions. In other words, the local LP algorithm provides near-optimal solutions under the considered random configurations.

Another observation is that the discrepancy between the local and centralized LP algorithms is proportional to the number of agents in the same grid. Figure 6.2 shows the percentage of trails that provide no mismatch, and one mismatch as a function of the number of defenders. In general, as the number of agents increases and their connectivity structure remains the same, the discrepancy is expected to increases.



(a)



(b)

Figure 6.2: LP-based Distributed vs. centralized setups. (a) No solution mismatch as a function of number of defenders. (b) Maximum of one solution mismatch as a function of number of defenders

6.2.2 Real-Time Simulation using Simulated Drones

In this section we discuss a more realistic simulation settings where the local LP algorithm is used in a complete attackers-defenders game with realistic simulated drones. Drones were chosen as a robot example as it can travel in higher speeds compared to, for example, ground robots. We also had access to actual drones in the RISC lab which is also used for actual hardware implementation which is discussed later. While moving in high speeds, defenders need to make fast decision which imposes a challenge on real-time constraints. Before we conducted a complete hardware implementation, we did a realistic simulation that is as close as possible to hardware implementation where different scenarios can be safely and cheaply examined in short time.

6.2.2.1 Simulation Setup

We use quadrotors as our testing platforms as they provide high maneuverability even in constrained test spaces. Each quadrotor is controlled by two levels of controllers. A low-level controller for attitude, velocity, and position stabilization and tracking, is handled by the open-source PX4 autopilot firmware[1]. A high-level controller executes the proposed algorithm and provides position setpoints to the low-level controller.

Our simulation setup includes four components as follows.

- **Local LP Algorithm**: An implementation of the proposed algorithm as a C++ class. The algorithm's class provides convenience functions to easily setup problem's parameters and inputs, executes the proposed LP algorithm, and extracts the problem solution. The algorithm uses the open-source linear program C++ library, `GLPK` [2] for fast linear program optimization. Also, a ROS C++ node (program) is created to interface between the algorithm class and ROS environment.

[1] `px4.io`

[2] `https://www.gnu.org/software/glpk/`

- **PX4**: Open-source autopilot software, which provides low-level control to the quadrotor's states including orientation, velocity, and position. The same software is used on actual autopilot hardware e.g. Pixhawk. However, since the autopilot is used in simulation, it expects to receive simulated sensors data from a simulator, Gazebo in this work, instead of actual hardware sensors. Also, commanded actuators signals are sent to Gazebo simulator to control the UAV's motion in the simulation. The PX4 firmware used in the simulation is called software-in-the-loop, SITL. The reader is referred to the PX4 developer guide[3] on how to setup the SITL simulation for multiple UAVs.

- **Gazebo**: A robotic simulator which provides realistic simulation of a robot's 3D motion dynamics in space. It also provides the autopilot software with simulated sensor data, e.g. IMU, barometer, and GPS measurements. It also receives the actuators' (motors') commanded signal from the autopilot in order to control the UAV motion. Moreover, it can simulate a more realistic environment by including a wind and magnetic field profiles.

- **ROS**: Robot Operating System, which is a middleware that provides a convenient interface between all simulation components and reduces (or can eliminate) required changes to transfer the implementation to actual hardware setup.

The proposed simulation setup is depicted in Fig. 6.3

The simulation setup was tested on an i7 computer which runs Ubuntu 16.04, ROS Kinetic, and Gazebo 7.

The game setup in the following simulation is defined as follows.

- **Agents**. The maximum number of agents used in the following simulations is 3 in each team. There is no particular reason on the selected number of agents and they can be selected arbitrary.

[3]`https://dev.px4.io/en/simulation/multi-vehicle-simulation.html`

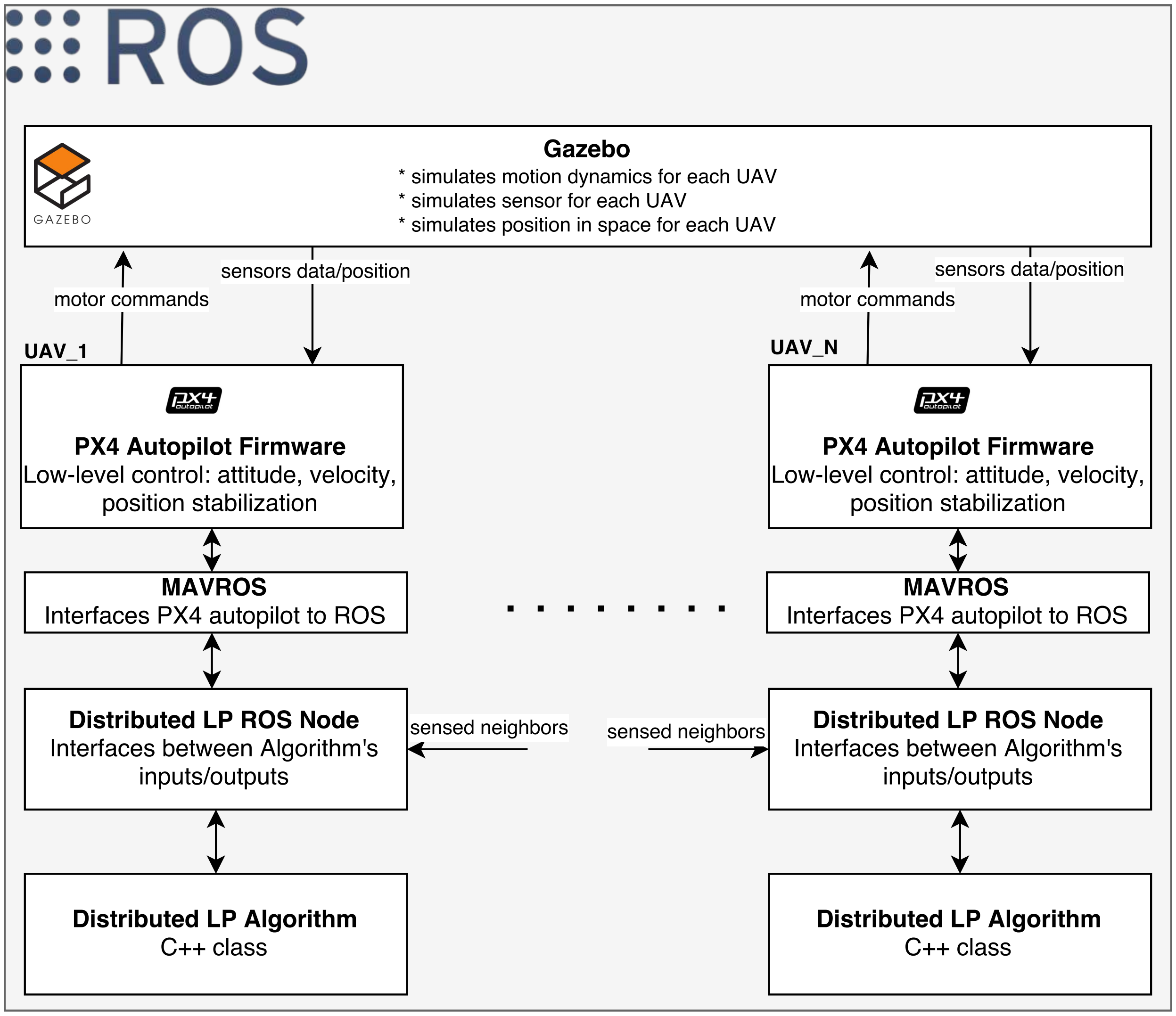

ROS
GAZEBO
Gazebo
* simulates motion dynamics for each UAV
* simulates sensor for each UAV
* simulates position in space for each UAV
motor commands
sensors data/position
UAV_1
PX4 Autopilot Firmware
Low-level control: attitude, velocity, position stabilization
MAVROS
Interfaces PX4 autopilot to ROS
Distributed LP ROS Node
Interfaces between Algorithm's inputs/outputs
sensed neighbors
Distributed LP Algorithm
C++ class
motor commands
sensors data/position
UAV_N
PX4 Autopilot Firmware
Low-level control: attitude, velocity, position stabilization
MAVROS
Interfaces PX4 autopilot to ROS
Distributed LP ROS Node
Interfaces between Algorithm's inputs/outputs
sensed neighbors
Distributed LP Algorithm
C++ class

Figure 6.3: Architecture of simulation setup

- **Grid**: We consider a $2D$ battle field represented by a 7×7 square grid. Each sector is $2 \times 2\ m^2$. The base zone is located at the first sector i.e $\mathcal{S}_{textbase} = \{s_1\}$, and the reference sectors surrounding the base sector are $\mathcal{S}_{\text{ref}} = \{s_2, s_8, s_9\}$

- **Capture Distance**: An enemy is captured if it is within a circle of radius 2.1 meters from a defender.

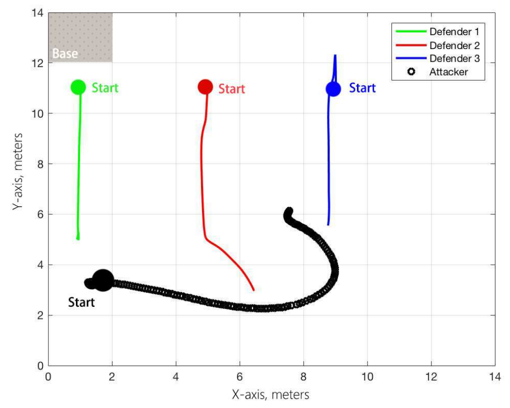
- **Objective Weights**: The attacking and defensive weights are $\alpha = -0.99$, $\beta = -0.01$, respectively. The weights are the same for all simulations scenarios unless mentioned otherwise.

In the following, we present simulation results of some test cases using the proposed simulation setup.

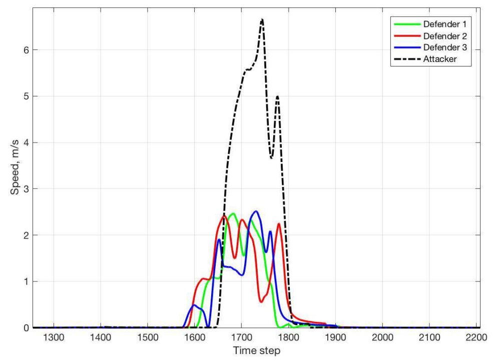
6.2.2.2 Simulation 1: Human controlled attacker vs. 3 defenders

In this scenario the problem setup includes 3 quadrotors in the defenders' team and one quadrotor in the attacker's team. To make the battle more interesting, the attacking quadrotor is controlled by a human operator through a joystick, which controls the quadrotor's lateral velocities. An interesting note is that, the human controlled quadrotor is allowed to exhibit any behavior although the defenders assume a specific predictive attacking model that can exhibit very different behaviors than the human one. This helps to evaluate the distributed attacking/defending strategies against non-modeled attacking behavior.

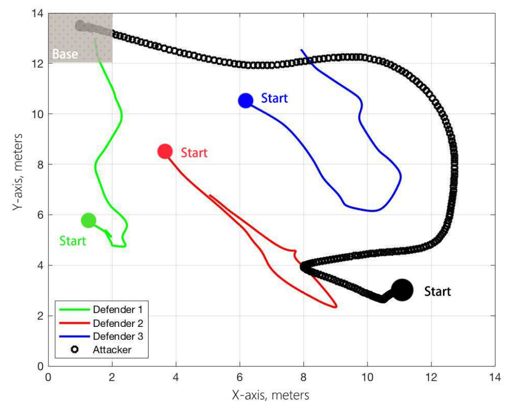
Fig. 6.4(a) shows the UAV trajectories during the battle. As we can see, defenders were able to distribute themselves in order to block the attacker from entering the base zone, before it is eventually captured at position $(7.8, 6.1)$ although the attacker had higher speeds than defenders.



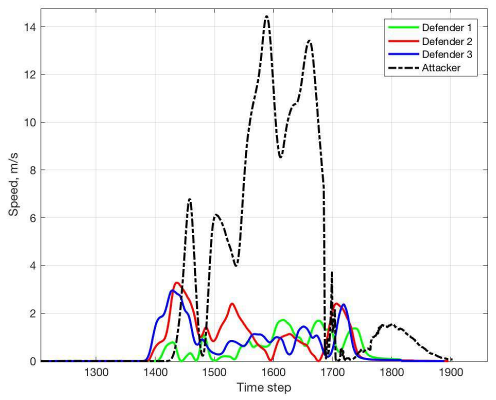
(a) Position trajectories of simulation trial 1



(b) Velocity trajectories of simulation trial 1



(c) Position trajectories of simulation trial 2



(d) Velocity trajectories of simulation trial 2

Figure 6.4: Fig (a) shows trajectories of all quadrotors in simulated scenario where attacker was captured at position $(7.8, 6.1)$. Fig. (b) shows the corresponding velocity trajectories during the trial. fig. (c) shows the position trajectories of trial 2 where the attacker was able to enter the base, as its maximum speed factor was much higher than defenders. Fig. (d) shows the corresponding speed profiles of trial 2

In Fig. 6.4(c), we can see that the human operator was able to drive the attacking UAV to the defense zone. Thus occurrence is mainly because the attacker's speed factor, see Fig. 6.4(d), was significantly increased while the defenders speed factor remained the same as in trial 1.

6.2.2.3 Simulation 2: Multi-attackers vs. multi-defenders with no obstacles

In this scenario we show a simulation of a game scenario where multiple attackers are available. The attackers in this scenario also try to optimize their motion planing using the centralized version of the LP algorithm in 4.21 where attackers have the advantage over defenders of knowing global information about where all players are at all times. Using this approach, attackers try two weighted behaviors,

- Evasion: where they try avoid collision with defenders. This is captured by $(\mathbf{X}^d)^T\mathbf{X}^a$. In this case attackers assume a model of defenders in order to compute $\mathbf{X}^d$ over a prediction time horizon. The assumed model of defenders is similar to the one mentioned in [10].
- Attacking: where they try to attack the defense zone captured by the term $(\mathbf{X}^{\text{ref}})^T\mathbf{X}^a$.

These two behaviors can be weighted by the objective parameters α and β. In this particular simulation $\alpha = 0.8,\ \beta = -0.2$ which results in more evasion than attacking. The objective function looks like

$$\left[\alpha\mathbf{X}^d + \beta\mathbf{X}^{\text{ref}}\right]^T \mathbf{X}^a$$

In this simulation, the grid size is increased in order to allow for more mobility for the drones. The grid size is 10×10 where each sector is $5 \times 5\ m^2$ resulting in grid

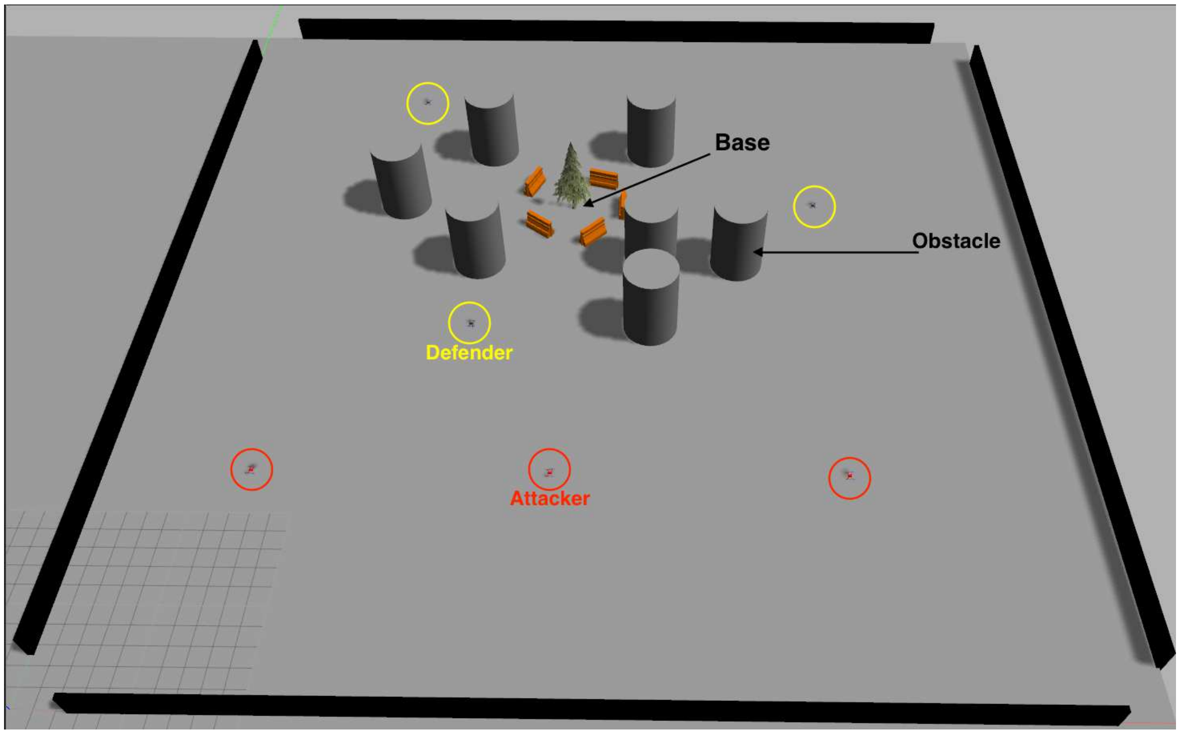


Figure 6.5: Simulated world in Gazebo simulator. This snapshot shows the initial position for 3 defenders, the black drones on the top side of the field marked by yellow circles, and 3 attackers, the red drones on the lower part of the field. The base is marked by the tree.

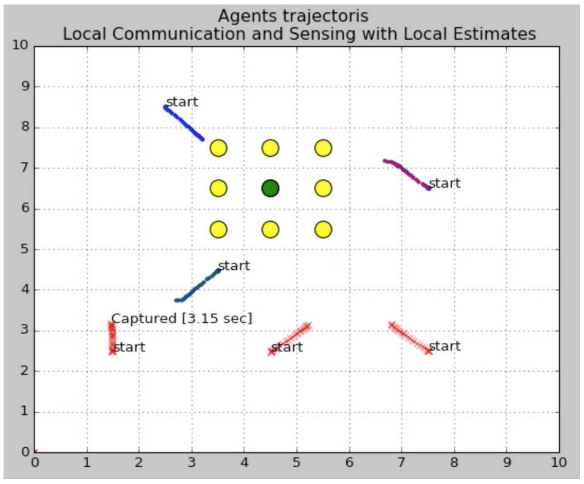
area of 50 × 50 m^2. The set of base locations contains only one sector, $mathcal S_{\text{base}} = \{35\}$ and the reference sectors set that surrounds the base location is $\mathcal{S}_{\text{ref}} = \{24, 25, 26, 34, 36, 44, 45, 46\}$. The Gazebo simulated battle field is shown in Figure 6.2.2.3.

In this simulation, agents start at predefined initial locations, shown in Figure 6.2.2.3 and hover in position until they receive the *battle* command. After the battle command is signaled, all agents start to battle. Each defender executes the local LP algorithm considering its neighboring defenders and local attackers that it can see. In particular, defenders and attackers that are in 2-sector away are considered. Each attacker, as mentioned, has access to all players (both defenders and attackers) positions at all time. Once an attacker is captured, it lands and is no more considered in the rest of the game. The game continuous as if this player does not exist.

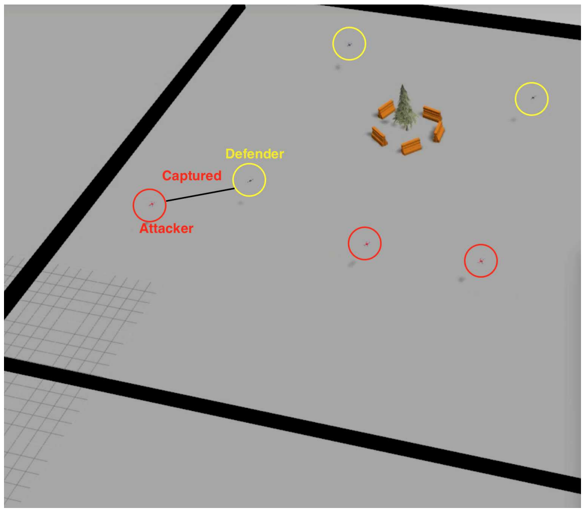
The figures in 6.2.2.3 show a sequence of snapshots of a game simulation trial with no static obstacles. The left-most attacker was captured by the defender in the lower part of the field as it was in its sensing region in the beginning of the game. After the defender captured the attacker, the attacker is disabled, and the defender wen back to protect the base. The other two defenders do not see attackers in their sensing region, so they stay around the base reference sectors. as the game evolves, the remaining two attackers try to go to the base while avoiding defenders. Once they became in sensing region of one of the defenders, they got captured and the game ends.

6.2.2.4 Simulation 3: Multi-attackers vs. multi-defenders with obstacles

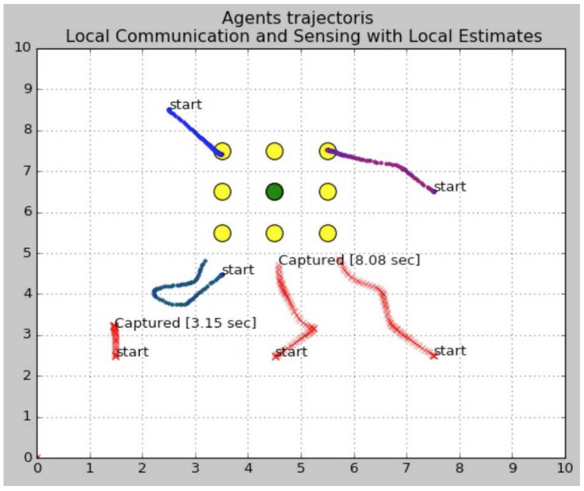
In this simulation case, static obstacles are added, see Figure 6.2.2.3, to the same game setup in the previous simulation case. Static obstacles can be encoded as single linear constraint $\mathbf{X}_{\text{obs}}^T \mathbf{X}^d = 0$. Figures in 6.2.2.4 show a game simulation trial where agents started at the same initial positions and were able to plane their motions during the battle while avoiding the static obstacles.



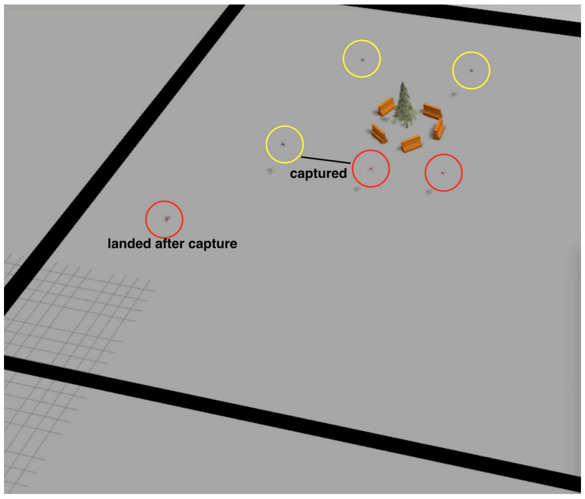
(a)



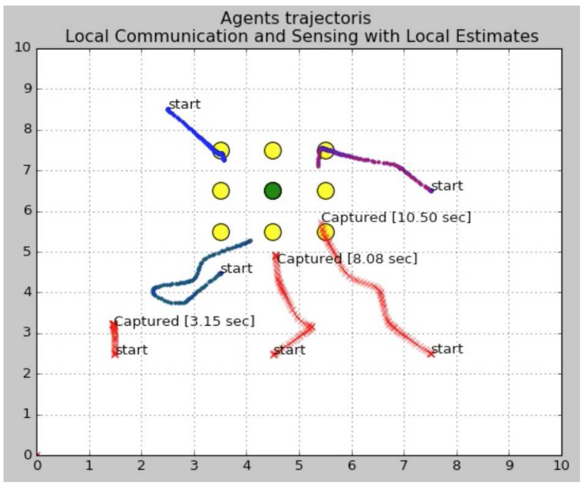
(b)



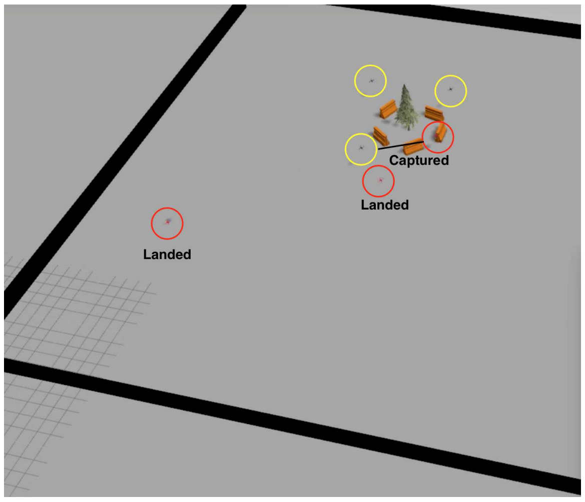
(c)



(d)

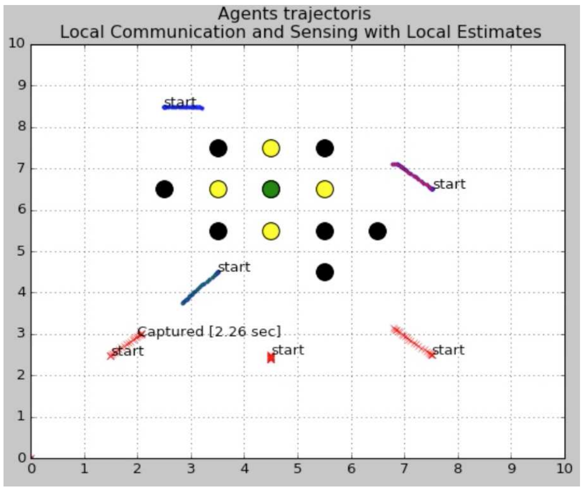


(e)

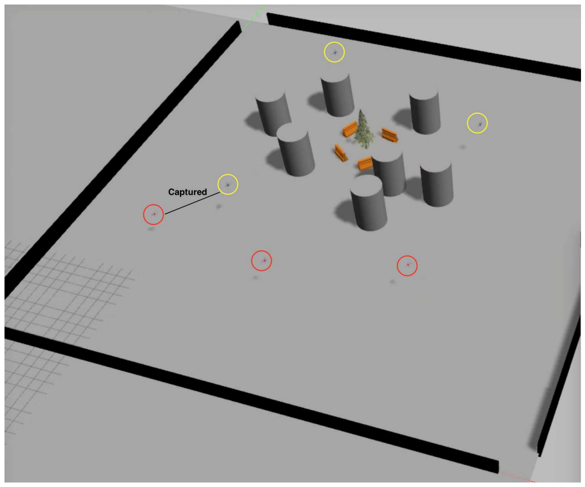


(f)

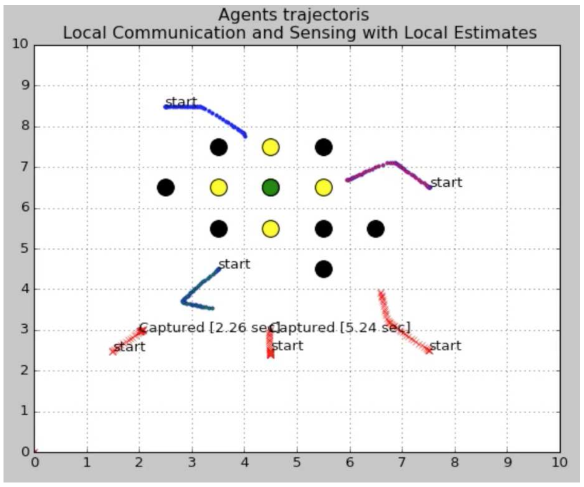
Figure 6.6: Sequence of snapshots in a game trial. (a) and (b) show the first capture. (c) and (d) show the second capture. (e) and (f) show the final capture. 2D figures show attackers trajectories in red, the remaining are for defenders. The green circle marks the base location surrounded by yellow ones which mark reference sectors.



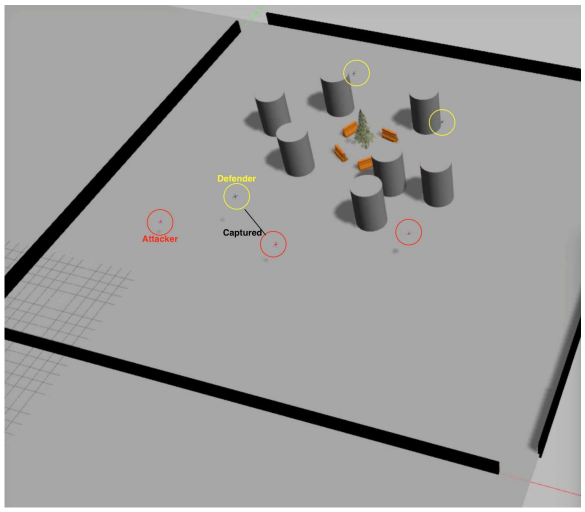
(a)



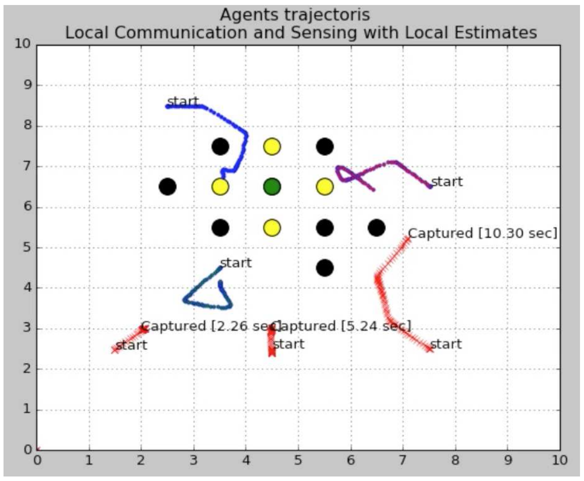
(b)



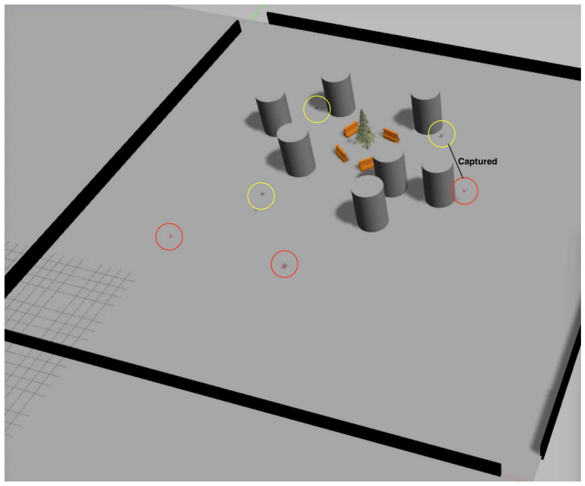
(c)



(d)



(e)



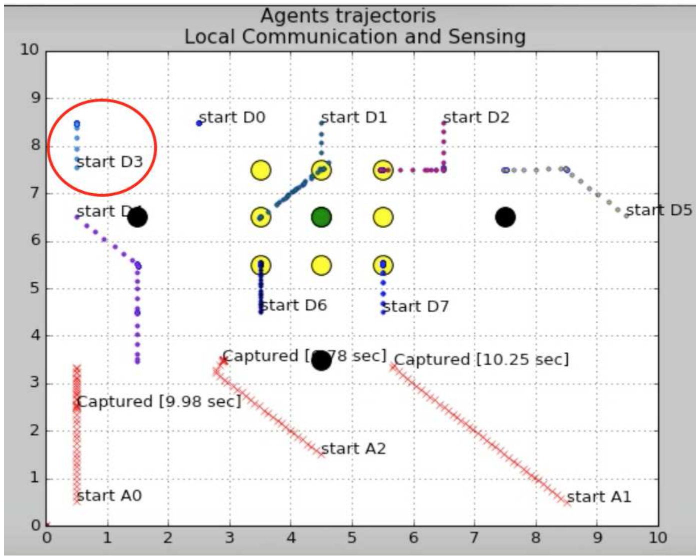
(f)

Figure 6.7: Sequence of snapshots in another game trial with static obstacle included.

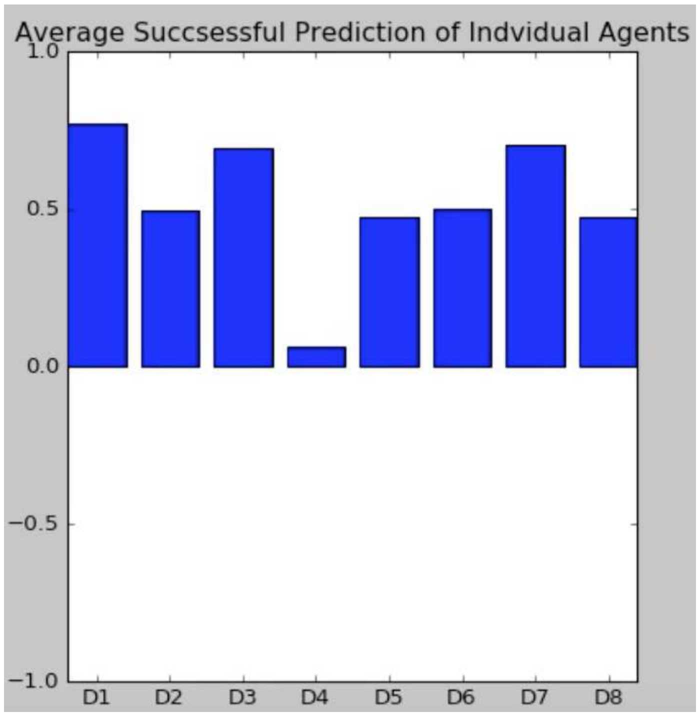
6.2.2.5 Defenders Prediction Quality

As mentioned, each defender make an optimistic prediction of its neighbors. In the following simulation, an experiment between 8 defenders and 3 attackers was performed. The three defenders are executing the distributed LP algorithm while the attackers are executing the centralized LP algorithm (with global information).

Figure 6.8 shows the trajectories of the players as well as the average prediction quality of each defender over the whole game time. As we can see, most of the defenders were able to make more than 50% successful prediction about their neighbors motion over the game time. Defender 4 (marked as D3 in the trajectory figure as the counter starts from 0) has less successful prediction rate compared to the rest. This is mainly that it was less connected to the rest of the defenders during the game. This observation trivially indicates that the distributed LP algorithm performance is a function of the network topology. It tends to give better prediction results when the agents network is more connected.



(a)



(b)

Figure 6.8: Prediction quality of defenders in 8-defender vs.3-attacker game. (a) shows the complete trajectories of all players during the game run. (b) shows the average prediction quality of each defender over the whole game time.

6.2.3 Comparison with Related Works

In this section we compare the main aspects of our distributed framework with other recent frameworks that we believe of relevance to this work. In particular, we make a comparison with works introduced in [87, 88].

In [88], the authors consider a pursuit-evasion setup that consists of a single evader and multiple pursuers in 2D plane. Capture guarantees are provided for a single pursuers and, therefore, can be generalized for multiple evaders. The strategy is based on a voronoi tessellation of 2D bounded environment where each player occupy a voronoi cell. The pursuer's policy that guarantees capturing the evader in finite time, $t_c < \infty$, is to head to the midpoint of the shared voronoi edge with the evader. Pursuers that do not share an edge with the evader (non-neighbors) just go straight ahead towards the evader position. This framework assumes global information of all players positions at all times.

In [87], the authors extended the work in [88] to multi-pursuer vs. multi-evader setup and beyond 2D space. Each evader is assigned to its nearest pursuers which share voronoi edges (or faces in higher dimensions). Then, the same strategy discussed in [88] of multi-pursers vs single evader is used to guarantee capture in finite time, $t_c < \infty$. a pursuer is not assigned to an evader, it moves straight to the nearest evader.

In both works, the capture guarantees are proved and the control strategy is a simple equation that can be implemented in real-time. Also, there is no assumption of the evader's motion model except that it has the same maximum pursuers' speed. However, there are no explicit collision avoidance mechanism and no coordination between pursuers that can greatly improve the capture time.

An example representation of the game in 2D is depicted in Figure 6.9. In the following section, we present simulations that highlights the main differences between these works and the proposed linear program framework.

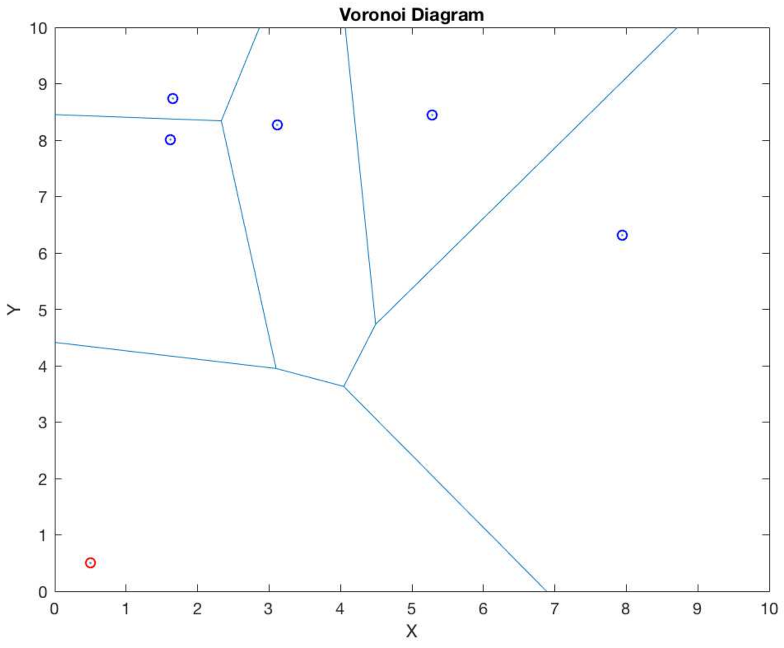


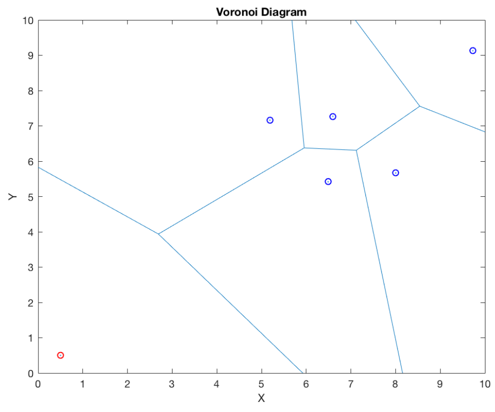
Figure 6.9: A 2D Voronoi tessellation of a single evader vs. multi-purser game in a 10×10 plane. The evader is represented by a red circle in the lower left corner. The remaining blue circles represent the locations of the pursuers. The evader's neighbors are defined by the pursuers who share a Voronoi edge.

6.2.3.1 Simulations and Discussion

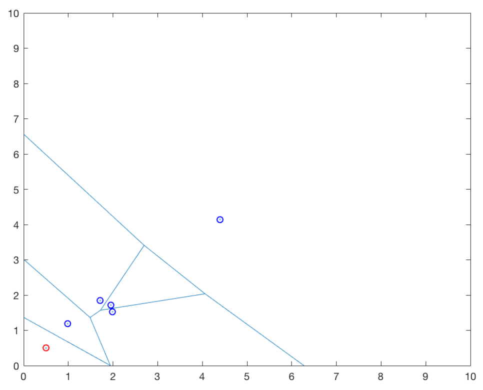
We discuss two simulation experiments of multi-pursuer vs. a single evader. In the first experiment, the evader is held stationary while being pursued. This is mainly to observe the group behavior of pursers in the game. In the second experiment, the evader is controlled by a human via a joystick, similar to the experiment done with the proposed linear program framework. The evader has a maximum speed that is equivalent to the pursuers'. The game is played in an area of the same size of the experiments discussed in Section 6.2.2.

Figure 6.10 show a simulation experiment of 5 pursuers vs. a stationary evader. As we can see from the trajectories in Figure 6.10(c), pursuers act independently with no effective coordination. The trajectories look like individual chasing which is less effective than strategies that would, for example, make pursuers enclose the evader. This observation is more obvious when the pursuers' start locations are almost co-linear as can be seen in Figure 6.11.

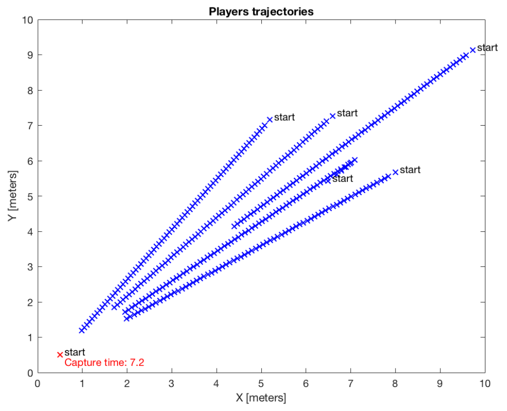
Figure 6.12 shows a different simulation run where a single evader is controlled via a joystick against 5 pursuers. Again, we can see the individual chase from the pursuers trajectories. This is mainly due to the fact that there is no coordination strategy between pursuers as the case in the LP-based approach. In particular, the LP-based approach, both the task assignment (which defender goes to which attacker) and the motion planning is solved concurrently. On the other hand, the Voronoi based approach solves only for the motion planning part.



(a)

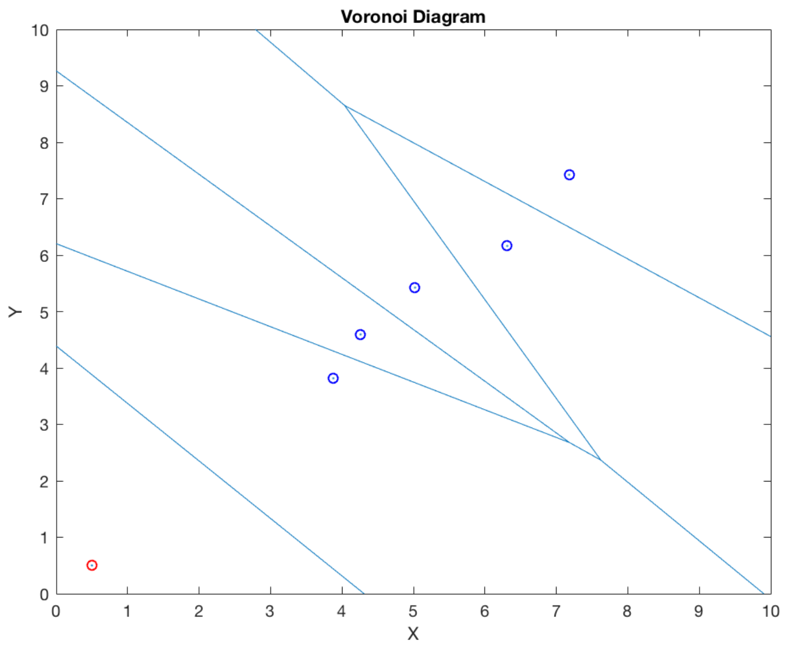


(b)

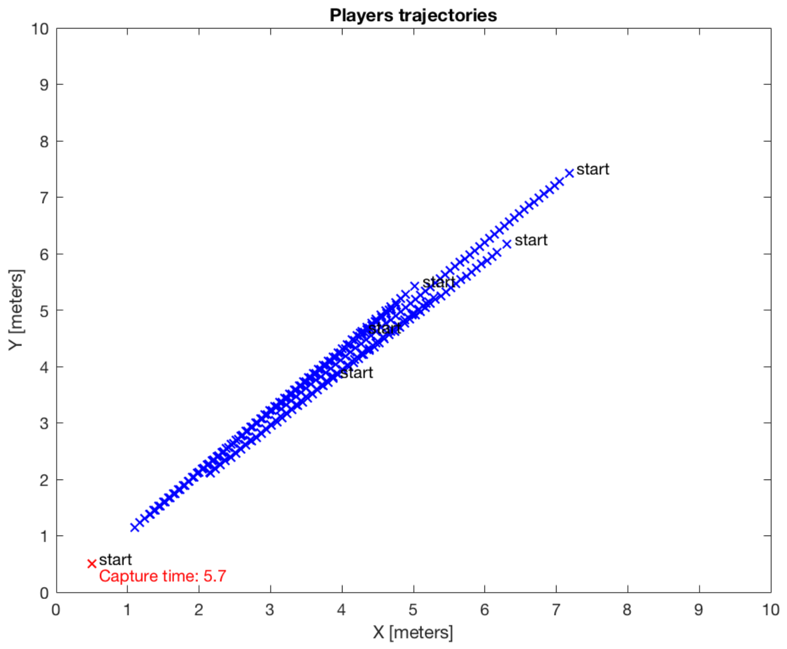


(c)

Figure 6.10: Voronoi diagram and trajectories of players in single evader vs. multi-pursuer game. The evader is stationary during the game. (a) shows the Voronoi tessellation of the initial locations of players. (b) shows the final Voronoi tessellation after the evader is captured at 1.0m distance. (c) shows the complete trajectory of the players during the game. The capture time is highlighted in red over the evader's capture location which is 7.2 seconds.

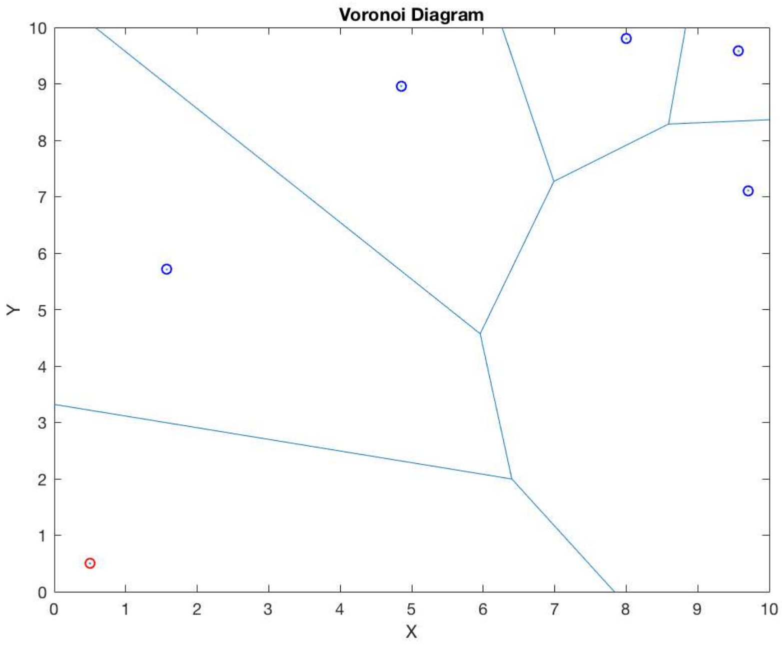


(a)

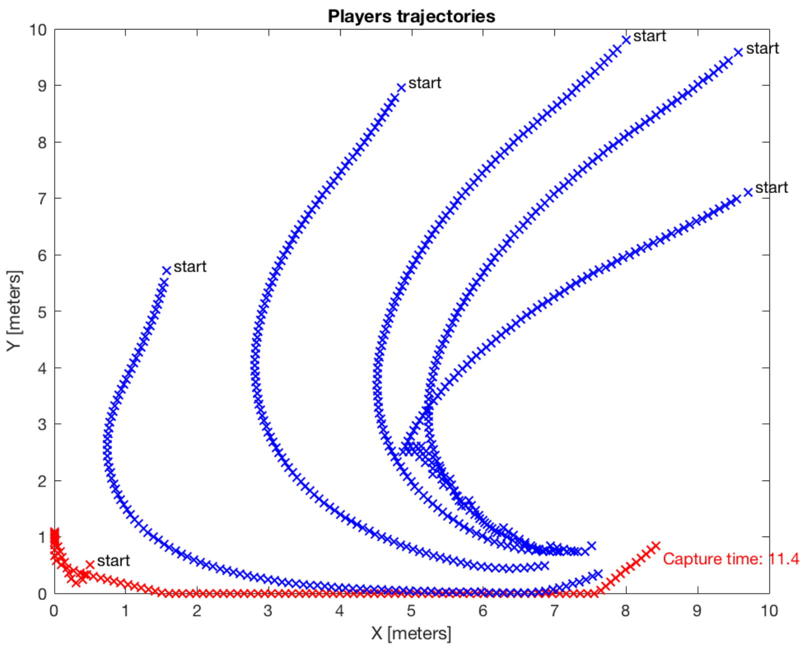


(b)

Figure 6.11: Simulation of a stationary evader vs. 5 pursers. (a) shows the initial players' configuration and they are almost co-linear. (b) shows players trajectories during the game run.



(a)



(b)

Figure 6.12: Simulation of joystick controlled evader against 5 pursuers.

6.3 Hardware Implementations

System integration and testing are key aspects of Robotics in general. As mentioned before, the main focus of this work, besides the algorithmic contribution, is to provide a complete physical system to validate the proposed real-time distributed algorithm. In multi-rotor UAVs, power consumption and weight are major constraints in their design, which limit their endurance. Therefore, it is always preferred to reduce payload and power consumption as much as possible. In that regard, we opt to use low-power and low-weight affordable components that allow us to achieve the computational needs and real-time execution for the proposed approach. We present a complete *indoor* and *outdoor* experimental setups that show that validity of the proposed framework.

In this section, we present our indoor/outdoor hardware setups and evaluate the performance of the proposed distributed algorithm in real-time. Although it is more realistic to perform experiments in a real outdoor environment, we start by testing it in a controlled indoor environment, in order to ensure better debugging and discover/fix possible malfunctioning.

6.3.1 Indoor Setup

The adversarial game is demonstrated in an indoor lab setup which is used to validate the overall system integration and the main framework aspects. The setup mainly consists of a set of small drones that carry computational modules and battle inside an arena. The indoor flying arena is equipped with a motion capture system (or MOCAP) to provide indoor localization needed by the drones as GPS devices do not work in such closed environment. The hardware components are summarized as follows:

- Indoor flying arena equipped with OptiTrack motion capture system. The motion capture system is used to provide local positioning for in indoor setup as GPS does not in such settings.

- Quadrotors: 25cm wide
- Pixhawk: Open-source autopilot that runs the PX4 firmware which is the same autopilot used in the simulation setup
- Odroid XU4: Onboard embedded Linux computer that executes the local LP algorithm. It also has WiFi communication modules to communicate positioning data to the low-level autopilot
- Communication: A single WiFi router is used to provide all necessary communication. All drones communicate to the WiFi router.

The system architecture is depicted in Fig. 6.13, and detailed description is as follows. We perform our indoor flight test, in the flying arena of the Robotics, Intelligent Systems & Control (RISC) lab. The arena has dimensions of $6 \times 7 \times 3\ m^3$. It is equipped with OptiTrack motion capture system[4] which acts as an indoor positioning system, as GPS receivers do not work indoors. The motion capture system provides position information of defined rigid bodies e.g. quadrotors. A desktop Linux machine equipped with ROS receives the position information from motion capture system and publishes them as ROS topics which are then transmitted to each UAV, to become aware of its position in the indoor environment.

Each UAV is a small quadrotor of 0.25m width, see Fig. 6.3.1. It is equipped with a Pixhawk autopilot which runs open-source PX4 firmware that is used for the attitude, velocity, and position stabilization. Pixhawk is serially connected to an onboard embedded Linux computer, ODROID XU4. ODROID XU4 runs ROS Indigo, MAVROS package (ROS interface to Pixhawk), and a ROS node that interfaces the

[4] `http://optitrack.com`

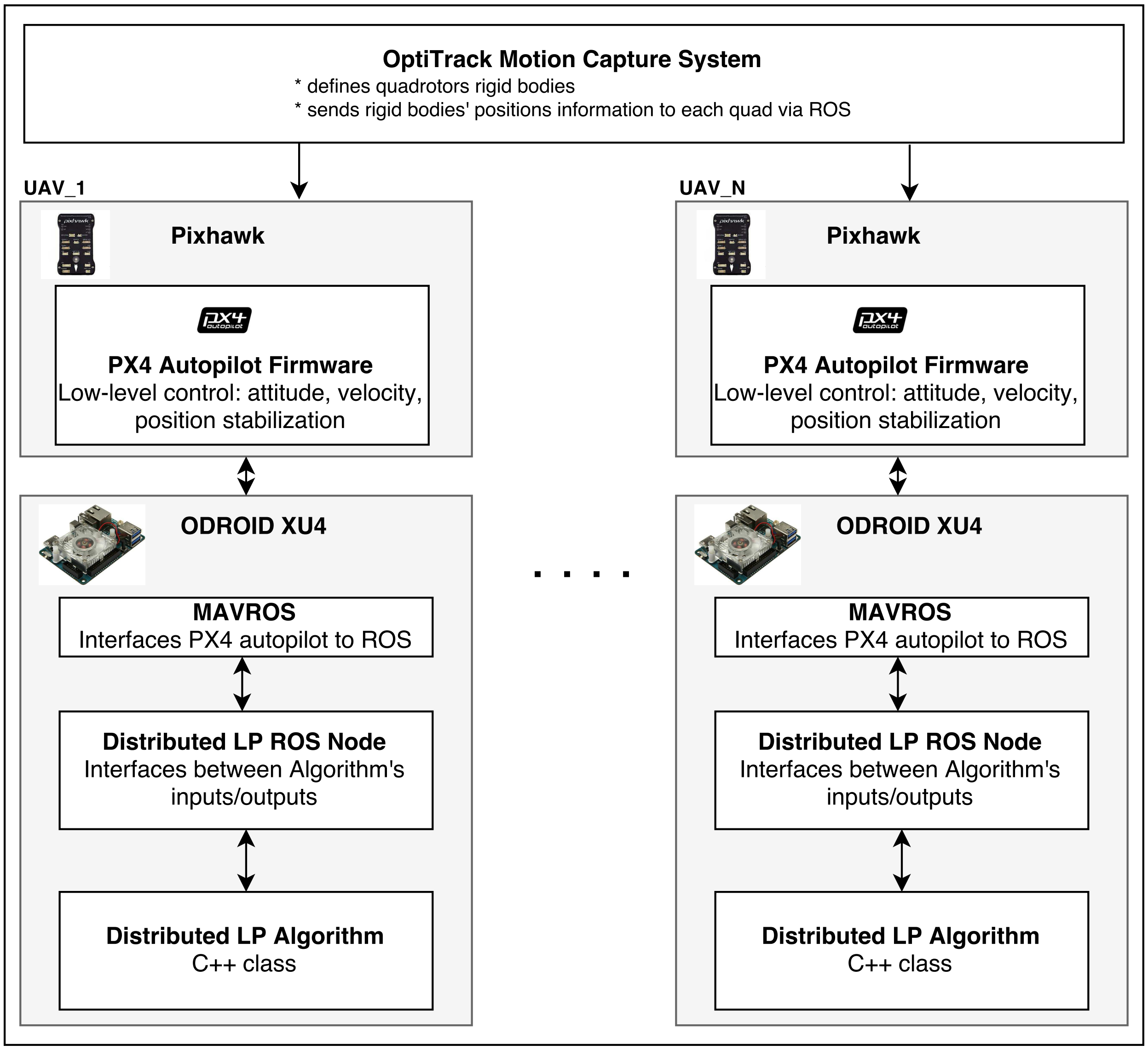

OptiTrack Motion Capture System
* defines quadrotors rigid bodies
* sends rigid bodies' positions information to each quad via ROS
UAV_1
Pixhawk
PX4 Autopilot Firmware
Low-level control: attitude, velocity, position stabilization
ODROID XU4
MAVROS
Interfaces PX4 autopilot to ROS
Distributed LP ROS Node
Interfaces between Algorithm's inputs/outputs
Distributed LP Algorithm
C++ class
. . . .
UAV_N
Pixhawk
PX4 Autopilot Firmware
Low-level control: attitude, velocity, position stabilization
ODROID XU4
MAVROS
Interfaces PX4 autopilot to ROS
Distributed LP ROS Node
Interfaces between Algorithm's inputs/outputs
Distributed LP Algorithm
C++ class

Figure 6.13: Architecture of hardware setup. A motion capture system (OptiTrack) is used to provide position information to all drones. Each drone uses the positions information to localize itself and to optimize its paths with respect to its neighbors. There are two computational modules. One is the Pixhawk flight conroller which provides low-level flight stability. A higher level module, ODROID XU4 is used to execute the planning algorithm.

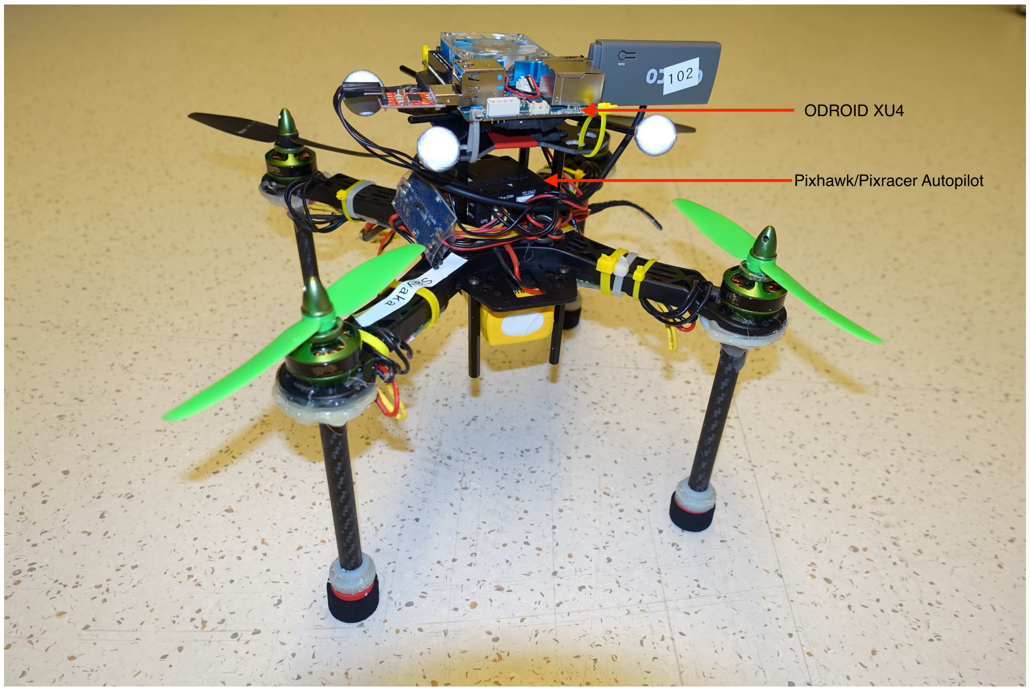


Figure 6.14: Quadrotor setup. The propulsion system composed of 2300Kv motors with 5-inch propellers and 20A speed controllers. The power is provided through a 3S LiPo battery

proposed algorithm to ROS. ODROID mainly receives positions from motion capture system via WiFi, executes the proposed algorithm, and sends position setpoints to Pixhawk.

6.3.1.1 Algorithm Onboard Execution Speed

Prior to flight test experiments, we first test the execution speed of the algorithm on ODROID XU4 for different problem setups. This allows the selection of a reasonable problem configuration that can be run in real-time on the hardware of interest. Table 6.1 shows the average execution speed of one run of the algorithm on ODROID XU4, in Hz, for different problem setups.

As we can see from Table 6.1, the proposed algorithm can be executed at relatively high rates for different problem configurations. In our experiments, we choose configuration 5. The corresponding optimization problem size in terms of number of variables

Table 6.1: Algorithm execution speed on ODROID XU4

Setup #	Sectors	Agents	Prediction steps (T_p)	**Freq.** (Hz)
1	10×10	3 vs. 1	3	2.3
2	10×10	3 vs. 1	2	5.5
3	10×10	3 vs. 1	1	22.7
4	7×7	3 vs. 1	3	11.0
5	7×7	3 vs. 1	2	27.8
6	7×7	3 vs. 1	1	111.0

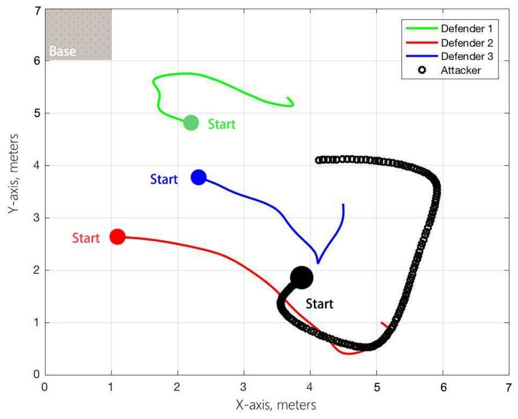
and constraints is $(n_s+n_s^2)T_p = 4900$, $(5n_s+2n_s^2)T_p = 10094$, respectively. It is important to note that the problem structure is highly sparse. The total number of non-zero element in the overall constraints matrix is of order $\mathcal{O}((2T_p + 4N^2)n_s^2)$, and the total number of all elements is of order $\mathcal{O}(2T_p^2 n_s^4)$. Where $N = \max_i |\mathcal{N}(s_i)| \ \ i \in \{1, \cdots, n_s\}$ is the maximum number of reachable sectors in 1 time step. For the chosen problem configuration, the sparsity is approximately 98%. This sparsity is taken into account in the algorithm implementation, which, in result, dramatically boosts the computation speeds on ODROID XU4 as presented in Table 6.1.

Fig. 6.15(c) show executions frequency profiles of 3 defenders quadrotors during a game trial. The average execution speed of the algorithm was around 30Hz. Such relatively high rate allows quadrotors to produce fast decisions, which result in smooth and rapid reactions against attackers.

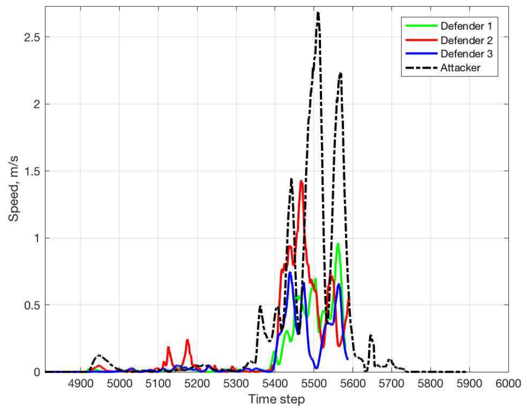
Although each drone can get all positions of all other drones in the indoor flying arena, each defender drone only uses the position information of defined neighbor drones only and ignores the rest. Neighbor drones are those within 2-sectors away in each direction.

6.3.1.2 Indoor Trial

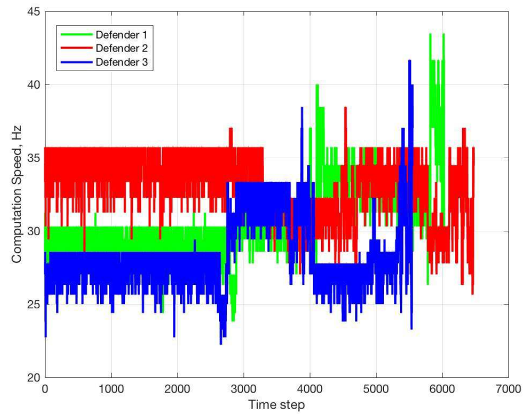
Hardware trials were performed using similar configuration as in the simulation test. The indoor setup can be seen in Figure 6.3.1.2. Figure 6.3.1.2 shows a sequence of



(a) Position trajectories



(b) Velocity trajectories



(c) Execution frequency profile of defenders during a real trial

Figure 6.15: Fig. (a) shows position trajectories of all quadrotors during a real experiment where the attacker was captured at position $(4.1, 4.1)$. Fig. (b) shows the corresponding speed profile. Fig. (d) shows the algorithm execution speed for all defenders during the game run, where the average is around 30 Hz

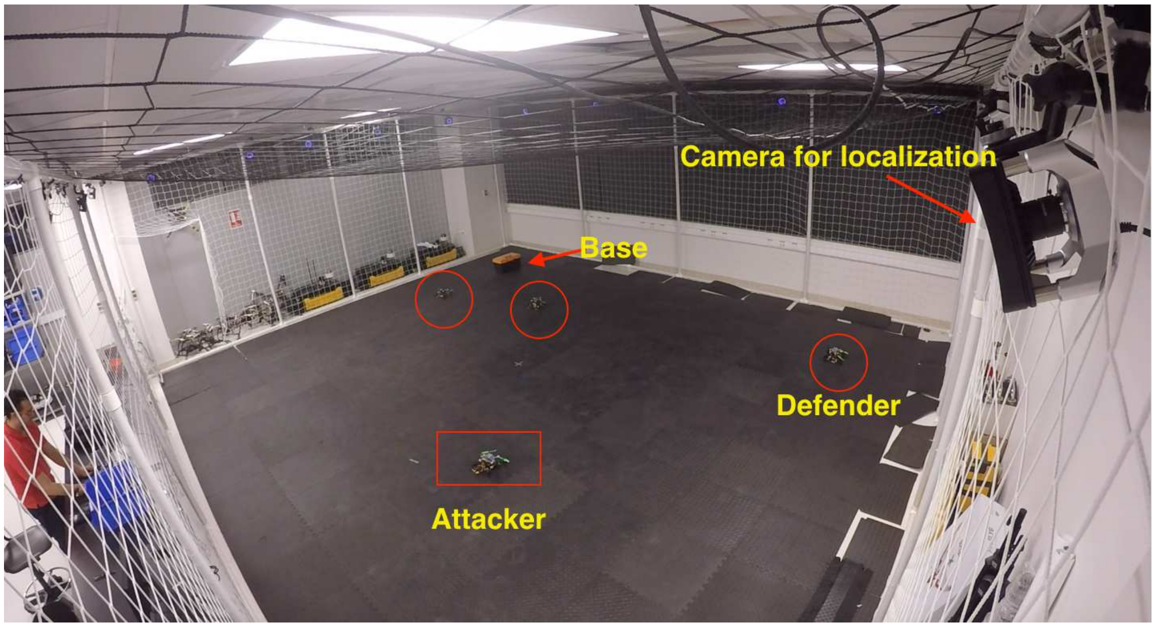


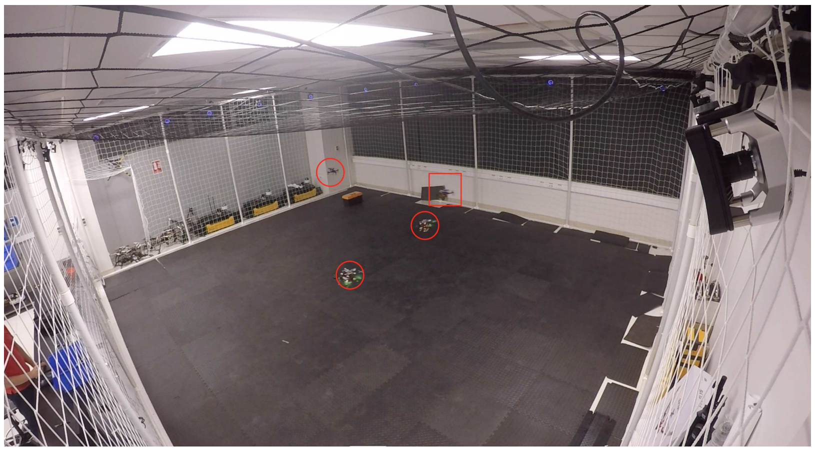
Figure 6.16: A Snapshot of hardware setup during a game trial of a human controlled attacker against 3 defenders executing the local LP algorithm. Defenders are surrounded by circles and the attacker is surrounded buy a rectangle. The base zone is represented by a toolbox at the upper-left corner.

snapshot from a game trial of a human controlled attacker against 3 defenders that executed the local LP algorithm. The corresponding trajectories are shown in Figure 6.15(a), where the attacker was captured at position $(4.1, 4.1)$. The corresponding velocity profile is depicted in Figure 6.15(b). In this trial the human operator tried to drag the defenders towards the back and then take a round path through the right of the flying arena. The two defenders with the red and blue trajectories were dragged towards the attacker (more weight on the attacking behavior). However, the third defender with the green trajectory were closer to the base zone. Eventually, the attacker was captured by both the green and blue defenders, being one sector away from them although it had more speed profile than the defenders.

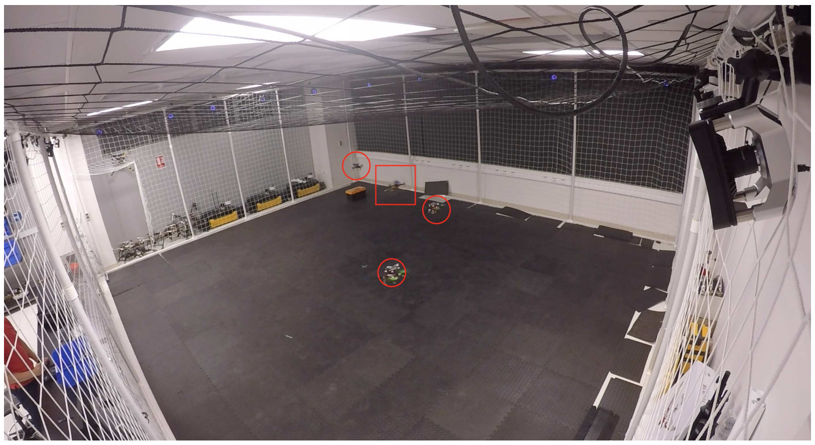
6.3.2 Outdoor Setup

At the submission time of this draft, the outdoor test were ongoing and not finalized. However, the latest updates are reported and final results will be included in the final

(a) Agents in initial maneuvers.



(b) Agents in pre-capture state.



(c) End of the game where attacker is captured.

Figure 6.17: Snapshots of agents maneuver during a game trial of a human controlled attacker (surrounded by a rectangle) against 3 defenders (surrounded by circles).

version of this thesis.

Outdoor tests provide much more realistic environment in several aspects. First, compared to limited indoor space, much larger spaces can be available which provides more opportunities to experience many different maneuverabilities from both attackers and defenders. The outdoor environment has different disturbance resources such as wind disturbance and communication interference. This can help to evaluate, for example, the actual motion of the robots against the assumed models.

The outdoor setup differs from the indoor one mainly in the localization method. In the discussed indoor setup, the localization accuracy was in the order of millimeter in ally 3 dimensions. In outdoor setup, however, it is not the case. Instead, an access to GPS positioning system is available. So, the drones are equipped with GPS receivers that provide an adequate horizontal positioning with $1m$ accuracy. The altitude estimation is usually unreliable if high accuracy is required. To overcome this limitation, we use accurate altimeter sensors, LIDAR, which provides precision altitude estimation with accuracy of the order of $10cm$. This is required as the game is assumed to be played in 2D, where the altitude is fixed.

6.3.2.1 Outdoor Scenario

We considered an outdoor scenario of 3-defender vs. a human-controlled attacker. The outdoor test was performed at a green park at KAUST with a predefined 8×8 grid cells with actual dimensions of 30×30 m^2. The main outdoor setup is summarized as follows.

- Drones: A $45cm$-wide quadrotor is used which carries the same modules as in the indoor setup. Additionally, they carry a GPS and altimeter modules for outdoor positioning. A picture of the drone is shown in Figure 6.3.2.1. Three quadrotors of equivalent capabilities were used as defenders against a fourth one that was used as human controlled attacker.

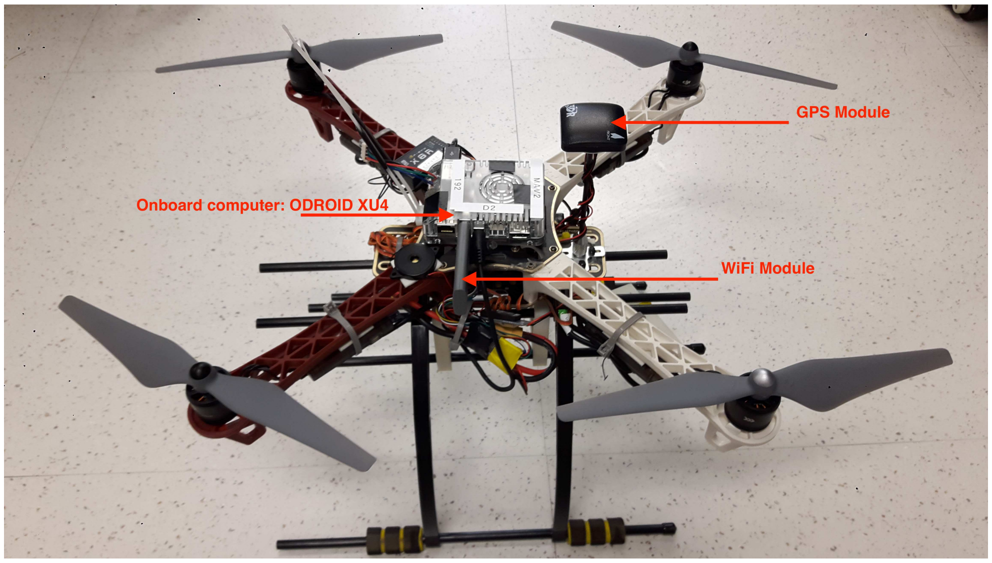


Figure 6.18: A 45cm-wide quadrotor which is used in the outdoor tests.

- Communication: All communication go through a single wifi router that operates in the 5GHz bandwidth which was found less interfered.
- Computational modules: The same computational modules as the indoor setup were used. A Pixhawk controller was used as the low-level autopilot, and an ODROID XU4 was used for the high-level path planning.

Similar to the simulation and indoor tests, the battle begins once a battle signal is triggered, and it ends once all attackers are captured. The capture event is defined by a specific distance between an attacker and any defender. An attacker is commanded to land once capture and considered out f the game. Once all attackers are captured, defenders are commanded to land. If an attacker wins, by entering the defense zone, all defenders are commanded to land and attackers are allowed to stay in the air until they are commanded to land by operators. In the considered outdoor setup, only one attacker is considered which is controller by a human operator.

An aerial shot during an outdoor test is shown in Figure 6.3.2.1. Figure shows a 2D representation of drones position trajectories during an outdoor experiment of 3-defender vs. a human-controlled attacker.

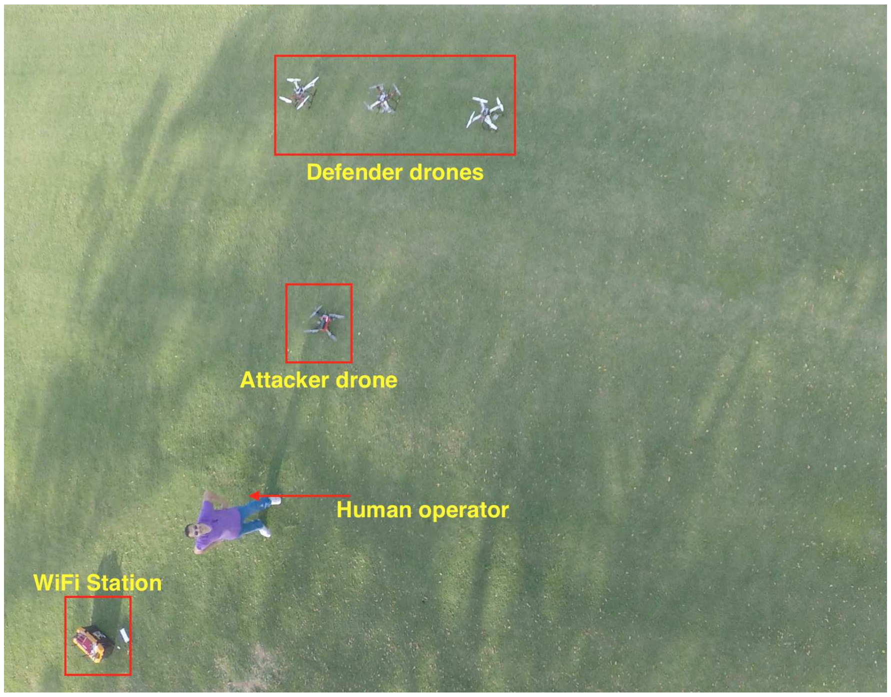


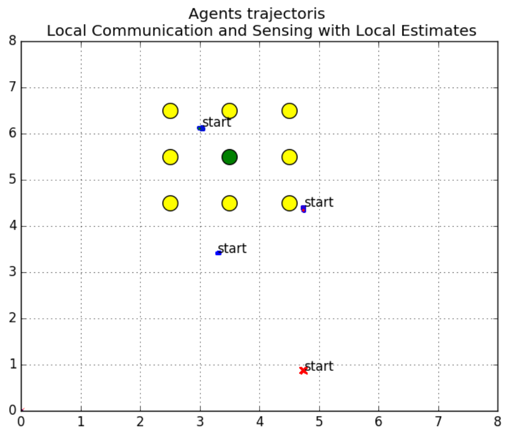
Figure 6.19: An aerial snapshot of the outdoor setup.

For the considered outdoor setup, grid of 8×8 and a prediction horizon of 3 steps in time, the algorithm was executed onboard at frequency of 10Hz, which is less than the smaller grid considered in the indoor setup, but in the acceptable range.

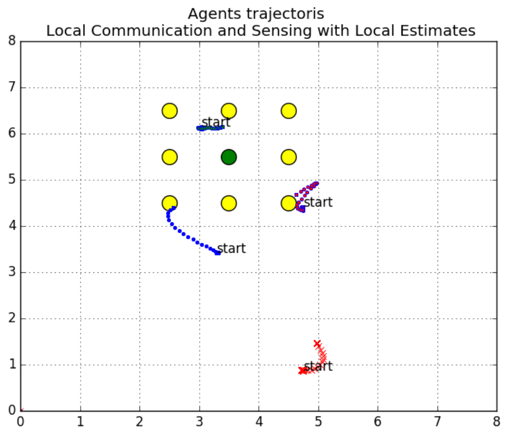
6.4 Summary

This chapter presented a realistic simulation and hardware implementation of the proposed distributed linear program algorithm in the application of adversarial game of interest. The Simulation is realistic in the sense that the adversarial game was implemented using simulated drones in the Gazebo Simulator as well as actual drones. The Gazebo simulator provides a realistic simulation of rigid body dynamics. This, indeed, helps evaluating the proposed algorithm in a more realistic simulation environments and also reduces the required efforts to transition to actual hardware implementation.

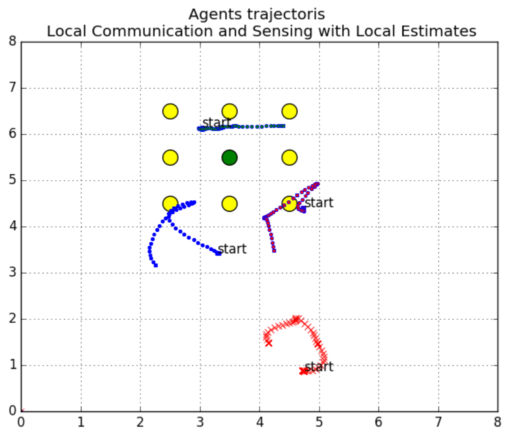
Another important contribution is the system integration and validation in a com-



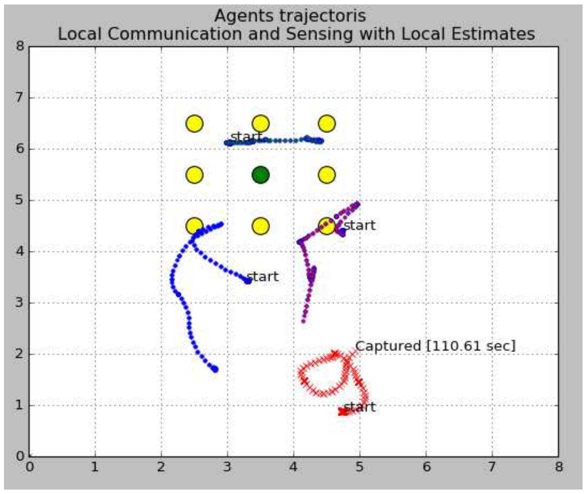
(a)



(b)



(c)



(d)

Figure 6.20: 2D representation of the drones trajectories in an outdoor experiment. Defenders have blue trajectories. The attacker has a red trajectory. The upper left figure shows the initial positions of the drones. The next ones are sequential snapshots of the trajectories ending with capture snapshot in the lower right figure.

plete hardware implementation. This is also accomplished using real drones that execute all required software including the distributed planning algorithm onboard. The most important point is that, it was shown that the algorithm executes in real-time, e.g. in the order of 10s of Hz on low power computing modules. This is attractive for large swarms of small drones that are limited in payload and power consumption. We provided two physical implementation setups, indoor and outdoor. The indoor setup help for rapid testing in a controlled environment and to validate important systems aspects such as real-time performance, communication stability, repeatability, and overall system integrity. The presented outdoor setup provides a more realistic testing cases where more maneuverability can be executed under real environment conditions.

Several game scenarios are presented to show the performance of the algorithm. It's trivial that the algorithm performance is not optimal compared to the centralized version because defenders information about other defenders and attackers is limited to a defined neighborhood. Whereas, in the centralized version, the position of all agents in the grid is known to all defenders at all times. However, defenders exploit the fact that they know each other's motion models in order to produce behaviors that are meaningful under the real-time constraints. Also in such applications, real-time urgency is favored over performance optimality.

The system integration and validation also revealed some areas of future improvements. One aspect is related to agents' motion modeling. Although multirotors are agile aerial vehicles which can execute aggressive maneuvers, they are still constrained by their inertia, which limit abrupt changes in flying direction. For example, a quadrotor cannot reverse its flying direction instantaneously, which is assumed by the motion model of the linear program algorithm. This also applies on the attackers motion model. A more realistic model that can capture such limitations, can help to better predict the decisions of both teammates and adversaries. However, this can add more complexity into the problem formulation, e.g. adds nonlinearity, which can break the linear program formulation that is appealing for real-time execution. Another aspect that can be improved is the real-time execution under larger grid. In some scenarioes, it might be required to consider larger grids in order to have more resolution and accurate maneuvers. As reported in Table 6.1, the execution speed on the considered onboard computer showed dramatic decrease as a function of the grid size. It would be more practical if the execution speed could be kept at constant rate or at least linear with respect to grid size in order to respect the real-time constraints.

One of the main outcomes of this work is that all the software that is used to obtain the simulation and hardware results is available as an open-source package which is also compatible with ROS. This helps to reproduce this work and to have less time improving on the existing work. The package is available at [86]

Chapter 7

Additional Contributions

7.1 Introduction

In this chapter I provide additional contributions besides the main PhD work of this thesis. In my broader interest in multi-robot applications, I was involved in three additional works that are summarized as follows.

- Design of a complete multi-drone system for collaborative search, pick, and drop of randomly placed objects
- Design of a realistic simulation environment for implementation of distributed motion planning based on submodular functions
- Design of drone platforms for onboard multi-robot localization for GPS-denied environment

My contributions in the previous works are detailed in the following sections.

7.2 Cooperative Multi-UAV System for Autonomous Aerial Grasping in Outdoor Environments

This project is motivated by the Mohamed Bin Zayed International Robotics Competition (MBZIRC) that took place in Abu Dhabi, UAE, in March 2017. The challenge setup considered in this work is given as follows. A team of UAVs has to collaborate in order to search, localize, track and pick up a set of static objects autonomously.

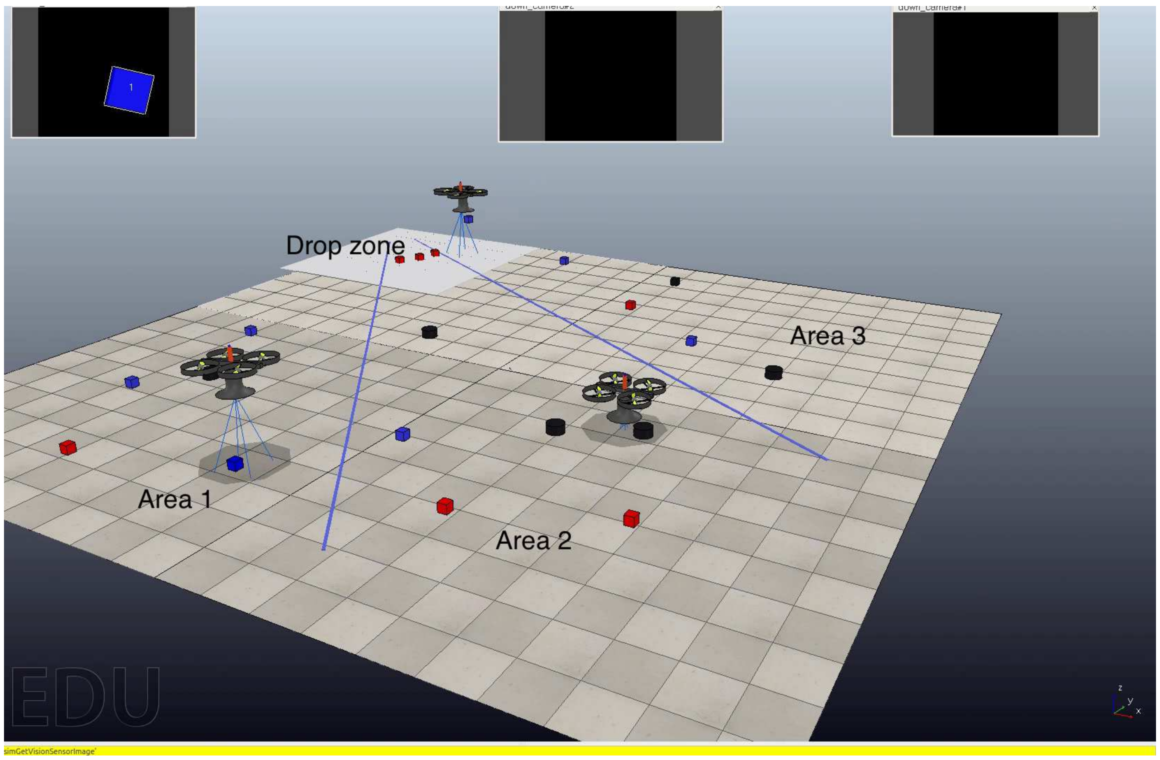


Figure 7.1: An example of the system setup while being simulated in the V-REP robotic simulator, where three drones are considered. The search area is divided into three sub-areas, one for each drone. Objects are placed randomly in the search space

The objects are known to be of a ferrous material, and may consist of various sizes, shapes, and colors which need to be transported to a dedicated single drop zone. The search area is defined by a set of GPS coordinates that are known a priori. This problem brings up a set of practical research and system design questions regarding multi-UAV coordinated control, aerial object grasping, and vision-based object identification and localization within a limited time. A simulation snapshot of the system is depicted in Figure 7.1.

Despite the excess of literature in this area, it is noticeable that most of the work that has been done in the past on the implementation of cooperative multi-agent systems using UAVs, is in indoor environments [89], [90], in presence of perfect positioning and precise localization, optimal lighting conditions, and robust communication infrastructure. However, in outdoor environments, the implementation of multi-UAV system is more challenging because of the disturbances and uncertainties in surroundings. In this paper, we focus on an outdoor implementation and integration of multiple UAVs to complete a given complex task cooperatively and autonomously. We tackle the challenge of autonomous cooperative multi-UAV aerial grasping in outdoor environments from a practical perspective. Another constraint that greatly hinders the operation of UAVs autonomously outdoors is the need for onboard computation, because of the power and payload limitations of UAVs. In most of the cited work, the computations and high-level algorithms are run off-board, which is an acceptable for indoor lab experiments, but for realistic and outdoor applications in which complete, or high-degree of autonomy of the robot is desired, onboard computation is a condition that must be satisfied. Hence throughout this paper, we only deal with and propose the strategies for fully onboard control and computation capability for all UAVs.

As discussed earlier, aerial grasping is among the few top research interests of people working in the field of aerial robotics. Several useful techniques have been proposed and experimented with UAVs for grasping of objects of various shapes, textures, weights, and sizes over the years [91], [92], [93]. Though a lot of these strategies focus on the versatility of grasping, rather than its robustness and precision, which is totally fine as far as the research is concerned, but it is easy to notice that for many practical and industrial applications, the need to grasp ferrous objects, in particular, remains a key objective. It is because of the well-known reliability and strength of the ferrous enclosures. Given most of the aerial grasping mechanisms put high constraints on the maximum payload limit as well as the aerial maneuvers of the robot itself, we came up with an intelligent and customized gripper design which is based on maximizing the payload capability while keeping the constraints on the aerial maneuvers of the robot to a minimum. We specifically design and implement a passive magnetic gripper for the outdoor multi-agent setup, by extending our concept of passive aerial grasping of ferrous objects in outdoor environments [94] and by combining it with the impulsive drop mechanism [95], for aerial grasping with UAVs.

The main attribute which makes the work presented in this paper unique is our low cost, fully onboard, and autonomous implementation of real-time cooperation strategies, on multiple UAVs for an important and practical application of industrial and commercial importance, that is aerial grasping. Being able to demonstrate the robust performance of our multi-agent system in outdoor environment adds further to the value of the project. Several pivotal components of the system such as the grasping and drop mechanisms have been developed completely in-house as well and their superior performance is demonstrated by experiments.

7.2.1 Contributions

In this work, my contributions are summarized as follows.

- Design of customized drone that carries a customized passive gripper for magnetic objects, vision system for object identification, and onboard computer for mission software execution

- Design of the software architecture which includes the finite state machine of the mission, communication and coordination protocol for dropping agreement, object picking strategy, and the ROS interface of the software

In this work we used three customized hexarotors to accomplish the mission, see Figures 7.2, 7.3. The onbarod components that each drone carries are summarized in Table 7.1.

7.2.1.1 Gripper Design

A major component of the grasping dron is the gripper design. There is a number of works which presented indoor demonstrations of aerial grasping. For example, a

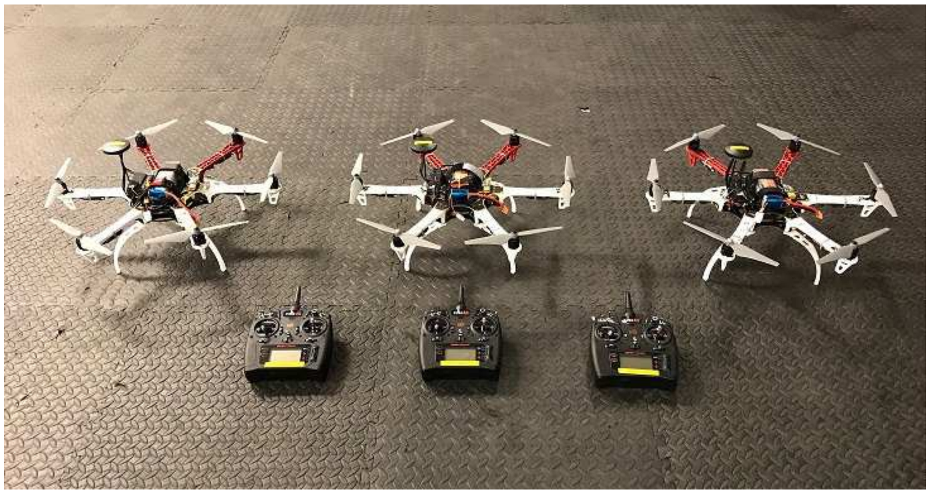


Figure 7.2: Cooperative multi-UAV system for aerial grasping in outdoor environments

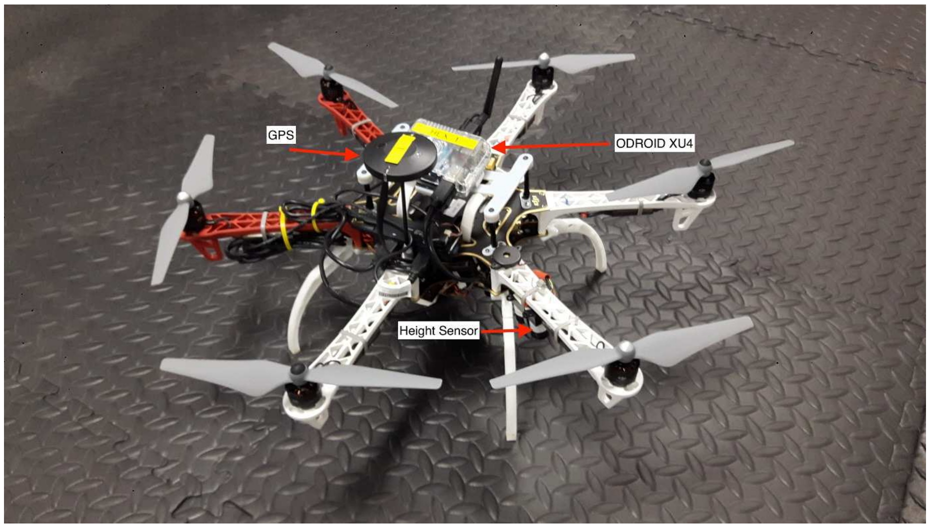


Figure 7.3: Fully equipped hexarotor platform

Table 7.1: UAV hardware components

Item	Description
Frame:	DJI flamewheel F550 hexacopter
Propulsion system:	DJI E310 900KV with 9 inch propellers
Battery:	10Ah 4 cells LiPo battery
Flight controller:	Pixhawk 2 (the Cube)
Onboard computer:	Odroid XU4
Altitude sensor:	LiDAR Lite v3 sensor with 40m range
Camera:	Fish eye ELP camera
Gripper:	Customized design with permanent magnets & release mechanism

multi quadrotor setup to carry a penetrable payload with grippers [89], or with cables [90]. Also, in [96] and [97], the authors presented multi-MAV systems for structure assembly. In all previous works, experiments were done indoors and with known locations of objects.

In this work, we present a simple light-weight gripping mechanism for ferrous objects with feedback on the picking state. An important aspect of this whole task was to design a robust and reliable grasping mechanism for the UAVs, which enables them to pick up and drop the target ferrous objects in a reliable manner. The payload is an important consideration while designing a gripper for drones. We would like to keep the grip as strong as possible while keeping its own weight to a minimum. Thus for ferrous grasping application, we investigated various options including electromagnets, Electro-Permanent Magnets (EPMs), and Permanent Magnets. Low power consumption compared to electromagnets, high payload capability, and convenient commercial availability of the EPMs, apparently, make them a default choice. However, we designed our own magnetic gripper and a novel impulsive servo-actuated drop mechanism that conforms with the requirements of the actual system. The mechanism is based on our earlier work [94] and is inspired by our participation in MBZIRC Robotics Challenge 2017 [98]. Fig. 7.4 shows the complete gripper assembly mounted to the hexarotor frame. All the assembly parts have been designed and printed via the Objet 30 Prime 3D printer at RISC Lab. The whole gripper, when assembled, weighs around 250g. The servo mount holds everything in place. The square magnetic pad at the heart of the mechanism is the key to spontaneous grasping. It employs four 6.33 mm cubes of N42 Neodymium magnets. These magnets are collectively capable of providing a net lift of around 0.76kg. For our testing, the test objects we used, weigh 350g at maximum. Thus, one pad does the job for us. It is further embedded, in its center with a push button, that is pressed every time the gripper picks up or drops an object. As is described later in this section, this is a vital feature for ensuring a flawless autonomous flow of the state machine during the grasping operation.

The drop mechanism as shown in the Fig. 7.4 consists of two high-speed servo actuators, which when activated push the object off the magnetic pad using their respective horns. the two servos are mounted at right angles to each other ensuring a strong push, when activated at the same time. This concept of impulsive drop [95] is quite efficient in terms of design simplicity as well as power consumption, since the only time the gripper consumes power is in the drop phase. The average power consumption over a complete pick and drop cycle of gripper operation is thus comparable or less than an EPM which requires additional power to activate the magnets as well, which in our case is zero, since the permanent magnets are always activated. An Arduino Nano serves as a dedicated ROS node for controlling the gripper actuation. It reads the push button feedback from the magnetic pad and publishes the pick/drop status to ODROID in real-time. It is subscribed to pick/drop commands from the ODROID as well, in response to which it either activates or deactivates the drop servo mechanism.

In addition to the gripper, the assembly also has a built-in small sized servo gimbal for the camera module. This customized 3D printed gimbal uses two Hitech ultra-nano servos to stabilize the roll and pitch of the camera as the UAV flies and carries

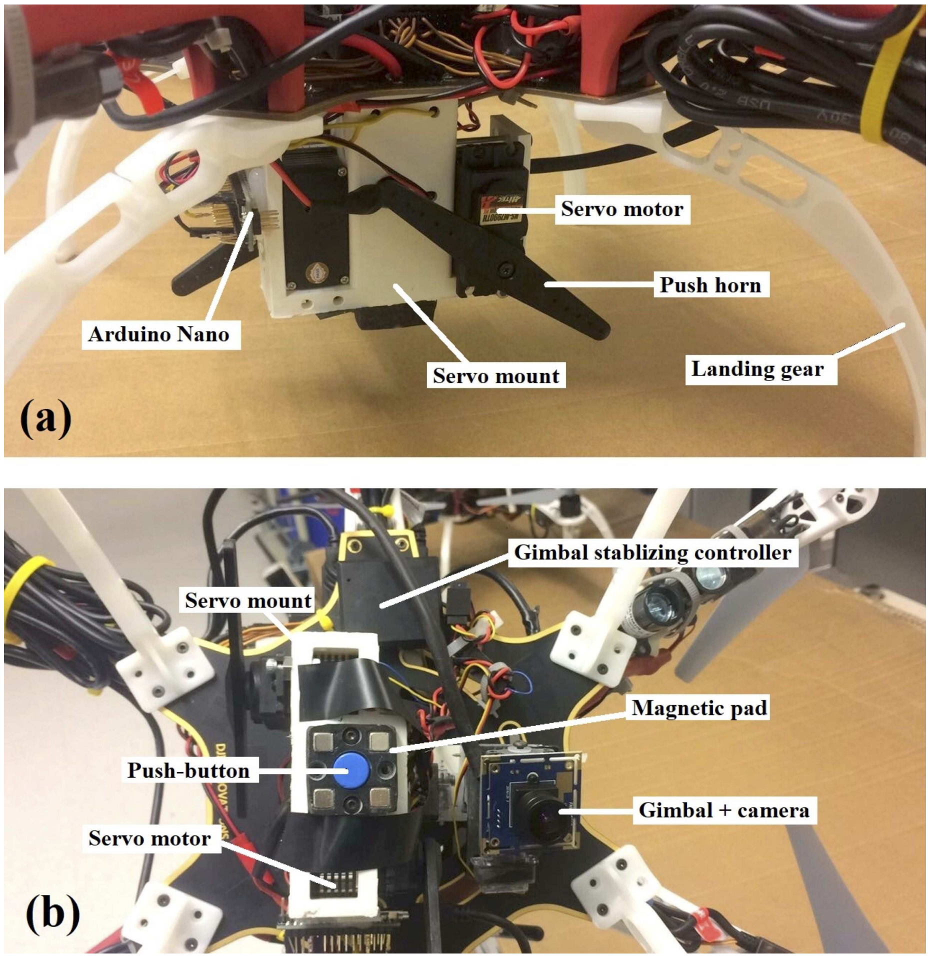


Figure 7.4: Gripper design: (a) side view, and (b) bottom view. The 3D printed gripper enclosure holds two servo motors, four permanent magnets, push-button for feedback, gimbaled camera with its holder, and Arduino Nano for actuation control and ROS interface

out various maneuvers. This gimbal keeps the camera faced down, aligned with the ground all the time, which makes the object tracking and detection convenient.

Each of the three UAVs in our setup was equipped with the same gripping mechanism. The actuation and grasping routine for any UAV proceeds as follows: The magnets are activated by default. In the pick-up state, the servos are deactivated i.e. the horns rest above the magnetic pad. Thus as a UAV tracks, descends and picks up an object, the feedback signal from the push button switches from '0' to '1'. A '0' means an object is not picked, while a '1' means that an object has been picked up successfully. Thus '1' message serves as a pick-up confirmation for the main state machine. Now when a UAV reaches the drop zone, the Arduino (ROS node) receives a drop signal from the ODROID (main state machine), and hence it activates the drop mechanism. As the object is dropped, the push-button feedback switches from '1' to '0'. Similar to pick up routine, the '0' message serves as the drop confirmation for the main state machine. Once it gets the confirmation, it proceeds to the next state (i.e. search and pick up) and also sends a pickup signal to the Arduino which deactivates the drop mechanism again, and the process continues.

7.2.1.2 Communications

In this work we used WiFi to establish peer-to-peer communication links between the three drones. Communication is only needed for the drones to resolve conflicts during the dropping stage as only one drone can be inside the drop zone. We also avoided the use of single ROS master in ROS system. In the standard ROS network, there is a single master node that establishes the communication links between all nodes in the ROS network. To avoid the single point of failure of the single master node, each drone runs its own ROS master independently. However, this way drones loose the convenient ROS communications to exchange information. For that, we established an additional peer-to-peer communications between drones using TCP

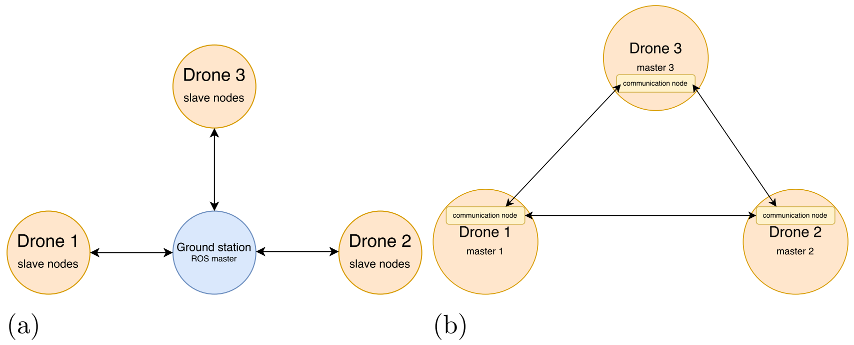
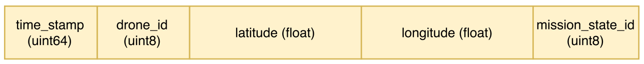


Figure 7.5: Inter-UAV communication architecture. Figure (a) shows the standard ROS communication way, which is prone to single point of failure issue. Figure (b) shows our customized communication method to mitigate this issue.



time_stamp (uint64)	drone_id (uint8)	latitude (float)	longitude (float)	mission_state_id (uint8)

Figure 7.6: Content of custom MAVLink message used in inter-drone communication

protocol. Figure 7.5 shows the standard ROS network and the modified one. The communicated information is encoded in custom MAVLink message, see Figure 7.6.

7.2.2 Outdoor Experiments

The complete system with the three drones was tested in an outdoor arena and a video of the complete system in action is available on RISC's YouTube channel [1].

Each drone was tested individually to confirm its successful operation including object search, identification and localization, object picking, and object dropping. Figure 7.7 shows snapshots of testing the individual tasks during an autonomous mission for a single drone.

Furthermore, a grand field experiment was conducted for the full autonomous mission in action and the order of execution was as follows.

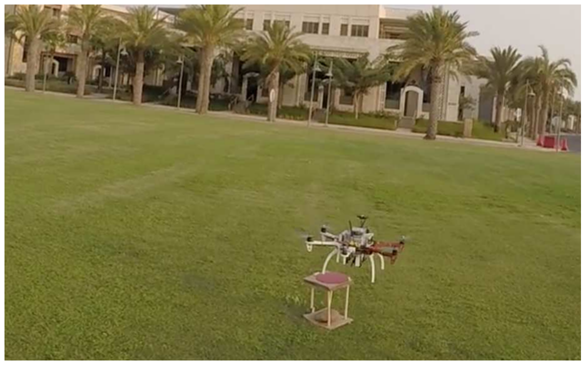
- Area sectioning: An area is defined by taking the GPS coordinates of its corners. The area is divided into non-shared sections, one for each drone.The GPS cor-

[1] `https://youtu.be/ZNolRs-CYew`

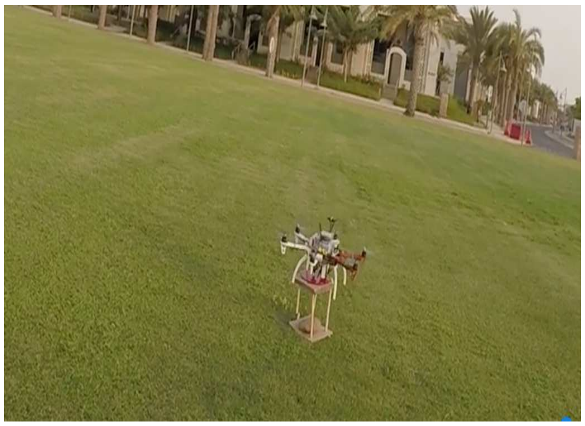
(a)

(b)

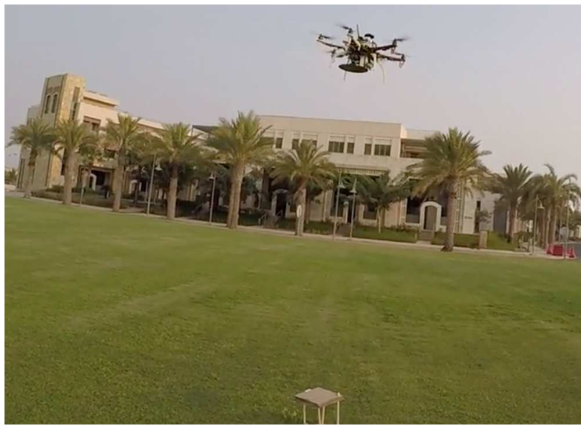
(c)



(d)



(e)



(f)

Figure 7.7: Single drone testing; (a) a snapshot of the drone in the search phase, (b) a snapshot of the drone after an object is found and selected, (c) a snapshot of the drone after an object is found and selected, (d) a snapshot of the drone while aligning over the object to prepare for picking, (e) a snapshot of the drone while picking the object, and (f) a snapshot of the drone after picking the object, going to the drop phase

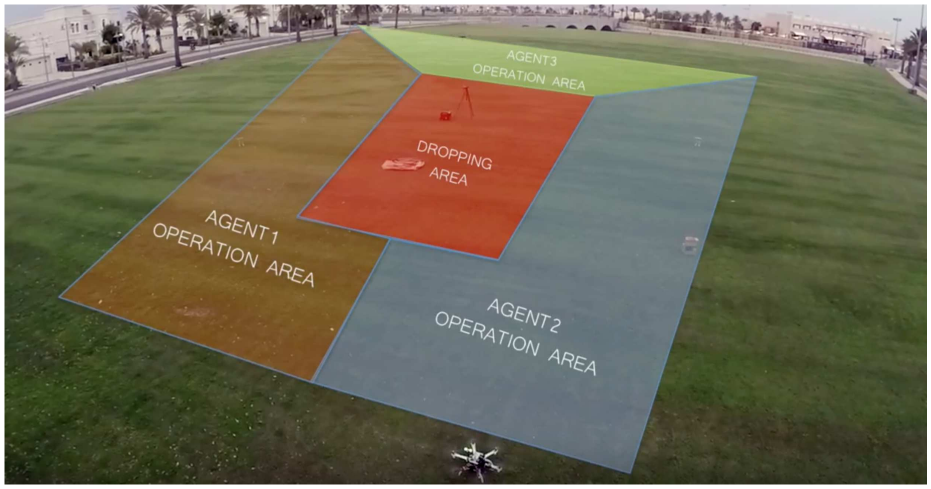
ners of the sections are recorded. Each drone is given the predefined coordinates and assigned to a particular section.

- Object Placement: As mentioned, magnetic discs of 20cm diameter are randomly placed inside the operational area.
- Drone placement: Each drone is randomly placed inside its assigned section.
- Mission Start: Once each drone is armed and ready, a start signal is triggered which initiates the full autonomous mission.
- Mission states: Each drone goes through a cycle of finite states until it clears all objects inside its assigned section. The states include object search, object picking, object dropping.
- Mission End: Once each drone has cleared its section, it lands. Once all drones are landed, the mission is complete.

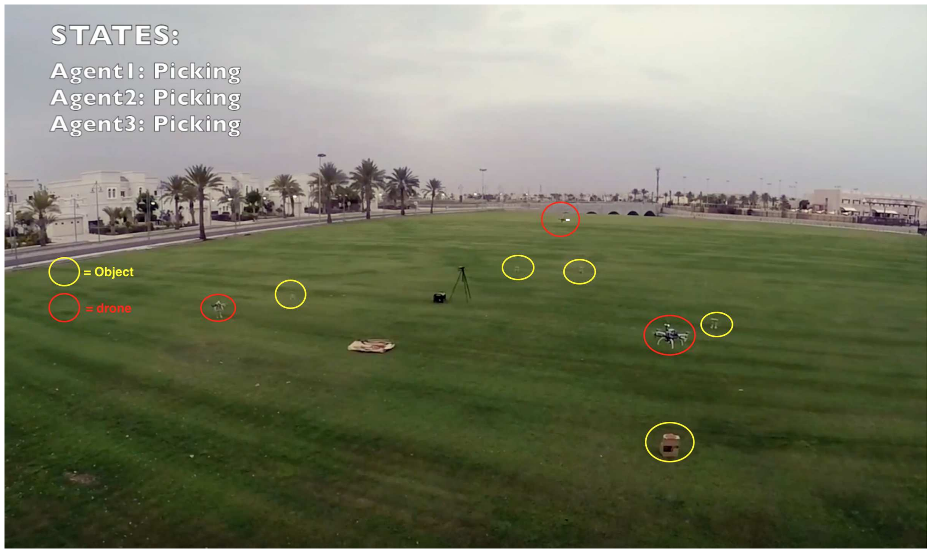
A single Linux computer was used to monitor the mission execution and drones states. The mission was completed successfully in 3 minutes. Fig. 7.8 shows snapshots of the grand experiment.

In our experiments, we used *rosbags* (data logging system in ROS) for data logging as it provides convenient tools for data visualization and time-stamped mission replays. Fig. 7.9 shows a snapshot of a log replay of one of the three drones during a complete mission. The logged data includes time-stamped processed gray-scale image where an object is encircled if it is detected, state of the mission, error distance to the current detected object, and gripping status (gripped or not).

During several autonomous missions, a main factor of success is the accurate object centering with respect to the drone's gripper which is a result of accurate object localization using vision. In Figure 7.11, a smooth descent can be seen while the object is being centered with respect to the gripper's center. This validates the

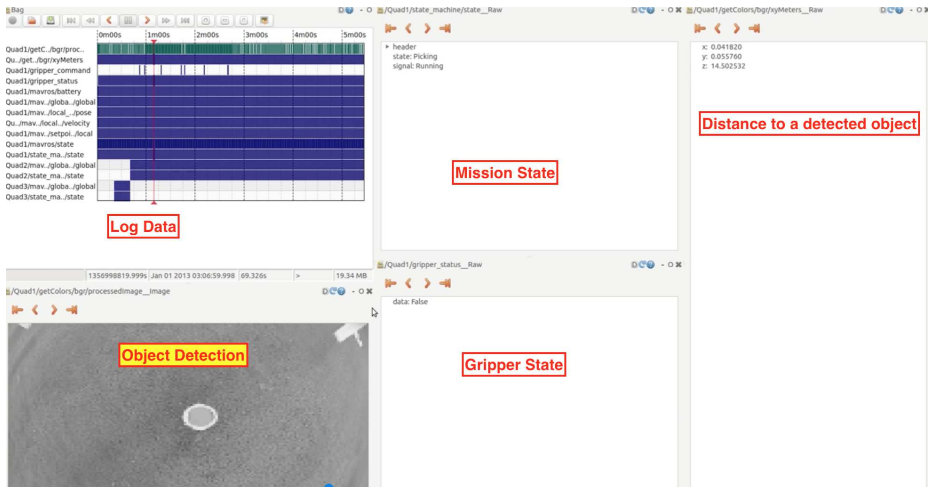


(a)



(b)

Figure 7.8: Snapshots from the grand field experiment


Bag
0m00s
1m00s
2m00s
3m00s
4m00s
5m00s
Quad1/getC../bgr/proc..
Qu../get../bgr/xyMeters
Quad1/gripper_command
Quad1/gripper_status
Quad1/mavros/battery
Quad1/mav../globa../global
Quad1/mav../local_../pose
Qu../mav../local../velocity
Quad1/mav../setpoi../local
Quad1/mavros/state
Quad1/state_ma../state
Quad2/mav../globa../global
Quad2/state_ma../state
Quad3/mav../globa../global
Quad3/state_ma../state
Log Data
/Quad1/state_machine/state__Raw
Mission State
/Quad1/getColors/bgr/xyMeters__Raw
Distance to a detected object
/Quad1/gripper_status__Raw
Gripper State
/Quad1/getColors/bgr/processedImage__Image
Object Detection

```
1356998819.999s | Jan 01 2013 03:06:59.998 | 69.326s | > | 19.34 MB
```

```
▸ header
  state: Picking
  signal: Running
```

```
x: 0.041820
y: 0.055760
z: 14.502532
```

```
data: False
```

Figure 7.9: Log replay of a drone for a complete mission

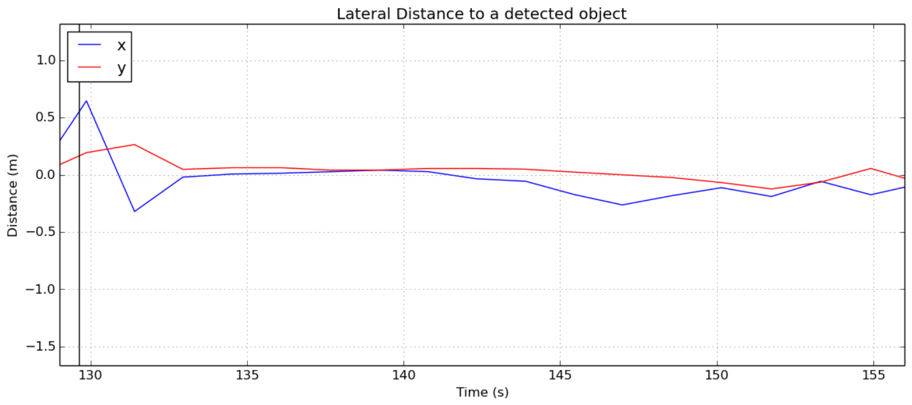


Figure 7.10: Decreasing distance error between drone position and detected object during picking phase

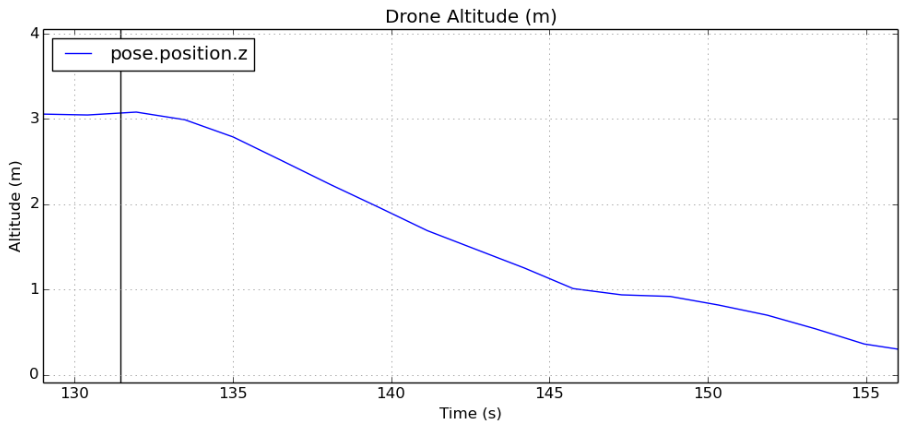


Figure 7.11: Drone's smooth altitude trajectory during pick up

effectiveness of the picking approach, which is one of the main contributions of this work.

Our experiments also showed the effectiveness of our proposed gripper mechanism over the electro-permanent magnets (EPM) solution. In particular, the proposed gripper mechanism outperformed EPM in gripping flat object when the gripper surface is not perfectly aligned with the object surface. A major issue we faced with EPM is that it required perfect alignment and touch with the object flat surface in order to be fully activated. For that, we installed EPMs on a large gimbal in order to stabilize them along with the camera. On the other hand, using the proposed gripper, perfect alignment is not required. Once the object is near the magnets active field, it is automatically pulled towards the magnets. Therefore, a gimbal for the gripper is not required.

Finally, we would like to highlight some of the challenges that were faced. For sake of the simplicity of the system, we used blob detection methods on low-computation modules for vision-based object detection. Such methods are usually tuned for particular colors at specific environmental effects e.g. light intensity. Therefore, it is challenging to use the same parameters to detect the same colors in different light conditions which we faced during field tests. More complex method can be used, but at the expense of more computation power. One possible solution is to use adaptive vision parameters (e.g. color thresholds) according to a trained model that accounts for environmental changes e.g. light intensity. The trained model can, then, be executed rapidly on low-computation modules.

7.2.3 Summary

In this work, we presented a fully integrated multi-agent UAV system for searching, collecting and transporting objects with unknown locations. The proposed system simplifies such complex tasks by introducing full autonomy which extends its application domains to real-life situations such as search and rescue missions and commercial package delivery. Objects were localized based on a monocular camera and the drone altitude, and picked up using our customized passive grasping mechanism with feedback. The overall system architecture was implemented and tested successfully in an outdoor environment using a simple yet effective approach with low-cost hardware, which makes it an appealing research test-bed. Further enhancements can be made in the design as well as the cooperative control techniques to incorporate robust performance of the system under varying environment conditions.

7.3 Distributed Motion Planing using Submodular Minimization

Submodular functions are discrete analog of convex functions. These functions are popular in combinatorial optimization because they can be efficiently minimized. Moreover, they have a lot of applications for instance in facility location, resource allocation, matching problems, and coverage problems etc. In [99], Jaleel and Shamma proposed a novel algorithm for the distributed minimization of submodular functions. They also showed that submodular functions can be used for distributed motion planning in multi-agent systems. To evaluate the real-time performance of this submodular minimization based distributed motion planning framework, I was involved in developing a realistic simulation setup. The problem setup was the same as attackers-defenders game used in this thesis. However, the defender team was solving a submodular minimization problem in a distributed manner. In this setup the neighboring agents were communicating with each other. However, even with the communication among neighboring agents, our simulations showed that the defenders were able to compute their control actions efficiently.

7.3.1 Contribution

As mentioned, my contributions in this work is in the development of a realistic simulation environment with integration into the Robot Operating System (ROS). This allows for seamless transfer to actual hardware system.

For a realistic simulation setup, we used Gazebo robot system simulator to simulate drone dynamics and sensor feedback. The drones that we used in the simulations are a realistic model of the *IRIS+* quadcopter. The stabilization and the low level control of the drones was handled by *PX4* open-source autopilot, which is used in actual quadcopter control. In the simulation setup, the autopilot received simulated information of sensors, which included GPS, accelerometers, gyroscopes, pressure and altitude, from Gazebo. The autopilot software processed the simulated feedback data form the sensors and computed actuator (motor) commands. The autopilot also received high-level commands (e.g position set-points) and was responsible to track them.

The building blocks of the simulation are as follows.

- **Algorithm**: The distributed subgradient based algorithm for computing the motion plan of the defense team was implemented in MATLAB. In the algorithm, we received the current positions of the attacker and defender drones as feedback at each decision time. Based on this feedback, the algorithm generated distributed decisions in the form of commanded position set-points for the autopilot to track. The update actions for attackers were also computed in MATLAB according to protocol described in the previous section.

- **Autopilot**: The autopilot that we used was the *PX4* autopilot, which is a well know open-source professional autopilot. In our simulation, the autopilot software was used for quadrotor stabilization and navigation. The autopilot received position set-points from the MATLAB node. In order to mimic a realistic scenario, each drone had an independent executable in which the *PX4* autopilot was operating.

- **Gazebo**: We used the robot simulator Gazebo to generate a realistic simulation of quadcopter dynamics. It provided the autopilot with simulated sensor information. Moreover, it received the actuator (motor) commands from the autopilot to control the quadrotor dynamics. The simulator communicated with each autopilot executable through a UDP communication.

- **ROS**: All software components were executed inside ROS which facilitated the software development and integration. It also allowed us to use the same exact

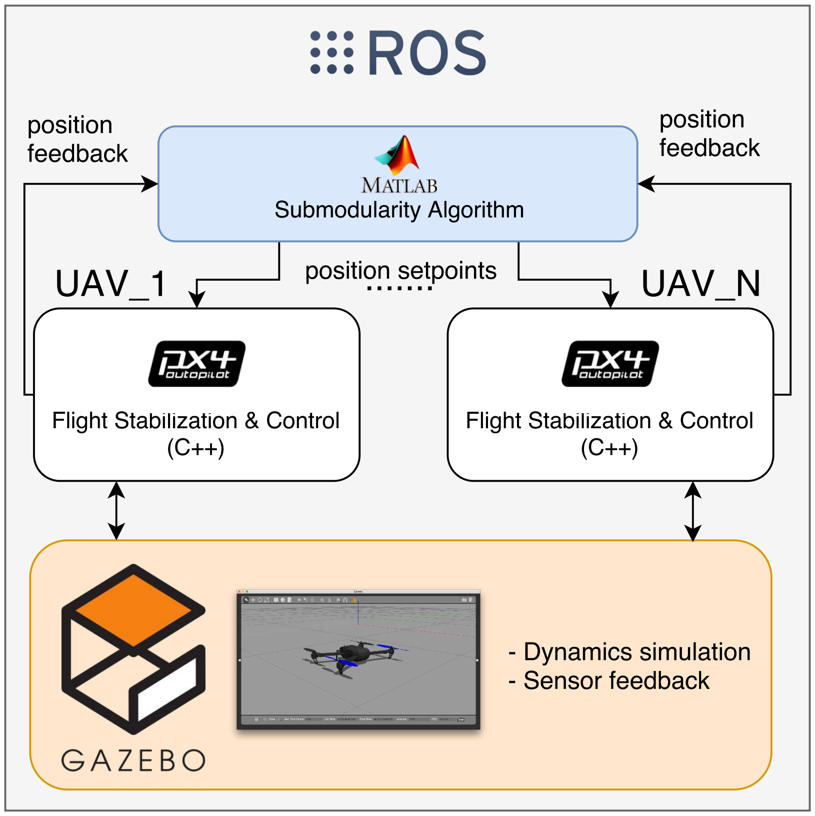


Figure 7.12: Architecture of the simulation framework. All software components run inside the ROS framework. Each drone has an identical autopilot executable. Gazebo simulator is used for dynamics simulation and sensor feedback.

code on actual hardware. This simulation framework is implemented as a ROS package.

The operational hierarchy and the interface between different components in the simulation setup are depicted in Fig. 7.12.

7.3.2 Results

The actual layout of the arena developed in Gazebo is shown in Fig. 7.13. In this simulation scene, the battle field is a square region of dimension 20×20 meter square. Each team has four players, i.e., $n_d = n_a = 4$. The defense zone is represented by short green poles on one side of the arena. The defense team comprises four black quadcopters deployed in front of the defense zone. The offense team comprises four red

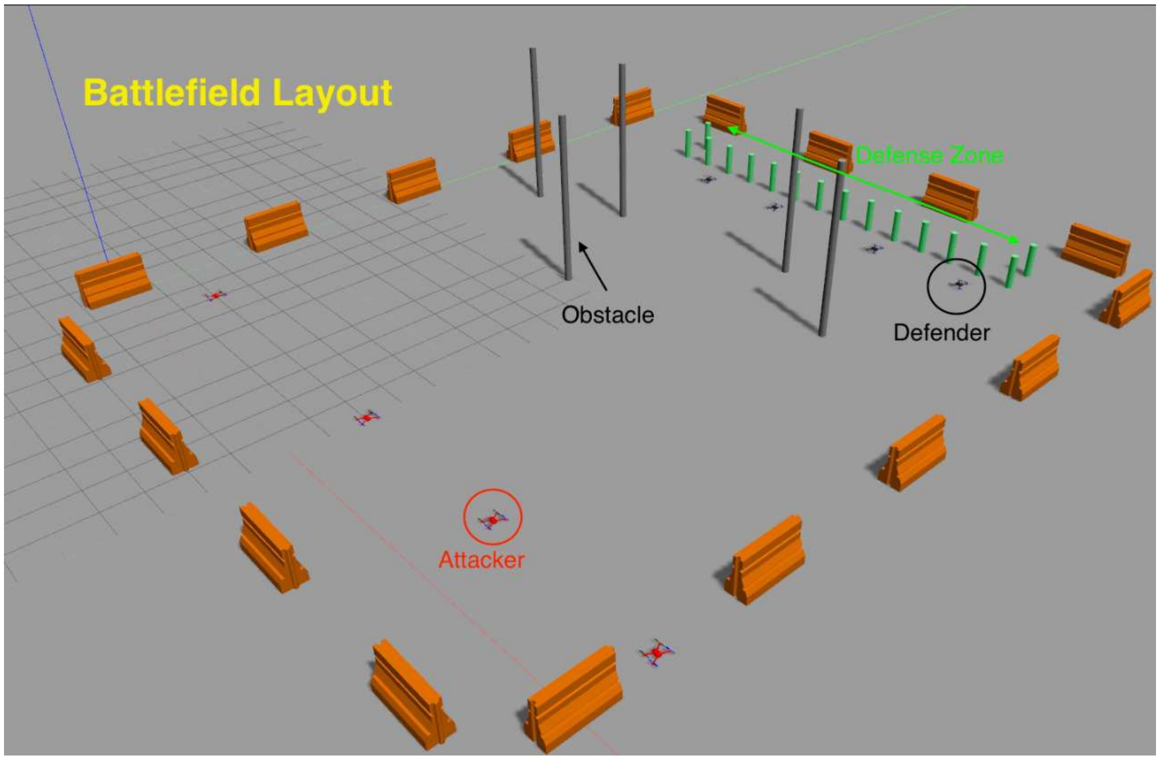


Figure 7.13: A layout of the battlefield developed in Gazebo. The small green poles at the top represent the defense zone. The tall grey poles in the middle correspond to the obstacles. The red quadcopters opposite to the defense zone are the attackers and the black qudcopters in front of the defense zone are defenders.

quadcopters deployed on the other side of the arena. Static obstacles are represented by long vertical columns that agents should avoid during their motion. The complete simulation details can be found in [99].

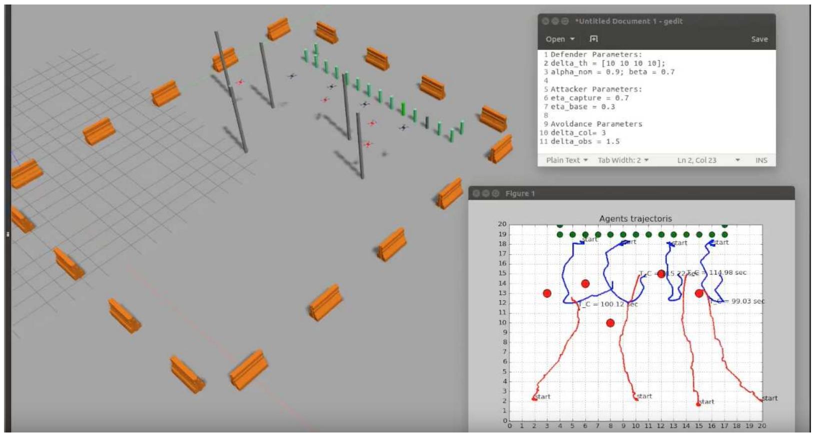
In the simulation setup, the duration of a game was $T = 75$ seconds. To avoid collisions among attackers and defenders, the z-coordinate of the defenders and attackers were set to 3 and 2 meters respectively. An attacker a_i was considered captured if

$$\min_{z^d \in P^d} \|z^d - z_i^a\| \leqslant 2 \text{ meter},$$

where P^d is the set of locations of all the defenders.

To run the simulation, the following sequence of steps were followed.

- All agents were commanded to takeoff and hover at their initial location, and

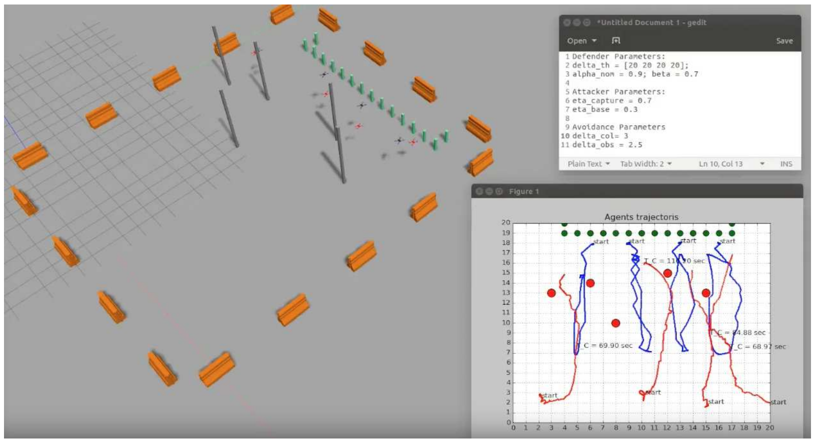


```
Defender Parameters:
delta_th = [10 10 10 10];
alpha_nom = 0.9; beta = 0.7

Attacker Parameters:
eta_capture = 0.7
eta_base = 0.3

Avoidance Parameters
delta_col= 3
delta_obs = 1.5
```

(a) Scenario1

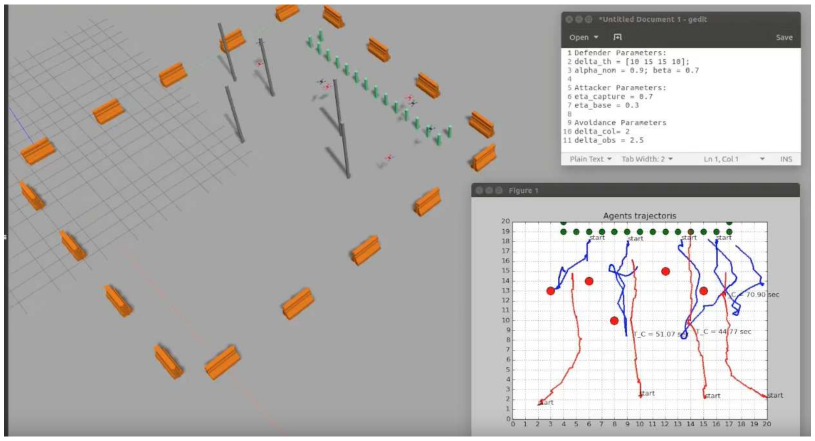


```
Defender Parameters:
delta_th = [20 20 20 20];
alpha_nom = 0.9; beta = 0.7

Attacker Parameters:
eta_capture = 0.7
eta_base = 0.3

Avoidance Parameters
delta_col= 3
delta_obs = 2.5
```

(b) Scenario 2



```
Defender Parameters:
delta_th = [10 15 15 10];
alpha_nom = 0.9; beta = 0.7

Attacker Parameters:
eta_capture = 0.7
eta_base = 0.3

Avoidance Parameters
delta_col= 2
delta_obs = 2.5
```

(c) Scenario 3

Figure 7.14: Snap shots of the simulation environment at the end of each game. In addition to the game arena, we present the design parameters for the offense and defense team in the top window and the trajectories followed by the attackers and defenders in the bottom window for each scenario.

wait for the battle signal

- Once the battle signal was triggered, the game started. During the game, each agent moved towards the commanded position computed by the distributed optimization algorithm executing in MATLAB.
- The game ended if one of the following conditions was satisfied
  - all attackers were captured at the same time,
  - maximum game time was reached

We simulated the game under three different scenarios. In the first scenario, the behavior of all the defenders was set to defensive mode. In the second scenario, the behavior of all the defenders was set to attack mode. In the last scenario, the behavior of two defenders in the middle was set to attack mode with and the two defenders on the side was set to defensive model. A snapshot for each game is shown in Fig. 7.14. For a detailed video of the games, we refer the readers to the YouTube link in [2]

As shown in the snapshots in Fig. 7.14, we superimposed the battlefield with two windows. In the small window at the top, we added all the parameters values for that particular simulation scenario. In the second window, we present the real-time trajectories of all the players. In the first scenario, all the defenders stayed at their initial locations till the attackers came close. Once the distance of the attackers from the defense zone was less than 10 meters, the defenders started pursing them. However, in the second scenario, the defenders started pursuing the attackers from the beginning of the game. In the third scenario, the two defenders in the middle started pursuing the attackers quickly since their behavior was set to attack. However, the defenders on the sides waited till the attackers came close to their area of responsibility.

[2]`https://youtu.be/kqPMLmbljjg`

7.3.3 Discussion

- We start the discussion by highlighting the fact that we developed a realistic simulation setup in which eight quadcopters were operating independently, controlled by actual autopilot software. Each of these quadcopters was receiving target set-point commands in real-time. These set-point commands were computed based on the real-time sensor feedback data, which was used in an online distributed optimization algorithm.

- From the videos of the capture the flag game under various scenarios, we can observe that we were able to compute feasible trajectories for the defense team from the distributed motion planning framework in an efficient manner. All the defenders were able to avoid collisions and obstacles effectively in all the scenarios. Moreover, they pursued the attackers by making real time decisions under an assumed attacker model. In terms of execution time, the MATLAB code was able to compute the target set points for all the attackers and defenders in approximately 0.43 seconds. Based on these results, we can conclude that the distributed motion planning framework presented in [**?**] is suitable for real-time applications of UAVs and UGVs.

- It is important to highlight that the actual motion plan of the attackers was significantly different from the model used by the defenders to make their decisions. The defenders assumed that the attackers were always proceeding towards the defense zone in a straight line. However, in the actual motion plan of the attackers, they were actively avoiding the defenders as well. The effect of this model mismatch can be observed from the simulations.

- Although the simulation setup was running on a high performance computing machine, we did face issues because of the large number of quadcopters. Having eight quadcopters as independent executables running *PX4* autopilot and communicating with Gazebo resulted in some delays in communication among different components of the simulation setup. The effects of delay in the feedback data, and the delay in the execution of commanded inputs can also be observed from the simulations.

- The distributed algorithm for computing the motion plan of defenders required local communication only. However, in the simulation setup, the algorithm was running on a centralized machine. A future direction of this work is to simulate and implement a completely distributed system in which the motion plan of each quadcopter is computed on its onboard computer based on real-time communications among neighboring agents in the network.

7.3.4 Conclusions

We developed a realistic simulation setup to verify the performance of a distributed motion planning framework with discrete inputs. This motion planning framework was based on distributed subgradient descent approach for submodular minimization. The simulation setup was developed in Gazebo interfaced with MATLAB where the high level scheduling was handled by ROS. They key feature of the simulation setup was that it was able to effectively manage eight quadcopters controlled by actual autopilot software. In this setup, we implemented capture the flag game with four quadcopters in each time. Based on the performance of the distributed motion planning framework in this realistic simulation setup, we were able to conclude that the framework is suitable for real-time applications.

7.4 Multi-Robot Onboard UWB Localization

Robots in a swarm take advantage of a motion capture system indoor or GPS sensors outdoor to obtain their global position. Motion capture systems are environment-dependent and GPS sensors are not reliable in occluded environments. Figure 7.15 shows the conventional fixed anchor configuration vs. the mobile anchor configuration which is desired in this work. Robots have to sense and interact locally in a swarm for a reliable and versatile operation. In this work, we consider the on board localization problem in multi-robot systems. A drone localizes another ground robot or aerial robot by using its on-board sensing and computational capabilities together with a set of ultrawideband sensors. The drone generates good enough relative position estimate by using the motion model of the localized robot in the filtering process. We validate our algorithm with outdoor experiments on a set of ground and aerial robots.

In this work, we use the onboard sensor configuration on a moving drone to localize another moving robot by the onboard computational unit of the drone. Specifically, we modify the localization algorithm of [100] where the anchor vehicle's model was assumed to be non-holonomic. Here, we assume that the drone's attitude dynamics is controlled by the low-level controller and represent the drone motion as a holonomic kinematics. The anchors are placed in a specific configuration on the drone. This configuration forms a virtual localization frame on the moving drone. In this configuration, the anchor separations are allowed to be very short (less than half a meter) compared to the fixed localization infrastructures (usually more than two meters).

We test the algorithm in two experimental scenarios. In the first scenario, the moving anchor drone estimates a moving ground robot's position and follows it at a desired relative position. In the second scenario, the moving anchor drone estimates another moving drone's position and follows it at a desired relative position. We show that the position error is within acceptable range with the proposed algorithm. We

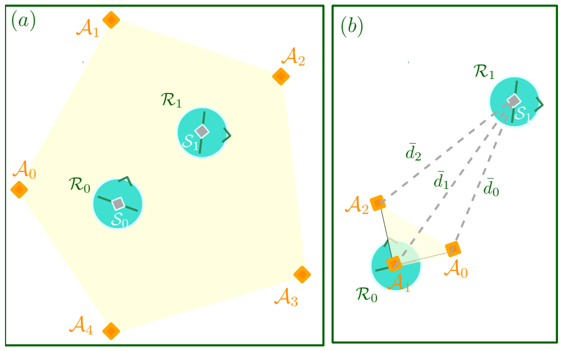


Figure 7.15: Illustration of two sensor network configurations. Orange and gray diamonds represent the anchors and sensors, respectively. *(a) Conventional anchor configuration*: The sensors are well separated from each other and the mobile vehicle mostly lies inside the convex hull of the sensor network (yellow shaded area); *(b) The onboard-anchor configuration*: The sensors are located very close to each other and the mobile vehicle always lies outside of the convex hull of the sensor network.

demonstrate that although the anchors are placed very close to each other, we obtain an acceptable localization performance by using the motion model and a rough initial position estimate of the localized robot. We employ only on-board IMU, laser range finder, and optical flow sensor to measure and control motion of the drone. Therefore, our algorithm is independent of the environment or fixed infrastructures. Moreover, we demonstrate with experiments that the motion control loop of the anchor drone can be closed with the localization estimate feedback.

7.4.1 Contributions

My contribution in this work was to develop and test the drone system that is capable of carrying the required components and provide hover stability and motion estimation without relying on GPS.

Two custom drones were developed and can be seen in Figures 7.16 and 7.17.

We performed experiments with two setups. In the first setup, we used a hexacopter drone that carries three UWB anchors and a ground robot that carries a single UWB sensor. In the second setup, we used two drones: a hexacopter that carries three UWB anchors and a quadrotor that caries a single UWB sensor. In both se-

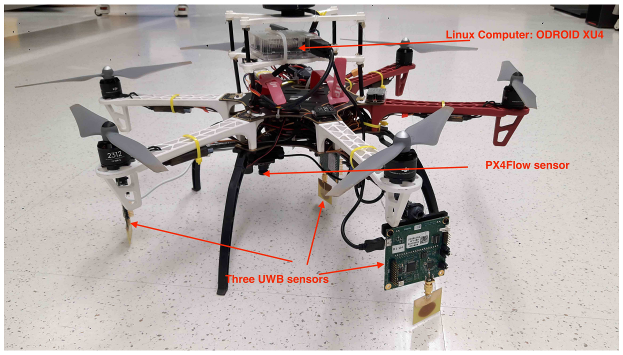


Figure 7.16: Hexacopter setup

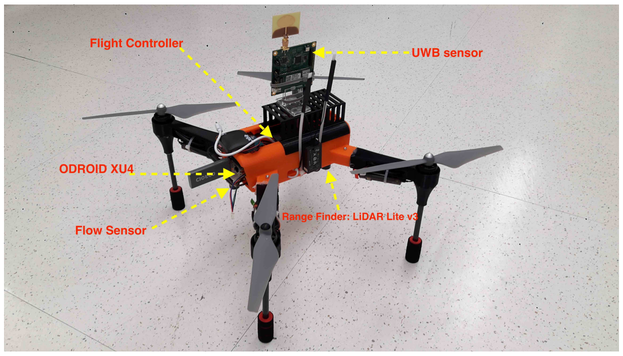


Figure 7.17: Quadrotor setup

tups, the hexacopter was to estimate the relative position with respect to the other robot (ground robot or quadrotor). Apart from the UWB sensors, both drones were equipped with similar components. In particular, they were equipped with a range finder for precision altitude control, camera-based flow sensor for hovering and velocity estimation, a flight controller for drone low-level control, and a low-power Linux computer for the localization and filtering computations. The detailed description of the on-board components is shown in Table 7.2. Figure 7.16 and 7.17 show the hexacopter and quadrotor used in the experiments. The hexacopter airframe was an of-the-shelf airframe while the quadrotor was a custom 3D printed design.

Each drone used the Pixhawk flight controller which runs the PX4 open-source autopilot firmware to provide attitude stability and velocity tracking. The flight controller used the PX4Flow sensor [101] to provide accurate velocity feedback and hovering. A PID velocity controller on the flight controller was used to track the

Table 7.2: Drone Components

Component	**Description**
Airframe	Hexacopter of diameter 50 cm. Custom 3D printed quadrotor frame of diamter 35 cm
Flight Controller	Pixhawk running PX4 autopilot firmware
Range Finder	LiDAR Lite v3 for precision altitude measurements
PX4FLOW	A camera flow sensor for hover positioning and velocity estimation
Onboard Computer	ODROID XU4 running Uhuntu 16 and ROS Kinetic for high-level computations
UWB sensors	For localization

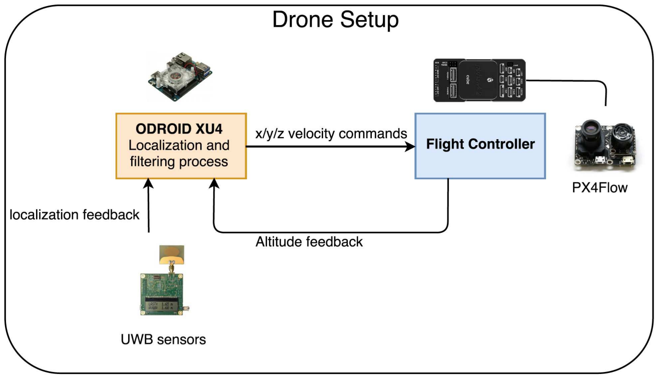


Figure 7.18: A diagram that shows the tasks executed on the drone.

commanded velocity by an onboard computer that was connected to the flight controller. We used ODROID XU4 as the onboard computer which provides sufficient computing power to execute the localization and filtering processes. The ODROID was used as the high-level controller that sends velocity commands to the Pixhawk flight controller based on localization feedback to follow the leader robot. Figure 7.18 shows a diagram of the tasks executed on the drone. All experiments were performed outdoor.

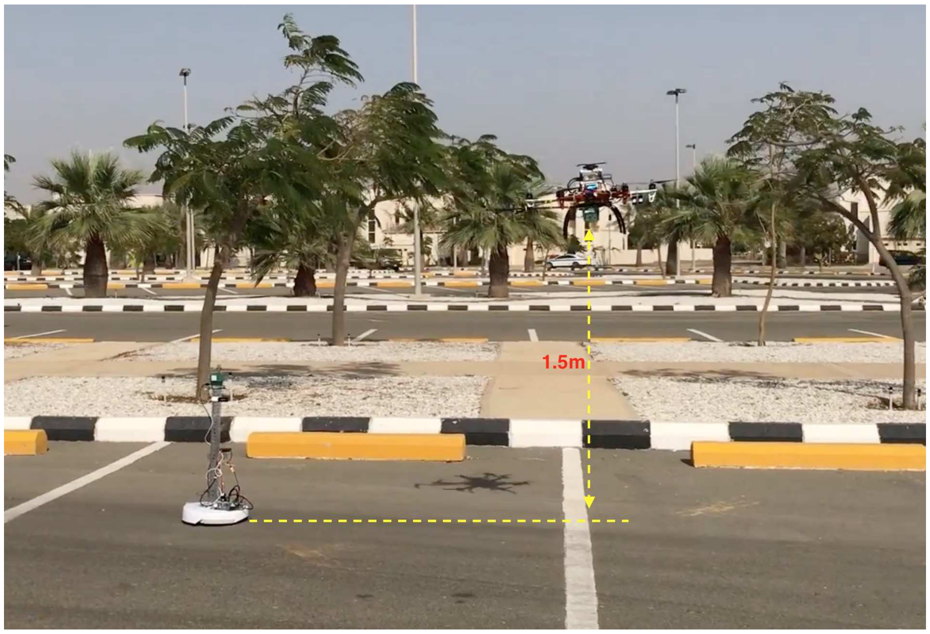


Figure 7.19: A snapshot of an outdoor trial with the ground robot moving with 0.2m/s forward velocity and the tracking drone at altitude 1.5m.

7.4.2 Results

We now present the outcome of four outdoor experiments. In the first set of experiments, we used the first experimental setup, that is, the hexacopter tracked the ground robot. In the second set of experiments, the hexacopter tracked the quadrotor. The experimental results are presented in the video[3] as well. All experiments were performed in windy weather outdoor.

[3]`https://drive.google.com/drive/folders/17nhsL-RjtpI36NKx_6UDOPRHFCNlw54v?usp=sharing`

7.4.2.1 Ground Robot Tracking by Hexacopter

In this scenario, the hexacopter's altitude was set to 1.5m. (Figure 7.19). The drone was to track the ground robot with the following control law:

$$\mathrm{v_A}[k] = [v_A^x, v_A^y]^\top, \tag{7.1}$$

$$v_A^x = \begin{cases} K_x e_x[k], & \text{if } e_x[k] > \epsilon \\ 0, & \text{otherwise,} \end{cases}$$

$$v_A^y = \begin{cases} K_y e_y[k], & \text{if } e_y[k] > \epsilon \\ 0, & \text{otherwise,} \end{cases}$$

where K_x, K_y are negative design constants, ϵ is a threshold to avoid chattering, and $\mathrm{e_x}[k] = \mathrm{r_x^{des}} - \mathrm{r_x}[k]$. The maximum velocity of the drone was set to 0.4m/s in x and y axes.

In the first trial, the ground robot was commanded to move with a constant speed $\mathrm{v_G}[k] = [0, 0.1]^\top$m/sec. The initial relative position estimate was set as

$$\hat{\mathrm{r}}[0] = [1.0,\ 1.0]^\top \text{ m}.$$

We present the estimated relative position and the commanded velocity of the drone in Fig. 7.20. The algorithm was able to estimate the relative position with a good enough performance and the vehicle successfully followed the ground robot within an acceptable error bound.

In the second trial, a non-holonomic ground robot was commanded to move on a circular trajectory with the following linear and angular speeds:

$$\mathrm{u_G}[k] = [v,\ \omega]^\top = [0.3, 0.05]^\top \text{m/sec}.$$

The hexacopter was commanded to track the ground robot with the control law 7.1,

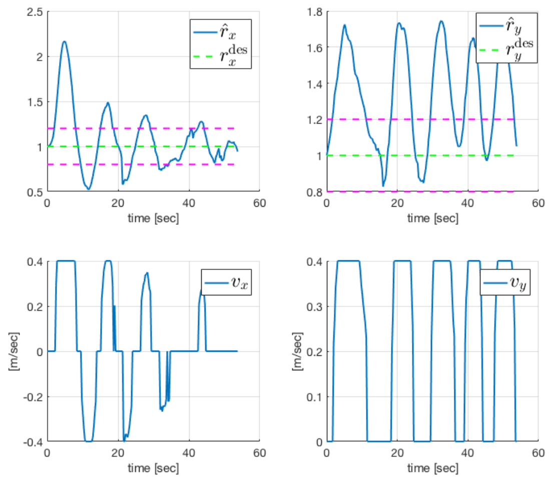


Figure 7.20: Experiment scenario 1, the drone tracks the ground robot: The relative positions $\mathrm{r_x}, \mathrm{r_y}$ (top) and the corresponding commanded velocity of the drone (bottom). The dashed lines represent the bands within the threshold ϵ.

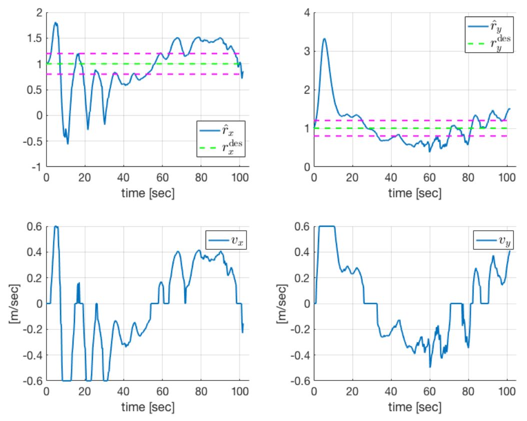


Figure 7.21: Experiment scenario 2, the drone tracks the ground robot moving on a circular trajectory: The relative positions $\mathrm{r_x}, \mathrm{r_y}$ (top) and the corresponding commanded velocity of the drone (bottom). The dashed lines represent the bands within the threshold ϵ.

with maximum velocity increased to 0.6m/sec. Figure 7.21 shows the of relative position estimates of the ground robot with respect to the hexacopter.

7.4.2.2 Quadrotor Tracking by Hexacopter

In this set of experiments, the hexacopter tracked the quadrotor based on the localization feedback. The UWB sensor was mounted on top of the quadrotor. To satisfy the LOS condition between the anchors and the sensor, the hexacopter's and quadrotor's

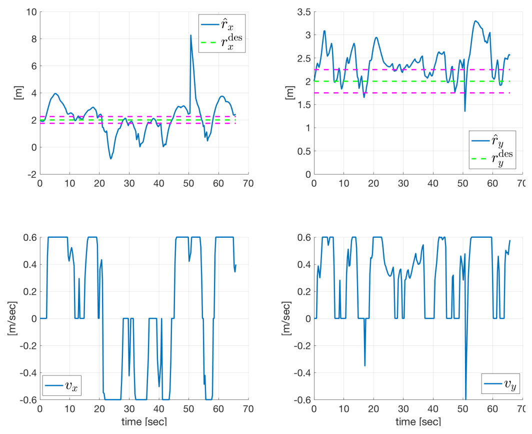


Figure 7.22: Experiment scenario 3, the hexacopter tracks the quadrotor: The relative positions $\mathrm{r_x}, \mathrm{r_y}$ (top) and the corresponding commanded velocity of the drone (bottom). The dashed lines represent the bands within the threshold ϵ.

altitudes were set to 2m. and 1m., respectively.

In the first trial, the quadrotor performed a zig-zag motion with the commanded velocity $\mathrm{v_G}[k] = [0.3a,\ 0.3]^\top$m/sec., where $a = \{-1, 1\}$. The hexacopter used the control law (7.1) with the maximum speed of 0.6m/sec. in both axes. We demonstrate the relative position r and the hexacopter's commanded velocities in Fig. 7.22. We note that although the quadrotor's motion was not smooth as in the case of ground robot tracking, the hexacopter showed an acceptable tracking performance.

Finally, we performed a sense-and-avoid experiment with the hexacopter and quadrotor. In this trial, the drones were located 6m. apart and they started to move to each other on a straight line. The quadrotor's forward direction headed the hexacopter and it was commanded to move with the velocity $\mathrm{v_G}[k] = [0,\ 0.2]^\top$m/sec. The hexacopter used the following repulsive control law:

$$\begin{aligned} \mathrm{v_A}[k] &= [v_A^x, v_A^y]^\top, \qquad (7.2) \\ &= \begin{cases} [0,\ 0.2]^\top, & \text{if } \|\hat{\mathrm{r}}[k]\| > d_{\mathrm{safe}} \\ [0.4\mathrm{sgn}(\mathrm{r_x}[k]),\ 0.4\mathrm{sgn}(\mathrm{r_y}[k])]^\top, & \text{otherwise,} \end{cases} \end{aligned}$$

where $d_{\mathrm{safe}} = 2$m. is the safe distance between the drones and sgn is the signum

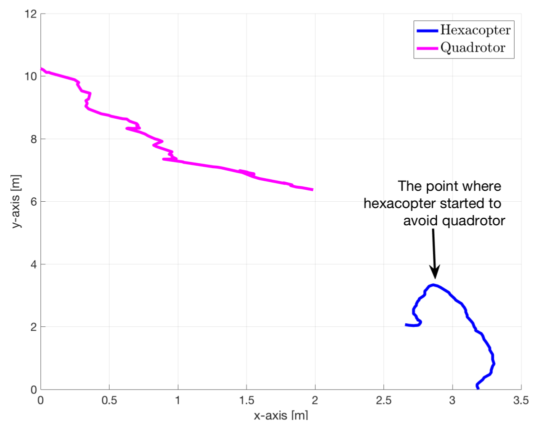


Figure 7.23: Experiment scenario 4, the hexacopter avoids the quadrotor once the distance between the drones goes below a safe threshold. The hexacopter starts to move away from the quadrotor.

function. We present the GPS data of both drones in Fig. 7.23. The quadrotor started motion from the upper-left corner and moved toward the hexacopter. The point where hexacopter started to avoid quadrotor is clearly seen in this figure. We emphasize that this sense-and-avoid behavior of the hexacopter was completely achieved by the on-board localization outputs.

7.4.3 Discussion on Experimental Results

In the first scenario, where the drone tracked the ground robot moving in a linear trajectory, the relative distance exceeded at most 2 meters away from the desired bounds in the x direction, and 0.7 meter in the y direction. This is an acceptable performance observing that the drone was standing against 10 m/s wind speed which contributed to oscillations seen in both x and y axes, see Figure 7.20. In the second scenario, the drone tracked the ground robot which executed a circular trajectory. The drone maximum speed was increased by 50% of the speed used in the first scenario in order to help the drone to better withstand the wind while tracking the ground robot. The relative distance in the x direction exceeded at most 1.5 meters from the desired bounds and 0.5 meter in the overall trajectory in the y direction. This shows an improvement over the first scenario where the drone had less maximum speed.

We saturated the hexacopter's maximum speed because the Px4Flow sensor was not able to track the hexacopter's motion for higher speed values. In fact, the quadrotor performed similar to the bang-bang control in most of the trials because the wind highly affected the motion. We argue that the quadrotor would show a smoother tracking performance in a steady weather.

We used the EKF parameters that gave the best observed performance. Especially, we used higher values in the measurement covariance matrix compared to the process covariances because the UWB distance values were not reliable due to the aggressive motion characteristics of the drones. We suggest tuning of the EKF parameters for specific anchor drone properties and weather conditions.

We considered only LOS case of the UWB sensors in this paper. However, there are cases where the anchor-sensor lines can be occluded with obstacles or by the vehicles. The NLOS characteristics of the UWB sensors need to be extracted for those cases. In addition, we obtained UWB distance measurements and generated velocity set-points with around 4 Hertz. In most cases, this frequency is not sufficient to track vehicles with aggressive maneuvers. We argue that a better UWB sensing mechanism would result in better tracking performance.

7.4.4 Conclusion

We have addressed the on-board localization problem in multi-robot systems. We have used an onboard anchor configuration of ultrawideband sensors on a drone to localize a moving ground or aerial vehicle by the drone. We have included the motion model of the localized vehicle in the extended Kalman filter algorithm and obtained reasonable estimation performance. The anchor drone used only on-board IMU and optical flow sensors and was able to maintain a fixed relative position to the localized ground robot and drone with the localization feedback in the experiments. The algorithm was also able to catch the slightly varying speed dynamics of the localized robot.

The proposed localization approach can be improved in many ways. We plan to apply different filtering methods and analyze the effects of the anchor separation distance to the estimation performance. Furthermore, we plan to extend our algorithm to allow coordination of more than two robots.

Chapter 8

Concluding Remarks

8.1 Summary

In this thesis, a distributed algorithm is proposed which tries to address the real-time requirement of fast decision making problem while providing meaningful decision that can be suboptimal. The approach is presented in the context of an adversarial game setting between two teams of robots of conflicting objectives, an attacking team and a defending team. We first discussed the centralized setting of the problem of interest and formulated as a dynamic program which provides the optimal policy for the defenders to minimize the time before an attacker infiltrates the defense zone. The main challenge of such approaches is the curse of dimensionality which makes the problem computationally prohibitive. An alternative centralized linear program formulation is discussed which provided a fast and online heuristic solution. Extending on the centralized LP approach, a local linear program algorithm that can be implemented in a distributed setting is proposed that allows agents to take local decisions based on simple local information of their neighbors (i.e. position information). The algorithm does not require iterative information exchange before taking a decision. Required information exchange happens only once and then local actions are generated. This approach exploits the common knowledge of the objective structure and the assumed agents motion models which are fused in the local optimization to provide meaningful distributed behavior in the absence of global information. The linear program was shown to be solved efficiently fast to provide a real-time implementation.

The considered environment in this work is a 2D grid. However, it can be generalized to 3D grid, but at the cost of more computation burden which affects the real-time performance. In the proposed work, the robots' motion model is assumed to be linear over the considered grid environment. We have seen this as an adequate assumption in the considered application as the algorithm operates on the stabilized closed loop of the lower-level system. In other words, we assume that the low-level control stabilizes the robots' dynamics and can track higher-level commands from the LP algorithm. If aggressive motion is considered, it might be required to include the higher order low-level dynamics of the robots which breaks the linear program formulation, hence affecting the real-time execution speed.

Also, another contribution of this work is real-time realistic simulation and indoor/outdoor hardware implementation using drones which are used to validate the approach performance in realistic game settings. It was shown that the framework was able to execute at 30Hz at a grid of size 7×7 with prediction horizon of 3 steps in time, and at 10Hz for grid of 8×8 with a prediction horizon of 3 steps in time, using only onboard low-power computation modules. This makes the framework attractive for swarms of small aerial robots that are constrained in payload and power consumption.

A bound on the solution of the proposed local LP algorithm is challenging to find. An alternative approach where an analytical bound between the distributed and centralized problems can be obtained is also proposed. The idea is to find a reasonable approximation of the centralized cost function where it can be less complicated to compute. Then, using conventional distributed optimization algorithms to solve the approximated centralized problem in a distributed setting.

Finally, I believe that reproducibility of research work is key factor for successful and rapid future improvements. For that matter, the software of the implementation is provided as an open-source package with ROS integration which allows to reproduce the results shown in this work.

8.2 Future Research Work

The work presented in this thesis can be extended in the following directions. First, it is challenging to find analytical bounds between the centralized and distributed setups specially when information exchange about the system state is limited between agent. This is called the suboptimality. An interesting problem is to be able to quantify the trade-off between optimality and communication bandwidth, or the amount of exchanged information. This, for example, can allow agents to decide when to communicate in order to not to exceed certain acceptable degradation level. This is a challenging problem that are now problem specific, i.e. it is not restricted for the attackers-defenders game.

Another specific challenge that is specific to the considered game setup is the computation time as a function of the grid size. Such computation time increases exponentially as the grid size increases. This may impose a real-time challenge on the LP approach. One suggestion that requires investigation is to choose a grid size that can be computed in real-time (definition of real-time is an application-dependent), even if the actual grid size is much larger. This acts as a local grid that moves with the agents. The information of the remaining part of the bigger grid could then be projected on the local one in order not to completely loose the overall perspective. The solution provided by the this approach need to be compared to the one when the overall grid is considered in the computation.

One more interesting direction is related to environmental learning. For example, in the attackers-defenders game, defenders were computing online decision against changes in the environment. The attackers team is considered as part of this changing environment. in the LP approach, agents used an assumed model of their opponents in order to help in the receding horizon planning (trajectory prediction). In general, this model can be in disagreement with the actual model the opponents use. One suggestion is to use a hybrid prediction approach where a set of assumed models such that agents can switch between different models according to certain measures with respect the environment changes (including the opponents behaviors) during the game. Another suggestion is to start with a nominal assumed model and tune it online based on certain measurements of environment. Reinforcement learning techniques provide a general framework of learning optimal policies by experience when prediction models are not available. However, the number of trials is generally large and might not be suitable for the considered application. In this case, an alternative approach is required where learning can converge fast (definition of fast is application-dependent).

REFERENCES

[1] T. Nieberg, “Distributed algorithms in wireless sensor networks,” *Electronic Notes in Discrete Mathematics*, vol. 13, pp. 81 – 83, 2003, 2nd Cologne-Twente Workshop on Graphs and Combinatorial Optimization. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1571065304004445

[2] N. Tziritas, T. Loukopoulos, S. Lalis, and P. Lampsas, “Algorithms for energy-driven agent placement in wireless embedded systems with memory constraints,” *Simulation Modelling Practice and Theory*, vol. 19, no. 6, pp. 1445 – 1464, 2011, performance of Networks and Ubiquitous Computing Systems: Techniques and Trends. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X10002443

[3] N. Al-Nakhala, R. Riley, and T. Elfouly, “Distributed algorithms in wireless sensor networks: An approach for applying binary consensus in a real testbed,” *Computer Networks*, vol. 79, pp. 30 – 38, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128614004654

[4] S. Kar, J. M. F. Moura, and K. Ramanan, “Distributed parameter estimation in sensor networks: Nonlinear observation models and imperfect communication,” *IEEE Transactions on Information Theory*, vol. 58, no. 6, pp. 3575–3605, June 2012.

[5] H. Wang, X. Liao, Z. Wang, T. Huang, and G. Chen, “Distributed parameter estimation in unreliable sensor networks via broadcast gossip algorithms,” *Neural Networks*, vol. 73, pp. 1 – 9, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608015001835

[6] N. Michael, M. M. Zavlanos, V. Kumar, and G. J. Pappas, “Distributed multi-robot task assignment and formation control,” in *2008 IEEE International Conference on Robotics and Automation*, May 2008, pp. 128–133.

[7] A. Tagliabue, M. Kamel, S. Verling, R. Siegwart, and J. Nieto, "Collaborative transportation using mavs via passive force control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5766–5773.

[8] R. D'Andrea and M. Babish, "The roboflag testbed," in *American Control Conference, 2003. Proceedings of the 2003*, vol. 1. IEEE, 2003, pp. 656–660.

[9] M. G. Earl and R. D'Andrea, *Multi-vehicle cooperative control using mixed integer linear programming.* John Wiley & Sons, Ltd, 2007, pp. 231–259. [Online]. Available: http://dx.doi.org/10.1002/9780470724200.ch10

[10] G. C. Chasparis and J. S. Shamma, "Lp-based multi-vehicle path planning with adversaries," *Cooperative Control of Distributed Multi-Agent Systems*, pp. 261–279, 2008.

[11] H. Huang, J. Ding, W. Zhang, and C. J. Tomlin, "Automation-assisted capture-the-flag: A differential game approach," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 3, pp. 1014–1028, 2015.

[12] M. Flint, M. Polycarpou, and E. Fernandez-Gaucherand, "Cooperative path-planning for autonomous vehicles using dynamic programming," in *Proceedings of the IFAC 15th Triennial World Congress*, 2002, pp. 1694–1699.

[13] W. B. Dunbar and R. M. Murray, "Model predictive control of coordinated multi-vehicle formations," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 4. IEEE, 2002, pp. 4631–4636.

[14] G. Ferrari-Trecate, L. Galbusera, M. P. E. Marciandi, and R. Scattolini, "Model predictive control schemes for consensus in multi-agent systems with single-and double-integrator dynamics," *IEEE Transactions on Automatic Control*, vol. 54, no. 11, pp. 2560–2572, 2009.

[15] M. A. Müller, M. Reble, and F. Allgöwer, "Cooperative control of dynamically decoupled systems via distributed model predictive control," *International Journal of Robust and Nonlinear Control*, vol. 22, no. 12, pp. 1376–1397, 2012.

[16] D. P. Bertsekas, *Dynamic Programming and Optimal Control.* Athena Scientific, 2005, vol. Volume I. [Online]. Available: http://www.athenasc.com/dpbook.html

[17] R. E. Bellman, *Dynamic Programming.* Princeton Univ. Press, 1957. [Online]. Available: https://press.princeton.edu/titles/9234.html

[18] D. P. de Farias and B. V. Roy, "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850–865, 2003. [Online]. Available: https://doi.org/10.1287/opre.51.6.850.24925

[19] R. Cogill, M. Rotkowitz, B. Van Roy, and S. Lall, *An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 243–256. [Online]. Available: https://doi.org/10.1007/11664550_13

[20] H. Lakshmanan and D. P. de Farias, "Decentralized approximate dynamic programming for dynamic networks of agents," in *2006 American Control Conference*, June 2006, pp. 6 pp.–.

[21] D. P. de Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Oper. Res.*, vol. 51, no. 6, pp. 850–865, Nov. 2003. [Online]. Available: http://dx.doi.org/10.1287/opre.51.6.850.24925

[22] P. J. Schweitzer and A. Seidmann, "Generalized polynomial approximations in markovian decision processes," *Journal of Mathematical Analysis and Applications*, vol. 110, no. 2, pp. 568 – 582, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022247X85903178

[23] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *In Proceedings of the Tenth International Conference on Machine Learning.* Morgan Kaufmann, 1993, pp. 330–337.

[24] M. N. Ahmadabadi and M. Asadpour, "Expertness based cooperative q-learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 1, pp. 66–76, Feb 2002.

[25] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, Jan 2009.

[26] D. Bertsekas, A. Nedić, and A. Ozdaglar, *Convex Analysis and Optimization*, ser. Athena Scientific optimization and computation series. Athena Scientific, 2003. [Online]. Available: https://books.google.com.sa/books?id=DaOFQgAACAAJ

[27] S. Boyd, *Distributed Optimization and Statistical Learning Via the Alternating Direction Method of Multipliers*, ser. Foundations and Trends(r) in Machine Learning. Now Publishers, 2011. [Online]. Available: https://books.google.com.sa/books?id=8MjgLpJ0_4YC

[28] A. Nedic, "Asynchronous broadcast-based convex optimization over a network," *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1337–1351, June 2011.

[29] A. Nedic, A. Ozdaglar, and P. A. Parrilo, "Constrained consensus and optimization in multi-agent networks," *IEEE Transactions on Automatic Control*, vol. 55, no. 4, pp. 922–938, April 2010.

[30] B. Johansson, T. Keviczky, M. Johansson, and K. H. Johansson, "Subgradient methods and consensus algorithms for solving convex optimization problems," in *2008 47th IEEE Conference on Decision and Control*, Dec 2008, pp. 4185–4190.

[31] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[32] D. Jakoveti, J. M. F. Moura, and J. Xavier, "Linear convergence rate of a class of distributed augmented lagrangian algorithms," *IEEE Transactions on Automatic Control*, vol. 60, no. 4, pp. 922–936, April 2015.

[33] B. Johansson, M. Rabi, and M. Johansson, "A randomized incremental subgradient method for distributed optimization in networked systems," *SIAM*

J. on Optimization, vol. 20, no. 3, pp. 1157–1170, Aug. 2009. [Online]. Available: http://dx.doi.org/10.1137/08073038X

[34] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the admm in decentralized consensus optimization," *IEEE Transactions on Signal Processing*, vol. 62, no. 7, pp. 1750–1761, April 2014.

[35] E. Wei and A. Ozdaglar, "Distributed alternating direction method of multipliers," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, Dec 2012, pp. 5445–5450.

[36] I. Lobel and A. Ozdaglar, "Distributed subgradient methods for convex optimization over random networks," *IEEE Transactions on Automatic Control*, vol. 56, no. 6, pp. 1291–1306, June 2011.

[37] I. Matei and J. S. Baras, "Performance evaluation of the consensus-based distributed subgradient method under random communication topologies," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 4, pp. 754–771, Aug 2011.

[38] I. Lobel, A. Ozdaglar, and D. Feijer, "Distributed multi-agent optimization with state-dependent communication," *Mathematical Programming*, vol. 129, no. 2, pp. 255–284, Oct 2011. [Online]. Available: https://doi.org/10.1007/s10107-011-0467-x

[39] S. Sundhar Ram, A. Nedić, and V. V. Veeravalli, "Distributed stochastic subgradient projection algorithms for convex optimization," *Journal of Optimization Theory and Applications*, vol. 147, no. 3, pp. 516–545, Dec 2010. [Online]. Available: https://doi.org/10.1007/s10957-010-9737-7

[40] H. Terelius, U. Topcu, and R. M. Murray, "Decentralized multi-agent optimization via dual decomposition," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 11 245 – 11 251, 2011, 18th IFAC World Congress. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016454212

[41] I. Matei and J. S. Baras, "Distributed algorithms for optimization problems with equality constraints," in *52nd IEEE Conference on Decision and Control*, Dec 2013, pp. 2352–2357.

[42] I. Matei, J. S. Baras, M. Nabi, and T. Kurtoglu, "An extension of the method of multipliers for distributed nonlinear programming," in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 6951–6956.

[43] M. Zhu and S. Martnez, "An approximate dual subgradient algorithm for multi-agent non-convex optimization," in *49th IEEE Conference on Decision and Control (CDC)*, Dec 2010, pp. 7487–7492.

[44] W. H. Kwon and S. Han, *Receding horizon control: model predictive control for state models.* Springer, 2005.

[45] C. Bordons and E. F. Camacho, *Model Predictive Control.* Springer Verlag London Ltd, 2007.

[46] J. B. Rawlings and D. Q. Mayne, *Model predictive control: theory and design.* Nob Hill Pub., 2009.

[47] M. D. Doan, T. Keviczky, and B. D. Schutter, *A Hierarchical MPC Approach with Guaranteed Feasibility for Dynamically Coupled Linear Systems.* Dordrecht: Springer Netherlands, 2014, pp. 393–406. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_24

[48] A. Ferramosca, *Cooperative MPC with Guaranteed Exponential Stability.* Dordrecht: Springer Netherlands, 2014, pp. 585–600. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_36

[49] I. Prodan, F. Stoican, S. Olaru, C. Stoica, and S.-I. Niculescu, *Mixed-Integer Programming Techniques in Distributed MPC Problems.* Dordrecht: Springer Netherlands, 2014, pp. 275–291. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_17

[50] A. Casavola, E. Garone, and F. Tedesco, *The Distributed Command Governor Approach in a Nutshell*. Dordrecht: Springer Netherlands, 2014, pp. 259–274. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_16

[51] R. Martí, D. Sarabia, and C. de Prada, *Price-driven Coordination for Distributed NMPC Using a Feedback Control Law*. Dordrecht: Springer Netherlands, 2014, pp. 73–88. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_4

[52] M. Y. Lamoudi, M. Alamir, and P. Béguery, *A Distributed-in-Time NMPC-Based Coordination Mechanism for Resource Sharing Problems*. Dordrecht: Springer Netherlands, 2014, pp. 147–162. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_9

[53] J. Liu, D. M. de la Peña, and P. D. Christofides, *Lyapunov-Based Distributed MPC Schemes: Sequential and Iterative Approaches*. Dordrecht: Springer Netherlands, 2014, pp. 479–494. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_30

[54] A. Grancharova and T. A. Johansen, *Distributed MPC of Interconnected Nonlinear Systems by Dynamic Dual Decomposition*. Dordrecht: Springer Netherlands, 2014, pp. 293–308. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_18

[55] E. Camponogara, *Distributed Optimization for MPC of Linear Dynamic Networks*. Dordrecht: Springer Netherlands, 2014, pp. 193–208. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_12

[56] I. Necoara, *Rate Analysis of Inexact Dual Fast Gradient Method for Distributed MPC*. Dordrecht: Springer Netherlands, 2014, pp. 163–178. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_10

[57] I. Jurado, D. E. Quevedo, K. H. Johansson, and A. Ahlén, *Cooperative Dynamic MPC for Networked Control Systems*. Dordrecht: Springer Netherlands, 2014, pp. 357–373. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_22

[58] Y. Hu and N. H. El-Farra, *Adaptive Quasi-Decentralized MPC of Networked Process Systems.* Dordrecht: Springer Netherlands, 2014, pp. 209–223. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_13

[59] M. A. Müller and F. Allgöwer, *Distributed MPC for Consensus and Synchronization.* Dordrecht: Springer Netherlands, 2014, pp. 89–100. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_5

[60] G. Betti, M. Farina, and R. Scattolini, *Distributed MPC: A Noncooperative Approach Based on Robustness Concepts.* Dordrecht: Springer Netherlands, 2014, pp. 421–435. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_26

[61] B. Biegel, J. Stoustrup, and P. Andersen, *Distributed MPC Via Dual Decomposition.* Dordrecht: Springer Netherlands, 2014, pp. 179–192. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_11

[62] R. Hermans, M. Lazar, and A. Jokić, *Distributed Lyapunov-Based MPC.* Dordrecht: Springer Netherlands, 2014, pp. 225–241. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_14

[63] F. Tedesco, D. M. Raimondo, and A. Casavola, *A Distributed Reference Management Scheme in Presence of Non-Convex Constraints: An MPC Based Approach.* Dordrecht: Springer Netherlands, 2014, pp. 243–257. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_15

[64] C. Ocampo-Martinez, V. Puig, J. M. Grosso, and S. Montes-de Oca, *Multi-layer Decentralized MPC of Large-scale Networked Systems.* Dordrecht: Springer Netherlands, 2014, pp. 495–515. [Online]. Available: https://doi.org/10.1007/978-94-007-7006-5_31

[65] *DISTRIBUTED MODEL PREDICTIVE CONTROL MADE EASY.* SPRINGER, 2016.

[66] R. D'Andrea and R. M. Murray, "The roboflag competition," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 1, June 2003, pp. 650–655 vol.1.

[67] T. W. McLain and R. W. Beard, “Cooperative path planning for timing-critical missions,” in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 1, June 2003, pp. 296–301 vol.1.

[68] P. R. Chandler and M. Pachter, “Research issues in autonomous control of tactical uavs,” in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No.98CH36207)*, vol. 1, Jun 1998, pp. 394–398 vol.1.

[69] P. R. Chandler, M. Pachter, and S. Rasmussen, “Uav cooperative control,” in *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, vol. 1, 2001, pp. 50–55 vol.1.

[70] A. Dogan, “Probabilistic approach in path planning for uavs,” in *Proceedings of the 2003 IEEE International Symposium on Intelligent Control*, Oct 2003, pp. 608–613.

[71] A. Bemporad and M. Morari, “Control of systems integrating logic, dynamics, and constraints,” *Automatica*, vol. 35, no. 3, pp. 407 – 427, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109898001782

[72] A. Richards, T. Schouwenaars, J. P. How, and E. Feron, “Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming,” *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 4, pp. 755–764, 2018/01/17 2002. [Online]. Available: https://doi.org/10.2514/2.4943

[73] A. Richards and J. P. How, “Aircraft trajectory planning with collision avoidance using mixed integer linear programming,” in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 3, May 2002, pp. 1936–1941 vol.3.

[74] T. Schouwenaars, B. D. Moor, E. Feron, and J. How, “Mixed integer programming for multi-vehicle path planning,” in *2001 European Control Conference (ECC)*, Sept 2001, pp. 2603–2608.

[75] M. G. Earl and R. D'Andrea, "A study in cooperative control: the roboflag drill," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 3, May 2002, pp. 1811–1812 vol.3.

[76] ——, "Modeling and control of a multi-agent system using mixed integer linear programming," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 1, Dec 2002, pp. 107–111 vol.1.

[77] F. Gentili and F. Martinelli, "Robot group formations: A dynamic programming approach for a shortest path computation," in *ICRA*, 2000.

[78] D. Bauso, L. Giarre, and R. Pesenti, "Multiple uav cooperative path planning via neuro-dynamic programming," in *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, vol. 1, Dec 2004, pp. 1087–1092 Vol.1.

[79] M. Flint, M. Polycarpou, and E. Fernandez-Gaucherand, "Cooperative control for multiple autonomous uav's searching for targets," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 3, Dec 2002, pp. 2823–2828 vol.3.

[80] S. D. Patek, D. A. Logan, and D. A. Castanon, "Approximate dynamic programming for the solution of multiplatform path planning problems," in *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, vol. 1, 1999, pp. 1061–1066 vol.1.

[81] S. Daniel-Berhe, M. Ait-Rami, J. S. Shamma, and J. L. Speyer, "Optimization based battle management," in *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, vol. 6, 2001, pp. 4711–4715 vol.6.

[82] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, 1st ed. Athena Scientific, 1997.

[83] M. Abdelkader, H. Jaleel, and J. S. Shamma, "A distributed framework for real time path planning in practical multi-agent systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10 626 – 10 631, 2017, 20th IFAC World

Congress. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896317315185

[84] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st ed. Athena Scientific, 1996.

[85] M. Abdelkader, Y. Lu, H. Jaleel, and J. S. Shamma, "Distributed real time control of multiple uavs in adversarial environment: Algorithm and flight testing results," in *International Conference on Robotics and Automation*, 2018, to appear.

[86] M. Abdelkader, "Dlp software package," 2018. [Online]. Available: https://github.com/mzahana/dlp

[87] A. Pierson, Z. Wang, and M. Schwager, "Intercepting rogue robots: An algorithm for capturing multiple evaders with multiple pursuers," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 530–537, April 2017.

[88] Z. Zhou, W. Zhang, J. Ding, H. Huang, D. M. Stipanovi, and C. J. Tomlin, "Cooperative pursuit with voronoi partitions," *Automatica*, vol. 72, pp. 64 – 72, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109816301911

[89] D. Mellinger, M. Shomin, N. Michael, and V. Kumar, "Cooperative grasping and transport using multiple quadrotors," in *Distributed autonomous robotic systems.* Springer, 2013, pp. 545–558.

[90] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," *Autonomous Robots*, vol. 30, no. 1, pp. 73–86, Jan 2011. [Online]. Available: https://doi.org/10.1007/s10514-010-9205-0

[91] C. C. Kessens, J. Thomas, J. P. Desai, and V. Kumar, "Versatile Aerial Grasping Using Self-Sealing Suction," in *IEEE International Conference on Robotics and Automation.* Stockholm: IEEE, 2016.

[92] E. W. Hawkes, H. Jiang, and M. R. Cutkosky, “Three-dimensional dynamic surface grasping with dry adhesion,” *The International Journal of Robotics Research*, p. 0278364915584645, 2015.

[93] P. E. Pounds, D. R. Bersak, and A. M. Dollar, “Practical aerial grasping of unstructured objects,” in *2011 IEEE Conference on Technologies for Practical Robot Applications*. IEEE, 2011, pp. 99–104.

[94] U. A. Fiaz, M. Abdelkader, and J. S. Shamma, “Practical aerial grasping with heavy duty passive magnetic gripper,” in *IEEE International Conference on Robotics and Automation*. Brisbane: IEEE, 2018.

[95] U. A. Fiaz, N. Toumi, and J. S. Shamma, “Passive aerial grasping of ferrous objects,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10 299–10 304, 2017.

[96] H. Durrant-Whyte, N. Roy, and P. Abbeel, *Construction of Cubic Structures with Quadrotor Teams*. MIT Press, 2012, pp. 177–184. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6301066

[97] F. Augugliaro, S. Lupashin, M. Hamer, C. Male, M. Hehn, M. W. Mueller, J. S. Willmann, F. Gramazio, M. Kohler, and R. D’Andrea, “The flight assembled architecture installation: Cooperative construction with flying machines,” *IEEE Control Systems*, vol. 34, no. 4, pp. 46–64, Aug 2014.

[98] MBZIRC, “Mohammed bin zayed international robotics challenge,” ”http://www.mbzirc.com/challenge”, 2017.

[99] H. Jaleel, M. Abdelkader, and J. S. Shamma, “Real-time distributed motion planning with submodular minimization,” in *IEEE Conference on Control Technology and Applications*. IEEE, 2018.

[100] S. Guler, J. Jiang, R. I. Masoud, A. A. Alghamdi, and J. S. Shamma, “Real-time onboard ultrawideband localization scheme for an autonomous two-robot system,” in *Proceedings 2018 IEEE Conference on Control Technology and Applications*, 2018 (submitted).

[101] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys, "An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 1736–1741.

APPENDICES

A Implementation Notes for the local LP Algorithm

This appendix presents some details related to the formulation of the local LP algorithm presented in Chapter 5 which is used in the implementations of the proposed framework.

Dynamics

Recall the dynamics of the grid state,

$$\mathbf{x}^{+} = \mathbf{x} + (\mathbf{B}_{\text{in}} - \mathbf{B}_{\text{out}})\mathbf{u} \tag{A.1}$$

Also recall the input matrix,

$$\mathbf{B} = \mathbf{B}_{\text{in}} - \mathbf{B}_{\text{out}} \in \mathcal{R}^{n_s \times n_u}$$

Where $\mathbf{u}$

$$\mathbf{u} = [\mathbf{u}_{s_1}^T \ \mathbf{u}_{s_2}^T \ldots \mathbf{u}_{s_{n_s}}^T]^T \in \mathcal{R}^{n_u}.$$

and $\mathbf{u}_{s_i}$ is all possible controls (transfers) from sector s_i to all other sectors in the grid.

$$\mathbf{u}_{s_i} = [u_{s_i \to s_1} \ldots u_{s_i \to s_{i-1}} u_{s_i \to s_i} u_{s_i \to s_{i+1}} \ldots u_{s_i \to s_{n_s}}]^T. \tag{A.2}$$

Notice that Eq. A.2 is different from the definition in 4.4. That is because $u_{s_i \to s_i}$ is added to the $\mathbf{u}_{s_i}$ vector. However, it is always multiplied by zero. It is added in order to make the implementation easier. Therefore, $n_u = n_s^2$ in this implementation.

A.0.0.0.1 Dynamics constraints : dynamics over prediction time horizon T_p,

$$\mathbf{X} = \mathbf{T}_{xu}\mathbf{U} + \mathbf{T}_{xx_0}\mathbf{x}[0] \tag{A.3}$$

where $\mathbf{T}_{xu}$ is,

$$\mathbf{T}_{xu} = \begin{bmatrix} \mathbf{B} & \cdots & \mathbf{0} & \\ \mathbf{B} & \mathbf{B} & \cdots & \mathbf{0} \\ \cdots & \vdots & \cdots & \\ \mathbf{B} & \mathbf{B} & \cdots & \mathbf{B} \end{bmatrix}_{(n_sT_p)\times(n_uT_p)}$$

and,

$$\mathbf{T}_{xx_0}\mathbf{x}[0] = \begin{bmatrix} \mathbf{x}[0] \\ \vdots \\ \mathbf{x}[0] \end{bmatrix}_{n_sT_p\times 1}$$

Assuming that the optimization vector is,

$$\hat{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \mathbf{U} \end{bmatrix}_{(n_s+n_u)T_p\times 1} \tag{A.4}$$

Eq A.3 can be written as,

$$\begin{bmatrix} \mathbf{I}_{n_sT_p} & -\mathbf{T}_{xu} \end{bmatrix} \hat{\mathbf{X}} = \mathbf{T}_{xx_0}\mathbf{x}[0] \tag{A.5}$$

It can be converted to inequalities as follows,

$$\begin{bmatrix} \mathbf{I}_{n_sT_p} & -\mathbf{T}_{xu} \\ -\mathbf{I}_{n_sT_p} & \mathbf{T}_{xu} \end{bmatrix}_{2n_sT_p\times(n_s+n_u)T_p} \hat{\mathbf{X}} \leqslant \begin{bmatrix} \mathbf{T}_{xx_0}\mathbf{x}[0] \\ -\mathbf{T}_{xx_0}\mathbf{x}[0] \end{bmatrix}_{2n_sT_p\times 1} \tag{A.6}$$

Which can be re-written as

$$\mathbf{A}_{\text{dynamics}}\hat{\mathbf{X}} \leqslant \mathbf{b}_{\text{dynamics}}$$

Flow Constraints

Flow constraints over prediction time horizon are described by,

$$\mathbf{T}_{xu,c}\mathbf{U} \leqslant \mathbf{T}_{xx_0,c}\mathbf{x}[0] \tag{A.7}$$

where,

$$\mathbf{T}_{xu,c} = \begin{bmatrix} \mathbf{B}_{\text{out}} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ -\mathbf{B} & \mathbf{B}_{\text{out}} & \mathbf{0} & \cdots & \mathbf{0} \\ -\mathbf{B} & -\mathbf{B} & \mathbf{B}_{\text{out}} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -\mathbf{B} & -\mathbf{B} & -\mathbf{B} & -\mathbf{B} & \mathbf{B}_{\text{out}} \end{bmatrix}$$

$\mathbf{T}_{xu,c} \in \mathcal{R}^{n_sT_p \times n_uT_p}$, and

$$\mathbf{T}_{xx_0,c}\mathbf{x}[0] = \begin{bmatrix} \mathbf{x}[0] \\ \vdots \\ \mathbf{x}[0] \end{bmatrix}_{n_sT_p \times 1}$$

Using the same optimization vector in (A.4), constraints (A.7) can be written as,

$$\begin{bmatrix} \mathbf{0}_{n_sT_p \times n_sT_p} & \mathbf{T}_{xu,c} \end{bmatrix}_{n_sT_p \times (n_s+n_u)T_p} \hat{\mathbf{X}} \leqslant \mathbf{T}_{xx_0,c}\mathbf{x}[0] \tag{A.8}$$

which can be written as,

$$\mathbf{A}_{\text{flow}}\hat{\mathbf{X}} \leqslant \mathbf{b}_{\text{flow}}$$

Boundary Constraints

Constraints on optimization vector $\hat{\mathbf{X}}$ are,

$$\mathbf{0} \leqslant \hat{\mathbf{X}} \leqslant \mathbf{1} \tag{A.9}$$

or,

$$\hat{\mathbf{X}} \leqslant \mathbf{1}_{(n_s+n_u)T_p}$$

$$-\hat{\mathbf{X}} \leqslant \mathbf{0}_{(n_s+n_u)T_p}$$

which can be written as,

$$\begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \end{bmatrix}_{2(n_s+n_u)T_p \times (n_s+n_u)T_p} \hat{\mathbf{X}} \leqslant \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix} \tag{A.10}$$

which can be written as

$$\mathbf{A}_{\text{boundary}}\hat{\mathbf{X}} \leqslant \mathbf{b}_{\text{boundary}}$$

Compact Form of Constraints

Using inequality (A.6), (A.8), and (A.10), the LP problem can be written in compact from as,

$$\min_{\hat{\mathbf{X}}} \mathbf{C}^T\hat{\mathbf{X}}$$

s.t.

$$\begin{bmatrix} \mathbf{I}_{n_sT_p} & -\mathbf{T}_{xu} \\ -\mathbf{I}_{n_sT_p} & \mathbf{T}_{xu} \\ \mathbf{0}_{n_sT_p\times n_sT_p} & \mathbf{T}_{xu,c} \\ \mathbf{I}_{(n_s+n_u)T_p} & \\ -\mathbf{I}_{(n_s+n_u)T_p} & \end{bmatrix}_{(5n_s+2n_u)Tp\times(n_s+n_u)T_p} \hat{\mathbf{X}} \leqslant \begin{bmatrix} \mathbf{T}_{xx_0}\mathbf{x}[0] \\ -\mathbf{T}_{xx_0}\mathbf{x}[0] \\ \mathbf{T}_{xx_0,c}\mathbf{x}[0] \\ \mathbf{1}_{(n_s+n_u)T_p} \\ \mathbf{0}_{(n_s+n_u)T_p} \end{bmatrix}_{(5n_s+2n_u)Tp\times 1} \tag{A.11}$$

which can be written as,

$$\begin{bmatrix} \mathbf{A}_{\text{dynamics}} \\ \mathbf{A}_{\text{flow}} \\ \mathbf{A}_{\text{boundary}} \end{bmatrix} \hat{\mathbf{X}} \leqslant \begin{bmatrix} \mathbf{b}_{\text{dynamics}} \\ \mathbf{b}_{\text{flow}} \\ \mathbf{b}_{\text{boundary}} \end{bmatrix} \tag{A.12}$$

Notes for GLPK Use

`GLPK` is an open source linear program and mixed-integer program efficient solver that is used in the implementation of the proposed framework. We would like to note that the boundary constraints in (A.9) and A.12 are not considered in constraints matrix (the rows of A). It is entered as separate types of constraints in the defined `glpk` problem.

Estimation of Enemy State, $\mathbf{X}^{\mathrm{e}}$

Attackers state trajectory $\mathbf{X}^{\mathrm{a}}$ is used in the LP objective vector $\mathbf{C} = \beta\mathbf{X}^{\mathrm{ref}} + \alpha\mathbf{X}^{\mathrm{a}}$ as follows,

$$\mathcal{J}_{T_p}(\hat{\mathbf{X}}) = min_{\hat{\mathbf{X}}}\ \mathbf{C}^T\hat{\mathbf{X}} \tag{A.13}$$

In this implementation, the attackers state is computes based on the following linear model,

$$\mathbf{x}^{\mathrm{a}}[t+1] = \mathbf{x}^{\mathrm{a}}[t] + \mathbf{B}\mathbf{u}^{\mathrm{a}}[t] \tag{A.14}$$

where,

$$\mathbf{u}^{\mathrm{a}}[t] = \mathbf{G}^{\mathrm{a}}\mathbf{x}^{\mathrm{a}}[t] \tag{A.15}$$

Hence,

$$\mathbf{x}^{\mathrm{a}}[t+1] = (\mathbf{I} + \mathbf{B}\mathbf{G}^{\mathrm{a}})^{t+1}\mathbf{x}_0^{\mathrm{a}} \tag{A.16}$$

based on (A.16), the attackers state trajectory can be written as,

$$\mathbf{X}^{\mathrm{a}} = \begin{bmatrix} (\mathbf{I} + \mathbf{B}\mathbf{G}^{\mathrm{a}}) \\ (\mathbf{I} + \mathbf{B}\mathbf{G}^{\mathrm{a}})^2 \\ \vdots \\ (\mathbf{I} + \mathbf{B}\mathbf{G}^{\mathrm{a}})^{T_p} \end{bmatrix}_{n_sT_p \times n_s} \mathbf{x}_0^{\mathrm{a}} \tag{A.17}$$

or,

$$\mathbf{X}^{\mathrm{a}} = \mathbf{T}_{\mathbf{G}}\ \mathbf{x}_0^{\mathrm{a}}$$

A.0.0.0.2 Calculation of $\mathbf{G}^{\mathrm{a}}$:

- This matrix contains the probabilities that an agent moves from one sector s_i to another sector $s_j \in N_{s_i}$.
- $\mathbf{G} \in \mathcal{R}^{n_u \times n_s}$

- $g_{s_i \rightarrow s_j}$ is the probability that an agent in sector s_i will move to a neighbor sector s_j.
- $u_{s_i \rightarrow s_j} = g_{s_i \rightarrow s_j} \cdot x_{s_i}$
- Therefore, each row in $\mathbf{G}$ contains only one non-zero element, $g_{s_i \rightarrow s_j}$.
- The non-zero elements in matrix $\mathbf{G}$ can be accessed in a `Eigen` matrix as $g_{s_i \rightarrow s_j} = \mathbf{G}(s_i n_s - n_s + (s_j - 1), s_i - 1)$. This assumes that matrix indexing starts from zero, which is the case if `Eigen` matrix is used.

B Papers Submitted and Under Preparation

- **Mohamed Abdelkader**, and Jeff S. Shamma, "Toward Distributed Real-Time Planning in Multi-Agent System: Application to UAV-Based Adversarial Game", *journal paper, in progress.*
- **Mohamed Abdelkader**, Usman A. Fiaz, Noureddine Toumi , Mohamed A. Mabrok, and Jeff S. Shamma, "Cooperative Multi-Agent System for Autonomous Aerial Grasping in Outdoor Environments", *journal paper, in progress.*
- Samet Guler, **Mohamed Abdelkader**, and Jeff S. Shamma, "Formation Control of Autonomous Aerial Vehicles with On-board Ultrawideband Localization", *Submitted to International Conference on Intelligent Robots and Systems 2018.*
- Hassan Jaleel, **Mohamed Abdelkader**, and Jeff S. Shamma, "Real-time Distributed Motion Planning With Submodular Minimization", 2nd *IEEE Conference on Control Technology and Applications*, *accepted.*
- **Mohamed Abdelkader**, Yimeng Lu, Hassan Jaleel, and Jeff S. Shamma, "Distributed Real Time Control of Multiple UAVs in Adversarial Environment: Algorithm and Flight Testing Results", *Accepted in the International Conference on Robotics and Automation, ICRA, 2018.*
- **Mohamed Abdelkader**, Hassan Jaleel, and Jeff S. Shamma, "A Distributed Framework for Real Time Path Planning in Practical Multi-agent Systems", *Published in the International Federation of Automatic Control, IFAC, 2016.*
- **M. Abdelkader**, M. Shaqura, and C. G. Claudel, "Lagrangian Flash-Flood Monitoring and Inverse Modeling Using Microsensors Equipped UAVs" *Book chapter in Principles of Cyber-Physical Systems. Cambridge University Press, 2015. To appear*
- **Mohamed Abdelkader**, Mohammad Shaqura, Mehdi Ghommem, Nathan Col-

lier, Victor Calo, and Christian Claudel, "Optimal multi-agent path planning for fast inverse modeling in UAV-based flood sensing applications", *Published in the International Conference on Unmanned Aircraft Systems (ICUAS), 2014.*

• **Mohamed Abdelkader**, Mohammad Shaqura, Christian G. Claudel, Wail Gueaieb, "A UAV based system for real time flash flood monitoring in desert environments using Lagrangian microsensors", *Published in the International Conference on Unmanned Aircraft Systems (ICUAS), 2013.*