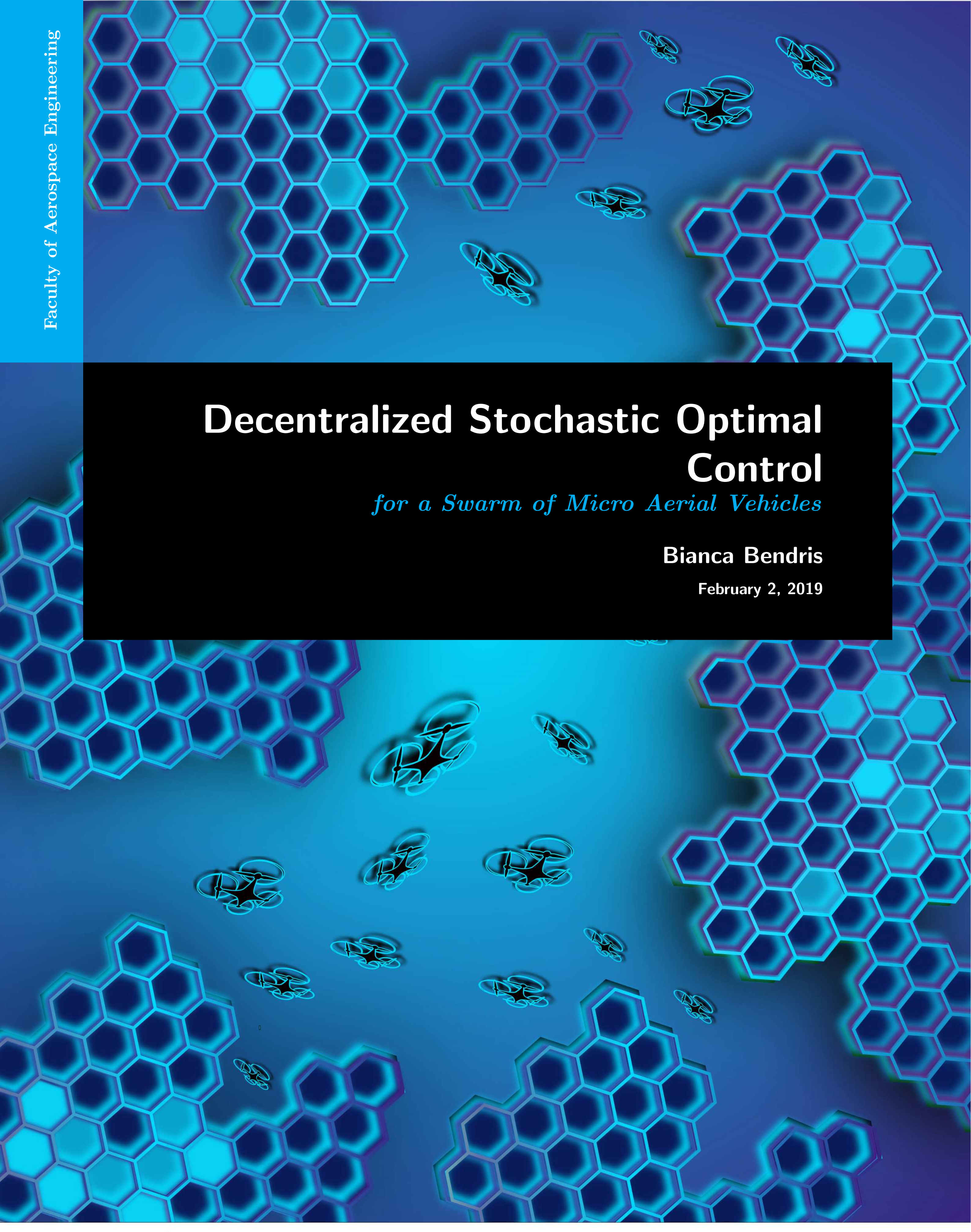
Faculty of Aerospace Engineering



Decentralized Stochastic Optimal Control

for a Swarm of Micro Aerial Vehicles

Bianca Bendris

February 2, 2019

Delft University of Technology

Challenge the future

Decentralized Stochastic Optimal Control

for a Swarm of Micro Aerial Vehicles

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

Bianca Bendris

February 2, 2019

Faculty of Aerospace Engineering · Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Decentralized Stochastic Optimal Control"** by **Bianca Bendris** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: February 2, 2019

Readers:

Dr. G. C. H. E. de Croon

Prof. dr. H.J. Kappen

Ir. C. de Wagter

Ir. K. N. McGuire

Acknowledgements

Undertaking this year long thesis project has been a truly challenging experience which would not have been possible without the support of many people. First of all, I want to thank my supervisors Guido de Croon and Kimberly McGuire for their contributions of time, ideas and support throughout the entire thesis. The thesis I am presenting today, would not have been the same without your guidance. A special mention also goes to Prof. Bert Kappen which helped me to better understand the mathematical derivations behind the path integral theory. Thank you for your kind explanations and patience.

A good support system is important if one wants to succeed with such a complex task as finishing a MSc thesis. I was lucky to be surrounded by friends and house-mates which made the process a lot easier. You are too many to name one by one, but you have become my small family in Delft and for that I am truly grateful.

Finally, and most importantly I wish to thank my family for their endless support and encouragement. The following words in Catalan go for you:

No hi ha paraules per descriure tot el que el vostre suport i confiança ha significat per mi. Nomès em queda dir, gràcies mama, Jordi, Ioana i Martí. Aquest treball no haguès estat possible sense vosaltres.

Abstract

There is a continuous demand for swarms of MAVs to perform tasks which cannot be completed by a single MAV. Tasks which require exploration of large spaces (e.g. environmental monitoring of lakes, oceans, or crops) or exploring unknown and possibly dangerous environments (e.g. forest fire detection, infrastructure inspection) would benefit from using a robust, flexible and scalable swarm of MAVs. Despite the existing demand, multiple open problems persist on how to control the swarm to guarantee the collective performance. Many existing solutions approach the formation control problem using optimal control techniques. However, these often use simplified system models to overcome the curse-of-dimensionality characteristic to complex systems such as MAVs or are applied in a centralized way, neglecting the decentralized behavior observed in natural swarms.

To overcome these limitations, a decentralized path integral (PI) control is proposed as an efficient method to solve a certain class of stochastic optimal control problems. The method builds upon the recent work on PI control theory. Solidly grounded in the stochastic optimal control theory, this method transforms the computation of the Hamilton-Jacobi-Bellman (HJB) equation into an approximation problem of a path integral without running into high-dimensionality problems. The approximation problem is further solved through sampling methods which are often inefficient and computationally expensive. In this thesis, we propose a probe enhanced importance sampling (PEIS) technique to guide the sampling process towards the optimal regions in an efficient way. The resultant PI-PEIS algorithm is used to solve the formation control problem for a holding and leader-follower task with up to four MAVs. Simulation and hardware experiments are presented to show the feasibility of the proposed methodology.

Besides the main research topic, this thesis also includes a review of relevant literature on path planning and optimal control methods with a special focus on path integral control theory. Preliminary simulation experiments of the decentralized PI controller and a step-response analysis of the real platform used for the hardware experiments are also presented.

Contents

Acronyms

ACO	Ant Colony Optimization
ALFURS	Autonomy Levels for Unmaned Rotorcraft Systems
AUV	Autonomous Underwater Vehicles
BVP	Boundary Value Problem
DOF	Degrees of Freedom
ESI	External System Independence
GA	Genetic Algorithms
GNC	Guidance, Navigation and Control
GPS	Global Position System
GPU	Graphics Processing Unit
HJB	Hamilton-Jacobi-Bellman
LF	Leader-Follower
LQ	Linear Quadratic
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
MAV	Micro Aerial Vehicle
MCS	Motion Capure System
MDP	Markov Decision Process
MPC	Model Predictive Control
NLP	Non-Linear Programming
PDE	Partial Differential Equation
PEIS	Probe Enhanced Importance Sampling
PI	Path Integral
POMDPs	Partially Observable Markov Decision Processes
PRM	Probabilistic Road Maps
PSO	Particle Swarm Optimization
RL	Reinforcement Learning

RM	Road Maps
RRT	Rapidly-exploring Random-Tree
RT	Real time
RUAS	Rotorcraft Unmanned Aircraft System
SDE	Stochastic Differential Equation
SOC	Stochastic Optimal Control
UAV	Unnmanned Aerial Vehicle
UWB	Ultra Wide Band

List of symbols

$x(\cdot)$	state variable
$x^*(\cdot)$	optimal state variable
$u(\cdot)$	control signal
$u^*(\cdot)$	optimal control signal
t	time
t_i	initial time
t_f	final time
T	finite time horizon
$f(\cdot)$	system dynamics
$g(\cdot)$	stochastic process dynamics
$C(\cdot)$	cost function
$L(\cdot)$	instantaneous cost
$\phi(\cdot)$	end cost
$J(\cdot)$	optimal cost-to-go
$W(\cdot)$	Wiener process
dw	infinitesimal increment of $W(\cdot)$
υ	variance of process noise
S	discrete states in MDP framework
a	discrete actions in MDP framework
$p(\cdot\|\cdot)$	transition probability in MDP framework
$R(\cdot)$	reward function in MDP framework
$V(\cdot)$	value function in MDP framework
π	control policy in MDP framework
γ	infinite horizon discount factor in MDP framework
$\mathcal{H}(\cdot)$	Hamiltonian
$\lambda(\cdot)$	Lagrangian multipliers
$\lambda^*(\cdot)$	Lagrangian multipliers when optimal control is applied

B	control transition matrix
$q(\cdot)$	state dependent instantaneous cost function
R	control cost weight matrix
λ	proportionality constant between the noise variance and the control cost
$\Psi(\cdot)$	desirability function
$S(\cdot)$	*Action* or cost of a path
W	weight of a path
N	number of sample trajectories
$\hat{\Psi}(\cdot)$	approximated desirability function
$\hat{u}$	approximated control signal
u_{exp}	exploratory controls used to guide the sampling
Z	number of exploratory probes
α	degrees around the velocity vector at which the probes are taken
β	separation degrees between each probe
p_{β°	probe taken at β° from the velocity vector
V_{rot}	matrix of rotated velocities vectors
U_{rot}	matrix of rotated control vectors
$R_{i\beta}$	rotation matrix for the angle $i\beta$
v_{best}	velocity vector of the minimum cost probe
u_{best}	control vector of the minimum cost probe
p_L	EN position vector of the leader
v_L	EN velocity vector of the leader
$q_L(\cdot)$	state dependent cost function of leader unit
$q_F(\cdot)$	state dependent cost function of follower unit
C_t	cost penalty for leader-target distance
C_h	cost penalty for leader-target heading deviation
C_{col}	cost penalty for collision risk between two units
C_{coh}	cost penalty for non-cohesive flight between leader and followers
C_p	cost penalty for parallel flight paths between two units
C_{vel}	cost penalty for flying with a lower velocity than the minimum allowed
C_{sep}	reward given for big inter-agent distances
C_{out}	cost penalty for out-of-bounds situations
d_{out}	distance outside the boundary area
d_{min}	minimum distance allowed between two units
v_{min}	minimum velocity allowed during the holding pattern
r	maximum distance allowed between leader and followers
CP_m	cross product of velocity vectors between two units
CP_{thr}	cross product threshold before considering parallel paths

List of Figures

Chapter 1

Introduction

1-1 Motivation

Swarm robotics is a field which studies the problem of coordinating multiple robots to achieve a common goal. It takes its inspiration from the intelligent behavior observed in social insects (e.g. bees, ants, wasps) which, as explained in Beni (2005):

"*act not just as a group, but show special characteristics such as decentralized control, lack of synchronicity and simple, (quasi) identical members*"

These properties lead to a highly robust, flexible and scalable swarm of insects which is able to collaboratively perform complex tasks. Having a multi-robot system displaying the same properties is very appealing. Thus, swarm robotics emerges with the goal of mimicking the natural swarm behavior and enabling swarm-like properties to multi-robot systems.

Within the range of robotic platforms that could be used to deploy the swarm of robots, many opt for small and relatively simple ground robots (Minchev et al., 2004; J. Chen et al., 2015). However, due to the rapid miniaturization of Micro Aerial Vehicles (MAVs), promoted by the decrease in size and cost of on-board processors and sensors (Floreano & Wood, 2015), MAVs became a suitable platform for multi-robot deployments (Bandyopadhyay et al., 2017; Quintero et al., 2013; Kushleyev et al., 2013; Michael et al., 2011). Unlike ground robots, MAVs are not limited to two dimensional movement and are generally faster and more maneuverable.

Possible applications for swarms of MAVS exploit the mentioned robustness, flexibility and scalability properties of natural swarms (Şahin, 2005) as well as the agility and speed of MAVs. They are particularly interesting for tasks which require exploration of large spaces (e.g. environmental monitoring of lakes, oceans, or crops) or exploring unknown and possibly dangerous environments (e.g. forest fire detection, infrastructure inspection). These types of tasks could benefit from the distributed sensing ability of swarms robotics which can be scaled as desired (scalability) while also taking advantage of the high velocity of MAVs to cover large spaces. Moreover, in case of failure or accident of an individual, the swarm could reconfigure and adapt to the new situation (flexibility) to successfully complete the task.

Despite the rapid advances on MAV platforms and the existing demand for operative MAV swarms, multiple open problems persist on how to control the MAV formation to guarantee a certain collective performance. Towards this goal, many have studied *formation control* strategies, as well as *collective motion planning and control* approaches (Y. Liu & Bucknall, 2018). Existing methods to formation control fall into three main categories: behavior-based, virtual structures and leader-follower. Due to the inherent formation convergence problem of behavior-based approaches (Lawton, Beard, & Young, 2003) and the large inter-robot communication bandwidth required by virtual structure methods (Lewis & Tan, 1997), the leader-follower (LF) approach is mostly used (Saska et al., 2014; Quintero et al., 2013; Kuriki & Namerikawa, 2014). The LF approach assigns the role of the leader to one unit while the others become followers, only required to maintain a desired distance with respect to the leader. Once the leader's motion is defined, the formation control problem can be seen as an extension to the classic tracking problem (Li, Xiao, & Tan, 2004). Consequently, well-known trajectory tracking techniques can be used to control the LF formation, solving many of the issues swarm robotics is dealing with. Therefore, within this work, one of the formation control strategy to be analyzed is the LF flight.

When discussing the collective motion planning and control of an MAV formation several criteria must be considered. Taking for example the exploration of a cluttered and unknown environment, feasible paths which avoid inter-robot collisions as well as obstacle collisions must be planned. Given the low energy resources available on the MAVs, these paths should also be efficient, optimizing the time and distance traveled. One common way of generating these efficient paths is by using optimization methods. This method formulates the collective motion of the MAVs as an optimization problem, following individual constraints (e.g. maximum velocity) and collective constraints (e.g. minimum distance among MAVs) which can be solved with optimal control theory. Applying optimal control theory to a system as complex and non-linear as an MAV while accounting for its inherent stochastic behavior, requires the use of a stochastic non-linear optimal control framework. The challenges of solving the stochastic optimal control problem for a non-linear system with high-dimensional state and control spaces are well known among the optimal control community (Stengel, 1994). Its solution requires solving a second-order, partial differential equation known as the Hamilton Jacobi Bellman equation for all states and controls. This operation becomes intractable for high-dimensional systems, such as MAVs.

Recently, a Path Integral stochastic optimal control approach was introduced by Kappen (2005) which opened the possibility of computing optimal control sequences for a certain type of non-linear, high-dimensional systems under stochastic effects. This technique replaces the demanding operation required to solve the HJB equation, with an inference problem which can be solved by means of sampling from a diffusion process. Since it was firstly introduced, the path integral (PI) control approach has been applied to a variety of multi-robot, high-dimensional, stochastic problems (Doerr et al., 2018; Kreuzer & Solowjow, 2018; Gómez et al., 2015). These implementations showed the feasibility of using a PI control to achieve collective motion of a team of robots. However, all these implementations followed a centralized strategy, in which an off-board centralized algorithm was employed to obtain the individual control commands for each robot (Gómez et al., 2015; Kushleyev et al., 2013; Milutinovi & Lima, 2006; Kuriki & Namerikawa, 2014). With the aim of obtaining a swarm-like behavior for the MAV formation, a decentralized scheme of the PI controller is be implemented in this work, meaning that the optimal control sequences will be computed independently and on-board each MAV. Although the decentralized scheme is appealing for several reasons (e.g. system robustness), it brings with itself an added complexity as the on-board controller must be light enough to ensure real-time performance on the limited computational platforms available on the MAVs. Since the sampling process is one of the most computationally expensive parts of the algorithm, many have worked towards achieving an efficient sampling procedure (Ha & Choi, 2016; Kappen & Ruiz, 2016; Arslan, Theodorou, & Tsiotras, 2014; Menchón & J. Kappen, 2018). Nevertheless, no computational complexity analysis was shown to demonstrate the feasibility of these methods on-board real platforms.

1-2 Objective and research questions

Based on the arguments provided in Section 1-1, the **objective** of this work is:

to demonstrate *formation* flight of a team of MAVs, by means of a *decentralized, stochastic, path integral controller*

As mentioned in Section 1-1, one of the main challenges is how to efficiently implement the path integral controller on-board the MAV which has limited computational resources available. Therefore, the first research question aims to find:

RQ1: How can the path integral controller be used to achieve formation flight of a team of MAVs with limited computational resources?

Since the sampling process involved in solving the estimation problem introduced by the PI control is one of the most computationally expensive parts of the algorithm, the second guiding research question is:

RQ2: How can useful samples be generated in real-time to obtain an accurate estimation of the optimal controls used to guide the MAV formation?

1-3 Document structure

The document is divided in three parts. First, the main contributions of this thesis are collected in the scientific paper presented in Part I. The paper can be read as a stand-alone document. It provides an introduction to the main concepts discussed and presents the relevant experimental results performed within this work. The remainder of the thesis provides background theoretical knowledge for most of the topics examined in this thesis.

Part II provides a literature review on the topics of path planning, optimal control and the path integral framework. An introduction of the main techniques used for path planning is given in Chapter 2. From this, motivation for the use of optimization methods, as a possible solution for the collective motion of a team of MAVs is obtained. Next, a theoretical introduction to the optimal control framework is presented in Chapter 3. Deterministic and stochastic approaches are discussed, and an overview of the methods used to solve optimal control problems is presented. In Chapter 4, the path integral control method is described and methods to solve the path integral control algorithm are given. Part II is closed with a discussion chapter on the main observations derived from the literature study (Chapter 5).

Part III contains the preliminary experiments which have lied the basis for the main experiments presented in the scientific paper in Part I. In Chapter 6 presents simulation experiments performed on a LF formation flight task controlled with a decentralized PI controller. Different scenarios are shown in which the performance of the LF flight and the runtime of the algorithm are analyzed. Next, in Chapter 7, the step response of the Parrot Bebop I platform used in the hardware experiments is analyzed. A synthesis of these preliminary experiments is given in Chapter 8.

Finally, detailed derivations and theoretical background of the presented techniques can be found in the Appendix A,B and C. A description of the code used for the preliminary simulation experiments is given in Appendix D.

Part I

Scientific Paper

Decentralized Stochastic Optimal Control for a Swarm of Micro Aerial Vehicles

MSc Student: Bianca Bendris [1]
Supervisors: Kimberly McGuire [2], Guido de Croon [2] and Hilbert Kappen [3]

Abstract—In this paper, we model a multi-robot formation planning and control task as an optimization problem, which we solve on-line and in a decentralized manner using the Stochastic Optimal Control (SOC) framework. Typically, the solution of a SOC problem requires solving the Hamilton-Jacobi-Bellman (HJB) equation for all system states and controls. However, this operation becomes intractable when high-dimensional systems are used. In recent years, advances on a certain type of SOC problem, which can be efficiently solved by sampling from a diffusion process have been presented and are better known as path integral (PI) control. We build upon this theory and implement a decentralized formulation of the PI algorithm to compute the optimal controls of real Micro Aerial Vehicles (MAVs) flying in formation using solely on-board computational resources. One challenging aspect of the PI control method is the efficient sampling of useful trajectories. It is not clear how to guide the samples towards the optimal states. To this end, we propose a probe enhanced importance sampling (PE-IS) method which performs a coarse exploration of the state space with the objective of identifying an optimal guiding trajectory around which the samples are taken. The feasibility of the proposed method is shown by means of simulation and real-hardware experiments with up to four MAVs in an indoor environment.

Index Terms—decentralized, optimal control, path integral, swarm robotics, MAVs

1 Introduction

Swarms of small-scale robots deployed in large numbers are particularly interesting for tasks which require exploration of large spaces (e.g. environmental monitoring of lakes, oceans, or crops) or the inspection of unknown environments (e.g. forest fire detection, infrastructure inspection). The unique characteristics of swarms in terms of robustness, flexibility and scalability [1] make them more appealing than single-agent systems. The rapid miniaturization of Micro Aerial Vehicles (MAVs), enabled by the decrease in size and cost of on- board processors and sensors [2], favored the large-scale deployment of these robots and their use in multi-agent systems. Generally faster and more maneuverable than ground robots, many studies involving multi-agent systems use MAVs as the robotic platform of choice [3]–[6].

Despite these rapid advances in MAV technology, a large gap exists between the potential applications of swarms and generally multi-agent systems and their actual deployment in real-life. Among others, the motion planning and control of the entire robot formation remains a challenge when faced with the characteristic uncertainties and non-linearities of real systems. Moreover, the common goal, inter-agent interactions and shared information of the formation adds a new layer of complexity in the planning and control strategy unseen in single-agent systems.

One common approach to such problems is to define the formation control as an optimization problem and employ the theoretical framework of Stochastic Optimal Control (SOC) to solve it. This method requires solving a non-linear, stochastic, second-order, partial differential equation (PDE) known as the Hamilton-Jacobi-Bellman (HJB) equation for all system states and controls. Discrete-time systems are typically modeled as Markov Decision Processes (MDPs) and solved through well-known dynamic programming methods [4], [7], [8]. In a similar way, continuous systems can also be discretized and modeled as MDPs. However, for a continuous system such as the MAV, this discretization becomes intractable due to the large state and control spaces in which it operates.

In recent years, progress has been made in solving high-dimensional SOC problems. A logarithmic transformation of the optimal cost-to-go function was proposed in [9], which for certain types of non-linear systems leads to a linear expression of the HJB equation. The linearity can be exploited such that the optimal controls can be approximated through forward sampling of trajectories or paths. This approach is known as path integral (PI) control and has lead to numerous successful application in a variety of fields such as spacecraft attitude control [10], field space exploration with Autonomous Underwater Vehicles (AUVs) [11], the control of a team of quad-rotors [12] or even aggressive driving with ground vehicles [13]. All these implementations follow a centralized strategy, in which an off-board centralized computer is employed to obtain the individual control commands for each robot [5], [12], [14], [15]. In this paper, a decentralized formulation of the PI controller is used.

The decentralized scheme has many advantages such as the increased system robustness and scalability. However,

[1] *MSc Student, Faculty of Aerospace Engineering, Delft University of Technology, The Netherlands*
[2] *Supervisor, Faculty of Aerospace Engineering, Delft University of Technology, The Netherlands*
[3] *Faculty of Science, Radboud University of Nijmegen, The Netherlands*

it brings with itself an added complexity as the control algorithm must be light enough to ensure real-time performance on the limited computational resources available on the MAVs. One way of mitigating the computational costs is by combining the PI control with an efficient Model Predictive Control (MPC) algorithm [11]–[13]. Within the robotics community, MPC has been used in combination with other control algorithms as a way of obtaining real-time motion control of aerial [16], [17], wheeled [18], [19] and humanoid robots [20], [21]. This approach allows the calculation of the optimal controls in closed loop over a receding finite horizon. For the PI control algorithm, this means that the samples are propagated over a short horizon time at each time-step and implies that the sampling process can be efficiently evaluated to compute the optimal controls in real-time. Yet, depending on the complexity of the task, many samples might be needed to accurately compute the optimal controls. Thus, obtaining an efficient sampling strategy is vital for the implementation of the PI control method in real-time.

Generally importance sampling schemes have been used in literature to estimate the optimal controls [11], [12], [22], [23]. This method consists in replacing the diffusion process from which samples are taken with another diffusion process that could lower the variance of the estimate, thus improving the efficiency of the sampling. Yet, how to choose an effective importance sampler remains an open question. A first theoretical analysis on the relation between the sampling efficiency and the accuracy of the obtained optimal controls is described in [24]. In [25], a cross-entropy method is used to iteratively improve the importance samplers. A rapidly-exploring random three (RRT) algorithm is used in [26] to find a baseline trajectory which can steer the samples towards a goal region in a configuration space with obstacles. A similar approach is taken in [23], where different topology class trajectories are identified and used as reference to guide the samples in a highly non-convex state space with multiple obstacles. All these methods have been validated using only numerical examples without considering the computational limitations of real platforms. Moreover, in [23], [26] single-agent examples were shown which had to move towards a clear goal region disregarding the added complexities of multi-agent systems.

The contribution of this paper is twofold. Firstly, a decentralized scheme of the PI control is shown to be computationally light enough to run on-board real MAV platforms and compute on-line the optimal controls. To the best of our knowledge, this has been the first decentralized implementation of PI control on-board real platforms. Secondly, we enhance the PI method with a probe exploration technique with the aim of increasing the efficiency of the sampling method while keeping the computational complexity of the algorithm low. By adding an exploration layer previous to the importance sampling procedure, we seek to guide the diffusion process towards the optimal state space regions in a faster way. Moreover, due to the periodic re-planning enabled by the MPC algorithm, the importance sampling distribution is continuously adapted.

The structure of the paper is as follows. In Section 2 we present the main concepts of the SOC theory and we review the PI control framework as presented in [9]. We then show, how importance sampling techniques can be used to solve the inference problem introduced by the PI control method. Building upon this theoretical knowledge, a probe enhanced importance sampling technique (PEIS) is proposed in Section 3. Section 4 describes the two multi-agent tasks used to evaluate the resultant PI-PEIS algorithm, namely a holding pattern and a leader-follower formation flight. The simulation and hardware experimental settings are described next. Afterwards, we show the main experimental results. These are divided in two sections: the holding pattern experiments are presented in Section 5, while the leader-follower experiments are shown in Section 6. Finally, a discussion of the results is given in Section 8 and conclusions are drawn in Section 9.

2 Preliminaries

2.1 Stochastic Optimal Control

Consider a continuous time stochastic system of the form:

$$dx = f(x,u,t)dt + dw \tag{1}$$

where $f(x,u,t)$ is an arbitrary function describing the controlled system dynamics in terms of the system's states $x \in \mathbb{R}^n$ and controls $u \in \mathbb{R}^m$ at time t. dw represents an increment of a Wiener process $W(t)$ with mean zero and variance $\langle dw^2 \rangle = v dt$, where v is a positive definite covariance matrix.

Let x_i be the initial state at time t_i and $x_{t_i \to t_f}$ the resulting state trajectory for a given realization of the Wiener process when the control $u_{t_i \to t_f}$ is applied over the finite-time horizon $[t_i, t_f]$. A cost can be associated to this trajectory as:

$$C(x_i, u_{t_i \to t_f}, t) = \phi(x_{t_f}) + \int_{t_i}^{t_f} L(x,u,t)dt \tag{2}$$

where $\phi(x_{t_f})$ represents the terminal cost at time t_f and $L(x,u,t)$ is the immediate cost at time t for being at a state x and choosing the control u.

The SOC problem consists of finding for each state x the optimal control u^*, which minimizes the above cost, thus solving:

$$\begin{aligned} J(x,t) &= \min_{u(t\to t_f)} \Big\langle C(x, u(t \to t_f), t) \Big\rangle \\ u^*(x,t) &= arg \min_{u(t\to t_f)} \Big\langle C(x, u(t \to t_f), t) \Big\rangle \end{aligned} \tag{3}$$

where $\langle \cdot \rangle$ denotes the expectation taken over all possible realizations of the Wiener process. $J(x,t)$ is the expected optimal cost-to-go at any state x if an optimal trajectory is followed from that point onwards $(t \to t_f)$. Applying Bellman's optimality principle to this equation results in the non-linear, second order PDE equation known as the stochastic Hamilton-Jacobi-Bellman (HJB) equation [27]. In order to solve the SOC problem and obtain the optimal controls, this equation has to be solved backwards in time from t_f to t_i and for the entire state and control spaces.

2.2 Path Integral Control

This section reviews the PI control theory developed in [9] as a method to solve continuous SOC problems. As mentioned in the previous section, the solution of such problems involves computing the stochastic HJB equation for all system's states and controls. For systems with high-dimensional state and control spaces, this operation is very challenging, as the computational effort required grows exponentially with the increasing dimensionality.

PI control is one of the methods [28]–[31] proposed to cope with what is widely known as the *curse-of-dimensionality*. To do so, the PI control algorithm assumes a certain type of continuous time SOC in which the system's dynamics are linear in the control (Eq. 4-a) and the control cost is quadratic (Eq. 4-b).

$$\begin{array}{ll} (a) & f(x,u,t) = f(x,t) + Bu \\ (b) & L(x,u,t) = q(x,t) + \frac{1}{2}u^T Ru \end{array} \tag{4}$$

Here, $f(x,t)$ represents the passive dynamics which can be arbitrary complex and non-linear while B is the control matrix. The immediate path cost $L(x,u,t)$ is divided between a state cost $q(x,t)$ and a quadratic control cost term, where R is a positive definite weight matrix. The stochastic HJB derived for the above system corresponds to the following equation:

$$\begin{aligned} -\partial_t J(x,t) = \min_{u(t\to t_f)} \Big\langle & q(x,t) + \frac{1}{2}u^T Ru + \\ & (f(x,t) + Bu)\partial_x J(x,t) + \frac{1}{2}\partial_x^2 J(x,t)\upsilon \Big\rangle_x \end{aligned} \tag{5}$$

To find the minimum of this expression, the gradient inside the parenthesis is taken with respect to u and set to zero. Due to the linear-quadratic assumption, an explicit form of the optimal control can be obtained as:

$$u^*(x,t) = -R^{-1}B^T \partial_x J(x,t) \tag{6}$$

Substituting this expression in Eq. (5) leads to the non-linear, second order PDE:

$$\begin{aligned} -\partial_t J(x,t) = & -\frac{1}{2R}\big(\partial_x J(x,t)^T B \partial_x J(x,t)\big) + q(x,t) \\ & + f(x,t)\partial_x J(x,t) + \frac{1}{2}\partial_x^2 J(x,t)\upsilon \end{aligned} \tag{7}$$

Solving this PDE for the value of $J(x,t)$ and computing its gradient to obtain the optimal controls as shown in Eq. (6) is except for some specific cases (i.e linear-quadratic (LQ) problems [32], [33]), a challenging task. The PI control approaches this problem by using a logarithmic transformation of the optimal cost-to-go in terms of a *desirability function* $\Psi(x,t)$. An expression of $J(x,t)$ is obtained as:

$$J(x,t) = -\lambda log \Psi(x,t) \tag{8}$$

Under the assumption $\upsilon = \lambda BR^{-1}B^T$ where λ is a constant, the non-linear terms in Eq. 7 cancel out, resulting in a linear expression of the HJB as:

$$-\partial_t \Psi(x,t) = \Big(-\frac{q(x,t)}{\lambda} + f(x,t)\partial_x + \frac{1}{2}\partial_x^2 \upsilon \Big)\Psi(x,t) \tag{9}$$

with boundary condition: $\Psi(x,t_f) = exp(-\frac{1}{\lambda}\phi(x))$. The assumption $\upsilon = \lambda BR^{-1}B^T$ couples the system dynamics noise with the control, such that both act on the same subspace and in the same direction but inversely related. For high noise directions, the control cost is small while for low noise directions this takes higher values. The intuition behind this assumption is that in higher noise conditions the control actions are less reliable. Thus, it would be unreasonable to assign high penalties to such control commands [22].

Using the Feynman-Kac lemma [34], this linear PDE can be solved by means of forward sampling of a diffusion process. Thus, instead of solving the PDE numerically, an explicit solution of $\Psi(x,t)$ is obtained as the following expectation:

$$\Psi(x,t) = E^{\rho(x_f,t_f|x_i,t_i)}\Big\{exp\Big(-\frac{1}{\lambda}\big(\phi(x_f) + \int_{t_i}^{t_f} q(x,t)dt\big)\Big)\Big\} \tag{10}$$

where $\rho(x_f,t_f|x_i,t_i)$ represents the probability that a sample path going from $x_i \to x_f$ is generated by the uncontrolled stochastic system dynamics:

$$dx = f(x,t)dt + dw \tag{11}$$

conditioned on the start state x_i at time t_i.

Computing the exact value of $\Psi(x,t)$ remains a challenge as it implies taking the expectation in Eq. 10 over uncountable many paths. We can estimate the expected value of $\Psi(x,t)$ using sampling methods to obtain an approximation of the optimal control. The following section introduces one of these techniques.

2.3 Importance sampling

A simple way of achieving an estimate of the desirability function $\hat{\Psi}(x,t)$ is through Monte-Carlo (MC) sampling. However, sampling directly from the probability distribution shown in Eq. (11) can be very inefficient. Many high cost samples are generated which do not contribute to the computation of the optimal controls.

To mitigate this problem, importance sampling schemes have been widely proposed within the path integral community [12], [23], [35], [36]. Importance sampling is a technique commonly used to lower the variance of an estimate and increase the sampling efficiency by changing the diffusion process from which the samples are taken. The underlying question now is which diffusion process can improve the sampling efficiency. As proposed in [35], the controlled system dynamics can be employed:

$$dx = f(x,t)dt + Bu_{exp}dt + dw \tag{12}$$

where u_{exp} represents the exploring controls from $t_i \to t_f$ used to guide the samples towards the minimum cost states. Thus, the samples are now taken around a deterministic reference trajectory guided by the exploring controls. Drawing N statistically independent random samples from this diffusion process, we can approximate $\Psi(x,t)$ as:

$$\hat{\Psi}(x,t) = \sum_{n=1}^{N} W_n \tag{13}$$

with:

$$W_n = \frac{1}{N} exp\Big(- \frac{1}{\lambda} S_{cost}(x_n(t_i \rightarrow t_f))\Big) \tag{14}$$

where $S_{cost}(x_n(t_i \rightarrow t_f))$ represents the cost of a sampled path n as given by Eq. (2) and W_n is the weight assigned to this path. An expression of the estimated optimal controls can now be formulated in terms of the desirability function as:

$$\hat{u} = u_{exp} + \frac{1}{\hat{\Psi}(x,t)dt} \sum_{n=1}^{N} W_n dw_n \tag{15}$$

where dw_n is the noise present for a given sample trajectory n. For a detailed derivation of these equations, the reader is referred to [9], [23], [37].

Eq. 15 shows how the approximated optimal controls are found by adjusting the exploring controls used in the sampling procedure. How much the control deviates from the value of u_{exp} is computed as the average of the noise directions of all samples weighted by their path cost over the finite interval $t_i \rightarrow t_f$ [35]. This expression reflects the importance of cleverly choosing the value of u_{exp}. As shown in [24], u_{exp} can be selected such that the variance of the optimal control estimation decreases, resulting in a more efficient sampling procedure. Thus, the closer u_{exp} is of the optimal control $u*$, the more efficient the sampling process will be.

As explained in Section 1, the PI algorithm can be combined with a MPC framework to obtain a real-time performance. This means that the importance sampling process has to be efficiently performed at every time-step. The amount of samples used must be sufficient to obtain an accurate approximation of the optimal control and at the same time low enough to enable the on-line implementation. Such a strategy was implemented in [12] to control a group of MAVs. To guide the samples, the authors first initialized u_{exp} to zero and took advantage of the constant re-planning of the MPC to substitute u_{exp} with the optimal controls computed in a previous time step, $t-1$. This strategy assumes that the optimal control computed in a previous time-step will remain optimal or close to optimality in the following time-step. This strategy assumes that the optimal control computed in a previous time-step will remain optimal or close to optimality in the following time-step. As all sample trajectories are taken around this previous optimum, the exploration of the state space is limited, which could be detrimental for tasks in which fast changes in direction are required. With this idea in mind, the following section introduces an enhanced importance sampling procedure.

3 Probe Enhanced Importance Sampling

In this section, a probe enhanced importance sampling (PEIS) method is proposed as an alternative technique to achieve a higher exploration of the state space with the objective of finding an optimal reference trajectory to guide the samples and improve the sampling efficiency. With this aim, the sampling process is divided in two phases: one focusing on the *exploration* of the state space and the other on refining the search by *exploiting* a smaller local area of the state space. As mentioned in the previous section, the PEIS method is used together with a MPC framework. Thus, both exploration and exploitation processes propagate the sample trajectories over a finite-time horizon H. The system is assumed to follow double integrator dynamics.

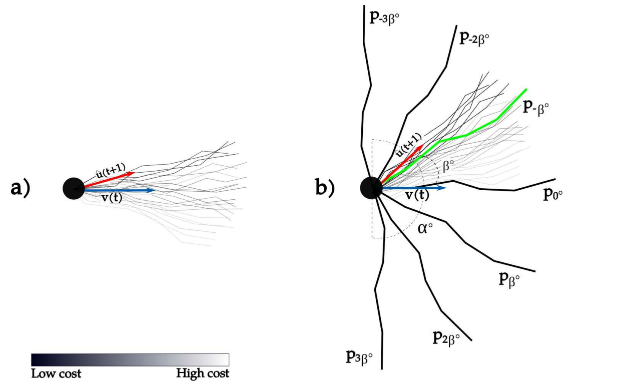


Fig. 1: a) **Importance sampling(IS) method.** Trajectories are sampled from the controlled system dynamics, in which the exploring controls u_{exp} used are the ones computed in the previous time step t. An estimate of the optimal control $\hat{u}(t+1)$ is obtained for time $t+1$. b) **Probe enhanced importance sampling (PEIS) method.** *Exploration part*: probes are generated α° around the current velocity vector $v(t)$ at β° separations (e.g. $p_{\beta^\circ}, p_{2\beta^\circ}$, etc). The minimum cost probe is highlighted in green. *Exploitation part*: trajectories are now sampled around the minimum cost probe (green).

3.1 Exploration

The first phase consists of taking Z samples or probes separated β degrees from each other. These probes P, are taken α degrees around the current motion direction of the agent $v(t)$ as:

$$P \in [P_{m\beta^\circ}, P_{(m-1)\beta^\circ}, ..., P_{\beta^\circ}, ..., P_{-(m-1)\beta^\circ}, P_{-(m)\beta^\circ}] \tag{16}$$

where $m = (\alpha/\beta)$.

As we wish to control the area of the state space which is being explored, these probes are propagated along the time horizon H in a deterministic manner. The simplest way of generating these probes is through the rotation of the current velocity vector towards the desired exploration directions and assuming a constant forward velocity motion throughout the entire planning horizon. However, this approach limits the exploration to areas of the state space which are reachable by the uncontrolled system dynamics, causing the probes to be far away from the optimal states. To avoid this problem, the controlled system dynamics are used instead to generate the probes. Both the exploring controls u_{exp} and the velocity vector $v(t)$ are rotated by means of rotation matrices $R_{i\beta}$ where $i \in [1, m]$ (Algorithm 1).

Algorithm 1: ROTATE(v_{init}, u_{exp})

```
    Data: $m, \beta$
    Input: $v_{init}, u_{exp}$
    Output: $V_{rot}, U_{rot}$
1 for $i$ = $m$,...,1 do
2     $V_{rot}[i] = R_{i\beta} \cdot v_{init}$; ;              // CW rotation
3     $U_{rot}[i] = R_{i\beta} \cdot u_{exp}$;
4     $V_{rot}[m+i] = R_{-i\beta} \cdot v_{init}$; ;     // CCW rotation
5     $U_{rot}[m+i] = R_{-i\beta} \cdot u_{exp}$;
6 end
7 $V_{rot}[2m+1] = v_{init}$; ;                           // No rotation
8 $U_{rot}[2m+1] = u_{exp}$;
```

Algorithm 2: SELECT(V_{rot}, U_{rot})

```
    Data: $H, Z$
    Input: $V_{rot}, U_{rot}$
    Output: $v_{best}, u_{best}$
1 for $z$ = 1,...,$Z$ do
2     $P_z$ = PROPAGATE($V_{rot}[z], U_{rot}[z]$);
3     Compute the cost $S_z$ with Eq. 2
4 end
5 $[P_{best}, P_{best\_index}] = min(S_z)$;
6 $v_{best} = V_{rot}[P_{best\_index}]$
7 $u_{best} = U_{rot}[P_{best\_index}]$
```

Assuming double integrator dynamics, the rotated velocity vectors V_{rot} and exploring controls U_{rot} are used to propagate the probes over the finite time horizon H in the desired directions (Eq. 16). This leads to Z deterministic probes separated β degrees around the initial direction of motion as illustrated in Fig. 1(b). Each probe is then evaluated using the cost function given in Eq. (2) and the minimum cost probe is selected as shown in Algorithm 2. With this procedure a rough optimal direction is found which will be further refined in the exploitation phase.

3.2 Exploitation

The information collected from the exploration of the state space is used in the second phase of the sampling procedure to alter the diffusion process so that the samples are now taken around the minimum cost probe. This is achieved by substituting the exploring controls u_{exp} in Eq. 12 by the rotated controls of the minimum cost probe, u_{best}. Moreover, it is assumed that the initial velocity vector of the sampling is equal to v_{best}. Taking N samples from this new diffusion process allows us to compute the optimal controls as showed in Eq. 6. Now, the reference trajectory used in the importance sampling corresponds to the best probe found in the exploration phase and the optimal controls are found by refining this reference path. To account for the assumption taken on the initial velocity vector, a correction factor is added to Eq. 6 leading to the following expression of the estimated optimal controls:

$$\hat{u} = u_{exp} + (v_{best} - v_{init})/dt + \frac{1}{\hat{\Psi}(x,t)dt}\sum_{n=1}^{N} W_n dw_n \quad (17)$$

A comparison between the classical IS and the PEIS method is schematically depicted in Fig. 1. Using the same number of samples and the same noise variance, the PEIS method enables a wider exploration area to be covered which could steer the samples faster towards the minimum cost states. Moreover, by first performing a coarse search of the optimal direction, the exploring controls are improved leading to an increase in sampling efficiency during the exploitation phase.

4 Decentralized Multi-MAV formation control

This section introduces the multi-MAV formation problem and the two tasks used to evaluate the decentralized PI control formulation with the proposed PEIS method. The experimental settings employed to obtain both simulated and real-hardware results are also presented.

4.1 Formation control design

Take the stochastic control problem of a team of K agents, $k = 1, ..., K$ moving in 2D space, each described by the state vector x_k composed of the East-North (EN) positions and EN velocities of each agent as $x_k = [p_k, v_k]^T$ where $p_k, v_k \in \mathbb{R}^2$. Similarly, the input vector u_k is formed of the EN accelerations where $u_k \in \mathbb{R}^2$. Each agent is assumed to follow double integrator dynamics:

$$\dot{x}_k(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} x_k(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} (u_k(t) + dw_k) \quad (18)$$

To achieve a decentralized control scheme, the formation control problem is approached from the perspective of one single agent, thus, removing the subscript k for the state x and control u vectors. This agent observes $K-1$ neighbors and receives their corresponding EN positions and EN velocities such that a relative state vector $x_{rel} = [p_1, v_1, p_2, v_2, ..., p_{K-1}, v_{K-1}]$ is available at time t. The agent assumes that all its neighbors are moving with a constant velocity in the time interval between two observations. No communication delays or uncertainties due to actuator noise are taken into account for the agent's movement. A global state vector $x_g = [x, x_{rel}]$ can then be defined for each agent.

Similar to [12], a hierarchical control approach is taken. The high-level control is given by the PI-PEIS controller implemented within a MPC framework. The exploration and exploitation phases are executed at every time-step and used to compute the optimal EN acceleration commands. These are then transformed to velocity commands as controls as $v_{comm}(t+dt) = v(t) + \hat{u}(t)dt$. The entire high-level controller is shown in Algorithm 3.

The output of the PI-PEIS algorithm serves as input to the low-level controller which is assumed to follow the velocity command with a first order delay such that:

$$\dot{v} = \tau^{-1}(v_{comm} - v) \quad (19)$$

where τ^{-1} is a diagonal matrix with the diagonal elements corresponding to the time delay. To account for this velocity delay, the agent dynamics are modified as:

$$\dot{x}_k(t) = \begin{bmatrix} 0 & -\tau^{-1} \\ 0 & 0 \end{bmatrix} x_k(t) + \begin{bmatrix} 0 \\ \tau^{-1} \end{bmatrix} (u_k(t) + dw_k) \quad (20)$$

As it will be shown in the following sections, the simple first order model is sufficient to achieve simulation results which are similar to the ones implemented on our real platforms. With these settings, we further evaluate the PI-PEIS method by means of simulation and hardware experiments on two different multi-MAV formation tasks.

4.2 Task I: Holding pattern

The main goal of the holding pattern task is to maintain the agents in proximity to a holding waypoint wp_h, while ensuring a safe inter-agent distance and a minimum velocity. This holding task was implemented in a centralized manner in [12] where it was shown how the resulting flight formation could be obtained as the optimal solution of a SOC problem. Here, we demonstrate how the flight formation can be achieved as the solution of a decentralized SOC problem and we study the effect of the probes on the task performance. This task is schematically depicted in Fig. 2.

The agent's state cost function $q(x_g, t)$ for this task is formulated as the sum of the following cost terms:

$$\begin{array}{ll} \textit{Holding wp} & exp(||p - wp_h||_2 - r^2) \\ \textit{Collision} & \sum_{k=1}^{K-1} P_1(x_g)\Big(exp(C_{col}({d_{min}}^2 - ||p - p_k||_2))\Big) \\ \textit{Min velocity} & P_2(x_g)\Big(C_{vel}(v_{min}^2 - ||v||^2)^2\Big) \\ \textit{Max separation} & \sum_{k=1}^{K-1} -C_{sep} \cdot ||p - p_k||_2 \end{array} \quad (21)$$

where r is the maximum deviation around wp_h at which the agent is allowed to be, $P_1(x_g)$ and $P_2(x_g)$ represent the conditional statements applied to the collision avoidance and the minimum velocity term as:

$$P_1(x_g) = \begin{cases} 1 & if \;\; ||p - p_k||_2 < {d_{min}}^2 \\ 0 & otherwise \end{cases} \quad (22)$$

$$P_2(x_g) = \begin{cases} 1 & if \;\; ||v||_2 < {v_{min}}^2 \\ 0 & otherwise \end{cases} \quad (23)$$

where d_{min} is the minimum inter-agent distance allowed and v_{min} is the minimum velocity permitted. C_{col} and C_{vel} are the corresponding collision penalty and minimum velocity penalties applied when the thresholds are crossed. The conditional statements applied to the collision avoidance and minimum velocity constraints are implemented with the objective of reducing the number of operations needed to compute the cost. The last constraint rewards the situations in which the separation among units is higher by assigning a negative cost proportional to C_{sep}.

To replicate the real experiment conditions, an out-of-bounds penalty has also been added to the state cost as:

$$\textit{Out-of-bounds} \quad exp(C_{out} \cdot d_{out}) \quad (24)$$

where C_{out} is the penalty applied whenever the agent is at a distance d_{out} outside the squared boundary area (red dashed line in Fig. 2). In addition to the state cost, a quadratic control cost is also applied to this task as described in Eq. (4) to penalize high values of commanded controls.

The optimal behavior for this task is that all four agents move in a circular pattern around the holding waypoint while maintaining an equal distant from each other. The radius of the circular pattern is determined by the parameter r and the minimum velocity of each agent is given by value of the parameter v_{min}. For safety reasons, the maximum velocity is limited by the parameter v_{max} outside the cost function.

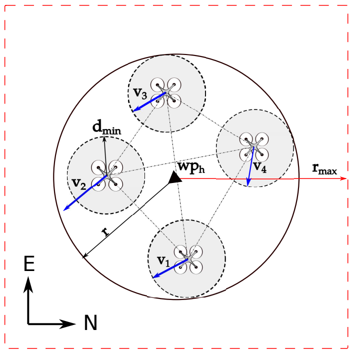


Fig. 2: Schematic representation of the holding task. Four drones are given the task of staying at a radius r around wp_h while also maintaining a safety distance from each other marked by the grey area around each drone. The straight lines connecting the drones represent the communication scheme of the formation. Each drone, receives the position and velocity from all its neighbors. The red dashed line limits the available flying area of the formation to a r_{max} distance around the holding waypoint.

Algorithm 3: PI-PEIS algorithm

```
   Data: $H, N, K, dt, dw$
   Input: $x_g, u_{exp}$
   Output: $v_{comm}, u_{exp}$
1  $v_{rot}, u_{rot}$ = ROTATE($v, u_{exp}$)
2  $v_{best}, u_{best}$ = SELECT($v_{rot}, u_{rot}$);
3  for n = 1,...,N do
4      Sample paths $x_n(t \to t + H)$ with Eq. 20
5      where $v = v_{best}$ and $u = u_{best}$
6  end
7  Store the noise realizations $dw_n$
8  for k = 1,...K-1 do
9      Propagate neighbor trajectory $p_k(t \to t + H)$ with
         constant $v_k$
10 end
11 Compute sample cost $S_n$ with Eq. 2
12 Compute sample weight $W_n$ with Eq. 14
13 $\hat{u} = u_{best} + \frac{1}{\sum_{n=1}^{N} W_n} \sum_{n=1}^{N} W_n dw_n$
14 $v_{comm} = v + \hat{u}dt$
15 $u_{exp} = u_{best}$
16 Save $u_{exp}$ for next time step t+dt
```

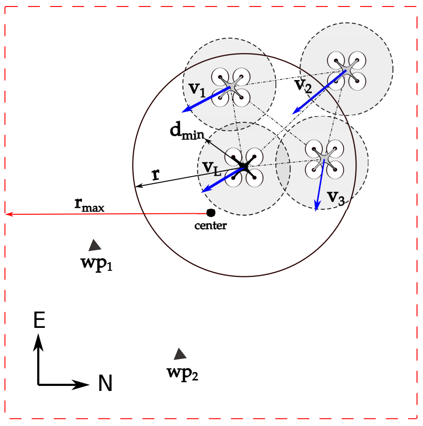


Fig. 3: Schematic representation of the leader-follower task. The leader drone is depicted in black, while the followers are depicted in grey. The goal of the leader is to fly towards the target waypoints (wp_1, wp_2). The followers have to stay at a maximum distance r from the leader. Similar to the holding task, all drones must maintain a safety distance between each other. The flying area is again restricted to r_{max} distance around the center of the arena.

4.3 Task II: Leader-follower flight

The second task is a leader-follower formation flight. In this scenario, one agent is assigned the role of the leader while the remaining agents are the followers. The leader's task is to fly towards a set of predefined target waypoints. The followers, which are not aware of the target waypoints, have to stay within a minimum and maximum distance with respect of the leader to ensure both collision avoidance and formation cohesion. A safe distance has to be maintained among the followers and between each follower and the leader. Given the decentralized scheme of the formation control problem, different state cost function are specified for the leader agent $q_L(x_{g_L}, t)$, denoted with the subscript L and for each follower agents $q_F(x_{g_L}, t)$ denoted with the subscript F. The leader's state cost $q_L(x_{g_L}, t)$ consists of summing the terms:

$$\begin{array}{ll} \textit{Distance target} & C_t||p_L - wp_t||_2 \\ \textit{Heading target} & C_h(v_L \times wp_t) \\ \textit{Collision} & \sum_{k=1}^{K-1} P_1(x_g)\Big(exp(C_{col}({d_{min}}^2 - ||p_L - p_k||_2))\Big) \end{array} \tag{25}$$

where wp_t is the target waypoint given in EN coordinates which the leader needs to follow at a certain time t and C_t is the penalty applied for being far from it. Similarly, a C_h penalty is applied when the leader's velocity vector is not pointing towards the target waypoint. The last term, assigns a C_{col} penalty when the minimum allowed distance between the agents is crossed, thus, when $P_1(x_g) = 1$.

The follower's state cost function $q_F(x_g, t)$ is described by the sum of the following terms:

$$\begin{array}{ll} \textit{Follow leader} & P_3(x_g)C_{coh}(r^2 - ||p - p_L||_2) \\ \textit{Collision} & \sum_{k=1}^{K-1} P_1(x_g)\Big(exp(C_{col}({d_{min}}^2 - ||p - p_k||_2))\Big) \end{array} \tag{26}$$

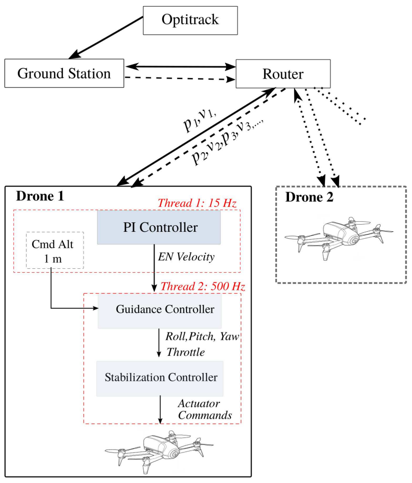


Fig. 4: *Communication scheme:* each drone receives its own position and velocity from the Optirack MCS passing through the Ground Station computer and sent through the router. The relative position and velocity of the neighboring drones is obtained by having the ground station broadcasting the information of all the drones currently tracked by Optitrack and connected to the same network. *Flight control system:* each computes on-board the optimal EN velocities at 15 Hz and sends these to the low-level controller. A guidance controller calculates the velocity error with respect to the sent commands and transforms it to angular commands. These are further given to a stabilization module which outputs the corresponding actuator commands

$$P_3(x_g) = \begin{cases} 1 & if\ \ ||p - p_L||_2 < r^2 \\ 0 & otherwise \end{cases} \tag{27}$$

where C_{coh} is the cohesion penalty applied to a follower when the maximum leader-follower separation r is crossed ($P_3(x_g) = 1$). As in the previous cases, a collision avoidance term is added to ensure a safe distance among the agents.

Similar to the holding task, an out-of-bounds penalty and a control cost is also applied. The leader-follower scenario is illustrated in Fig. 3.

4.4 Experimental Set-up

This section describes the setup used to perform the simulation and hardware experiments for the holding and leader-follower tasks. For the simulation experiments the agents were modeled as 2D point-mass systems and followed the dynamics given in Eq. 20. The time discretization used was $\triangle t = 0.05$ for the main simulation loop and $\triangle h = 0.2$ for the planning loop over the finite horizon H. This time-scale difference was taken into account when the exploring controls u_{exp} were saved from one time-step to the other. The communication among agents was assumed to be flawless and without delay.

For the hardware experiments, the Parrot Bebop I drone was used. This platform has a P7 Cortex 9 Dual Core CPU processor and runs the open-source autopilot framework Paparazzi UAV[1]. The PI controller is implemented as a

1. http://wiki.paparazziuav.org/wiki/Main_Page

module of the Paparazzi autopilot which runs on a separate, slower thread. This module outputs the optimal EN velocity commands (high-level control) which serve as inputs to the guidance controller. This layer then outputs the desired angles and thrust setpoints which are further transformed into actuator commands by the stabilization module (low-level control). Fig. 4 shows the control hierarchy as implemented in Paparazzi. As mentioned in Section 4.1, the optimal controls are computed only for the horizontal motion, leaving the altitude constant. During the hardware experiments, a constant altitude of 1 m is set. The position and velocity of the drone are obtained from an Optitrack Motion Capture System (MCS) available in the indoor arena. This system is also used to obtain the relative position and velocity of all neighboring units at a frequency of 20 Hz.

5 Experiments: Task I Holding pattern

In this section we analyze the performance of the decentralized PI-PEIS algorithm on the holding task described in Section 4.2. First, simulation experiments are presented. These provide an overview on how the different probe configurations of the PEIS method affect the performance of the formation and how this method compares to the standard IS technique. Next, we show real hardware experiments with the aim of investigating the feasibility of the proposed method as a real-time, on-board solution to the holding problem (Fig. 5).

5.1 Simulation experiments

We have simulated the holding task with four agents for different probe configurations. In these simulations, a planning horizon of $H = 2$ s and $N = 100$ samples was used. These values were chosen based on preliminary simulation experiments, which showed that a holding task performed with only four agents did not require a high amount of samples to converge to the optimal behavior. The minimum velocity allowed was set to $v_{min} = 0.5$ m/s while the maximum velocity was limited to $v_{max} = 1$ m/s. The agents were constrained to maintain a maximum distance

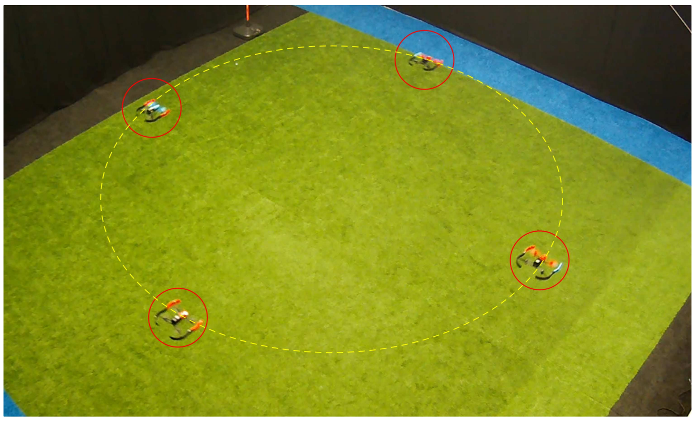


Fig. 5: Four Bebop I drones (indicated by the red circles) performing a holding task around the center of the arena. The dashed yellow ellipse approximately indicates a circle of $r = 2.5m$.

TABLE 1: Mean state cost corresponding to the holding task simulation results (Fig. 6a)

	Mean state cost	
β	$\alpha = 180°$	$\alpha = 360°$
45°	0.03	0.12
30°	-1.21	-0.39
15°	-1.29	-0.51
no probes	1.57	

of $r = 2.5$ m from the holding point while ensuring a safety separation of $d_{min} = 1$ m.

Fig. 6(a) shows the mean path cost evolution of all four agents taken over 25 simulations in which different noise realizations were used. Although all probe configurations, including the scenario without probes, reach a stable formation, the use of probes decreases the cost attained at convergence. For these lower cost situations, a higher inter-agent separation is observed (notice the negative cost values indicating the reward given for large inter-agent separation in Eq. 21). Thus, the estimated optimal controls are less accurate when no probes are employed leading to a less optimal final configuration.

The use of probes also decreases the convergence time, specially for values of $\beta = (30°, 15°)$. However, we observe higher costs during the transition phase (from t = 60 to t = 200) for the configurations in which probes are used when compared with the one without probes. The reason behind this observation is that the agents were moving with a lower velocity than the specified v_{min}. This is due to the fact that the PEIS method undergoes several changes in the probe selection and sometimes commands rapid velocity changes, which cannot be perfectly followed by the simulated velocity controller. As expected, this effect is more pronounced for higher values of β.

When comparing the scenarios with $\alpha = 180°$ and $\alpha = 360°$ we see that both converge to similar cost values: for $\beta = 45°$ the cost converges to a value of -1.9 and for $\beta = (30°, 15°)$ to a value of -2.3. The only difference between these two conditions is a higher cost during the transition phase for $\alpha = 360°$ which is again caused by the changing velocity commands given to the low-level controller. This causes the mean cost to be slightly higher for $\alpha = 360°$ (Table 1).

To determine the sampling efficiency, we use the Effective Sample Size (ESS) parameter as in [12]. This is computed as $ESS = 1/\sum_{n=1}^{N} W_n^2$, where N is the number of samples used and W_n is the corresponding sample weight. For this scenario where 100 samples have been used, a value of $ESS = 100$ indicates that all samples have contributed equally to compute the optimal controls. A value of $ESS = 1$ represents a scenario in which the optimal controls have been computed in base of only one sample, reflecting an inefficient sampling. In Fig. 6(b), we can see a clear advantage in the use of probes. This behavior was expected as the exploration phase is meant to change the diffusion process from which the samples are taken in such a way that more *good samples*, samples with a low cost are found. For values of $\beta = 15°$, we see a significant improvement as the selection of slightly rotated probes favors the

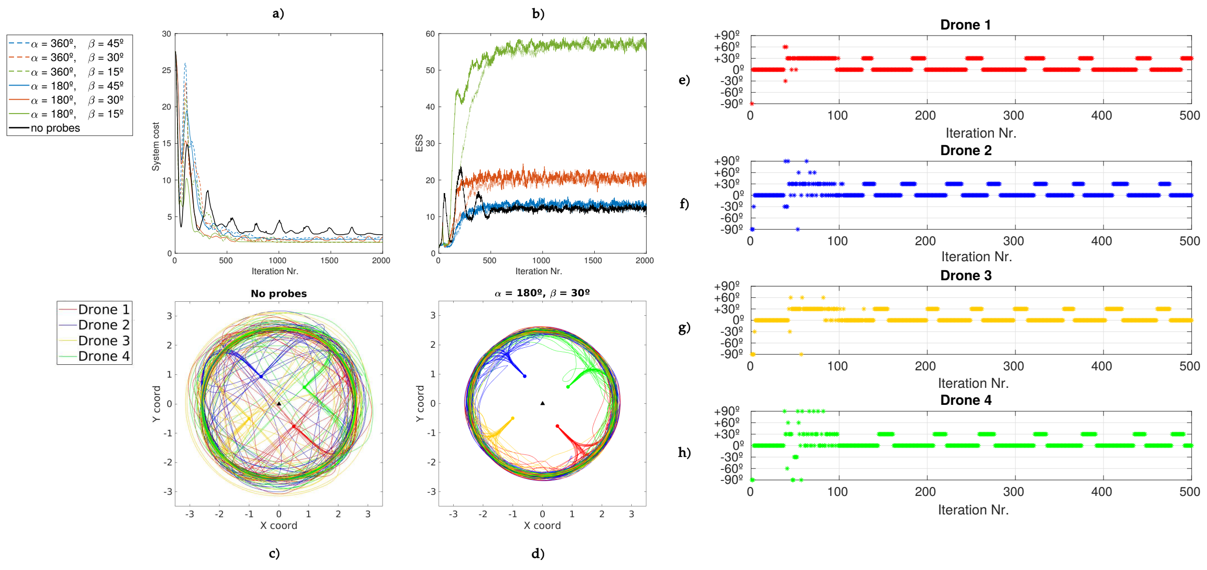


Fig. 6: **Simulation results holding pattern:** a) System cost evolution for different probe configurations. b) ESS evolution for different probe configurations c) 25 trajectories with different noise realizations of four agents performing a holding task without probes. The initial position is marked with a dot while the holding waypoint is marked by the black triangle. d) Trajectories corresponding to the PEIS implementation. e-f) Probe selection over time for all four agents corresponding to one of the 25 noise realization.

circular pattern. However, this behavior is a consequence of the specific optimal formation geometry and cannot be generalized to other tasks.

We compare the 2D trajectories of all 4 agents performing the holding task for a scenario with and without probes (Fig. 6(c-d)). As the scenario using $\alpha = 180°$ has performed slightly better, only these results will be shown. Moreover, a value of $\beta = 30°$ is selected. The previously made observations are reflected in these graphs. We clearly see that the use of probes decreases the convergence time yielding a smoother transition towards the circular pattern.
The wider state space exploration enabled by the probes, allows the agents to start the circling motion earlier in time choosing either a clock-wise or a counter-clock wise direction. This is also reflected in Fig. 6(e), where the probe selection is shown for all 4 drones. We can observe that before converging to the circular pattern the agents shift from the $0°$ probe to the $+30°$ probe achieving a smoother transition to the optimal behavior, which in this particular case is a clock-wise circling pattern. Afterwards, the probes continue to be used to generate the rotative motion. In contrast, the decision to start circling is delayed when no probes are used. This leads to a longer transient period during which the agents organize themselves to reach the optimal circling behavior.

5.2 Hardware experiments

We extend the simulated holding task to a hardware domain. In this section we present real-hardware experiments performed with 4 MAVs which illustrate two main aspects of the system performance. First, we show that the decentralized PI controller can be implemented on-board the real MAV platform and control the multi-MAV formation in real-time. Secondly, we emphasize the benefits of the PEIS method over the standard IS.

For these experiments, we use the same settings as implemented for the simulation experiments. Fig. 7(a-b) shows the trajectories of all 4 drones for the scenario without probes and the one in which the proposed PEIS method is implemented with $\alpha = 180°$ and $\beta = 30°$. In line with the simulation results presented in section 5.1, the use of probe leads to a smoother transition to the optimal circular formation. Clearly, the transient period is higher when no probes are used and the drones have to organize and avoid possible collisions before reaching the optimal configuration. This is also reflected in Fig. 7(c-f) where the state cost evolution of each drone is showed. With the exception of Drone 4, the state cost takes overall higher values for the scenario without probes (See also Table 2). Similar to the simulation results, the use of probes causes the drones to have a slightly slower velocity during the transition phase resulting in a high peak of the cost function as seen in Fig. 7(c-f) during the 70-75 seconds of simulation. This behavior is specially seen for Drone 4, which leads to a lower performance with respect to the other drones.

TABLE 2: Mean state cost corresponding to the holding task hardware results (7(c-f))

	Mean state cost	
Drone nr.	PEIS	IS
1	-0.09	8.57
2	-1.30	6.77
3	-0.31	4.38
4	4.67	3.15

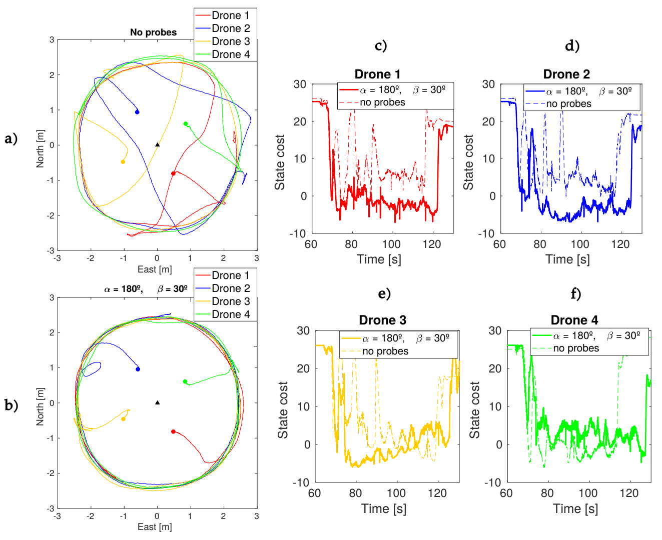


Fig. 7: **Hardware results holding pattern:** a-b)Trajectories of 4 Bebop I drones performing the holding task with the standard IS method and the PEIS method. The initial positions of the drones are marked with a dot while the holding waypoint is indicated in black. c-f) State cost evolution of each drone for both scenarios. The higher initial and end cost value represent the take-off and landing of each drone.

6 Experiments: Leader-follower task

In this section, we show the simulation and hardware experiments corresponding to the leader-follower task described in Section 4.3. Unlike the holding task, the hardware experiments have been performed with four, three and two drones with the objective of highlighting the benefits of the PEIS method.

6.1 Simulation experiments

The leader-follower task has been simulated with 4 agents, for different probe configurations as well as with no probes. Similar to the holding task, a horizon time of H = 2 s and N = 100 samples was used. For this task, the leader is commanded to fly towards target waypoints located at 1.5 m from the center continuously until the end of the simulation. Followers 1 and 2 have to maintain a r = 2 m from the leader while Follower 3 must be at a maximum of r = 2.5 m. These settings are chosen to avoid a very compact leader-follower formation. Moreover, all agents must ensure a safety distance of d_{min} = 1 m among each other and a maximum velocity of $v_{max} = 0.5$ m/s.

The mean state cost of 25 simulations with different noise realizations is shown in Fig. 8(a) for the leader drone and in Fig. 8(b) for the follower drones. It can be noticed that the leader's state cost evolution does not converge to any value. Instead, it oscillates from higher values when the leader is far from the next target waypoint to lower values when the leader arrives at the target waypoint. Except for $\beta = 15°$, no major differences are observed between the scenarios using $\alpha = 360°$ and $\alpha = 180°$. For some noise realizations with $\alpha = 180°$ and $\beta = 15°$, the leader reaches the target waypoints with a certain delay, caused by the position of the followers which fly in front of the leader agent obstructing its movement. This delay lowers the amplitude of the oscillation seen in Fig. 8(a) for $\alpha = 180°$ and $\beta = 15°$. Unlike the holding scenario, the benefit of using the probes is less noticeable for the leader agent. We can only see a difference in the leader's trajectory, which is achieved with sharper angles compared with the situation in which no probes are used. This is also seen in Fig. 8(e) where the periodic turns are reflected by the peaks appearing in the probe selection.

TABLE 3: Mean state cost corresponding to the leader-follower simulation results (Fig. 8(a-b)).

	Leader drone		Follower drones	
β	$\alpha = 180°$	$\alpha = 360°$	$\alpha = 180°$	$\alpha = 360°$
45°	141.6	142.7	7.5	6.4
30°	146.6	138.8	10.0	8.4
15°	145.4	145.1	27.1	18.9
No probes	147.5		15.8	

The probes do manage to lower the cost of the followers drones for values of $\beta = 45°$ and $\beta = 30°$, while for $\beta = 15°$, the mean cost is even worse than the scenario using no probes (Table 3). For this probe configuration, the slower turns lead to less optimal formation flight. Despite the fact that not all simulations converge to the same solution, specially for the follower drones which have a less constrained cost function, we can observe some differences between the two methods. For the scenario in which the PEIS method was employed, we can see how Follower 3 tries to remain in close proximity with the leader by flying inside the squared pattern performed by the leader itself (Fig. 8(f)). To do this, bigger turns need to be performed at a faster rate which is enabled by the selection of further away probes as seen in Fig. 8(k). Followers 1 and 2 take outer trajectories in order to maintain the required minimum distance from the leader (Fig. 8(d-e)). These paths require less turns and are more similar to the scenario without probes, which is also shown by the probe selection presented in Fig. 8(i) and Fig. 8(j) where mostly the 0° probe is selected.

6.2 Hardware experiments

In the previous section we described the probe configuration effect on the performance of a leader-follower task by means of simulation experiments. In this section, we present experimental results performed on real platforms using the settings explained in Section 4.4. We look at three different leader-follower scenarios. First we recreate the leader-follower task with four real drones using the same parameters as in the simulation experiments. As will be explained later in this section, the relatively unconstrained cost function of the followers combined with the limited flying area make the assessment of proposed PEIS method difficult. Thus, a second and third scenario is implemented using three and two drones respectively.

We first analyze the leader follower task with four drones. Fig. 9(a) and Fig. 9(b) show the trajectories flown by these drones with and without using probes. The paths followed by the real drones are similar to the ones obtained through simulation. For the scenario in which probes are used, we can see how Follower 3 approaches the problem with a

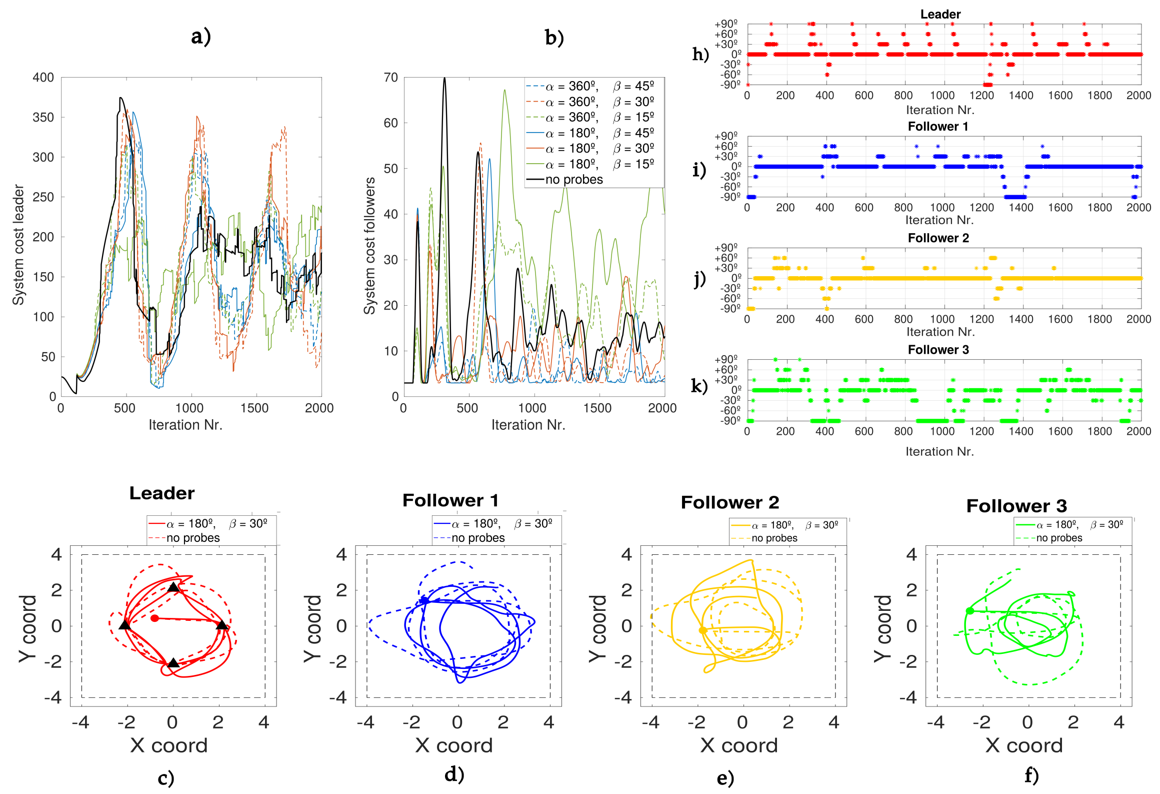


Fig. 8: **Simulation results of leader-follower task:** a) Mean state cost evolution for the follower agents corresponding to 25 noise realizations. The mean is obtained from 25 different noise realizations. b) State cost evolution for the follower agents corresponding to 25 noise realizations. c-d) Trajectories of all 4 agents performing the leader-follower task. For clarity, only 10 noise realizations are plotted. e-h) Probe selection over time of all agents corresponding to one noise realization.

TABLE 4: Mean state cost corresponding to leader-follower hardware experiments with 4 drones (Fig. 9)

	4 Drones		3 Drones		2 Drones	
Drone	PEIS	IS	PEIS	IS	PEIS	IS
L	81.2	80.6	41.5	53.4	38.83	37.62
F1	238.5	159.8	63.7	77.8	1.529	8.94
F2	130.3	918.5	65.0	22.4		
F3	194.7	160.5				

similar strategy, by flying mostly inside the squared path performed by the leader. Moreover, we see how the flexible formation shape allowed by the cost function results in the followers changing their position around the leader (Fig. 10).

However, when no probes are used the task is not completed as Follower 2 flies outside the allowed area and is forced to land (Fig. 9(a)). This does not occur with the proposed PEIS method as the use of probes enables the drones to perform faster turns. Despite this improvement, the use of probes does not manage to decrease the state cost of the drones as showed in Fig. 9(c-f) (See also Table 4). This is caused by the saturation of the low-level controller which affects the yaw control leading to an undesired heading change. The stabilization module quickly corrects the heading deviation but affects the reaction capability of the drone by slowing it down. The result is a worsening in performance of the leader-follower task as the followers get further away from the leader when a yaw deviation occurs. This situation has been also noticed for the scenario in which no probes are used. Nonetheless, the frequency of yaw deviations is higher when the probes are employed due to the even faster changes in velocity commanded. The small flying area available for the four drones combined with the freedom in formation shape have contributed to the saturation of the low-level controller. Therefore, experiments using three and two drones have been performed with the aim of removing this undesired effect.

To highlight the difference between the scenario using probes from the one not using them, we changed the order of the target waypoints to which the leader drone has to fly in the experiment performed with only three drones. Thus, a trajectory requiring faster change in direction has to be followed by the leader drone and consequently, also by the follower drones. Fig. 9(g-h) show the obtained paths for this experiment. At a first sight, we can observe that when the probes are used the formation completes the task using less of the available space (Table 5). Regarding the cost optimization, a slight improvement is seen for the leader

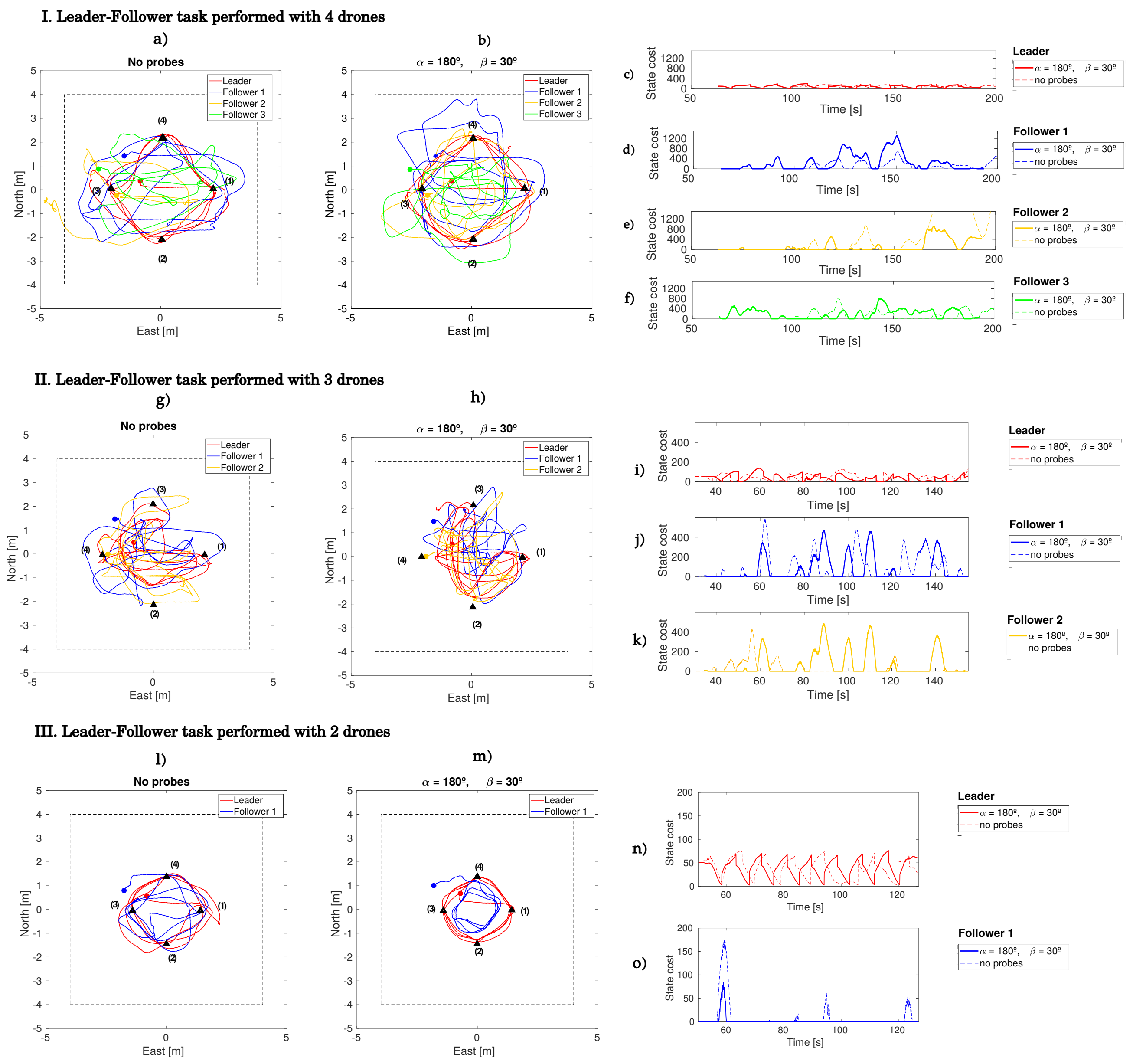


Fig. 9: **Hardware results of leader-follower task:** a-b)Trajectories of 4 drones performing a LF task with the PEIS method and the standard IS method. The initial positions of the drones are marked by the dots. The target waypoints given to the leader drone are marked with black triangles. c-f) State cost evolution of all drones for both scenarios.

and follower 1 drone. However, follower 2 undergoes yaw deviations more frequently which is reflected in the cost function optimization (Fig. 9(i-k)). Both methods manage to complete the task and maintain the separation constraints during most of the experiment duration. Unlike the leader-follower task with four drones, here only the PEIS method has experienced saturations of the low-level controller.

Finally, a simpler scenario using only 2 drones has been performed. In this case, the low-level controller experienced no saturation for neither of the two methods. We can see in Fig. 9(l-m) the different behaviors taken by the follower drone. When probes are used, the strategy of the follower consists of flying in a concentric square shape similar to the one the leader is performing. This allows the follower to maintain a constant separation from the leader minimizing perfectly the cost function (Fig. 9(o)). On the other hand, when no probes are used the follower flies an almost delayed trajectory of the leader but still achieves to maintain

TABLE 5: Area covered by all drones when using the standard importance sampling (IS) and the proposed PEIS method.

	IS	**PEIS**	**% difference**
4 Drones	22.32 m^2	22.92 m^2	$\uparrow$ 2.6%
3 Drones	17.04 m^2	13.08 m^2	$\downarrow$ 23.2%
2 Drones	9.64 m^2	6.32 m^2	$\downarrow$ 34.4%

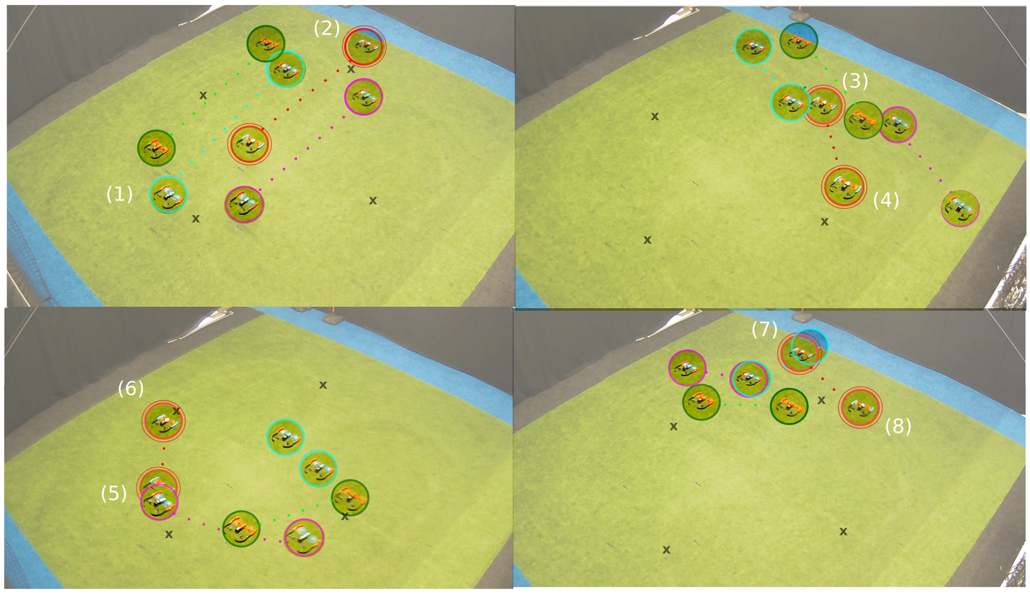


Fig. 10: Snapshots from the LF flight performed with 4 Parrot Bebop I drones. The leader drone is the one highlighted in red. Its task consists in flying towards 4 different target waypoints indicated with a cross while maintaining a safe distance from the followers. The remaining drones try to follow the leader and perform a rotating pattern around the leader.

the desired separation during most of the flight, which is seen in the relatively low cost values attained.

7 CPU TIME ANALYSIS

As mentioned in Section 1, there is an increasing interest in the deployment of robot swarms for a variety of tasks, particularly for small-scale robots which would be less costly and more agile. However, the decrease in size means also a decrease in computational power. This section investigates the effect of a using a smaller processor to run the PI control method with the proposed PEIS technique. To this end, the algorithm has been implemented on a 40-gram pocket drone equipped with a STM32F4 microprocessor (Fig. 11). As the sampling process is the most computationally expensive part of the algorithm, different number of samples have been used. Fig. 12 shows the CPU-time required to compute the optimal controls for one time-step on the pocket drone and the Bebop I drone. The results correspond to a simulated follower task with three neighbors. During these tests the motors were not on but all the remaining modules and sensors of the MAVs were running on-board.

Fig. 11: Pocket drone used for the computational tests

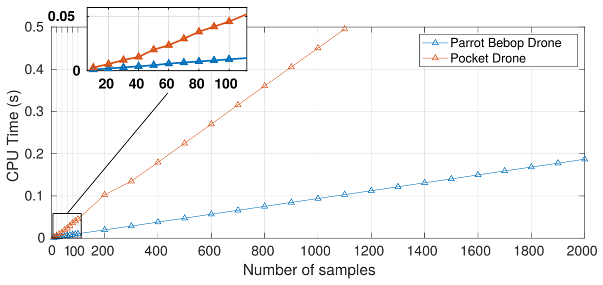


Fig. 12: CPU time required by a Parrot Bebob I drone and a Pocket drone to compute the optimal controls for one time-step. A follower task with three neighbors is simulated to obtain these results.

As expected, the microprocessor on-board the pocket drone runs slower than the Bebop I processor being unable to use more than 1100 samples, point at which the drone stops responding and shuts down. In the previous section, using only 100 samples was shown to be sufficient for the holding and leader-follower tasks. For this amount of samples, both platforms require a low computational time enabling the PI controller to run at a maximum frequencies of 50 Hz and 22 Hz for the Bebop I and pocket drone respectively. These results indicate that the decentralized PI controller is light enough to run on-board a pocket drone and provide high-level control commands in real-time.

8 DISCUSSION

In this work, we studied the use of SOC methods for the motion planning and control of a multi-agent formation. We built upon the existing PI control framework [35] to obtain the optimal controls of each agent in a decentralized manner. Motivated by the existing computational limitations on-board real-platforms, we proposed a probe enhanced importance sampling scheme to guide the samples towards optimal state space regions faster and more efficiently than current methods.

The proposed PEIS method splits the sampling process into an exploration and an exploitation phase. In Sections 5 and 6 we showed how the choice of the exploration strategy depends on the optimal formation geometry of each task. For the holding task, the use of slightly rotated probes ($\beta = 15^\circ$) was preferred due to the emergent circular pattern. In contrast, the diamond-shaped trajectory followed by the leader and the sharper turns performed by the followers favored the selection of probes at 45° and 30°. Although the β parameter does not influence the maximum rotation angle which can be selected, it was noticed that the turns were performed more gradually when lower values of β were selected. This observation made the probe configurations with $\beta = 15^\circ$ less efficient for tasks in which quick and sharp turns were needed. Nevertheless, we saw that choosing a value of $\beta = 30^\circ$ within the simulation experiments lead to a more optimal formation for both tasks, when compared to the scenario in which no probes were used.

When comparing the performance of the proposed PEIS method with the standard importance sampling without probes, we saw a clear improvement for the holding task. In this case, the use of probes brought clear benefits in terms of convergence time and optimality of the final configuration, for both simulation and real-hardware experiments. With only two possible optimal solutions (circling in a clockwise direction (CW) or in a counter-clockwise (CCW) direction), the most challenging part of this task was the transition period in which the agents had to organize to reach the optimal configuration. The choice of circling in CW or CCW direction is consequence of the experimented *symmetry breaking* of the optimal control [9], [35]. When this choice is made depends on the different characteristics of the problem, such as the time to reach the optimal configuration or the noise of the system. The use of probes helped in making the decision earlier in time avoiding possible inter-agent conflicts.

For the leader-follower task, the comparison between the two methods gave different results for the leader than for the followers. On one hand, the leader had to perform a specific and very constrained trajectory which left little margin to improvement. Thus, both PEIS and IS methods had a similar performance. The follower agents on the other hand, were less constrained allowing the PEIS method to optimize their trajectories and achieve better results for the simulation experiments. The use of probes also helped the followers to perform sharper turns and achieve a more compact formation flight. We analyzed the probe selection of the different agents and saw how the probes are used intermittently, specially in cases in which the most optimal state space region is not located in the proximity of the current motion direction. What we can deduce from these results is that the probes are complementary to the classic importance sampling. Unlike other exploration methods which expand a graph into the state space to locate the optimal direction [23], the PEIS method is computationally light, adding almost no additional cost to the existing importance sampling method.

When comparing the simulation and hardware experimental results, not much difference is observed for the holding task. However, the quick velocity changes commanded for the leader-follower task lead to the saturation of the low-level controller when implemented on the real platforms. For these cases, improving the low-level controller response to the velocity command is of course a possibility. However, as with every real-platform, there will always be a certain delay. Future work on this matter could consider the re-design of the follower's cost function. Despite its simplicity, this cost function is non-smooth, as some constraints are only applied when a given threshold is crossed. The reason behind this cost-function design was to reduce the number of operations required. Yet, this approach could cause in some situations rapid velocity changes to be commanded. Future experiments with smoother cost functions could be performed to study the effect on the low-level controller.

The computational feasibility of the decentralized PI algorithm was demonstrated with the hardware experiments showed in Section 5 and Section 6. The optimal controls for each drone were computed with a frequency of 15 Hz using only the on-board computational resources. Moreover, the results shown in Section 7 indicate that the PI algorithm could also be used to control smaller platforms as far as the number of samples used is kept low.

Although in this work, real-experiments with up to 4 MAVs have been showed, the decentralized PI-PEIS algorithm is easily scalable to incorporate more platforms. The only limitations are imposed by the communication network. As the Optitrack motion capture system is used to send the positions and velocities of all neighbor agents, the maximum number of platforms which can be accurately tracked by this system determines the size of the experiments. To avoid this restriction, on-board sensors could be used to determine the relative positions and velocities of nearby agents. Preliminary simulation experiments have been performed in which an Ultra Wide Band (UWB) module was used to obtain the relative state information. Despite the promising results obtained, more work is needed to study the effect of erroneous or delayed packages sent by this module.

9 Conclusions

This paper extends the recent work on PI control as a method to efficiently solve stochastic optimal control problems, to a decentralized formulation and enhances the sampling procedure with a probe exploration phase. The resultant PI-PEIS algorithm is applied to the real-time coordination of multiple MAVs in an indoor environment. For a simple holding pattern scenario, the proposed method shows better performance, achieving a more optimal formation than using the importance sampling scheme without exploration. This was observed for both simulation and hardware experiments in which four MAVs were used. In more complex scenarios such as leader-follower flight, wider state space exploration enabled by the proposed PEIS method lead to a more compact formation flight when three and two agents were used. When four agent leader-follower flight was performed, the improvements were mostly noticeable in the simulation experiments. For the hardware experiments with four MAVs, real-life limitations made the assessment of the proposed methodology difficult.

The performed experiments put in evidence that the method proposed in this paper can be successfully implemented on-board real MAVs achieving real-time performance. Although, future work should address the observed challenges of implementing the PI-PEIS controller for more complex tasks, we consider that the probe enhanced exploration improves the performance of the PI algorithm and is computationally light enough to be executed in real-time.

References

[1] E. Şahin, "Swarm robotics: From sources of inspiration to domains of application," in *Swarm Robotics*, E. Şahin and W. M. Spears, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 10–20.

[2] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, pp. 460–466, 2015.

[3] S. Bandyopadhyay, S.-J. Chung, and F. Hadaegh, "Probabilistic and distributed control of a large-scale swarm of autonomous agents," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1103–1123, 2017.

[4] S. A. P. Quintero, G. E. Collins, and J. P. Hespanha, "Flocking with fixed-wing uavs for distributed sensing: A stochastic optimal control approach," in *2013 American Control Conference*, June 2013, pp. 2025–2031.

[5] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, Nov 2013.

[6] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," *Autonomous Robots*, vol. 30, no. 1, pp. 73–86, 2011.

[7] J. Capitan, L. Merino, and A. Ollero, "Decentralized cooperation of multiple UAS for multi-target surveillance under uncertainties," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2014, pp. 1196–1202.

[8] A. A. Munishkin, D. Milutinovi, and D. W. Casbeer, "Stochastic optimal control navigation with the avoidance of unsafe configurations," in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2016, pp. 211–218.

[9] H. J. Kappen, "Path integrals and symmetry breaking for optimal control theory," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 11, p. P11011, 2005.

[10] B. G. Doerr, R. Linares, and C. D. Petersen, "Spacecraft attitude control using path integral method via riemann manifold hamiltonian monte carlo," in *2018 Space Flight Mechanics Meeting*, 2018, p. 0204.

[11] E. Kreuzer and E. Solowjow, "Learning environmental fields with micro underwater vehicles: a path integral—gaussian markov random field approach," *Autonomous Robots*, vol. 42, no. 4, pp. 761–780, Apr 2018.

[12] V. Gómez, S. Thijssen, A. Symington, S. Hailes, and H. J. Kappen, "Real-time stochastic optimal control for multi-agent quadrotor swarms," *Robotics and Autonomous Systems. arXiv*, vol. 1502, 2015.

[13] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1433–1440.

[14] D. Milutinovi and P. Lima, "Modeling and optimal centralized control of a large-size robotic population," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1280–1285, Dec 2006.

[15] Y. Kuriki and T. Namerikawa, "Consensus-based cooperative formation control with collision avoidance for a multi-UAV system," in *2014 American Control Conference*, June 2014, pp. 2077–2082.

[16] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 236–243.

[17] K. Alexis, G. Darivianakis, M. Burri, and R. Siegwart, "Aerial robotic contact-based inspection: planning and control," *Autonomous Robots*, vol. 40, no. 4, pp. 631–655, 2016.

[18] Y. Cong, H. Chen, and B. Gao, "Real-time path tracking method using differential flatness for car-like mobile robot," *International Journal of Robotics and Automation*, vol. 33, no. 6, pp. 584–593, 2018.

[19] S. Raziei and Z. Jiang, "Nonlinear model predictive motion control of differential wheeled robots," vol. 2018-July, 2018, pp. 443–450.

[20] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1168–1175.

[21] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," vol. 2015-February, no. February, 2015, pp. 292–299.

[22] E. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11, no. Nov, pp. 3137–3181, 2010.

[23] J. Ha and H. Choi, "A topology-guided path integral approach for stochastic optimal control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 4605–4612.

[24] S. Thijssen and H. Kappen, "Path integral control and state-dependent feedback," *Physical Review E*, vol. 91, no. 3, p. 032104, 2015.

[25] H. J. Kappen and H. C. Ruiz, "Adaptive importance sampling for control and inference," *Journal of Statistical Physics*, vol. 162, no. 5, pp. 1244–1266, Mar 2016.

[26] O. Arslan, E. A. Theodorou, and P. Tsiotras, "Information-theoretic stochastic optimal control via incremental sampling-based algorithms," in *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Dec 2014, pp. 1–8.

[27] R. F. Stengel, *Optimal control and estimation*. Dover Publications, 1994.

[28] Y. Tassa, T. Erez, and W. D. Smart, "Receding horizon differential dynamic programming," in *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds. Curran Associates, Inc., 2008, pp. 1465–1472.

[29] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.*, June 2005, pp. 300–306 vol. 1.

[30] M. B. Horowitz, A. Damle, and J. W. Burdick, "Linear hamilton jacobi bellman equations in high dimensions," *53rd IEEE Conference on Decision and Control*, pp. 5880–5887, 2014.

[31] E. Theodorou, J. Buchli, and S. Schaal, "Reinforcement learning of motor skills in high dimensions: A path integral approach," *2010 IEEE International Conference on Robotics and Automation*, pp. 2397–2403, 2010.

[32] P. L. Kempker, A. C. M. Ran, and J. H. van Schuppen, "A formation flying algorithm for autonomous underwater vehicles," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, Dec 2011, pp. 1293–1298.

[33] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 773 – 11 780, 2014, 19th IFAC World Congress.

[34] B. ksendal, *Stochastic Differential Equations: An Introduction with Applications*, 01 2000, vol. 82.

[35] H. J. Kappen, "An introduction to stochastic control theory, path integrals and reinforcement learning," in *AIP conference proceedings*, vol. 887, no. 1. AIP, 2007, pp. 149–181.

[36] E. Theodorou, F. Stulp, J. Buchli, and S. Schaal, "An iterative path integral stochastic optimal control approach for learning robotic tasks," vol. 18, no. PART 1, 2011, pp. 11 594–11 601.

[37] S. Satoh, H. J. Kappen, and M. Saeki, "An iterative method for nonlinear stochastic optimal control based on path integrals," *IEEE Transactions on Automatic Control*, vol. 62, no. 1, pp. 262–276, 2017.

Part II

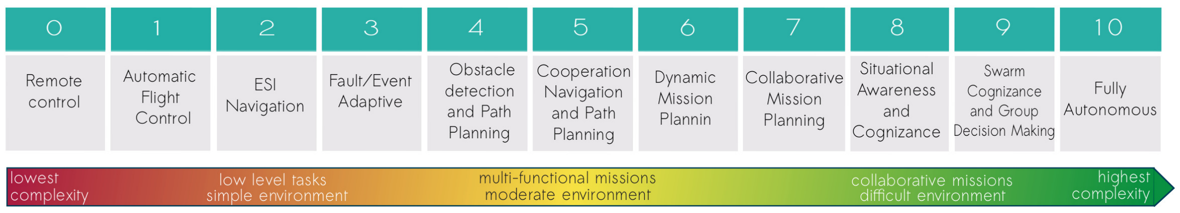
Literature study

Chapter 2

Path Planning

2-1 Introduction

To enable unmanned aerial vehicles (UAVs) to perform complex missions (Omidshafiei et al., 2017; Quintero et al., 2013) without human intervention, a certain level of autonomy is required. Different metrics and taxonomies for autonomy classification of UAVs can be found in literature (Kendoul, 2013; Sholes, 2007). Usually, these levels of autonomy are related with the capabilities of the guidance, navigation and control systems (GNC) of the UAV. Within the guidance system, path planning capabilities represent the first step towards a mid-level mission complexity and system independence (Figure 2-1). Leader-follower formation flight is already a more complex task that involves cooperation among the UAVs. Moreover, uncertain elements are taken into account as described in Section 1-1. In this scenario, pre-computed trajectories are not suitable anymore. The path planning problem has to be solved in real-time in order to cope with the uncertain environment and the collaborative task in which vehicles need to maintain a safe distance among each other. Thus, there is a growing interest in developing advanced guidance systems to enable autonomous trajectory generation for these kinds of complex missions and environments.



0
Remote control
1
Automatic Flight Control
2
ESI Navigation
3
Fault/Event Adaptive
4
Obstacle detection and Path Planning
5
Cooperation Navigation and Path Planning
6
Dynamic Mission Plannin
7
Collaborative Mission Planning
8
Situational Awareness and Cognizance
9
Swarm Cognizance and Group Decision Making
10
Fully Autonomous
lowest complexity
low level tasks simple environment
multi-functional missions moderate environment
collaborative missions difficult environment
highest complexity

Figure 2-1: ALFURS (Autonomy Levels for Unmaned Rotorcraft Systems) autonomy levels as described in (Kendoul, 2013). Acronyms: ESI (External System Independence)

Vehicle path planning is a fundamental problem in robotics which is in general hard to solve as the number of degrees of freedom increase (Kempker, Ran, & Schuppen, 2011; Jang, Chae, & Choi, 2017). Typically in a UAV application, the vehicle is modeled as a two to four degrees of freedom (DOF) system. The motion is performed in three dimensional space while fulfilling differential constraints and bounded velocities and acceleration values are used (Goerzen, Kong, & Mettler, 2010). Thus, the planning algorithm needs to solve the vehicle motion problem and generate feasible and, if possible, optimal paths within the high dimensional problem space. To account for dynamic, unpredictable environments, these paths must be computed in real-time.

In this section, the commonly used methods in UAV path planning are going to be reviewed. As real-time performance is an essential criterion for path planning algorithms, a brief description of the computational complexity of these techniques will also be given.

2-2 Methods and algorithms

Different classifications of the algorithms used to solve the path planning problem exist (Kendoul, 2012; Yang, Qi, Xiao, & Yong, 2014; Y. Liu & Bucknall, 2018; González, Pérez, Milanés, & Nashashibi, 2016). A distinction can be made between deterministic and heuristic methods, local or global path planners, 2D or 3D path planners and planning with or without differential constraints (Goerzen et al., 2010).

This section will use a similar classification as given in (Kendoul, 2012) and (Y. Liu & Bucknall, 2018). This choice is motivated by the fact that optimization methods, the type of methods that are going to be used throughout this work, appear as an explicit class in these classifications. The class of heuristic approaches and methods which take into account uncertainty mentioned in (Kendoul, 2012), are excluded from this classification as they can be considered as special cases of the remaining three big classes, namely road maps, potential fields and optimization methods. In addition, due to their relevance in multi-vehicle path planning, evolutionary algorithms and bio-inspired methods are added as a fourth class (Y. Liu & Bucknall, 2018). The classification used in this report is showed in Figure 2-2.

2-2-1 Road maps

Road maps (RM) methods solve the path planning problem by searching for the shortest path in a given road map or graph. The road map is constructed in such a way that it captures the problem of planning a trajectory through a given space. Each node of the graph represents a possible position of the vehicle and each edge corresponds to a collision-free path between nodes. Obstacles are defined as inaccessible points through which no edge can pass (Zammit & Van Kampen, 2018). Search algorithms such as Dijkstra's method, A* or D* (Ferguson, Likhachev, & Stentz, 2005) can be used to find the best path. Within the road maps methods, different ways of constructing the graphs exist such as visibility graphs, voronoi diagrams or probabilistic road maps.

Visibility graphs

Visibility graphs use a simple polygonal representation of the obstacles in the environment and take the obstacle's vertices as the graph vertices (Figure 2-3(a)). Edges are drawn between every pair of vertices if the line connecting them does not intersect with any obstacle. Given a

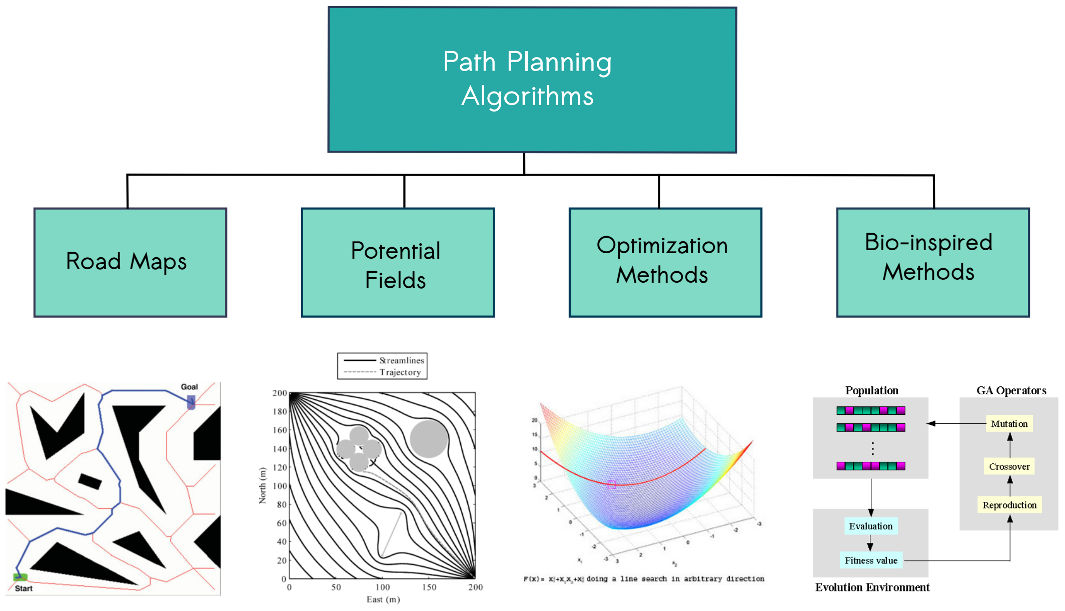


Figure 2-2: Path Planning Algorithms classification: Road Maps Voronoi example (García & Gómez-Bravo, 2012), Potential Fields example (Daily & Bevly, 2008), Optimization Methods line search graph example (How., 2008), Evolutionary Methods (Liao & Sun, 2001)

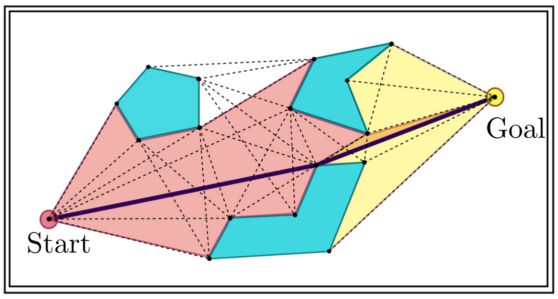
start and a goal position, edges are also drawn to all obstacle vertices that are "visible" from that start or goal position. Every pair of vertices that does not intersect with an obstacle is said to be visible. Shortest path search algorithms are then used to find the best trajectory. The obtained trajectory must be first adapted to fulfill all UAV system dynamics constraints and a safe distance from the obstacles must be ensured before implementing it. Although this method is generally used in 2D situations, a generalization to a minimum length path problem for an UAV in 3D space has been implemented in (Schøler, Cour-Harbo, & Bisgaard, 2012).

Voronoi diagrams

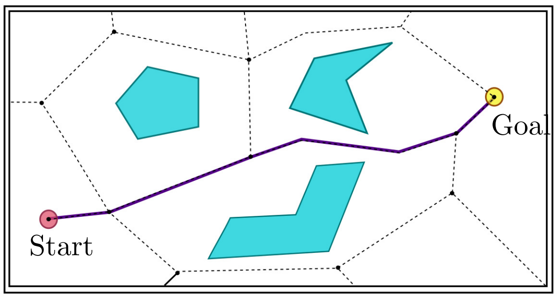
Voronoi diagrams divide the plane into polygonal regions each associated with a central point. The edges separating two regions are equidistant from two or more obstacles or boundaries of the surrounding area (Figure 2-3(b)). This partition is specially appealing for the path planning problem because opposite to the visibility graph approach, here the edges are located as far as possible from the obstacles. Similarly to the visibility graphs, a start and goal position are defined and connected to the voronoi diagram. The connection is done through the closest edge from the diagram. A Voronoi-based planner was implemented by NASA (Howlett, Whalley, Tsenkov, Schulein, & Takahashi, 2007) on their RMAX helicopter to navigate through a region with known obstacles. This method was also used in combination with a particle swarm optimization algorithm to obtain multi-UAV cooperative trajectory planning with time-constraints in (Tong, Chao, Qiang, & Bo, 2012). The Voronoi diagram was employed to find the initial trajectories which the PSO algorithm would then optimize to enable synchronized motion.

Probabilistic road maps

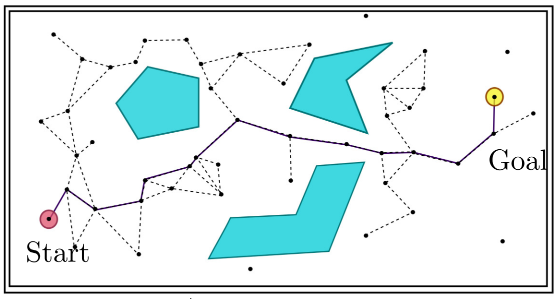
Probabilistic road maps (PRM) is one of the most used RM approach. It is a sample-based method that generates random points on a plane and tries to connect these points to the k-nearest neighbors without intersecting with any obstacle (Figure 2-3(c)). In a second stage, a start and goal location are added to the created road map and a search algorithm is used to find the shortest collision free path (Kendoul, 2012). This method can be used as an approximated solution for large or high-dimensional continuous spaces as it only takes into account the discrete randomly sampled points for the construction of the road map (P. Chen & Waslander, 2010). Although generally it is used for single vehicle path planning, the integration of PRM methods with flocking techniques has been demonstrated in (Bayazıt, Lien, & Amato, 2005). An improved, iterative counterpart of the PRM algorithm is the Rapidly-exploring Random-Tree (RRT) method. Instead of randomly sample the plane, this technique randomly expands a tree (a directed graph) from the initial point towards locations where no vertices have been constructed yet (Goerzen et al., 2010). This process is repeated until the goal position is reached. One of the main advantages of this method is that it can take into account the system's dynamics when computing the trajectory. However, these trajectories may still be suboptimal as was demonstrated in (Karaman & Frazzoli, 2011).



a) Visibility graph



b) Voronoi diagram



c) Probabilistic road maps

Figure 2-3: Road map techniques

2-2-2 Potential fields

Potential fields methods consider the vehicle as a particle subjected to the force of a potential field. The force fields can either be attractive forces used to guide the vehicle towards the goal, or repulsive forces meant to keep the vehicle far from obstacles or other undesired regions. The potential field is defined across the entire problem space from which the resultant induced force is then calculated. The vehicle's movement is generated as a consequence of applying the induced force (Kendoul, 2012).

One of the advantages of using these type of methods is their low computational requirements enabling them for real-time planning tasks. However, potential fields approaches are particularly prone to get trapped in local-minima. Due to its simple nature, situations in which the induced force equals zero are commonly encountered. Take for example the case of a MAV flying towards a goal waypoint and facing a U-shaped obstacle as represented in Figure 2-3. The counteracting forces in this case, will trap the vehicle in a point where the induced force is zero and no movement can be generated (Dudek & Jenkin, 2010).

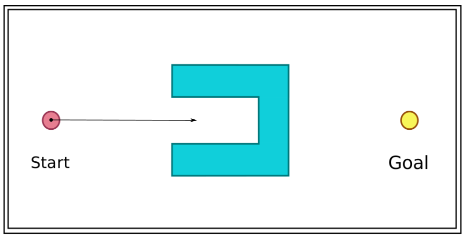


Figure 2-3: Example of local-minima with an U-shaped obstacle

Techniques to avoid these situations have been proposed such as using a combination of global and local planner (Scherer, Singh, Chamberlain, & Elgersma, 2008), formulate new types of fields such as the Harmonic Potential Fields (Panati, Baasandorj, & Chong, 2015) or employing Fast Marching Methods (FMM) to construct the potential field based on the propagation properties of electromagnetic waves (Garrido, Moreno, & Lima, 2011).

2-2-3 Optimization methods

Optimization methods formulate the path planning problem as an optimization problem in which a cost function needs to be minimized and where obstacles, vehicle dynamic limits and mission boundaries are formulated as mathematical constraints (Kendoul, 2012). When compared with previous mentioned techniques (road maps, potential fields), optimization methods can easily incorporate vehicle's dynamics constraints when computing the optimal trajectory. Moreover, while these methods would only compute feasible trajectories, optimization methods try to find the optimal path for a defined cost function. The main drawback of using these techniques is the high computational requirements needed to solve the path planning problem. This is specially problematic in cases where large or high-dimensional state spaces are used (Stengel, 1986). Ways to cope with this issue have been intensively studied within the robotics community (Gorodetsky, Karaman, & Marzouk, 2015; E. Theodorou, Stulp, Buchli, & Schaal, 2011; Vernaza & Lee, 2011) as all robot's actuation space is generally large, causing the application of optimization methods to be particularly hard as real-time performance is required. Despite the broad range of application, three main techniques have been identified to solve an optimization problem: dynamic programming methods, indirect methods and direct methods (Betts, 1998). A detailed explanation of all these approaches is given in Section 3-1.

2-2-4 Bio-inspired methods

Bio-inspired methods such as genetic algorithms (GA), particle swarm optimization (PSO) and ant colony optimization (ACO) can be used to find optimal trajectories for path planning problems (Y. Liu & Bucknall, 2018). Among these, genetic algorithms are the most popular ones and have been extensively used within the robotics community (C. Liu, Liu, & Yang, 2011; Roberge, Tarbouchi, & Labonte, 2013; Cheng, Sun, & Liu, 2011).

Using a GA approach, possible trajectories are treated as individuals of a population. Similar to a natural selection scenario, the best individuals of the population are selected to evolve, mutate and reproduce giving place to new individuals. A pre-defined fitness function is then used to evaluate the quality of the individuals and select the ones which have a better fitness value. The selection, mutation, evaluation sequence is repeated through many generations of populations until convergence is achieved and the optimal trajectory is retrieved from the best individuals.

It can be noted, that GA are a special case of optimization methods. In this case, the fitness function is equivalent to the cost function usually defined in optimal control problems. The main difference is that GA are heuristic search methods through which only an estimate of the optimal trajectories can be obtained.

2-3 Comparison of methods

Among the previously mentioned path planning techniques, a preference is seen towards potential fields, optimization methods and bio-inspired methods in multi-vehicle planning problems. Road map techniques are generally used for single vehicle path planning (Y. Liu & Bucknall, 2018). From these, potential fields methods and their simple formulation are generally used as collision avoidance techniques or reactive path planning. They are usually combined with other methods to enable more complex and collaborative tasks such as leader-follower flight (Kuriki & Namerikawa, 2014).

Both optimization methods (Kushleyev et al., 2013; Mansouri, Nikolakopoulos, & Gustafsson, 2015; Saska et al., 2014) and bio-inspired methods (Kala, 2012; Korayem, Hoshiar, & Nazarahari, 2016) are capable of computing optimal trajectories for multiple vehicles. However, evolutionary methods require problem-specific parameters and have no guarantees of optimality making them less appealing for path planning problems (S. M. LaValle, 2006). These observations motivate the choice of using an optimization method to solve the path planning problem for the leader-follower task. Therefore, the following sections will focus on stochastic optimal control methods and how they are implemented to enable autonomous path planning capabilities.

Chapter 3

Optimal Control

This chapter is a brief introduction to the broad field of optimal control with a particular focus on stochastic optimal control. First an introduction of the general concepts of optimal control and the main fields of application is given. Next, an example of optimal control implementation in robotic applications is presented and problem complexity aspects are discussed. The section continues with the mathematical foundation of both deterministic and stochastic optimal control where the fundamental optimality principle and Bellman equation are described. A classification of the principal methods of solving an optimal control problem, namely the Dynamic Programming method, the Indirect Method and the Direct Method is given. Finally, state-of-the-art techniques to solve stochastic optimal control formulations of the path planning problem for robotic applications are described.

3-1 Introduction to Optimal Control

To optimally control a dynamic system with respect to some performance criteria is the objective of many problems arising in a variety of fields (Powell, 2007). Examples ranging from optimizing the retirement decision problem (Rust, 1989), to vehicle fleet repositioning (Song & Earl, 2008) or optimal motion planning for robots (S. LaValle & Hutchinson, 1998) are all problems which can be solved through optimization methods. As described in Powell (2007), three main communities that have been using and developing these techniques can be identified:

- **Control theory community** has applied optimization techniques to well known engineering problems such as the control of chemical plants, power grids or controlling MAVs (Powell, 2012). Typically, it focuses on continuous time problems with continuous state and control parameters (Powell, 2007). The systems that need to be optimized usually have complex dynamics and large or high-dimensional state spaces (Bangura & Mahony, 2014; Todorov & Tassa, 2009). Generally only one system is controlled at the same time.

- **Operations research community** focuses on problems such as the management of resources, routing and scheduling of individual entities or inventory problems (Powell, 2012). These applications are generally formulated as discrete, stochastic problems where multiple, simple systems or entities (Song & Earl, 2008) are treated. Therefore, methods to deal with uncertainty have been exhaustively studied within this community, culminating in the vast theory of Markov Decision Processes (MDPs).

- **Computer science community** and particularly artificial intelligence community has also applied optimization principles to solve a certain class of machine learning problems known as Reinforcement Learning RL (Powell, 2007). Learning how to play games, describing animal behavior and even controlling robots are some examples of RL applications (Powell, 2012). Classic RL formulations also use MDPs as in Operation Research and thus, have been mainly used to solve discrete, stochastic problems (Abouheaf, Lewis, Vamvoudakis, Haesaert, & Babuska, 2014).

Despite the wide application range, the underlying concept is similar: one or more agents have to plan their actions over time subjected to a dynamical system whose states are affected by those actions while optimizing a desired performance criterion usually formulated as a cost function (Todorov, 2009).

The same principle can be applied to tasks from our daily life. Take for example a ping-pong game. The movement of the arm that will send the ball into the opponent's side of the net can be seen as an optimal control problem. In this example, the system is the human arm. This system follows complex dynamics and has many degrees of freedom. The cost function that has to be minimized includes, in this case, a path cost and an end cost. The path cost describes the effort needed to contract the muscles and perform the arm movement, while the end cost reflects the success of the action. It will assign a high cost to all cases in which the ball has not reached the opponent's side of the net. Thus, the optimal arm trajectory which minimizes the cost function is the one that needs the least amount of energy to perform the action and is still capable to correctly send the ball into the opponent's side.

Now imagine that a robot arm attempts to perform the same action (Koç et al., 2018; Serra, Satici, Ruggiero, Lippiello, & Siciliano, 2016). Achieving an optimal movement of the robot arm and guide it to hit a ping-pong ball into the opponent's side of the table is a complex task. Implementing optimal control strategies for systems that need to interact with uncertain environments (like the robot arm that needs to adapt its movement depending on the ping-pong ball location) or uncertain system dynamics (the imperfect robot arm actuators cannot perfectly execute the planned action) increases the complexity of the problem. These and other variables that affect the complexity of an optimal control problem are described in the next section.

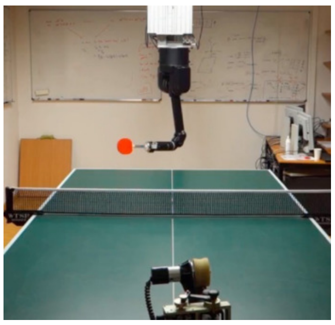


Figure 3-1: Ping-pong robot setup developed by (Koç et al., 2018)

3-1-1 Problem complexity

Knowing the amount of computation required by an optimal control algorithm is very important, especially for control tasks applied to robotics where real-time performance of the control algorithms is vital. In this section, the main aspects that affect the complexity of a control task are described.

- **Deterministic vs Stochastic**. Taking a deterministic or a stochastic approach is a decision that drastically changes the control approach that must be followed. Deterministic approaches assume systems have perfect state information and noiseless actuators. Stochastic approaches on the other hand, take into account the inherent noise of real systems. There are two main sources of system noise: noise coming from the actuators and noise coming from the sensor's measurements.

  Noisy actuators The control commands given to the robotic arm from the previous example or to a follower UAV that has to maintain a distance from the leader as in our task, will not result in perfect deterministic movement. Actuator noise coming from either electrical or mechanical sources will affect the resultant motion making it to some extent unpredictable (Thrun, Burgard, & Fox, 2005). This means that after each control input, the system's states can not be known precisely.

  Noisy measurements are a direct consequence of sensor's limitations. To obtain closed loop control, the corresponding states need to be measured. These measurements are not perfect as sensor's performance is affected by a variety of physical effects such as thermal or vibrational effects. If system uncertainty is taken into account, a stochastic or probabilistic approach of the control algorithm is needed. These algorithms must take into account all possible situations (e.g. the noisy actuator could have moved the robot in a different position than expected) increasing the complexity of the control algorithm.

- **Low vs High Dimensional Problem** High-dimensionality of state and action spaces is currently one of the most challenging and limiting factors for optimal control algorithms. In order to find the optimal actuator command, classic optimal control approaches have to compute the cost function for all possible states and actions (Stengel, 1986). This computation is feasible only for low-dimensional problems. When many possible states and actions are available, this computation increases exponentially making the optimal control problem intractable.

  High-dimensionality of the problem is inherent when the control algorithm is applied to a real-system (e.g. robot, UAV) which works with continuous inputs and output signals. Take for example most robotic applications which span over large continuous state spaces (e.g. the position and velocity space of the robot arm or of a UAV) or include multi-dimensional action spaces (e.g. controlling multiple joints of the robot arm) or a combination of both. Although this is inherently present in continuous-time systems, it can also be seen in discrete cases (Silver et al., 2017).

  Methods to solve the optimal control problem for these cases are presented in Section 3-2.

- **Simple vs Complex Systems** When solving an optimal control problem, a model describing how the system evolves over time is needed (Stengel, 1986). In control theory this is known as transfer function or system dynamics while in MDP formulations and model-based RL this represents the transition model. Models are abstraction of the real word that consider only partial information of the underlying physical processes (Thrun et al., 2005). This inaccuracy adds additional uncertainty to the control problem, making the prediction of future states harder. Moreover, complex systems are generally follow non-linear dynamics which increases the complexity of the control algorithm.

- **Total vs Partial state observation** As previously mentioned, sensors available to measure the states are ultimately limited and imperfect. Real sensors have a range within which measurements can be obtained. This sensor range limits the known environment and forces the system to work with partial information. Moreover, some states can be essentially unobservable or expensive measurement equipment could be needed to obtain the entire state value (Stengel, 1986). These problems are often encountered within the robotics community and their solutions are not particularly easy. In these cases, the control algorithm has to be adapted to work with incomplete information of the system's states. The probability of being in one state or another has to be calculated which increases the complexity of the control algorithm (Thrun et al., 2005).

- **Single vs. Multiple agents** In principle, optimal control can be extended to N dynamical systems by considering N coupled differential equations and by formulating their cooperative performance as a single cost function. However, the computational complexity needed to solve the coupled optimal control problem becomes prohibitive for large values of N (Foderaro & Ferrari, 2010). Thus, when multiple agents or systems have to be controlled at the same time, as is usually the case for swarm robotics applications, the control algorithm has to be designed in such a way that it scales with the number of systems without adding complexity. Distributed control algorithms are usually used in these cases (Guo & Parker, 2002).

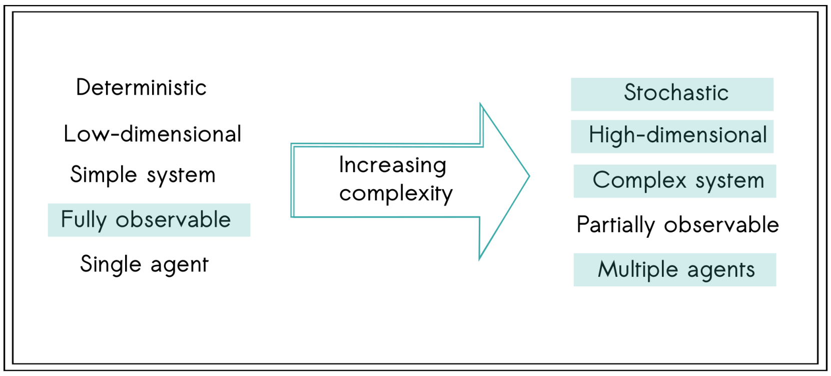


Figure 3-2: Leader-follower task complexity classification

As summarized in Figure 3-2, the leader-follower task described in Chapter 1 requires a stochastic continuous-time formulation of the optimal control problem that can scale to multiple complex systems acting in a fully observable environment. These requirements make the task particularly complex and limit the range of possible algorithms that can be employed.

3-1-2 Deterministic Optimal Control

Before describing the theoretical framework of stochastic optimal control theory, an introduction to the deterministic optimal control case is presented in this section. Although no real system can generally be called deterministic, this assumption simplifies the formulation of the optimal control problem. Following the notation used in (Kappen, 2007) a continuous, deterministic system is considered as following:

System Dynamics

$$\dot{x}(t) = f(x(t), u(t), t) \tag{3-1}$$

where the state vector at time t is denoted by $x(t)$, $u(t)$ is the vector of control variables at time t and $f(\cdot)$ represents the system dynamics as a function of both states and controls at time t. A fixed initial state is assumed as $x(t_i) = x_i$.

Cost Function

For a given task, a cost function over a finite time interval $(t_i \to t_f)$ can be defined by a *path cost* and an *end cost*. When both a path cost and an end cost are included in the cost function, it is usually referred to a problem of Bolza, or Bolza form (Betts, 1998). If the cost function contains only a path cost it is known as a problem of Lagrange, and when only the end cost is considered, it is known as a Mayer form (Betts, 1998). Using one form or the other, it is mainly related with the nature of the task itself. Regardless of the form, the cost function usually includes the desired performance constraints (e.g. minimum time, minimum energy), limitations of the states or controls (e.g. maximum velocities, maximum control inputs) and the desired goal (e.g. assigning less cost for positions closer to the goal waypoint, assigning a negative end cost when the robot arm hits the ping-pong ball).
Using the Bolza form, the cost function can be written as:

$$C(x_i, t_i, u(t_i \to t_f)) = \phi(x(t_f)) + \int_{t_i}^{t_f} L(x(t), u(t), t)dt \tag{3-2}$$

where $\phi(x(t_f))$ is the end cost or the penalty for ending with a certain value of the state space and $L(x(t), u(t), t)$ is the path cost. The integral over the path cost function represents the penalty of transitioning from the initial state x_i to the final state x_f when the control sequence $u(t_i \to t_f)$ is being applied. The objective is then to find a control sequence $u(t_i \to t_f)$ that minimizes the above cost function over the finite time interval.

Planning Horizon

So far, the controlled time interval has been taken as the finite interval $[t_i, t_f]$. However, different options are possible. The length of the controlled time interval is usually known as the *planning horizon* or *time horizon*, T. As explained in (Thrun et al., 2005), three main strategies for choosing the value of T can be distinguished:

Greedy case

1. T=1. A simple, *greedy strategy* consists of minimizing the cost function for the immediate next time step only. Although this approach is less computationally expensive, it does not take into account the effect of future actions, beyond the first time step.

Finite horizon

2. T = *finite*. A different approach is to take T higher than one but still finite. This is known as the finite-horizon case. Here, the optimal control is calculated for a longer time interval taking into account actions performed further in the future. The computed control sequence is then applied to the system in an open-loop fashion without taking into account the system's states. The execution of the control algorithm increases linearly with the value of T making it more difficult to use long planning horizons when real-time performance is needed.

   To benefit from a larger planning horizon while also enabling a closed-loop control and alleviating the running time requirements, a receding horizon or Model Predictive Control (MPC) technique can be applied (Stengel, 1986). Using a receding horizon framework translates in calculating the optimal control for the entire horizon but only applying the first control input of the computed sequence. At each time step t the optimal control is computed for the dynamic interval $[t, t+T]$ where T is generally a finite value larger than 1 as the goal is to achieve a longer planning horizon in a more efficient way. This method is constantly re-planning, meaning that the optimal control is re-calculated at each time step taking into account the new measured states. The approach follows a closed-loop scheme as state feedback is included in the computation of the optimal control inputs. Moreover, in this case the computed optimal control is time-independent. From the system point of view, nothing will change from one time step to another. The system faces the same problem over and over again losing the dependency with time (Kappen, 2007).

Receding horizon

Infinite horizon

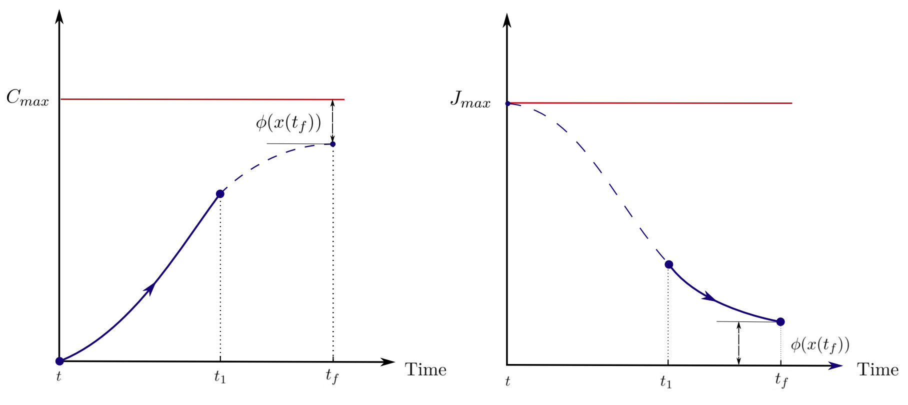
3. $T = \infty$. For the infinite-horizon approach, the optimal control is computed by minimizing the cost function over an infinite amount of time. In this case, the optimal control is time-independent as at each time step an infinite amount of remaining time-steps have to be taken into account. So, as in the receding-horizon case, the system always faces the same problem. This approach is typically used in reinforcement learning techniques. In RL problems, a reward function has to be maximized instead of minimizing a cost function as in control theory. Here, the difficulty consists in obtaining a finite value of the reward function given the fact that it has to be calculated for infinite time-steps. A common approach is to use a discounted factor γ ($0 < \gamma < 1$). With $\gamma < 1$, future rewards count less than immediate rewards. In this way, a bounded value of the cost function can be found. More detailed information on how this infinite-horizon problems can be solved can be found in (Sutton & Barto, 1998).

Optimal cost-to-go

The optimal control problem is solved when the control sequence $u(t_i \rightarrow t_f)$ that minimizes the cost function $C(x_i, t_i, u(t_i \rightarrow t_f))$ over the finite-time horizon $[t_i, t_f]$ is found (Stengel, 1986). Obtaining this control sequence requires for an additional function known as the optimal cost-to-go function or value function defined as:

$$J(x,t) = \min_{u(t \rightarrow t_f)} C(x, t, u(t \rightarrow t_f)) \tag{3-3}$$

As explained in (Stengel, 1986), the optimal cost-to-go function represents the cost of being in a state x at a time t and following an optimal control from that moment onwards. The difference between the optimal cost-to-go function and the cost function can better be seen if we plot their values from an initial time t to a final time t_f passing through an intermediate time t_1 (Figure 3-3).



a) Cost function b) Optimal cost-to-go function

Figure 3-3: Cost function vs. Optimal cost-to-go function

It can be noticed that when the cost function has a maximum value, the optimal cost-to-go function takes a minimum value. At time t_1 for example, the cost increases due to the accumulation of penalties along the trajectory that took the system from a state x at t to a state x_1 at t_1. On the other hand, the optimal cost-to-go at t_1 represents the remaining cost of the trajectory (path cost up to t_f + end cost) if an optimal control is applied from that moment onwards. The optimal cost-to-go represents a way of comparing how good a control sequence is. The optimal control sequence will be the one that minimizes the remaining cost.

Principle of optimality

To compute the Equation (3-3) for all states along the entire time interval, the *Principle of Optimality* developed by Bellman in the early 1950s can be applied (Bellman, 1957).

The *Principle of Optimality* simply states:

> *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision (Bellman, 1957).*

What this principle comes to say is that we can divide an optimal control problem in smaller sub-problems which can be solved separately and further combined to obtain the overall solution. Take for example the optimal state trajectory shown in Figure 3-4 which has been obtained by applying first the optimal control sequence from [t,t1) and then from [t1, tf]. Following the optimality principle, both sub-trajectories will continue to be optimal with respect to their initial state, (x, t) and (x_1, t_1) respectively. As this optimality is conserved, the optimal control sequence can be split and found separately.

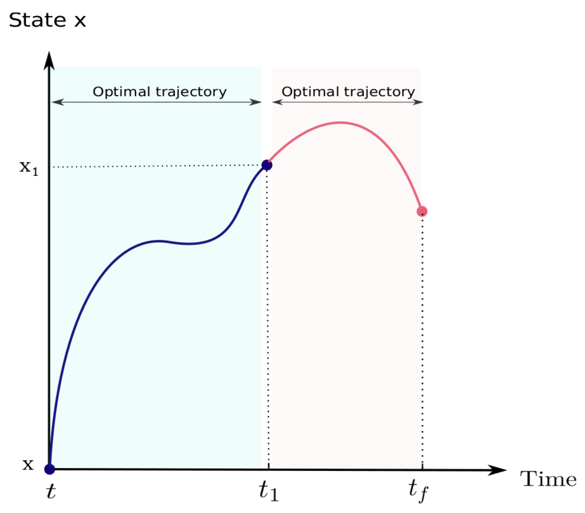


Figure 3-4: Optimal state trajectory when optimal control is applied

For the optimal cost-to-go equation, this means that we can split the equation at any point along the optimal trajectory. Taking as an example the intermediate point t_1 of the cost-to-go function as seen in Figure 3-2, by substituting the expression of the cost-function (3-2), the (3-3) equation can be rewritten as:

Bellman equation

$$\begin{aligned} J(x,t) &= \min_{u(t\to t_f)} \left(\phi(x(t_f)) + \int_t^{t_1} L(x(t),u(t),t)dt + \int_{t_1}^{t_f} L(x(t),u(t),t)dt \right) \\ &= \min_{u(t\to t_f)} \left(\int_t^{t_1} L(x(t),u(t),t)dt + \min_{u(t\to t_f)} \left((\phi(x(t_f)) + \int_{t_1}^{t_f} L(x(t),u(t),t)dt \right) \right) \\ &= \min_{u(t\to t_f)} \left(\int_t^{t_1} L(x(t),u(t),t)dt + J(x(t_1),t_1) \right) \end{aligned} \tag{3-4}$$

In the above equation the minimization interval is split in two, achieving a recursive expression of the optimal cost-to-go expression. This equation is valid for any intermediate time t_1 ($t < t_1 < t_f$) (Kappen, 2007) and is known as the *Bellman equation.* What is particularly interesting here is the recursive property that allows us to solve the initial problem by splitting it in smaller sub-problems. Thus, to calculate the value of $J(x,t)$, we just need to know the path cost of going from the initial state to the intermediate state x_1 at time t_1 and the remaining cost from that state onwards expressed by $J(x(t_1),t_1)$. Having $J(x,t)$ given in terms of $J(x(t_1),t_1)$ which represents a different value of x at a future time t_1, results in having to solve the above equation for all states x simultaneously. As it is not known which value of x is needed to obtain the optimal cost-to-go, the function has to be computed for the entire state space. Moreover, the computation is performed backwards in time due to the recursive property from $t_{f+1} \rightarrow t$ assuming $J(x,t_{f+1}) = 0$.

Dynamic Programming

Solving an optimal control problem through the recursive Bellman equation is known as *Dynamic Programming.* If this method is applied to continuous time problems, a time discretization is needed. Thus, the value of t_1 can be written as $t_1 = t + dt$ where dt is infinitesimally small in the continuous case. Substituting this into the $J(x(t_1),t_1)$ expression and using the Taylor series expansion around t we obtain:

$$J(x(t_1),t_1) = J(x(t),t) + \partial_t J(x(t),t)dt + \partial_x J(x(t),t)dx \tag{3-5}$$

Now, substituting again Equation (3-5) into the Bellman equation:

$$J(x,t) = \min_{u(t\rightarrow t+dt)} \Big(L(x(t),u(t),t)dt + J(x,t) + J_t(x,t)dt + J_x(x,t)f(x,u(t),t)dt \Big) \tag{3-6}$$

where the following equality $dx = f(x,u(t),t)dt$ has been used from Equation 3-1.

It can be noticed that now the minimization interval spans over an infinitesimal time period dt. In the limit case, this will approach zero reducing the minimization interval to a point-wise variable u at time t (Kappen, 2007). Dividing both sides by dt and taking the limit of $dt \rightarrow 0$, the terms can be rearranged and the equation can be written in a more compact way as:

Hamilton Jacobi bellman

$$-J_t(x,t) = \min_u \Big(L(x,u,t) + J_x(x,t)f(x,u,t) \Big) \tag{3-7}$$

Equation 3-7 is known as the *Hamilton-Jacobi-Bellman* (HJB) equation due to the close similarity of this approach with the Hamilton-Jacobi theory in classical mechanics (Bagchi, 1993).

This equation must be solved with the boundary condition:

$$J(x,t_f) = \phi(x(t_f)) \tag{3-8}$$

where $\phi(x(t_f)$ is the end cost as defined in Equation (3-2).

As it can be noticed, the HJB equation is a partial differential equation (PDE) that as Bellman's equation, has to be solved for each value of x simultaneously and backwards in time. The solution of this equation is the optimal cost-to-go function. If this solution is found, then the optimal control can be found as:

Optimal Control

$$u(x,t) = arg \min_u \Big(L(x,u,t) + J_x(x,t)f(x,u,t) \Big) \tag{3-9}$$

The optimal control is computed in open-loop. Strategies that allow close-loop implementations and methods to solve both Bellman's equation for discrete cases and the HJB equation for continuous cases are detailed in Section 3-2.

3-1-3 Stochastic Optimal Control

Classical robotics usually assume deterministic systems moving in deterministic environments (Thrun et al., 2005). Practical applications however occur in stochastic worlds with noisy systems. Take for example the simple task of a MAV that needs to plan its trajectory to reach a goal position as represented in Figure 3-5.

In the deterministic case, the optimal trajectory would be to just fly straight through the narrow corridor (Figure 3-5 (a)). An optimal control sequence could be computed and the system could blindly follow it without taking any measurements of the states as no uncertainty (neither from the environment, nor from the system) could deviate it from the optimal path. However, in uncertain environments (e.g. wind, moving obstacles) and with a noisy system (e.g. noisy actuators, noisy measurements) blindly following the control inputs will not result in the same trajectory as in the deterministic case. The AV's real position could deviate from the predicted one increasing the risk of colliding with the walls of the narrow corridor (notice the position error bounds in Figure 3-5 (b)). In this case, a safer, longer trajectory is chosen.

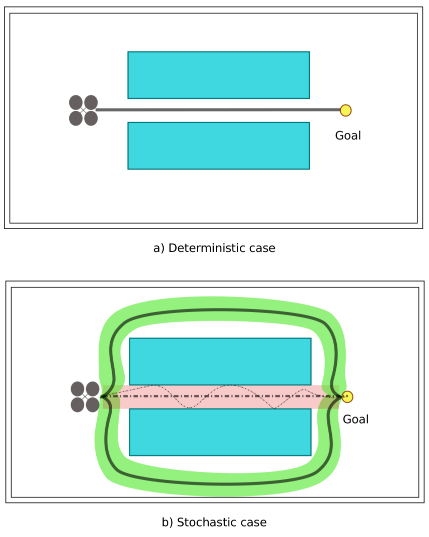


Figure 3-5: Deterministic vs. Stochastic task

In this section, an extension to the deterministic continuous optimal control problem is given for a system under stochastic effects. The optimal control equation is going to be re-defined for the case in which the system dynamics are formulated as a stochastic differential equation. In the general case, the stochastic differential equation can be formulated as:

Stochastic system dynamics

$$dx = f(x(t), u(t), t)dt + dw \tag{3-10}$$

Wiener process

where $f(\cdot)$ is an arbitrary function describing the system dynamics and dw represents an increment of a Wiener process $W(t)$ [a] with mean zero and variance

[a]A detailed explanation of the characteristics of Wiener process and its relation with other well-known stochastic processes is given in Appendix A-1.

$\langle dw^2 \rangle = \upsilon dt$, where υ is the variance of the noisy process dynamics which has been assumed to have no dependency with the system's states and controls. Consequently, the cost function is now written as the expectation value over all possible future realizations of the Wiener process:

Stochastic cost function

$$C(x_i, t_i, u(t_i \to t_f)) = \Big\langle \phi(x(t_f)) + \int_{t_i}^{t_f} L(x(t), u(t), t)dt \Big\rangle_{x_i} \tag{3-11}$$

where the subscript x_i highlights the fact that the expectation (denoted by the angle brackets) is taken for all stochastic trajectories that start in x_i (Kappen, 2007). Similar to the deterministic case, the optimal cost-to-go is defined as:

$$J(x,t) = \min_{u(t \to t_f)} \Big\langle C(x, t, u(t \to t_f)) \Big\rangle_x \tag{3-12}$$

The principle of optimality can be applied as described in the previous section and a recursive stochastic expression of the optimal cost-to-go is obtained as following:

Stochastic optimal cost-to-go

$$J(x,t) = \min_{u(t \to t_f)} \Big\langle \int_t^{t_1} L(x(t), u(t), t)dt + J(x(t_1), t_1) \Big\rangle_x \tag{3-13}$$

Once again taking $t_1 = t + dt$, the $J(x(t_1), t_1)$ expression can be expanded as a Taylor series around t. Applying $\hat{I}to$ stochastic calculus rules the function is expanded to first order in dt and second order in dx as following:

$$\begin{aligned} \Big\langle J(x(t+dt), t+dt) \Big\rangle &= J(x,t) + \partial_t J(x,t)dt + \partial_x J(x,t) f(x,u,t)dt \\ &+ \frac{1}{2}\partial_x^2 J(x,t)\upsilon dt \end{aligned} \tag{3-14}$$

The complete derivation of the above formula is given in Appendix A-2. Substituting equation (3-14) into the recursive expression of the optimal cost-to-go yields:

Stochastic HJB

$$-\partial_t J(x,t) = \min_{u(t \to t_f)} \Big\langle L(x,u,t) + f(x,u,t)^T \partial_x J(x,t) + \frac{1}{2}\partial_x^2 J(x,t)\upsilon \Big\rangle_x \tag{3-15}$$

Equation (3-15) is the Stochastic Hamilton-Jacobi-Bellman equation. As in the deterministic case, this equation has to be solved backwards in time for all states x and with boundary condition $J(x, t_f) = \phi(x)$. It can be noted, that for the case in which υ is zero, this equation reduces to the deterministic HJB equation as the second order term in dx is eliminated. The optimal control u can be found as:

$$u(x,t) = arg \min_{u(t \to t_f)} \Big\langle L(x,u,t) + f(x,u,t)^T \partial_x J(x,t) + \frac{1}{2}\partial_x^2 J(x,t)\upsilon \Big\rangle_x \tag{3-16}$$

3-2 Solving Optimal Control Problems

Given the wide range of applications in which optimal control theory has been used (Rust, 1989; Song & Earl, 2008; S. LaValle & Hutchinson, 1998), it is not surprising to see so many different methods to solve optimal control problems.

For some of these applications (e.g. optimizing satellite trajectories) accurate solutions are needed. However, due to the intrinsic nature of optimal control problems, a closed-form analytical solution is except for some specific cases (Suicmez & Kutay, 2014), rarely obtained. Most of the techniques, used to solve optimal control problems are numerical. Therefore, in this section, only the numerical methods are going to be described. Analytical solutions are going to be briefly mentioned as special cases problems for which closed-form solutions exist.

Among the numerical techniques, a strict classifications is almost impossible. However, big part of the optimal control community (Betts, 1998; Biral, Bertolazzi, & Bosetti, 2016; Frego, 2014) distinguishes between three main classes: Dynamic programming, Indirect methods and Direct methods. The following sub-sections will introduce the theoretical background, the different sub-classes as well as the advantages and disadvantages of each one of these classes. A complete overview of the classification followed in this report can be seen in Figure (3-6).

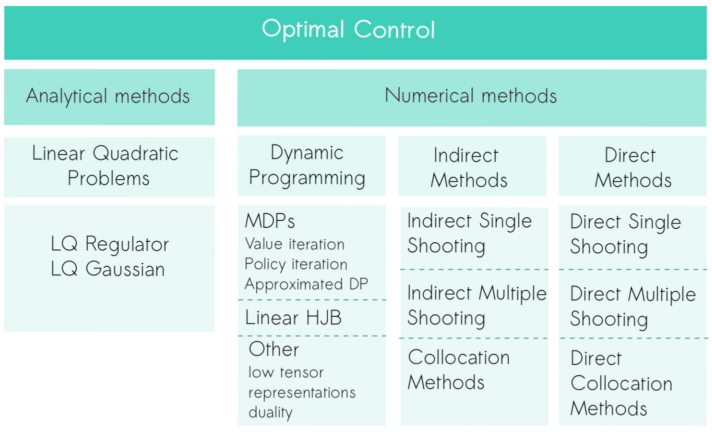


Figure 3-6: Optimal control analytical and numerical methods classification

3-2-1 Linear Quadratic Problems

For systems with linear dynamics and quadratic cost functions, an analytic closed-form solution of the optimal control can be found and thus, the optimal cost-to-go does not have to be computed (Stengel, 1986). In these cases, the optimal control is given in closed-form as the solution of a number of coupled ordinary differential (Ricatti) equations that can be efficiently solved (B. D. Anderson & Moore, 2007). This is known as the Linear Quadratic (LQ) problem or Linear Quadratic Regulator (LQR) and it is often used within the control community because many engineering problems can be modeled and solved following this approach (Kappen, 2007). If Gaussian noise is added to the linear system, the problem is known as Linear Quadratic Gaussian (LQG) and it is solved in a similar manner.

This approach can also be useful for non-linear systems. The nonlinearities can be linearized around certain points and the LQR or LQG approach can be employed. Some of the advantages of using this method is that an exact solution of the optimal control can be found. Moreover, this solution is given as state-feedback enabling closed-loop control.

However, most applications involve systems with non-linear dynamics that cannot be easily linearized. For these cases, a closed-form solution of the optimal control cannot be found. Therefore, numerical methods have to be employed.

3-2-2 Dynamic Programming

Using Dynamic Programming techniques, the solution of an optimal control problem is found by solving the recursive Bellman equation for discrete problems or the HJB equation for continuous problems. As mentioned in the previous chapter, these equations have to be solved for the entire state space. By computing the optimal cost-to-go for all states, a global comparison among all the different possibilities is performed. Thus, this techniques is a global optimization technique and provides necessary and sufficient conditions that ensure the optimality of the solution (Stengel, 1986).

Different approaches to solve the recursive Dynamic Programming equation can be taken depending on the type of problem (discrete vs. continuous, deterministic vs. stochastic). This section will briefly introduce some of the main approaches seen in literature.

Markov Decision Process Framework

As big part of the Dynamic Programming theory was developed within the Operations Research community, large part of the methods used to solve optimal control problems are formulated as MDPs. Consequently, these methods solve discrete stochastic problems as usually encountered in the Operations Research field (e.g. vehicle fleet scheduling, inventory problems). Nonetheless, this approach can also be used to solve continuous optimal control problems such as the ones treated within the control theory community. For these cases, the continuous states and controls have to be discretized and the problem has to be modeled as a MDP. Before entering into the details of how the optimal control is found with this technique, an introduction to MDPs and its nomenclature is given.

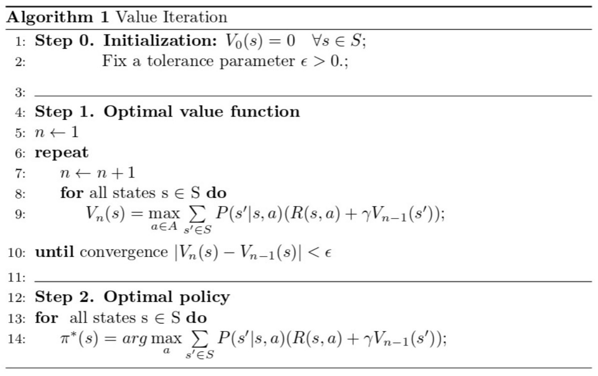
MDPs provide a way to model the decision making process in discrete stochastic settings. Unlike control theory problems, the system's states and actions are discrete. The system dynamics are now expressed as the probability to transition from state S_t to a state S_{t+1} when the action a_t is applied. Instead of minimizing a cost function $C(x_t, u_t)$ over a finite horizon, the objective now is to maximize a reward function $R(s_t, a_t)$ usually, over an infinite horizon. Moreover, a control policy is defined as a mapping function that assigns actions to states. The main goal is to find the optimal policy that will assign the optimal action when the system is in a given state. Table 3-2-2 summarizes these main nomenclature differences between the control theory and the MDP framework.

is a minimum **Control Theory**	**MDP Framework**
continuous states x	discrete states S
continuous controls u	discrete actions a
system dynamics $f(x, t)$	transition probability $p_t(S_{t+1}\|S_t, a_t)$
cost function $C(x_t, u_t)$	reward function $R(s_t, a_t)$
optimal cost-to-go $J(x_t, u_t)$	value function $V(s_t, a_t)$
-	control policy $\pi : S_t \rightarrow a_t$
finite horizon T	infinite discounted horizon

To obtain the optimal control policy, the main algorithms used within the MDP framework are value iteration and policy iteration algorithms (Powell, 2007).

Value Iteration

Value iteration is a direct application of the recursive Bellman equation. It estimates the value function iteratively by solving Bellman's equation backwards in time for all states (Powell, 2007). The value iteration algorithm is showed in Figure (3-7) for an MDP with discounted infinite horizon (note the discount factor γ). This algorithm has been proved to converge to its optimal value (Bellman, 1957). Thus, the optimal value function can be found and further employed to obtain the optimal policy.



```
Algorithm 1 Value Iteration
Step 0. Initialization: $V_0(s) = 0 \quad \forall s \in S$;
            Fix a tolerance parameter $\epsilon > 0$.;

Step 1. Optimal value function
$n \leftarrow 1$
repeat
    $n \leftarrow n + 1$
    for all states s ∈ S do
        $V_n(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)(R(s,a) + \gamma V_{n-1}(s'))$;
until convergence $|V_n(s) - V_{n-1}(s)| < \epsilon$

Step 2. Optimal policy
for all states s ∈ S do
    $\pi^*(s) = arg \max_{a} \sum_{s' \in S} P(s'|s,a)(R(s,a) + \gamma V_{n-1}(s'))$;
```

Figure 3-7: Value iteration algorithm

Policy Iteration

Policy iteration obtains the optimal policy directly rather than finding it by means of the optimal value function (Powell, 2007). Instead of updating the value function, this algorithm updates the policy function. As it can be seen in Figure (3-8), an arbitrary policy is first selected. Then, the policy is evaluated by computing the value function for all states when the actions described by this policy are used. The next step is to try to improve the policy. The policy is iteratively modified for all states with the hope of finding a new improved policy. If an improvement is achieved, the algorithm updates the old policy and keeps iterating until no better policy can be found. In terms of convergence, this approach has been found to arrive to the optimal value earlier than the value iteration algorithm (Powell, 2007).

```
Algorithm 2 Policy Iteration
Step 0: Initialization: Set $\pi$ arbitrarily;
repeat
    noImprovement ← $True$
    ________________________________________
    Step 1: Evaluate
    for all states s ∈ S do
        Solve $V(s) = \sum_{s' \in S} P(s'|s, \pi(s)(R(s, \pi(s)) + \gamma V_{n-1}(s'))$;
    ________________________________________
    Step 2: Improve
    for all states s ∈ S do
        Best = $V(s)$
        for all actions a ∈ A do
            $V_a(s) = \sum_{s' \in S} P(s'|s, a(R(s, a) + \gamma V_{n-1}(s'))$;
            if $V_a(s) > Best$ then
                noImprovement ← $False$
                $\pi(s) \leftarrow a$
until noImprovement
return $\pi$
```

Figure 3-8: Policy iteration algorithm

Although these algorithms can obtain the optimal policy for systems with a low number of states and actions, computational problems start to appear when high-dimensional states and actions spaces are used (Powell, 2007). This observation is everything but new as Bellman itself warned about the issues of applying dynamic programming methods to high-dimensional systems in the early 1950s (Bellman, 1957). He named this problem *the curse of dimensionality*.

Since then, many alternative methods have been developed that try to overcome the limitations of dynamic programming (Powell, 2007; Sutton & Barto, 1998). Knowing that computing Bellman's equation backwards in time will increase the computational requirements exponentially with the number of states, a forward computing method has been proposed. This approach uses iterative algorithms to approximate the value function. The main differences between performing a forward dynamic programming or a backward dynamic programming are as explained in (Powell, 2007) related with:

- **Making decisions**. When using backward dynamic programming, the recursive Bellman equation allowed us to compute the exact value function that was further used to obtain the optimal control. Now, if we step forward in time, this equation can no longer be used. Therefore, a new way of achieving the optimal controls or decisions is needed. What is generally done is to obtain an approximation of the value function and use this estimate $\widetilde{V_t(s_t)}$ to compute the optimal decisions. This saves us of having to loop over all possible states to compute the exact value function, but introduces the new problem of how to obtain a good estimate of the value function.

- **Stepping forward in time**. Using the optimal decision found with the approximated value function, a forward step in time can now be made from s_0 to s_1. However, it is not known how the stochastic effects will affect the system in this future time step. Therefore, we need a way to represent this missing information that arrived between t=0 and t=1. To solve this problem, samples of this random information are needed. The easiest way to obtain them is by modeling the stochastic variable and sample from this modeled distribution.

If we can approximate the value function and we can obtain samples of the missing information, we can step forward in time repeatedly and evaluate the performance of our decisions. The step forward is repeated until we reach the end of a finite time horizon. After that, the process is again repeated with different samples and the performance is compared with the previous iteration. The optimal decisions or actions are found by repeatedly trying different actions and keeping track of the ones which worked best.

This strategy has been used in a variety of fields and it has been referred to with a diversity of names such as forward dynamic programming (Kuffner, Nishiwaki, Kagami, Inaba, & Inoue, 2001), iterative dynamic programming (Field & Stepanenko, 1996), heuristic dynamic programming (Qiao, Harley, & Venayagamoorthy, 2009), neuro-dynamic programming (Bertsekas & Tsitsiklis, 1995) or reinforcement learning (Sutton & Barto, 1998) within the computer science community. The differences among these methods usually lie in the type of value function approximation used: lookup tables or aggregation techniques (Bertsekas & Castanon, 1989), parametric representations which require an initial design of the value function as a collection of *features* or *basis functions* the weight of which must be estimated (e.g. support vector regression (Bethke, How, & Ozdaglar, 2008)) or nonparametric ones which avoid the manual feature design and use observations to obtain a local approximation of the value function (e.g. neural networks[1] (Prokhorov & Wunsch, 1997)). However, they can all be grouped into what is commonly known as *approximated dynamic programming* techniques (Powell, 2007).

Linear HJB

For continuous-time optimal control problems, a different approach has been proposed recently (Kappen, 2005) to overcome the curse of dimensionality. It consists of providing a linear expression for the stochastic HJB equation through which the direction of computation can be inverted. As in the discrete case, a forward time computation for a special type of continuous-time system can be achieved, eliminating thus, the loop over the entire state space. A more detailed explanation can be found in Chapter 4 of this report.

Other

Other methods to overcome the curse of dimensionality consist in achieving a low-rank tensor representation[2] of the system model or the HJB expression. The computational requirements needed for this method scale linearly with increasing dimensionality of the state space (Gorodetsky et al., 2015). In addition, some explore the relationship of duality between non-linear stochastic systems and estimation techniques to solve optimal control problems (R. P. Anderson & Milutinović, 2014; Todorov, 2008). This relationship is well-known for LQG problems as Kalman discovered (Kalman, 1960) but has been quite difficult to extend for non-linear systems.

[1]Note that neural networks do need estimate parameters. Here it is categorized as nonparametric because no initial parametric model of the value function is needed.

[2]These methods convert a multidimensional function (e.g. the HJB) into a multidimensional array or tensor with the objective of reducing its dimensions and ease the computational requirements needed to solve the optimal control problem.

3-2-3 Direct methods

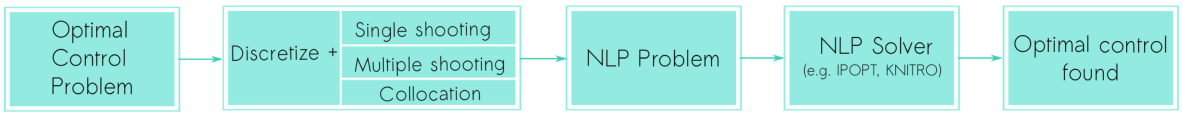


Figure 3-9: Process of solving an optimal control problem with direct methods

Direct methods solve the continuous-time optimization problem by applying a parametrization of the control and state spaces. Once the parametrization is performed, the optimal control problem can be written as a Non-Linear Programming (NLP) problem which can be solved with off-the-shelf software.[3] To obtain a NLP formulation of the optimal control problem different techniques can be used (Figure 3-9). The main differences lie in the variables to be parametrized (e.g. control and states) and how the continuous system dynamics are approximated (Biral et al., 2016). Among these techniques, two main classes are identified:

- **Shooting methods** represent one of the most simple ways of formulating an optimization problem. Their early applications involved the task of hitting a target with a cannonball, hence explaining the origin of the name (Betts, 2010). In this simple task, an initial quantity of gunpowder and an initial angle were chosen to try to reach the target. If the target was not reached, these simple control variables were adjusted iteratively until the target was reached optimizing for example the amount of gunpowder used.

  For a more complex optimal control problem, this translates in parameterizing the control space and using gradient descend methods to iteratively update the parameters in order to minimize a certain cost function. If the optimal control problem is defined over a finite time interval T, starting from an initial guess of the control parameters, the system dynamics differential equations are solved via numerical integration. At the end of the interval, the boundary or end cost condition is checked. These steps are repeated until the end error is minimized (Betts, 2010). As the iteration is performed in one step, from time t_i to t_f, this method is known as *single shooting method.*

  In the single shooting method the initial guess has an important effect on the final error at time t_f. To overcome this problem *multiple shooting methods* have been proposed. Unlike single shooting methods, multiple shooting divides the integration interval in several segments. In this way, the initial guess errors are propagated over a smaller time interval before they are modified again at the beginning of the following segment. A comparison between the two methods is given in Figure 3-10 (a)-(b).

- **Collocation methods** follow a similar procedure as shooting methods; however, now both states and control spaces are parametrized and the continuous time differential equations describing the system dynamics are converted into algebraic constraints that must be satisfied at each collocation point (Figure 3-10 (c)). This approach increases the number of variables of the NLP problem. However, the sparse structure of the NLP can be exploited alleviating thus the complexity of the problem.

[3]Some of the commonly used software are: IPOPT, KNITRO, WORHP (Paiva, 2014)

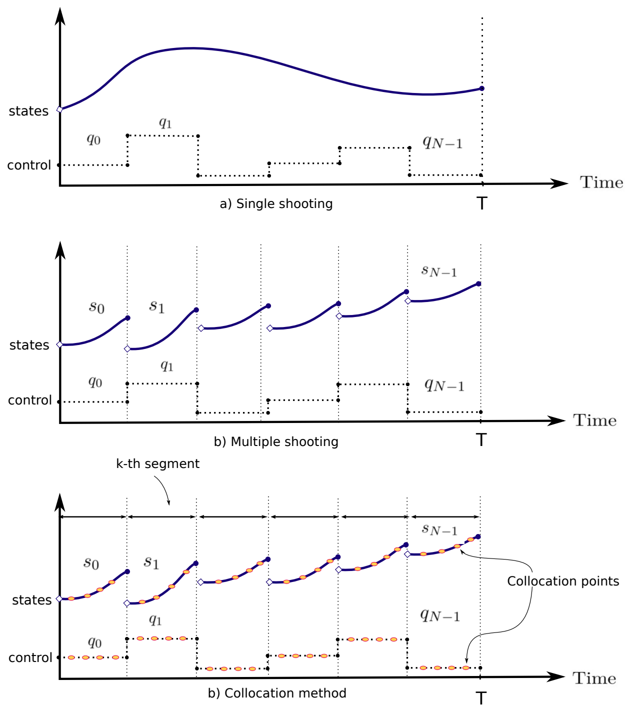


Figure 3-10: Shooting vs. Collocation methods. (a) Parametrized controls are input over a time interval T. The states trajectory is obtained through numerical integration in a single shooting manner. (b) Controls and states are parametrized. The state trajectory is obtained over the discrete time intervals in a multiple shooting fashion. (c) Collocation points are added to the discrete controls and states. Constraints have to be now fulfilled at each collocation point.

3-2-4 Indirect methods

Using indirect methods, the optimal control is found by satisfying a set of optimality conditions instead of minimizing a cost function as in Dynamic Programming methods. This approach of optimal control is related to the field of calculus of variations and is by far the oldest one when compared to Dynamic Programming or Direct methods. It first appeared in the early 17th century when Johan Bernoulli applied calculus of variations theory to solve the optimal path problem of a particle traveling in a gravitational field (Passenberg, 2012). However, it was the Russian mathematician Lev Pontryagin and his coworkers that in 1956 formulated the necessary optimality conditions for an optimal solution to be found (Passenberg, 2012).

Pontryagin Minimum Principle

His theory, also known as Pontryagin's Minimum (or Maximum) Principle is based on the fact that *necessary and sufficient conditions* have to be satisfied to ensure the optimality of a solution. If *necessary conditions* are satisfied, the cost function is stationary, meaning that small perturbations of the control input should have small effects that will vanish in time on the cost function (Stengel, 1994). For a given continuous-time system:

$$\dot{x}(t) = f(x(t), u(t), t) \tag{3-17}$$

the following cost-function has to be minimized over the fixed interval $[t_i, t_f]$

$$C(t_i, x_i, u(t_i \to t_f)) = \phi(x(t_f)) + \int_{t_i}^{t_f} L(x(t), u(t), t)dt \tag{3-18}$$

Necessary Conditions For Optimality

For this system, the necessary conditions for optimality are:

$$\begin{cases} \dot{\lambda}^T = -\frac{\partial \mathcal{H}}{\partial x} \\ \lambda^T = -\left.\frac{\partial \phi}{\partial x}\right|_{t=t_f} \\ \frac{\partial \mathcal{H}}{\partial u} = 0 \end{cases} \tag{3-19}$$

where λ describes the Lagrange multipliers of a constrained optimization in which the system dynamics Equation (3-17) is the equality constraint. The $\mathcal{H}$ represents the Hamiltonian defined as:

$$\mathcal{H}(x(t), u(t), \lambda(t), t) = L(x(t), u(t), t) + \lambda^T(t)(f(x(t), u(t), t) \tag{3-20}$$

A complete derivation of the necessary conditions for optimality is given in Section B.
If a trajectory is found to satisfy these necessary conditions, this trajectory is called an *extremal* because it still remains to determine if it is a minimum or a maximum of the cost function. To resolve this ambiguity, higher-order sensitivity evaluation can be performed. For example, if the control perturbations applied at any time the fixed time interval $[t_i, t_f]$ can only increase the cost, then we know that the stationary solution is a minimum (Stengel, 1994). This condition is expressed as:

$$\mathcal{H}(x^*(t), u^*(t), \lambda^*(t), t) \leq \mathcal{H}(x^*(t), u(t), \lambda^*(t), t) \tag{3-21}$$

Minimum Principle

where $x^*(t)$ is the optimal state trajectory, $u^*(t)$ is the optimal control trajectory and $\lambda^*(t)$ is the corresponding Lagrange multipliers vector. Equation (3-21) represents the Minimum Principle of optimality and gives necessary and sufficient conditions to find the optimal solution. The optimal control problem can then be written as:

$$\begin{cases} x(t_i) = x_i & & \textit{initial value} \\ \dot{x}(t) = f(x(t), u(t), t), & t \in [t_i, t_f] & \textit{state equation} \\ \dot{\lambda}^T = -\frac{\partial \mathcal{H}}{\partial x} & t \in [t_i, t_f] & \textit{adjoint equation} \\ \frac{\partial \mathcal{H}}{\partial u} = 0 & t \in [t_i, t_f] & \textit{control equation} \\ \lambda^T = -\left.\frac{\partial \phi}{\partial x}\right|_{t=t_f} & & \textit{adjoint final value} \end{cases} \tag{3-22}$$

Boundary Value Problem

Thus, through indirect methods, an optimal trajectory is achieved when the above set of equations are fulfilled simultaneously with the boundary constraints defined at t_i and t_f. In this case the the optimality conditions leads to a two-point Boundary Value Problem (BVP). The BVP can be solved analytically for some special cases of optimal problems with linear systems and quadratic cost functions but in most situations numerical methods have to be implemented to iteratively obtain the solution of the BVP (Passenberg, 2012). Similarly to direct methods, shooting and collocation techniques can be used to obtain a NLP formulation of the optimal control problem. However, here the state and adjoint variables must be initially guessed and the state and adjoint differential equations must be integrated over time. Unlike direct methods where the initialization errors are mitigated through multiple shooting techniques, here this initialization remains problematic. The main difficulty is to guess the value of the adjoint variable which has no physically intuitive interpretation. This is one of the main reasons why indirect methods are harder to implement and thus less used in literature. They require a deeper knowledge of the optimal control problem to formulate the adjoint equations.

3-3 Stochastic Optimal Control Approaches to Path Planning

Among the methods explained in Section 3-2, LQG strategies are mostly employed for accurate trajectory following, in which the trajectory is already given. Despite seeing some approaches which study the combination of both control and planning methods using LQG and RRT methods (Berg, Abbeel, & Goldberg, 2011), the linear-quadratic assumption limits its use for complex non-linear systems. When comparing direct and indirect methods, a clear preference for direct methods is noticed. As the software designed to solve NLP problems in direct methods is continuously improving, it becomes increasingly easier to implement an optimal controller without much knowledge of the underlying control theory. This method has been applied to control 16 MAVs flying in formation in (Kushleyev et al., 2013) by solving a Mixed-Integer Quadratic Program. Similarly, a Sequential Convex Programming problem was solved to control another formation of 16 MAVs while avoiding static and dynamic obstacles (Alonso-Mora, Montijano, Schwager, & Rus, 2016). Despite the easy implementation, these methods require state and controls discretization and are generally implemented off-board, in a centralized manner.

Within the large range of Dynamic Programming methods, most approaches model the optimal control problem as a MDP which can be solved via value iteration or policy iteration algorithms (Munishkin, Milutinović, & Casbeer, 2016). Generally, the systems are considered to have observable states over the entire environment, but POMDPs have also been used to model partially-observable system states (Capitan, Merino, & Ollero, 2014). The classic MDP formulation that still requires backward computation of value function, is known to be computationally expensive when high-dimensional states and controls are used. Approximated dynamic programming methods have been proposed as one of the alternatives to try to overcome this problem. Examples of such implementations can be found in (Tassa, Erez, & Smart, 2008; Todorov & Li, 2005). An alternative is the recent line of investigation into linear HJBs. Successful implementations have been shown for high-dimensional systems in combination with a MPC strategy (Gómez et al., 2015), with low-tensor representation of system dynamics (Horowitz, Damle, & Burdick, 2014) or within a RL framework (E. Theodorou, Buchli, & Schaal, 2010b).

Chapter 4

A Path Integral Control

4-1 Introduction to Path Integrals

Path integral techniques were first used within the field of quantum mechanics to provide an alternative method to compute the probability of a particle being in a particular state (Feynman, 1948; Perepelitsa, n.d.). Given a particle traveling from a point A to a point B (Figure 4-1), Feynman proved that the state evolution over time of the particle while traveling from A to B could be computed as a sum over all possible trajectories that the particle could take to go from A to B. This is known as the *path integral formulation* of quantum mechanics (Feynman, 1948).

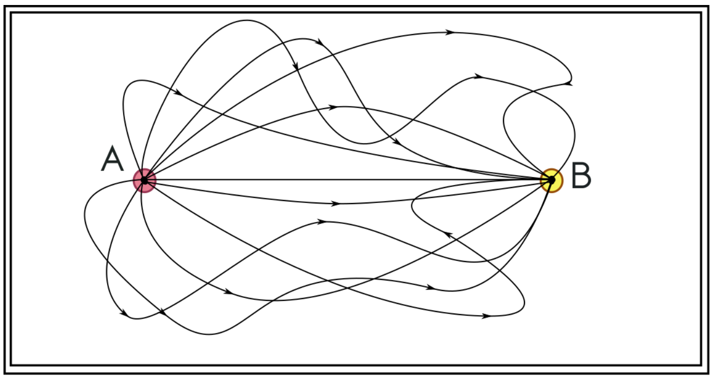


Figure 4-1: Multiple possible trajectories of a particle traveling from A to B

The possibility of applying a similar formulation to obtain the optimal state trajectory for a particular type of continuous-time stochastic control problems was recently introduced by Kappen (2005). His work shows how assuming a certain structure of the system dynamics, a linear stochastic HJB equation can be obtained and further expressed in terms of a path integral. The optimal control is then computed as the sum of contributions of all possible state trajectories of the system over a finite time interval. A similar formulation was also showed for discrete-time problems (Todorov, 2009).

In the following sections, the derivation of the linear HJB as well as the path integral formulation of the optimal control problem is showed. Finally, methods for solving the path integral are explained.

4-1-1 The linear HJB

Obtaining a linear expression of the HJB equation has been demonstrated in (Kappen, 2005) for continuous-time stochastic systems with linear dynamics and quadratic cost in the controls u. The linearity translates in having the $f(x, u, t)$ function from Equation (3-10) as:

Linear in the control

$$f(x, u, t) = f(x, t) + Bu \tag{4-1}$$

where $f(x, t)$ denotes the passive dynamics and can be arbitrary complex and B is the control matrix.
Next, the quadratic relation of u is expressed in the path cost function $L(x, u, t)$ from Equation (3-11) as:

Quadratic in the control

$$L(x, u, t) = q(x, t) + \frac{1}{2}u(t)^T Ru(t) \tag{4-2}$$

where $q(x, t)$ is an arbitrary function of x and t and R is a positive definite matrix.
Recall the stochastic HJB expression (3-15) from Section 3-1-3. If we now derive the HJB for the above system we obtain:

$$-\partial_t J(x, t) = \min_{u(t \to t_f)} \left\langle q(x, t) + \frac{1}{2}u^T Ru + (f(x, t) + Bu)\partial_x J(x, t) + \frac{1}{2}\partial_x^2 J(x, t)v \right\rangle_x \tag{4-3}$$

As explained in (Kappen, 2005), an explicit expression of the control u can now be derived due to the linear-quadratic relation of u. Thus, taking the gradient of the expression inside the parenthesis (4-3) with respect to u and set it to zero we obtain the following expression of the optimal control:

Explicit control

$$u^*(x, t) = -R^{-1}B^T \partial_x J(x, t) \tag{4-4}$$

Equation (4-4) can be interpreted as the optimal control $u^*(x, t)$ moving the system towards the direction of the minimum optimal cost-to-go $J(x, t)$. It can be noticed that the optimal control is proportional to the negative direction of the gradient of the optimal cost-to-go, projected on the state space x and weighted with the inverse of the control cost matrix R (E. A. Theodorou, 2011). Substituting this expression in Equation 4-3 results in a non-linear, second order PDE of the HJB as the following one:

$$-\partial_t J(x,t) = -\frac{1}{2R}\big(\partial_x J(x,t)^T B \partial_x J(x,t)\big) + q(x,t) + f(x,t)\partial_x J(x,t) + \frac{1}{2}\partial_x^2 J(x,t)\upsilon \tag{4-5}$$

To compute the optimal control, this PDE must be solved for the value of $J(x,t)$ and then, compute its gradient as showed in Equation (4-4). As explained in Section 3-2, except for some specific cases (e.g. LQ problems), solving this PDE is challenging, specially when high dimensional state spaces are used. As has been found in (Kappen, 2005; Todorov, 2009), an alternative method to solve the PDE can be used if a *desirability* function $\Psi(x,t)$ exits, such that the optimal cost-to-go can be written as follows:

DESIRABILITY FUNCTION

$$J(x,t) = -\lambda log \Psi(x,t) \tag{4-6}$$

LOG TRANSFORM

where λ is a constant. As will be explained in the following section, the desirability function $\Psi(x,t)$ can be efficiently computed by sampling from a diffusion process. Thus, the curse of dimensionality can be avoided if we use the relation of Equation (4-6). This logarithmic transformation has its origin in the field of quantum mechanics. A more detailed explanation of this analogy is given in Appendix C-1.

Taking the corresponding gradients of the new optimal cost-to-go expression (4-6) and assuming that $\upsilon = \lambda B R^{-1} B^T$, the HJB equation can be re-written in terms of $\Psi(x,t)$ as:

$$-\partial_t \Psi(x,t) = \Big(-\frac{q(x,t)}{\lambda} + f(x,t)\partial_x + \frac{1}{2}\partial_x^2 \upsilon\Big)\Psi(x,t) \tag{4-7}$$

Nonlinear terms successfully cancel out due to the assumption made and Equation (4-7) is now linear in $\Psi(x,t)$. Similar to previous HJB expressions, this equation must be computed backwards in time and fulfill the end-cost constraint $\Psi(x,t_f) = exp(-\frac{1}{\lambda}\phi(x))$.

CONTROL AND NOISE RELATION

Obtaining this linear expression of the HJB has added the constraint of having the matrices υ and $BR^{-1}B^T$ proportional with proportionality constant λ. Assuming B as an identity matrix, we can see that υ is inversely proportional to the control matrix R ($\upsilon \approx R^{-1}$). Intuitively, this can be seen as having expensive controls (R is large) in low noise directions and cheap control (R is small) in high noise directions (Kappen, 2005). This means that only small control inputs are allowed in low noise directions, with the limiting case of no control being allowed at all in noiseless directions. Therefore, to fulfill this constraint but avoid the imposed control restrictions, the control and the noise should operate in the same dimension (Kappen, 2005).

4-1-2 A Path Integral formulation

FEYNMAN KAC LEMMA

The second order, linear HJB equation (4-7) obtained in the previous section has the same structure as the Kolmogorov backward equation [a] and thus, it can be solved applying the Feynman-Kac lemma (Kappen, 2005; E. A. Theodorou, 2011). The Feynman-Kac lemma gives a connection between stochastic differential equations (SDE) (or diffusions) and PDEs; allows a probabilistic solution of a PDE to be found by means of forward sampling of a diffusion process (E. A. Theodorou, 2011). Applying this method, the direction of the computation can be inverted and Equation (4-7) can now be solved by going forward in time from t_i to t_f.
As seen in the paper of Satoh, Kappen, & Saeki, 2017; E. A. Theodorou, 2011; Horowitz, 2014, the Feynman-Kac formula can be applied to find an explicit solution of the Equation (4-7) as the following expectation:

$$\Psi(x,t) = E^{\rho(x_f,t_f|x_i,t_i)}\Big\{exp\Big(-\frac{1}{\lambda}\big(\phi(x_f)+\int_{t_i}^{t_f} q(x,t)dt\big)\Big)\Big\} \tag{4-8}$$

where $\rho(x_f,t_f|x_i,t_i)$ represents the probability that a sample path going from $x_i \rightarrow x_f$ is generated by the uncontrolled stochastic system dynamics:

$$dx = f(x,t)dt + dw \tag{4-9}$$

conditioned on the start state x_i at time t_i.
To numerically implement this method, a time discretization of the above formula is needed. Partitioning the finite time interval from t_i to t_f into n divisions of equal length ϵ with $t_j = t + (j-1)\epsilon$, $x_0 = x_i$, $x_n = x_f$, the expectation can now be written as a path integral as:

PATH INTEGRAL

$$\Psi(x,t) = \frac{1}{(\sqrt{2\pi\upsilon\epsilon})^n}\int dx_1...dx_n exp\Big(-\frac{1}{\lambda}S(x_{0:N})\Big) \tag{4-10}$$

where

PATH COST

$$S(x_{0:n}) = \phi(x_n) + \sum_{j=0}^{n-1}\epsilon q(x_j,t_j) + \sum_{j=0}^{n-1}\frac{R}{2}\Big(\frac{x_{j+1}-x_j}{\epsilon} - f(x_j,t_j)\Big) \tag{4-11}$$

Equation (4-10) is an integral over discrete paths $x_{0:n} \in (x_0, x_1, ..., x_N)$ all starting at x_0. The paths are weighted by $exp\Big(-\frac{1}{\lambda}S(x_{0:N})\Big)$ where $S(x_{0:n})$ represents the *Action* or cost of the path. A complete derivation of the Equations (4-10) and (4-11) is given in Appendix C-4.
Arriving to this result allows us to avoid the curse-of-dimensionality present in the backward computation of the stochastic HJB.

[a]The Kolmogorov backward equation is a second order, linear PDE addressing the problem of the probability of a system being in a state x at time t and reaching a state x_1 at a future time t_1 ($t < t_1$). This equation has to be solved backwards in time from t_1 to t (E. A. Theodorou, 2011).

Now, the optimal cost-to-go is approximated by the *desirability function* $\Psi(x,t)$ which can be obtained more efficiently by computing expectation values under a forward diffusion process. In the limit $\epsilon \to 0$ a compact form of the optimal cost-to-go function can be obtained as:

Optimal cost-to-go

$$J(x,t) = -\lambda \int [dx]_x exp\Big(-\frac{1}{\lambda} S(x(t_i \to t_f))\Big) \tag{4-12}$$

where $\int [dx]_x$ is taken over all paths starting in x and $S(x(t_i \to t_f))$ is the same expression as in Equation (4-11) in which the sum in the exponent becomes now an integral as $\epsilon \sum_{j=0}^{n-1} \to \int_{t_i}^{t_f}$ (Kappen, 2005).
Consequently, the optimal control is obtained as in Equation (4-4) by combining Equation (4-6) and (4-10):

Optimal Control

$$u^*(x,t) = -\upsilon B^T \partial_x \log \Psi(x,t) \tag{4-13}$$

where the relation $\lambda^{-1} = R\upsilon$ has been used.
This new expression of the optimal control tries to maximize the exponentiated value of $\Psi(x,t)$ while in the former optimal control formulation (4-4), the system was steered towards state spaces that minimizes the optimal cost-to-go $J(x,t)$. This change is a result of the exponential relation between the $J(x,t)$ and $\Psi(x,t)$ functions as $\Psi(x,t) = exp(-\lambda^{-1} J(x,t))$. In this case, a lower value of the $J(x,t)$ will result in a higher value of $\Psi(x,t)$. Therefore, sampled paths with higher costs will contribute less to the optimal control value $u^*(x,t)$ (E. A. Theodorou, 2011).
Computing the value of $\Psi(x,t)$ needed to obtain the optimal control sequence is generally not possible as it implies integrating over uncountably many paths. However, there exist other methods that can obtain an approximated value $\hat{\Psi}(x,t)$. How to obtain this approximation to finally compute the optimal control sequence is treated in the following section.

4-2 Control computation

Approximating the path integral in order to compute the optimal control as described in Equation (4-13) can generally be done using three different methods (Kappen, 2005). In presence of low noise, a Laplace approximation technique can be used. When noise is higher, sampling methods such as Monte Carlo sampling or variational methods can be employed. Although both sampling methods and variational methods (Kleinert, 2009) can in theory be used to approximate the path integral, no application of the latter one for optimal control problems has been found in literature. Thus, in this section, only the Laplace approximation and sampling methods are described.

4-2-1 Laplace Approximation

For small noise situations (e.g. υ is small), the optimal control can be computed using the Laplace Approximation to recover the indirect method formulation based on the Pontryagin Minimum Principle (Section 3-2-4) used for deterministic cases (Kappen, 2005). This approach considers an arbitrary path as the sum of the classical or deterministic path and some fluctuations caused by the small noise. The cost of the path $S(x_{0:N})$ is approximated with a second order expansion around the deterministic path. When this approximation is substituted in Equation (4-10), we obtain a n-dimensional Gaussian integral which can be solved analytically. Thus, the path integral is now replaced by a Gaussian integral centered on the path that minimizes the Action (Eq. 4-11).

This method can achieve good results in situations in which low noise is present (Kappen, 2005). However, it is not accurate enough for cases in which the noise is higher. Although not included in this report, a complete explanation of this method can be found in (Kappen, 2005).

4-2-2 Sampling Methods

A different approach to obtain an approximated value of $\Psi(x,t)$ consists of taking N trajectory samples from the diffusion process described by the uncontrolled system dynamics (Eq. 4-9). The sampling procedure can be performed in different manners (e.g. Monte Carlo sampling, Metropolis-Hasting sampling, etc.). In the following subsections the simplest sampling method (Monte Carlo sampling) will be described. Moreover, as the problem of having a low sampling efficiency is well-known when using naive Monte Carlo sampling, improvement techniques are also presented.

Naive Monte Carlo Sampling

The simplest Monte Carlo sampling consists in running the diffusion process (Eq. 4-9) N times, to obtain N statistically independent random samples which will provide an empirical estimate of $\Psi(x,t)$ as:

$$\hat{\Psi}(x,t) = \frac{1}{N}\sum_{i=1}^{N} exp\Big(-\frac{1}{\lambda}S_{cost}(x_i(t_i \rightarrow t_f))\Big) \tag{4-14}$$

with:

$$S_{cost}(x_i(t_i \rightarrow t_f)) = \phi(x_f) + \int_{t_i}^{t_f} q(x,\tau)d\tau \tag{4-15}$$

where $\tau = (x_i, ..., x_f)$ represents a sampled path. By taking the derivative of $\Psi(x,t)$ with respect to the state x as shown in Equation (4-13) and substituting $\Psi(x,t)$ by its estimate $\hat{\Psi}x, t$ as defined in Equation (4-14), an expression of the approximated optimal control $\hat{u}$ can be obtained as:

$$\hat{u}dt = \frac{1}{\hat{\Psi}(x,t)}\sum_{i=1}^{N}\frac{1}{N}exp\Big(-\frac{1}{\lambda}S_{cost}(x_i(t_i \rightarrow t_f))\Big)dw_i(t) \tag{4-16}$$

where $dw_i(t)$ is the noise present for a given trajectory i. A more intuitive interpretation of Equation (4-16) is that now the optimal control is obtained by averaging the noise directions of all samples at time t, weighted by their path cost over the finite interval $t_i \rightarrow t_f$. An extended derivation of Equation (4-16) can be found in Kappen (2005).

Importance sampling

The naive Monte Carlo sampling procedure can be quite inefficient (Kappen & Ruiz, 2016). It is not hard to imagine that many of the paths sampled from the uncontrolled system dynamics will not result in *good paths*, which are paths with a low cost. As the states are propagated over time according to the system dynamics, the added noise component will generate many undesired, high cost paths. The precision of the estimated $\hat{\Psi}(x,t)$ will depend on the variance of the noise. This is a well-known problem of sampling methods (Landau & Binder, 2014). The simplest step towards an improvement of the sampling procedure is to use importance sampling (Kappen & Ruiz, 2016).

The idea behind importance sampling is simply to choose a different diffusion process from which to sample that can lower the variance of the estimation and thus, increase the efficiency of the sampling (e.g. with equal number of samples N, obtain a higher accuracy of the approximated control). Moreover, a relation has been found between the efficiency of the sampling and the accuracy of optimal control approximation (Kappen & Ruiz, 2016), which shows that better samplers (in terms of efficiency) result in better controllers. Thus, the problem of computing the optimal controls is now transformed into the problem of achieving an efficient sampler.

A straightforward solution described in Kappen (2005) is to first obtain a deterministic control trajectory with the Laplace approximation and afterwards, define a new diffusion process around this deterministic solution. In this way, the new diffusion process is guided in the direction of the deterministic trajectory, eliminating many irrelevant samples (e.g. samples going in the opposite direction of the goal). Although this can be a good-enough option for some tasks, determining which is the best diffusion process from which to sample is still a challenge. It is not clear how to improve the sampling efficiency, thus, many samples are usually needed to obtain a good approximation of the optimal control. This observation could be a problem for real-time, on-board implementations in which limited computational resources are available.

4-3 Path Integral Control Approaches to Path Planning

Path Integral control has been recently applied to a variety of applications such as spacecraft attitude control (Doerr et al., 2018), field space exploration with Autonomous Underwater Vehicles (AUVs) (Kreuzer & Solowjow, 2018), controlling a team of quad-rotors (Gómez et al., 2015) or even aggressive driving with ground vehicles (Williams, Drews, Goldfain, Rehg, & Theodorou, 2016). Despite the initial open-loop formulations of PI control, all these applications use the computed optimal control in a close-loop scheme. One way of obtaining this is by applying a MPC control setting (Williams, Aldrich, & Theodorou, 2017; Gómez et al., 2015; Williams et al., 2016). In this way, the PI algorithm is used to find open-loop optimal control sequences over the planning horizon while only applying the first control input of the sequence. The main difficulty of this approach is the fact that the PI algorithm must constantly run at each time-step over the entire planning horizon. This means that the sampling generation and evaluation has to be performed at the same frequency. Moreover, if the system has complex dynamics, generating the needed amount of samples and still being able to perform real-time control might be challenging. One solution is to take advantage of the recent development in parallel computation by using a Graphical Processing Unit (GPU) (Williams et al., 2017; Williams, Rombokas, & Daniel, 2015). Although this approach is very appealing, a decentralized control implementation must run on-board the MAVs and most quadrotors are not equipped with a GPU. Moreover, tiny MAVs, weighting just 50 grams, have very limited computing and energy resources, making this approach infeasible.

A different option is to use a hierarchical control scheme in which a simplified model of the system dynamics (e.g. point mass) is used for the sampling procedure (high-level control) and a more accurate model is used in the low-level controller (Gómez et al., 2015). This approach is combined with an importance sampling scheme to obtain a more efficient sampling method by using samples from the controlled system dynamics. Starting from an initial control strategy, the PI algorithm is used to obtain an optimal control increment. This way, the initial control is updated and improved at each iteration. As the control improves, the sampling from the controlled system dynamics also becomes more efficient, lowering the probability of picking a high-cost sample (Gómez et al., 2015). Using this importance sampling scheme, we obtain a feedback system in which the accuracy of the controls affects the sampling efficiency.

Path integral control has also been used as a policy improvement technique for control policy search in RL (E. Theodorou, Buchli, & Schaal, 2010a). This off-line PI algorithm is implemented iteratively to find parametrized control policies, which can then be applied as feedback control.

Based on our observations on the earlier mentioned implementations, it is clear that PI control can successfully be used to solve high-dimensional stochastic control problems. However, certain difficulties are still present. It is not obvious how to achieve an efficient sampling method and how path integral control can be used for decentralized, on-board applications. With the objective of implementing the path integral controller on-board a swarm of MAVs with low computational resources, a similar approach as presented in Gómez et al. (2015) will be taken. However, here a decentralized approach will be used. Preliminary experiments are presented in the following section with the aim of benchmarking the computational requirements of this method.

Chapter 5

Summary of the Literature

In the introduction chapter of this preliminary report, motivation for the use of MAVs swarms flying in LF formation was given as a possible solution to large area exploration tasks or dangerous tasks such as forest fire detection. From this starting point, the main objective of this work was set *to demonstrate LF flight of a team of MAVs to study the use of a decentralized, stochastic, path integral controller* as a possible solution to multi-agent path planning and control. To gain insights on how this control approach compares with the current state-of-the-art, an extensive literature study was performed on the existing optimal control methods for multi-agent formations.

LQG	Dynamic Programming	Indirect methods	Direct methods	PI control
✓ high-dimensionality	! high-dimensionality	✓ high-dimensionality	✓ high-dimensionality	✓ high-dimensionality
✗ non-linearity	✓ non-linearity	✓ non-linearity	✓ non-linearity	✓ non-linearity
✓ optimality	✓ optimality	✓ optimality	✓ optimality	✓ optimality
✓ computationally efficient	! computationally efficient	✓ computationally efficient	! computationally efficient	! computationally efficient
✓ multi-robot system	✓ multi-robot system	✗ multi-robot system	✓ multi-robot system	✓ multi-robot system
✓ stochasticity	✓ stochasticity	✗ stochasticity	✗ stochasticity	✓ stochasticity

Figure 5-1: Comparison table of the main optimal control approaches seen in the literature study. The green tick mark means that the criterion is taken into account by that method. The orange exclamation sign represents an existing challenge for that method, while the red cross means either that no work has been found in literature of such an implementation, or that it is not possible to incorporate such criterion within that method.

To compare the different optimal control methods, multiple criteria can be derived from the formulation of the objective alone. As mentioned, the implementation is meant for a MAV platform, which is a *high-dimensional*, *non-linear system*. Due to the weight limitations, MAVs are known to have low energy resources which, in turn, limit the computational platforms that can be carried on-board. Thus, the path planning and control algorithm has to be able to plan optimal paths which optimize the time, distance or energy consumption of a certain task (*optimality*). Moreover, as the chosen implementation scheme is decentralized, the control algorithm has to run on-board the MAV using only the computational resources available, which are generally low (*computationally efficient*). Furthermore, an entire team of MAVs (*multi-robot system*) has to be controlled to fly in formation in a real environment (*stochastic*). This translates in the need of a scalable algorithm which can cope with the inherent of the MAV.

With the goal of summarizing the main findings of the performed literature review on optimal control methods, Figure 5-1 is presented. Using the previously mentioned criteria, the different optimal control methods are evaluated. While acknowledging that the picture shown in this figure does not reflect the entire landscape of existing optimal control approaches, it provides an overview of the most used ones.

As observed, Dynamic Programming methods from which the PI control approach branches off, has similar challenges regarding the required computational complexity. However, the PI control manages to overcome the curse of dimensionality. It must be noted that Dynamic Programming comprises a vast number of techniques, some of which also try to avoid the high-dimensionality issue as explained in 3-2-2. However, most applications found in literature use an MDP framework to model the stochastic optimal control problem and employ value or policy iteration algorithms to solve it, which, cannot cope with high-dimensional systems. The remaining methods presented in Figure 5-1 are either not suitable for non-linear systems or encounter difficulties when applied to stochastic, multi-robot systems.

The main challenge that still needs to be faced by the PI control method is the computational efficiency. Although its use has been demonstrated in formation control experiments with real MAV platforms (Gómez et al., 2015), no on-board implementation has been attempted yet. Towards this goal, further preliminary experiments are presented in the following part of this report.

Part III

Preliminary experiments

Chapter 6

PI controller simulations for a LF task

For a better understanding of the PI control algorithm and based on the observations from the literature study included in Part I, several scenarios of a LF formation flight controlled with the PI algorithm have been simulated. In these scenarios, the LF formation consisted of one leader and two followers. The goal of the formation is to fly from an initial position towards a set of predefined target waypoints. The location of the target waypoints is only known by the leader unit. The followers are not aware of the collective goal. They only use their relative distances from the leader to navigate from one point to another. Figure 6-1 illustrates the described LF scenario in which two target waypoints are given.

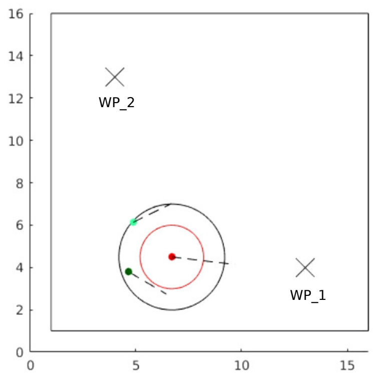


Figure 6-1: Leader-follower simulation. The red dot represents the leader, while the two green dots represent the followers. The concentric circles around the leader mark the maximum allowed distance (black) and the minimum allowed distance (red). The planned trajectory over the planning horizon is shown with dashed lines.

For these simulations, both leader and followers were modeled as simple 2D point-mass systems following double integrator dynamics. Each unit's state vector $\mathbf{x}$ is composed of its East-North (EN) positions and EN velocities, as well as the communicated relative positions and velocities of the remaining neighboring units. For example, the leader's state vector is given by $\mathbf{x} = [p_L, v_L, x_{rel}]$, where x_{rel} contains the relative positions and velocities of the followers as $x_{rel} = [p_1, v_1, p_2, v_2]$ and $p, v \in \mathbb{R}^2$. Similarly, each unit input vector $\mathbf{u}$ consists of its EN accelerations where $\mathbf{u} \in \mathbb{R}^2$.

To obtain the local control inputs of each member of the formation a separate PI controller is used. The cost function to be optimized by each PI controller is different, depending on the role of the unit. For the leader unit, the state dependent cost function is given by:

$$q_L(x,t) = C_t||p_L - wp_t||_2 + C_h(v_L \times wp_t) + \sum_{i=1}^{K-1} exp(C_{col}({d_{min}}^2 - ||p_L - p_i||_2)) \tag{6-1}$$

where C_t is distance penalty for not reaching the target waypoint wp_t and C_h is the penalty for having a heading deviation. A safety distance d_{min}, is defined and a cost penalty C_{col} is applied for all followers $K-1$, whenever this distance is crossed.

The cost function of the followers is constructed in a similar way but without including any target related penalties. Moreover, a cohesion penalty C_{coh}, meant to keep the followers in formation with the leader is included. A penalty for flying parallel paths C_p is also added. The follower's cost function is then defined as:

$$\begin{aligned} q_F(x,t) = &\sum_{i=1}^{K-1} \Big(exp(C_p \times (CP_{thr} - CP_i)) + exp(C_{col}({d_{min}}^2 - ||p - p_i||_2)) \Big) \\ &+ exp(C_{coh} \times (r^2 - ||p - p_L||_2)) \end{aligned} \tag{6-2}$$

where CP_m is the absolute value of the cross product of velocity vector between the follower and another unit i and C_p is the cost penalty applied whenever this value crosses the threshold given by CP_{thr}. A cohesion penalty $C_{cohesion}$ is applied when the relative distance between the leader and one of the followers is bigger than the maximum allowed, r.

In-line with the observations of Part I, a MPC scheme is implemented to obtain closed-loop controls and an importance sampling scheme is used to increase the efficiency of the sampling procedure. Moreover, actuator noise is taken into account and modeled as Wiener noise with variance ν.

6-1 Performance analysis

To evaluate the PI controller performance, the cost functions of the leader and the followers are analysed. For this, a LF task has been simulated using N = 2000 samples and a finite planning horizon of H = 2 seconds. We show the resulting cost function of each unit and their corresponding flown trajectories in Figure 6-2. As observed, the leader's cost function is increasing after each target waypoint selection and clearly decreasing as the leader is approaching the target. The followers state cost functions have overall a smaller value which only increases at particular moments when the imposed separation distances are violated. This can be seen in Figure 6-2, where the PI controller applies a high penalty to Follower 1 when this gets outside the cohesion radius of the leader. A closer look to the separation error

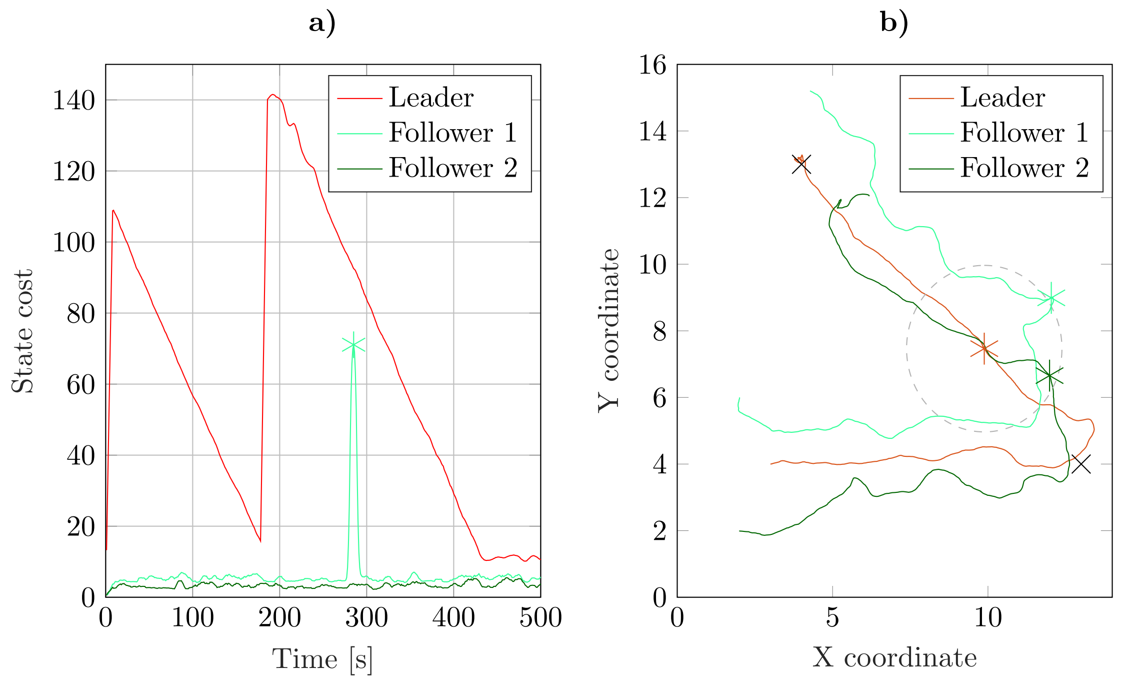


Figure 6-2: In this simulation, $C_p = 10$, $C_{coh} = 25$, $C_{col} = 15$, $C_t = 50$, $C_h = 2$. **a)** State cost function for the leader and the two followers. **b)** XY trajectory of the units. The two target waypoints are marked with a cross.

is given in Figure 6-3, where a comparison is made with a random walk scenario (Figure 6-3a) and the relative distance between units is measured over the entire simulation time (Figure 6-3b). As observed, the PI controller can satisfy the imposed separation constraints for most of the simulation time. When compared to the random walk implementation, we see that the number of times in which the separation error has been violated is significantly higher. This is caused not as much by the collision distance violation, but because the cohesion of the group is clearly not fulfilled in the simple random walk scenario, causing the large separation errors between the leader and the followers as seen in Figure 6-3a.

6-2 Runtime analysis

Before implementing the PI algorithm on-board a MAV with limited computational resources, the runtime of the algorithm must be measured off-line. This measure will determine the maximum frequency at which the optimal controls can be generated. The main parameters affecting the algorithm's runtime are the number of samples used and the length of the planning horizon. A higher number of samples will increase the number of operations needed to compute the optimal controls, as more possible trajectories would need to be simulated and evaluated. Similarly, a longer planning horizon implies having to propagate each sample for a longer period of time, increasing again the number of operations.

To investigate the relation between the number of samples and the horizon length with the runtime of the algorithm, we have varied these parameters and measured the runtime as

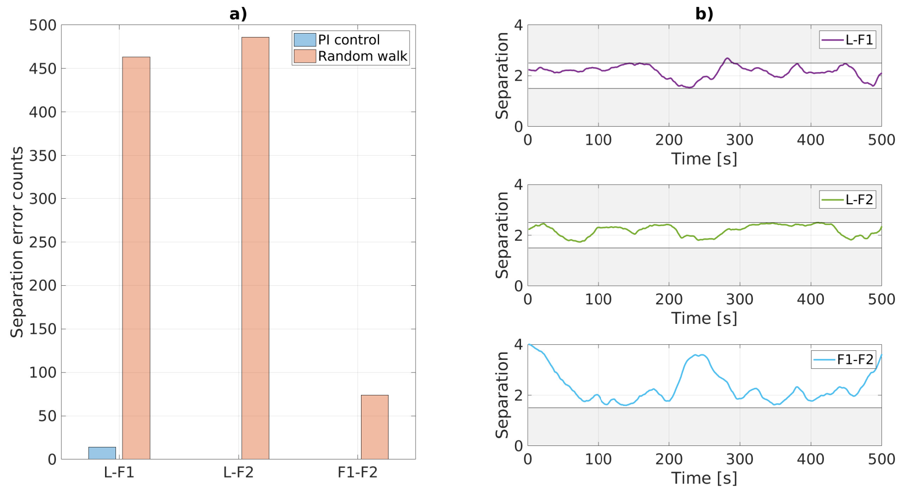


Figure 6-3: Separation error results. a) Number of times the separation constraints have been violated using the PI controller and a simple random walk. **b)** Relative distance between the different MAVs using the PI controller. Areas beyond the maximum $r = 2.5$ and minimum $d_{min} = 1.5$ separation distance are marked in gray.

shown in Figure 6-4. All results presented here are obtained using a 2.3GHz Core-i5 Quad Core desktop PC. For each scenario, the mean cost of the formation over the entire simulation time is also calculated.

As expected, the running time increases linearly with both the number of samples (correlation coefficient = 0.9997) and the length of the planning horizon (correlation coefficient = 0.9999). On the other hand, the performance of the task represented here by the mean cost, does not follow the same relation. The mean cost is approximately constant when the number of samples are increased. The fact that the simulated LF task does not have a very high complexity can explain this, allowing the PI controller to optimize the cost function while using a low number of samples as seen in Figure 6-4a.

For the horizon time, we can see that the PI controller needs a certain planning period to obtain a better task performance. With horizon lengths lower than 0.5 seconds, the mean cost is high, which indicates a bad performance of the LF task. In these cases, the leader is not flying towards the target waypoint. As all planned trajectories are short because they are propagated over a small planning horizon, none takes the leader significantly closer to the target waypoint. Thus, all are assigned a similar cost. This situation changes as the horizon length is increased, observing an almost constant performance from 0.9 seconds onwards.

These results indicate that an implementation of the PI controller on-board a real MAV platform could achieve real-time performance if a low number of samples and a relatively low horizon length is selected. However, it must be taken into account that these simulations were performed assuming a simple MAV model. When using the real platform, more samples could be needed to achieve a similar performance as the controller would have to compensate for the inherent uncertainties of the MAV.

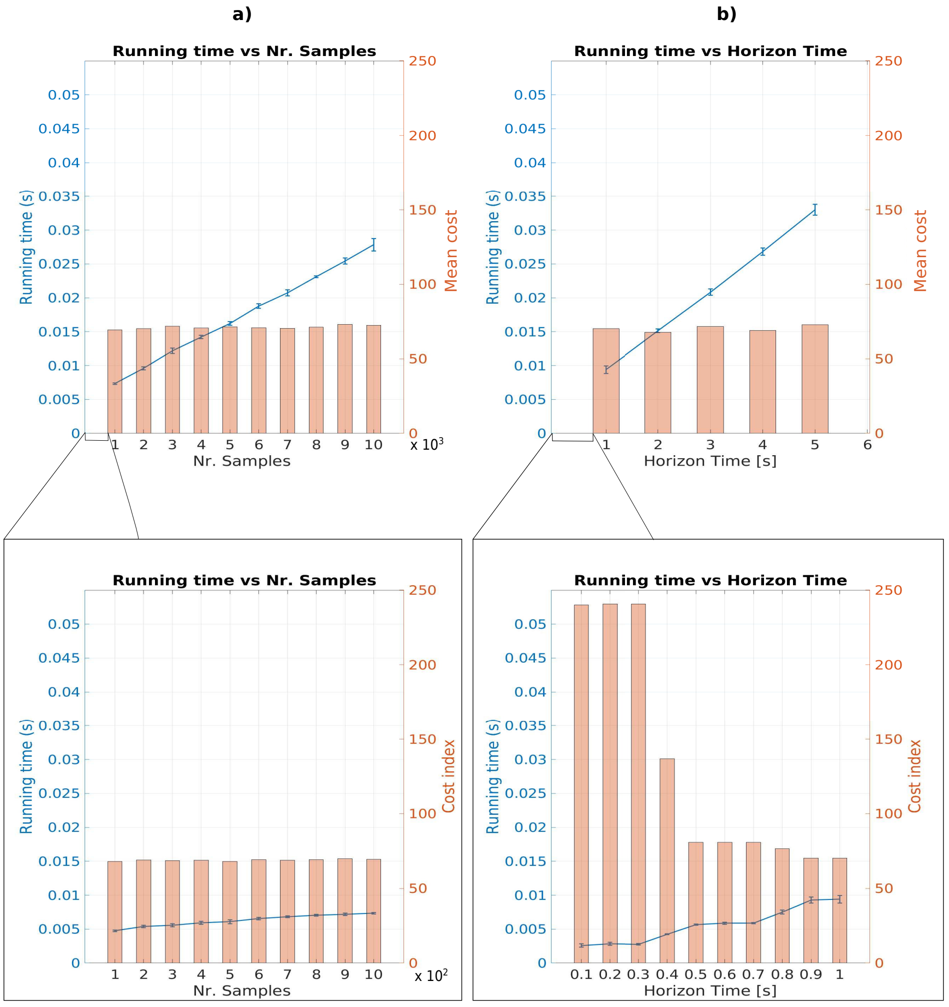


Figure 6-4: Runtime analysis results. a) Running time and mean cost with respect to the nr. of samples used. In these simulations a constant H = 1 second was used. Error bars correspond to 5 different random noise realizations. **b)** Running time and mean cost with respect to horizon length. Here a constant N = 2000 samples was used.

6-3 Sampling efficiency

One of the most important metrics for any sampling based method is the sampling efficiency. For real-time performance, an efficient sampling procedure is very important, as the goal is to achieve the best approximation of the optimal controls using the lowest number of samples possible. For the LF task, sampling is used within the planning loop to generate possible trajectories and evaluate them, to determine which is the optimal direction that must be followed by each unit of the formation. Here, the objective is to have all samples converging towards the same direction, preferably the optimal one. To measure the convergence of the sampling procedure, the Effective Sample Size (ESS) metric is used as defined in Gómez et al. (2015). This is computed as $ESS = 1/\sum_{k=1}^{N} w_k^2$, where N is the number of samples used and w_k is the corresponding sample weight. This weight is computed as shown in Appendix D and a softmax function is applied to normalize its value between 0 and 1, 0 being assigned to the highest cost sample and 1 to the lowest cost sample. The value of ESS then ranges between 1 and N. Values closer to 1 indicate a poor convergence, implying that the optimal direction and thus, the optimal controls are dominated by one single sample. This situation occurs when all samples have a very high cost. Then all sample weights are zero except for the lowest cost sample which is has a weight of 1. On the other hand, values closer to N indicate that all N samples contributed to determine the optimal controls (e.g. all samples have the same cost).

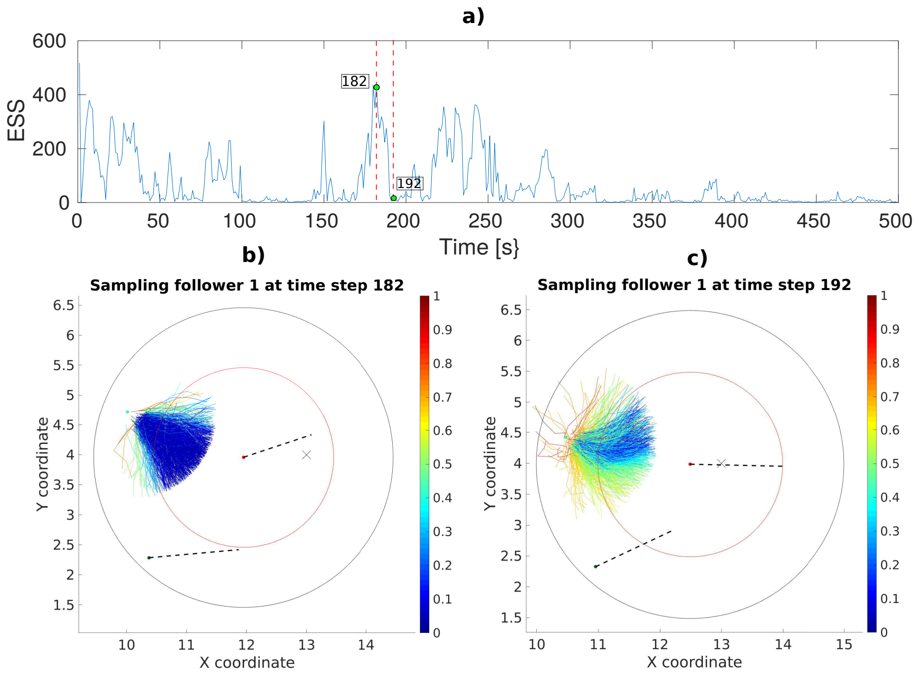


Figure 6-5: In this simulation the following parameters were used: N=1000 samples, H=1 s. **a)** Effective Sample Size for the Follower 1. **b)** Follower 1 sample trajectories for a high ESS value. The color map indicates each sample's normalized cost. The leader and Follower 2 positions are also shown, as well as their planned trajectory (dashed lines). **c)** Follower 1 sample trajectories for a low ESS value.

Figure 6-5 shows the change in ESS value for the entire simulation time, as well as the sample trajectories of a follower unit (Follower 1), for different values of ESS. When the convergence of the sampling is high (ESS high), we can see that many sample trajectories have low cost. Therefore, all these low cost samples will have a high weight and will contribute to the computation of the optimal controls. In situations in which only a few samples have a low cost (Figure 6-5c), the ESS takes a lower value as less samples are used to compute the optimal controls.

This change in ESS value is partly due to the relative location and velocity of the neighboring units and how their respective planned trajectories are obtained. In these simulations, the planned trajectories of the neighboring units are obtained by simply linear integrating their initial positions over the planning horizon. Thus, each unit assumes that the other units will move with a constant velocity straight ahead from their communicated positions. This simple assumption of the future trajectory of the neighboring units affects the weighting of the samples. As seen in Figure 6-5c, the planned trajectory of the second follower is pointing slightly upwards than in a previous time step (t=182). As a result, the cost of all samples that would take the units too close to each other is now higher, leaving a thin area of low cost samples. These situations are continuously encountered along the entire simulation, making the ESS value to change as seen in the top graph. If the planned trajectory of neighboring units could be better approximated, the weighting procedure would also be more accurate, possibly increasing the ESS convergence. One way of improving the accuracy of the planned trajectories is by also communicating the control inputs among the members of the formation (Kappen, 2007). This, would then allow the computation of more accurate planned trajectories of neighboring units at the expense of requiring higher communication bandwidth to send the additional information. Figure 6-6a shows how such a scenario affects the sample trajectories of Follower 1. We can observe how a more accurate assumption of the other units trajectories increases the sampling convergence. Now, Follower 1 can foresee the turn in the trajectory of the other follower and can assign a low cost value to a higher number of samples.

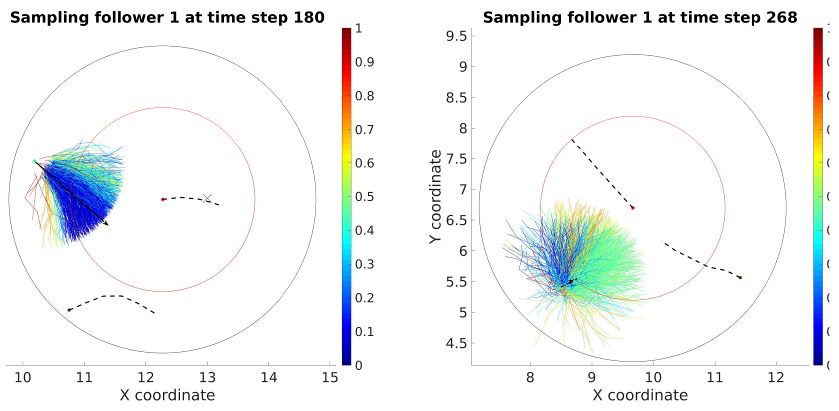


Figure 6-6: a) Sample trajectories for Follower 1 when the planned controls of the neighboring units are also communicated. The black arrow indicates the previous commanded velocity vector. Given the high control inputs applied, the importance sampling scheme concentrates all the trajectories around the previously taken direction. **b)** Sample trajectories for Follower 1. Here, the commanded velocity vector has a smaller magnitude decreasing the effect of the importance sampling. Now, sample trajectories are more scattered.

Besides the accuracy of the neighboring units planned trajectories, the ESS is also affected by the importance sampling scheme. To decrease the number of irrelevant samples, in these simulations possible trajectories are sampled from the controlled system dynamics. This method is thus, highly dependent on the value of the optimal control inputs applied to each unit. When the control inputs are small, the system is less controlled making the importance scheme less efficient. When the control inputs are higher, the importance sampling has a stronger effect and samples are only taken around the optimal direction. This difference can be seen in Figures 6-6a and 6-6b where the commanded velocity vector at the previous time step is marked with a black arrow. When the applied control inputs are higher, the sample trajectories are concentrated around the previous optimal direction. With a less controlled system (Figures 6-6b), the sample trajectories are more scattered and the number of irrelevant directions increased. As a result, the ESS value decreases.

6-4 Communication sensor

As explained in Chapter 1, the deployment of multi-robot systems introduces additional challenges not present in single-robot systems. To safely enable LF formation flight with the MAVs, inter-robot collision avoidance must be performed by the on-board PI controller. This requires precise and frequent knowledge of the relative positions between the MAVs. This information could be obtained using external localization systems such as the Global Position System (GPS) in outdoor environments or a Motion Capture System (MCS) in indoor environments. However, the low accuracy of the GPS and its uneven signal reception make it unreliable for determining the short inter-robot distances within the swarm. Moreover, a realistic indoor environment will not be equipped with a MCS. Thus, an on-board solution is generally preferred.

Among the proposed on-board solutions, (e.g. radio based, camera-based, infra-red based (Navarro & Matía, 2013), in this work, an approach as taken in Helm et al. (2018) is going to be followed. In their work, the relative localization between the units of a LF formation was enabled with an on-board Ultra Wide Band (UWB) range sensor. To achieve the inter-robot localization, they observed that the LF formation was constrained to fly non-parallel trajectories. This was implemented in their work through the design of a flight path which avoided parallel configurations. Building upon their work, here, this constraint will be added to the control algorithm, making the controller responsible now, for avoiding parallel flight trajectories. To test its effect on the LF task performance, a scenario in which the LF formation performs a straight, long flight towards a single target waypoint is simulated.

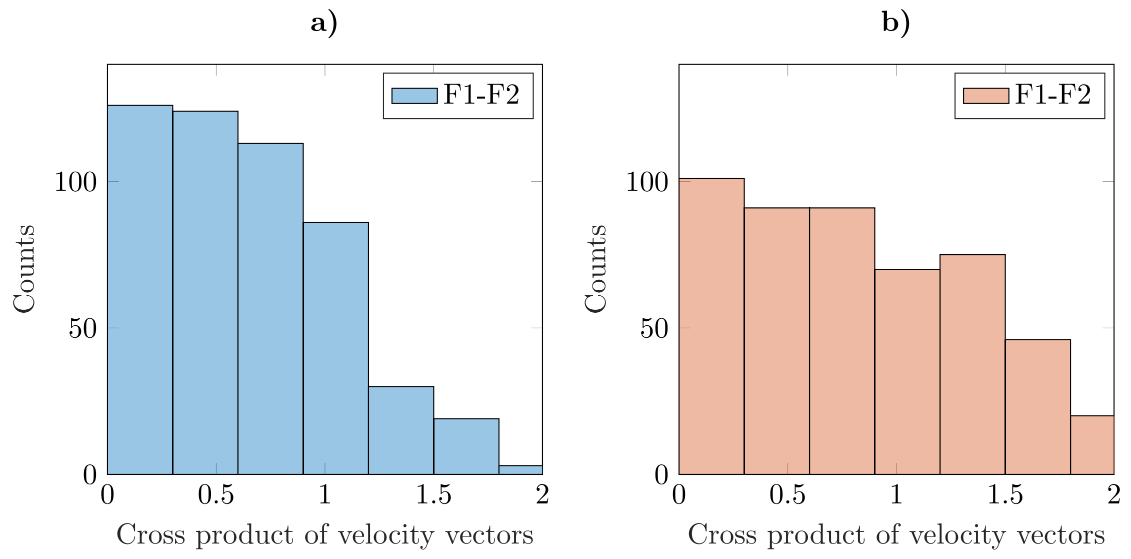


Figure 6-7: Velocity vector cross product with $CP_{thr} = 0.3$ **a)** Without applying the non-parallel flight constraint **b)** Applying the non-parallel flight constraint

Using a $CP_{thr} = 0.3$, a comparison is made in Figure 6-7 between applying the non-parallel constraint or not. When the constraint is applied (Figure 6-7b), the PI controller tries to minimize the situations in which the absolute value of the cross product of velocity vectors is higher than the imposed threshold while still fulfilling the other restrictions (e.g. separation distances). As observed, these situations are not completely avoided. However, perfect parallel

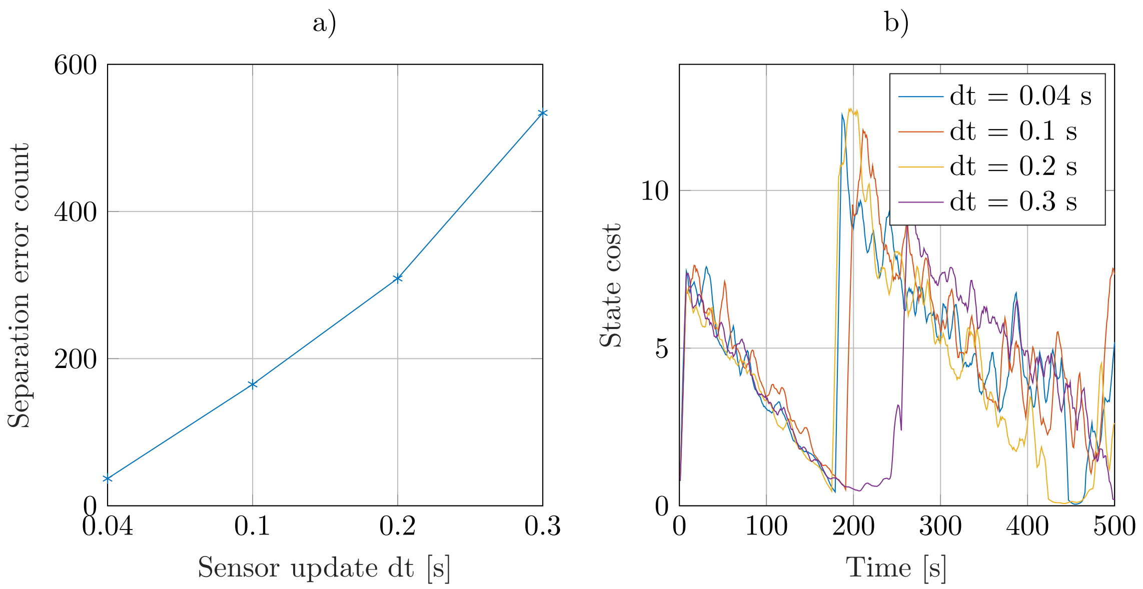


Figure 6-8: a) Separation error count vs. sensor update time step dt **b)** Leader state cost function vs. sensor update time step dt

flight is only achieved when the cross product is zero. Thus, here the 0.3 threshold is taken as a safety measure. Moreover, and what is perhaps more important is that values lower than CP_{thr} are never maintained for periods longer than 0.5 seconds. Next to that, it was also noted that even if the non-parallel flight constraint is not applied, the formation does not follow prolonged parallel paths. This is a result of not having the followers constrained to fly in a rigid pose with respect to the leader. As only the distance with respect to the leader is constrained, a higher flexibility of the formation structure is allowed.

Another interesting observation from the work of Helm et al. (2018), is the effect of the sensor update frequency on the localization error. It was noticed that the UWB sensor used to obtain the relative position of the MAVs experimented sudden drops in the update frequency, which caused the localization error to increase. To test the effect of these frequency drops on the LF task performance, we simulated scenarios in which with a probability of 10%, an update frequency drop was experimented. Figure 6-8b shows the relation between the sensor update time step dt and the number of times the separation limits have been violated. As expected, the separation limits are more often crossed when the sensor update frequency is lower (higher time step dt). To better understand how the cost function is affected in these situations, Figure 6-8a shows the leader's state cost function as a function of the sensor update frequency. It can be noticed that a controller delay is experimented for higher sensor time steps, but this is still able to minimize the cost function at the expense of a lower performance of the LF formation.

Chapter 7

Parrot Bebop I step response

In this chapter, we analyze the step response of the Bebop I drone to motivate the assumption used in the scientific paper presented in Part I where a first order delay was used to model the real velocity controller. To this end, experiments in which the Bebop I drone was commanded to fly with a forward and backward direction of 0.5 m/s are presented.

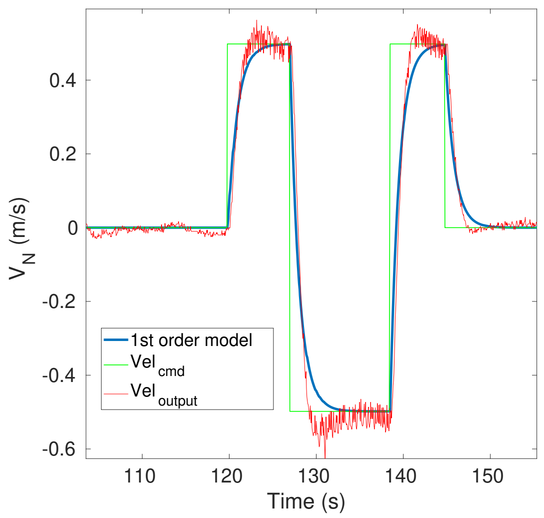


Figure 7-1: Step response of the Bebop I drone to several forward and backward velocity commands of 0.5 m/s. The first order approximation used to model this behavior is also shown.

Fig. 7-1 shows the commanded velocity input, the step response of the Bebop I drone and the first order approximation used. Taking the transfer function of the first order delay as:

$$H(s) = \frac{1}{\tau s + 1} \tag{7-1}$$

a value of $\tau = 1$ was used to approximate the step-response of the real platform. This value was chosen empirically. Observing Fig. 7-1, we see that the model used does not match the drone's step response perfectly. However, as shown in the real-experiments presented in the scientific paper, this simple approximation is sufficient to control the MAV formation.

Chapter 8

Discussion of preliminary results

Based on the preliminary experiments results presented in Chapter III, it is possible to conclude that a decentralized PI controller can be employed to guide and control a LF formation flight under the conditions described in the previous chapter. With the defined cost functions for the leader and the followers, the formation could successfully fly towards two predefined target waypoints. A safety distance was maintained among the members of the formation although not for the entire simulation time. In several occasions, the allowed separation among units was briefly crossed. This could be easily overcome by applying a higher cost penalty to collision events. However, a very constrained task could deteriorate the performance of the PI controller as more samples would now be needed to compute the optimal controls.

Regarding the computational time needed for the PI controller to obtain the optimal controls, multiple LF flight tasks were simulated and the runtime of the algorithm was measured. The simulated scenarios differed in the number of samples or the planning horizon length which was considered. The results showed a linear increment of the runtime when these two variables were increased. However, the computational time was small, reaching a maximum value of only 0.035 seconds for the scenario in which 2000 samples and a horizon length of 5 seconds was used. The small runtime observed, makes us believe that real-time performance could indeed be achieved on-board a real MAV platform with limited computational resources.

Besides the runtime of the algorithm, the performance of the task was also measured. Due to the relatively simple task, the performance remained constant with the number of samples, since the MAVs where able to fly towards the two target waypoints using only 100 samples. For small planning horizons ($< 0.5s$), the leader was not able to find the optimal direction towards the target waypoints leading to a strong decrease in performance. Although higher cost penalties could be assigned to force the leader to steer towards the target and therefore improve the performance using even shorter planning horizons, the computation time that could be saved does not compensate the risks that come with it. Using very short planning horizon would mean to have a similar behavior to a reactive controller, which only reacts to nearby stimuli. Therefore, given the low increase in computational time, a planning horizon of at least one second is recommended.

Despite the fact that the LF task could be performed with a relatively small amount of samples, more complex tasks involving a higher number of units or a more restrictive formation structure will require a higher number of samples. In these cases, it becomes important to have an efficient sampling procedure, in which all samples converge towards the optimal direction. From the experiments performed, we noticed that the sampling convergence (ESS) varies during the simulation time, taking higher values when more samples with low cost, *good samples*, are found and lower values when less of these samples are encountered. This, is strongly related with the relative position of the neighboring units and how their planned trajectories are computed. A lower number of *good samples* generally reflects a very constrained situation in which many cost penalties are being applied. Although this could indeed be the case for a complex task, it was observed that the inaccuracy of the planned trajectories of the neighboring units plays a big role in the resulting sampling convergence. It was showed how a more accurate computation of the planned trajectories could improve the sampling convergence by communicating also the control inputs among the units. However, the limited communication bandwidth of the on-board UWB sensor has to be taken into account and therefore a trade-off must be established.

We have also seen how the importance sampling scheme affects the sampling convergence. Using the controlled system dynamics to obtain the sample trajectories is beneficial for the LF scenario described here. We saw that the sampling area is concentrated around the previous optimal direction when the applied optimal controls take a big enough value. This eliminates many irrelevant samples and smooths the trajectory. Nonetheless, it could be argued that this strategy is not always the ideal one. Its effectiveness varies according to the optimal control values. Moreover, for an exploration task with unknown obstacles, using this strategy combined with a short planning horizon could result in dangerous situations. The MAVs could get too close to an obstacle and due to the importance sampling applied, would not be able to steer away, as all samples would be taken around the previous control input (e.g. straight away). This situation could be avoided if the radius of obstacle detection is big enough or with a longer planning horizon.

Finally, some aspects of the on-board communication sensor were tested and the effect of the non-parallel flight constraint on the PI controller performance was analyzed. The parallel paths were avoided by the PI controller, reducing the number parallel vectors encounters by 20% while still maintaining the imposed separation constraints. Moreover, parallel paths were never prolonged for more than 0.5 seconds. Despite the promising results, the potential sensor errors combined with the intrinsic actuator delay of a real MAV platform presented in Chapter 7, could lead to a lower performance of the PI control algorithm. As the main objective of this thesis is to prove the real-time capabilities of a decentralized PI control algorithm, an external MCS is going to be used to obtain the relative localization among the MAVs.

Appendix A

Stochastic Optimal Control

A-1 The Wiener process

Many stochastic algorithms use Wiener processes to mathematically model the random effects present in stochastic control problems (Gorodetsky et al., 2015; Huynh, Karaman, & Frazzoli, 2016; Fu & Topcu, 2015). A Wiener process is a continuous-time stochastic process also called Brownian motion. To better understand what a Wiener process is and how it is related to the physical effect known as Brownian motion, a brief description of random walks is firstly needed.

Random walks

A random walk is a discrete stochastic process that predicts the value of a system at time t to be equal to the value of the last period plus a stochastic white noise component. One simple example of random walk is a particle located at the origin of a discrete one dimensional axis. At time $n = 0$ the particle can move either forward +1 or backward -1 with probabilities p and $q = 1 - p$ respectively (Bhattacharya & Waymire, 2009). In this case the discrete stochastic process would be the sequence of independent displacements of the particle over the one-dimensional axis (Figure A-1).

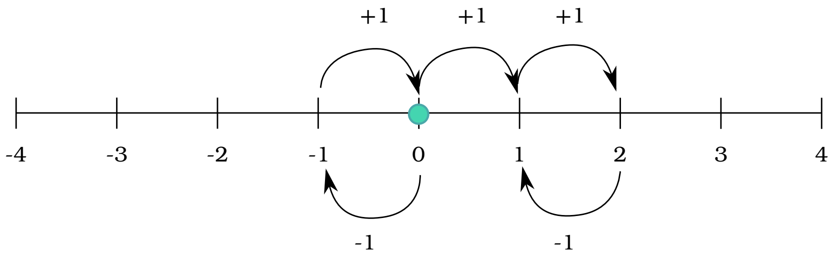


Figure A-1: Example of random walk as the random movement of a particle on a 1D axis

Brownian motion

The continuous analog of this type of stochastic process is the Brownian motion. Brownian motion is a continuous time stochastic process that describes the random motion of a solute particle immersed in a fluid, liquid or gas as a consequence of it's collision with the molecules of the fluid, liquid or gas (Bhattacharya & Waymire, 2009). This continuous stochastic motion is a limiting form of the random walk. To better understand this relation, a similar example as given in (Bhattacharya & Waymire, 2009) is going to be followed. Let's take for example one particle immersed in a fluid that undergoes an average of f collisions per second with the molecules of the fluid. To simplify the problem, it is assumed that the particle can only move on a one dimensional axis like in the case of the random walk. Thus, with every collision, the particle can only move $+\delta$ or $-\delta$ with probabilities p and $q = 1 - p$ respectively. In this simplified scenario, the particle in the fluid performs an one-dimensional random walk with step size δ. Let's continue to assume that the fluid is contained in a very large recipient making the random walk unrestricted by any boundary. Then, at time $t > 0$ the particle will have suffered $n = tf$ independent displacements. The position of the particle at a certain time $t, (t > 0)$ is calculated as the sum of all the independent random displacement. This, by the central limit theorem is approximately Gaussian (Bhattacharya & Waymire, 2009). As $f \to \infty$ the mean of the Gaussian converges to $t\mu$ and the variance converges to $t\sigma^2$. Thus, the displacement of the particle at time $t > 0$ is Gaussian with mean $t\mu$ and variance $t\sigma^2$. It can be noticed that the Gaussian has a direct relationship with the time. This reflects the fact that further in time, it is harder to estimate the position of the particle. Both the drift μ and the diffusion σ^2 increase with time. When the drift is zero and the diffusion coefficient is one the motion is known as the standard Brownian motion.

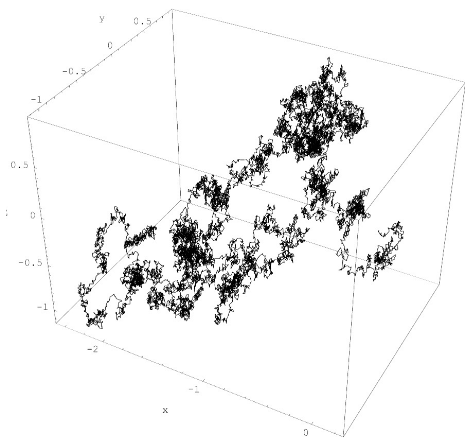


Figure A-2: Three-dimensional brownian motion simulation for a time interval of 2 seconds (Wikipedia contributors, 2018)

Wiener process

The relation of Brownian motion with the Wiener process $W(t)$ comes from the mathematical result of Norbert Wiener, which first formulated that the path of a particle immersed in a fluid can be taken as continuous. The Wiener process is thus, a mathematical representation of the physical phenomena known as Brownian motion. Likewise it is a continuous-time stochastic process with the following properties:

1. *Independence.* Increments of $W(t)$ are independent. As explained in the Brownian motion example, the particle movement can be described as the sum of independent displacements. Therefore, taking $W(t) - W(s)$ is independent of $W(\tau)$ where $\tau < s$ for any $0 \leq s \leq t$.

2. *Normal increments.* $W(t) - W(s) \approx N(0, t-s)$ considering a standard Brownian motion.

3. *Continuity.* $W(t)$ is a continuous function

A-2 $\hat{I}to - Taylor$ Expansion

The general stochastic differential equation describing the stochastic system dynamics is :

$$dx = f(x(t), u(t), t)dt + g(x(t), u(t), t)dw \tag{A-1}$$

To obtain the Hamilton-Jacobi-Bellman equation for the stochastic case, the optimal cost-to-go function $J(x,t)$ at time $t1 = t + dt$ is approximated via a Taylor expansion:

$$\begin{aligned}\Big\langle J(x(t+dt), t+dt)\Big\rangle &= J(x,t) + J_t(x,t)dt + J_x(x,t)dx + \frac{1}{2}J_{xx}(x,t)dx^2 \\ &+ \frac{1}{2}J_{tt}(x,t)dt^2 + \frac{1}{2}J_{xt}(x,t)dxdt + \text{higher order terms}\end{aligned} \tag{A-2}$$

where J_x, J_t, J_{xx}, J_{tt} and J_{xt} represent partial derivatives of the optimal cost-to-go function $J(x,t)$.

From equation (A-1) the following expressions can be derived and written in compact form:

$$(dx)^2 = f^2dt^2 + g^2dw^2 + 2fgdtdw \qquad dxdt = fdt^2 + gdwdt \tag{A-3}$$

Substituting the expressions in (A-3) into (A-2) we obtain:

$$\begin{aligned}\Big\langle J(x(t+dt), t+dt)\Big\rangle &= J(x,t) + J_tdt + J_x(fdt + gdw) + \frac{1}{2}J_{xx}\big(f^2dt^2 + g^2dw^2 + 2fgdtdw\big) \\ &+ \frac{1}{2}J_{tt}dt^2 + \frac{1}{2}J_{xt}\big(fdt^2 + gdwdt\big) + \text{higher order terms}\end{aligned} \tag{A-4}$$

Now, applying the standard $\hat{I}to$ differential rules (Handel, May 2007):

$$dt^2 = 0 \qquad dw^2 = dt \qquad dwdt = 0 \tag{A-5}$$

the Taylor expansion can be further simplified as:

$$\Big\langle J(x(t+dt), t+dt)\Big\rangle = J(x,t) + J_tdt + J_xfdt + \frac{1}{2}J_{xx}\big(g^2dt\big) \tag{A-6}$$

Appendix B

Necessary Conditions for Optimality

The necessary conditions for optimality are derived using the theory of Calculus of Variations. This branch of mathematics uses variations (small changes in function) to find maxima and minima of functions of continuous variables. Thus to derive these conditions, an expression of the cost-function when small control perturbations have been applied is obtained. A similar derivation as used in (Stengel, 1994) is presented in this section.

As before, given the deterministic continuous-time system dynamics:

$$\dot{x}(t) = f(x(t), u(t), t) \tag{B-1}$$

And the cost-function that the optimal solution must minimize:

$$C(x_i, t_i, u(t_i \to t_f)) = \phi(x(t_f)) + \int_{t_i}^{t_f} L(x(t), u(t), t)dt \tag{B-2}$$

To take into account the system dynamics in the minimization, an augmented cost-function is defined as:

$$C_A(x_i, t_i, u(t_i \to t_f)) = \phi(x(t_f)) + \int_{t_i}^{t_f} L(x(t), u(t), t) + \lambda(t)\Big(f(x(t), u(t), t) - \dot{x}(t)\Big)dt \tag{B-3}$$

where Lagrange multipliers contained in the constant adjoint vector λ are used to augment the cost function with the equality constraint:

$$f(x(t), u(t), t) - \dot{x}(t) = 0 \tag{B-4}$$

Because equation B-4 must be satisfied over the entire time interval, it is adjoined to the integrand term of the cost function. This expression can be written in terms of the Hamiltonian $\mathcal{H}$:

$$\mathcal{H}(x(t), u(t), \lambda(t), t) = L(x(t), u(t), t) + \lambda^T(t)(f(x(t), u(t), t) \tag{B-5}$$

Substituting equation B-5 in B-3 we obtain:

$$C_A(x_i, t_i, u(t_i \to t_f)) = \phi(x(t_f)) + \int_{t_i}^{t_f} \mathcal{H}(x(t), u(t), \lambda(t), t) - \lambda^T(t)\dot{x}(t)dt \tag{B-6}$$

Separating the integrals and using integration by parts, the following augmented cost-function is obtained:

$$\begin{aligned} C_A(x_i, t_i, u(t_i \to t_f)) = \phi(x(t_f)) &+ [\lambda^T(t_i)x(t_i) - \lambda^T(t_f)x(t_f)] \\ &+ \int_{t_i}^{t_f} \mathcal{H}(x(t), u(t), \lambda(t), t) + \lambda^T(t)\dot{x}(t)dt \end{aligned} \tag{B-7}$$

As previously defined, a stationary solution will be the one with zero effect of control variations on the cost function. Applying calculus of variations theory, the first order variations of the cost function's components can be defined as:

$$\Delta(\cdot) = \frac{\partial(\cdot)}{\partial u}\Delta u + \frac{\partial(\cdot)}{\partial x}\Delta x(\Delta u) \tag{B-8}$$

where $\Delta x(\Delta u)$ represents the effect of the control variations on the system states. Applying equation B-8 to B-7 we obtain:

$$\begin{aligned} \Delta C = &\Big([\frac{\partial \phi}{\partial x} - \lambda^T]\Delta x(\Delta u)\Big)\Big|_{t=t_f} + \big[\lambda^T \Delta x(\Delta u)\big]\big|_{t=t_i} \\ &+ \int_{t_i}^{t_f} \Big[\frac{\partial \mathcal{H}}{\partial u}\Delta u + [\frac{\partial \mathcal{H}}{\partial x} + \dot{\lambda}^T]\Delta x(\Delta u)\Big]dt \\ &\triangleq \Delta C(t_f) + \Delta C(t_i) + \Delta C(t_i, t_f) \end{aligned} \tag{B-9}$$

where $\Delta u(t)$ is a small variation of the controls. If a trajectory satisfying the necessary conditions for optimality must have a stationary cost function in presence of control variations, it is clearly seen that all three terms in equation B-9 must be equal to zero.

It can be assumed that the initial control does not affect the initial state conditions, so $\Delta C(t_i) = 0$. Moreover, the adjoint vector λ can be chosen as:

$$\dot{\lambda}^T = -\frac{\partial \mathcal{H}}{\partial x} \tag{B-10}$$

subject to terminal conditions:

$$\lambda^T = -\left.\frac{\partial \phi}{\partial x}\right|_{t=t_f} \tag{B-11}$$

If moreover, the following conditions is fulfilled:

$$\frac{\partial \mathcal{H}}{\partial u} = 0 \tag{B-12}$$

then all three terms $\Delta C(t_f), \Delta C(t_i)$ and $\Delta C(t_i, t_f)$ are zero. Equations B-10,B-11 and B-12 form the set of necessary conditions for optimality. An optimal trajectory is achieved when these equations are fulfilled simultaneously with the boundary constraints defined at t_i and t_f.

Appendix C

Linear HJB

C-1 The log transform

To obtain the linear HJB expression, the optimal cost-to-go function $J(x,t)$ has been substituted by a logarithmic expression of $\Psi(x,t)$ also called the *desirability* function. This logarithmic transformation has its origin in the field of quantum mechanics, where a similar expression is used to decompose a wave function that satisfies the Schrödinger equation,Ψ as an amplitude and a phase (Feynman, Hibbs, & Styer, 2010):

$$\Psi = \sqrt{\rho} exp(iJ/\hbar) \tag{C-1}$$

where $\hbar$ is the Planck constant and ρ satisfies a Fokker-Planck equation and J satisfies a HJB equation. As explained in Kappen (2005), Equation C-1 can be related with the logarithmic transformation used in Chapter 4. The main difference is that only the J, HJB equation appears in the equation used here. When applying this logarithmic transformation within the field of optimal control, ρ, satisfying the Fokker-Planck equation, is used as an alternative to computing the HJB equation, whereas in the quantum mechanics field, both ρ and J are computed together.

C-2 The Path Integral formulation

Before deriving the path integral formulation of equation (4-8), we must discuss how the probability of a path traveling from $x_i \rightarrow x_f$ under the uncontrolled system dynamics is defined. To obtain an expression of $\rho(x_f, t_f; x_i, t_i)$, the connection between the SDE of the uncontrolled dynamics and a forward PDE is exploited. As explained in (E. A. Theodorou, 2011) both PDEs and SDEs formulations represent the same physical process, in this case a forward diffusion. The main difference is the fact that the PDE formulation gives a macroscopic view of the process while the SDE formulations approaches it in a microscopic view. Thus, the

SDE defining the uncontrolled dynamics can be written as a forward PDE defined by the Fokker-Planck equation[1], also known as the forward Kolmogrov equation. The transition probability for a single time step ($\epsilon = x_j \rightarrow x_{j+1}$) that satisfies the Fokker-Planck equation is a Gaussian defined as:

$$\rho(x_{j+1}, t_{j+1}\epsilon | x_j, t_j) = \frac{1}{\sqrt{2\pi \upsilon \epsilon}} exp\Big(-\frac{\epsilon}{\lambda}\Big[\frac{R}{2}\Big(\frac{x_{j+1} - x_j}{\epsilon} - f(x_j, t_j)\Big)\Big]\Big) \tag{C-2}$$

with variance $\sigma^2 = \upsilon\epsilon$ and mean value $x + f(x,t)\epsilon$. Recall from Section 3-1-3 that the process noise (dw) included in the uncontrolled system dynamics is modeled as a Wiener process follows a Gaussian distribution with mean zero and variance proportional to the time step, it can be noted that the Gaussian's variance is also proportional to the time step ϵ. As the mean of the noise is zero, the transition probability has a mean defined only by the deterministic system dynamics equal to $x + f(x,t)\epsilon$. To obtain equation C-2, the relation $\upsilon^{-1} = R/\lambda$ has been used.

The probability of a sample path going from $x_i \rightarrow x_f$ can now be written as a product of n infinitesimal transition probabilities over a time step ϵ as:

$$\begin{aligned}\rho(x_i, t_i; x_f, t_f) &= \int\int dx_1, ..., dx_{n-1}\rho(x_f, t_f; x_{n-1}, t_{n-1})...\rho(x_2, t_2; x_1, t_1)\rho(x_1, t_1; x_i t_i) \\ &= \Big(\frac{1}{\sqrt{2\pi\upsilon\epsilon}}\Big)^n \int exp\Big(-\frac{\epsilon}{\lambda}\Big[\frac{R}{2}\Big(\frac{x_{j+1} - x_j}{\epsilon} - f(x_j, t_j)\Big)\Big]\Big)\end{aligned} \tag{C-3}$$

with $t_j = t + (j-1)\epsilon$, $x_0 = x_i$ and $x_n = x_f$. Combining equation (C-3) and equation (4-8) we obtain the path integral formulation as:

$$\Psi(x,t) = \frac{1}{(\sqrt{2\pi\upsilon\epsilon})^n}\int dx_1...dx_n exp\Big(-\frac{1}{\lambda}S(x_{0:N})\Big) \tag{C-4}$$

where

$$S(x_{0:n}) = \phi(x_n) + \sum_{j=0}^{n-1}\epsilon q(x_j, t_j) + \sum_{j=0}^{n-1}\frac{R}{2}\Big(\frac{x_{j+1} - x_j}{\epsilon} - f(x_j, t_j)\Big) \tag{C-5}$$

[1]The Fokker Planck equation is a PDE that describes the time evolution of the probability density function of a stochastic variable (E. A. Theodorou, 2011).

Appendix D

PI algorithm for a LF task

The LF task has been implemented using PI control as explained in Chapter 4. The pseudocode of this algorithm is presented in this section. For a better understanding of the code, all functions used in the main program are presented separately. Both the main program 1 and the functions used within it, have an **input**, **output** and **data** parameters. The parameters specified within the **data** block are constant and common to all algorithms.

Algorithm 1 is the main program. It runs the main simulation loop, initializes the units (M) and calls the different sub-functions.

Algorithm 2 simply shifts over time the controls used for importance sampling $\tilde{U}$ to obtain the new exploring controls U. This shift is needed because the $\tilde{U}$ controls were calculated dt seconds ago and need to be shifted in time.

Algorithm 3 takes the position and velocity of the units at time t and initialized a number of N samples with those values as X and V. Initially the cost of all samples (S) is zero. Moreover, a random noise vector for all samples and for the entire planning horizon H is initialized. Finally the communicated position (x_comm) and velocity (v_comm) of the units for which the controls are not being calculated is initialized.

Algorithm 4 computes the cumulative cost of all samples which will be further used to obtain the controls for the leader unit. First, the velocities and positions of the samples are being propagated. Then, the cost function penalizes for being too far away from the target waypoint as well as having a heading not pointing towards it. Moreover, a control penalty is added as a function of R and $d\xi$.

Algorithm 5 computes the cumulative cost for all samples used to obtain the controls for the follower units. Similar to Algorithm 4, the velocities and positions of the samples are propagated. However, in this case, the cost function penalizes for being too far way from the leader ($C_{cohesion}$) as well as being too close to other units ($C_{collision}$). Moreover, the non-parallel constraint is implemented so followers are penalized ($C_{parallel}$) whenever the cross product between their velocity vector and other's units velocity vector is smaller than a certain threshold (cross_pr_thr). A control penalty is also added in this case.

Algorithm 6 simply steps the system forward in time. This procedure takes into account the inherent noise of the system's actuators and adds a certain random noise component to the velocity vector.

Algorithm 1: Main

```
Data: $N, H, dh, T, dt, R, \nu, M, penalties, constraints$
Input: $x_0, v_0$
Output: $x, v, u$
Compute $iT = T/dt$, $iH = H/dh$, $stdv = sqrt(\nu * dh)$, $\lambda = R * \nu$ ;
Define initial positions $x_0$,velocities $v_0$ and goal waypoints $wp$ ;
Initialize: $x, v, \tilde{U}$ ;                                   // $\tilde{U} \rightarrow [iH \times 2M]$
for $t$=1,...,$iT$ do
    U = $shift(\tilde{U})$ ;                                       // Shift controls with dt
    for $m$=1,...,$M$ do
        [X,V,S,$d\xi$,x_comm,v_comm] = init_samples(x[t][m],v[t][m]);
        if $unit$ = 1 then
            [S] = compute_cost_leader(X,V,S,U,x_comm,v_comm,wp,data) ;      // Compute sample's cost
        end
        else
            [S] = compute_cost_follower(X,V,S,U,x_comm,v_comm,wp,data);
        end
        W = compute_weight(S,$\lambda$) ;                          // Assign sample weights
        for $h$=1,...,$H$ do
            $\tilde{u}[h][m] = U[h][m] + \Sigma_n(W \Delta d\xi[h])/(dh \Sigma_n W)$ ;   // Approximate optimal control
        end
        Save controls for importance sampling;
    end
    $[x, v] = step(x, v, \tilde{u})$ ;                             // step system forward
    if $wp$ reached then
        take the following wp as the goal wp
    end
end
```

Algorithm 2: shift

```
Data: $iH, dh, dt, M$
Input: $\tilde{U}$
Output: $U$
$U = \tilde{U}$ ;
for $m$=1,...,$M$ do
    for $h$=1,...,$iH$-1 do
        $U[m][h] *= (dh - dt)/dt$; ;                               // Substract dt
        $U[m][h] += U[m][h+1]\Delta dt/(dh - dt)$; ;               // Propagate dt proportionally
    end
    $U[m][iH]* = (dh - dt)/dh$;
end
```

Algorithm 3: init_samples

```
Data: $v, dh, dt, N, h, M$
Input: $x[t][m], v[t][m]$
Output: $X, V, S, d\xi, x_comm, v_comm$
1  $U = \tilde{U}$ ;
2  for $n$=1,...,N ;                                              // for all samples
3   do
4      $S[n] = 0$ ;
5      $X[n] = x[t][m]$;
6      $V[n] = v[t][m]$;
7      for $h$=1,..,$iH$ do
8          $d\xi[n][h] \sim \aleph(0, \nu \Delta dh)$;
9      end
10     Obtain communicated position and velocity of other units x_comm, v_comm ;
11 end
```

Algorithm 4: compute_cost_leader

```
Data: $iH, dh, N, wp, R$
Input: $X, V, S, U, x_comm, v_comm, wp$
Output: $S$
1  for $n$=1,..,N do
2      for $h$=1,...,$iH$ do
3          $V[n] = V[n] + U[h]dh + d\xi[n][h]$; ;         // U[h] is constant for all samples
4          $X[n] = X[n] + V[n]dh$; ;                                    // propagate X,V
5          Compute target distance $D_{target}$;
6          $S[n] += C_{dist} * D_{target} * dh$ ;                   // Distance to target cost
7          Compute target heading $H_{target}$;
8          $S[n] += C_{head} * H_{target} * dh$ ;                   // Heading deviation cost
9          $S[n] += U[h]\Delta(U[h]dh/2 + d\xi[n][h])R$ ;                      // Control cost
10     end
11 end
```

Algorithm 5: compute_cost_follower

```
Data: $iH, dh, N, wp, R$, cross_pr_thr, collision_distance, cohesion_distance
Input: $X, V, S, U, x_comm, v_comm, wp$
Output: $S$
1  for $n$=1,..,N do
2      for $h$=1,...,iH do
3          $V[n] = V[n] + U[h]dh + d\xi[n][h]$ ;          // U[h] is constant for all samples
4          $X[n] = X[n] + V[n]dh$; ;                      // propagate X,V
5          for unit in M-1 do
6              Compute distance $D_{unit}$, cross product velocity vector $Cross_{unit}$;
7              if $Cross_{unit}$ ¡ cross_pr_thr then
8                  $S[n] += C_{parallel} * Cross_{unit} * dh$ ;      // Parallel flight cost
9              end
10             if $D_{unit}$ ¡ collision_distance then
11                 $S[n] += C_{collision} * D_{unit} * dh$ ;        // Collision cost
12             end
13             if $D_{unit}$ ¿ cohesion_distance then
14                 $S[n] += C_{cohesion} * D_{unit} * dh$ ;         // Cohesion cost
15             end
16         end
17         $S[n] += U[h]\Delta(U[h]dh/2 + d\xi[n][h])R$ ;           // Control cost
18     end
19 end
```

Algorithm 6: step

```
Data: $t, \nu, dh, M$
Input: $x, v, \tilde{u}$
Output: $x, v$
1  for m=1,...,M ;                                      // for all units
2   do
3      $d\xi[m][t] \sim \aleph(0, \nu\Delta dh)$;
4      ; $v[m][t+1] = v[m][t] + u[m][t]dt + d\xi[m][t]$;
5      $x[m][t+1] = x[m][t] + v[m][t]dt$;
6  end
```

An explanation of the notation used in the algorithms is given below.

N	number of samples
H	horizon time in seconds
dh	time increment for the planning loop
iH	number of discrete steps for the planning loop
T	simulation time in seconds
dt	time increment used in the simulation loop
iT	number of discrete steps for the simulation loop
M	number of units (MAVs) used
υ	variance of the process noise
R	control cost scaling
x_0	initial position of the units
v_0	initial velocity of the units
wp	target waypoints given to the leader unit
x	position of the units over all simulation time
v	velocity of the units over all simulation time
x_comm	communicated position of the other units
v_comm	communicated velocity of the other units
U	controls for all units and time horizon
$\tilde{U}$	controls for all units and time horizon used for importance sampling. Initialized with zeros.
$\tilde{u}$	approximated optimal controls for all units and all simulation time
S	cumulative cost over time of all samples
$d\xi$	random noise for all samples and all planning horizon
X	position vector of all samples at certain time t
V	velocity vector of all samples at certain time t
W	weight vector for all samples
cross_pr_thr	cross product threshold allowed to consider that two velocity vectors are not parallel
collision_distance	minimum distance allowed between two units before consider it as a collision
cohesion_distance	maximum distance allowed between the followers and the leader to ensure the cohesion of the group
D_{unit}	distance between two units
$C_{parallel}$	penalty assigned to samples with parallel velocity vectors of two units
$C_{collision}$	penalty assigned to samples beyond the collision_distance threshold
$C_{cohesion}$	penalty assigned to samples beyond the cohesion_distance threshold

Bibliography

Abouheaf, M. I., Lewis, F. L., Vamvoudakis, K. G., Haesaert, S., & Babuska, R. (2014). Multi-agent discrete-time graphical games and reinforcement learning solutions. *Automatica*, *50*(12), 3038 - 3053. Available from `http://www.sciencedirect.com/science/article/pii/S0005109814004282`

Alonso-Mora, J., Montijano, E., Schwager, M., & Rus, D. (2016, May). Distributed multi-robot formation control among obstacles: A geometric and optimization approach with consensus. In *2016 IEEE international conference on robotics and automation (icra)* (p. 5356-5363).

Anderson, B. D., & Moore, J. B. (2007). *Optimal control: linear quadratic methods.* Courier Corporation.

Anderson, R. P., & Milutinović, D. (2014). Stochastic optimal enhancement of distributed formation control using kalman smoothers. *Robotica*, *32*(2), 305–324.

Arslan, O., Theodorou, E. A., & Tsiotras, P. (2014, Dec). Information-theoretic stochastic optimal control via incremental sampling-based algorithms. In *2014 ieee symposium on adaptive dynamic programming and reinforcement learning (adprl)* (p. 1-8).

Bagchi, A. (1993). *Optimal control of stochastic systems.* Prentice-Hall, Inc.

Bandyopadhyay, S., Chung, S.-J., & Hadaegh, F. (2017). Probabilistic and distributed control of a large-scale swarm of autonomous agents. *IEEE Transactions on Robotics*, *33*(5), 1103-1123.

Bangura, M., & Mahony, R. (2014). Real-time model predictive control for quadrotors. *IFAC Proceedings Volumes*, *47*(3), 11773 - 11780. Available from `http://www.sciencedirect.com/science/article/pii/S1474667016434890` (19th IFAC World Congress)

Bayazıt, O. B., Lien, J.-M., & Amato, N. M. (2005). Swarming behavior using probabilistic roadmap techniques. In E. Şahin & W. M. Spears (Eds.), *Swarm robotics* (pp. 112–125). Berlin, Heidelberg: Springer Berlin Heidelberg.

Bellman, R. (1957). *Dynamic programming.* Princeton University Press.

Beni, G. (2005). From swarm intelligence to swarm robotics. In E. Şahin & W. M. Spears (Eds.), *Swarm robotics* (pp. 1–9). Berlin, Heidelberg: Springer Berlin Heidelberg.

Berg, J. van den, Abbeel, P., & Goldberg, K. (2011). Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, *30*(7), 895-913.

Bertsekas, D. P., & Castanon, D. A. (1989, Jun). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, *34*(6), 589-598.

Bertsekas, D. P., & Tsitsiklis, J. N. (1995, Dec). Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE conference on decision and control* (Vol. 1, p. 560-564 vol.1).

Bethke, B., How, J. P., & Ozdaglar, A. (2008, Dec). Approximate dynamic programming using support vector regression. In *2008 47th IEEE conference on decision and control* (p. 3811-3816).

Betts, J. T. (1998). Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, *21*(2), 193–207.

Betts, J. T. (2010). *Practical methods for optimal control and estimation using nonlinear programming* (Vol. 19). Siam.

Bhattacharya, R. N., & Waymire, E. C. (2009). *Stochastic processes with applications* (Vol. 61). Siam.

Biral, F., Bertolazzi, E., & Bosetti, P. (2016). Notes on numerical methods for solving optimal control problems. *IEEJ Journal of Industry Applications*, *5*(2), 154–166.

Capitan, J., Merino, L., & Ollero, A. (2014, May). Decentralized cooperation of multiple UAS for multi-target surveillance under uncertainties. In *2014 international conference on unmanned aircraft systems (ICUAS)* (p. 1196-1202).

Chen, J., Gauci, M., Li, W., Kolling, A., & Groß, R. (2015, April). Occlusion-based cooperative transport with a swarm of miniature mobile robots. *IEEE Transactions on Robotics*, *31*(2), 307-321.

Chen, P., & Waslander, S. (2010). Kinodynamic motion planning for holonomic UAVs in complex 3D environments.

Cheng, Z., Sun, Y., & Liu, Y. (2011). Path planning based on immune genetic algorithm for uav. In (p. 590-593).

Daily, R., & Bevly, D. M. (2008). Harmonic potential field path planning for high speed vehicles. *2008 American Control Conference*, 4609-4614.

Doerr, B. G., Linares, R., & Petersen, C. D. (2018). Spacecraft attitude control using path integral method via riemann manifold hamiltonian monte carlo. In *2018 space flight mechanics meeting* (p. 0204).

Dudek, G., & Jenkin, M. (2010). *Computational principles of mobile robotics.* Cambridge university press.

Ferguson, D., Likhachev, M., & Stentz, A. (2005). A guide to heuristic-based path planning. In *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (icaps)* (pp. 9–18).

Feynman, R. P. (1948, Apr). Space-time approach to non-relativistic quantum mechanics. *Rev. Mod. Phys.*, *20*, 367–387.

Feynman, R. P., Hibbs, A. R., & Styer, D. F. (2010). *Quantum mechanics and path integrals.* Courier Corporation.

Field, G., & Stepanenko, Y. (1996, Apr). Iterative dynamic programming: an approach to minimum energy trajectory planning for robotic manipulators. In *Proceedings of IEEE international conference on robotics and automation* (Vol. 3, p. 2755-2760 vol.3).

Floreano, D., & Wood, R. J. (2015). Science, technology and the future of small autonomous drones. *Nature*, *521*, 460-466.

Foderaro, G., & Ferrari, S. (2010, Dec). Necessary conditions for optimality for a distributed optimal control problem. In *49th IEEE conference on decision and control (cdc)* (p. 4831-4838).

Frego, M. (2014). *Numerical methods for optimal control problems with application to autonomous vehicles.* Unpublished doctoral dissertation, University of Trento.

Fu, J., & Topcu, U. (2015). Computational methods for stochastic control with metric interval temporal logic specifications. In *Decision and control (cdc), 2015 IEEE 54th annual conference on* (pp. 7440–7447).

García, D. A. L., & Gómez-Bravo, F. (2012). Vodec: A fast voronoi algorithm for car-like robot path planning in dynamic scenarios. *Robotica*, *30*, 1189-1201.

Garrido, S., Moreno, L., & Lima, P. U. (2011). Robot formation motion planning using fast marching. *Robotics and Autonomous Systems*, *59*(9), 675 - 683. Available from `http://www.sciencedirect.com/science/article/pii/S0921889011000959`

Goerzen, C., Kong, Z., & Mettler, B. (2010). A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, *57*(1-4), 65.

Gómez, V., Thijssen, S., Symington, A., Hailes, S., & Kappen, H. J. (2015). Real-time stochastic optimal control for multi-agent quadrotor swarms. *Robotics and Autonomous Systems. arXiv*, *1502*.

González, D., Pérez, J., Milanés, V., & Nashashibi, F. (2016, April). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *17*(4), 1135-1145.

Gorodetsky, A., Karaman, S., & Marzouk, Y. (2015). Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition. In (Vol. 11).

Guo, Y., & Parker, L. E. (2002). A distributed and optimal motion planning approach for multiple mobile robots. In *Proceedings 2002 IEEE international conference on robotics and automation (cat. no.02ch37292)* (Vol. 3, p. 2612-2619).

Ha, J., & Choi, H. (2016, May). A topology-guided path integral approach for stochastic optimal control. In *2016 ieee international conference on robotics and automation (icra)* (p. 4605-4612).

Handel, R. van. (May 2007). *Lecture notes in stochastic calculus, filtering,and stochastic control.* Princeton University.

Helm, S. van der, McGuire, K. N., Coppola, M., & Croon, G. C. de. (2018). On-board range-based relative localization for micro aerial vehicles in indoor leader-follower flight. *arXiv preprint arXiv:1805.07171*.

Horowitz, M. B., Damle, A., & Burdick, J. W. (2014). Linear hamilton jacobi bellman equations in high dimensions. *53rd IEEE Conference on Decision and Control*, 5880-5887.

How., J. (2008). *Principles of optimal control.* (Massachusetts Institute of Technology: MIT OpenCourseWare,https://ocw.mit.edu)

Howlett, J., Whalley, M., Tsenkov, P., Schulein, G., & Takahashi, M. (2007, 01). Flight evaluation of a system for unmanned rotorcraft reactive navigation in uncertain urban environments. , *1*, 507-523.

Huynh, V. A., Karaman, S., & Frazzoli, E. (2016). An incremental sampling-based algorithm for stochastic optimal control. *The International Journal of Robotics Research*, *35*(4), 305-333.

Jang, D.-S., Chae, H.-J., & Choi, H.-L. (2017). Optimal control-based UAV path planning with dynamically-constrained TSP with neighborhoods. *2017 17th International Conference on Control, Automation and Systems (ICCAS)*, 373-378.

Kala, R. (2012). Multi-robot path planning using co-evolutionary genetic programming. *Expert Systems with Applications*, *39*(3), 3817–3831. Available from `http://www.sciencedirect.com/science/article/pii/S0957417411014138`

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, *82*(1), 35–45.

Kappen, H. J. (2005). Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory and Experiment*, *2005*(11), P11011.

Kappen, H. J. (2007). An introduction to stochastic control theory, path integrals and reinforcement learning. In *AIP conference proceedings* (Vol. 887, pp. 149–181).

Kappen, H. J., & Ruiz, H. C. (2016, Mar 01). Adaptive importance sampling for control and inference. *Journal of Statistical Physics*, *162*(5), 1244–1266.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846-894.

Kempker, P. L., Ran, A. C. M., & Schuppen, J. H. van. (2011, Dec). A formation flying algorithm for autonomous underwater vehicles. In *2011 50th IEEE Conference on Decision and Control and European Control Conference* (p. 1293-1298).

Kendoul, F. (2012). Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, *29*(2), 315–378.

Kendoul, F. (2013). Towards a unified framework for UAS autonomy and technology readiness assessment (ATRA). In *Autonomous Control Systems and Vehicles* (pp. 55–71). Springer.

Kleinert, H. (2009). *Path integrals in quantum mechanics, statistics, polymer physics, and financial markets.* World scientific.

Koç, O., Maeda, G., & Peters, J. (2018). Online optimal trajectory generation for robot table tennis. *Robotics and Autonomous Systems*.

Korayem, M. H., Hoshiar, A. K., & Nazarahari, M. (2016, dec). A hybrid co-evolutionary genetic algorithm for multiple nanoparticle assembly task path planning. *The International Journal of Advanced Manufacturing Technology*, *87*(9-12), 3527–3543. Available from `http://link.springer.com/10.1007/s00170-016-8683-4`

Kreuzer, E., & Solowjow, E. (2018, Apr 01). Learning environmental fields with micro underwater vehicles: a path integral—gaussian markov random field approach. *Autonomous Robots*, *42*(4), 761–780.

Kuffner, J. J., Nishiwaki, K., Kagami, S., Inaba, M., & Inoue, H. (2001). Footstep planning among obstacles for biped robots. In *Proceedings 2001 IEEE/rsj international conference on intelligent robots and systems. expanding the societal role of robotics in the the next millennium (cat. no.01ch37180)* (Vol. 1, p. 500-505 vol.1).

Kuriki, Y., & Namerikawa, T. (2014, June). Consensus-based cooperative formation control with collision avoidance for a multi-UAV system. In *2014 american control conference* (p. 2077-2082).

Kushleyev, A., Mellinger, D., Powers, C., & Kumar, V. (2013, Nov 01). Towards a swarm of agile micro quadrotors. *Autonomous Robots*, *35*(4), 287–300.

Landau, D. P., & Binder, K. (2014). *A guide to monte carlo simulations in statistical physics.* Cambridge university press.

LaValle, S., & Hutchinson, S. (1998). Optimal motion planning for multiple robots having independent goals. *IEEE Transactions on Robotics and Automation*, *14*(6), 912-925.

LaValle, S. M. (2006). *Planning algorithms.* Cambridge university press.

Lawton, J. R. T., Beard, R. W., & Young, B. J. (2003, Dec). A decentralized approach to formation maneuvers. *IEEE Transactions on Robotics and Automation*, *19*(6), 933-941.

Lewis, M. A., & Tan, K.-H. (1997, Oct 01). High precision formation control of mobile robots using virtual structures. *Autonomous Robots*, *4*(4), 387–403.

Li, X., Xiao, J., & Tan, J. (2004, Aug). Modeling and controller design for multiple mobile robots formation control. In *2004 IEEE international conference on robotics and biomimetics* (p. 838-843).

Liao, Y.-H., & Sun, C.-T. (2001, May). An educational genetic algorithms learning tool. *IEEE Transactions on Education*, *44*(2), 20 pp.-.

Liu, C., Liu, H., & Yang, J. (2011). A path planning method based on adaptive genetic algorithm for mobile robot. *Journal of Information and Computational Science*, *8*(5), 808-814.

Liu, Y., & Bucknall, R. (2018). A survey of formation control and motion planning of multiple unmanned vehicles. *Robotica*, 1–29.

Mansouri, S. S., Nikolakopoulos, G., & Gustafsson, T. (2015, Nov). Distributed model predictive control for unmanned aerial vehicles. In *2015 workshop on research, education and development of unmanned aerial systems (RED-UAS)* (p. 152-161).

Menchón, S., & J. Kappen, H. (2018, 05). Learning effective state-feedback controllers through efficient multilevel importance samplers. *International Journal of Control*, 1-8.

Michael, N., Fink, J., & Kumar, V. (2011). Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, *30*(1), 73-86.

Milutinovi, D., & Lima, P. (2006, Dec). Modeling and optimal centralized control of a large-size robotic population. *IEEE Transactions on Robotics*, *22*(6), 1280-1285.

Minchev, Z., Manolov, O., Noykov, S., Witkowski, U., & Riickert, U. (2004, June). Fuzzy logic based intelligent motion control of robot swarm simulated by khepera robots. In *Intelligent systems, 2004. proceedings. 2004 2nd international IEEE conference* (Vol. 1, p. 305-310 Vol.1).

Munishkin, A. A., Milutinović, D., & Casbeer, D. W. (2016, June). Stochastic optimal control navigation with the avoidance of unsafe configurations. In *2016 international conference on unmanned aircraft systems (ICUAS)* (p. 211-218).

Navarro, I., & Matía, F. (2013, sep). An Introduction to Swarm Robotics. *ISRN Robotics*, *2013*, 1–10.

Omidshafiei, S., Agha–Mohammadi, A., Amato, C., Liu, S., How, J. P., & Vian, J. (2017). Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions. *The International Journal of Robotics Research*, *36*(2), 231-258.

Paiva, L. T. d. F. R. (2014). *Numerical methodes for optimal control and model predictive control.* Unpublished doctoral dissertation, Universidade do Porto (Portugal).

Panati, S., Baasandorj, B., & Chong, K. (2015). Autonomous mobile robot navigation using harmonic potential field. In (Vol. 83).

Passenberg, B. (2012). *Theory and algorithms for indirect methods in optimal control of hybrid systems.* Unpublished doctoral dissertation, Technische Universität München.

Perepelitsa, D. V. (n.d.). *Path integrals in quantum mechanics.* Citeseer.

Powell, W. B. (2007). *Approximate dynamic programming: Solving the curses of dimensionality* (Vol. 703). John Wiley & Sons.

Powell, W. B. (2012). Ai, or and control theory: A rosetta stone for stochastic optimization.

Prokhorov, D. V., & Wunsch, D. C. (1997, Sep). Adaptive critic designs. *IEEE Transactions on Neural Networks*, *8*(5), 997-1007.

Qiao, W., Harley, R. G., & Venayagamoorthy, G. K. (2009, June). Coordinated reactive power control of a large wind farm and a statcom using heuristic dynamic programming. *IEEE Transactions on Energy Conversion*, *24*(2), 493-503.

Quintero, S. A. P., Collins, G. E., & Hespanha, J. P. (2013, June). Flocking with fixed-wing uavs for distributed sensing: A stochastic optimal control approach. In *2013 american control conference* (p. 2025-2031).

Roberge, V., Tarbouchi, M., & Labonte, G. (2013). Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning. *IEEE Transactions on Industrial Informatics*, *9*(1), 132-141.

Rust, J. P. (1989). A dynamic programming model of retirement behavior. In *The economics of aging* (pp. 359–404). University of Chicago Press.

Şahin, E. (2005). Swarm robotics: From sources of inspiration to domains of application. In E. Şahin & W. M. Spears (Eds.), *Swarm robotics* (pp. 10–20). Berlin, Heidelberg: Springer Berlin Heidelberg.

Saska, M., Vonásek, V., Krajník, T., & Přeučil, L. (2014). Coordination and navigation of heterogeneous MAV–UGV formations localized by a 'hawk-eye'-like approach under a model predictive control scheme. *The International Journal of Robotics Research*, *33*(10), 1393–1412.

Scherer, S., Singh, S., Chamberlain, L., & Elgersma, M. (2008). Flying fast and low among obstacles: Methodology and experiments. *The International Journal of Robotics Research*, *27*(5), 549-574.

Schøler, F., Cour-Harbo, A. la, & Bisgaard, M. (2012). Generating approximative minimum length paths in 3D for UAVs. In *Intelligent vehicles symposium (iv), 2012 IEEE* (pp. 229–233).

Serra, D., Satici, A. C., Ruggiero, F., Lippiello, V., & Siciliano, B. (2016). An Optimal Trajectory Planner for a Robotic Batting Task: The Table Tennis Example. In *ICINCO (2)* (pp. 90–101).

Sholes, E. (2007). Evolution of a UAV autonomy classification taxonomy. In *Aerospace Conference, 2007 IEEE* (pp. 1–16).

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*(7676), 354.

Song, D.-P., & Earl, C. (2008). Optimal empty vehicle repositioning and fleet-sizing for two-depot service systems. *European Journal of Operational Research*, *185*(2), 760-777.

Stengel, R. F. (1986). *Stochastic optimal control: Theory and application.* John Wiley and Sons.

Stengel, R. F. (1994). *Optimal control and estimation.* Dover Publications.

Suicmez, E., & Kutay, A. (2014). Optimal path tracking control of a quadrotor uav. In (p. 115-125).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1) (No. 1). MIT press Cambridge.

Tassa, Y., Erez, T., & Smart, W. D. (2008). Receding horizon differential dynamic programming. In J. C. Platt, D. Koller, Y. Singer, & S. T. Roweis (Eds.), *Advances in neural information processing systems 20* (pp. 1465–1472). Curran Associates, Inc.

Theodorou, E., Buchli, J., & Schaal, S. (2010a). A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, *11*(Nov), 3137–3181.

Theodorou, E., Buchli, J., & Schaal, S. (2010b). Reinforcement learning of motor skills in high dimensions: A path integral approach. *2010 IEEE International Conference on Robotics and Automation*, 2397-2403.

Theodorou, E., Stulp, F., Buchli, J., & Schaal, S. (2011). An iterative path integral stochastic optimal control approach for learning robotic tasks. In (Vol. 18, p. 11594-11601).

Theodorou, E. A. (2011). *Iterative path integral stochastic optimal control: Theory and applications to motor control.* University of Southern California.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics.* MIT press.

Todorov, E. (2008, Dec). General duality between optimal control and estimation. In *2008 47th IEEE conference on decision and control* (p. 4286-4292).

Todorov, E. (2009). Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, *106*(28), 11478–11483.

Todorov, E., & Li, W. (2005, June). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, american control conference, 2005.* (p. 300-306 vol. 1).

Todorov, E., & Tassa, Y. (2009). Iterative local dynamic programming. In (p. 90-95).

Tong, H., Chao, W., Qiang, H., & Bo, X. (2012). Path planning of UAV based on voronoi diagram and dpso. In (Vol. 29, p. 4198-4203).

Vernaza, P., & Lee, D. (2011). Learning dimensional descent for optimal motion planning in high-dimensional spaces. In (Vol. 2, p. 1126-1132).

Wikipedia contributors. (2018). *Brownian motion — Wikipedia, the free encyclopedia.* Available from `https://en.wikipedia.org/w/index.php?title=Brownian-motion` ([Online; accessed 15-May-2018])

Williams, G., Aldrich, A., & Theodorou, E. A. (2017). Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, *40*(2), 344–357.

Williams, G., Drews, P., Goldfain, B., Rehg, J. M., & Theodorou, E. A. (2016). Aggressive driving with model predictive path integral control. In *Robotics and automation (icra), 2016 IEEE international conference on* (pp. 1433–1440).

Williams, G., Rombokas, E., & Daniel, T. (2015). Gpu based path integral control with learned dynamics. *arXiv preprint arXiv:1503.00330*.

Yang, L., Qi, J., Xiao, J., & Yong, X. (2014). A literature review of UAV 3D path planning. In *Intelligent Control and Automation (WCICA), 2014 11th World Congress on* (pp. 2376–2381).

Zammit, C., & Van Kampen, E.-J. (2018). Comparison between a* and rrt algorithms for UAV path planning. In *2018 aiaa guidance, navigation, and control conference* (p. 1846).