The Report Committee for Martin Braquet
Certifies that this is the approved version of the following Report:

Decentralized Auction-based Task Allocation with Guaranteed Collision Avoidance in Dynamic Environments

APPROVED BY

SUPERVISING COMMITTEE:

Efstathios Bakolas, Supervisor

David Fridovich-Keil

Decentralized Auction-based Task Allocation with Guaranteed Collision Avoidance in Dynamic Environments

by

Martin Braquet

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2022

"It is the time that you have wasted for your rose that makes your rose so important." **- Antoine de Saint-Exupéry**

Acknowledgments

This work has been supported in part by NSF under ECCS-1924790 and by a Fellowship of the Belgian American Educational Foundation.

Abstract

Decentralized Auction-based Task Allocation with Guaranteed Collision Avoidance in Dynamic Environments

Martin Braquet, M.S.E.
The University of Texas at Austin, 2022

Supervisor: Efstathios Bakolas

This report aims at solving task allocation and obstacle avoidance problems in a general context whose applications include disaster responses or packages deliveries by unmanned aerial / ground vehicles in dynamic unknown environments. More formally, we design a task allocation framework with collision avoidance capabilities in an environment populated by multiple mobile obstacles.

First, we propose a decentralized auction-based algorithm for the solution of dynamic task allocation problems for spatially distributed multi-agent systems. In our approach, each member of the multi-agent team is assigned to at most one task from a set of spatially distributed tasks, while several agents

can be allocated to the same task. The task assignment is dynamic since it is updated at discrete time stages (iterations) to account for the current states of the agents as the latter move towards the tasks assigned to them at the previous stage. Our proposed methods can find applications in problems of resource allocation by intelligent machines such as the delivery of packages by a fleet of unmanned or semi-autonomous aerial vehicles. In our approach, the task allocation accounts for both the cost incurred by the agents for the completion of their assigned tasks (e.g., energy or fuel consumption) and the rewards earned for their completion (which may reflect, for instance, the agents' satisfaction). We propose a Greedy Coalition Auction Algorithm (GCAA) in which the agents possess bid vectors representing their best evaluations of the task utilities. The agents propose bids, deduce an allocation based on their bid vectors and update them after each iteration. The solution estimate of the proposed task allocation algorithm converges after a finite number of iterations which cannot exceed the number of agents. We use numerical simulations to illustrate the effectiveness of the proposed task allocation algorithm (in terms of performance and computation time) in several scenarios involving multiple agents and tasks distributed over a spatial 2D domain.

Secondly, we present an algorithm for local motion planning in environments populated by moving elliptical obstacles whose velocity, shape and size may change with time. We base the algorithm on a collision avoidance vector field (CAVF) that aims to steer an agent to a desired final state whose motion is described by a double integrator kinematic model. In addition to handling

multiple obstacles, it is applicable in bounded environments for more realistic applications (e.g., motion planning inside a building). We also incorporate a method to deal with agents whose control input is limited so that they safely navigate around the obstacles. To showcase our approach, extensive simulations results are presented in 2D and 3D scenarios.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Task Allocation

First, we present a decentralized auction-based algorithm to address dynamic task allocation problems for multi-agent systems. In our problem, the agents have to complete a set of tasks which are distributed over a given spatial domain. We propose a decentralized solution for the computation of task assignment profiles based on auction-based negotiations between the agents. Our proposed methods can find applications in problems in which agents (e.g., autonomous vehicles, humans, robots, intelligent machines, etc.) have to share resources and distribute the workload among them in order to accomplish one or more tasks (see Fig. 1.1). In industrial environment (e.g., Amazon fulfillment centers), robots have to accomplish diverse tasks such as helping workers carry products. Disaster response by a fleet of unmanned aerial vehicles (UAV) which have to assess the severity of the situation and discover where help is needed more as well as the delivery of packages by autonomous or semi-autonomous ground or aerial robots are two characteristic examples.

Literature review: There are several types of task allocation problems for multi-agent systems depending on the ability of each agent to handle mul-

Figure 1.1: Example of disaster response mission: cars and robots collaborate and interact with people, robots remove debris in dangerous areas and drones extinct wildfires.

tiple tasks (involving task scheduling) and on whether it is possible to have multiple agents assigned to the same task (thus, allowing for the formation of coalition of agents). These problems can be addressed by auction-based techniques, distributed and / or multi-objective optimization, game-theoretic methods and machine-learning algorithms, to name but a few.

An important consideration when developing algorithms for multi-agent task allocation is the ability of these algorithms to be deployed in systems where there is no single entity that allocates tasks and workload among the agents. In this regard, centralized methods rely on a single point of operation in the sense that the agents negotiate with each other under the direction of a central entity [1]. Decentralized methods avoid this single point of failure by allowing each agent to consult directly with the other agents and compute their own task assignments. Decentralized execution, however, adds significant

computation time [2, 3, 4].

Auction-based approaches are derived from market economy principles in which each agent tries to maximize his own profit, based on the total reward that will then be redistributed among them. These methods are receiving an increased amount of attention (e.g., satellites in [5], drones in [6] mainly because of certain key benefits such as the worst-case global utility that can be derived theoretically by using them [7], their fast convergence, low complexity and high computational efficiency [8, 9]. Auction-based methods have also been boosted by recent breakthroughs in reinforcement learning [10]. The consensus-based bundle algorithm [2] (CBBA) utilizes a market-based decision strategy as the mechanism for decentralized task selection and uses a consensus routine based on local communication as a conflict resolution mechanism to achieve agreement on the winning bid values. Finally, other decentralized auction algorithms based on local communication have been developed to allow the agents to bid on a task asynchronously [11].

One of the simplest approaches to solve decentralized auction-based problems is via greedy algorithms, which consider the optimal (in a myopic sense) choice that maximizes a global objective [12]. Some approaches handle heterogeneous agents (with different traits / capabilities) by computing their utilities based on their own local information, and the task allocation is solely determined by their bids [13]. In this regard, such algorithms can calculate the agents' utilities based on their resource levels and the possibility of visiting refill stations [14]. Auction-based techniques have been proven to efficiently

produce suboptimal solutions [15] with a guaranteed convergence to a conflict-free assignment. Other advantages of auctions are their high scalability and robustness to variations in the communication network topology [16, 17].

Other types of task allocation methods include those which are based on game theory and in particular, potential games for the computation of mutually agreeable task assignments. Although the negotiation protocols are proven to converge to mutually agreeable tasks [18], their convergence is only guaranteed for the case in which the game remains the same (task utilities are constant) which is not the case in a dynamic task allocation problem. Other algorithms aim to compute a mutually agreeable profile corresponding to a Nash equilibrium (game-theoretic formulation of task allocation problems) for all agents [19]. Game theory is an important tool to extend task allocation problems to multiple agents but finding *efficient* Nash equilibria (task assignment profiles giving high global utility) is not guaranteed (solutions based on individual rationality may not automatically lead to high global utility) and computational cost can be significant. Likewise, constrained optimization approaches based on nonlinear programming tools require in general significant computational power and time. More efficient optimization-based task allocation methods that rely on tools from quadratic programming have been introduced in [20]. Recently, machine-learning algorithms have started to receive a significant amount of attention mainly because they can process a lot of information (by utilizing, for instance, neural networks) and handle unknown environments via reinforcement learning (especially Deep Q-Learning [21]. Re-

current neural networks also find applications in scheduling problems for clustered tasks in Multi-Task Robots Single-Robot Tasks Assignment problems [22].

Contributions: In the first part of this dissertation, we propose a dynamic auction-based task allocation algorithm. In our approach, the task utilities depend on both the rewards earned by the agents for accomplishing their assigned tasks as well as the costs they incur while doing so (the latter correspond to cost-to-go functions of relevant optimal control problems). The utilities are thus in general dependent on the state of the agents. In this context, the agents can only perform one task while several agents can be assigned the same task (if this is beneficial to them and their team). In contrast with game-theoretic algorithms which may not always achieve high global utility for the team (inefficient Nash equilibria), our proposed auction-based task allocation mechanism finds task assignments that maximize the global utility of the system. A key advantage of our proposed approach is time efficiency, yet with reasonably high global utility.

We propose a Greedy Coalition Auction Algorithm (GCAA) where the agents negotiate while moving in their state space towards their assigned tasks. When an agent changes his assignment, he needs to recompute the cost estimate and thus his own state-dependent utility. In contrast to game-theoretic solutions [19] which aim for individual rationality but cannot guarantee good team performance, we do not seek a mutually agreeable task assignment but consider instead a broader set of solutions that allows for a higher global util-

ity. Furthermore, in contrast with the CBBA algorithm which clusters and schedules a sequence of tasks for each agent, in this work the problem is composed of multiple agents making a coalition for a specific task that is spatially distributed (which is the only task for that agent). This work hence falls under the category of *Single-Task Robots Multi-Robot Tasks Instantaneous Assignment* (ST-MR-IA) problem, also known as the coalition formation problem [15].

1.2 Obstacle Avoidance

Collision avoidance constitutes a fundamental problem in robotics. For example, unmanned aerial vehicles (UAVs) have found many applications related to search and rescue, payload and package delivery, and surveillance, to name but a few. In the multi-agent settings, the agents need to first solve a task allocation problem before considering the motion planing and its underlying obstacle avoidance. To accomplish their mission, robots have to plan their path in environments that are often populated by obstacles. Such obstacles are not precisely known or can be mobile and hence are often characterized in probabilistic ways (e.g., a confidence or probability ellipsoid). Due to the motion and rotation of these obstacles, as well as the dynamic information gathered by the robot, the probability density of the obstacles (i.e., their shape) vary with time. Additionally, they are required to react in real time to pop-up tasks and addition or removal of some other agents. Therefore, a well-designed algorithm for such problems needs to be decentralized, reactive, computationally

inexpensive, and handle fast collision avoidance.

Literature review: Some of the earliest and most notable algorithms tackling obstacle avoidance problems consist of creating a graph by discretizing the autonomous agent's free configuration space, considering only the geometric requirement of finding an obstacle-free path between two points. Some of the most popular graph-based search algorithms are Dijkstra's ([23]) and A* ([24, 25]), which require a configuration of the space graph using basic or more advanced sampling techniques such as Voronoi diagrams and projections ([26, 27]). However, discrete methods rely on search-space granularity (i.e., searching a complex discrete space) to compute the optimal solution, and therefore, are not very suitable for real-time applications.

Many relevant algorithms have been developed for motion planning in dynamic environments using *velocity obstacles*, which consist in selecting avoidance maneuvers to avoid obstacles in the velocity space ([28]). These algorithms are generated by selecting robot velocities outside of the velocity obstacles, which represent the set of robot velocities that would result in a collision. Optimal reciprocal collision avoidance (ORCA) is an extension of the velocity obstacle algorithm to include multiple agents by resolving the common oscillation problem in multi-agent navigation ([29, 30, 31]). ORCA takes into account the reactive behavior of the other agents by implicitly assuming that they make a similar collision-avoidance reasoning. The agent chooses a velocity that lies outside any of the velocity obstacles induced by the moving obstacles. If among the free velocities, the velocity chosen is the one that is

in the most direct way towards the agent's goal position, the agent will safely navigate towards its goal. ORCA has however two major limitations: deadlocks and local minima (of the objective function representing the path cost) ([32, 33]). Indeed in more complex scenarios such as cluttered / obstacle rich environments and moving obstacle environments, the agents can get stuck in local minima where the obstacles block the path to the target or produce poor trajectories as they do not plan for future obstacle locations. Algorithms based on vector fields are able to avoid such local minima by creating a null vector field, and thus a local minimum, only at the desired final location. In this paper, we give a guarantee that the agent reaches the goal and does not get stuck. To take the *probabilistic* motion into account, an extended velocity obstacle (EVO) validation system extends ORCA in the real world by including state uncertainties via a position filtering system which handles a noise model that is discontinuous and non-linear ([34]).

More recently, new algorithms have been leveraging the qualities of machine learning to tackle problems in unknown / sparse dynamic environments. For example, an algorithm for socially aware multiagent collision avoidance with deep reinforcement learning (SA-CADRL) has been designed to produce more time-efficient paths than ORCA ([35, 36]).

Algorithms based on harmonic potential flow can deal with multiple moving convex and star-shaped concave obstacles by applying a dynamic modulation matrix, dependent on the characteristics of the obstacles, to the original dynamic system ([37, 38]). However, this velocity-based controller does

not take into account the inertia of the robot or more realisitc kino-dynamic constraints.

Local methods represent a robot as a particle in the configuration space under the influence of an artificial potential field, whereas the direction of motion is generally chosen according to the gradient of the potential function ([39, 40]). However, such methods suffer from important limitations: trap situations due to local minima (cyclic behavior), no passage between closely spaced obstacles, oscillations in the presence of multiple obstacles and in narrow passages ([41]).

Global path planning methods such as rapidly-exploring random trees (RRT) have the advantage that they can be directly applied to nonholonomic and kinodynamic planning ([42]). Indeed, RRT generates a tree by providing the required control input making the link between two adjacent states. General motion planning between two locations can be used more efficiently than the classical RRT by incrementally building two rapidly-exploring random trees (RRTs) rooted at the start and the goal configurations ([43, 44, 45]). The previously mentioned field-based algorithms (harmonic flow, potential, etc) can further be combined with such global path planning to solve the problem of artificial potential field algorithms being liable to fall into a a local minimum ([46]).

Other popular approaches for collision avoidance rely on curve parameterization for trajectory generation ([47]) or optimization-based algorithms that aim to solve nonlinear programs with different nonlinear programming

(NLP) solvers ([48]). Still, these methods are computationally expensive and not suited for real-time applications.

To avoid the computational complexity of the previously discussed techniques, algorithms based on collision avoidance vector fields have been developed for specific types of dynamics. For example, [49] consider UAV applications by assuming constant-altitude operations and thus approximating the system with a planar kinematic Dubins model. In this case, the agent aims to avoid obstacles while keeping its steering angle close to a desired angle. However, the applicability of the method proposed in [49] is limited to circular obstacles.

Contributions: The second part of this dissertation presents a novel local motion-planning algorithm in environments populated by moving obstacles with time-varying shapes. We base our algorithm on a collision avoidance vector field (CAVF) that aims to steer an agent to a desired final state under double integrator dynamics in an environment populated with multiple moving elliptical obstacles.

We first assume that the agent has access to the exact position and velocity of the center of mass of the obstacles, as well as their shape and orientation. Secondly we consider obstacles with uncertain motion resulting in a covariance of their probability distribution growing over time. We also consider bounded environments for more realistic applications. Finally, we incorporate a method to deal with agents whose control input is limited so that they safely navigate around the obstacles.

Multi-agent systems (MAS) further require a task assignment scheme in addition to the avoidance of obstacles and other agents. This work is extended to multi-agent settings by incorporating the fast task allocation algorithm described in Chapter 2 wrapped around our proposed collision avoidance method ([50]).

Chapter 2

Task Allocation

Disclaimer: The material in this chapter is based on our previously paper published at MECC 2021 [50].

2.1 Preliminaries

We assume a multi-agent system (MAS) comprised of n agents. These agents are called active agents when they are far from their target so that they can recompute their best task assignment while moving toward the target, otherwise they are called passive agents when they are too close to the target to consider other targets (they are then permanently assigned to this final task). Let $x_i \in \mathcal{S}_i \in \Sigma$ and $u_i \in U_i$, for $i \in [1, n]_d$ be the state and input of the i-th agent of the MAS at time $t \geq 0$ ($\mathcal{S}_i$ being his state space and U_i his input space), and $\Sigma \subseteq \mathbb{R}^m$. We also define $\boldsymbol{x} \in \mathcal{S}$ the joint state of the MAS, in which $\boldsymbol{x} := (x_1, \ldots, x_n)$ and $\mathcal{S} := \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ (joint state space). Let $\boldsymbol{u} \in U$ be the joint input of the MAS, where $\boldsymbol{u} := (u_1, \ldots, u_n)$ and $U := U_1 \times \cdots \times U_n$ (joint input space).

The motion of the i-th agent is described by

$$\dot{x}_i = f_i(\boldsymbol{x}, \boldsymbol{u}), \qquad x_i(0) = x_i^0, \qquad i \in [1, n]_d, \tag{2.1}$$

where $x_i^0 \in \mathcal{S}_i$ is the initial state of the i-th agent and $f_i : \mathcal{S}_i \times U_i \to \mathcal{S}_i$ is his associated vector field. Consequently, $\boldsymbol{x}^0 = (x_1^0, \ldots, x_n^0) \in \mathcal{S}$ is denoted as the joint initial state.

In general, task allocation aims to assign individual tasks for n agents and p tasks, $\mathcal{T} := \{\mathcal{T}_1, \ldots, \mathcal{T}_p\}$. Let $X_\mathcal{T}$ be the set of states associated with the given tasks, where $X_\mathcal{T} := \{x_{\mathcal{T}_1}, \ldots, x_{\mathcal{T}_p}\}$, and $\mathcal{A}_i := \{a_i^k : \ k \in [1, \text{card}(\mathcal{A}_i)]_d\}$ the set of possible task assignments for the i-th agent given a set of tasks $\mathcal{T}$. While the agents have limited communication between each other, we suppose that they have complete information about all the tasks available. Each assignment $a_i^k \in \mathcal{A}_i$ is equal to either a task in $\mathcal{T}$, that is, $a_i^k = \mathcal{T}_\ell$ where $\mathcal{T}_\ell \in \mathcal{T}$, or the null assignment, that is, $a_i^k = a_\varnothing$.

Additionally, we denote the set of active agents as $\mathcal{N}_a \subseteq [1, n]_d$ and we fix the assignment a_i of agent i (thus switching his status from active to passive) for all $t > t_p$ if the agent lies inside the boundary of the target, that is, $\Phi_i(x_i(t_p), x_{\mathcal{T}_j}) < 0$ where $\Phi_i(x_i(t_p), x_{\mathcal{T}_j})$ is a boundary constraint; for instance $\Phi_i(x_i(t_p), x_{\mathcal{T}_j}) := \|x_i(t_p) - x_{\mathcal{T}_j}\| - R_p$ where R_p is the minimum agent-to-target distance to make the task assignment permanent.

2.2 Task Utilities

In this work, the task utility is characterized by a reward obtained for the completion of the task $\mathcal{T}_j \in \mathcal{T}$ and a state-dependent cost to finish this task (for example, the transition cost due to the motion of the agent).

Static task utility: Given an assignment profile $\boldsymbol{a} = (a_1, \ldots, a_n)$, we denote by $\mathcal{T}_j^{-1}(\boldsymbol{a})$ the index-set corresponding to the agents assigned to task $\mathcal{T}_j \in \mathcal{T}$ under the particular profile. Since a task is not necessarily accomplished when an agent is assigned to it, we let $p_{ij} \in [0, 1]$ be the probability of the task $\mathcal{T}_j$ to be completed successfully by the i-th agent. In this case, the probability that the task is successfully completed by at least one agent increases with the number of agents assigned to this task. The expected reward for completing task $\mathcal{T}_j$ is defined as [19]:

$$r_{\mathcal{T}_j}(\boldsymbol{a}) = \bar{r}_{\mathcal{T}_j} \left[1 - \prod\nolimits_{i \in \mathcal{T}_j^{-1}(\boldsymbol{a})} (1 - p_{ij}) \right], \tag{2.2}$$

where $\bar{r}_{\mathcal{T}_j}$ is the nominal reward of $\mathcal{T}_j$. Indeed, the probability that at least one agent completes the task is equal to the complementary of the probability that no agent completes the task, i.e. $\prod_{i \in \mathcal{T}_j^{-1}(\boldsymbol{a})} (1 - p_{ij})$. It is worth noting that the assignments (and their associated utility) of the passive agents are also taken into account to compute the total reward.

State-dependent task completion cost: The cost to finish the task $\mathcal{T}_j$ associated with the state $x_{\mathcal{T}_j}$ at time $t = t_{\mathsf{f},\mathcal{T}_j}$ by the i-th agent is defined as the optimal cost related to the optimal control problem presented in Problem 1.

Problem 1. *Let $a_i = \mathcal{T}_j$, where $\mathcal{T}_j \in \mathcal{T}$ and $i \in [1, n]_d$. Furthermore, we denote $x_{\mathcal{T}_j} \in \mathcal{S}_i$ as the state linked to $\mathcal{T}_j$ and $t_{\mathsf{f},\mathcal{T}_j} > 0$ as the related completion time for $\mathcal{T}_j$. The goal is to obtain an optimal control input $u_i^\star(\cdot) : [0, t_\mathsf{f}] \to U_i$ that is piece-wise continuous and minimizes the functional given by:*

$$\mathcal{J}_i(u_i(\cdot); x_i^0, x_{\mathcal{T}_j}) := \int_0^{t_\mathsf{f}} \mathcal{L}_i(x_i(t), u_i(t)) \mathrm{d}t, \tag{2.3}$$

such that the dynamic constraints (2.1) *and the following terminal constraint* $\Psi_i(x_i(t_{\mathsf{f}}), x_{\mathcal{T}_j}) = 0$*, where* $\Psi_i(\cdot; x_{\mathcal{T}_j})$ *is a given* C^1 *function, are respected. Finally, the minimum cost is given by* $\rho_i(x_i^0; x_{\mathcal{T}_j}) := \mathcal{J}_i(u_i^\star(\cdot); x_i^0, x_{\mathcal{T}_j})$.

Remark 1 The terminal constraint function Ψ_i follows either

$$\Psi_i(x_i(t_{\mathsf{f}}), x_{\mathcal{T}_j}) = x_i(t_{\mathsf{f}}) - x_{\mathcal{T}_j}$$

implying $x_i(t_{\mathsf{f},\mathcal{T}_j}) = x_{\mathcal{T}_j}$, or

$$\Psi_i(x_i(t_{\mathsf{f}}), x_{\mathcal{T}_j}) = \left\| x_i(t_{\mathsf{f}}) - x_{\mathcal{T}_j} \right\| - R_{\mathcal{T}_j}$$

requiring some agents to loiter around the target $\mathcal{T}_j$ with a certain radius $R_{\mathcal{T}_j}$ during a loitering time $\tau_{\mathcal{T}_j} \in [0, t_{\mathsf{f},\mathcal{T}_j}]$, in which case these agents start loitering at time $t_{\mathsf{f},\mathcal{T}_j} - \tau_{\mathcal{T}_j}$.

Total Task Utility: The total completion cost of task $\mathcal{T}_j$ given the assignment profile $\boldsymbol{a} = (a_1, \ldots, a_n)$ is given by the sum of the individual task completion costs of all the agents assigned to that task. More precisely,

$$\mathfrak{R}_{\mathcal{T}_j}(\boldsymbol{a}; \boldsymbol{x}^0, x_{\mathcal{T}_j}) := \sum\nolimits_{i \in \mathcal{T}_j^{-1}(\boldsymbol{a})} \rho_i(x_i^0; x_{\mathcal{T}_j}), \tag{2.4}$$

which leads to the definition of the total task utility associated with task $\mathcal{T}_j$ for a given $\boldsymbol{x}_0$

$$\mathcal{U}_{\mathcal{T}_j}(\boldsymbol{a}; \boldsymbol{x}^0) := r_{\mathcal{T}_j}(\boldsymbol{a}) - \lambda_{\mathcal{T}_j} \mathfrak{R}_{\mathcal{T}_j}(\boldsymbol{a}, \boldsymbol{x}^0; x_{\mathcal{T}_j}) \tag{2.5}$$

where $\lambda_{\mathcal{T}_j}$ is a constant which is used to convert the cost-to-go to the same units as the reward (e.g. from a loss of energy to a loss of money).

Individual, Team Utilities: Let us denote the global utility as

$$\mathcal{U}(\boldsymbol{a};\boldsymbol{x}^0) := \sum\nolimits_{\mathcal{T}_j\in\mathcal{T}} \mathcal{U}_{\mathcal{T}_j}(\boldsymbol{a};\boldsymbol{x}^0). \tag{2.6}$$

The goal is to set this team's utility equal to the sum of each individual utility in order to maximize each individual utility separately. In this regard, based on the task assignment $\boldsymbol{a}$, we set the individual utility of agent i equal to his marginal contribution to the global utility $\mathcal{U}(\boldsymbol{a};\boldsymbol{x}^0)$:

$$\begin{aligned}\mathcal{U}_i(\boldsymbol{a};\boldsymbol{x}^0) &:= \mathcal{U}(\boldsymbol{a};\boldsymbol{x}^0) - \mathcal{U}((a_\varnothing, a_{-i});\boldsymbol{x}^0) \\ &= \mathcal{U}_{\mathcal{T}_j}(\boldsymbol{a};\boldsymbol{x}^0) - \mathcal{U}_{\mathcal{T}_j}((a_\varnothing, a_{-i});\boldsymbol{x}^0).\end{aligned} \tag{2.7}$$

2.3 Dynamic Task Allocation

2.3.1 Preliminaries: open-loop task allocation

In the formulation of the open-loop task allocation problem, the agents' utilities (or, more precisely, their functional descriptions) do not change with time, as the agents progress towards the states of their assigned tasks. This is because their estimated task completion costs are based on knowledge available at time $t = 0$ and these estimates are not updated afterwards. To the i-th individual task assignment $a_i^\star$ from the optimal profile $\boldsymbol{a}^\star$, where, say, $a_i^\star = \mathcal{T}_j$, we associate a corresponding state $x_{\mathcal{T}_j}$, which in turn determines the terminal constraint $\Psi(x_i(t_\mathsf{f}; x_{\mathcal{T}_j}) = 0$ in Problem 1. Because all the task assignments are time-invariant, the control input $u_i^\star(\cdot)$ that solves Problem 1 will not be updated along the i-th agent's ensuing trajectory.

2.3.2 Problem formulation

The task allocation is called dynamic since the utilities of the agents change along their path towards their target (state-dependent cost and agents obtain new information by communicating with other agents in the surrounding). In this case, a new assignment profile $\boldsymbol{a}^\star(t)$ has to be selected at each time step $t \in [0, t_\mathsf{f}]$ as the agents evolve in their state space.

Problem 2 (DTA: Dynamic Task Allocation)**.** *Let $t_\mathsf{f} > 0$ and $\boldsymbol{x}^0 \in \mathcal{S}$, find a time-varying task assignment profile $\boldsymbol{a}^\star(\cdot) : [0, t_\mathsf{f}] \to \mathcal{A}$ for all the remaining active agents $i \in \mathcal{N}_a$, that maximizes the global utility in a decentralized way (communication constraints) according to the permanent assignment a_{i_p} of the passive agents $i_p \in \mathcal{N}_p = [1, n]_d \setminus \mathcal{N}_a$ and the terminal constraints.*

2.3.3 Auction protocols for decentralized task allocation

The main principle of auctions consists in the computation of agents' individual utility for some tasks. Based on these proposed bids, the agents communicate between each other in order to deduce the best allocation for each of them. A key point is that for their realization, an agent does not have to know the utilities of his teammates (decentralized implementation). The main idea behind the algorithm is to find the best task coalition for the multi-agent network by allocating the tasks to the agents obtaining the highest utility (also called greedy approach).

2.3.4 Greedy Coalition Auction Algorithm

The GCAA is an auction-based algorithm that leverages the simplicity of greedy approaches to provide a solution with fast convergence. The main idea is to iterate between an auction phase and a consensus phase such that it converges to a winning bids list ([2]).

Each agent has three vectors that are constantly updated at each iteration step t. The first vector $\boldsymbol{z}_i \in [0, p]_d^n$ is the list of selected tasks among $\mathcal{T}$, meaning that agent i possesses a vector $\boldsymbol{z}_i$ of length n where the k-th element of the vector is the expected task assignment of agent k to the best knowledge of agent i. The second vector $\boldsymbol{y}_i \in \mathbb{R}_{>0}^n$ is the list of winning bids (agent's utilities), that is, the k-th element of $\boldsymbol{y}_i$ is the expected individual utility of agent k by selecting the task $z_{i,k}$ (k-th element of the vector of selected tasks $\boldsymbol{z}_i$). The third vector $\boldsymbol{c}_i \in [0, 1]_d^n$ is the list of finalized (or completed) allocations and informs agent i about the status of the allocation for the other agents. In particular, the k-th element of $\boldsymbol{c}_i$ is set to 1 if the agent k does not plan to change his target anymore, and 0 otherwise. This way, the agents for which the assignment is completed are not taken into account for the auction process in subsequent steps. Based on these three vectors that are first updated, each agent will decide and propose the best assignment for themselves (i.e., maximizing their own utility). The main algorithm is presented in Main Algorithm and the two associated phases are explained next.

Main Algorithm: Greedy Coalition Auction Algorithm
Input: $\boldsymbol{x}^0$
Output: $\boldsymbol{z}(t)$

1: $t = 0$
2: $\boldsymbol{y}(0) = \mathbf{0}$
3: $\boldsymbol{z}(0) = \mathbf{0}$
4: $\boldsymbol{c}(0) = \mathbf{0}$
5: **while** $\exists i : c_{i,i}(t) = 0$ **and** $c_{i,i}(t-1) = 0$ **do**
6: SelectBestTask()
7: ShareStateVectors()
8: UpdateStateVectors()
9: $t = t + 1$

2.3.4.1 Auction process:

The first phase of the algorithm is the auction process. Here, each agent aims to select his best task based on his own utility. At lines 2–4 of Algorithm 1, the previous bid vectors are copied into the current ones. If the task selected by one agent i is not finalized (line 5), agent i picks the task J_i that maximizes his expected utility (lines 6–7). Agent i then updates his bid vector with the selected task (line 8) and the associated utility (line 9).

2.3.4.2 Consensus process:

In Algorithm 2, the consensus process first aims to share the bid vectors $\boldsymbol{y}_i$, $\boldsymbol{z}_i$, $\boldsymbol{c}_i$ with the other agents within the communication range of agent i. For each agent i, the agents k within the communication range of agent i (satisfying $g_{ik}(t) = 1$ at lines 1–2) send their bid vectors $y_{k,k}(t), z_{k,k}(t)$ and $c_{k,k}(t)$ (lines 3–5). Then in Algorithm 3, based on his winner bids vector, agent i determines the set of agents $\tilde{\mathcal{A}}_i(t)$ allocated to the same selected task

Algorithm 1 Select the best task for agent i

Function SelectBestTask
Input: $\boldsymbol{y}(t-1), \boldsymbol{z}(t-1), \boldsymbol{c}(t-1), \boldsymbol{x}^0$
Output: $\boldsymbol{y}(t), \boldsymbol{z}(t)$

1: **for** $i \in [1, n]_d$ **do**
2: $\quad \boldsymbol{y}_i(t) = \boldsymbol{y}_i(t-1)$
3: $\quad \boldsymbol{z}_i(t) = \boldsymbol{z}_i(t-1)$
4: $\quad \boldsymbol{c}_i(t) = \boldsymbol{c}_i(t-1)$
5: $\quad$ **if** $c_{i,i}(t) = 0$ **then**
6: $\quad\quad \boldsymbol{a} = \boldsymbol{z}_i(t)$
7: $\quad\quad J_i = \text{argmax}_j \, \mathcal{U}_i((z_{i,j}(t), a^\star_{-i}); \boldsymbol{x}_i^0)$
8: $\quad\quad z_{i,i}(t) = J_i$
9: $\quad\quad y_{i,i}(t) = \mathcal{U}_i(\boldsymbol{z}_i(t); \boldsymbol{x}_i^0)$

(line 2) and extracts the winner based on their utility (line 3). He adds the winner to the list of finalized allocations $\boldsymbol{c}_i$ (line 4) and resets the values of the losers in the bids $\boldsymbol{y}_i$ and tasks $\boldsymbol{z}_i$ (lines 5–8).

Algorithm 2 Share the bid vectors to agent i

Function ShareStateVectors
Input: $\boldsymbol{y}(t), \boldsymbol{z}(t), \boldsymbol{c}(t)$
Output: $\boldsymbol{y}(t), \boldsymbol{z}(t), \boldsymbol{c}(t)$

1: **for** $i \in [1, n]_d$ **do**
2: $\quad$ **for** $k \in \{k \,|\, g_{ik}(t) = 1\}$ **do**
3: $\quad\quad z_{i,k}(t) = z_{k,k}(t)$
4: $\quad\quad y_{i,k}(t) = y_{k,k}(t)$
5: $\quad\quad c_{i,k}(t) = f_{k,k}(t)$

Then the time is updated ($t \leftarrow t + 1$) and the main algorithm loops to Algorithm 1. Finally, the algorithm has converged when the finalized choices are validated for some agents ($c_{i,i} = 1$) and the other agents not assigned to a task ($c_{i,i} = 0$) have not changed since the past iteration (meaning that the cost to reach each task is higher than the marginal reward they can obtain).

Algorithm 3 Update the bid vectors of agent i according to the winners/losers

Function UpdateStateVectors
Input: $\boldsymbol{y}(t), \boldsymbol{z}(t), \boldsymbol{c}(t)$
Output: $\boldsymbol{y}(t), \boldsymbol{z}(t), \boldsymbol{c}(t)$

for $i \in [1, n]_d$ **do**
  $\tilde{\mathcal{A}}_i(t) = \{k \mid z_{i,k}(t) = z_{i,i}(t), f_{i,k}(t) = 0\}$
  $K_i = \text{argmax}_{k \in \tilde{\mathcal{A}}_i(t)} y_{i,k}(t)$
  $c_{i,K_i}(t) = 1$
  for $k \in \tilde{\mathcal{A}}_i(t) \setminus K_i$ **do**
    $z_{i,k}(t) = 0$
    $y_{i,k}(t) = 0$

Once the task allocation is completed, the agents move according to the solution of Problem 1 minimizing the cost from the agent to the target. In order to prevent abrupt trajectory changes during the dynamic allocation, we stop the computation of the allocation when the agents are close to their associated target, that is, if $t > t_{\mathsf{s},\mathcal{T}_j}$ where $t_{\mathsf{s},\mathcal{T}_j}$ is the stop time for agent $\mathcal{T}_j$.

2.3.5 Application example

To illustrate the main steps of the algorithm through a simple example with 2 tasks and 4 agents, Fig. 2.1 shows a task allocation along with their bid vectors. The communication links are shown with red dashed lines and the final task allocation is given with green dashed lines.

Since agents A_1 and A_3 cannot communicate directly with each other, they assume that they will obtain the entire reward by completing their selected task T_2 while they will actually need to split it. In more details, each agent i fills in his bid vector (associated to his i-th row) depending on his best

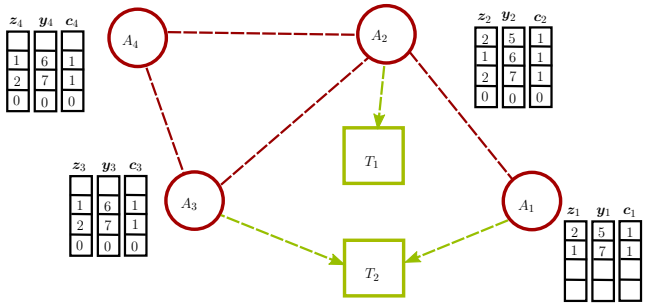


Figure 2.1: Graphical illustration of the auction-based greedy algorithm.

assignment during the first iteration. For example, agent A_1 chooses task T_2 with a utility of 5 while agent A_2 chooses task T_1 with a utility of 6. Then, they share their bid vector (i.e., fill in their rows) with their neighbors only, so that A_1 does not have information about A_3, and reciprocally. Each agent finally updates his bid vector by selecting the task with the highest utility and setting the associated assignment status $\boldsymbol{c}$ to 1 (e.g., at the first iteration, all agents finalize the assignment of A_2 because he proposes a utility of 7). At the next iterations, the assignment of A_2 is no longer computed and the other agents take into account the permanent assignment of A_2 for the computation of their own utility (e.g., A_4 no longer proposes a bid for T_2 because the reduction of the marginal reward associated to the coalition with A_2 dropped his marginal utility below zero, it is thus preferable for A_4 not to select any task by securing a null utility). At the second iteration, A_1 and A_3 propose and finalize their assignment for T_2 since they think that they are completing T_2 individually (no communication between them) and A_4 does not propose

any assignment. This example thus shows that communication constraints can lead to suboptimal solutions because the actual utility that A_1 and A_3 will receive by completing T_2 is lower than their prediction.

Theorem 2.3.1. *Consider the auction-based task allocation process solved by the GCAA algorithm (Main Algorithm) where the communication range can be limited. Let n be the number of agents, then GCAA converges to an assignment within at most n steps.*

Proof. The proof is derived from the definition of greedy algorithms. In particular at each time iteration t and for all agents $i \in [1, n]_d$, one element (index K_i as presented in Algorithm 3) of $\boldsymbol{c}_i$ is set to 1 as the task of agent i is set to be finalized. As a consequence at time t, there are t elements of $\boldsymbol{c}_i$ set to 1 and $n - t$ elements still initialized to 0. Hence at time $t = n$, all the elements of $\boldsymbol{c}_i$ are set to 1 for each agent i which means that the stopping criteria in Main Algorithm ($c_{i,i} = 1$ for all agents i) is necessarily verified. The algorithm is thus proven to converge after at most n steps (the number of agents). □

Remark 2 This convergence theorem guarantees that the computation time is growing linearly with the number of agents.

2.4 Numerical Simulations

In this section, we present numerical simulations[1] to illustrate the main ideas of the methods proposed so far. We consider a team of agents with double integrator dynamics, that is, $\ddot{p}_i = u_i$, with $p_i(0) = p_i^0$ and $\dot{p}_i(0) = v_i^0$, where $p_i \in \mathbb{R}^2$ ($p_i^0 \in \mathbb{R}^2$) and $\dot{p}_i \in \mathbb{R}^2$ ($v_i^0 \in \mathbb{R}^2$) denote, respectively, the position and velocity of the i-th agent at time t ($t_0 = 0$), $i \in [1, n]_d$. The performance index is given by the control effort $\mathcal{J}(u_i(\cdot)) := (1/2) \int_0^{t_\mathsf{f}} |u_i(t)|^2 \mathrm{d}t$ and the conversion constant is $\lambda_{\mathcal{T}_j} = 1$ ($j \in [1, p]_d$). By setting $x_i := (p_i,\ \dot{p}_i) \in \mathbb{R}^4$ and $x_{\mathcal{T}_j} := (p_{\mathcal{T}_j}, 0) \in \mathbb{R}^4$, the terminal constraint function is chosen randomly between:

- $\Psi_i(x_i(t_{\mathsf{f},\mathcal{T}_j}); x_{\mathcal{T}_j}) := x_i - x_{\mathcal{T}_j}$, which means that the i-th agent tries to reach the position $p_{\mathcal{T}_j}$ associated with his assigned task $\mathcal{T}_j$ at time $t = t_{\mathsf{f},\mathcal{T}_j}$ and with terminal velocity $\dot{p}_{\mathcal{T}_j}$ (randomly selected).

- $\Psi_i(x_i(t_{\mathsf{f},\mathcal{T}_j}); x_{\mathcal{T}_j}) := \|p_i - p_{\mathcal{T}_j}\| - \tilde{R}_{\mathcal{T}_j}$, which means that the i-th agent tries to reach the circle (with radius $\tilde{R}_{\mathcal{T}_j}$) around his assigned task $\mathcal{T}_j$ at time $t = t_{\mathsf{f},\mathcal{T}_j} - \tau_{\mathcal{T}_j}$ and then loiters around the target until $t_{\mathsf{f},\mathcal{T}_j}$. In this work, the best entry point to enter the circle is selected by discretizing the circle in 10 points and selecting the point that minimizes the cost function[2].

[1] Source code available at `https://github.com/MartinBraquet/task-allocation-auctions`.

[2] The best solution can also be found by optimal control methods in a systematic / rigorous way and will be considered in further work

Both terminal constraints are associated with an optimal control problem with non-zero initial and terminal velocities. It turns out (see, for instance, [51]) that the optimal control input is given by

$$u_i^\star(t; t_\mathsf{f}, x_i^0, x_{\mathcal{T}_j}) = \frac{4}{t_\mathsf{f} - t}\left[\dot{p}_{\mathcal{T}_j} - \dot{p}_i(t)\right] + \frac{6}{(t_\mathsf{f} - t)^2}\left[p_{\mathcal{T}_j} - p_i(t) - \dot{p}_{\mathcal{T}_j}\,(t_\mathsf{f} - t)\right] \tag{2.8}$$

which defines a second-order differential equation with time-varying coefficients where $u_i^\star(t) = \ddot{p}_i(t)$. It is solved numerically using integration tools (ODE45) in MATLAB.

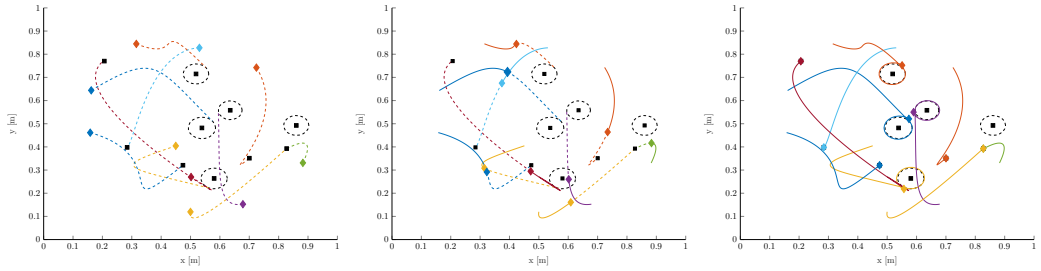
While problems with zero terminal velocities have an analytical solution ([52]), problems with non-zero terminal velocities require more computation time due to the numerical integration. The optimal cost-to-go is then obtained via the definition of $\mathcal{J}(u_i(\cdot))$. In addition to this dynamic solution, a drag term (or friction force) $-k_\mathrm{d}\,\dot{p}_{\mathcal{T}_j}$ proportional to the agent's velocity is used to refine the previous ideal equations of motion. It will thereby slow down to zero velocity an agent when he is not subject to any input control (i.e., he does not have any assigned task) while being negligible when the agent is subject to a typical control input.

A dynamic task allocation is then performed and presented through the dynamic map of the allocation. Fig. 2.2 illustrates trajectories of the agents in the absence of communication constraints (or limitations) while the agents in Fig. 2.3 can only communicate[3] with the other agents within range $\varrho = 0.3$.

[3]The communication range is not shown in the figure for clarity.

We set the simulation time to $t_\mathsf{f} = 10$ and discretize it in $k = 1000$ iterations which implies a constant time step $\delta t = t_\mathsf{f}/k = 0.01$. Due to the fact that the computation time required for the numerical integration is substantial, we only consider scenarios with $n = 10$ agents and $p = 10$ tasks. The completion time $t_{\mathsf{f},\mathcal{T}_j}$, which is dependent on the task $\mathcal{T}_j$, is computed randomly such that $t_{\mathsf{f},\mathcal{T}_j}/t_\mathsf{f} \in [0.9, 1]$. 5 tasks are fixed targets with non-zero terminal velocities $\dot{p}_{\mathcal{T}_j} \in [-0.1, 0.1]$ (black squares). The other 5 tasks are dynamic, the agents need to loiter at a radius $\tilde{R}_{\mathcal{T}_j} \in [0.032, 0.048]$ and complete one loop at velocity $\dot{p}_{\mathcal{T}_j} \in [-0.1, 0.1]$ for a time $\tau_{\mathcal{T}_j}$ such that $\tau_{\mathcal{T}_j}/t_\mathsf{f} \in [0.15, 0.25]$ (black dashed circle around a dashed square).

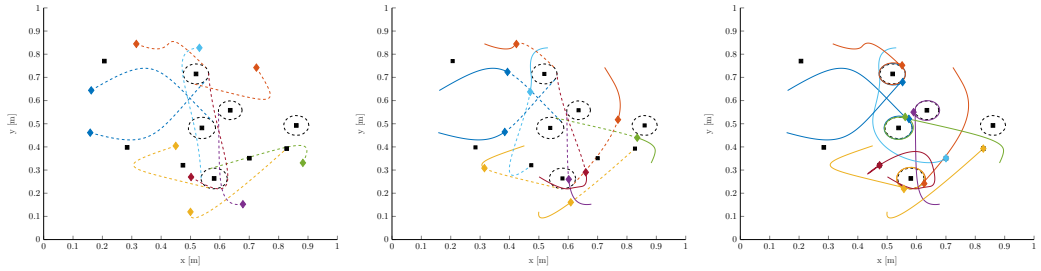
The time t_p after which the algorithm preserves the same allocation for an agent is satisfying $\Phi_i(x_i(t_p), x_{\mathcal{T}_j}) = \|x_i(t_p) - x_{\mathcal{T}_j}\| - 2\tilde{R}_{\mathcal{T}_j} < 0$ so that the allocation is blocked when the agent enters the circle of radius $2\tilde{R}_{\mathcal{T}_j}$ centered in $x_{\mathcal{T}_j}$ before starting loitering. The nominal rewards are such that $\bar{r}_{\mathcal{T}_j} \in [0, 0.2]$ for the fixed tasks, they are higher ($\bar{r}_{\mathcal{T}_j} \in [0, 1]$) for the loitering tasks since they typically require more cost to achieve the rotation. The success probability p_{ij} is chosen randomly between 0 and 1. As seen in Fig. 2.2 for the range unconstrained case $\varrho \to \infty$, the agents (colored diamonds) can freely communicate from start and thereby directly find the best allocation (dashed lines), which is maintained all along the trajectory (plain lines).



(a) $t = 0^+$ (b) $t = 4$ (c) $t = 10$

Figure 2.2: Dynamic task allocation for the range unconstrained case ($n = p = 10$, $\varrho \to \infty$, $t_\mathsf{f} = 10$, $\mathcal{U} = 1.804$).

Conversely in Fig. 2.3 where the range ϱ is limited to 0.3, several agents are allocated to the same task because they are not in communication with all the other agents. These agents estimate their utility solely based on the reward of the task while their marginal utility is actually lower (see comparison in Table 2.1). When the agents come closer and enter in communication, the agents start assessing their marginal utility correctly and thus consider other tasks that might increase their own utility. Toward the end of the simulation (Fig. 2.3(c)), the agents' trajectory is subject to sharper changes of direction (e.g. the red and light blue curves) compared to the range unconstrained case (Fig. 2.2(c)).



(a) $t = 0^+$ (b) $t = 4$ (c) $t = 10$

Figure 2.3: Dynamic task allocation for the range constrained case ($n = p = 10$, $\varrho = 0.3$, $t_\mathsf{f} = 10$, $\mathcal{U} = 1.515$)

Range ϱ	Utility $\mathcal{U}$
0.3	1.515
∞	1.804

Table 2.1: Comparison of utilities for different communication ranges

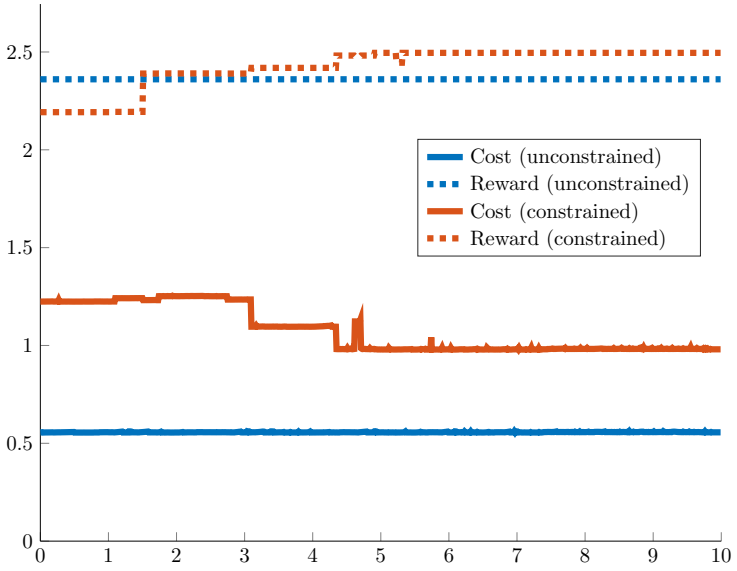


Figure 2.4: Total cost and reward.

Fig. 2.4 and 2.5 quantitatively show the allocation presented above, for which the data from the range unconstrained case (blue lines) are constant over time. When the communication is limited, the total utility increases step-by-step as the agents start communicating with their neighbors (red line in Fig. 2.5) but still remains lower than the utility obtained when the communication is not limited. It is worth noting that even though the final reward is higher for the constrained case (dashed red line in Fig. 2.4), the higher cost produced by abrupt trajectory changes makes its final utility lower than the utility generated without communication limitation. Finally, the noisy curves are due to approximation errors in the numerical integration.

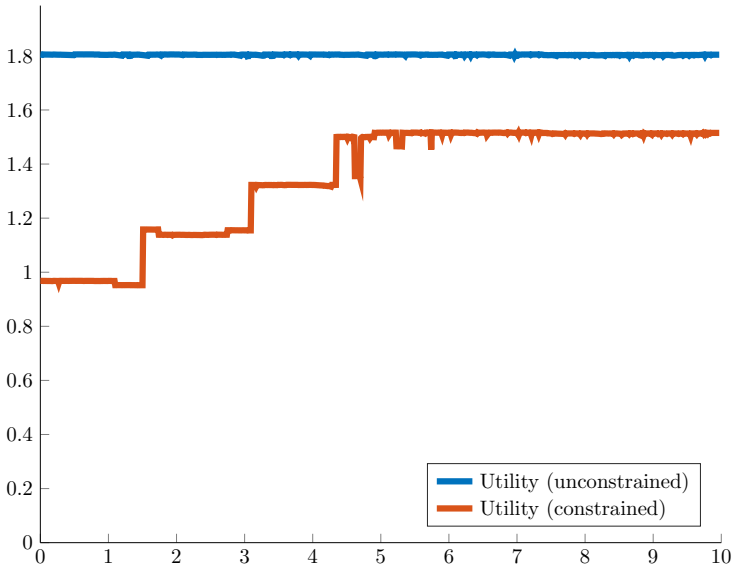


Figure 2.5: Total utility.

The impact of several parameters on the utility and the computation time is then performed (with 10 agents and 10 tasks if not mentioned). In

Fig. 2.6, the global utility increases progressively with the communication range. For a map width of 1, a communication range of $\sqrt{2}$ corresponds to the unconstrained communication.

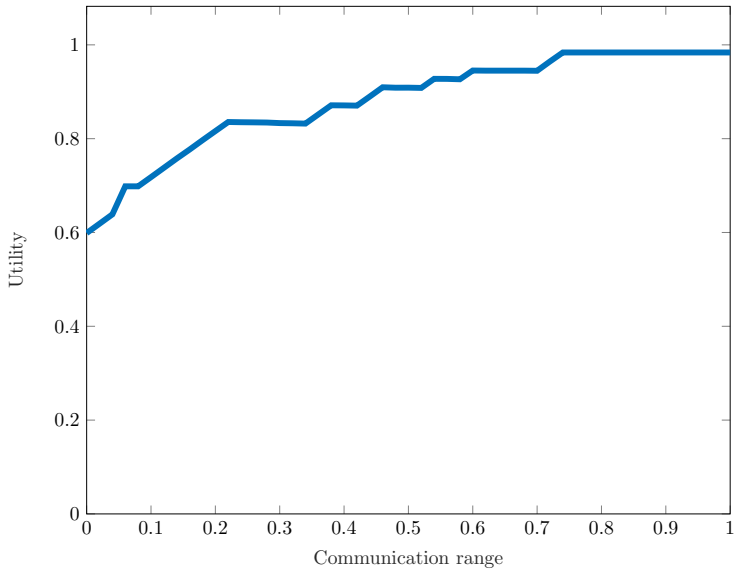


Figure 2.6: Impact analysis of the communication range on the utility.

In Fig. 2.7, we perform a simulation with 20 tasks and we progressively replace the tasks at a fixed location with loitering tasks. Since the number of numerical integrations is ten times higher for the latter, the computation time for 20 fixed tasks is approximately ten times higher than the one for 20 loitering tasks.

The effect of the number of agents / tasks on the computational cost is analyzed separately in Fig. 2.8. For a fixed number of agents, the global utility is linear with the number of tasks. It is however interesting to note that for a fixed number of tasks (e.g., 80 at the graph boundary), the utility

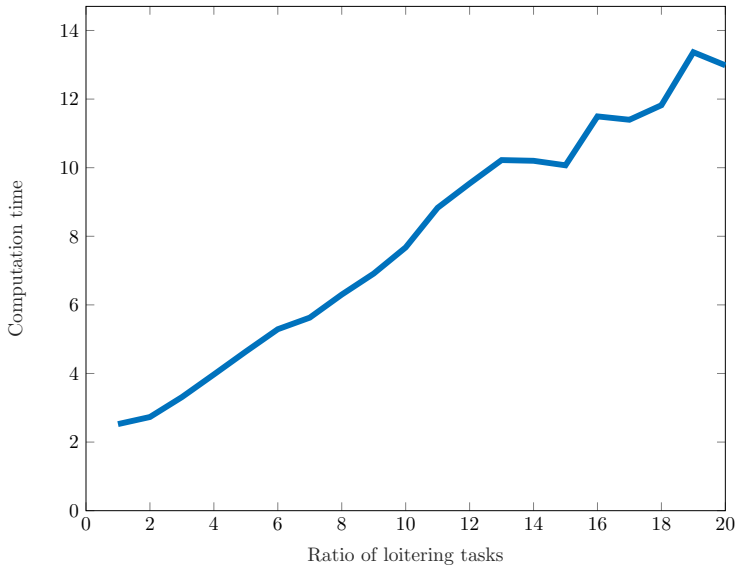


Figure 2.7: Impact analysis of the ratio of loitering tasks on the computation time.

increases exponentially with the number of agents. This shows that in this framework, multiplying the fleet of agents by $m \in \mathbb{R}$ will result in a global utility lower than $m\,\mathcal{U}$. This can be illustrated by considering one task $\mathcal{T}_1$ and n agents with probability p to complete the task, then the utility is given by $\bar{r}_{\mathcal{T}_1}[1-(1-p)^n]$.

Furthermore in Fig. 2.9, the computation time is more dependent on the number of tasks than the number of agents (the allocation with 1 task and 50 agents is straightforward while the allocation with 50 tasks and 1 agent requires the agent to iterate over all tasks). As a consequence, problems with a large number of agents are more tractable than problems with a high number of tasks. To mitigate this issue, one could for example restrict the

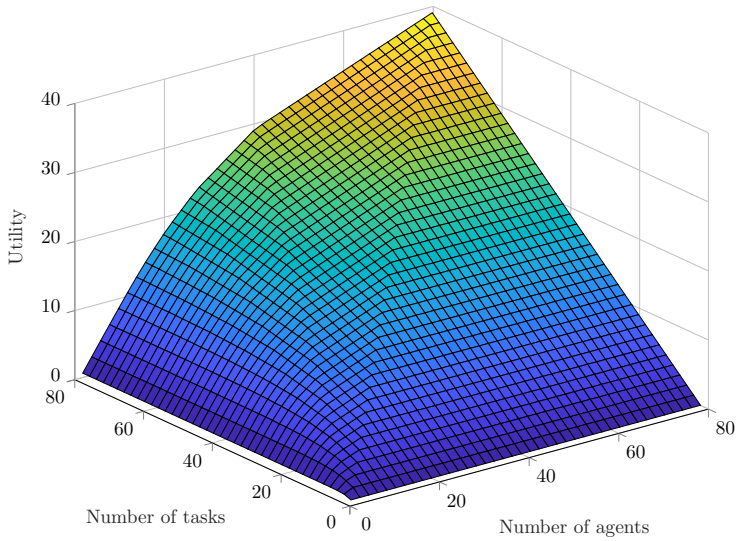


Figure 2.8: Impact analysis of the number of tasks / agents on the global utility.

considered tasks to the tasks near the agents (but at the cost of the utility, revealing a trade-off between the computation and the optimal allocation with the maximum utility).

In Fig. 2.10, the global utility naturally increases with the nominal reward and the success probability of a task.

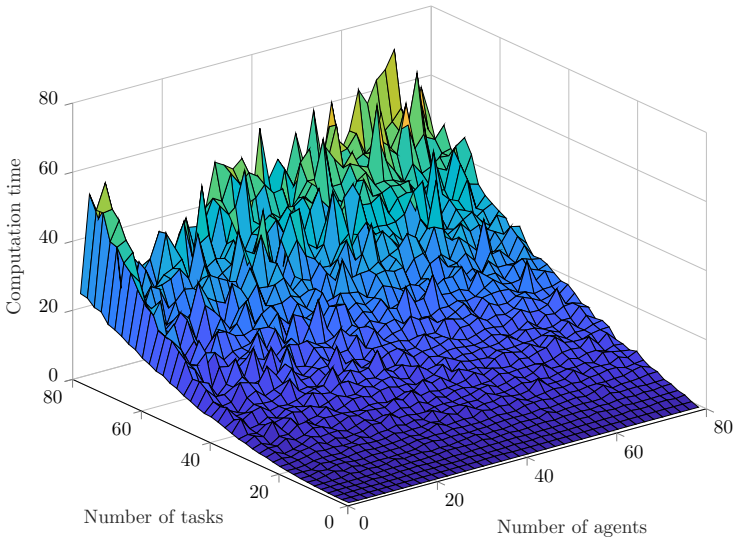


Figure 2.9: Impact analysis of the number of tasks / agents on the computation time.

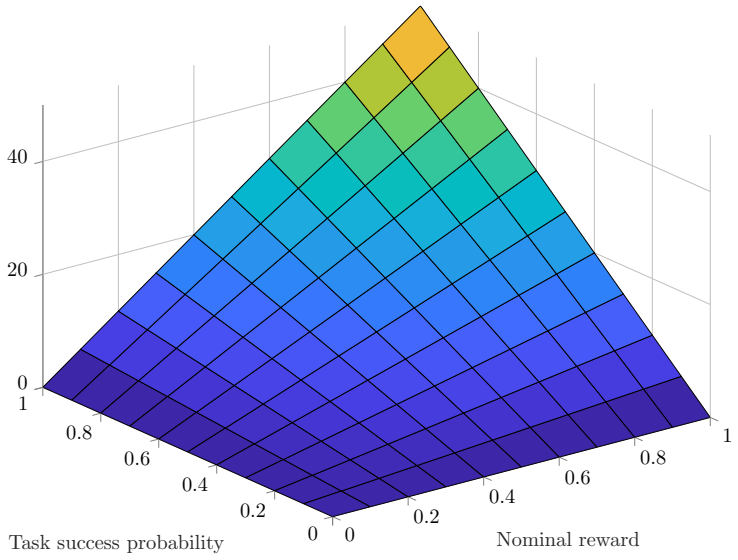


Figure 2.10: Nominal reward and task success probability.

Chapter 3

Obstacle Avoidance

Disclaimer: The material in this chapter is based on our paper submitted at MECC 2022 [53] (currently, under review).

3.1 Problem Setup

We consider an agent moving in an N-dimensional position space $\Omega \subseteq \mathbb{R}^N$ (where $N = 2$ or $N = 3$) which can be a bounded set. The state is defined as

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{P} \\ \boldsymbol{V} \end{pmatrix}$$

where $\boldsymbol{P} \in \Omega$ is the agent's position of the agent and $\boldsymbol{V} \in \mathbb{R}^N$ is the agent's velocity. We consider an obstacle whose center of mass has an initial position (at time $t = 0$) $\boldsymbol{P}_o \in \Omega$ and constant velocity $\boldsymbol{V}_o \in \mathbb{R}^N$, which implies that the obstacle motion is reduced to a straight line if $\boldsymbol{V}_o \neq 0$ (if $\boldsymbol{V}_o = 0$, the obstacle is static).

We also assume that the obstacle boundary χ^b can be parametrically described as a function of two dummy variables $u \in [0, 2\pi]$ and $v \in [0, \pi]$. In the case of an ellipse (2D), we have

$$\boldsymbol{P}^b(t; u) = \begin{pmatrix} a(t)\cos(u) \\ b(t)\sin(u) \end{pmatrix},$$

whereas in the case of an ellipsoid (3D), we have

$$\boldsymbol{P}^b(t;u,v) = \begin{pmatrix} a(t)\cos(u)\sin(v) \\ b(t)\sin(u)\sin(v) \\ c(t)\cos(v) \end{pmatrix}.$$

The velocity $\boldsymbol{V}_e^{\boldsymbol{P}}$ due to expansion/contraction at a point $\boldsymbol{P}^b \in \chi^b$, corresponding to a pair (u,v), can be described as

$$\boldsymbol{V}_e^{\boldsymbol{P}} = \frac{\partial \boldsymbol{P}^b(t;u,v)}{\partial t}.$$

The agent is subject to a double integrator dynamics;

$$\dot{\boldsymbol{x}}(t) = \begin{pmatrix} \boldsymbol{V}(t) \\ \boldsymbol{u}(t) \end{pmatrix}, \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 = \begin{pmatrix} \boldsymbol{P}_0 \\ \boldsymbol{V}_0 \end{pmatrix} \tag{3.1}$$

where $\boldsymbol{u}(t) \in \mathbb{R}^N$ is the control input.

The problem targeted by this paper can be described as follows.

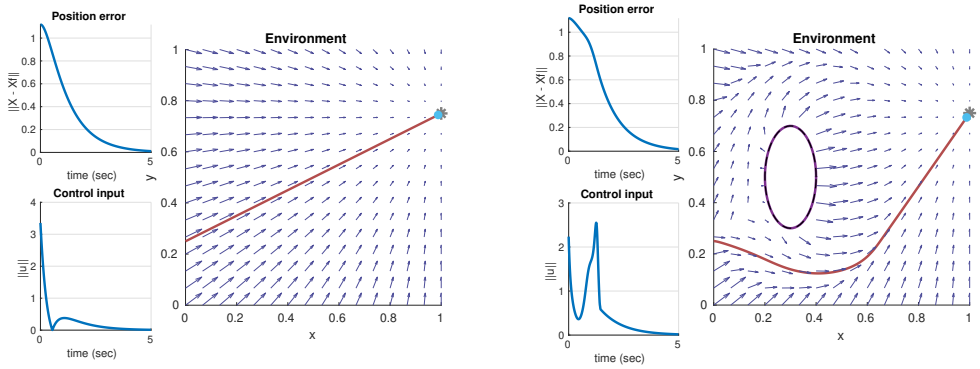
Problem 1. Let an agent be subject to the dynamics given in Eq. (3.1), Then, find a control input $\boldsymbol{u}(t) : [0, t_f] \to \mathbb{R}^N$ so that the agent reaches a final desired location $\boldsymbol{P}_f \in \Omega$ with zero velocity at a free final time t_f:

$$\boldsymbol{x}(t_f) = \boldsymbol{x}_f = \begin{pmatrix} \boldsymbol{P}_f \\ \boldsymbol{0} \end{pmatrix}.$$

3.2 CAFV design

The collision avoidance vector field (CAVF) is designed to match the desired agent's velocity along the state space. For example, the CAFV is directed towards the final location and is null at the final location. Because of obstacles appearing on the way from the agent's location to its final location,

the CAFV has to be modulated so that the agent does not enter into collision (see Fig. 3.1).



(a) Without obstacle.

(b) With obstacle.

Figure 3.1: In the absence of obstacle (a), the CAFV points towards $\boldsymbol{P}_f$. In the presence of obstacles (b), the CAFV is modulated to avoid the obstacle ($\boldsymbol{P}_0 = [0, 0.25]^\top$ and $\boldsymbol{P}_f = [1, 0.75]^\top$).

3.2.1 Static Obstacle

For a static obstacle, the collision avoidance vector field $\boldsymbol{h}$ has to satisfy the following impenetrability condition at any point $\boldsymbol{P}^b$ of the obstacle boundary:

$$\boldsymbol{h}(\boldsymbol{P}^b) \cdot \hat{\boldsymbol{n}}(\boldsymbol{P}^b) \geq 0 \qquad \text{for all } \boldsymbol{P}^b \in \chi^b. \tag{3.2}$$

where $\hat{\boldsymbol{n}}(\boldsymbol{P})$ is the unit vector that is normal to the obstacle boundary and is given by

$$\hat{\boldsymbol{n}}(\boldsymbol{P}) = \begin{cases} \frac{\boldsymbol{P} - \boldsymbol{P}^c(\boldsymbol{P})}{\|\boldsymbol{P} - \boldsymbol{P}^c(\boldsymbol{P})\|} & \text{if } \boldsymbol{P} \notin \chi^b \\ \frac{\boldsymbol{P}_l(\boldsymbol{P}) - \boldsymbol{P}}{\|\boldsymbol{P}_l(\boldsymbol{P}) - \boldsymbol{P}\|} & \text{if } \boldsymbol{P} \in \chi^b \end{cases}$$

where $\boldsymbol{P}^c(\boldsymbol{P})$ is the point on the obstacle boundary that is closest to $\boldsymbol{P}$, and $\boldsymbol{P}^l(\boldsymbol{P})$ is any point satisfying $\boldsymbol{P}^c(\boldsymbol{P}^l(\boldsymbol{P})) = \boldsymbol{P}$ (i.e., any point on the line

determined by $\boldsymbol{P} + \alpha\,\hat{\boldsymbol{n}}(\boldsymbol{P})$ for all $\alpha > 0$). The point $\boldsymbol{P}^c(\boldsymbol{P})$ can be computed via an iterative optimization algorithm ([54]). We define the vector field $\boldsymbol{h}$ as the superposition of an obstacle avoidance field $\boldsymbol{h}_o$ and a destination steering field $\boldsymbol{h}_f$ (to reach the aimed location); that is, $\boldsymbol{h}_o(\boldsymbol{P}) = \boldsymbol{h}_o(\boldsymbol{P}) + \boldsymbol{h}_f(\boldsymbol{P})$:

$$\boldsymbol{h}_o(\boldsymbol{P}) = \boldsymbol{R}(\boldsymbol{P})\ \|\boldsymbol{P}_f - \boldsymbol{P}\|^{-p}\ \gamma\,\|\boldsymbol{P}_f - \boldsymbol{P}\|\hat{\boldsymbol{n}},$$
$$\boldsymbol{h}_f(\boldsymbol{P}) = \boldsymbol{R}(\boldsymbol{P})\ \|\boldsymbol{P}_f - \boldsymbol{P}\|^{-p}\ (\boldsymbol{P}_f - \boldsymbol{P})$$

and hence

$$\boldsymbol{h}(\boldsymbol{P}) = \boldsymbol{R}(\boldsymbol{P})\ \|\boldsymbol{P}_f - \boldsymbol{P}\|^{-p}\Big[\gamma\,\|\boldsymbol{P}_f - \boldsymbol{P}\|\hat{\boldsymbol{n}} + (\boldsymbol{P}_f - \boldsymbol{P})\Big] \tag{3.3}$$

where $p \in]0, 1[$ is a modulation exponent which controls the difference of intensity across the vector field. We introduce a location-dependent factor $\gamma \in [0, 1]$ which is defined as follows:

$$\gamma(\boldsymbol{P}) = \frac{a_i\ x(\boldsymbol{P})}{\sqrt{1 + (2\ a_i\ x(\boldsymbol{P}))^2}} + \frac{1}{2}$$

where $x(\boldsymbol{P}) := (d(\boldsymbol{P}) + d_2(\boldsymbol{P}))/(d(\boldsymbol{P})\,d_2(\boldsymbol{P}))$, d_i is the influence distance of the obstacle (typically in the order of the obstacle semi-major axis), $d(\boldsymbol{P}) = \|\boldsymbol{P} - \boldsymbol{P}^c(\boldsymbol{P})\|$, $d_2(\boldsymbol{P}) = d(\boldsymbol{P}) - d_i$. In addition, a_i is a modulation constant to modify the shape of the sigmoid given by γ. This vector field $\boldsymbol{h}$ guarantees that $\boldsymbol{h}_o = \boldsymbol{0}$ when the agent is located at the influence distance since $d = d_i, x \to -\infty$ and hence $\gamma = 0$. Furthermore, $\boldsymbol{R}(\boldsymbol{P})$ is a rotation operator of angle α_R is applied to the vector field so that the agent turns around the obstacle when the obstacle is on the line of sight between the agent and the target position.

More formally, for $N = 2$, the matrix $\boldsymbol{R}$ at a point $\boldsymbol{P}$ is defined as

$$\boldsymbol{R}(\boldsymbol{P}) = \begin{bmatrix} \cos(\alpha_R(\boldsymbol{P})) & \sin(\alpha_R(\boldsymbol{P})) \\ -\sin(\alpha_R(\boldsymbol{P})) & \cos(\alpha_R(\boldsymbol{P})) \end{bmatrix}.$$

Let us define $\beta(\boldsymbol{P}) = \exp(-b_i\ x^2)$ where b_i is a fixed parameter. The rotation angle α_R is then computed proportionally to the angle between $\hat{\boldsymbol{n}}$ and $\boldsymbol{P}_f - \boldsymbol{P}_o$:

$$\alpha_R(\boldsymbol{P}) = \frac{\beta(\boldsymbol{P})}{2} \angle(\hat{\boldsymbol{n}}(\boldsymbol{P}), \boldsymbol{P}_f - \boldsymbol{P}_o)$$

where $\boldsymbol{P}_f - \boldsymbol{P}_o$ corresponds to the vector pointing from the obstacle center $\boldsymbol{P}_o$ to the final position $\boldsymbol{P}_f$. The factor β is introduced to ensure that the vector field is not rotated (that is, $\boldsymbol{R} = \mathbb{I}_N$ where $\boldsymbol{I}_N$ is the identity matrix of order N) when the agent is located at the influence distance of the obstacle (continuity of the vector field), and when the agent is located on the obstacle boundary (non-penetrability of the obstacle).

Now that the vector field has been described analytically, we can analyze two particular cases. The first scenario consists in the vectors $\hat{\boldsymbol{n}}$ and $\boldsymbol{P}_f - \boldsymbol{P}_o$ making an angle of π (singular case), corresponding to a point $\boldsymbol{P}$ exactly behind the obstacle with respect to the desired location. In this case, the field is rotated by an angle $\alpha \in [-\pi/2, \pi/2]$ so that the agent deviates from its initial trajectory to go around the obstacle (forming a singularity in the vector field). The second scenario appears when these two vectors $\hat{\boldsymbol{n}}$ and $\boldsymbol{P}_f - \boldsymbol{P}_o$ are aligned, that is, when the point $\boldsymbol{P}$ lies in between the obstacle and the target. In this case, the vector field is not rotated ($\alpha = 0$) since the agent is in front of the obstacle and can freely move to the target. The vector

field is illustrated in Fig. 3.2 where the target position is represented by a red cross.

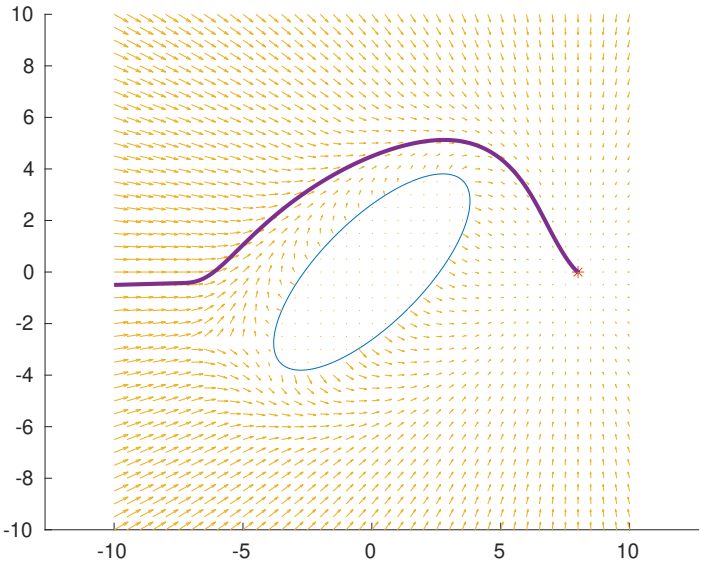


Figure 3.2: Vector field and associated path line for an initial agents location $\boldsymbol{P}_0 = [-10, 0]^\top$. The singularity appears around $[-5, -3]^\top$ (behind the obstacle).

Proposition 1. For a vector field defined in Eq. (3.3), the condition of impenetrability given in Eq. (3.2) is guaranteed for the case of a static obstacle.

Proof. When the agent is located at a point $\boldsymbol{P}^b \in \chi^b$ (on the boundary of the obstacle), we obtain $d = 0$, $x \to \infty$ and hence $\gamma = 1$. In addition, we have $\beta = 0$, $\alpha_R = 0$ and thus $\boldsymbol{R} = \boldsymbol{I}_N$. Given the particular value of the vector field

$$\boldsymbol{h}(\boldsymbol{P}^b) = \|\boldsymbol{P}_f - \boldsymbol{P}\|^{-p}\Big[\|\boldsymbol{P}_f - \boldsymbol{P}\|\hat{\boldsymbol{n}} + (\boldsymbol{P}_f - \boldsymbol{P})\Big],$$

we obtain

$$\begin{aligned}\boldsymbol{h}(\boldsymbol{P}^b)\cdot\hat{\boldsymbol{n}} &= \|\boldsymbol{P}_f-\boldsymbol{P}\|^{-p}\Big[\|\boldsymbol{P}_f-\boldsymbol{P}\|+(\boldsymbol{P}_f-\boldsymbol{P})\cdot\hat{\boldsymbol{n}}\Big]\\ &= \|\boldsymbol{P}_f-\boldsymbol{P}\|^{-p}\Big[\|\boldsymbol{P}_f-\boldsymbol{P}\|+\underbrace{(\boldsymbol{P}_f-\boldsymbol{P})\cdot\hat{\boldsymbol{n}}}_{\geq-\|\boldsymbol{P}_f-\boldsymbol{P}\|}\Big]\\ &\geq 0,\end{aligned}$$

which verifies Eq. (3.2) of impenetrability and thus concludes the proof. □

3.2.2 Moving Obstacle

The velocity $\boldsymbol{V}_b$ of the obstacle boundary at $\boldsymbol{P}^b$ can be the resulting of multiple contributing terms. This paper will examine 2 movements: constant velocity translation ($\boldsymbol{V}_o$) and expansion/contraction[1] of the boundary in time ($\boldsymbol{V}_e^{\boldsymbol{P}}$) so that $\boldsymbol{V}_b = \boldsymbol{V}_o + \boldsymbol{V}_e^{\boldsymbol{P}}$. Additionally, we consider the obstacle's velocity in the normal direction of the obstacle boundary only if it is greater than zero ($\boldsymbol{V}_b\cdot\hat{\boldsymbol{n}} > 0$) so that the agent is not attracted to the obstacle when it is moving away from it.

To account for the fact that the obstacle is moving, we introduce an additional term $\boldsymbol{h}_v(\boldsymbol{P}) = \gamma\boldsymbol{V}_b$ to the vector field such that the total vector field $\boldsymbol{h}_m(\boldsymbol{P})$ for a moving obstacle is given by

$$\boldsymbol{h}_m(\boldsymbol{P}) = \boldsymbol{h}(\boldsymbol{P}) + \boldsymbol{h}_v(\boldsymbol{P}). \tag{3.4}$$

[1]This scenario also considers an obstacle whose state is not precisely known by the agent. To this end, the state $\boldsymbol{x}_o$ of the obstacle is represented with a Gaussian density function characterized by its mean $\bar{\boldsymbol{x}}_o$ and covariance matrix $\boldsymbol{\Sigma}_o$. When the obstacle is moving, the uncertainty (i.e., the covariance matrix $\boldsymbol{\Sigma}_o$) is growing with time, resulting in a positive velocity $\boldsymbol{V}_e^{\boldsymbol{P}}$.

When the obstacles are moving, the impenetrability condition requires that the component of the vector field normal to the boundary has to satisfy the following inequality:

$$\boldsymbol{h}_m(\boldsymbol{P}^b) \cdot \hat{\boldsymbol{n}}(\boldsymbol{P}^b) \geq \boldsymbol{V}_b \cdot \hat{\boldsymbol{n}}(\boldsymbol{P}^b) \qquad \text{for all } \boldsymbol{P}^b \in \chi^b. \tag{3.5}$$

Proposition 2. For a vector field defined in Eq. (3.4), the condition of impenetrability given in Eq. (3.5) is guaranteed for an obstacle moving at constant velocity $\boldsymbol{V}_b$.

Proof. When the agent is located at the boundary $\boldsymbol{P}^b$ of the obstacle, we have $\gamma = 1$. We conclude that

$$\begin{aligned} \boldsymbol{h}_m(\boldsymbol{P}^b) \cdot \hat{\boldsymbol{n}}(\boldsymbol{P}^b) &= \underbrace{\boldsymbol{h}(\boldsymbol{P}^b) \cdot \hat{\boldsymbol{n}}(\boldsymbol{P}^b)}_{\geq 0 \text{ from Eq. (3.2)}} + \boldsymbol{V}_b \cdot \hat{\boldsymbol{n}}(\boldsymbol{P}^b) \\ &\geq \boldsymbol{V}_b \cdot \hat{\boldsymbol{n}}(\boldsymbol{P}^b), \end{aligned}$$

which verifies Eq. (3.5) of impenetrability and thus concludes the proof. □

In addition, $\boldsymbol{h}_v(\boldsymbol{P}) = \boldsymbol{0}$ when the agent is located at the influence distance ($d = d_i$) so that $\boldsymbol{h}(\boldsymbol{P}) = \boldsymbol{h}_f(\boldsymbol{P})$ as if there is no obstacle.

3.2.3 Multiple Obstacles

Next, we consider the case of multiple obstacles. Let $d_j = \|\boldsymbol{P} - \boldsymbol{P}_j^c(\boldsymbol{P})\|$ be the distance between $\boldsymbol{P}$ and $\boldsymbol{P}_j^c(\boldsymbol{P})$, where $\boldsymbol{P}_j^c(\boldsymbol{P})$ is the point on the boundary of obstacle j that is closest to $\boldsymbol{P}$. When there are M obstacles, the vector field is extended by taking the sum of the local CAVFs weighted by their

distance to the agent's position $\boldsymbol{P}$ so that the vector field at the boundary of obstacle i is only determined by the local vector field $\boldsymbol{h}_i(\boldsymbol{P})$ associated to this obstacle:

$$\boldsymbol{h}(\boldsymbol{P}) = \sum_{i=1}^{M} w_i(\boldsymbol{P})\, \boldsymbol{h}_i(\boldsymbol{P}), \tag{3.6}$$

where

$$w_i(\boldsymbol{P}) = \frac{\prod_{j\neq i}^{M} d_j}{\sum_{i=1}^{M} \prod_{j\neq i}^{M} d_j} \quad \text{and hence} \quad \sum_{i=1}^{M} w_i(\boldsymbol{P}) = 1.$$

Proposition 3. For a vector field defined in Eq. (3.6), the condition of impenetrability given in Eq. (3.2) is guaranteed for multiple obstacles.

Proof. When the agent is located at the boundary $\boldsymbol{P}_j^b$ of obstacle j, we obtain $w_i(\boldsymbol{P}_j^b) = 1$ for $i = j$ and $w_i(\boldsymbol{P}_j^b) = 0$ for $i \neq j$. The resulting vector field is thus given by $\boldsymbol{h}(\boldsymbol{P}_j^b) = \boldsymbol{h}_j(\boldsymbol{P}_j^b)$. The impenetrability condition for multiple obstacles is therefore guaranteed since the impenetrability condition at the boundary of a single obstacle has already been proven in Proposition 1. □

3.2.4 Bounded Environment

We now consider an environment that is a convex polygonal set such that the agent is not allowed to exit through its boundaries (for instance Ω is a rectangular domain in 2D or a parallelepiped in 3D). This extension is implemented by considering each edge as a flat ellipsoidal obstacle. More formally, for an ellipse whose semi-axis lengths are given by a and b, we consider a large semi-major over semi-minor length ratio: $a/b \gg 1$.

3.3 Control Law

As mentioned earlier, we consider a double integrator dynamics given in Eq. (3.1). Since the vector field $\boldsymbol{h}$ reflects the desired agent's velocity $\boldsymbol{V}(t)$, we aim to find a control law providing a link between $\dot{\boldsymbol{h}}$ and $\dot{\boldsymbol{V}}(t) = \boldsymbol{u}(t)$. We therefore consider a control law mixing the proportional gain and the derivative of the vector field:

$$\boldsymbol{u} = k_p(\boldsymbol{h} - \boldsymbol{V}) + k_v\dot{\boldsymbol{h}}$$

where $k_p \in \mathbb{R}$ is the proportional gain, $k_v \in \mathbb{R}$ is the derivative gain and $\dot{\boldsymbol{h}} = \nabla\boldsymbol{h} \cdot \boldsymbol{V}$ following the chain rule. The agent will only follow the CAVF (velocity vectors) if the initial velocity matches the velocity of the vector field at the initial position. For example when $\boldsymbol{V} = \boldsymbol{0}$ at start, we have $\dot{\boldsymbol{h}} = \nabla\boldsymbol{h} \cdot \boldsymbol{V} = \boldsymbol{0}$ as well. If we use the simple control law $\boldsymbol{u} = \dot{\boldsymbol{h}}$, we would obtain $\boldsymbol{u} = \boldsymbol{0}$ and the agent would never start. We thus also need a proportional controller at start to launch the agent and make it match the desired velocity. Once the desired speed is similar to the local vector field ($\boldsymbol{h} \simeq \boldsymbol{V}$), the second term of the proposed control law based on $\dot{\boldsymbol{h}}$ takes the lead, taking into account not only the value of $\boldsymbol{h}$ at the agent's location but also the variation $\nabla\boldsymbol{h}$ of the vector field around this location to better predict the best trajectory. This gradient can be computed numerically via the symmetric difference quotient. In addition, gravity compensation can be included according to $\boldsymbol{u}_g = \boldsymbol{u} + \boldsymbol{g}$ provided that the value $\boldsymbol{g}$ of the gravity is perfectly known.

3.3.1 Convergence to Target State

Convergence to the target state is proven below in the simple case of an environment which is not (locally) populated with obstacles.

Theorem 3.3.1. *Consider an agent subject to the collision avoidance problem (Problem 1) and an environment which is not (locally) populated with obstacles, the algorithm presented in Section 3.3 guarantees that the agent reaches the desired final state with a given maximum error δ after a finite time t_f: $\boldsymbol{P}(t_f) = \boldsymbol{P}_f$ and $\boldsymbol{V}(t_f) = \boldsymbol{0}$.*

Proof. Since the vector field is null only at the boundaries of the obstacles (which are not reachable due to the impenetrability condition), the vector field does not have any reachable local minimum. □

Remark 3 The above theorem proves that the agent is guaranteed to reach a sphere around the desired final location in a finite time. In the case of multiple obstacles, the final time cannot be directly computed.

Characterizing the value of δ in environments populated with obstacles is not trivial. However we now restrict the problem to the particular case of an obstacle-free environment to derive the error δ in function of the desired final time t_f. We also consider that the tracking of the vector field is well established so that $\boldsymbol{V}(\boldsymbol{P}) = \boldsymbol{h}(\boldsymbol{P})$ since the proportional gain in the control law provides a convergence of $\boldsymbol{V}$ to the vector field if they locally differ.

The dynamics is hence given by $\dot{\boldsymbol{P}} = \boldsymbol{P}_f - \boldsymbol{P}$, $\boldsymbol{P}(0) = \boldsymbol{P}_0$ which results in

$$\boldsymbol{P}(t) = (\boldsymbol{P}_0 - \boldsymbol{P}_f)\exp(-t) + \boldsymbol{P}_f$$

The error is

$$e(t) = \boldsymbol{P}(t) - \boldsymbol{P}_f = (\boldsymbol{P}_0 - \boldsymbol{P}_f)\exp(-t)$$

from which it follows:

$$\|e(t)\| = \|\boldsymbol{P}_0 - \boldsymbol{P}_f\|\exp(-t).$$

We conclude that the error indeed converges as $t \to \infty$. More precisely, the agent is guaranteed to lie inside a sphere of radius δ after a time t_f such that

$$\delta = \|\boldsymbol{P}_0 - \boldsymbol{P}_f\|\exp(-t_f) \Longrightarrow t_f = \ln\left(\frac{\|\boldsymbol{P}_0 - \boldsymbol{P}_f\|}{\delta}\right).$$

3.3.2 Norm-Constrained Input Control

When the input is constrained, the agent might not be able to escape an obstacle if its speed is too high near the obstacle. We constrain the input control such that $\boldsymbol{u} \in \mathcal{U} = \{\boldsymbol{u} \in \mathbb{R}^N : \|\boldsymbol{u}\| \leq \bar{u}\}$ where $\bar{u} \in \mathbb{R}^+$. The time to stop the agent at velocity $\boldsymbol{V}_0$ is then $t = \|\boldsymbol{V}_0\|/\bar{u}$. Based on the motion equation $x(t) = -\bar{u}t^2/2 + V_0 t$, the distance that has been achieved in this time is given by

$$R = \frac{V_0^2}{2\bar{u}}$$

where V_0 is the radial velocity of the agent towards the obstacle. We thus inflate each obstacle by the distance R so that the agent adjusts its trajectory to not lie inside the inflated boundary of the obstacle.

When the obstacle is moving, the velocity of the agent in the obstacle frame is given by $V_0 - V_b$ where V_b is the velocity at the obstacle boundary along the line of sight towards the agent (composed of the obstacle velocity and the expansion velocity). If the agent lies inside this inflated boundary, obstacle avoidance is still possible but it is not guaranteed.

3.4 Numerical Simulations

The following simulations are computed in 2D and 3D scenarios for a system with double integrator dynamics. The agent is considered to be a particle (point mass) and all the simulations have been realized with the following parameters: $d_i = 0.3, a_i = 0.01$ and $p = 1/2$. The video presenting all the cases presented in this section is available at `https://youtu.be/xSIyvFngA1M`. The source code used to produce the paper is available at `https://github.com/MartinBraquet/vector-field-obstacle-avoidance`.

3.4.1 Static Obstacles

The CAVF design for 2D static obstacles has been presented in Section 3.2 (see Fig. 3.1). Fig. 3.3 presents the motion of an agent in a 3D space, moving around an ellipsoid and reaching the desired final location.

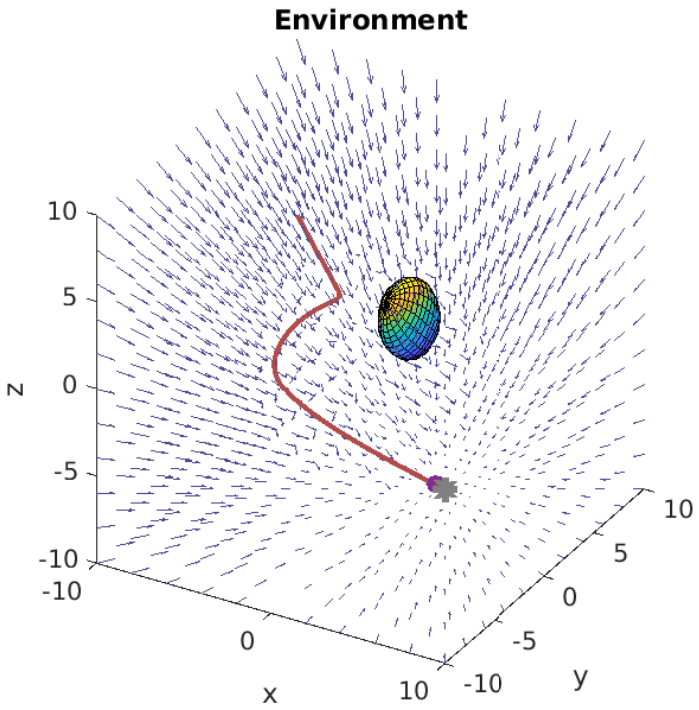


Figure 3.3: 3D obstacle avoidance with ellipsoid ($\boldsymbol{P}_0 = [-10, 10, 0]^\top$ and $\boldsymbol{P}_f = [10, -10, 0]^\top$).

3.4.2 Moving Obstacles

In Fig. 3.4, the elliptical obstacle is located in the bottom-right quadrant at start ($\boldsymbol{P}_o = [0.6, 0.2]^\top$) and is moving upwards at constant velocity (dashed lines). The agent starts avoiding the obstacle by going above the ellipse. Since the agent is moving with a similar speed as the obstacle, he then decides to slow down and let the obstacle move upwards so that he reaches his target location by going below the obstacle.

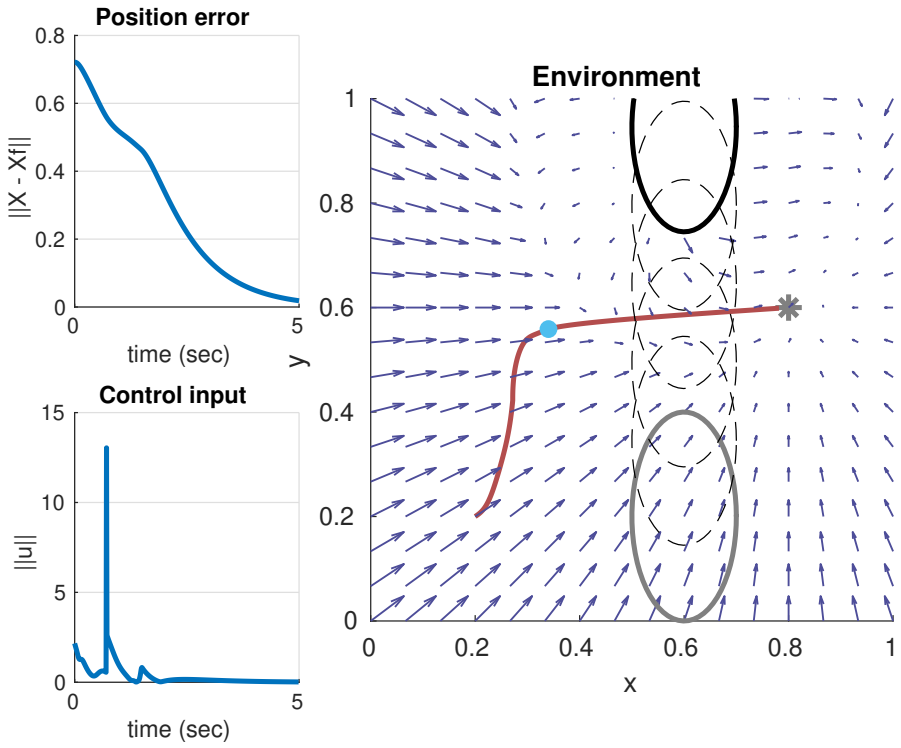


Figure 3.4: 2D obstacle avoidance with a moving ellipse represented in plain gray (initial position), dashed black (intermediate positions) and plain black (final position) ($\boldsymbol{P}_0 = [0.2, 0.2]^\top$ and $\boldsymbol{P}_f = [0.8, 0.6]^\top$).

Additionally, moving obstacles lead to growing uncertainty in probabilistic motion. As depicted in the video whose link is given at the start of the section, we consider scenarios where such uncertainty is represented by an ellipse growing with time.

3.4.3 Multiple Obstacles

As depicted in Fig. 3.5, the agent is moving to his target by slaloming between three ellipses on his way. The norm of the position error, given by $\|e(t)\| = \|\boldsymbol{P}(t) - \boldsymbol{P}_f\|$, is decreasing exponentially. The norm of the control input $u(t)$, which is assumed unbounded in this simulation is high at the beginning since the initial zero velocity does not match the desired velocity of the vector field at the initial agent's location.

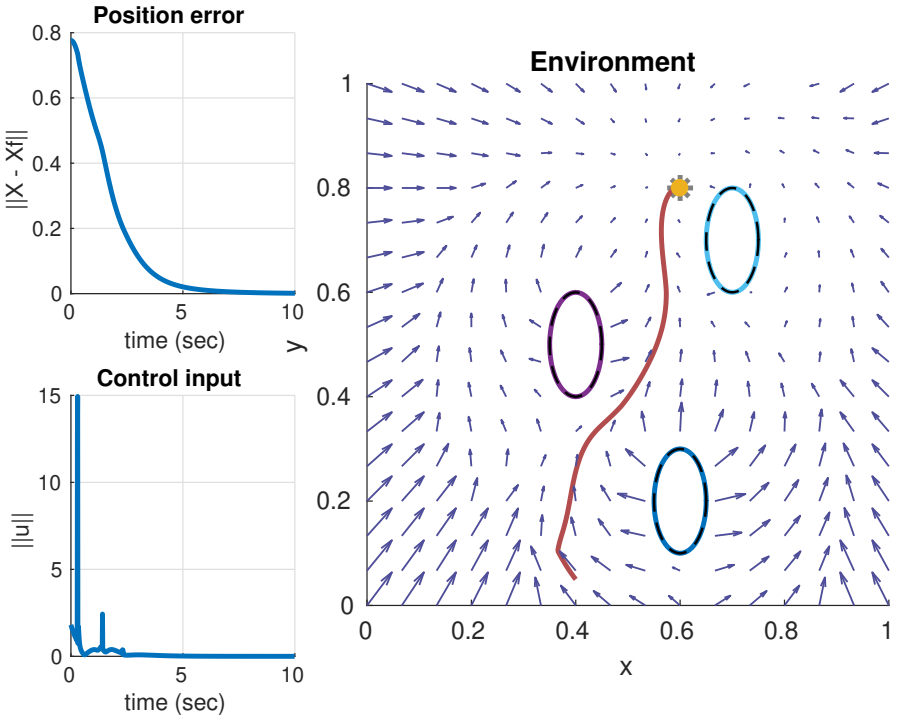


Figure 3.5: 2D obstacle avoidance with multiple ellipses ($\boldsymbol{P}_0 = [0.4, 0.05]^\top$ and $\boldsymbol{P}_f = [0.6, 0.8]^\top$).

The 3D case with multiple ellipsoids is presented in Fig. 3.6. It is

worth noting that in this scenario the agent is correctly progressing towards the target by going around the obstacles.

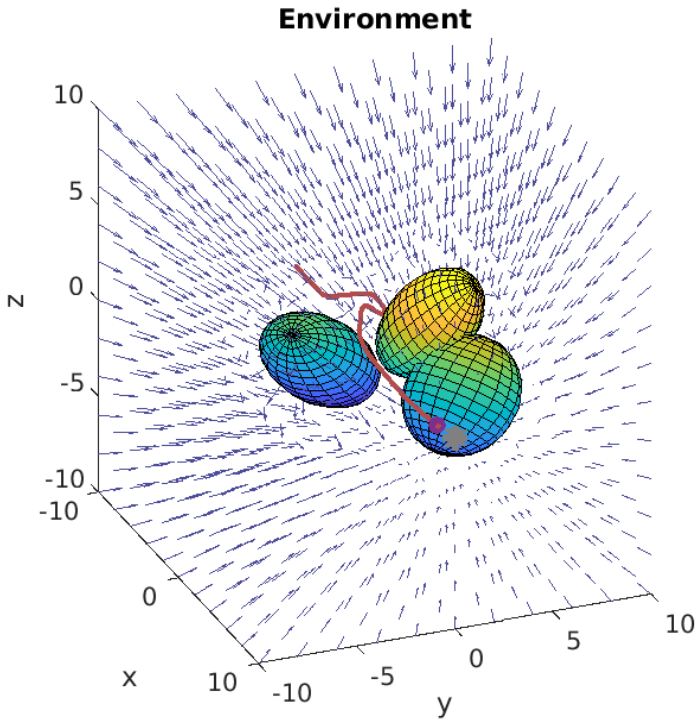


Figure 3.6: 3D obstacle avoidance with multiple ellipsoids.

3.4.4 Bounded Environment

As shown in Fig. 3.7, the environment is a square domain, that is, bounded by four walls, and an ellipse is located at the center. The vector field is directed both outward around the ellipse and inward next to the walls. The agent is thus guaranteed to stay in the desired area and he is correctly moving towards his target.

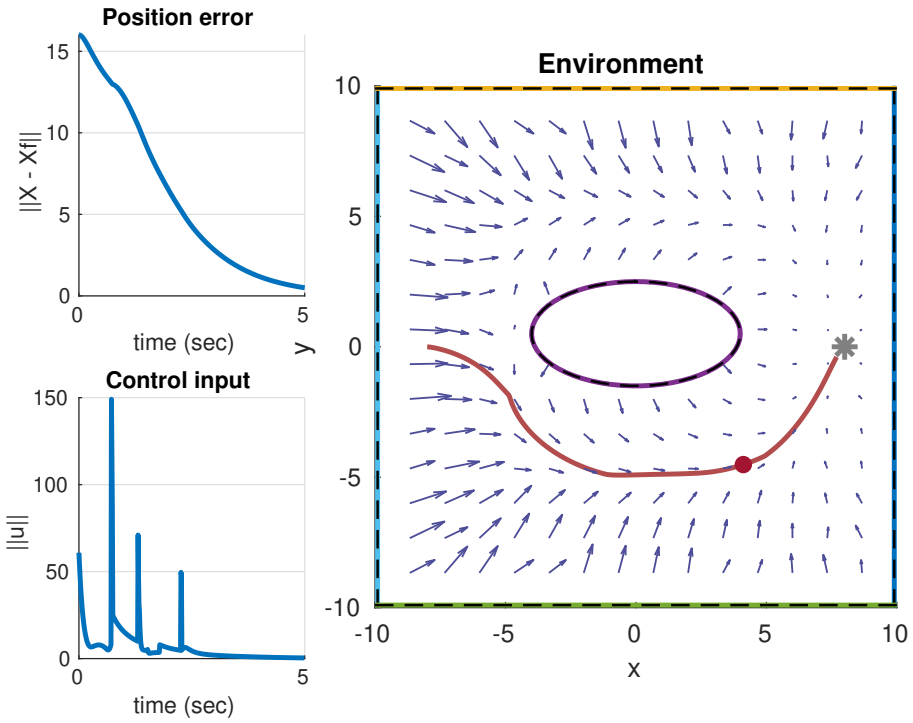


Figure 3.7: 2D obstacle avoidance with an ellipse in a bounded environment ($\boldsymbol{P}_0 = [-8, 0]^\top$ and $\boldsymbol{P}_f = [8, 0]^\top$).

3.4.5 Norm-Constrained Input Control

We finally consider the case where the control input is constrained such that $\|\boldsymbol{u}(t)\| \in \mathcal{U}$ for all times $t \geq 0$. For the simulations shown in Fig. 3.8, we set the maximum norm of the control input to be $\bar{u} = 1\,\mathrm{m/s^2}$. The black dashed lines represent the inflated boundary of the obstacles depending on the speed of the agent. It should be noted that the obstacle in the top-right quadrant is not inflated since the agent, located at $\boldsymbol{P} = [0.5, 0.5]^\top$ in the figure, is not in its area of influence. We can notice that the agent is correctly

reaching his target while his control input has been capped twice.

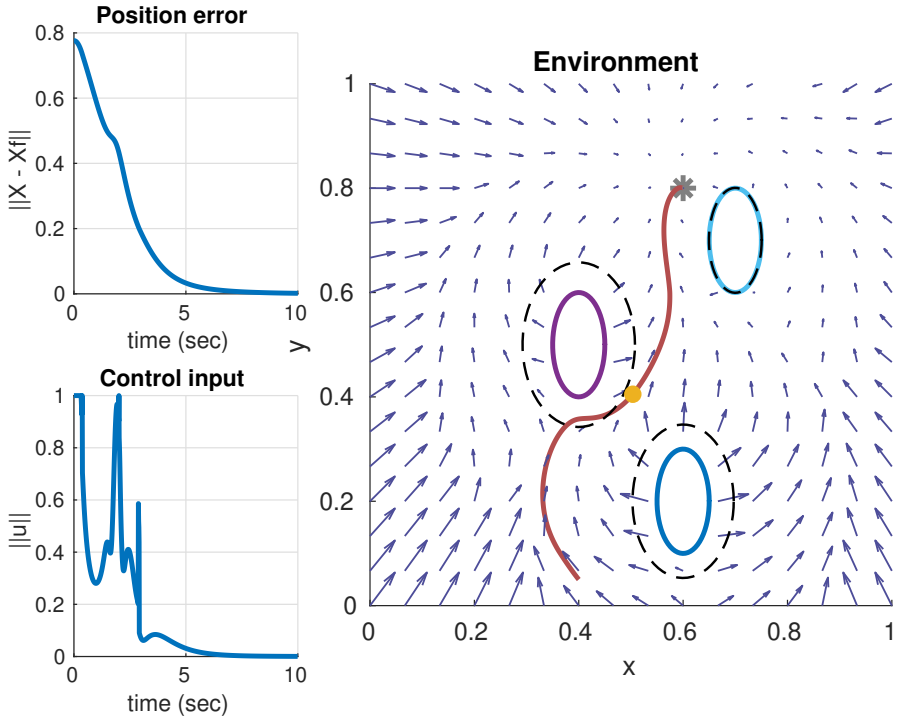


Figure 3.8: 2D obstacle avoidance with constrained input control ($\boldsymbol{P}_0 = [-8, 0]^\top$ and $\boldsymbol{P}_f = [8, 0]^\top$).

3.4.6 Task Allocation with CAVF

As defined in Section 2.1, we set the state domains $\Sigma := \mathbb{R}^{2N}$ and $\mathcal{S}_i := \Omega$. The motion of each agent is described by a double integrator dynamics given in Eq. (3.1). Each task $\mathcal{T}_i$ is characterized by a desired final agent's state $\boldsymbol{x}_{\mathcal{T}_i} = (\boldsymbol{P}_{\mathcal{T}_i}, \boldsymbol{0})$. The problem is then given as follows.

Problem 2. Find a time-varying task assignment profile that maximizes the global utility in a decentralized way (communication constraints) steering the

MAS to their desired task locations and using a collision avoidance vector field to compute the desired agents' trajectory and their associated motion cost. Additionally, each task needs to be allocated by at least one agent if $n \geq p$.

The task assignment is achieved via the Greedy Coalition Auction Algorithm defined in Section 2.3.4 [50]. In the context of this chapter, the control effort of agent i to reach a task is given by (half of) the square of the control input over time: $\mathcal{J}(u_i(\cdot)) := (1/2)\int_0^{t_f} |u_i(t)|^2 \mathrm{d}t$. As a consequence, the completion cost between each agent and each task is obtained by numerically computing the control input $\boldsymbol{u}_i(t)$ according to the vector field detailed in Section 3.2.

To illustrate the results, Figure 3.9 presents a task assignments with $n = 5$ agents and $p = 3$ tasks. As required, all the tasks are allocated and the agents safely avoid the elliptical obstacles. The video describing other examples of task allocation is available at `https://youtu.be/vixDPivkg1s`.

In this work, the completion time varies with the intensity of the vector field. Future work requiring a fixed terminal time (whether it be identical or different for each agent) can be achieved by deriving a relation between the intensity of the vector field and the final time (e.g. assuming a quasi-inverse relationship[2] or by iteration over the trajectory). This final time can

[2]Assuming a perfect controller (that is, $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{V}(\boldsymbol{x})$), the vector field is an image of the current agent's velocity. Since the completion time is given by $t_f = \int_{\boldsymbol{x}_0}^{\boldsymbol{x}_f} \frac{\mathrm{d}x}{\boldsymbol{V}(\boldsymbol{x})}$, the initial / final positions are fixed and the velocity is only space-dependent, multiplying the velocity

be bounded below depending on the agent's maximum velocity and acceleration, and bounded above at the convenience of the operator depending on the urgency of the task. It is worth mentioning that these considerations introduce a framework for *task scheduling*.

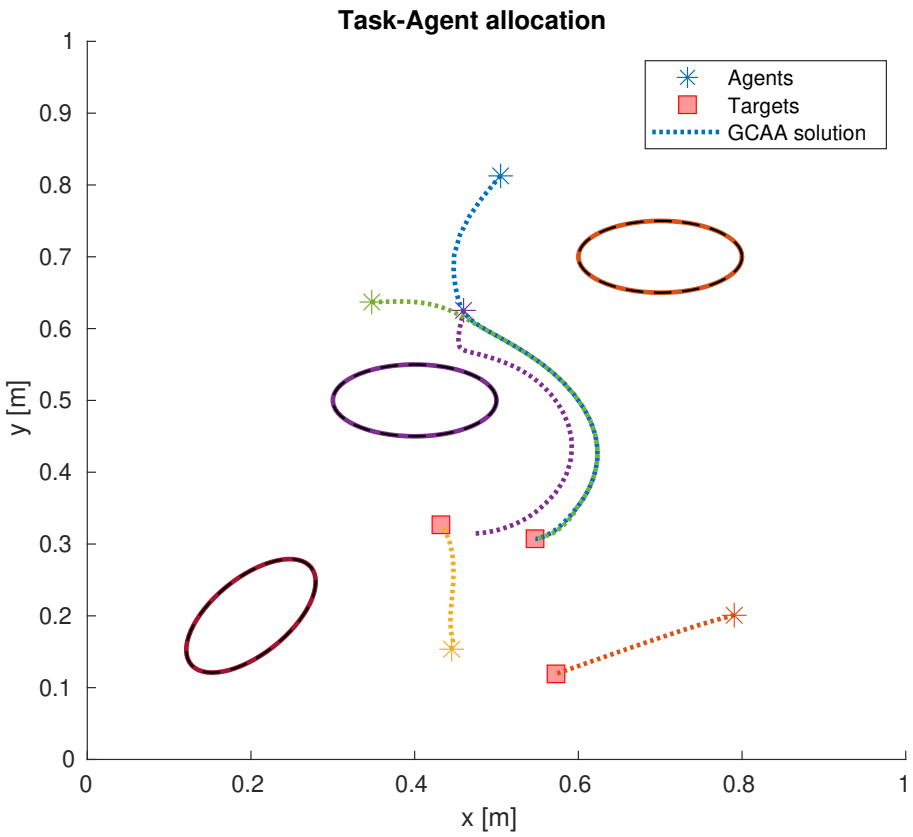


Figure 3.9: 2D task allocation with obstacle avoidance ($n = 5$ agents and $p = 3$ tasks).

by a factor k will divide the final time by k as well.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

Firstly, we have presented an auction-based framework to address dynamic task allocation problems for multi-agent systems with state-dependent utilities and various task characteristics (such as terminal constraints, completion time, etc.). Our greedy approach offers a practical, yet efficient, solution to a class of more realistic and challenging dynamic task allocation problems for autonomous mobile agents.

Secondly, we have addressed the problem of moving an agent in an environment populated with static and dynamic elliptical obstacles whose shape, position and velocity can change with time. The resulting algorithm is based on a vector field designed to provide information about the desired local agent's velocity and steer the system to a desired target under a double integrator dynamics. Our work has been extended to include moving obstacles, bounded environments and limited control inputs. Next, this vector field has been merged with a task allocation algorithm to incorporate multi-agent settings.

4.2 Future Work

For large fleets of autonomous systems, some scalability issues arise in the task allocation algorithm due to the computation time. Further research can focus more on machine learning and its recent achievements (specifically deep reinforcement learning for multi-agent robots) to mitigate this issue and build upon the algorithm presented in this work. We further plan to extend the results presented herein to even more realistic task allocation problems with deadlines, logical constraints, pop-up tasks and agents with varying capabilities and preferences.

Further research can be also done to extend the work about obstacle avoidance to obstacles of any convex shape, and then to obstacles with non-convex shapes. Moreover, additional research about vector field-based algorithms can extend this work, whose applicability is limited to agents with double integrator kinematics, to problems with more general dynamics. Finally, the task allocation can be improved to avoid the computation of the control cost for each agent and task, hence reducing the computation time and producing an algorithm more suitable for real-time systems.

Bibliography

[1] B. P. Gerkey and M. J. Mataric, "Sold!: Auction methods for multirobot coordination," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 758–768, 2002.

[2] H. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[3] M. Nanjanath and M. Gini, "Repeated auctions for robust task execution by a robot team," *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 900 – 909, 2010.

[4] J. Capitan, M. T. Spaan, L. Merino, and A. Ollero, "Decentralized multi-robot cooperation with auctioned POMDPs," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 650–671, 2013.

[5] S. Phillips and F. Parra, "A case study on auction-based task allocation algorithms in multi-satellite systems," *AIAA Scitech 2021 Forum*, 2021.

[6] S. Hayat, E. Yanmaz, C. Bettstetter, and T. X. Brown, "Multi-objective drone path planning for search and rescue with quality-of-service requirements," *Autonomous Robots*, vol. 44, no. 7, pp. 1183–1198, 2020.

[7] G. Qu, D. Brown, and N. Li, "Distributed greedy algorithm for multi-agent task assignment problem with submodular utility functions," *Automatica*, vol. 105, pp. 206 – 215, 2019.

[8] K.-S. Kim, H.-Y. Kim, and H.-L. Choi, "Minimizing communications in decentralized greedy task allocation," *Journal of Aerospace Information Systems*, vol. 16, pp. 1–6, 06 2019.

[9] H.-S. Shin, T. Li, and P. Segui-Gasco, "Sample greedy based task allocation for multiple robot systems," *arXiv preprint arXiv:1901.03258*, 2019.

[10] S. Rahili, B. Riviere, and S.-J. Chung, "Distributed adaptive reinforcement learning: A method for optimal routing," *arXiv preprint arXiv:2005.01976*, 2020.

[11] L. B. Johnson, S. S. Ponda, H. Choi, and J. How, "Asynchronous decentralized task allocation for dynamic environments," *Infotech@Aerospace 2011*, 2011.

[12] L. Luo, N. Chakraborty, and K. Sycara, "Competitive analysis of repeated greedy auction algorithm for online multi-robot task assignment," in *2012 IEEE International Conference on Robotics and Automation*, pp. 4792–4799, May 2012.

[13] H. Ravichandar, K. Shaw, and S. Chernova, "Strata: A unified framework for task assignments in large teams of heterogeneous robots," *Autonomous Agents and Multi-Agent Systems*, vol. 34, p. 38, 03 2019.

[14] D.-H. Lee, "Resource-based task allocation for multi-robot systems," *Robotics and Autonomous Systems*, vol. 103, pp. 151–161, 2018.

[15] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[16] A. Whitbrook, Q. Meng, and P. W. H. Chung, "Addressing robustness in time-critical, distributed, task allocation algorithms," *Applied intelligence*, vol. 49, no. 1, pp. 1–15, 2019.

[17] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multi-robot task allocation in communication limited environments," *Autonomous Robots*, vol. 44, no. 3, pp. 547–584, 2020.

[18] G. Arslan, J. R. Marden, and J. S. Shamma, "Autonomous Vehicle-Target Assignment: A Game-Theoretical Formulation," *J. Dyn. Syst. Meas. Control*, vol. 129, pp. 584–596, 04 2007.

[19] E. Bakolas and Y. Lee, "Decentralized game-theoretic control for dynamic task allocation problems for multi-agent systems," in *2021 American Control Conference (ACC)*, 2021.

[20] S. Bakshi, T. Feng, Z. Yan, and D. Chen, "A regularized quadratic programming approach to real-time scheduling of autonomous mobile robots in a prioritized task space," *2019 American Control Conference (ACC)*, pp. 1361–1366, 2019.

[21] P. Gautier, L. D. Johann, and J.-P. Diguet, "Comparison of Market-based and DQN methods for Multi-Robot processing Task Allocation (MRpTA)," *IEEE International Conference on Robotic Computing (IRC)*, Nov. 2020.

[22] S. Bakshi, T. Feng, Z. Yan, and D. Chen, "Fast scheduling of autonomous mobile robots under task space constraints with priorities," *ASME. J. Dyn. Sys., Meas., Control.*, 2019.

[23] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[24] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[25] H. I. Yang and Y. J. Zhao, "Trajectory planning for autonomous aerospace vehicles amid known obstacles and conflicts," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 6, pp. 997–1008, 2004.

[26] D. P. Huttenlocher, K. Kedem, and M. Sharir, "The upper envelope of voronoi surfaces and its applications," *Discrete & Computational Geometry*, vol. 9, no. 3, pp. 267–291, 1993.

[27] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[28] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[29] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *2008 IEEE International Conference on Robotics and Automation*, pp. 1928–1935, IEEE, 2008.

[30] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*, pp. 3–19, Springer, 2011.

[31] J. Snape, J. Van Den Berg, S. J. Guy, and D. Manocha, "The hybrid reciprocal velocity obstacle," *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[32] P. Trautman and A. Krause, "Unfreezing the robot: Navigation in dense, interacting crowds," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 797–803, IEEE, 2010.

[33] M. Rufli, J. Alonso-Mora, and R. Siegwart, "Reciprocal collision avoidance with motion continuity constraints," *IEEE Transactions on Robotics*, vol. 29, no. 4, pp. 899–912, 2013.

[34] A. Levy, C. Keitel, S. Engel, and J. McLurkin, "The extended velocity obstacle and applying orca in the real world," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16–22, IEEE, 2015.

[35] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1343–1350, IEEE, 2017.

[36] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6252–6259, IEEE, 2018.

[37] S. M. Khansari-Zadeh and A. Billard, "A dynamical system approach to realtime obstacle avoidance," *Autonomous Robots*, vol. 32, no. 4, pp. 433–454, 2012.

[38] L. Huber, A. Billard, and J.-J. Slotine, "Avoidance of convex and concave obstacles with convergence ensured through contraction," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1462–1469, 2019.

[39] J.-C. Latombe, *Potential Field Methods*, pp. 295–355. Boston, MA: Springer US, 1991.

[40] C. W. Warren, "Multiple robot path coordination using artificial potential fields," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 500–505, 1990.

[41] Y. Koren, J. Borenstein, *et al.*, "Potential field methods and their inherent

limitations for mobile robot navigation.," in *ICRA*, vol. 2, pp. 1398–1404, 1991.

[42] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[43] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

[44] S. M. LaValle, J. J. Kuffner, B. Donald, *et al.*, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and computational robotics: new directions*, vol. 5, pp. 293–308, 2001.

[45] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, pp. 65–100, 2010.

[46] Z. He, Y. He, and B. Zeng, "Obstacle avoidence path planning for robot arm based on mixed algorithm of artificial potential field method and rrt," *Industrial Engineering Journal*, vol. 20, no. 2, p. 56, 2017.

[47] S. Upadhyay and A. Ratnoo, "Smooth path planning for unmanned aerial vehicles with airspace restrictions," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 7, pp. 1596–1612, 2017.

[48] C. Sun, Y.-C. Liu, R. Dai, and D. Grymin, "Two approaches for path planning of unmanned aerial vehicles with avoidance zones," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 8, pp. 2076–2083, 2017.

[49] A. Marchidan and E. Bakolas, "Collision avoidance for an unmanned aerial vehicle in the presence of static and moving obstacles," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 1, pp. 96–110, 2020.

[50] M. Braquet and E. Bakolas, "Greedy decentralized auction-based task allocation for multi-agent systems," *IFAC-PapersOnLine*, vol. 54, no. 20, pp. 675–680, 2021.

[51] R. Battin, "An introduction to the mathematics and methods of astrodynamics," *AIAA Education*, pp. 559–561, 1987.

[52] E. Bakolas, "A decentralized spatial partitioning algorithm based on the minimum control effort metric," *2014 American Control Conference*, pp. 5264–5269, 2014.

[53] M. Braquet and E. Bakolas, "Vector field-based collision avoidance for moving obstacles with time-varying shape," *Modeling, Estimation and Control Conference*, 2022.

[54] R. Nürnberg, "Distance from a point to an ellipse," *URl: http://wwwf. imperial. ac. uk/˜ rn/distance2ellipse. pdf*, 2006.

Index

Vita

Martin Braquet was born in Ottignies-Louvain-la-Neuve, Belgium. He received the Bachelor of Science (2018) and Master of Science (2020) degrees in Electromechanical Engineering, majoring in mechatronics, from the Catholic University of Louvain (Belgium). He pursued a research internship in the Space Systems Laboratory at the Massachusetts Institute of Technology in the summer 2019. He joined the Aerospace Engineering graduate program at The University of Texas at Austin in August, 2020.

Contact email address: martin.braquet@gmail.com