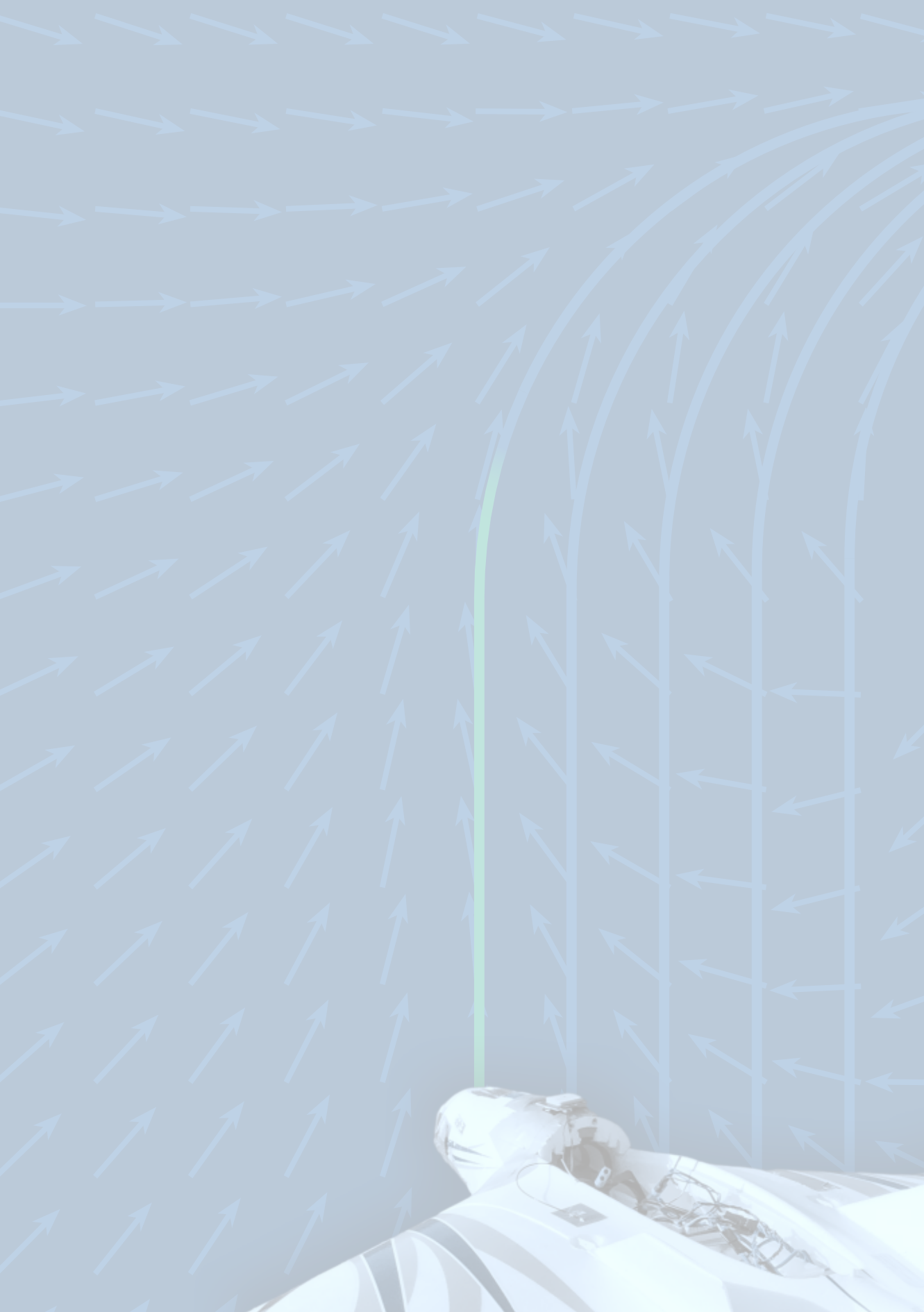
PhD Thesis

Adam Seewald

Energy-Aware Coverage Planning and Scheduling for Autonomous Aerial Robots



University of Southern Denmark

Energy-Aware Coverage Planning and Scheduling for Autonomous Aerial Robots

A Dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Robotics

by

Adam Seewald

Approved by:

Prof. Ulrik Pagh Schultz, Advisor
Unmanned Aerial Systems Center
University of Southern Denmark

Approved on:

2021

https://doi.org/10.

The typesetting is done using LaTeX. Figures are generated with PGF/Ti*k*Z. Fonts are EB Garamond, Helvetica, and Iwona for body, headers, and equations respectively.

First edition, 2021

Published by University of Southern Denmark, in Odense, Denmark

"[I]t was not possible to run it more than a minute or two at a time. In these short tests the motor developed about nine horse power. We were then satisfied that, with proper lubrication and better adjustments, a little more power could be expected."

— O. Wright, 1913

Acknowledgements

Contents

Figures

Notation

$\exists$	there exists
$\in$	is an element of the set
$\notin$	is not an element of the set
$:=$	is defined
$\approx$	approximately equal
$f(\cdot)$	is function f
$f : \mathbb{C} \to \mathbb{D}$	f maps set $\mathbb{C}$ to set $\mathbb{D}$
$\mathbb{C} \cap \mathbb{D}$	intersection of set $\mathbb{C}$ with set $\mathbb{D}$
$\mathbb{C} \cup \mathbb{D}$	union of set $\mathbb{C}$ with set $\mathbb{D}$
∇	del operator
$\mathbb{Z}$	set of integers
$\mathbb{Z}_{\geq 0}$	set of positive integers
$\mathbb{Z}_{>0}$	set of strictly positive integers
$\mathbb{R}$	set of reals
$\mathbb{R}_{\geq 0}$	set of positive reals
$\mathbb{Z}^m, \mathbb{R}^m$	integer- or real-valued vector of m elements
$\mathbb{Z}^{m \times n}, \mathbb{R}^{m \times n}$	integer- or real-valued matrix of m rows, n columns, and mn elements
$\emptyset$	empty set
$\mathbb{C} \setminus \mathbb{D}$	elements of set $\mathbb{C}$ not in set $\mathbb{D}$
$\mathbb{C} \supseteq \mathbb{D}$	set $\mathbb{C}$ is a superset of set $\mathbb{D}$
$\mathbb{C} \supset \mathbb{D}$	set $\mathbb{C}$ is a strict superset of set $\mathbb{D}$
$\mathbb{C} \subseteq \mathbb{D}$	set $\mathbb{C}$ is a subset of set $\mathbb{D}$
$\mathbb{C} \subset \mathbb{D}$	set $\mathbb{C}$ is a strict subset of set $\mathbb{D}$
$[x]$	set of positive integers up to $x \in \mathbb{Z}_{\geq 0}$
$[x]_{>0}$	set of strictly positive integers up to $x \in \mathbb{Z}_{>0}$
$\mathbf{x}$	vector
X	matrix

$\mathbf{x}'$, X'	transpose of a vector $\mathbf{x}$ or of a matrix X
$\mathbf{x}^{-1}$, X^{-1}	inverse of a vector $\mathbf{x}$ or of a matrix X
$\|\|\mathbf{x}\|\|$	euclidean norm of vector $\mathbf{x} \in \mathbb{R}^n$
x_i	ith set of parameters
$x_{i,j}$	jth parameter of the ith set of parameters
$\underline{x}_{i,j}$	lower bound of the parameter $x_{i,j}$
$\overline{x}_{i,j}$	upper bound of the parameter $x_{i,j}$
$\|x\|$	cardinality of a set x
x^*	optimal with respect to a given cost
$\lfloor x \rfloor$	the floor function or integer division of $x \in \mathbb{R}$, if x is a set, the closest item in the set s.t. $x - \lfloor x \rfloor$ is positive
$\lceil x \rceil$	the ceiling function of $x \in \mathbb{R}$, if x is a set, the closest item in the set s.t. $x - \lceil x \rceil$ is negative
$v_1\big\vert_{v_2}$	edge connecting vertices v_1 and v_2

Abbreviations

LP	linear program
QP	quadratic program
MPC	model predictive control
NLP	non linear program
UAV	unmanned aerial vehicle
UAS	unmanned aerial system
OCP	optimal control problem
BVP	boundary-value problem
RSTA	reconnaissance, surveillance, and target acquisition
GNSS	global navigation satellite system
IMU	inertial measurement unit
RPV	remotely piloted vehicle
GPS	global positioning system
SoC	state of charge
M&CE	difference of motion and computations energy
CNN	convolutional neural network
FPS	frames per second
FSM	finite state machine
DVS	dynamic voltage scaling
DFS	dynamic frequency scaling
DNN	deep neural network
QoS	quality of service
CPP	coverage path planning
UGV	unmanned ground vehicle
MSE	mean square error
SLAM	simultaneous localization and mapping
CSV	comma-separated values
ECM	equivalent circuit model

FCN	fully convolutional network

Chapter 1

Introduction

> *"Fixed-wing[s] [...] tend to be more stable in the air in the face of both piloting and technical errors as they have natural gliding capabilities even without power, and they are able to travel longer distances on less power."*
>
> — X. Wang et al., 2017

Mobile robots are a class of robots with the ability to move through the environment (Corke, 2017). Most of these robots are power-demanding devices constrained by battery limitations. Although such limitations are a common challenge in many areas, they are critical in mobile robots' design and development. They influence the level of autonomy (Seewald, Garcia de Marina, Midtiby, et al., 2020), which in turn is expected to increase in the foreseeable future (Fisher et al., 2013).

To move, mobile robots combine different components which sense and interact with the surrounding environment (Mei, Y.-H. Lu, Y. C. Hu, et al., 2006). The analysis and interpretation of the data originating from these components often rely on high-level decisions. These are usually implemented on energy-demanding heterogeneous computing hardware. Planning an energy-aware path with a power-saving scheduling policy on such hardware is an underrepresented topic in the literature. Many past approaches focus almost exclusively on one of these topics. Some generate an energy-optimized path, despite the computations energy almost equaling the motion energy of some instances of low-energy mobile robots (Sudhakar et al., 2020). Moreover, due to the recent advancements in the computational capabilities of heterogeneous computing hardware, such as the introduction of powerful portable GPUs (Rizvi et al., 2017), the use of computations is on the rise (Abramov et al., 2012; Jaramillo-Avila et al., 2019; Satria et al., 2016). Others provide a power-saving scheduling policy. Yet, moving a mobile robot requires considerable energy expenditure over mere computations (Mei, Y.-H. Lu, Y. C. Hu, et al., 2004, 2005).

Figure 1.1. Opterra fixed-wing aerial robot employed for precision agriculture (photo credit: Amit Ferencz Appel).

In this work, we focus on dynamic energy planning. We generate both an energy-aware plan of the path and a power-saving schedule of the computations. We plan these two aspects simultaneously, exploring their tradeoffs. We demonstrate the approach both in simulation and empirically, and we focus on aerial mobile robots. Although these systems share with the broader class of mobile robots stringent battery limitations, handling the battery in flight introduces additional complications. It is generally required landing to replace or recharge the battery (Zamanakos et al., 2020). Aerial robots are thus an ideal instance of energy-constrained systems that would benefit from a dynamic energy planning technique. In the remainder of this work, we thus focus our analysis on aerial robots.

There are many autonomous use cases involving aerial robots, such as precision agriculture, search and rescue, payload delivery, transportation, and many others. To formulate a mobile robot planning problem later in Chapter 2, we focus on precision agriculture and progressively build dynamic planning of a fixed-wing aerial robot flying over an agricultural field with little human input.

We investigate different physical aerial robotics platforms in this work but we focus most on the Opterra fixed-wing aerial robot (Hobby, 2020) adapted for precision agriculture. Precision agriculture is often put into practice (Hajjaj and Sahari, 2014) with ground mobile robots used for harvesting (Aljanobi et al., 2010; De-An et al., 2011; F. Dong et al., 2011; Edan et al., 2000; Z. Li et al., 2008; Qingchun et al., 2012), and unmanned aerial vehicles (UAVs) for preventing damage and ensuring better crop quality (Daponte et al., 2019; Puri et al., 2017). The aerial robot is shown in Figure 1.1.

In the remainder of the chapter and before we introduce the problem in Chapter 2, we briefly investigate the evolution of the field of aerial robotics in the next section. We then analyze the

main aerial robots available today, and how they apply to dynamic energy planning in Section 1.2. We then provide some further motivation for our planning in Section 1.3 and the outline of the approach in Section 1.4. Finally, we provide the structure of the remaining chapters in Section 1.8.

This chapter connects to the remainder of this work as follows. Here we introduce and motivate the dynamic energy planning for autonomous aerial robots. We formalize the planning problem in Chapter 2. We describe the derivation of a proper energy model to predict the future energy consumption of the planning problem in Chapter 4. We estimate some coefficients of the model using robust estimation techniques in Chapter A. We implement an optimal configuration of the path and computations using data-driven control and other modern optimal control techniques in Chapter 5. We use such configuration for guidance and scheduling of the aerial robot. The guidance moves physically the robot; the scheduling defines the granularity of the tasks being executed in flight. Moreover, we discuss previous approaches to solve the dynamic planning problem in Chapter 3.

1.1 From UAVs to Modern Aerial Robots

Modern aerial robots are a valuable tool in robotic research and aerospace. They are found with different names in the literature. These include unmanned aerial vehicles (UAVs)[1], unmanned aerial systems (UASs), flying robots, or drones. Usually, we refer to drones, UAVs, and UASs when these systems are semi-autonomous, operated from the ground. UAS often denotes the entire infrastructure of unmanned flight in the aerospace jargon. Aerial or flying robots, on the other hand, have advanced levels of autonomy (Siciliano and Khatib, 2016). Nevertheless, all these systems have basic autonomous features such as position and altitude holding and leveling. The position holding is usually implemented using global navigation satellite system (GNSS), altitude holding using a barometer, and leveling using inertial measurement unit (IMU).

The origin of the field of aerial robotics, which deals with the design and development of aerial robots, dates back to the first guided missiles: unmanned (or uninhabited) flight has more than a century of developments (Siciliano and Khatib, 2016). Hewitt-Sperry Automatic Airplane, also denominated "flying bomb", developed in 1917 during World War I (WWI) (Keane and Carr, 2013; Valavanis and Vachtsevanos, 2015) is often referred to as the first unmanned flying machine. It is shown in Figure 1.2. It has been developed 14 years after the first heavier-than-air flight in history, demonstrated on December 17, 1903, with the Wright Flyer I by Wilbur and Orville Wright (or the Wright brothers). The Hewitt-Sperry Automatic Airplane used a gyroscope similarly to modern aerial robots. The device, invented by Elmer Sperry (Keane and Carr, 2013), was mechanically connected to the control surfaces and successfully implemented a control feedback loop (Siciliano and Khatib, 2016).

In the early days, the first flying machines were referred to as remotely piloted vehicles (RPVs) (Anderson, 2005). Many instances from WWI on of these vehicles were designed for military purposes. In the 1950s, United States used a remotely controlled vehicle, the Ryan Firebee, for reconnaissance in Vietnam, and Israel was the first to use a RPV in a combat situation (Ander-

[1] The term unmanned is sometimes replaced by uninhabited

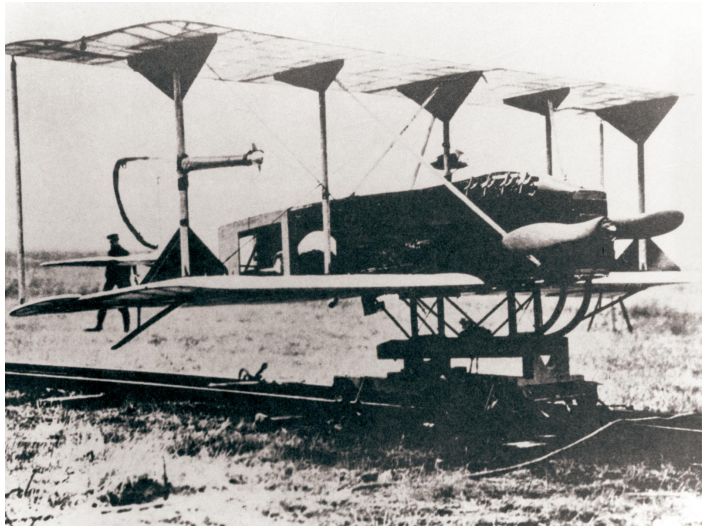


Figure 1.2. The Hewitt-Sperry Automatic Airplane, also denominated "flying bomb", was developed in WWI and represents the first instance of a UAV (photo credit: United States Naval Institute).

son, 2005). Other instances of these vehicles include the V-1 flying bomb from 1944 (deployed by the unified armed forces of nazi Germany) and the Lockheed D-21 from 1962 (deployed by the United States Air Force). Global positioning system (GPS) at the end of 1970 allowed some more recent remotely piloted vehicles to be used in surveillance. These systems were later integrated with cameras and other sensors (Siciliano and Khatib, 2016), in what are the modern aerial robots.

In recent years, the unmanned flight has been applied in many civilian applications (González-Jorge et al., 2017). Modern aerial robots are increasingly used in remote sensing (Colomina and Molina, 2014; Milas et al., 2018; Noor et al., 2018; Tang and Shao, 2015), surveillance (Acevedo et al., 2014; Basilico and Carpin, 2015; Bürkle, 2009; Paucar et al., 2018; Ramasamy and Ghose, 2017), meteorology (Renzaglia et al., 2016; Schuyler et al., 2019), search and rescue (Cui et al., 2015; Hayat et al., 2017; Karaca et al., 2018; Pensieri et al., 2020; Seguin et al., 2018), precision agriculture (Daponte et al., 2019; Lottes et al., 2017; Popović et al., 2017; Puri et al., 2017; Sa et al., 2018), transportation, and payload delivery (Kellermann et al., 2020). The former four categories fall into the area of reconnaissance, surveillance, and target acquisition (RSTA) and do not require advanced autonomy. Precision agriculture, transportation, and payload delivery are often realized using to a greater or lesser extent some advanced levels of computational intelligence (Siciliano and Khatib, 2016). Modern aerial robots are designed to handle unexplored terrain with little interaction as opposed to the past UAVs operated mainly by a human operator (Siciliano and Khatib, 2016). Instances of modern aerial robots are expected to autonomously adapt and possibly interact in a broad variety of environmental conditions.

In summary, aerial robots have a relatively recent past. Some initial experiments of unmanned flights were performed shortly after the first heavier-than-air manned, powered flight. These

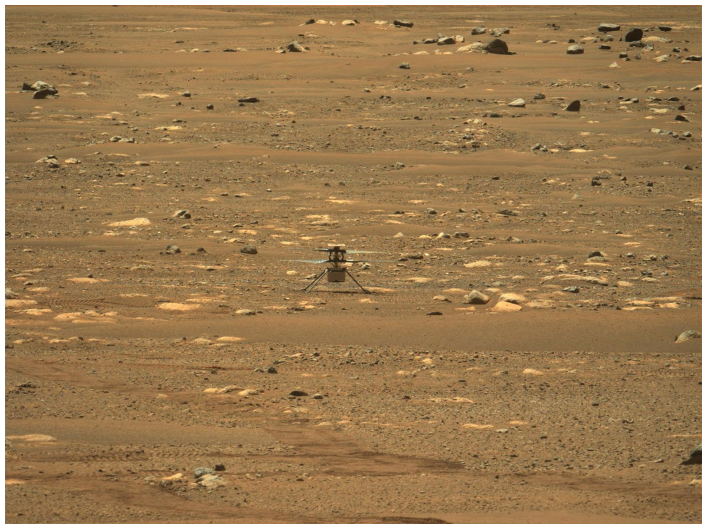


Figure 1.3. NASA's Ingenuity Mars Helicopter. A rotary-wing coax aerial robot that achieved the first powered, controlled flight on another planet on April 19, 2021. The project is solely a demonstration of technology (photo credit: NASA Jet Propulsion Laboratory).

initial experiments were mostly developed for military purposes, whereas modern aerial robots are used in a broad range of civil applications. Aerial robots are expected to grow significantly in numerous areas of robotics research ranging from agriculture to planetary exploration. For the latter, Figure 1.3 shows NASA's Ingenuity Mars Helicopter. A small coaxial aerial robot that performed the first powered, controlled flight on another planet on April 19, 2021. Aerial robots for planetary exploration are to be further deployed in future explorations endeavors, for instance, to study Saturn's moon Titan (Voosen, 2019).

Finally, we conclude this section with a quote from (Anderson, 2005): "the Wright brothers worked so hard to put humans in the air in flying machines, a hundred years later some [...] are working hard to take them out of flying machines".

1.2 Common Classes of Aerial Robots

Numerous different types of aerial robots have emerged ever since their first introduction. We briefly investigate the most studied classes in the robotics literature and relate them to the dynamic energy planning that we focus on in this work. The two most generic classes are heavier-than-air and lighter-than-air aerial robots. Heavier-than-air aerial robots are divided into fixed- and rotary-wings (Siciliano and Khatib, 2016), and some recent developments in bio-inspired robotics study flapping-wings (Floreano and Wood, 2015).

Rotary-wing aerial robots are highly maneuverable and can perform stationary vertical flight (commonly referred to as hovering) (Siciliano and Khatib, 2016). These systems can be classified into further categories, which include multirotors (such as quadrotors or quadcopters, hexa-

Figure 1.4. Skye, an omnidirectional spherical blimp developed by ETH Zürich for entertainment purposes. It has a camera system and combines the energy-efficient flight of a blimp with the characteristics of a quadrotor (photo credit: ETH Zürich).

copters, and octocopters), conventional helicopters (these have one main and one tail rotor), and a coax (these have counter-rotating coaxial rotors) (Corke, 2017). Some examples of quadrotors are DJI Mavic Mini in (h), and DJI Phantom 4 in (j) in Figure 1.5. In the same figure, DJI Agras T16 in (g), and DJI Matrice 600 in (f) are hexacopters.

Fixed-wing aerial robots have wings to provide the lift, some control surfaces for maneuvering, and a propeller for forward thrust; a shared principle with a common passenger aircraft (Corke, 2017). An example constitutes the Opterra adapted for precision agriculture in Figure 1.1, Cumulus in (b), Ebee in (d), and Penguin BE in (c) (Haugen and Imsland, 2016) in Figure 1.5. Examples of flapping-wings are Delfly II in (k) (Percin et al., 2012), and Nano-Hummingbird in (l) in Figure 1.5. We discuss in detail heavier-than-air aerial robots in this work, as they are a common platform for robotics research. We focus on the dynamic forces which govern the flight of rotary- and fixed-wings aerial robots in Chapter 5.

Common instances of lighter-than-air aerial robots are blimps (or non-rigid airships). They usually rely on a gas–such as helium enclosed in a protected envelope (Burri et al., 2013)–to generate the lifting force (Fui Liew et al., 2017). An omnidirectional spherical blimp is shown in Figure 1.4. Blimps are similar to balloons but provide basic maneuverability, whereas, in a balloon, only the altitude can be controlled (Colombatti et al., 2011).

Some other classifications found in aerial robotics literature trade size and maneuverability and include classes as micro aerial vehicles (MAVs) or vertical take-off and landing (VTOLs) aerial robots. The former are aerial robots with all dimensions lower than 15 cm. The latter are aerial robots flying in a fixed-wing configuration except if taking-off and landing where they use thrust from rotors rather than lift from wings.

Among the classes in this section, rotary-wings are the most maneuverable and lighter-than-air aerial robots are the least. These, however, have the highest flight time followed by fixed-wing aerial robots. Mixed configurations, such as VTOLs, fall into the intersection of rotary and fixed-wings for what concerns maneuverability and flight time (Siciliano and Khatib, 2016). The energy requirements are critical for all aerial robots, but the difference of motion and computations energy (M&CE) varies greatly. The energy–inversely proportional to flight time–is highest in rotary-wings and lowest in lighten-than-air aerial robots as shown later in Figure 1.5. The dynamic energy planning approach thus relies on both power-saving task scheduling and energy-efficient path planning for the fixed-wings robots while relying almost exclusively on energy-efficient path planning for rotary-wing aerial robots. In the former M&CE is close to zero, in the latter is usually high except in energy optimized designs such as some rotary-wing MAVs. Hypothetically, in lighter-than-air aerial robots, M&CE might be negative; the energy planning approach would rely heavily on power-saving task scheduling.

The classification that we proposed in this section serves the scopes of our work. There are similar effort for classification by size and propulsion technology (Cabreira, Brisolara, et al., 2019; Hoffer et al., 2014) but of course other classifications are possible. For instance categorizing aerial robots by the altitude they usually operate at (Watts et al., 2012).

1.3 Motivation

In this section, we provide a brief motivation for dynamic energy planning of these systems. Many applications that involve aerial robots have strict battery constraints. Although this is a common problem to most mobile robots, aerial robots are of particular concern. The autonomy of these systems is affected by the availability of the power source. A typical example of an aerial robotics scenario is a robot following a path and performing some onboard computations. For instance, the robot might detect ground patterns and notify other ground-based actors with little human interaction in precision agriculture or another application. Aerial robots in this situation often carry some heterogeneous computing hardware and a microcontroller. Energy requirements of such computing hardware are a further complication. Computing hardware is involved in autonomous capabilities and planning itself–both often require computer vision or other complex algorithms–whereas microcontroller runs motion primitives by directly interfacing actuators (such as servos for the control surfaces) and motors. Energy-wise, it is advantageous to schedule the computations on the computing hardware and simultaneously to plan the path.

1.3.1 Path planning and computations scheduling

It is uncommon to find a ready-to-use solution for both planning the path and scheduling the computations of these systems in an energy-aware fashion (we discuss in detail the state-of-the-art in this regard in Chapter 3). Planning the two aspects simultaneously would pose an advantage in terms of autonomy by, e.g., optimizing the path and computations in the function of the battery state of charge (SoC).

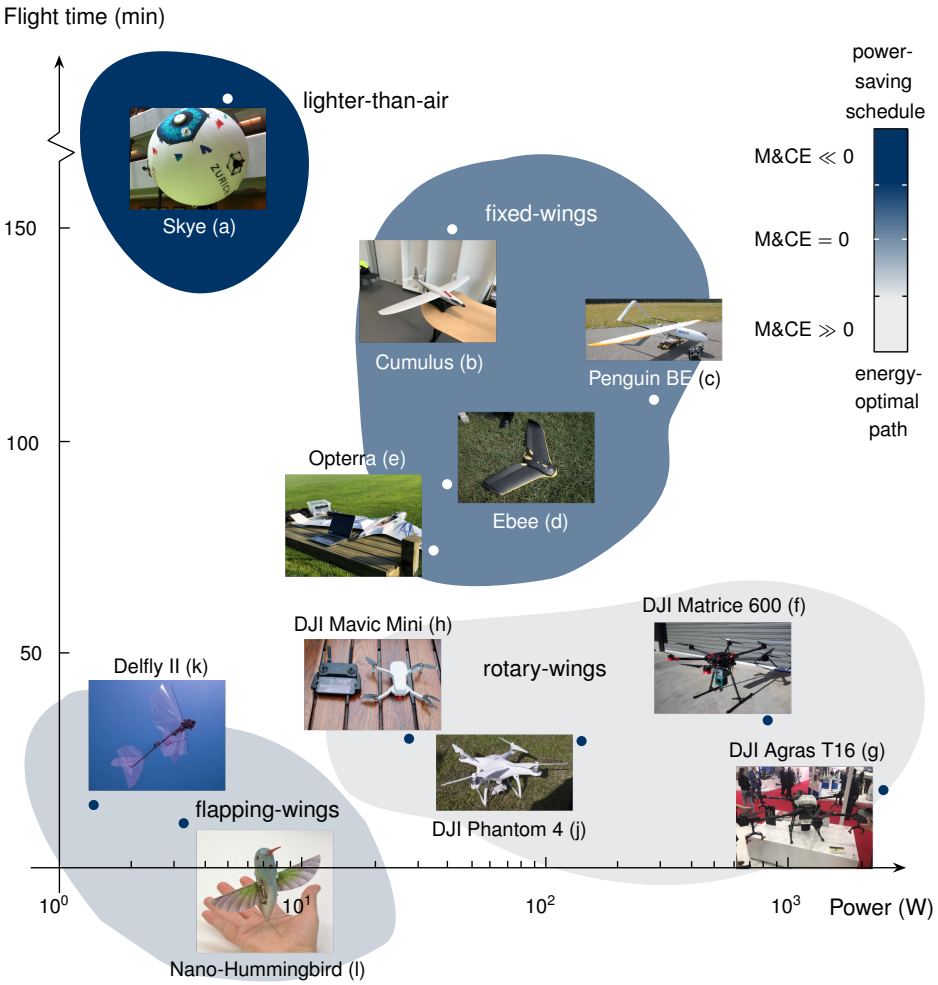


Figure 1.5. Different aerial robots in relation to the power, flight time, and M&CE with a hypothetical fixed costs for computations energy. The power is expressed using a logarithmic scale. Heavier-than-air aerial robots (b)–(l) include flapping-wings (k), (l) and have a negative M&CE (computations scheduling is to be accounted for most in planning), whereas rotary-wings have a positive M&CE (f), (g) (path planning is to be accounted for most in dynamic planning). Some smaller rotary-wings have a M&CE closer to zero (h), (j). In general, rotary-wings have a short flight time. Fixed-wings (b)–(e) have a considerably longer flight time and M&CE close to zero (computations scheduling and path planning have both to be accounted for in dynamic planning). Lighter-than-air aerial robots (a) have hypothetically a relatively long flight time and lower than zero M&CE (photos credit: (b) to Sky-Watch, (f) to Rise Above, (g) to Aeromotus, (h) to Digital Photography Review, (j) to ePHOTOzine, and (l) to DARPA).

Figure 1.6. The coverage problem in a precision agriculture scenario. The aerial robot has to cover an agricultural field that forms a polygon (blue/transparent area in the frame) and run some autonomous tasks (photo credit: Amit Ferencz Appel).

For certain classes of aerial robots with M&CE close or lower than zero, the autonomy can directly influence the battery state. For these classes, it is desirable to reschedule the computations energy-wise in-flight during a motion energy-demanding phase. For instance, a fixed-wing aerial robot might be flying headwinds (with the wind vector parallel and opposite to the direction of motion) and utilizing more energy than planned. It would be of advantage to reschedule the tasks accordingly to save energy needed for motion. During the same flight, the wind direction might suddenly change. The fixed-wing craft, now flying tailwinds, requires less motion energy. It could then potentially increase the level of computations by rescheduling the tasks. Later in the flight, the battery might be subject to sudden drops (due to e.g., temperature changes) requiring replanning again by, e.g., shortening the path. It is clear that planning the path and scheduling the computations simultaneously in all these cases is the most desirable course of action.

In Figure 1.5 we show the M&CE against the flight time of different aerial robots. We observe that fixed-wings are the aerial robots that would advantage most from simultaneous path planning and computations scheduling. They have a M&CE close to zero and a relatively long flight time. Although some rotary-wings have M&CE also close to zero, their flight time is generally relatively short. Flapping-wing and lighter-than-air aerial robots require considerably less energy for the motion, and have a negative M&CE.

1.3.2 Objective

In the remainder of this work, we refer to computational tasks that can be scheduled in an energy-aware fashion as computations, opposed to the other tasks with no significant effect on energy consumption. We assume the aerial robot runs the computations on the heterogeneous computing hardware. As an example, we refer to an aerial robot in a precision agriculture scenario

in Figure 1.1, which covers an agricultural field. In abstract terms, the field is represented by a polygon in Figure 1.6 and the aerial robot detects some patterns on such polygon using a convolutional neural network (CNN). The aerial robot further communicates the position of a detected pattern to other, ground-based actors. Detecting the patterns usually involves heterogeneous computing elements (GPU and/or multiple CPU cores) and consumes a significant amount of energy as opposed to communication. Detection is a computation, while communication is an (energy-inexpensive) task.

We are interested in the energy optimization of the path and computations under uncertainty (atmospheric interferences) in-flight and refer to it as dynamic energy planning. Unlike most of the past mobile robotics planning literature, the approach that we propose plans these two aspects simultaneously. Such planning would find optimal tradeoffs between the path, computations, and energy requirements. Current generic planning solutions for aerial robots do not plan the path and computations dynamically, nor are they energy-aware. They are often semi-autonomous: the path and computations are static and usually defined using planning software (Daponte et al., 2019). Instances of the planning software include famous flight controllers (Paparazzi, 2016; PX4, 2016). Such a state of practice has prompted us to investigate the topics presented in this work. We will gradually propose a dynamic energy planning approach for aerial robots. It plans both the path and computations while the aerial robot is flying and its batteries draining.

We formally define the robot planning problem after outlining the approach in the next section.

1.4 Outline of the Approach

In this work, we address the increasing demands for autonomy of modern aerial robots with a dynamic planning approach. The approach provides a simultaneous energy-aware plan of the path and a power-saving schedule of the computations. To this end, we first need a user-defined initial plan that is then replanned energy-wise using an energy model for future prediction. Later, an algorithm inputs the initial plan, estimates the state of the energy model, evaluates future energy consumption and replans parameters relative to the path and computations. The plan consists of different stages. At each stage, the aerial robot flies a path and executes some computations. Furthermore, it contains some additional parameters to alter the path and computations along with an energy budget. The alterations are bounded. There are path constraint sets that bound the path alterations and computations constraint sets, one per each computation, that bound computations alterations.

We use the concept of different stages to model complex paths. For instance, the path might contain multiple circles and lines. We will see such plan later in Section 2.6. The approach guides the aerial robot on different paths (the plan is composed of) using vector field (Garcia De Marina et al., 2017). The robot switches between the paths as soon as it reaches specific triggering points.

The approach further relies on `powprofiler`, a profiling and modeling tool that we introduce in Section 4.1. The tool models the energy consumption of the computations and is later em-

ployed to estimate the future energy of the aerial robot in flight. To this end, we empirically derive and formally prove a periodic energy model that accounts for the uncertainty. We use Fourier analysis to derive the model and state estimation to address the uncertainty. Periodicity is often present due to repetitive patterns in the plan (Seewald, Garcia de Marina, Midtiby, et al., 2020). Indeed, mobile robots often iterate over a set of tasks and paths (Seewald, Garcia de Marina, and Schultz, n.d.). Given that the plan is periodic, we expect the energy consumption to evolve (approximately) periodically. We describe in detail such model in Section ??.

The replanning of the path and computations is achieved with modern optimal control techniques, such as state estimation in Chapter A, and model predictive control (MPC) (Rawlings et al., 2017) in Chapter 5. The control is data-driven. Energy sensor data estimates some coefficients of the model used to predict the future energy consumption with uncertainty. The replanning is done under an energy budget–the battery capacity and other battery parameters. Our goal is to complete the plan with the highest possible parameters configuration.

1.5 Applications

The dynamic energy planning in this work applies to modern aerial robots with a certain degree of autonomy. By the latter, we mean that the robot performs at least a predefined set of tasks over a given space. Although most of the guidance in Chapter 5 is designed for aerial robots specifically, the approach can be easily adapted to other mobile robots with energy constraints. We discuss applications out of aerial robotics domain further in Chapter 7. For instance, we have applied the approach to the space robotics context (Seewald, 2020).

In the remainder of this work, we focus on the precision agriculture scenario in Figure 1.6. In summary, the aerial robot covers a given agricultural field (a polygon) and searches for ground hazards. With the scenario, we validated the work experimentally. Path-wise, the aerial robot flies in circles and lines covering the polygon. Computation-wise, it detects hazards using the CNN and notifies grounded mobile robots employed for, e.g., harvesting. The approach alters the plan; it controls the processing rate and the radius of the circles affecting the distance between the lines. We will see this concrete scenario described formally in Section 2.6 and solved progressively in the remaining chapters.

1.6 Methodology

1.7 Contribution

With this work, we contribute with a dynamic planning approach for aerial robots performing autonomous scenarios for different use-cases, under energy constraint, and in an uncertain environment. In particular, we plan both an energy-optimized path and a power-saving schedule. Indeed dynamic planning that incorporates both these aspects simultaneously is underrepresented in the literature.

The contribution in this work builds upon some other past contributions. In particular, our computational energy modeling relies on an automated profiling and modeling methodology that we proposed in (Seewald, Schultz, Ebeid, et al., 2021). The methodology is based on the `powprofiler` tool to derive an energy model for future energy prediction. We proposed an extension of `powprofiler` with a component-based energy modeling approach to abstract per-component energy in a dataflow computational network in (Seewald, Schultz, Roeder, et al., 2019). We later integrated the tool in a Robot Operating System (ROS) (Quigley et al., 2009) in (Zamanakos et al., 2020).

The agricultural scenario that we briefly introduced in Section 1.3.2 is based on simulation of detections over a field under varying atmospheric conditions and with different scheduling options that we proposed in (Zamanakos et al., 2020), the energy modeling approach on some empirical observations. We proposed a periodic energy modeling in (Seewald, Garcia de Marina, Midtiby, et al., 2020) using the empirical energy data of the Opterra aerial robot flying the precision agriculture scenario.

A dynamic planning approach that we proposed in (Seewald, Garcia de Marina, and Schultz, n.d.) relies on all the concepts that we propose in this work and derives simultaneously the energy-optimized path and the power-saving schedule. Additionally to dynamic planning for aerial robots, we studied the approach on different robots in (Seewald, 2020).

1.8 Structure

The remainder of this work is illustrated in Figure ?? structured as follows. In this chapter, we introduced the dynamic energy planning for aerial robots. We define the planning problem in Chapter 2, and propose the solution to the problem in Chapter A, which builds upon the topics presented in the remaining chapters. In particular, in Chapter 3 we present the state-of-the-art in energy modeling and planning for mobile robots. We include some energy models for heterogeneous computing hardware.

Dynamic planning requires accurate future energy predictions. We present the derivation of some energy models to address future energy predictions in Chapter 4. Specifically, we propose energy models for computations, motion, and the battery. We derive and formally prove a periodic energy model that we use in the remainder of this work to predict future energy consumption. The periodic energy model's accuracy depends on the availability of measurements of energy data. We detail the process in Chapter A. We describe how to estimate the coefficients of the periodic energy model and refine the future predictions with the available data.

Flying on a dynamic plan requires the derivation of a guidance action to guide the aerial robot in space. This process is elaborated in Chapter 5. We discuss an approach to guide the aerial robot in space on the planned path. An optimal solution for the dynamic mobile robot planning problem relies on optimal control techniques. We describe such techniques also in Chapter 5. Specifically, after summarizing some of these techniques, we propose an optimization algorithm to select the optimal parameters configuration for the path and computations simultaneously.

Finally, we conclude our work in Chapter 7 along some recommendations for future research direction.

We then present some additional information in appendices. In Appendix B we propose the implementation in MATLAB (R) of the planning problem, the model, the estimation technique, the guidance action, and the derivation of the optimal control.

Chapter 2

Problem Formulation

"While we will often speak of the [coverage] problem as 'milling' with a 'cutter', many of its important applications arise in various contexts outside of machining."

— E. M. Arkin, Bender, et al., 2001

In this chapter, we discuss the planning and coverage problems that we are interested in solving. The coverage problem is the problem of finding the path that covers all the points in a given space, for instance, the agricultural field in Section 1.3. The coverage path with some user-defined computations forms the plan. The planning problem is then the problem of replanning the plan. It is replanned energy-wise in the eventuality of energy constraints dissatisfaction, and whenever the uncertainty affects the flight unexpectedly. To define both the problems, we need some basic constructs. In particular, we provide formal definitions in Sections 2.1–2.3 that include the computations, motion, computations energy, and motion energy. We then define the difference between computations and motion energy (M&CE), which we encountered in Section 1.2, path, and other plan-specific constructs. We then use all these to formulate the planning and coverage problems in Section 2.5.1. We illustrate the problem with an example of the precision agriculture scenario in Section 2.6.

In Chapter 5, we propose two algorithms. A first algorithm generates the coverage path, and another algorithm replans the plan in time, solving the coverage and planning problems. The replanning is energy-aware: the algorithm outputs the best trajectory of the path and computations alteration for an aerial robot with varying battery and atmospheric conditions.

The chapter connects to the remainder of this work as follows. Here we formalize the plan, the planning and coverage problems, and some other basic constructs. We use the plan characteristics to derive an energy model in Chapter 4, which we estimate from measured data in Chapter A. We solve the planning problem using modern optimal control techniques and the coverage problem using a planning algorithm in Chapter 5. With the solution to the coverage and planning problem, we output an initial plan, and refine the plan in terms of the path and computations.

We guide the aerial robot using a vector field, where we use the plan's building blocks from this chapter. We discuss past approaches to solve the planning problem in Chapter 3 and the concrete implementation in Appendix B.

2.1 Definitions of computations and motion

Firstly, we define computations, motion, their energy, and M&CE. Our dynamic planning depends on these basic concepts.

Definition 2.1.1: Computations/motion
Computations are energy-demanding computational tasks. The aerial robot runs the computations on heterogeneous computing hardware that interfaces microcontrollers.
Motion is the act of the aerial robot moving in the surrounding environment. The aerial robot runs some primitives on microcontrollers that interface actuators, motors, and other components.

Autonomous capabilities are often achieved by interconnecting heterogeneous computing hardware and microcontrollers. We assume for the computations that the heterogeneous computing hardware runs a schedule parametrized by some parameters. For instance, for the CNN detection from the precision agriculture scenario in Section 1.3.2, a parameter is the frames per second (FPS) rate. Alike for the computations, we assume for the motion that the robot travels some paths parametrized by some other parameters. For instance, for the coverage, a parameter changes the distance between the survey lines.

Definition 2.1.2: Computations/motion and overall energy
Given a path parametrized by ρ parameters $c_i^{\rho} := \{c_{i,1}, c_{i,2}, \ldots, c_{i,\rho}\}$, the *motion energy* is the energy spent by the aerial robot while moving on the path.
Given a schedule parametrized by parameters $c_i^{\sigma} := \{c_{i,\rho+1}, c_{i,\rho+2}, \ldots, c_{i,\rho+\sigma}\}$, the *computations energy* is the energy spent by heterogeneous computing hardware executing the schedule.
The *overall energy* is the sum of motion energy and computations energy.

Physically, motion energy is the energy spent by all the systems powering the aerial robot, excluding the heterogeneous computing hardware. We use watts for instantaneous or average measures of computations, motion, and overall energies, whereas joules for measures over a given time interval. We show in Section 2.3 what we mean by the parametrization of paths and computations with some parameters.

M&CE that we introduced in Section 1.2 is the difference between average motion and computations energy. It is measured in watts, and it gives a measure of which of the two energy components is predominant. For M&CE greater than zero, the motion energy dominates over the computations. The dynamic planning approach plans first an energy-efficient path. If we return briefly to Figure 1.5, it is the case for rotary-wing aerial robots (f) and (g). On the contrary, for M&CE lower than zero, the computations energy dominates. The dynamic planning

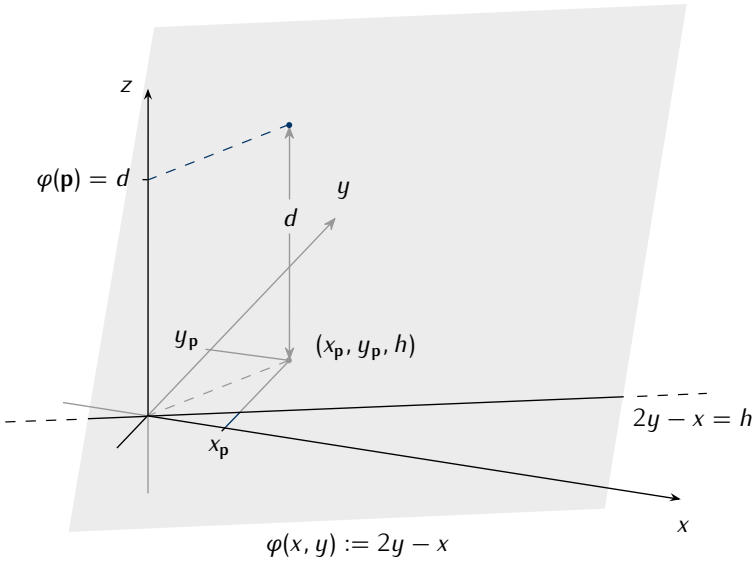


Figure 2.1. The path is mathematical function $\varphi(\mathbf{p}(t)) = h$ that represents a line at an altitude h. A generic point $\mathbf{p}$ in 2D space intersects the plane formed by φ at a given value d of the z-axis.

approach plans first a power-saving schedule. It is the case for the lighter-than-air aerial robot (a) in Figure 1.5. For M&CE close to zero, both energy components are important energy-wise. The dynamic planning approach plans both an energy-efficient path and a power-saving schedule to a similar extent. This M&CE is characteristic of fixed-wing aerial robots, such as (b)–(e) in Figure 1.5.

2.2 Definition of path functions

To model the path, we use multiple mathematical functions $\varphi_1, \varphi_2, \dots$ that we call path functions. These functions express the path in 2D space at an altitude $h \in \mathbb{R}$ for an inertial navigation frame $\mathcal{O}_W$. We discuss a generalization of the path functions in 3D space and how it affects the guidance in Chapter 7.

Definition 2.2.1: Path functions
$\varphi_i : \mathbb{R}^2 \to \mathbb{R}$, $\forall i \in \{1, 2, \dots\}$ are *path functions* used to model the path. They are a function of a generic time-dependent point $\mathbf{p}(t) := (x_{\mathbf{p}(t)}, y_{\mathbf{p}(t)})$ of the aerial robot flying in the 2D space and are continuous and twice differentiable.

We use this notion to guide the aerial robot using vector field in Chapter 5. For instance, one can define a line as a path function with

$$\varphi(x, y) := ax + by + c, \tag{2.1}$$

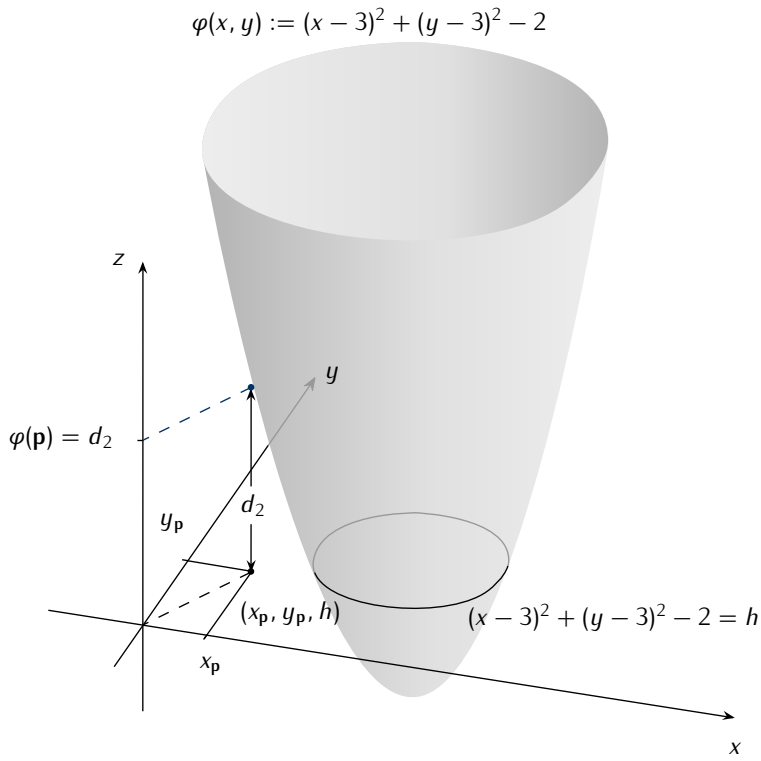


Figure 2.2. The path function now represents a circle at an altitude h. $\mathbf{p}$ intersects the cone formed by φ at a given value d_2 of the z-axis.

where $a, b, c \in \mathbb{R}$ are given constants. The generic point $\mathbf{p}(t)$ intersects $\varphi(x, y)$ on a specific value d of the z-axis $(\mathbf{p}(t), d)$. We illustrate the concept in Figure 2.1 for c zero and a, b minus one and two and h zero for simplicity. The point intersects the plane formed by the path function

$$\varphi(x, y) := 2y - x - h, \tag{2.2}$$

at $d = \varphi(\mathbf{p})$. The path that the aerial robot follows is then $\varphi(x, y) = h$. $\varphi(\mathbf{p}) - h$ is the distance on the z-axis.

Likewise with the line, one can define a circle as a path function with

$$\varphi(x, y) := (x - x_c)^2 + (y - y_c)^2 - r^2, \tag{2.3}$$

where x_c, y_c are given coordinates of the center and $r \in \mathbb{R}_{>0}$ the radius. We illustrate the circle path function in Figure 2.2 for x_c, y_c both three and $\sqrt{r}$ two and h zero for simplicity.

In this work, we use lines and circles as path functions. We connect these functions using some specific points–the triggering points. However, one can define any mathematical function, with the only requirement being continuity and twice differentiability. We use the first derivative for the vector field, the second derivative to derive the control action. We explain both derivatives further in Chapter 5.

2.3 Definitions of stages and triggering points

The plan has several stages $i = \{1, 2, \dots\}$, and we assume that at each stage i, the aerial robot runs a schedule and travels a path function φ_i using a parameters set c_i. Parameters are variable values that we use for replanning the path and computations since they influence the computations/motion energy in Definition 2.1.2. The path parameters are real-valued variable values ($\mathbb{R}$), the computations parameters are integer-valued variable values ($\mathbb{Z}$). We use the notation $c_{i,j}$ to denote the jth parameter of the ith parameters set c_i

$$c_i = \{c_{i,1}, c_{i,2}, \dots, c_{i,j}, \dots\}. \tag{2.4}$$

Parameters are bounded. $\underline{c}_{i,j}$ is the lower bound of the parameter $c_{i,j}$. $\overline{c}_{i,j}$ is the upper bound

$$\underline{c}_{i,j} \leq c_{i,j} \leq \overline{c}_{i,j}. \tag{2.5}$$

We assume there are ρ *path*-specific *parameters* and σ *computations*-specific *parameters* for every stage. It means that the path at stage i can be replanned with ρ path parameters $c_i^\rho := \{c_{i,1}, c_{i,2}, \dots, c_{i,\rho}\}$, while the computations (i.e., the energy-demanding computational task executed on heterogeneous computing hardware in Definition 2.1.1) with σ computation parameters $c_i^\sigma := \{c_{i,\rho+1}, c_{i,\rho+2}, \dots, c_{i,\rho+\sigma}\}$.

Returning to Section 2.1, by parametrization of the path with parameters, we mean that we enhance φ_i with the path parameters c_i^ρ. $\varphi_i : \mathbb{R}^2 \times \mathbb{R}^\rho \to \mathbb{R}$ is thus a (continuous twice differentiable) function of a point and the path parameters. We use the parameters to alter the path and change the energy consumption. Similarly, by parametrization of the schedule with parameters, we mean that we express the computations as the value of the computations parameters. These are also employed for energy alteration (by, e.g., decreasing the granularity of a given computation we lower instantaneous energy consumption). We discuss the alteration of the energy with path parameters and computation parameters further in Section 4.3.2 and Section 4.3.3 respectively.

Since at every stage the aerial robot travels a path and executes a schedule both parametrized by the parameters, we define the stage as a set that contains the path and path and computations parameters.

Definition 2.3.1: Stage

Given $\mathbf{p}(t)$, the ith *stage* Γ_i at time instant t of a plan Γ is

$$\Gamma_i := \{\varphi_i(\mathbf{p}(t), c_i^\rho), c_i^\sigma \mid \forall j \in [\rho]_{>0},\ c_{i,j} \in \mathcal{C}_{i,j}, \quad \forall k \in [\sigma]_{>0},\ c_{i,\rho+k} \in \mathcal{S}_{i,k}\},$$

where $\mathcal{C}_{i,j} := [\underline{c}_{i,j}, \overline{c}_{i,j}] \subseteq \mathbb{R}$ is the jth path parameter constraint set, and $\mathcal{S}_{i,k} := [\underline{c}_{i,\rho+k}, \overline{c}_{i,\rho+k}] \subseteq \mathbb{Z}_{\geq 0}$ the kth computation parameter constraint set.

We clarify in the next section why the stage contains the generic point of the aerial robot flying in 2D space.

For simplicity, we merge the computations and path constraint sets in a single constraint set. ith stage constraint set is then

$$\mathcal{U}_i(c_{i,j}) := \begin{cases} \mathcal{C}_{i,j} & \text{for } c_{i,j} \text{ with } j \leq \rho \\ \mathcal{S}_{i,j-\rho} & \text{for } c_{i,j} \text{ with } \rho < j \leq \sigma \end{cases}, \tag{2.6}$$

the stage can be thus simplified $\Gamma_i := \{\varphi_i(\mathbf{p}(t), c_i^\rho), c_i^\sigma \mid \forall j \in [\rho + \sigma]_{>0},\ c_{i,j} \in \mathcal{U}_i(c_{i,j})\}$.

To move from a given stage Γ_i to the next stage Γ_{i+1}, we define some specific points $\mathbf{p}_{\Gamma_i}$. As soon as the aerial robot reaches the proximity of these points, it switches to the next stage

$$\left\|\mathbf{p}(t) - \mathbf{p}_{\Gamma_i}\right\| < \varepsilon, \tag{2.7}$$

where $\varepsilon \in \mathbb{R}$ is a given constant value expressing the radius of an imaginary circle over the point $\mathbf{p}_{\Gamma_i}$.

Definition 2.3.2: Triggering and final point

The point $\mathbf{p}_{\Gamma_i}$ that allows the transition between Γ_i and Γ_{i+1} is called the *triggering point*. The last triggering point $\mathbf{p}_{\Gamma_l}$ relative to the last stage Γ_l is called the *final point*.

2.4 Definition of the plan

Our dynamic planning relies on the concept of the aerial robot flying a set of paths and computations autonomously. Such an autonomous flight plan often presumes a certain degree of periodicity. One can observe the periodicity in the precision agriculture example in Section 1.3.2. An ideal way to cover the agricultural field in Figure 1.6 (i.e., to visit all the points in the space) is to define a basic pattern. The aerial robot flies over the field once and iterates the basic pattern until it covers the desired area. It means that we can define the plan just as a set of stages, triggering points, a final point, and finally a shift that tells how these constructs (except the final point) shifts in space each period. To simplify the planning problem to this latter case of plans with a pattern iterated over time, we consider some primitive paths.

Definition 2.4.1: Primitive paths

Given $n \in \mathbb{Z}_{>0}$, the paths $\varphi_1, \ldots \varphi_n$ are called *primitive paths* if all the remaining paths in the plan are built from these paths with a *shift* $\mathbf{d} := (x_\mathbf{d}, y_\mathbf{d})$.

Let us assume the number of stages in the plan is known and is $l \in \mathbb{Z}$. If the plan is built from the primitive paths, it means that n in Definition 2.4.1 respects the inequation

$$n < l,\ l/n \in \mathbb{Z}. \tag{2.8}$$

This means that n is a multiplier of l, whereas l/n is the multiplicand. One can then write the remaining paths from the n primitive paths as $\varphi_{n+1}, \varphi_{n+2}, \ldots, \varphi_{n+n}, \ldots, \varphi_l$, or more generally

$\varphi_{(i-1)n+1}, \varphi_{(i-1)n+2}, \ldots, \varphi_{(i-1)n+n}$, $\forall i \in [l/n-1]_{>0}$. It can also be written formally

$$\varphi_{(i-1)n+j}(\mathbf{p}+(i-1)\mathbf{d}, c_1^{\rho}) - \varphi_{in+j}(\mathbf{p}+i\mathbf{d}, c_1^{\rho}) = e_j, \tag{2.9}$$

for a given shift $\mathbf{d}$, initial point $\mathbf{p}$, and initial value of path parameters c_1^{ρ}. The Equation (2.9) holds $\forall i \in [l/n-1]_{>0}, j \in [n]_{>0}$. $e_j \in \mathbb{R}$ is the jth constant difference.

Generally, if the plan is built from the primitive paths, it is not required to know a priori the number of stages l. The paths can be iterated up until the final point $\mathbf{p}_{\Gamma_l}$. We use the concept of primitive paths for the energy modeling in Chapter 4, where we show from collected energy data that plans built from primitive paths often have a periodic energy evolution. Alternatively to the primitive paths, one can define the plan as a mere linear succession of stages along with the triggering and final points. In the latter case, the energy can be periodic, aperiodic, or semi-periodic. Aperiodicity affects the modeling and thus future energy predictions. We discuss the concrete meaning of periodicity, aperiodicity, and semi-periodicity in the context of energy modeling in Section ??.

Formally, we define the plan as a finite state machine using the constructs of path functions in Definition 2.2.1, stages and triggering points in Definitions 2.3.1–2.3.2.

Definition 2.4.2: Plan

Given $\mathbf{p}(t)$, the *plan* is a finite state machine (FSM) Γ, where the state-transition function $s : \bigcup_i \Gamma_i \times \mathbb{R}^2 \to \bigcup_i \Gamma_i$ maps a stage and a point to the next stage

$$s(\Gamma_i, \mathbf{p}(t)) := \begin{cases} \Gamma_{i+j} & \exists j \in \mathbb{Z}, \text{ if } \left\|\mathbf{p}(t) - \mathbf{p}_{\Gamma_i}\right\| < \varepsilon \\ \Gamma_i & \text{otherwise} \end{cases}.$$

The value ε in Definition 2.4.2 is the same ε in Equation (2.7). We illustrate the definition of the plan, stage, triggering, and final points in Figures 2.3–2.5.

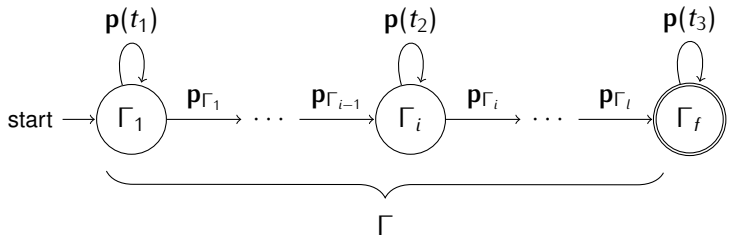


Figure 2.3. Definition of a plan Γ as an FSM. Each state is a stage Γ_i, the transition happens in the proximity of specific points called triggering points $\mathbf{p}_{\Gamma_i}$. The accepting stage Γ_f indicates the termination of the plan.

In Figure 2.3 we illustrate a plan with a linear succession of stages. The triggering point $\mathbf{p}_{\Gamma_{i-1}}$ allows the transition to the stage Γ_i. The robot remains in the stage with any generic point $\mathbf{p}(t_2)$, where $t_1 < t_2 < t_3$ are three different time instants. It eventually enters the stage Γ_{i+1} with

the triggering point $\mathbf{p}_{\Gamma_i}$ and so on, until it reaches the final point. Γ_f is the accepting stage (it indicates that the robot has completed the plan). In Figure 2.4 we illustrate that generally, one can

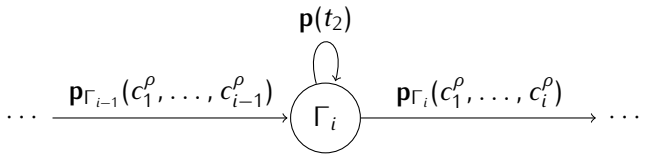


Figure 2.4. Detail of the stage Γ_i in the FSM. The triggering points used to transition between stages (states in the FSM) are expressed in the function of the last and/or previous triggering points.

express the basic constructs–such as path functions and triggering points–in the function of the ith trajectory parameters c_i^ρ, or any previous trajectory parameters, propagating the information therein if necessary. We further expand on this notion in the example in Section 2.6, where we propagate a path parameter to all the following triggering points and path functions.

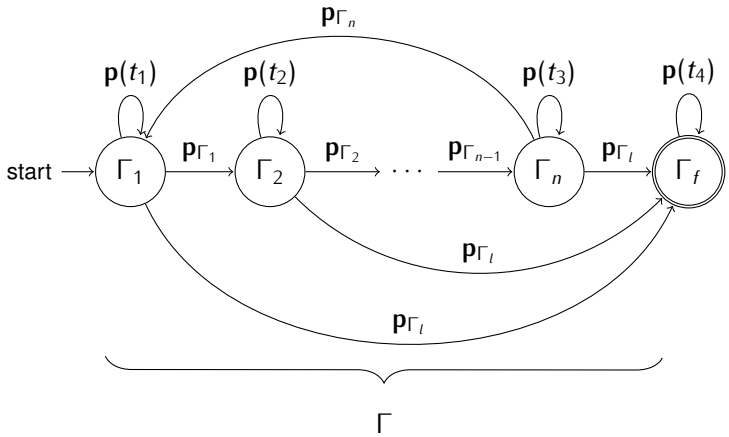


Figure 2.5. Definition of a plan Γ with periodic patterns. Stages $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ containing primitive paths $\varphi_1, \varphi_2, \ldots, \varphi_n$ are iterated with a shift $\mathbf{d}$.

In Figure 2.5 we illustrate a plan composed of n stages $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ (containing primitive paths $\varphi_1, \varphi_2, \ldots, \varphi_n$) that are reiterated with the shift $\mathbf{d}$. $t_3 < t_4$ is another time instant. We will see in Chapter 5 that the algorithm that solves the coverage problem outputs primitive paths and the corresponding plan to be replanned is equivalent to Figure 2.5.

A concept that we use in the remainder of this work, and particularly in energy modeling in Chapter 4 and estimation in Chapter A.3, is the concept of period–the time required to fly the primitive paths $\varphi_1, \varphi_2, \ldots, \varphi_n$ (or generally $\varphi_{(i-1)n+1}, \varphi_{(i-1)n+2}, \ldots, \varphi_{(i-1)n+n}$).

Definition 2.4.3: Period
For a given stage Γ_i and $j \in \mathbb{Z}_{>0}$, the *period* $T \in \mathbb{R}_{>0}$ is the flight time measured in seconds between $\varphi_{(i-1)n+j}$ and φ_{in+j}.

We assume the initial period is one and measure the period required to fly the paths physically or in simulation. The periods might be different for different js due to atmospheric interferences. For the path functions, the coverage algorithm defines the plan using primitive paths $\varphi_1, \varphi_2, \ldots, \varphi_n$ but can alternatively define all the stages explicitly and find n searching the value which satisfies the Equation (2.9). If there is no such value (e.g., when the plan is composed of only one stage or the plan is aperiodic), the period T from Definition 2.4.3 can be determined empirically from energy data or assumed as the total flight time. The latter eventuality affects the energy model. We discuss the period estimation further in Section A.3.2.

2.5 Problem Statement

We split the overall strategy for energy-aware planning and scheduling into two sub-problems. The planning problem is the problem of replanning a plan, whereas the coverage problem is the problem of defining the plan for a given space to cover.

2.5.1 Planning problem

More precisely, the planning problem is then the problem of finding the optimal configurations of the parameters within some criteria. In our approach, we focus on energy criteria, such as the battery constraints. In particular, in the solution to the planning problem in Section 5.3.3, we use a cost function (i.e., the function to maximize) that incorporates the energy model in Section ??. Solving the planning problem by finding the optimal configurations within different criteria, such as shortest time, highest security, path tracking with the shortest detour, or others is equally possible. In the context of the TeamPlay project that funded a considerable part of this work, we aim to find, for instance, the tradeoffs between time, energy, and security criteria for a variety of use cases. We discuss the eventuality of solving the planning problem with criteria different from energy in Chapter 7. We now use the concepts that we introduced in the previous sections and provide a formal definition of the planning problem that we are interested in solving in the remainder of this work.

Problem 2.5.1: Planning problem
Consider an initial plan Γ in Definition 2.3.1. It is either composed of l stages or n stages and a shift $\mathbf{d}$. We are interested in the *planning* of the *path* and *computations parameters* c_i, $\forall i \in [l]_{>0}$, or $\forall i \in \{1, 2, \ldots\}$ under energy constraints and uncertainty.
We are further interested in the guidance to the path and the scheduling of the computations resulting from such planning.

In the definition, there is a fixed or variable number of stages, in the sense that all the stages are reiterated using the primitive paths in Definition 2.4.1 up until the aerial robot reaches the last triggering point $\mathbf{p}_l$.

2.5.2 Coverage problem

Given a polygon representing the space to be covered, some possible obstacles within the polygon, a starting point, and a turning radius, we want to find the route that covers the polygon. This problem is to be found in the robotics research literature as coverage path planning (CPP) (H. Choset, 2001; H. Choset and Pignon, 1998; Galceran and Carreras, 2013). There are numerous approaches to solve CPP that ensure the completeness of the coverage and include algorithms optimized for mobile robots. We discuss the state of the art in CPP in detail in Chapter 3. We assume that the free space where the robot moves can be summarized by a set of vertices while the obstacles by another set of vertices. The planning problem is then the problem of covering the free space without covering the obstacles. Physically, the obstacles space can be seen as areas where, e.g., the aerial robot shouldn't detect hazards in the agricultural scenario in Section 1.3.2.

Problem 2.5.2: Coverage problem

Consider a finite set of vertices of a polygon to be covered $v := \{v_1, v_2, \dots\}$ and of the obstacles $o := \{o_1 := \{o_{1,1}, o_{1,2}, \dots\}, o_2 := \{o_{2,1}, o_{2,2}, \dots\}, \dots\}$ where each vertex $v_i := (x_{v_i}, y_{v_i})$, $o_{j,k} := (x_{o_{j,k}}, y_{o_{j,k}})$, $\forall i \in |v|$, $\forall j \in |o|, k \in |o_j|$ is a point w.r.t. $\mathcal{O}_W$ and given minimum turning radius $r \in \mathbb{R}_{>0}$ of the aerial robot flying and a starting point $\mathbf{p}(t_0)$ at time t_0.

We are interesting in *finding a plan* Γ that covers v while avoiding o starting from $\mathbf{p}(t_0)$ with a turn radius greater or equal than $\underline{r} \in \mathbb{R}_{\geq 0}$ and a final triggering point $\mathbf{p}_{\Gamma_l}$.

The coverage plan is a solution to the coverage problem and with the user-defined computations an input to the planning problem.

2.6 Precision agriculture example

In this section, we discuss an example of a plan for the Opterra autonomous aerial robot in a precision agriculture scenario addressing the coverage problem in Figure 1.6. Path-wise, the aerial robot covers a polygon with variable quality of coverage. Computation-wise, it detects ground hazards and communicates eventual detection with other ground-based actors. To simplify the notation, we split the plan into two sub-plans, one containing exclusively the paths and the other the computations. We discuss the paths sub-plan in Section 2.6.1 and the computations sub-plan in Section 2.6.2.

2.6.1 Paths sub-plan

In the paths sub-plan, we define the paths that form the plan of the agricultural scenario. We recall that in the scenario, the aerial robot covers the polygon in Figure 1.6. For simplicity, we assume the polygon has four sides with four vertices $v := \{v_1, \ldots, v_4\}$ (it is a rectangle) and has no obstacles $o := \{\emptyset\}$. We will later propose in Chapter 5 an approach that can deal with an arbitrary number of polygon and obstacles vertices. An intuitive static plan Γ can be composed of lines connected by circles. The distance between the lines can be then used as a measure of the quality of the coverage. Such intuitive static plan is illustrated in Figure 2.6.

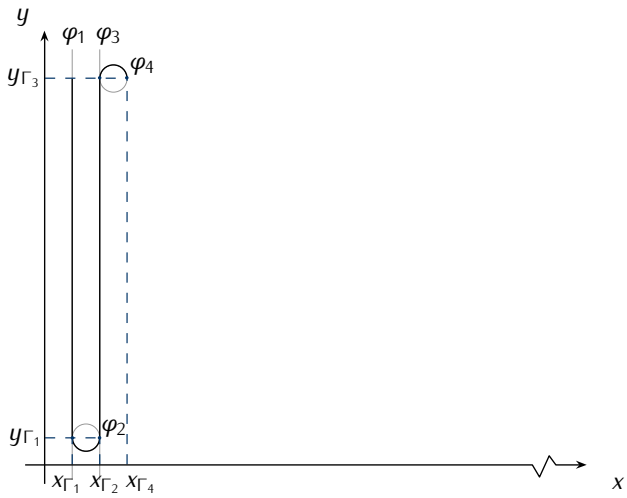


Figure 2.6. An intuitive plan to cover a regular polygon with four sides. The plan is composed of circles φ_2, φ_4 and lines φ_1, φ_3. To switch between paths the aerial robot reaches the proximity of triggering points ($\mathbf{p}_{\Gamma_1} := (x_{\Gamma_1}, y_{\Gamma_1})$, $\mathbf{p}_{\Gamma_2}, \mathbf{p}_{\Gamma_3}, \mathbf{p}_{\Gamma_4}$). The dashed blue line indicates the triggering points, and the black line is the planned flight. The rest of the polygon is covered iterating the primitive paths and triggering points with a shift.

We note that in the literature, the pattern in Figure 2.6 is similar to boustrophedon motion (H. Choset, 2001; H. M. Choset et al., 2005; LaValle, 2006), but for the circles that are straight lines parallel to the segments connecting the corresponding vertices. We can use the intuitive plan to solve the coverage problem with a rotary-wing aerial robot; in fact, the boustrophedon motion is abundant in aerial robotics literature relative to CPP (Araújo et al., 2013; Artemenko et al., 2016; Cabreira, Franco, et al., 2018; Di Franco and Buttazzo, 2015). However, it is unsuitable for a fixed-wing aerial robot such as the Opterra. Fixed-wing aerial robots have considerable nonholonomic constraints and overall reduced maneuverability compared to rotary-wings (Dille and Singh, 2013; Mannadiar and Rekleitis, 2010; Xu et al., 2011, 2014). They are generally unable to perform quick turns in flight (X. Wang et al., 2017). We illustrate an updated version of the intuitive plan for fixed-wing aerial robots in Figure 2.7. This latter variation of the coverage is similar to Zamboni motion in the literature (Araújo et al., 2013). We term the plans in Figure 2.6 and Figure 2.7 boustrophedon-like and Zamboni-like motions because they are similar to the

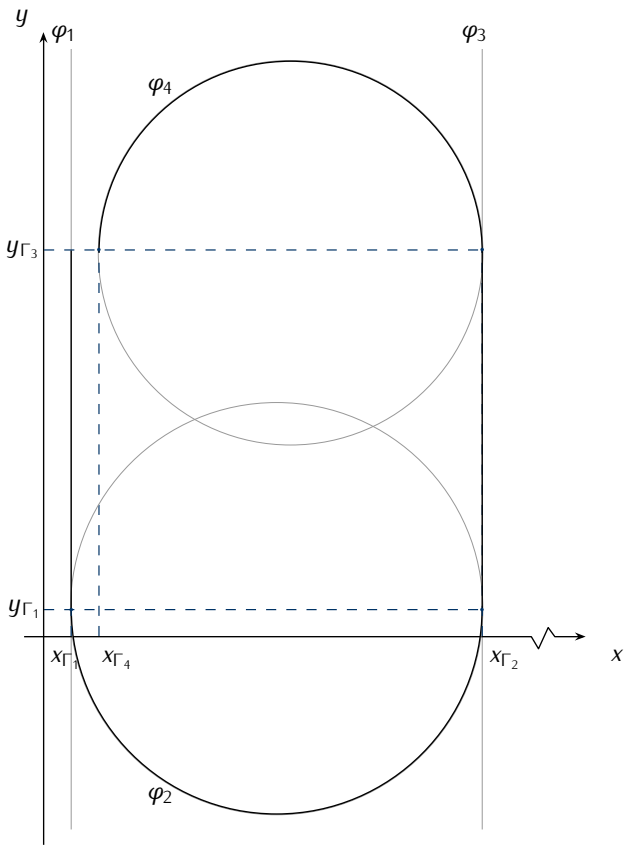


Figure 2.7. Fixed-wing aerial robot plan to cover a regular polygon with four sides. The plan covers the polygon with the same principle of the intuitive plan in Figure 2.6 but preserving long turns necessary for the flight of a fixed-wing aerial robot. Likewise in Figure 2.6, the entire polygon is covered iterating the primitive paths (gray lines) and triggering points. The dashed blue line indicates the triggering points, and the black line is the planned flight. The height and length of the polygon are $y_{\Gamma_3} - y_{\Gamma_1}$ and $lx_{\mathbf{d}}/4$.

robotics literature but optimized to our use-case and potentially a broad class of aerial robots with turning constraints. Indeed these constraints are commonly treated in the aerial robotics literature (Artemenko et al., 2016; Y. Li et al., 2011; Xu et al., 2011, 2014). We discuss further the boustrophedon, Zamboni, and other motions for the coverage in Chapter 3 and include them in our coverage planning in Chapter 5.

The Zamboni-like motion in Figure 2.7 is composed of four primitive paths

$$\varphi_1(\mathbf{p}(t)) := x - 10, \tag{2.10a}$$
$$\varphi_2(\mathbf{p}(t)) := (x - 85)^2 + (y - 10)^2 - 5625, \tag{2.10b}$$
$$\varphi_3(\mathbf{p}(t)) := x - 160, \tag{2.10c}$$
$$\varphi_4(\mathbf{p}(t)) := (x - 90)^2 + (y - 140)^2 - 4900, \tag{2.10d}$$

where x, ys are the x- and y-coordinates of a generic point $\mathbf{p}(t)$. The triggering points (the points in which proximity occurs the change of stages) are then the points

$$\mathbf{p}_{\Gamma_1} := (10, 10),\ \mathbf{p}_{\Gamma_2} := (160, 10),\ \mathbf{p}_{\Gamma_3} := (160, 140),\ \mathbf{p}_{\Gamma_4} := (20, 140). \tag{2.11}$$

The coverage problem can be solved using the paths in Equation (2.10), the triggering points in Equation (2.11), the remaining paths $\varphi_5, \varphi_6, \ldots, \varphi_l$, and triggering points $\mathbf{p}_{\Gamma_5}, \mathbf{p}_{\Gamma_6}, \ldots, \mathbf{p}_{\Gamma_l}$ defined similarly to Equations (2.10–2.11). A generic solution for the coverage problem defined as a pattern iterated over time is then defined with the primitive paths

$$\varphi_i(\mathbf{p}(t)) := x - x_{\Gamma_1} - \lfloor i/4 \rfloor x_{\mathbf{d}}, \tag{2.12a}$$
$$\varphi_{i+2}(\mathbf{p}(t)) := x - x_{\Gamma_2} - \lfloor i/4 \rfloor x_{\mathbf{d}}, \tag{2.12b}$$

$\forall i \in \{1, 5, 9, \ldots\}$. $x_{\mathbf{d}}$ is a shift on the x-axis, $\lfloor i/4 \rfloor$ is the integer division. The expressions in Equation (2.12) correspond to the generalizations of the lines in Equation (2.10a) and Equation (2.10c). The generalizations of the circles in Equation (2.10b) and Equation (2.10d) are

$$\varphi_{i+1}(\mathbf{p}(t)) := (x - x_{\Gamma_1} - r_1 - \lfloor i/4 \rfloor x_{\mathbf{d}})^2 + (y - y_{\Gamma_1} - \lfloor i/4 \rfloor y_{\mathbf{d}})^2 - r_1^2, \tag{2.13a}$$
$$\varphi_{i+3}(\mathbf{p}(t)) := (x - x_{\Gamma_2} + r_2 - \lfloor i/4 \rfloor x_{\mathbf{d}})^2 + (y - y_{\Gamma_3} - \lfloor i/4 \rfloor y_{\mathbf{d}})^2 - r_2^2, \tag{2.13b}$$

where index i is defined the same way as in Equation (2.12) along the shift (additionally, $y_{\mathbf{d}}$ is a shift on the y-axis) and integer division. $r_1 > r_2 > \underline{r}$ are given radiuses of the circles φ_2 and φ_4 in Figure 2.7 and $\underline{r}$ is the turning radius in Definition 2.5.2.

The triggering points can be expressed similarly with the expressions

$$\mathbf{p}_{\Gamma_i} := (x_{\Gamma_1} + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.14a}$$
$$\mathbf{p}_{\Gamma_{i+1}} := (x_{\Gamma_1} + 2r_1 + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.14b}$$
$$\mathbf{p}_{\Gamma_{i+2}} := (x_{\Gamma_1} + 2r_1 + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.14c}$$
$$\mathbf{p}_{\Gamma_{i+3}} := (x_{\Gamma_1} + 2r_1 - 2r_2 + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}). \tag{2.14d}$$

We note from Figure 2.7 and Equations (2.12–2.14) that $y_{\Gamma_3} - y_{\Gamma_1}$ is the height of the polygon to be covered, $2(r_1 - r_2)$ the distance between the covering lines (which is equal to the shift $x_{\mathbf{d}}$), and $l x_{\mathbf{d}}/4$ is the length of the polygon if the number of stages l is known.

The plan in Equations (2.12–2.14) is static. There is no path parameter that allows to alter the plan. We can easily transform such plan with the addition of a path parameter $c_{4,1}$ relative to the radius of the second circle φ_4 in Figure 2.7

$$r_2(c_{4,1}) := (r + c_{4,1}), \tag{2.15}$$

where $\underline{r} < r < r_1$ is a given positive constant initial radius and $c_{4,1} \in (\underline{r} - r, 0]$. We assume that the highest value is thus $\overline{c}_{4,1} = 0$, the lowest is strictly higher than the difference between the turning and constant initial radiuses $\underline{c}_{4,1} > \underline{r} - r$.

We note from the expression in Equation (2.15) that Equation (2.14d) and Equation (2.13b) depend on the parameter $c_{4,1}$ (they contain r_2, which depend on $c_{4,1}$). They can be thereby

dynamically replanned, resulting in an alteration of the quality of the coverage. They indeed change the distance between the survey lines. We can bound the alteration with

$$c_{i,1} \in [\underline{c}_{4,1}, \overline{c}_{4,1}] =: \mathcal{C}_{4,1} = (\underline{r} - r, 0],\ \forall i, \tag{2.16}$$

where for simplicity, we assume that we can change the parameter in advance at any stage (thus we use $\forall i$ in the equation).

The concept is illustrated in Figure 2.8. The black line is the un-altered path until the triggering point $\mathbf{p}_{\Gamma_3}$, where the path splits depending on the value of the path parameter $c_{4,1}$. The alteration of the plan shortens or extends the flying time and thus influence the energy consumption over time. Since the last path φ_4 is a function of the parameter $c_{4,1}$, the correct expression for Equation (2.13b) is

$$\begin{aligned}\varphi_{i+3}(\mathbf{p}(t), c_{4,1}) := (x - x_{\Gamma_2} + r_2(c_{4,1}) - \lfloor i/4 \rfloor x_{\mathbf{d}})^2 + \\ (y - y_{\Gamma_3} - \lfloor i/4 \rfloor y_{\mathbf{d}})^2 - r_2(c_{4,1})^2.\end{aligned} \tag{2.17}$$

The last triggering point $\mathbf{p}_{\Gamma_4}$ of the example in Equation (2.14d) is likewise a function of the parameter $c_{4,1}$

$$\mathbf{p}_{\Gamma_{i+3}}(c_{4,1}) := (x_{\Gamma_1} + 2r_1 - 2r_2(c_{4,1}) + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.18}$$

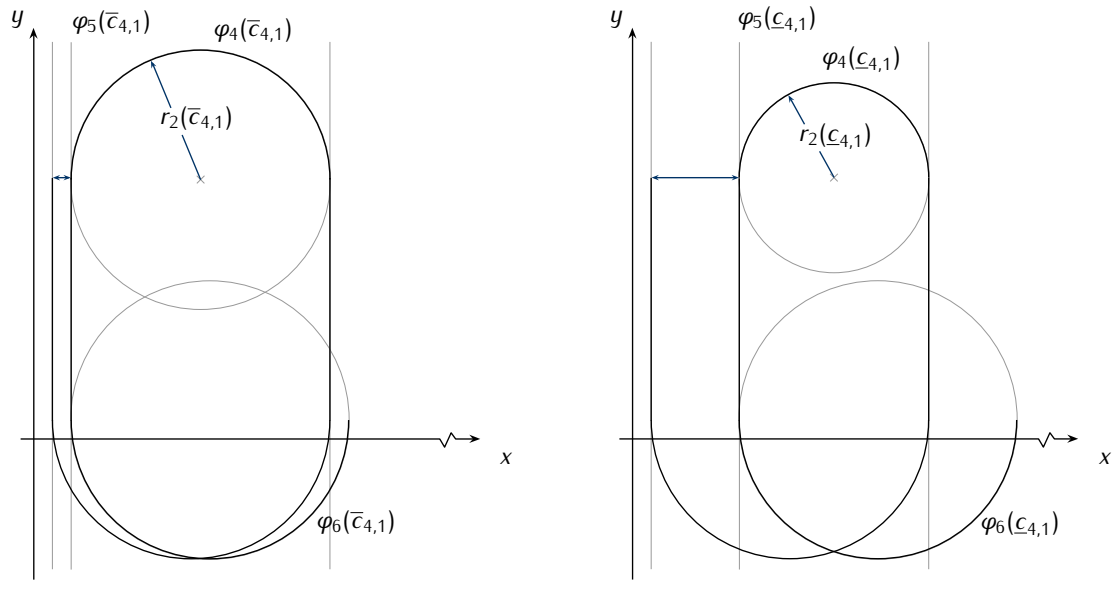
as it depends on the radius r_2 of the circle φ_4. The same applies to all the following functions φ_5, φ_6, φ_7, since these are all a function of the triggering point $\mathbf{p}_{\Gamma_4}$. The path φ_8 is then a function of the parameter $c_{4,1}$ and $c_{8,1}$ propagating the parameters for all the following paths φ_9, φ_{10}, φ_{11} and so on.

We discuss further the coverage with the Zamboni-like motion in Section 5.2.

2.6.2 Computations sub-plan

In the computations sub-plan, we define the computations that form the plan of the agricultural scenario, opposed to the paths defined in the previous section. In the scenario, we are interested in monitoring the field in Figure 1.6. We detect ground hazards and communicate with other ground-based actors. We use a CNN implemented in a ROS node to detect the ground hazards. The CNN detection uses image frames from a downward-facing camera mounted on the aerial robot. We assume that there is one centralized workstation on the ground to communicate with the ground-based actors. The communication between the aerial robot and the centralized work station occurs on another ROS node, which uses the technical standard for wireless communication IEEE 802.11 (Crow et al., 1997). It sends the detected images either unencrypted or using public-key infrastructure for encryption. We refer to these two computations as CNN and encryption ROS nodes. The schedule for the CNN ROS node is parametrized by the FPS rate at which the frames are sent from the camera. The schedule for the encryption ROS node is parametrized by a binary value that indicates whenever the encryption is enabled.

There are thus two computations parameters. A computation parameter is the FPS rate, and another computation parameter the encryption binary value. The CNN ROS node's computation parameter is $c_{i,2}$, and the encryption ROS node's computation parameter is $c_{i,3}$.



(a) original path

(b) replanned path

Figure 2.8. Alteration of a path parameter of the fixed-wing aerial robot's plan in Figure 2.7. The black line is the un-altered path up to the triggering point $\mathbf{p}_{\Gamma_3}$, where the path can be altered with the parameter $c_{4,1}$ relative to the radius r_2 of φ_4. The alteration changes the distance between the survey lines hence extending or shortening the flying time. The longest distance is then $2(r_1 - r_2(\overline{c}_{4,1}))$, the shortest $2(r_1 - r_2(\underline{c}_{4,1}))$. Same principle applies to path parameter $c_{8,1}$ for the next two survey lines, $c_{12,1}$, and so on.

The upper and lower bounds of $c_{i,3}$ are

$$c_{i,3} \in \mathcal{S}_{i,3} := \{0, 1\},\ \forall i, \tag{2.19}$$

the parameter value thus indicates if the encryption is active (one) or data are sent unencrypted (zero).

For the upper and lower bound of $c_{i,2}$, we first note from the path plan in Section 2.6.1 that during the turns the aerial robot is not surveying the polygon. We thus place different constraints for different paths. For the circles in Equation (2.22b) and Equation (2.22d) the computation parameters constraint sets are

$$c_{i+1,2}, c_{i+3,2} \in \mathcal{S}_{i+1,2} = \mathcal{S}_{i+3,2} := \{0\},\ \forall i \in \{1, 5, 9, \dots\}, \tag{2.20}$$

and thus the CNN ROS node is not detecting any hazard when it flies out of the polygon. On the contrary, for the lines in Equation (2.22a) and Equation (2.22c)

$$c_{i,2}, c_{i+2,2} \in \mathcal{S}_{i,2} = \mathcal{S}_{i+2,2} := [2, 10],\ \forall i \in \{1, 5, 9, \dots\}, \tag{2.21}$$

the the CNN ROS node is detecting hazard on the ground with a FPS rate between two and ten.

Energy-wise we expect the highest configuration of computations parameters to correspond to the highest instantaneous energy consumption. We use `powprofiler`, the profiling and modeling tool that we briefly outlined in Section 1.4 and that we introduce in Section 4.1, to measure and model the future energy consumption of different computations' configurations.

2.6.3 Planning problem with paths and computations sub-plans

The plan for the precision agriculture scenario is composed of stages containing the primitive paths in Equations (2.12–2.13), and the parameter dependent version in Equation (2.17). It further contains the path parameter $c_{i,1},\ \forall i \in \{4, 8, \dots\}$, the computations parameters $c_{i,2}$ and $c_{i,3},\ \forall i \in \{1, 2, \dots\}$. In particular, the stages corresponding to the primitive paths are

$$\Gamma_i := \{\varphi_i(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}), c_{i,2}, c_{i,3}\}, \tag{2.22a}$$
$$\Gamma_{i+1} := \{\varphi_{i+1}(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}), c_{i+1,2}, c_{i+1,3}\}, \tag{2.22b}$$
$$\Gamma_{i+2} := \{\varphi_{i+2}(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}), c_{i+2,2}, c_{i+2,3}\}, \tag{2.22c}$$
$$\Gamma_{i+3} := \{\varphi_{i+3}(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}, c_{i+3,1}), c_{i+3,2}, c_{i+3,3}\}, \tag{2.22d}$$

$\forall i \in \{1, 5, 9, \dots\}$. The path constraint set for the path parameter $c_{i,1}$ is then defined in Equation (2.16), the computation constraint set for the computation parameter $c_{i,2}$ in Equation (2.21)

and for the computation parameter $c_{i,3}$ in Equation (2.19). The triggering points

$$\mathbf{p}_{\Gamma_i}(c_{4,1},\dots,c_{i-1,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.23a}$$

$$\mathbf{p}_{\Gamma_{i+1}}(c_{4,1},\dots,c_{i-1,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + 2r_1 + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.23b}$$

$$\mathbf{p}_{\Gamma_{i+2}}(c_{4,1},\dots,c_{i-1,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + 2r_1 + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.23c}$$

$$\mathbf{p}_{\Gamma_{i+3}}(c_{4,1},\dots,c_{i+3,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + 2r_1 - 2r_2(c_{i+3,1}) + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}). \tag{2.23d}$$

$\forall i \in \{5, 9, \dots\}$. The initial points x_{Γ_1}, y_{Γ_1}, and y_{Γ_3} are given along the shift $\mathbf{d} = (x_{\mathbf{d}}, y_{\mathbf{d}})$, the radius of the first circle r_1, and the last triggering point $\mathbf{p}_l$. The function r_2 which returns the radius of the second circle and it is a function of the path parameter $c_{i,1}$ is in Equation (2.15) contains r which is likewise given (we can estimate it from the desired distance of the covering lines: $r = r_1 - x_{\mathbf{d}}/2$).

The solution to the planning problem are thus the three trajectories $c_i^*(t) = \{c_{i,1}^*(t), c_{i,2}^*(t), c_{i,3}^*(t)\}$, $\forall i \in \{1, 2, \dots\}$ that are energy-aware under given battery budget and uncertainty. Since each quadruple of stages in Equation (2.22) and triggering points in Equation (2.23) depends only on the last path parameter (of the quadruple), we further assume that $c_{1,1} = c_{1,2} = c_{1,3} = c_{1,4}$ and more generally

$$c_{i,1} = c_{i+1,1} = c_{i+2,1} = c_{i+3,1},\ \forall i \in \{1, 5, 9, \dots\}. \tag{2.24}$$

We derive the solution for the planning problem of the precision agriculture example in Section 5.3.1.

2.7 Summary

In this chapter, we defined the planning and coverage problems in Section 2.5.1. The solution to the coverage problem is a static cover tour, to the planning problem energy-aware replanning. Both problems are interconnected and require some basic concepts, including path functions in Section 2.2 and plan in Section 2.4. The latter is composed of stages, triggering, and final points in Section 2.3, and parameters for the replanning itself. Some parameters parametrize the path, and some parameters parametrize the computations. Alternatively to defining all the stages manually, it is possible to define an autonomous scenario with a certain degree of periodicity using primitive paths and a shift as we saw in Section 2.4. We illustrated the concept with the precision agriculture autonomous scenario, first introduced in Section 1.3.2, in Section 2.6. We discuss later in Chapter 4 the physical intuition behind the degree of periodicity, with some empirical data proving a periodic energy evolution and thus helping us to define a model for future energy estimations.

Chapter 3

State of the Art

> *"Even though [dynamic voltage scaling] and energy conservation for mobile robots have been studied, the close interaction between computation and motion remains unexplored."*
>
> — Brateman et al., 2006

In this chapter, we discuss the state of the art in energy modeling and optimization of computing hardware and mobile robots and in planning for mobile robots. There is a considerable body of knowledge in energy modeling for computing hardware, especially if battery-powered (V. Rao et al., 2005). When such hardware is onboard a mobile robot, studies on energy efficiency often focus on the energy for the motion on e.g., a planned path, and neglect the computing hardware (Ondrúška et al., 2015). Moreover, planning algorithms in robotics literature center around robot motion planning and deal with problems such as swarms, dynamics, and uncertainty (LaValle, 2006). By illustrating several contributions applied to a variety of robots, we primarily focus on the approaches that, similarly to ours, plans the path and schedules the computations. We especially emphasize studies applied to mobile robots with energy constraints.

We split the chapter into multiple sections and replicate our workflow throughout the topic. Initially, we analyzed some contributions that quantify the energy consumption of computing hardware carried by mobile robots. Modeling the energy of these devices has laid the foundations for our energy-aware planning. In Section 3.1, we report our findings spanning broadly to generic computations energy modeling. We discuss the available approaches for battery modeling and the battery modeling and optimization of the mobile computing hardware in Section 3.2. We then discuss studies on motion planning for mobile robots generically and planning for aerial robots in Sections 3.3–3.4. For both, we detail approaches in coverage path planning and the derivation of optimal coverage. Although simultaneous planning of computations and motion remains mostly unexplored (Brateman et al., 2006; Ondrúška et al., 2015; Sudhakar et al., 2020),

some studies have proposed various techniques and further motivated our analysis. We report these in detail in Section 3.5.

This chapter connects to the remainder of this work as follows. Here we discuss the state of the art on topics that allowed us to derive a coverage planning and scheduling approach for autonomous aerial robots. Based on these findings, we later propose a computations energy modeling technique to derive future energy consumption of mobile computing hardware along with motion in Chapter 4. Planning in the context of mobile robots, for aerial robots, of the coverage, and computations and motion, are the basis for the derivation of the optimal configuration in Chapter 5. We use such configuration to replan an initial plan; to solve the planning and coverage problems that we defined in Chapter 2. Chapter A then contains some further discussion of known principles in optimal control and state estimation that we extensively use in Chapter 5.

3.1 Computations Energy Modeling

There are several different energy modeling and optimization approaches for computations, usually under the topic of energy efficiency for computing hardware. Generally, energy efficiency is critical for battery-constrained devices (V. Rao et al., 2005) and a limiting factor in improving further computing performance (Horowitz, 2014). Modern computing hardware–carried by aerial robots that we analyze in this work–is often composed of heterogeneous elements: one or more CPUs and a GPU as we outlined in Section 1.3. We split some of the available approaches in the literature into different classes, depending on their modeling and optimization approach. Due to the unpredictable nature of the heterogeneous elements, many contributions to energy modeling observe hardware characteristics and perform physical energy consumption measurements to derive an energy model. We analyze some of these contributions first. They treat the heterogeneous elements altogether and are of particular interest to our approach where we use heterogeneous mobile computing hardware. We then analyze the techniques that focus on the energy of GPUs. Finally, we analyze techniques that treat CPUs. Some of the latter are based on dynamic voltage scaling (DVS) and dynamic frequency scaling (DFS) that scales down the supply voltage and the frequency when there is no high computations demand (X. Chen and Touba, 2009; Flautner et al., 2001). Most of the studies we analyze include an energy model that is either an analytical expression (in the function of some architectural parameters of the computing hardware) or based on regression. The studies then provide an energy optimizing technique derived from the modeled energy. This latter is usually either the derivation of a selection of hardware parameters or a configuration of some computations parameters. We illustrate an overview of the studies we consider in Table 3.1, where we distinguish approaches by the heterogeneous elements, the model, and the technique they employ. We further state if a given study use DVS and/or DFS, summarize the accuracy, and report the experimental computing hardware when available (mobile and non).

There are some reviews available for computations energy modeling literature: O'Neal and Brisk (O'Neal and Brisk, 2018) summarize techniques with different computing elements and focus on predictive modeling emphasizing heterogeneous models based on machine learning.

O'Brien et al. (O'Brien et al., 2017) and Czarnul et al. (Czarnul et al., 2019) review energy modeling focusing on high-performance computing (HPC); Czarnul et al. report existing tools for energy prediction in HPC systems and O'Brien et al. the accuracy of different models in the literature.

3.1.1 Heterogeneous elements modeling

Modern computing hardware energy modeling and optimization techniques often deal with the heterogeneous computing elements altogether using statistical tools. Such tools are inexpensive to deploy and relatively accurate in predicting the future energy consumption of computations (Bailey et al., 2014). Although there are further optimizations available by looking at the elements (CPUs, GPUs) separately, these are often application and hardware-dependent. Instead, we focus on a generic computations energy model that can be used independently of the hardware and computations under analysis.

Marowka derives a model (Marowka, 2017) for heterogeneous elements. The model uses some power metrics: the scaled speedup, scaled performance per watt, and scaled performance per joule. The approach increases energy efficiency by choosing the configuration of the heterogeneous components–for instance, by enabling computations only on CPU cores–and hence, investigates the impact of different architectural choices on energy efficiency. In particular, the model is used to analyze the energy of three processing schemes: symmetric, asymmetric, and simultaneous. The former uses merely a multicore CPU. Asymmetric processing scheme both CPU and GPU. The software benchmark on this scheme consists of running a program on one of the computing elements at a time. The latter processing scheme is similar to the previous, but the software benchmark runs on CPU and GPU simultaneously. Our work extends the model and builds a computations model to the simultaneous processing scheme.

Bailey et al. derive a more sophisticated statistical model (Bailey et al., 2014) relying on multivariate linear regression for heterogeneous elements. The model eases the selection of the application's configuration that maximizes performance under given power constraints. It is trained offline with a small set of benchmarks and works across multiple devices. The overall flow of the proposed approach is composed of two stages. The first stage is the offline training stage that utilizes a small training set of benchmarks split into clusters. The model is built then with a regression per each cluster. After this stage, follows the online predicting stage. The latter uses the model to predict the power and performance of a large set of applications, opposed to the relatively small number of benchmarks in the offline stage. Our work similarly models a subset of samples and infers properties of the entire search space. Opposed to Bailey et al., our work further focuses on mobile computing hardware.

Goraczko et al. develop a resource model (Goraczko et al., 2008) for heterogeneous mobile devices and consider both the time and power of a given run-time mode. Goraczko et al. use the term mode rather than configuration used by Marowka and Bailey et al. and use a CPU and a microcontroller as a heterogeneous platform instead of a CPU and a GPU. Energy-wise, the resource model uses DVS as some other models in Section 3.1.3 but accounts for heterogeneous elements. The approach models multiple processors with a state machine involved in a software

		Model	Technique	Accuracy	DVS	DFS	Platform	Mobile
Heterogeneous	Marowka	Analytical	Selection	-	✗	✗	Intel Core-i7-960 (CPU), NVIDIA GTX 280 (GPU)	✗
	Bailey et al.	Regression	Configuration	91%[a]	✗	✗	AMD A10-5800K (CPU), Radeon HD7660D (GPU)	✗
	Goraczko et al.	Analytical	Configuration[b]	-	(✓)	(✓)	ARM7 OKI ML675003 (CPU), TI MSP430F1611 (microcontroller)	✓
	Ma et al.	-	Configuration	-	✓	✓	AMD Phenom II X2 (CPU), NVIDIA GeForce 8800 (GPU)	✗
	Calore et al.	Analytical	Selection	-	✗	[✓]	ARM Cortex-A15 (CPU), NVIDIA GK20A (GPU)	✓
	Yang et al.	Analytical	Pruning	-	✗	✗	-	✗
GPU	Hong and Kim	Analytical	Selection	91.06%[c]	✗	✗	NVIDIA GTX280 (GPU)	✗
	Wu et al.	Regression (ML)	Selection	90%[d]	✗	[✓]	AMD Radeon HD 7970 (GPU)	✗
	Collange et al.	-	Selection	-	✗	✗	NVIDIA G80, G92, GT200 (GPU)	✗
	Luo and Suda	Analytical	-	88.87%[d]	✗	✗	NVIDIA Tesla C2050 (GPU)	✗
	Leng et al.	Analytical	Selection	88.35%[d]	✓	✓	NVIDIA GTX 480, Quadro FX5600 (GPU)	✗
CPU	Lee and Brooks	Regression	Selection	95.7%[e]	✗	✗	-	✗
	Takouna et al.	Regression	Selection	93%[f]	✗	✓	Intel Xeon E5540 (CPU)	✗
	Reddy et al.	Analytical	Selection	94%[g, d]	✓	✓	ARM Cortex-A15 (CPU)	✓
	Nikov et al.	Regression	Selection	92–95%[b]	✓	✓	ARM Cortex-A15, Cortex-A7 (CPU)	✓
	Nunez-Yanez and Lore	Regression	Selection	95%	✗	✗	ARM Cortex-A9 (CPU)	✓
	Walker et al.	Regression	Selection	96.2–97.2%	✓	✓	ARM Cortex-A15, Cortex-A7 (CPU)	✓

Table 3.1. Comparison of different computations energy models: the model is either an analytical expression or a regression. The energy optimization technique is the selection of some architectural parameters or computations configurations. Scaling is split into DVS and DFS: (✓) scaling is used only in the model, not in the optimization technique. [✓] values are changed statically (or manually where appropriate such as in Marowka).

[a] accuracy under-limit against oracle with perfect knowledge
[b] optimization problem solved with ILP
[c] average accuracy for both power and time models against measuring hardware
[d] accuracy against built-in power monitors (if the accuracy is reported for multiple hardware or benchmarks, it is average)
[e] accuracy against median error, leveraging only the most relevant samples
[f] accuracy of 95% of the predictions
[g] accuracy on 60 workloads
[b] accuracy against related work

partitioning problem–the problem of deriving the optimal mode solved with an optimization technique: integer linear programming (ILP). The optimization occurs over an energy cost and with given deadline constraints. The intuition of formally defining the problem of deriving the optimal mode is similar in our work, expanded further by merging the path in our energy-aware planning and scheduling.

Ma et al. propose a holistic approach (Ma et al., 2012) for heterogeneous elements that achieves energy efficiency by splitting and distributing the workload among the heterogeneous elements. Ma et al. then use frequency scaling for the CPU and GPU and DVS for the CPU. GPU-side, the frequency is determined with a lightweight machine learning algorithm. Energy in this approach is analyzed by empirical means, using two power meters. The testbed under analysis–NVIDIA GeForce GPUs and AMD Phenom II CPUs–does not include built-in power monitors, and overall, Ma et al. do not consider mobile computing heterogeneous hardware nor derive a model for the energy. Nevertheless, the approach is of interest for energy implications of heterogeneous elements.

Our initial approach (Seewald, Ebeid, et al., 2019) relied on external meters rather than internal power monitors. Calore et al. developed a similar technique (Calore et al., 2015) to measure the energy efficiency of HPC systems. Both our initial and Calore et al.approaches are tested on the NVIDIA Jetson TK1 computing hardware that does not include built-in power monitors, opposed to other computing hardware that we analyze.

Recently, approaches are emerging to model the energy consumption of machine learning algorithms. Yang et al. developed a computations-specific modeling approach (Yang, Y.-H. Chen, Emer, et al., 2017) in this context, while García-Martín et al. surveyed several studies (García-Martín et al., 2019), motivated by the lack of appropriate tools to build and measure power models in existing machine learning suites. García-Martín et al. describe the state of the art of energy estimation for convolutional neural networks (CNNs) and data mining, whereas Yang et al. evaluate an energy model of deep neural networks (DNNs) based on a network bitwidth, sparsity, and architecture. The methodology applies exclusively to DNNs, but has been extended and used with CNNs (Yang, Y.-H. Chen, and Sze, 2017) in an optimization loop to reduce the computations energy consumption. Mittal then proposes a survey (Mittal, 2019) to optimize and evaluate neural networks on some of the embedded platforms that we use in this work (NVIDIA Jetson TK1 and TX2).

Kim et al. reviewed (Y. G. Kim et al., 2018) the operating system-level energy management techniques for heterogeneous elements. Although the review does not deal with energy models, it lists energy management techniques such as power-saving schedule that we utilize to dynamically optimize the energy resource of the aerial robots. They also detail aspects such as Quality of Service (QoS) that we use to define execution boundaries for the computations.

3.1.2 GPU features modeling

GPUs are used in several applications despite increasing energy demands (Mittal and Vetter, 2014) due to their high computational resources (Kasichayanula et al., 2012). It is hence unsurprising

that for computations energy modeling, GPUs have their own modeling approaches. Here we discuss some studies of interest to our work. Mittal and Vetter and Bridges et al. propose extensive reviews of the available methodologies (Bridges et al., 2016; Mittal and Vetter, 2014). Both are, however, not tailored to mobile computing hardware.

Hong and Kim derive an energy model (S. Hong and H. Kim, 2010) for GPU features. The contribution consists of an integrated power and performance prediction model to derive the optimal number of active cores for a given application to achieve energy savings (S. Hong and H. Kim, 2010). The model is GPU specific: it consists of an analytical expression that estimates the GPU power and time in function of some parameters. Opposite to our work, it does not model the energy of the entire platform, nor is deployed on mobile computing hardware. Nevertheless, it achieves high accuracy in terms of GPU power and time prediction.

Wu et al. derive another model (G. Wu et al., 2015) with a similar focus on GPU features. The approach uses machine learning techniques for performance and estimates the power from real GPU hardware. In particular, Wu et al. train a neural network with different performance counters for various GPU configurations of a collection of applications and derive the optimal values of these counters. The approach works across multiple hardware; data gathered from one hardware estimates the power and performance of multiple other GPU hardware. However, Wu et al. focus purely on GPU modeling. Moreover, machine learning is an energy-expensive computation itself (García-Martín et al., 2019; Yang, Y.-H. Chen, Emer, et al., 2017), thus deterring similar approaches in our planning, where we aim to preserve the energy as far as possible.

Collange et al. propose an empirical approach (Collange et al., 2009) for energy evaluations of GPUs during various computations in the CUDA environment. The approach measures and analyzes how computations impact instantaneous energy consumption. It observes a significant energy impact of memory accesses and generally analyzes the energy cost of parallel GPU computations. The work by Collange et al. thus aims to derive the optimal memory configuration of a given benchmark. However, it does not use the measured data to calibrate a model for energy estimation nor focus on mobile computing devices.

Contrary to Collange et al.'s approach of pure empirical measurements, Luo and Suda(C. Luo and Suda, 2011) propose an analytical model for energy and performance estimation of GPUs. The model contains execution time and energy consumption prediction sub-models, where the latter follows from the former. The final analytical model for the energy estimation multiplies the execution time and the estimated power derived using an analytical expression. The work derives some observations on the energy implications of a given benchmark, which is then not used nor hypothesized in any energy adaptation technique. The accuracy analysis, redefined by the authors with another analytical expression, shows a strong correlation between observation and the analytical model. We also derive an analytical expression further motivated with empirical observation of energy data and employ a multivariate linear regression. We then focus on heterogeneous elements and mobile computing hardware.

Leng et al. propose a model (Leng et al., 2013) based on a performance-per-watt metric for GPUs. Developed for general-purpose GPU (GPGPU) powered devices, it is composed of multiple stages. An initial model is validated with some empirical measurements and eventually refined.

The contribution suggests an integrated power and performance modeling framework, which inputs a configuration to define micro-architectural and component parameters. It is then used in a feedback-driven optimization loop with the GPGPU. In our approach, we similarly input a configuration to the model, but we specify the computing hardware computations rather than architectural parameters. We use the output of the model in an optimization loop but extend this latter stage together with the path. Computation-wise, we use both heterogeneous elements.

3.1.3 CPU features modeling

Numerous other contributions model the CPU energy specifically and investigate how to lower the power (Chowdhury and Chakrabarti, 2005; I. Hong et al., 1999; J. Luo and Jha, 2001) by some system-level optimization techniques. These include DVS, DFS, multiple asymmetric cores (such as the ARM big.LITTLE architecture), and power gating (Walker et al., 2017). They usually incorporate information about configuration parameters into the scheduler (Seewald, Schultz, Ebeid, et al., 2021) and focus on homogeneous opposite to heterogeneous systems in our work.

Lee and Brooks provide extensive coverage of regression modeling and propose a regression-based model (B. C. Lee and Brooks, 2006a,b) for microprocessors. The model uses a small number of samples of both the power and performance in a joint architecture-computations search space. The entire search space is sampled uniformly at random, an approach that allows identifying trends and trade-offs between the parameters (B. C. Lee and Brooks, 2006a). The results show high accuracy even with a relatively small number of samples. We likewise use a regression model in our computations energy modeling approach but cover all the heterogeneous elements. Nonetheless, Lee and Brooks' approach is of interest to evaluate the accuracy of regression techniques and distributions of samples. Depending on the configuration space, we sample the computations uniformly using a linear or exponential distribution of samples (we specify the sampling distance in a configuration file), whereas Lee and Brooks sample architectural parameters.

Takouna et al. propose statistical multicore CPU power models (Takouna et al., 2011) based on the average running frequency and the number of active cores. The models use multivariate linear regression commonly to other computations energy models in this section. Although insightful in terms of multicore CPU power models, the approach does not model GPUs nor heterogeneous elements and is tailored on virtualized serves, opposed to mobile computing hardware that we focus on in this work. Moreover, Takouna et al. do not consider computations configurations in the regression but a variable selection of architectural parameters.

Reddy et al. propose an empirical CPU power model (Reddy et al., 2017) that uses measured data and simulates the energy consumption of a quad-core ARM Cortex-A15 and gem5–a full-system architectural performance simulator. To collect the measurements, the model uses built-in power monitors in the ODROID-XU3 platform. Reddy et al. evaluate the model against sixty workloads reporting high accuracy on the ODROID-XU3 computing hardware. Our model shares compatibility with this computing hardware but is not simulator-dependent; we further evaluate the model on multiple heterogeneous computing hardware such as NVIDIA Jetson TK1, TX2, and Nano platforms.

Nunez-Yanez and Lore and Nikov et al. develop other models (Nikov et al., 2015; Nunez-Yanez and Lore, 2013) for mobile computing hardware. These models use hardware event registers and capture the state of the CPU under a representative workload. The latter study proposed by Nikov et al. is of particular insight, as they focus on ODROID-XU3 computing hardware; the model has three modeling stages: data collection of a benchmark running on the platform occurs in the first stage. The second and third stages are performed offline on a different architecture. They process the collected data (in the second stage) and generate the model (in the third stage) (Seewald, Schultz, Ebeid, et al., 2021). The model further deals with ARM big.LITTLE architecture. We use these models to evaluate the performance of our computations energy modeling.

Walker et al. propose a run-time model (Walker et al., 2017) that uses performance monitoring counters (PMCs) for mobile computing hardware, implemented on mobile CPUs such as ARM Cortex-A7 and Cortex-A15. Walker et al. use the architectural performance simulator gem5 comparably to Reddy et al. and the ODROID-XU3 computing hardware to Nikov et al. The approach is to build a linear regression for a given CPU power prediction experiment. Although the study considers CPU-powered computing hardware only, it is insightful in terms of proposed statistical rigor in building run-time CPU power models.

3.2 Battery Modeling

In this section, we describe some battery modeling approaches and focus on battery models that we can use with both the computing hardware and the aerial robot. There are several models for this purpose, each with its advantages and disadvantages. Physical models use ordinary and partial differential equations, accurately predicting the battery evolution in time (R. Rao et al., 2003) with a considerable time and computational cost (Doyle et al., 1993; Lotfi et al., 2017; Marcicki et al., 2013; Moura et al., 2017). Hybrid models are similar but less accurate, predicting the battery with less computational complexity (T. Kim and Qiao, 2011; T. Kim, Qiao, and Qu, 2019). Empirical models use a set of trials (Pedram and Q. Wu, 1999; Syracuse and Clark, 1997), similarly to some computations energy models we discuss in Section 3.1, and mixed models use empirical data to derive some parameters for analytical expressions (Rakhmatov and Vrudhula, 2001; R. Rao et al., 2003). Abstract models use techniques such as the time evolution of an equivalent electrical circuit (Benini et al., 2001; Gold, 1997; Hasan et al., 2018; X. Hu et al., 2012; S. Lee et al., 2008; Xing et al., 2014) comparably to hybrid models (T. Kim and Qiao, 2011) but with less computational complexity. They have compelling trade-offs concerning analytical insight, accuracy, and complexity (R. Rao et al., 2003) and are the models of our choice for battery modeling. Rao et al. survey (R. Rao et al., 2003) the state of the art in battery modeling.

Xing et al. and Hasan et al. propose such abstract models (Hasan et al., 2018; Xing et al., 2014) with an equivalent electrical circuit. These models model the battery state of charge (SoC) with the time evolution of a differential model of an equivalent circuit. The first model (Xing et al., 2014) utilizes an unscented Kalman filter to tune some parameters at each sampling step, and the second (Hasan et al., 2018) estimates the SoC utilizing an eXogenous Kalman filter (Johansen

and Fossen, 2017). We use the model from Hasan et al. for the battery modeling. We derive an equivalent circuit for a Li-ion aerial robot battery and evaluate the SoC over time with a differential equation similar to the one proposed by Hasan et al.

Rao et al. (V. Rao et al., 2005) propose a battery model for computing hardware specifically. The approach consists of an abstract stochastic model that uses Markov processes with probabilities related to the physical characteristics of the battery (Panigrahi et al., 2001). Rao et al. further improve the accuracy of the model, including some physical battery characteristics. The approach is insightful in terms of evaluating models for computing hardware. Indeed our approach relies similarly on an abstract model (Hasan et al., 2018), whereas we focus on an equivalent electrical circuit which requires less battery-specific knowledge for modeling. We can then model the SoC with little information regarding what battery the aerial robot mounts.

Zhang et al. propose a modeling technique for smartphones (L. Zhang et al., 2010) that consists of an automated modeling technique based on the battery discharge behavior and voltage sensors. The technique models the power using a multivariate regression (similarly to our approach and many computations energy models in Section 3.1) and the battery using an equivalent circuit. Although the analysis is limited to smartphones, it provides insights into automated modeling techniques for computing hardware. We have similarly derived an automated modeling technique for computations, battery, and entire system model in Chapter 4 that we integrate into our energy-aware coverage planning and scheduling.

Benini et al. propose a battery model (Benini et al., 2001) for the low-power design of computing hardware. The model, discrete-time and intended for system-level design environments, is also derived from an equivalent circuit. Although specific to the design of computing hardware applications, it evaluates different battery types: a first Li-ion implementation of the model is extended to various battery types with both non and rechargeable cells.

3.3 Motion Planning

Planning algorithms for mobile robots include broad and diverse topics. Energy-wise, the algorithms select an energy-optimized trajectory (Mei, Y.-H. Lu, Y. C. Hu, et al., 2004), e.g., by maximizing the operational time (Wahab et al., 2015). However, many studies apply to a limited number of robots (C. H. Kim and B. K. Kim, 2005) and focus exclusively on planning the trajectory (H. Kim and B.-K. Kim, 2008), despite compelling evidence for the energy consumption also being significantly influenced by computations (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005; Ondrúška et al., 2015). In the remainder of this chapter, and before discussing the studies in mobile robotics that deal with computations and motion energy altogether, we detail planning in the literature. We are interested in planning and scheduling, and further recall that we are specifically interested in coverage planning of a given space in an energy-efficient way. We discuss the available literature on coverage planning for mobile robots later in this section and emphasize relevant studies for aerial robots in Section 3.4.1. There are multiple approaches in this sub-topic of motion planning (Cabreira, Brisolara, et al., 2019; H. Choset, 2001), including studies that derive

an energy-efficient cover (Cabreira, Franco, et al., 2018; Wei and Isler, 2018). In Section 3.3.2 and Section 3.4.2, we discuss these studies respectively for mobile robots and aerial robots.

Within simple motion planning instead of the derivation of a cover, Mei et al. propose an energy-efficient approach for mobile robots distinguishing the two energy consumers–the computations and motion energy. In (Mei, Y.-H. Lu, Y. C. Hu, et al., 2004), they focus on optimizing the motion energy and analyze the computations energy in a later iteration of their work (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005). We discuss the latter study in Section 3.5. Their motion plan is composed of a path plan and a velocity schedule: the path plan is similar to ours in Definition 2.4.2, and the velocity schedule contains velocities, accelerations, and decelerations along the route. Mei et al. then analyze the most energy-efficient path plan and velocity schedule; they evaluate the energy cost of a plan–of turns and accelerations–differentiating between the traveled against motors velocities. We have also optimized the coverage plan in Section 2.6.1 compared to traditional coverage planning that uses, e.g., boustrophedon decomposition (LaValle, 2006). We do not consider a velocity schedule in our dynamic planning due to the physical constraints of airborne systems. Nevertheless, we also optimize the plan against sudden accelerations. Recall the plan in Figure 2.6 requires sudden decelerations in contrast to the plan in Figure 2.7 for fixed-wing crafts. Indeed one of the observations Mei et al. underline is the energy cost of turns. As an experimental setup, the study uses Palm Pilot Robot Kit with polyurethane omnidirectional wheels and three MS492MH DC servo motors. It relies on external power monitors to measure the power drain of the systems and five batteries: a 9 V battery and four 1.5 V batteries for the control circuit and motors. They measure the energy efficiency in terms of squared meters over jouls and report an improvement of 50% by using an energy-optimized path plan. Mei et al. study has encouraged our initial analysis in energy-aware planning and scheduling in many respects. Mei et al. are the first study we found that analyzed motion and computations energy distinguishing explicitly for the need to account for computations energy (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005). However, Mei et al. lack further energy efficiency analysis in path variations. Indeed they focus on straight lines, spirals, and the energy cost of turns, whereas we focus on a generic variation in path within given limits. They do not explicitly address computations and motion planning yet propose foundations for future studies.

Dressler and Fuchs undertake a different approach to energy-efficient motion planning. In their work (Dressler and Fuchs, 2005), they focus on energy modeling of motion plans for given tasks (the term is not to be confused with computations), where, e.g., the mobile robot has to explore and supervise unknown surroundings. Dressler and Fuchs propose a future planning direction based on energy control and battery management for efficient tasks allocation but do not investigate further energy-aware planning. The study models energy-consuming parts of the robot with some corresponding characteristic curves. Dressler and Fuchs observe a linearly approximable curve for the SoC for some sub-tasks of a given task, such as movement and wireless communication. Although we do not model the energy of a given task, we similarly formulate the plan as a set of paths and computations in Definition 2.4.2 (of which we derive the optimal configuration energy-wise). The study has thus inadvertently introduced the differentiation between computations and motion energy that we have built upon. The reported sub-tasks energy

models are indeed relative to one of the two components: wireless communication to a computation that transfers data airborne, the movement to a path to reach the location. Dressler and Fuchs have further developed the approach, focusing on sensor networks (Dressler and Dietrich, 2006; Fuchs et al., 2006).

The majority of other contributions focus on optimizing motion planning to increase power efficiency. There are several studies that survey the available literature for mobile robots' planning, along with some textbooks (H. M. Choset et al., 2005; LaValle, 2006). Notably, La Valle's textbook (LaValle, 2006) groups planning algorithms literature and focus on robot motion planning. Juliá et al. propose an extensive study of the most important methods for motion planning for autonomous exploration and mapping of unknown environments. The survey presents different techniques and classifies them for the level of multi-robot coordination and integration with simultaneous localization and mapping (SLAM) algorithm.

3.3.1 Coverage path planning

In autonomous mobile robots scenarios, it is often required to explore every location in a given elemental region (Cao et al., 1988); this latter problem is defined in the literature coverage path planning (CPP) problem. In this section, we briefly discuss some surveys that deal with CPP in mobile robotics generically. We detail approaches that involve aerial robots in this latter sub-topic of motion planning in Section 3.4.1; in our work, we are dealing with this problem when we cover a given agricultural area in Problem 2.5.1.

CPP is related to the covering salesman problem, a variation of the famous traveling salesman problem (TSP): the salesman visits a neighborhood of each city, minimizing the travel length (E. M. Arkin and Hassin, 1994), and passes over all points rather than through all the neighborhoods (H. Choset, 2001). This problem is sometimes referred to as the lawnmower problem (Galceran and Carreras, 2013) and is proven to be NP-hard (E. M. Arkin, S. P. Fekete, et al., 2000). Early studies solve it using a heuristic with no coverage guarantee (H. Choset, 2001). For instance, Arkin and Hassin propose simple heuristic-based algorithms for constructing the tours (E. M. Arkin and Hassin, 1994). There have emerged a variety of approaches ever since. Arkin appears in other studies that propose a continuous version for the covering salesman problem algorithm (E. Arkin et al., 1993; E. M. Arkin, S. P. Fekete, et al., 2000; S. Fekete et al., 1994). These studies solve the problem with lawn moving and milling algorithms (E. M. Arkin, S. P. Fekete, et al., 2000). Many approaches either implicitly or explicitly use cellular decomposition to guarantee the coverage (exact or approximate), decomposing the space to cover into cells; the overall solution to the coverage planning problem is then the union of the solutions of the cells (H. Choset, 2001).

To cover each cell in a cellular decomposition, the robot can use simple back and forth motion parallel to the cell's boundaries–often referred to as boustrophedon motion (LaValle, 2006)–and reduce the coverage planning to motion planning between the cells (H. Choset, 2001). The cells are geometric structures to represent the robots' free space and are decomposed into trapezoids in a popular class of solutions under the name of boustrophedon decomposition (since they are

related to the boustrophedon motion). In this decomposition, the algorithm splits the origin cell in case of an obstacle (change in connectivity) (H. Choset, E. Acar, et al., 2000). Choset et al. propose exact cellular decomposition method (H. Choset and Pignon, 1998) that they complement with different motions in a later instance of their work (H. Choset, E. Acar, et al., 2000) (they consider, e.g., spiral, spike, diamond pattern, and others). Within the cellular decomposition method, some approaches (E. U. Acar et al., 2002; H. Choset, E. Acar, et al., 2000) work with the concept of Morse function to indicate the location of cell boundaries. We show how does the boustrophedon decomposition proposed by Choset et al. work in Section 5.2.1. Grid decomposition is another way of solving the coverage in the literature (Gabriely and Rimon, 2002; Shnaps and Rimon, 2016; Wei and Isler, 2018; Zelinsky et al., 1993) that divides the space into equally sized cells. Other approaches are based on graphs (Cheng et al., 2019) or derive an optimal coverage (Huang, 2001; T.-K. Lee et al., 2011; Y. Li et al., 2011; Wei and Isler, 2018; Xu et al., 2011). The choice of the approach depends on the practical application (Wei and Isler, 2018). The optimal coverage approaches are often a variation of known studies but with a cost that has to be optimized (Galceran and Carreras, 2013). Our work fits into this latter class of approaches. In the remainder of this section, we discuss two surveys that deal with coverage planning and further detail relevant studies on optimal coverage.

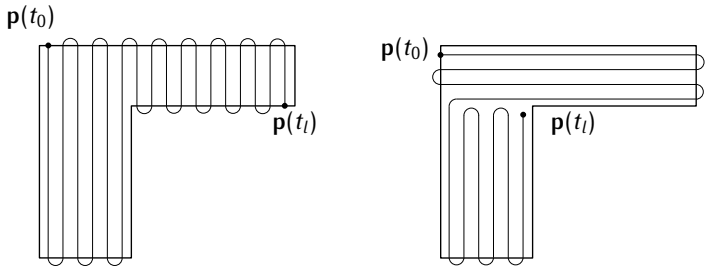
Choset proposes a survey (H. Choset, 2001) of literature on coverage for robotics. The survey classifies the studies according to the decomposition algorithm. The classification includes heuristic and cellular decomposition-based algorithms (approximate, partial-approximate, or exact cellular decomposition algorithms). Choset report that many algorithms for coverage planning with some guarantee in terms of either optimality or completeness are based on cellular decomposition. Approximate cellular decomposition algorithms described by Choset correspond to grid-based algorithms in other studies.

Galceran and Carreras propose one of such studies: a survey (Galceran and Carreras, 2013) of the different approaches for CPP. The survey classifies the algorithms among the others in online and offline classes, where the former rely on stationary information under a known environment and the latter on sensor measurements to sweep the space (Galceran and Carreras, 2013). Like Choset, Galceran and Carreras report that most coverage planning algorithms decompose the space in sub-regions called cells to achieve coverage. The survey is extensive in terms of reviewed literature. It compromises studies focusing on cellular decomposition, cellular decomposition based on critical points of Morse functions (the latter can deal with generalized obstacles), graph (such as roads and streets networks), and grid algorithms. The work further surveys studies based on the detection of natural landmarks and approaches for robots with contact sensors. It covers methodologies operating in rectilinear environments, for 3D environments, for optimal coverage, for multiple robots, and based on reduction of the localization error.

3.3.2 Optimal coverage

Huang analyzes an optimal coverage methodology based the cellular decomposition. The study (Huang, 2001) emphasizes the cost of performing turns and uses a boustrophedon-like

motion that minimizes the number of turns. Huang utilizes round turns like the intuitive plan in Figure 2.6. Huang minimizes the sum of sub-region altitudes–a metric related to the sweep direction of a sub-region. We illustrate the principle in Figure 3.1, where different sweeping direc-



(a) same sweep direction (b) different directions

Figure 3.1. An example (Huang, 2001) showing that different sweep directions for sub-regions in a cellular decomposition algorithm produce a coverage with an optimized number of turns; the robot moves from point $\mathbf{p}(t_0)$ to $\mathbf{p}(t_l)$ and in (b) has almost half the turns of (a).

tions produce a different number of turns. Huang proposes the methodology for both convex and non-convex shapes and uses dynamic programming for the optimal direction in the coverage. The approach performs better than other approaches by considering possible different sweep directions for different sub-regions. The study is insightful in terms of analyzing the coverage optimality of different directions in a boustrophedon-like motion. We similarly try to optimize the turns in our dynamic planning.

Arkin et al. have focused (E. M. Arkin, Bender, et al., 2001, 2005) on optimal coverage and have, similarly to Huang, considered the length of the tour with the number of turns. Both Arkin et al. and Huang remark that in many routing problems, the turns dominate the cost since the robot is often required to slow (E. M. Arkin, Bender, et al., 2001). Arkin et al.'s studies are insightful in terms of algorithmic rigor. They prove that coverage planning with turn costs is NP-complete even when the objective is merely to minimize the turns. To this end, they propose various approximation algorithms to compute nearly optimal covering tours. Nevertheless, the two studies and Huang's study are inconclusive in terms of nonholonomic robot constraints: they do not consider aspects such as the radius of the turn for optimal coverage. They also do not consider the eventuality of replanning in case external interferences affect the feasibility of the original plan. The latter scenario can happen in the eventuality of an aerial robot suffering a sudden battery drop while on a given optimal tour.

Shnaps and Rimon propose a solution to this latter problem and focus on robotics scenarios explicitly. In the study (Shnaps and Rimon, 2016), a robot has to cover an unknown environment with a strict battery constraint. Starting from a given point, the robot equipped with position and obstacles sensors navigates the environment and eventually covers the entire space. In a battery discharge event, the robot returns to the starting point to recharge the battery. Shnaps and Rimon model the energy cost with the path length and divide the space to cover into equally sized cells in a grid. Although insightful in terms of coverage planning for battery-powered robots, the

approach does not account for aerial robots. Indeed in this latter class, the robots would require to land to recharge or replace the battery. Wei and Isler also consider the eventuality of strict energy constraints for mobile robots and revisit Shnaps and Rimon's approach. In the study (Wei and Isler, 2018), they propose an algorithm restricted to axis-parallel motion generalized to arbitrary polygons. In both studies, the methodology is to divide the space into equally sized cells in a grid. The approach is tested on aerial robots, but is limited to rotary-wing crafts and generally unsuitable for nonholonomic robots' constraints. In both Shnaps and Rimons's and Wei and Isler's approaches, the energy optimality is considered within CPP rather than an integrated dynamic planning and scheduling approach in our work.

3.4 Planning for Autonomous Aerial Robots

For what concerns planning for aerial robots, Popović et al. propose a study (Popović et al., 2017) for planning and subsequent replanning to satisfy dynamic constraints. The study addresses the environment's disturbances and sensors' uncertainty using a noise-dependent model and accounts for a limited time budget. Popović et al. plan path online combining evolutionary optimization and global viewpoint selection. They validate the approach with a precision agriculture scenario of detecting weeds. Although the proposed agricultural scenario has similarities with ours, it does not account for different aerial robots nor perform a dynamic energy replanning (of both the path and computations) in the function of battery state. Furthermore, the scenario does not require complete coverage, making it unsuitable for some applications we consider, such as hazard detection. Hayat et al. propose an approach (Hayat et al., 2017) based similarly on evolutionary optimization in terms of multiple aerial robots planning of tasks and paths (the notion of task refers to a specific action rather than computation) for a search and rescue scenario. The approach has the same limitations as Popović et al.

Many other studies in planning for aerial robots under some energy constraints limit to energy-aware motion planning (Kreciglowa et al., 2017; Morbidi et al., 2016; X. Wang et al., 2017). They do not consider coverage planning nor disturbances. Kreciglowa et al. focus on the best trajectory between two hovering configurations in terms of energy efficiency (Kreciglowa et al., 2017). Morbidi et al. generate energy-optimal paths solving optimal control problems (OCPs) to obtain the angular accelerations of four electrical motors of a quadrotor rotary-wing aerial robot (Morbidi et al., 2016). Wang et al. propose a study (X. Wang et al., 2017) of motion planning applied to the fixed-wing aerial robot. The approach passes through a set of waypoints by satisfying a given constraint on curvature.

3.4.1 Aerial coverage path planning

In the context of aerial robots, the survey (Galceran and Carreras, 2013) proposed Galceran and Carreras that we discussed in Section 3.3 focus on optimal coverage (Xu et al., 2011) and multi-robot coverage (Ahmadzadeh et al., 2008; Araújo et al., 2013; Barrientos et al., 2011; Maza and Ollero, 2007). Our work come into the intersection of optimal coverage and coverage

performed using aerial robots. Indeed in our coverage planning we dynamically refine a plan to achieve energy-aware behavior.

Cabreira et al. propose a survey (Cabreira, Brisolara, et al., 2019) that covers CPP exclusively for aerial robots. The study classifies the algorithms using the classification introduced by Choset (H. Choset, 2001) and Galceran and Carreras (Galceran and Carreras, 2013) from Section 3.3 (the algorithms are split by cellular decompositions employed, and if they perform coverage online or offline). The survey further focuses on the shape of the coverage area and reports the performance metrics such as the path length, coverage time, and the number of turning maneuvers. Cabreira et al. focus on single and cooperative strategies for exact cellular decomposition and report the type of information used for the decomposition (full or partial). Remarkably, they analyze studies based on genetic algorithms and ant colony optimization and focus on different covering strategies and describe the so-called Zamboni motion similar to the fixed-wing plan in Figure 2.7. Araújo et al. also analyze covering strategies (Araújo et al., 2013) for a given cell of interest. They describe the boustrophedon (that appears under the name "lawnmower") and Zamboni motion discussed in this work and additional spiral, spiral-like, Dubins path, modified boustrophedon, and modified Zamboni motions.

While CPP under uncertainty for aerial robots appears in Cabreira et al.'s survey, there are two surveys (Dadkhah and Mettler, 2012; Goerzen et al., 2010) with Mettler that review approaches dealing with uncertainty in general terms of aerial robots motion planning. The former survey (Goerzen et al., 2010) emphasizes a classification based on differential constraints, grouping the motion planning algorithm with and without differential dynamics. The survey observes a lack in approaches dealing with uncertainty, whereas the latter (Dadkhah and Mettler, 2012) analyzes these explicitly. It groups the motion planning approaches by the class of uncertainty. For instance, in the case of uncertainty in vehicles' dynamics, the survey proposes studies based on optimal control and artificial intelligence. In the case of environment knowledge uncertainty, studies based on the mapping.

Nam et al. propose an approach (Nam et al., 2016) for CPP for aerial robots that generates the covering tour offline. The approach uses a grid decomposition method, whereas the grids are visited using a wavefront algorithm–a specialized version of Dijkstra algorithm that optimizes the number of stages to reach the goal (LaValle, 2006). Using the wavefront algorithm, they generate the optimal path between given points in space but do not focus further on optimal criteria. Indeed the practical analysis shows the result not being optimized with regards to the number of turns. Moreover, the approach focuses on rotary-wing aerial robots. Nam et al. do not deal with replanning in the eventuality of an unexpected event occurring nor account for energy explicitly, and the planning happens before the flight.

Sadat et al. propose an interesting approach (Sadat et al., 2014) for non uniformly shaped areas distributed into clusters, performing adaptive coverage depending on the distributions of the regions of interest in the space. Sadat et al. use coverage trees for the purpose (a structure where the child nodes cover the same area as the parents but with higher resolution) and propose different strategies to visit the tree (breadth-first, depth-first, and shortcut heuristic). Although

insightful in defining the coverage area sparsely, the approach does not account for the aerial robot's battery nor discuss other aerial robots other than the rotary-wings.

3.4.2 Optimal aerial coverage

Di Franco and Buttazzo propose an energy-aware CPP for aerial robots. The study (Di Franco and Buttazzo, 2015) focuses on energy and other requirements, such as the completeness of the coverage and resolution. The energy model is generic for a given drone and outputs the energy consumption as a function of velocity and operating conditions using an interpolating curve of the power measurements. In particular, for future energy predictions, Di Franco and Buttazzo derive an energy model from measurements flying the drone in several conditions (during maximum acceleration and deceleration, horizontal flight, climbing, descending, hovering, and turns). The study hence proposes different analytical expressions derived from the interpolations for the scenarios. It covers a polygon with the boustrophedon decomposition similarly to Figure 2.6 but with sharp turns. Di Franco and Buttazzo then consider the quality of the coverage with varying distances between the coverage lines when the total modeled energy is lower than the available energy in an extension (Di Franco and Buttazzo, 2016) of the original study (Di Franco and Buttazzo, 2015). However, the latter study does not plan the coverage in flight nor focus on other aspects of the aerial robot. Furthermore, the energy model is plan specific–different paths have a different analytical expression for the future energy. Our model in Section ?? is generic, once trained with enough measurements for a given plan variation. The study is insightful for defining the energy cost of a given coverage and analyzes how variations in the coverage quality affect the power consumption. It uses an IRIS rotary-wing aerial robot (with four 850 Kv motors), a GoPro camera mounted on a gimbal stabilizer, and a PX4 flight controller. The systems are powered with a 3S 11 volts and 5.5 amperes per hour lithium polymer battery (LiPo) battery. The aerial robot is not equipped with mobile computing hardware for computations, and the study does not consider other classes of aerial robots.

Concerning other related approaches for aerial CPP, Li et al. propose a study (Y. Li et al., 2011) based on an enhanced cellular decomposition method. Li et al. plan the coverage in a polygon area and derive a covering methodology that minimizes the number of turns. The turns, already considered in the generic CPP by Arkin et al. and Huang (E. M. Arkin, Bender, et al., 2001, 2005; Huang, 2001), are here further showed less efficient from energy, duration, and tour length points of view. For complex polygons, Li et al. propose a convex decomposition algorithm for minimum width sum based on the greedy recursive method. To connect the decomposed sub-regions, the authors propose a minimum traversal algorithm of a weighted undirected graph. The approach is complete in terms of algorithmic analysis. Li et al. provide the algorithms, analyze their complexity, and simulate the coverage on polygons of various shapes. They show their algorithms being optimal in terms of turns nonetheless, they do not consider further requirements for different aerial robots. Indeed a fixed-wing aerial robot has a greater turning radius compared to a rotary-wing aerial robot. Moreover, Li et al. plan the coverage offline. They do not consider unexpected occurrences derived from the robot's and environment's uncertainty, such as sudden

battery discharge, wind gusts, and other atmospheric conditions; we take these aspects into account in our coverage planning. They further consider only path planning, opposed to planning the path along with the computations energy-wise while optimizing the battery usage.

Mannadiar and Rekleitis and Xu et al. propose studies (Mannadiar and Rekleitis, 2010; Xu et al., 2011, 2014) on near-optimal complete coverage computed using a sequence of waypoints. The algorithms are near-optimal due to waypoint control having less maneuverability than velocity control (Xu et al., 2014). They cannot thus guarantee optimality in terms of the tour length. The coverage algorithm (Mannadiar and Rekleitis, 2010) was first presented by Mannadiar and Rekleitis and extended to non-holonomic robots (Xu et al., 2011, 2014) by Xu et al. The algorithm uses the cellular decomposition of a known environment with an arbitrary number of obstacles. The algorithm feds the decomposed cells into a Reeb graph–an encoding used to compute a cyclic path where critical points are vertices and cells are edges (Fomenko and Kunii, 1997). It then solves the Chinese postman problem (Eiselt and Laporte, 2000) on the graph to derive the coverage order. The resulting path ensures that no cells are not traversed more than twice, with the robot returning to the starting point. Xu et al. analyze possible planning strategies for fixed-wing aerial robots and affirm that this latter class of robots lack the maneuverability needed to follow a sharp-turned path. Like Xu et al., we also generate paths that are to be followed by various classes of aerial robots, including fixed-wings. Xu et al. append a turning segment to the path by adding curlicue orbits at turns; whereas we incorporate the turning maneuver in our coverage planning rather than embedding external segments. Moreover, the studies do not focus on energy-aware coverage planning, on eventual replanning in case of adverse events, nor involve power-saving scheduling.

There are several other approaches for energy-aware aerial coverage. Cabreira et al. and Artemenko et al. propose studies (Artemenko et al., 2016; Cabreira, Franco, et al., 2018) in this direction, focusing on photogrammetry in localization scenarios. Cabreira et al. propose spiral coverage for some polygon shapes and alter the velocity to achieve energy saving. Artemenko et al. propose a boustrophedon motion with smoothed turns using Bézier curves but analyze other motions. However, Cabreira et al. and Artemenko et al.'s studies focus on a peculiar scenario rather than generic coverage. Another study provides an efficient coverage using different motion patterns for specific locations such as urban areas (Dille and Singh, 2013) but does not derive a generalized methodology. Some studies deal with optimal coverage deriving approaches (Bouzid et al., 2017; Valente et al., 2013) for rotary-wing aerial robots using interesting novel algorithms to find the optimal tour in a graph. In particular, Valente et al. use harmony search–a meta-heuristic algorithm based on musical harmony (Geem, 2009)–and Bouzid et al. use a genetic algorithm. Generally, all the studies in this paragraph focus on rotary-wing aerial robots. Although interesting in terms of analyzing the energy efficiency the studies do not propose replanning, nor power-saving scheduling.

3.5 Planning Computations with Motion

Sudhakar et al. examine the trade-off between motion and computation energy for mobile robots in their recent study (Sudhakar et al., 2020), focusing on robots with similar motion and computations energy. In the study, the robot moves on a path with a given length and velocity and, computations-wise, computes a specific number of nodes. Sudhakar et al. first derive an analytical expression for energy prediction that incorporates the path's length and velocity and the number of computations' nodes. The approach they propose consists of an algorithm for anytime planning–a planning approach that identifies an initial feasible plan then refined towards optimal over time (Karaman, Walter, et al., 2011). The algorithm eventually stops the refinements when it estimates computations energy exceeding the potential savings in terms of motion energy (Sudhakar et al., 2020). To derive a path, the algorithm uses Bayes estimation on the edges of a graph in a sampling-based path planning algorithm–probabilistic roadmaps (PRM*) widely used in practice (Karaman and Frazzoli, 2011; LaValle, 2006). The study motivates our investigation and proposes an initial step towards further analysis. Nevertheless, it lacks rigor in deriving an energy model, and it does not account for battery SoC. Notably, Sudhakar et al. emphasize the importance of considering both the motion and computations for robots' energy-awareness; we share similar findings except for the study's claimed primate in analyzing trade-offs between motion and computations energy. By a closer inspection of the scientific literature, other studies have emerged in this direction. Mei et al., Sadrpour et al., and many others propose studies that deal with mobile robots motion and computations energy (Brateman et al., 2006; Mei, Y.-H. Lu, Y. C. Hu, et al., 2005; Mei, Y.-H. Lu, Y. C. Hu, et al., 2006; Sadrpour et al., 2013a,c; W. Zhang and J. Hu, 2007). Recent contributions include work carried by Ondrúška et al. and Lahijanian et al. (Lahijanian et al., 2018; Ondrúška et al., 2015)

Mei et al., which we already analyzed for their contribution to motion planning in Section 3.3, have proposed a study (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005) analyzing both motion and computations energy. The study differentiates the microcontroller and embedded computer the mobile robot carries for a more flexible robot design. To this end, Mei et al. derive an energy model from empirical data and show that the motion accounts for less than 50% of the total power consumption. Motivated by this finding on Pioneer 3DX ActivMedia–Pioneer mobile robots from Adept MobileRobots were popular at the time of publication within the research community (Anguelov et al., 2004; Erickson, 2003; Lemay et al., 2004)–they propose real-time scheduling and dynamic power management to reduce the power consumption. The latter dynamically adjusts power states (e.g., DVS, DFS, and peripherals selection) of components without compromising overall performance (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005), and the former schedule computations energy-wise. The study is of particular interest as they quantify the computations' contribution to the overall energy expenditure of mobile robots. We further extend the study; it was indeed one of the starting points of our analysis. Yet, it does not implement the proposed techniques nor focus on planning within battery constraints–a research gap we are filling with our energy-aware coverage planning and scheduling. Mei et al. have extended their analysis in future instances (Mei, Y.-H. Lu, Y. Hu, et al., 2005a,b; Mei, Y.-H. Lu, Y. C. Hu, et al., 2006) to CPP

using multiple robots with both time and energy constraints. They have then solved repeated coverage (Mei, Y.-H. Lu, C. Lee, et al., 2006) but have not implemented computations scheduling nor focused on dynamic coverage replanning with sudden battery discharge. This latter aspect is of particular interest to aerial robots.

Brateman et al. propose an approach (Brateman et al., 2006) on the intersection of modeling techniques for CPUs, which we discuss in Section 3.1.3, and planning computations and motion for mobile robots. The methodology varies the computing hardware's frequency and voltage via DFS and DVS and the motor's speed for energy efficiency. To find the best trajectory of the frequency, voltage, and speed over time, Brateman et al. formulate a nonlinear optimization problem (NLP) that they then solve using numerical optimization methods. The predicted energy is an analytical expression derived via probabilistic analysis that the methodology employs as an optimization cost within a time constraint. There are many more approaches that employ optimization techniques for energy-efficiency of computations and motion in mobile robotics (Lahijanian et al., 2018; Ondrúška et al., 2015; W. Zhang and J. Hu, 2007). Zhang et al. vary the robot's speed and the computing hardware's frequency using optimal control on a random horizon. To this end, they transform an optimal control problem (OCP) into a nonlinear optimization problem (NLP) and solve the latter using a standard numerical optimization approach. The OCP has an analytical expression of the frequency and speed as the cost (W. Zhang and J. Hu, 2007). Interestingly, they also derived an analytical solution to the OCP for some simplified costs and found up to 30% energy savings compared to the heuristics methods. Like Brateman et al. and Zhang et al., we derive an OCP for replanning the coverage in Chapter A but vary the coverage path and the schedule on the computing hardware rather than the frequency, voltage, and speed. The intuition behind our more complex planning is that by scheduling and replanning appropriately, we can ensure coverage despite external adversities.

Sadrpour et al. focus on mission energy prediction–the energy needed to complete a given set of tasks by the robot traveling some paths while interacting with the environment (Sadrpour et al., 2013a)–for unmanned ground vehicles (UGVs). In their two studies (Sadrpour et al., 2013a,c), Sadrpour et al. propose two prediction approaches depending on a priori mission knowledge (i.e., constant power drain of static components, driving style, road rolling resistance, road grade information, and vehicle internal resistance). In case of no a priori knowledge, they propose linear regression and Bayesian estimation otherwise. Sadrpour et al. report the findings on the energy prediction using PackBot UGV (Yamauchi, 2004): the computations have an order of magnitude lower consumption than motion. Notably, Sadrpour and other researchers shared some findings in later studies (Ersal et al., 2014; Sadrpour et al., 2013b) in the direction of UGVs performing CPP. Although the studies and their later iterations focus on energy modeling, they propose a future instance in dynamic mission decision-making. When the energy exceeds a reasonable threshold, the UGV might, e.g., revise the initial plan or tune energy-expensive components (Sadrpour et al., 2013a). Our dynamic planning shares a similar principle of tuning the plan to accommodate energy expenditure. We further focus on computational aspects, as in the mobile robots under our study–and nonetheless, in recent energy-efficient robotics platforms–the motion energy has the same order of magnitude as computational energy (Sudhakar et al., 2020).

More recently, Ondrúška et al. proposed a study (Ondrúška et al., 2015) to reduce energy consumption by scheduling navigation at given times while traveling a path. The study focuses on Oxford Robotics Institute (ORI) ARC Q14 mobile ground-based robot–a robotic platform used mainly for research in planetary exploration rovers (Yeomans et al., 2017)–following a predetermined path and accounts directly for computations energy. Ondrúška et al. propose to schedule the navigation so that the robot remains within a margin from a given path. For the scheduling itself, Ondrúška et al. derive two algorithms. A greedy algorithm guarantees feasibility by employing a simple heuristic, and a belief planning algorithm uses optimal control to provide energy-efficient schedules on a given horizon. The approach considers path following rather than CPP, yet it is of interest in many respects. Firstly, Ondrúška et al.'s algorithm is based on optimal control. It is similar to ours based on output model predictive control (MPC). Secondly, the algorithm runs on a finite horizon. We employ the same technique for replanning. Finally, the approach further motivates our planning: Ondrúška et al. report a saving of 11.5% using a power-saving scheduler on the robot. Our methodology alters both the schedule and the path for aerial CPP.

Lahijanian et al. have further extended Ondrúška et al.'s work and proposed a framework (Lahijanian et al., 2018) for resource-performance trade-offs exploration. They find a schedule for mobile robots under a given resource budget using quantitative multi-objective verification and controller synthesis–a technique that starting from constraints, such as time and energy, produces a set of optimal achievable trade-offs via Pareto front (Forejt et al., 2012). Although insightful in selecting the best schedule, the approach does not account for dynamic planning, whereas in our work, an initial plan accounts for the highest achievable performance. These are then replanned both path- and computations-wise in an energy-aware fashion. Lahijanian et al. also use the ARC Q14 robot.

3.6 Summary

Chapter 4

Energy Models

"Robots require energy to operate. Yet they only have access to limited energy storage during missions."

— Ondrúška et al., 2015

Energy is an essential aspect of many autonomous mobile robotics scenarios (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005) and often a limiting factor to improving computing performance (Horowitz, 2014). In the previous chapters, we emphasized the growing importance of both computations and motion energy components. Indeed, limited energy availability against high computing performance requirements of autonomous aerial robots is the motivation for the energy-aware coverage planning and scheduling for autonomous aerial robots in this work. For this purpose, we first need an accurate energy model that predicts future energy consumption. We derive such a model in this chapter, providing different energy models predicting the energy of the motion and computations and the battery state of an aerial robot. For the former two, we first derive a way to accurately predict the energy spent running a given set of computations on the computing hardware. We then merge the resulting computations energy model with a motion energy model using some properties of our autonomous scenario. The energy output of the motion model is the input of the battery model, which predicts the battery evolution over time.

We saw in Chapter 4 some computations energy and battery models, and we discussed some energy-aware approaches for aerial robots (and mobile robots broadly) performing coverage path planning (CPP) or motion planning generally. In this chapter, we then use the previous literature and derive the energy models for our scenario: an autonomous aerial robot employed in coverage planning, monitoring an agricultural field. Although applied to a given use case, the approach is general in terms of the proposed methodology. In Section 4.1, we derive the model for the energy of computations based on regressional modeling. In Section 4.2, we provide a battery model based on an equivalent electrical circuit, and in Section ??, we derive a model for the motion that

incorporates the computations energy model. We then use the models in Chapter 5 to plan and schedule altogether.

This chapter connects to the remainder of this work as follows. We provided some energy implications of autonomous aerial robots in Chapter 1 and formulated the basic constructs in Chapter 2, including the concepts of computations and motion energy. We discussed the past energy modeling and efficiency studies in Chapter 3, along with other literature. We then use all the information we discussed in the previous chapters for the energy models in this chapter. In Chapter 5, we use the models to replan the coverage and schedule the computations in an energy-aware fashion. The replanning uses some optimal control techniques that we introduce in Chapter A.

4.1 Energy Model of the Computations

This section describes the computations energy model to predict the energy consumption of a given configuration of computations. In Section 2.1 and the precision agriculture example in Section 2.6.2, we parametrized the computations by a set of σ computations parameters $c_i^\sigma := \{c_{i,\rho+1}, \ldots, c_{i,\rho+\sigma}\}$, where ρ is the number of path parameters. We parametrize the computations that impact the overall energy consumption (as we described in Definition 2.1.1). For instance, in the agricultural scenario, we parametrize the computation object detection. We mean by parametrization that we enable the computations to be dynamically replanned with an appropriate choice of the parameters, and run on, e.g., a lower/higher rate requiring lower/higher power. Practically, this means that if our system is composed of σ computations, the configuration of computations parameters $c_i^\sigma(\mathcal{T})$ for each stage i over time $\mathcal{T} := [t_0, t_l]$ (where t_0, t_l are respectively the first time instant and the time instant when the aerial robot reaches the final point $\mathbf{p}_{\Gamma_l}$ in Definition 2.3.2) is a schedule of the computations. In Chapter 5, we will derive an energy-aware schedule via the model that we propose in this section. In the remainder of this section, we derive an energy model that maps any choice of parameters c_i^σ to the power at any $t \in \mathcal{T}$, extending the past work on energy modeling for heterogeneous computing hardware.

In Section 4.1.1, we summarize the heterogeneous elements modeling from Section 3.1.1 and outline our approach, which consists of two layers. We detail the layers in Sections 4.1.2–4.1.3 and describe an automated modeling tool we developed in Section 4.1.4 along with its configuration specification in Section 4.1.5. The tool works well for computing hardware equipped with internal power meters, yet, some computing hardware does not provide any. We discuss how to model such hardware also in Section 4.1.4.

4.1.1 Model for the heterogeneous elements

We saw in Chapter 2 that traditionally, models for computing hardware focus on a specific computing element, such as CPU or GPU, or model the elements heterogeneously. The models provide a regression function (measuring the power consumption over time), analytical (using some architectural parameters), or other expressions to infer future energy consumption. The

regression functions, analytical or other expressions might depend on low-level architectural parameters and be employed in an energy-efficient selection of such parameters, including voltage and frequency. Alternatively, they might depend on high-level parameters (such as the computations parameters c_i^σ in this work) and be employed in an energy-aware configuration of software (and hardware), e.g., the tasks allocation on the CPU cores. An expression depending on a configuration is more common for heterogeneous models, as we summarized in Table 3.1. We focus on these models; in Chapter 1 and formally in Definition 2.1.1, we assumed the aerial robot caries heterogeneous computing hardware for energy-demanding computations. We refer to Section 4.1 for an extensive discussion of some energy modeling approaches in the literature for CPUs and GPUs powered and heterogeneous computing hardware.

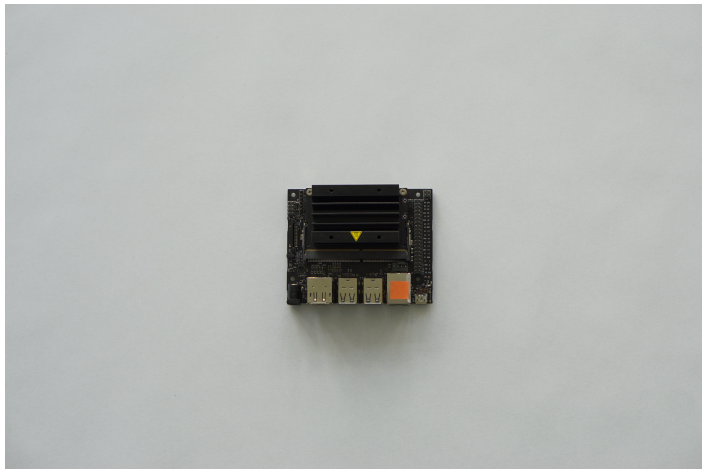


Figure 4.1. NVIDIA Jetson Nano heterogeneous computing hardware that we model the computations energy on. The hardware in the image includes the Jetson Developer Kit board that the Nano heterogeneous board is mounted on and has a total size of 100x80 millimeters and a weight of approx. 140 grams.

Our approach is based on our work (Seewald, Schultz, Ebeid, et al., 2021) on heterogeneous computing devices' energy modeling and generic, modeling virtually a wide range of heterogeneous computing hardware energy consumption in real use-cases with little users effort. We use statistical methods to derive a regression-based model segmented into two layers. In the first, the measurement layer, we map the time to the power, and in the second, the predictive layer, we map the computations configuration $c_i^\sigma(t)$ to the power. To generate a model, our approach inputs a user-defined configuration file, which simply specifies the computations along with the computation parameter constraint sets $\mathcal{S}_{i,k}$ in Definition 2.3.1, per each computation $k \in [\sigma]_{>0}$ and stage $i \in [l]_{>0}$ (or $[n]_{>0}$ if the computations are specified along the primitive paths and reiterated when the aerial robot reaches $\mathbf{p}_{\Gamma_n}$ with a shift $\mathbf{d}$ in Figure 2.3). From the configuration, an automated modeling tool termed `powprofiler` measures the energy consumption of a discrete set of configurations $c_{i,\rho+j} \in \mathcal{S}_{i,j}$, per each computation parameter $j \in [\sigma]_{>0}$ and derives the measurement layer. The tool allows merging then a set of measurement layers into

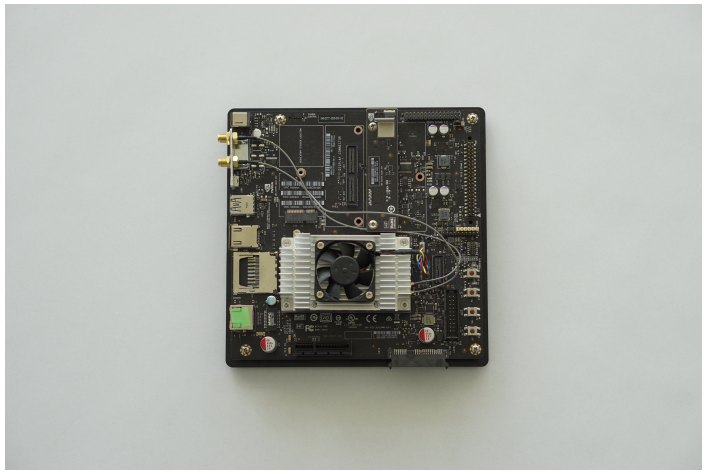


Figure 4.2. NVIDIA Jetson TX2 heterogeneous computing hardware mounted on a Jetson Developer Kit board as Nano in Figure 4.1 with a total size of 170x170 millimeters and a weight of approx. 512 grams.

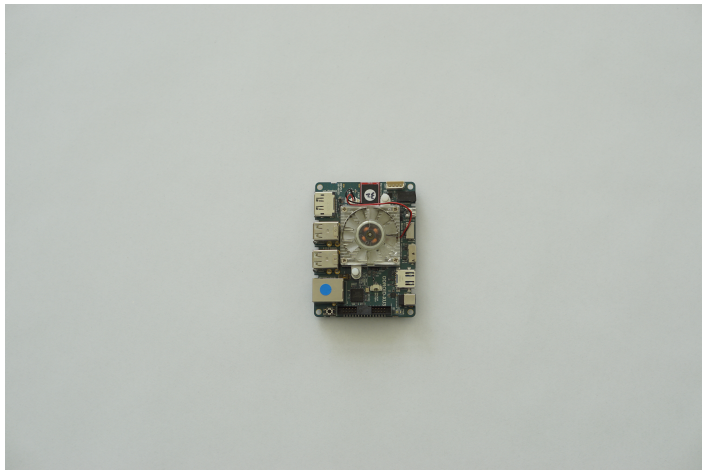


Figure 4.3. ODROID XU3 heterogeneous computing hardware with a total size of 94x70 millimeters and a weight of 70 grams.

a predictive layer via linear regression. As heterogeneous computing hardware, we use different devices, such as NVIDIA Jetson Nano, TX2, and TK1 in Figures 4.1–4.2 and 4.4 and ODROID XU3 in Figure 4.3. We summarize the computing hardware that we explicitly consider in this work in Table 4.1. These devices are commonly employed in robotics literature to power complex computations. For instance, Jetson TX2 has been employed in path planning (Dharmadhikari et al., 2020; Ryou et al., 2018), simultaneous localization and mapping (SLAM) (Aldegheri et al., 2019), and object detection via convolutional neural networks (CNNs) (Andrew et al., 2019). Jetson Nano in SLAM and CNNs (Alexey et al., 2021; T. Peng et al., 2019; L. Wang et al., 2020),

Hardware	CPU	GPU	Memory	Sensor
NVIDIA Jetson Nano	-A57	NVIDIA Maxwell	4 GB LPDDR4 RAM	✓
NVIDIA Jetson TX2	-A57	NVIDIA Pascal	8 GB LPDDR4 RAM, 32 GB NV	✓
NVIDIA Jetson TK1	-A15	NVIDIA Kepler	1 GB DDR3L RAM, 16 GB NV	✗
ODROID XU3	-A15, -A7	MALI	2 GB LPDDR3 RAM	✓

Table 4.1. Mobile computing hardware explicitly analyzed in this work. All the CPUs are ARM Cortex. The majority of the embedded boards provide random access memory (RAM), and some a non-volatile (NV) memory. All the boards have energy measuring capabilities except NVIDIA Jetson TK1.

and ODROID XU3 (Bhat et al., 2019; Giusti et al., 2016; Papachristos et al., 2015) and Jetson TK1 (Gong et al., 2016; Holper et al., 2017) in similar applications. Although we report some in this paragraph, heterogeneous embedded boards power computations in many other mobile robots use-cases, thus possibly broadening the applicability of our work, which we further in Chapter 7. In the remainder of this section, we detail our energy model for the heterogeneous computing hardware.

4.1.2 Measurement layer

The measurement layer is the basic building block of the computations energy model: one or more measurement layers form the predictive layer–the model output–which we describe in Section 4.1.3. For a specific computations parameters configuration $c_i^\sigma(t)$, the measurement layer maps the time $t \in [t_0, t_f] := \mathcal{T} \subset \mathbb{R}_{>0}$ (the final and initial time instants t_0, t_f are given) to the power measured in watts, the energy measured in joules (watts per unit of time), and the battery state of charge (SoC) expressed in percentages. The measurement layer thus provides a primitive model for c_i^σ of power, energy, and SoC over a given time interval. The derivation of the layer is automated with `powprofiler`, the modeling tool that we describe in detail in Section 4.1.4, which outputs the layer after executing c_i^σ on $\mathcal{T}$. Additionally, the model (and the `powprofiler` tool) can output the triplet of metrics from the measurement layer per each energy sensor: some computing hardware that we analyze indeed provide different sensors for different computing elements, i.e., NVIDIA Jetson TX2, Nano, and ODROID XU3 boards all provide energy-sensing capabilities for CPU, GPU, overall, and/or memory.

Let us define the measurement layer formally, assuming there are one or more energy sensors or other energy measuring devices (these include, e.g., internal power resistors or shunt resistors, amperometers, and multimeters) in Definition 4.1.1.

Definition 4.1.1: Measurement layer

Given a specific energy measuring device, computations parameters configuration $c_i^\sigma(t)$, and an initial and final time instant t_0, t_f such that $t \in \mathcal{T} := [t_0, t_f]$, the *measurement layer* is the function $\mathbf{g} : \mathbb{Z}_{>0} \times \mathbb{Z}^\sigma \times \mathcal{T} \to \mathbb{R}^3$. It returns the power in watts, energy in joules, and SoC in percentages of an energy measuring device, a configuration of computations parameters, and time interval.

The measurement layer physically samples the computing hardware for $\mathcal{T}$ and returns the metrics, but sampling-to-completion is also possible, where a configuration runs up until it terminates rather than for a given interval. In the latter eventuality, $\mathcal{T}$ is $\emptyset$.

As an instance of the measurement layer, let us return briefly to the precision agriculture example in Section 2.6, which describes the agricultural use-case (the aerial robot detects ground hazards and communicates the detections to a ground station). The computations parameters in Section 2.6.2 are $c_{i,2}$, the frames per second (FPS) rate, and $c_{i,3}$, encryption or no encryption of the robot—ground station data link. The configurations $c_{i,2}(t) \in \mathcal{S}_{i,2}$, $c_{i,3}(t) \in \mathcal{S}_{i,3}$ have any value within the constraint sets in Equations (2.19–2.21). The constraint sets are the same in each stage i in the plan Γ except for the circles where the aerial robot travels the turns out of the boundaries, and the detections are inhibited. To build the measurement layer, we can discretize the configurations with a given δ_1, δ_2 for parameters $c_{i,2}$ and $c_{i,3}$. The measurement layer is then built by sampling the power for $\mathcal{T}$, one for all the possible configurations

$$c_i^\sigma := \{c_{i,2}, c_{i,3} \mid \forall j, k \in \mathbb{Z},\ \underline{c}_{i,2} + j\delta_1 \in \mathcal{S}_{i,2}, \underline{c}_{i,3} + k\delta_2 \in \mathcal{S}_{i,3}\}. \tag{4.1}$$

A value can be, e.g, $\delta_1 = \delta_2 = 2$, so that there are ten configurations and consequently ten measurement layers. The `powprofiler` tool automatically builds the layers, storing the results in comma-separated values (CSV) files (we see further the tool in Section 4.1.4). Equation (4.1) provides a way to sample the search space linearly, but other sampling strategies are equally possible, such as exponential sampling

$$c_i^\sigma := \{c_{i,2}, c_{i,3} \mid \forall j, k \in \mathbb{Z},\ {\delta_1}^j \in \mathcal{S}_{i,2}, {\delta_2}^k \in \mathcal{S}_{i,3}\}, \tag{4.2}$$

where δ_1, δ_2 are now bases, or random sampling with the condition merely $c_{i,j} \in \mathbb{Z}_{>0}$. Currently, the `powprofiler` tool supports automated linear and exponential sampling and complex sampling formed by different sampling strategies (Seewald, Schultz, Ebeid, et al., 2021), e.g.,

$$c_i^\sigma := \{\{c_{i,2} \mid \forall k \in \mathbb{Z},\ \underline{c}_{i,2} + k\delta_1 \in \mathcal{S}_{i,2}\}, \{c_{i,3} \mid \forall k \in \mathbb{Z}, {\delta_2}^k \in \mathcal{S}_{i,3}\}\}. \tag{4.3}$$

Parameter ranges $(\mathcal{S}_{i,2}, \mathcal{S}_{i,3})$ choice is dictated by the range at which the computations run at runtime, whereas δs choice is made so that the modeling terminates in a reasonable amount of time (Seewald, Schultz, Ebeid, et al., 2021).

With the measurement layer described in this section, we know the power and other energy metrics of the (measured) configurations. However, we want to predict the energy consumption for any configuration of parameters in the constraint sets (and not only the sampled ones). We

address this latter requirement in the next section, merging the measurement layers with linear regression.

4.1.3 Predictive layer

The predictive layer describes coarse-grained metrics, such as the power over FPS rate, mapping them to the metrics from the measurement layer, thus providing energy data for each configuration of parameters (or for each scheduling policy). To this end, it uses the set of measurement layers, building a two-by-two linear regression between consecutive layers. In the precision agriculture example with ten measurement layers, the predictive layer consists of regression between data points $\{c_{i,2}, c_{i,3}\}$ for all the possible computations parameters (recall that a measurement later corresponds to a sampled computations configuration), opposed to the sampled ones in Section 4.1.2. Definition 4.1.2 details the resulting model.

Definition 4.1.2: Predictive layer

Given a specific energy measuring device and computations parameters configuration $c_i^\sigma(t)$ *predictive layer* is the function $g : \mathbb{Z}_{\geq 0} \times \mathbb{Z}^\sigma \rightarrow \mathbb{R}^3$. It returns the power in watts, energy in joules, and SoC in percentages of any configuration of parameters within the constraint sets.

Analogously to the measurement layer, there can be various energy measuring devices, resulting in multiple triples of energy, power, and SoC, one per device. The predictive layer returns the same metrics of the measurement layer in Definition 4.1.1, without physically sampling the computing hardware but using the stored measurement layers. Indeed due to a potentially large search space in computational energy modeling, it is critical to infer the energy properties of the entire search space from a subset of all the possible samples (Bailey et al., 2014; B. C. Lee and Brooks, 2006a,b). The linear regression to infer such properties utilizes a method that we term the approximation method (Seewald, Schultz, Ebeid, et al., 2021). It builds a linear regression between two adjacent data points rather than merely for all the data points. Indeed we do not assume an apriori knowledge of the computations energy evolution with explicit models for all the data points such as linear or exponential, but rather model the energy linearly on tuples of data points (Seewald, Schultz, Ebeid, et al., 2021).

Given an (unsampled) configuration c_i^σ, the predictive layer is approximated with

$$g(c_i^\sigma) = (\mathbf{g}(\lceil c_i^\sigma \rceil, \mathcal{T}_1) - \mathbf{g}(\lfloor c_i^\sigma \rfloor), \mathcal{T}_2)(c_i^\sigma - \lfloor c_i^\sigma \rfloor)/(\lceil c_i^\sigma \rceil - \lfloor c_i^\sigma \rfloor) + \mathbf{g}(\lfloor c_i^\sigma \rfloor, \mathcal{T}_2), \tag{4.4}$$

where $\lceil c_i^\sigma \rceil$, $\lfloor c_i^\sigma \rfloor$ are the two adjacent measurement layers of the computations configuration c_i^σ (e.g., if we use $\delta_1 = 2$ in Equation (4.1), c_i^σ with just the parameter $c_{i,2}$ has $\lfloor c_i^\sigma \rfloor$ equal to two and $\lceil c_i^\sigma \rceil$ to four), and $\mathcal{T}_1, \mathcal{T}_2$ are the two time intervals in the measurement layer for configurations $\lfloor c_i^\sigma \rfloor$, $\lceil c_i^\sigma \rceil$ respectively. We illustrate the principle in Figure ??.

4.1.4 The `powprofiler` tool

The `powprofiler`[1] tool is an automated profiling and modeling utility that generates the measurement layers for a discrete set of possible computations configurations (automated profiling) and merges these layers later, providing the predictive layer (modeling). We proposed an early version of the tool earlier in our work (Seewald, Schultz, Ebeid, et al., 2021; TeamPlay Consortium, 2019), which we extended later to support per-component energy modeling in a dataflow computational network (Seewald, Schultz, Roeder, et al., 2019), and integrated (Zamanakos et al., 2020) with Robot Operating System (ROS) middleware (Quigley et al., 2009). The tool is written in C++ and distributed under an MIT license. It supports all the computing hardware explicitly mentioned in this work, i.e., NVIDIA Jetson TX2, Nano, and TK1 and ODROID XU3 in Table 4.1. It is predisposed further for extensibility supporting possibly any Linux-based computing hardware that provides energy measuring devices or that alternatively provide a mechanism to measure the energy with an external device.

The tool uses an object-oriented programming approach (Stroustrup, 1988; Wegner, 1990), where each computing hardware has its own class that inherits from the class `sampler` the functions `get_sample` and `dryrun`. The former function returns a power measurement from all the measuring devices on specific computing hardware (power for the measurement layer in Section 4.1.2 for, e.g., CPU, GPU, overall, etc…), and the latter simply attempts to read from the measuring devices returning a boolean value indicating if the attempt was successful. The tool contains classes `sampler_tx2`, `sampler_nano`, and `sampler_odroid` already implementing the necessary utilities to store the models from the computing hardware that we explicitly analyze.

The function `get_sample` further relies on a specific data type, which we term `vectorn`. Each value in `vectorn` has its flag, indicating the metric and the measuring device. For instance `power_cpu`, `soc_gpu` are two flags indicating that the metric is for the power and the measuring device is of the CPU and the SoC of the GPU respectively. The enumeration `vectorn_flags` contains all the flags. The tool stores a set of `vectorn`s sampled at discrete intervals (`powprofiler` is highly personalizable, allowing to change the frequency via the configuration specification in Section 4.1.5) in another structure termed `pathn`. Each `pathn` corresponds to a measurement layer in Section 4.1.2. The tool further provides mechanisms, such as the overload of the constructor, to automatically load the layers from a previously stored CSV file. Internally the tool stores `pathn`s (the measurement layers) in a wrapper, `model_1layer`, which contains information such as the parameters configuration and the set $\mathcal{T}$; `model_2layer` is then another internal structure that returns the predictive layer in Section 4.1.3. In this setting, NVIDIA Jetson TK1 computing hardware does not include any energy measuring device. `powprofiler` allows an external device in the sense that it can import data in the model directly through the overload of the constructor into a measurement layer (and therefore a set of measurement layers into a predictive layer). An early instance of our work (Seewald, Ebeid, et al., 2019) consisted of three hardware units for the purpose of this latter energy modeling of the TK1 computing hardware, similarly to another study in the literature (Calore et al., 2015). The main hardware unit in the early instance was

[1] The tool can be retrieved from `https://github.com/adamseew/powprofiler`

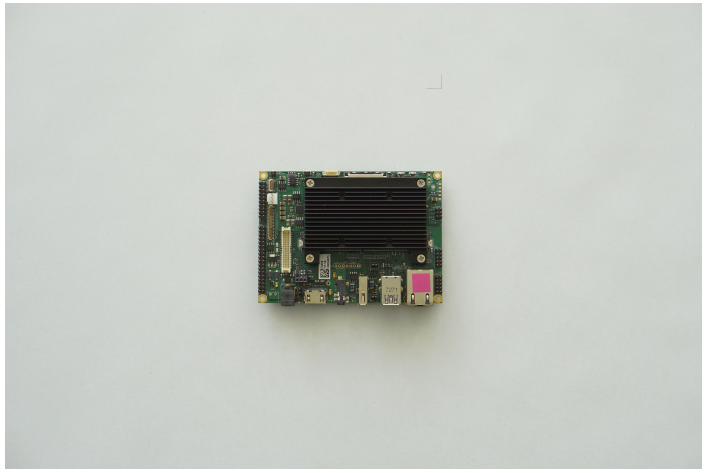


Figure 4.4. NVIDIA Jetson TK1 heterogeneous computing hardware mounted on a Toradex Ixora carrier board with a total size of 125x95 millimeters and a weight of approx. 160 grams.

the computing hardware itself, whereas the others were a multimeter and a workstation that interprets the data from the multimeter for subsequent processing by the tool (Seewald, Schultz, Ebeid, et al., 2021).

The tool further interoperates with ROS middleware, generating a measurement layer for configurations of computations implemented in the middleware. It can be imported in an existing project as a library; in C/C++ by simply adding the preprocessor's directive `#include` with `<powprof/async.h>`. In a setting where an energy-expensive ROS node is a computation (we implement both the CNN detection and encryption in Section 2.6.2 as ROS nodes), the user simply instances `model_1layer` with a specific computations configuration and calls function `start` to start profiling, and `stop` to stop. The latter then returns a measurement layer. The modeling can, in this fashion, happen online by running different computations configurations and generating an appropriate computations energy model corresponding to a realistic run-time computations load.

Alternatively to ROS middleware, the tool runs from a configuration specification, detailing each computation, the constraints sets, and the δs, along with some other, model-specific details. We discuss such configuration specification in the next section.

4.1.5 Configuration specification

The configuration specification is simply a way to communicate the plan Γ to the `powprofiler` tool, along with some other model-specific details. To run `powprofiler` with a configuration specification, the user invokes the command `powprofile`, followed by the path of the configuration. If, for instance, the configuration specification is stored in `config.cfg` in the current directory, the user invokes `powprofile config.cfg`. The tool then parses the configuration specification to reconstruct the computations configurations and the constraints sets, to automatically gener-

ate measurement layers and consequently provide a predictive layer. The configuration specification starts with the line `[settings]` that indicates to `powprofiler` all the following lines are a configuration specification. In the lines that follow, it contains a set of key-value properties delimited by an equal char. The property `frequency` indicates the frequency measured hertz the tools samples at (e.g., with ten seconds, the property is set as `frequency=10`). The property `h` is the integration step the battery model integrates at (we discuss further the battery model in Section 4.2). The property `directory` indicates the path where the models are stored. Additionally, the tool allows an arbitrary number of commands and white spaces, and the parser does not require a specific indentation. Any data followed by char `#` are ignored up to the next line, allowing to write eventual comments.

The following set of lines specifies the configuration c_i^σ for each computation parameter $c_{i,\rho+1}, c_{i,\rho+2}, \ldots, c_{i,\rho+\sigma}$, starting with the line `[components]`, followed by `[component.computation]` where `computation` is a string uniquely identifying each computation (there cannot be two computations with the same string, but the name is arbitrary). Per each computation, the configuration file then contains a set of key-values properties. The property `src` indicates the executable source of the computation. If `powprofiler` runs with a configuration specification rather than a library, we assume the source accepts as arguments the configurations, e.g., $c_{i,\rho+1}, \ldots$ along with other eventual arguments. The following properties then specify these arguments, ordered as they appear in the configuration specification. The property `range` indicates that the argument is a computation parameter. Let us assume the parameter is $c_{i,\rho+1}$. If it is sampled linearly as in Equation (4.1), the value contains $\underline{c}_{i,\rho+1}$, $\overline{c}_{i,\rho+1}$, and δ delimited by commas, where δ is the step to sample $c_{i,\rho+1}$ in a reasonable amount of time in Equation (4.1). If it is sampled exponentially as in Equation (4.2), the value contains the same data as before, but for δ that is expressed `pow(`δ`)` and δ is the base. Additional properties are then: `fixed` that indicates another eventual argument the computation might have (that is not a computation parameter), and `runtime` the value $t_f - t_0$. If `runtime` is not specified, the tool assumes $\mathcal{T} = \emptyset$.

4.2 Battery Model

The battery model is an abstraction that predicts how the draining power at future time instants–due to varying computations and motion load–affects the SoC of an aerial robot's battery. Generally, battery SoC is the most important measure for battery management, yet, it cannot be directly measured (Xia et al., 2015). There are numerous approaches to formulate its model, and we discuss the most common ones in the literature in Section 3.2, including physical, hybrid, empirical, mixed, and abstract models (R. Rao et al., 2003). In this section, we then use the past literature to derive a battery model of an aerial robot's battery. The model that we derive is an abstract model. Such models do not require detailed information about, e.g., battery chemistry; nonetheless, they accurately predict future SoC at a relatively low computational complexity compared with complex and exhaustive models, such as physical models (R. Rao et al., 2003).

We detail the equivalent electrical circuit in Section 4.2.2; it is the building block of our battery model. We then provide some further information on the implementation of the equivalent electrical circuit in the `powprofiler` tool in Section 4.2.2. We will use the battery model in Section 5.3.1 to define the output constraint in Definition 5.3.1

4.2.1 Equivalent electrical circuit

Equivalent electrical circuits are abstract models, frequently referred to as battery equivalent circuit models (ECMs). They are common in the literature for battery SoC estimation (C. Zhang, Allafi, et al., 2018) and treated in numerous studies relative to Li-ion rechargeable battery cells (Hasan et al., 2018; Hinz, 2019). Although there are more accurate models to predict SoC for these batteries from a given power and time trajectories–namely physical or electrochemical models (R. Rao et al., 2003)–for mobile robots and more in general for resource-constrained systems it is usually required to balance the models' complexity and the accuracy (Hasan et al., 2018; R. Rao et al., 2003). In these constrained systems, ECMs have relatively good modeling accuracy and easy implementation (Hasan et al., 2018; Madani et al., 2019; C. Zhang, Allafi, et al., 2018; F. Zhang, Liu, and Fang, 2009; F. Zhang, Liu, Fang, and H. Wang, 2012). ECMs use different constructs to model the battery SoC from several parameters. These constructs include RC (resistor-capacitor) circuits for dynamic loads, resistors for internal resistances, and other components (Hamza and Ayanian, 2017). Their complexity depends on the level of detail required and the parameters involved in modeling. The parameters are usually estimated with empirical data (C. Zhang, K. Li, et al., 2014). For what concerns the specific battery chemistries to be modeled, we focus on Li-ion batteries. These batteries have broad applications involving electric vehicles, mobile, and aerial robots (Hasan et al., 2018; Shi and Zhao, 2006; Xia et al., 2015; C. Zhang, K. Li, et al., 2014), due to their characteristics such as low self-discharge rate, absence of memory effect, and high power and energy density (C. Zhang, K. Li, et al., 2014).

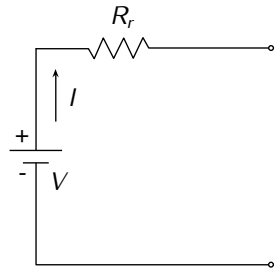


Figure 4.5. Equivalent electrical circuit for battery modeling with one resistor, representing the internal battery resistance.

Here, we propose a simplistic battery model to model a Li-ion battery of an aerial robot in flight, focusing on lesser complexity rather than accuracy. The battery SoC changes–when computations and motion generate a current to be drawn from the battery–according to the equa-

tion (Hasan et al., 2018; C. Zhang, Allafi, et al., 2018)

$$\dot{b}(y(t)) = -I(y(t))/Q_c, \tag{4.5}$$

where $Q_c \in \mathbb{R}$ is the battery constant nominal capacity measured in amperes per hour, $I(y(t)) \in \mathbb{R}$ is the internal current that we derive later in this section, and $y(t) \in \mathbb{R}_{\geq 0}$ is a power drawn, i.e., the power needed for the computations and the motion. If we use the computations energy model in Section 4.1, it is the power metric in Definition 4.1.2 or the value of the measurement layer in Definition 4.1.1 for each time step.

We propose a simplistic ECM with an internal resistance from the literature (He et al., 2011; Hinz, 2019; Mousavi G. and Nikdel, 2014) in Figure 4.5, sometimes termed the "Rint" model (He et al., 2011; Hinz, 2019). The circuit models the battery simply as a perfect voltage source connected with a resistor $R_r \in \mathbb{R}$ measured in ohms, representing the internal battery resistance. The voltage $V \in \mathbb{R}$ measured in volts is the internal battery voltage, which depends on SoC (Hasan et al., 2018) and can be retrieved from a battery data sheet (Hinz, 2019), and I is the current running through the circuit that depends on the power requirements of the load.

The voltage on the extremes of the ECM then respects

$$V_e = V - R_r I, \tag{4.6}$$

where $V_e \in \mathbb{R}$ is the external battery voltage at the extremes of the circuit in Figure 4.5. If we assume that the voltage needed by the computations and motion is stable, let's call it $V_s \in \mathbb{R}$ and is measured in volts, and that the current required by the load (computations and motion) is I_l, we can write

$$V_s I_l = V_e I, \tag{4.7}$$

using simply Kirchhoff's circuit laws (the power into the load should exactly match the power out). Combining the Equations (4.6–4.7), we obtain the quadratic expression $R_r I^2 - VI + V_s I_l = 0$, which leads to

$$I(y(t)) = \left(V - \sqrt{V^2 - 4R_r y(t)} \right) / (2R_r), \tag{4.8}$$

where $I_l := y(t)/V_s$ is the current of the load depending on the computations and motion power $y(t)$ at a given time instant t in Equation (4.5). Furthermore, we take the negative solution of the quadratic expression: when I_l is zero, I should also be zero. With the internal current in Equation (4.8) combined with the battery SoC in Equation (4.5), we can model how the computations and motion power trajectory $y(t)$ on $t \in \mathcal{T} := [t_0, t_f]$ for given initial and final time instants (t_0, t_f respectively) affects the battery. In one of our earlier intuitions (Seewald, Schultz, Ebeid, et al., 2021), we expected a constant energy load to result in a better overall SoC compared to, e.g., a spiked one, even if the two have the same overall energy. The model above confirms this intuition. In Figure ??, we show the evolution of I in Equation (4.8) for a given linear load I_l from zero to approximately one-third, assuming $V = V_s = R_r$ all one. The curve in the plot bends upwards for the plotted range: a line between two points will always be above the curve; it implies that a constant load is to be preferred compared to a load that repeatedly changes from high to low.

There are also more complex ECMs in the literature (Hasan et al., 2018; Hinz, 2019), which add additional elements to, e.g., account for the changes in the load current. One such ECM is the Thevenin model and the Thevenin-based model (M. Chen and Rincon-Mora, 2006; Hasan et al., 2018; Hinz, 2019; Mousavi G. and Nikdel, 2014; Salameh et al., 1992; C. Zhang, Allafi, et al., 2018). Sometimes termed the dual-polarization model (He et al., 2011), we illustrate the

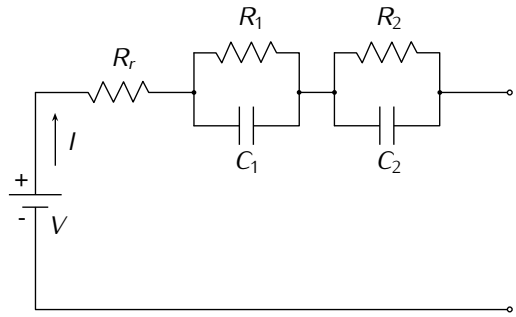


Figure 4.6. Thevenin-based equivalent electrical circuit for battery modeling with one resistor and two RC internal elements. The two elements add some complexity, making the model able to account for changes in the load current.

ECM in Figure 4.6. It models further details, such as the short-term transient behavior with the first RC element (R_1, C_1) and the long-term transient behavior with the second RC element (R_2, C_2) (Hinz, 2019). It is particularly suitable to model the polarization characteristic of Li-ion battery cells (He et al., 2011).

4.2.2 Battery model in the `powprofiler` tool

The `powprofiler` tool allows automated battery modeling and directly derives battery SoC for each measurement layer in Section 4.1.2. Indeed in Definition 4.1.1, the function **g** returns a triplet of values, including the SoC. For the predictive layer in Definition 4.1.2 in Section 4.1.3, the tool similarly outputs the SoC for a given configuration of parameters and energy sensor or other energy measuring device. To this end, it implements the simplistic equivalent electrical circuit in Figure 4.5 from the literature (He et al., 2011; Hinz, 2019; Mousavi G. and Nikdel, 2014), with the class `soc_1resistor`, where the constructor accepts in input the parameters I_l, V, R_r, V_s and Q_c that we discussed in Section 4.2.2. The current load I_l is expressed via the data type `pathn` in Section 4.1.4. One can implement a similar battery model, e.g., the Thevenin-based ECM in Figure 4.6, by simply inheriting from class `first_derivative` the function `get_value`. It returns the modeled battery SoC at the next time instant from an independent variable, i.e., time, and a dependent variable represented via the data type `soc_1resistor` in Section 4.1.4.

Internally, the tool implements a numerical simulator based on the Runge-Kutta methods for numerical integration in Section A.4.2. One can then personalize the size of the integration step $h \in \mathbb{R}_{>0}$ via the property `h` in the configuration specification in Section 4.1.5 (a typical practical value of such property is, e.g., one hundredth). The tool first derives a model for the power and

energy and later numerically simulates the battery model via the equivalent electrical circuit in Figure 4.5 adjoining the battery SoC.

4.3 Energy Model of the Motion

In this section, we implement a model for the motion of aerial robots in coverage path planning (CPP). We first derive a differential periodic energy model and provide a formal proof in Section 4.3.1. This model server to model the motion. We enhance the model with the path and computations parameters in Section 4.3.2, to predict the energy consumption of a given configuration of parameters. We explain how we convert the parameters into actual energy consumption in Section 4.3.3, and discuss the model's behavior for aperiodic power evolutions in Section 4.3.4.

Let us suppose the aerial robot is operating in an autonomous scenario, planning the coverage and scheduling some computations for detections and encryption in Section 2.6. It expectedly

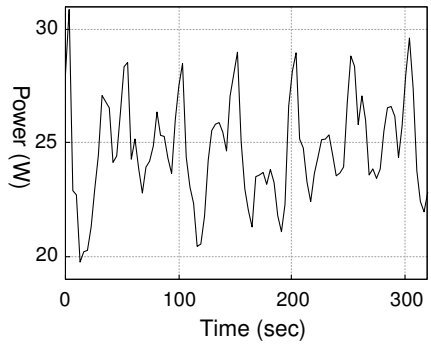


Figure 4.7. Empirical energy data of the aerial robot in Figure 1.6 in coverage planning. The data shows that the energy signal is periodic over time as the fixed-wing aerial robot reiterates a set of paths.

iterates some paths (the primitive paths) for CPP and schedules the computations periodically, as we outlined in Section 1.4, and further backed with the concept of primitive paths in Definition 2.4.1. Since the paths and tasks are periodically iterated over time, we expect the energy to evolve similarly. This assumption is further supported by our previous contribution on the topic (Seewald, Garcia de Marina, Midtiby, et al., 2020), showing that a Fourier series of a given order can model the energy of the aerial robot (in the contribution, we used the order three). We will ease the assumption of the periodic evolution in practice to periodic with disturbance or aperiodic evolutions in Section 4.3.4. We motivate the choice of a periodic energy model further with some empirical energy data of the Opterra fixed-wing aerial robot flying the agricultural scenario in Figures 4.7–4.8. The data shows the robot's energy along its frequency spectrum. The latter is centered at zero frequency, and peaks at four hundred kilo decibels. The peak visually depicts the shift on the power axis in Figure 4.7 (and further backs the choice of the Fourier series of third order in our earlier work, illustrating that the power evolution needs approximately

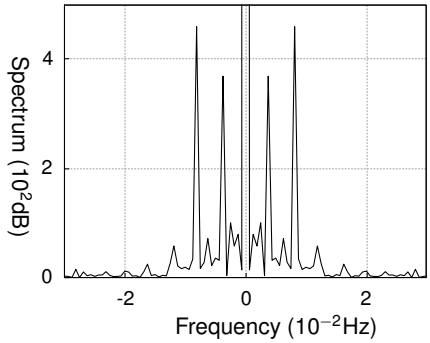


Figure 4.8. Frequency spectrum of the empirical energy data in Figure 4.7 where the aerial robot does the coverage planning.

three frequencies to be modeled). To obtain the spectrum in frequency space, we computed the Fourier transform.

4.3.1 Derivation of the differential periodic model

In the remainder of this section, we refer to the power (or instantaneous energy consumption) evolution simply as the energy signal. We model the signal using energy coefficients vector $\mathbf{q} \in \mathbb{R}^m$ that characterize the energy signal. We derive the coefficients from Fourier analysis: the size of the vector m is then related to the order of the series. We prove a relation between the energy signal and the energy coefficients in Lemma 4.3.1.

First, let us consider a periodic energy signal of period $T \in \mathbb{R}_{>0}$, and a Fourier series of an arbitrary order $r \in \mathbb{Z}_{\geq 0}$ for the purpose of modeling the signal

$$h(t) = a_0/T + (2/T)\sum_{j=1}^{r}\left(a_j \cos \omega j t + b_j \sin \omega j t\right), \tag{4.9}$$

where $h : \mathbb{R}_{\geq 0} \to \mathbb{R}$ maps time to the power, $\omega := 2\pi/T$ is the angular frequency, and $a_0, a_j, b_j \in \mathbb{R}$ the Fourier series coefficients $\forall j \in [r]_{>0}$.

The energy signal can be modeled by Equation (4.9) and by the output of a linear model

$$\dot{\mathbf{q}}(t) = A\mathbf{q}(t) + B\mathbf{u}(t), \tag{4.10a}$$
$$y(t) = C\mathbf{q}(t), \tag{4.10b}$$

where $y(t) \in \mathbb{R}$ is the power at time instant t. We discuss matrices A, C, and B in Equation (4.12), Equation (4.14), and Equation (4.46) respectively (we discuss the nominal control $\mathbf{u}$ in Section 4.3.2).

The state $\mathbf{q}(t)$ contains the energy coefficients

$$\mathbf{q}(t) := \begin{bmatrix} \alpha_0(t) & \alpha_1(t) & \beta_1(t) & \cdots & \alpha_r(t) & \beta_r(t) \end{bmatrix}', \tag{4.11}$$

where $\mathbf{q}(t) \in \mathbb{R}^m$ with $m = 2r + 1$. We will estimate the value of $\mathbf{q}$ with a state estimator in Chapter A.3.

The state transition matrix

$$A = \begin{bmatrix} 0 & 0^{1\times 2} & 0^{1\times 2} & \dots & 0^{1\times 2} \\ 0^{2\times 1} & A_1 & 0^{2\times 2} & \dots & 0^{2\times 2} \\ 0^{2\times 1} & 0^{2\times 2} & A_2 & \dots & 0^{2\times 2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0^{2\times 1} & 0^{2\times 2} & 0^{2\times 2} & \dots & A_r \end{bmatrix}, \tag{4.12}$$

where $A \in \mathbb{R}^{m\times m}$. In matrix A, the top left entry is zero, the diagonal entries are $A_1, \dots, A_r$, the remaining entries are zeros. Matrix $0^{i\times j}$ is a zero matrix of i rows and j columns. The submatrices $A_1, A_2, \dots, A_r$ or generically

$$A_j := \begin{bmatrix} 0 & \omega j \\ -\omega j & 0 \end{bmatrix}, \tag{4.13}$$

$\forall j \in [r]_{>0}$. The output matrix

$$C = (1/T) \begin{bmatrix} 1 & 1 & 0 & \cdots & 1 & 0 \end{bmatrix}, \tag{4.14}$$

where $C \in \mathbb{R}^m$.

The linear model in Equation (4.10) allows us to include the control in the model of Equation (4.9), a concept that we build upon further in Section 4.3.2 where we merge the computations and motion energies. In the remainder, we formally prove an important concept in our work: we can use Equation (4.10) to model the energy signal of an aerial robot, assuming the robot iterates periodically a set of paths and computations to achieve a given space coverage. We already know that we can model a periodic energy signal with Equation (4.9), so we prove the equivalence and equality of the models in Equation (4.9) and Equation (4.10).

Lemma 4.3.1: Signal, output equality

Suppose control $\mathbf{u}$ is a zero vector, matrices A, C are described by Equations (4.12–4.14), and the initial guess at a given time instant $t_0 \in \mathbb{R}_{>0}$ $\mathbf{q}(t_0)$ is

$$\mathbf{q}(t_0) = \begin{bmatrix} a_0 & a_1/2 & b_1/2 & \cdots & a_r/2 & b_r/2 \end{bmatrix}'.$$

Then, the signal h in Equation (4.9) is equal to the output y in Equation (4.10).

Proof. The proof justifies the choice of the items of the matrices A, C and the initial guess $\mathbf{q}(t_0)$ in Equations (4.11–4.14). We write these elements such that the coefficients of the series $a_0, \dots, b_r$ are the same as the coefficients of the state $\alpha_0, \dots, \beta_r$.

Let us re-write the Fourier series expression in Equation (4.9) in its complex form with the well-known Euler's formula

$$e^{it} = \cos t + i \sin t, \tag{4.15}$$

where i is the imaginary unit. With $t = \omega jt$, we find the expression for

$$\cos \omega jt = (e^{i\omega jt} + e^{-i\omega jt})/2, \tag{4.16a}$$

$$\sin \omega jt = (e^{i\omega jt} - e^{-i\omega jt})/(2i), \tag{4.16b}$$

by substitution of $\sin \omega jt$ and $\cos \omega jt$ respectively. This leads to (Kuo, 1967)

$$\begin{aligned} h(t) = a_0/T + (1/T) \sum_{j=1}^{r} e^{i\omega jt}(a_j - ib_j) + \\ (1/T) \sum_{j=1}^{r} e^{-i\omega jt}(a_j + ib_j). \end{aligned} \tag{4.17}$$

The solution at time t of the model in Equation (4.10) under the assumptions in the lemma (the control is a zero vector) can be expressed

$$\mathbf{q}(t) = e^{At}\mathbf{q}_0. \tag{4.18}$$

Both the solution in Equation (4.18) and the system in Equation (4.10) are well-established expressions derived using standard textbooks (Kuo, 1967; Ogata, 2002). To solve the matrix exponential e^{At} in Equation (4.18), we use the eigenvectors matrix decomposition method (Moler and Van Loan, 2003). The method works on the similarity transformation of the form

$$A = VDV^{-1}. \tag{4.19}$$

The power series definition of e^{At} implies then that (Moler and Van Loan, 2003)

$$e^{At} = Ve^{Dt}V^{-1}. \tag{4.20}$$

In this latter expression, let us consider the non-singular matrix V, whose columns are eigenvectors of A. Notation-wise, we can write that

$$V := \begin{bmatrix} v_0 & v_1^0 & v_1^1 & \dots & v_r^0 & v_r^1 \end{bmatrix}. \tag{4.21}$$

and that the diagonal matrix of eigenvalues

$$D = \operatorname{diag}(\lambda_0, \lambda_1^0, \lambda_1^1, \dots, \lambda_r^0, \lambda_r^1), \tag{4.22}$$

where λ_0 is the eigenvalue associated with the first item of A. λ_j^0, λ_j^1 are the two eigenvalues associated with the block A_j. We can then write

$$AV = VD, \tag{4.23}$$

by simply reordering Equation (4.19).

We apply the approach in terms of Equation (4.10) and under the assumptions that we made in the Lemma 4.3.1

$$\dot{\mathbf{q}}(t) = A\mathbf{q}(t). \tag{4.24}$$

The linear combination of the initial guess $\mathbf{q}(t_0)$ and the generic solution can be then expressed

$$F\mathbf{q}(t_0) = \gamma_0 v_0 + \sum_{k=0}^{1}\sum_{j=1}^{r} \gamma_j v_j^k, \tag{4.25a}$$

$$F\mathbf{q}(t) = \gamma_0 e^{\lambda_0 t} v_0 + \sum_{k=0}^{1}\sum_{j=1}^{r} \gamma_j e^{\lambda_j t} v_j^k, \tag{4.25b}$$

where t is a generic time instant and

$$F = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}, \tag{4.26}$$

$F \in \mathbb{R}^m$ is a column vector of ones.

Let us consider the expression in Equation (4.25b). It represents the linear combination of all the coefficients of the state at time t. It can also be expressed in the following form

$$\begin{aligned} F\mathbf{q}(t)/T = \gamma_0 e^{\lambda_0 t} v_0/T + (1/T)\sum_{j=1}^{r} \gamma_j e^{\lambda_j^0 t} v_j^0 + \\ (1/T)\sum_{j=1}^{r} \gamma_j e^{\lambda_j^1 t} v_j^1, \end{aligned} \tag{4.27}$$

where we split the sum and divided each item by the period T.

We prove that the eigenvalues λ and eigenvectors V are such that Equation (4.27) is equivalent to Equation (4.17). To this purpose, we note that matrix A is a block diagonal matrix, and we can express its determinant as the multiplication of the determinants of its blocks

$$\det(A) = \det(0)\det(A_1)\det(A_2)\cdots\det(A_r). \tag{4.28}$$

We now conclude the proof by computing the first determinant and the others separately. By computing the first determinant, we prove that the first terms in Equation (4.17) and Equation (4.27) match. We find the eigenvalue from $\det(0) = 0$, which is $\lambda_0 = 0$. The corresponding eigenvector can be chosen arbitrarily

$$(0 - \lambda_0)v_0 = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}, \tag{4.29}$$

$\forall v_0$, thus we choose

$$v_0 = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{4.30}$$

The sizes of the zero vector and of v_0 in Equations (4.29–4.30) are both $\mathbb{R}^m$.

We find the value γ_0 in Equation (4.27) so that the terms are equal

$$\gamma_0 = \begin{bmatrix} a_0 & 0 & \cdots & 0 \end{bmatrix}, \tag{4.31}$$

where $\gamma_0 \in \mathbb{R}^m$.

Then, we prove the other determinants. In this way, we prove that all the terms in the sum of both Equation (4.17) and Equation (4.27) match. By computing the second determinant, we prove that the first terms in both summaries in Equation (4.17) and Equation (4.27) match. We thus focus on the first block A_1 and find the eigenvalues from

$$\det(A_1 - \lambda I) = 0. \tag{4.32}$$

The polynomial $\lambda^2 + \omega^2$, gives two complex roots–the two eigenvalues

$$\lambda_1^0 = i\omega, \tag{4.33a}$$
$$\lambda_1^1 = -i\omega. \tag{4.33b}$$

The eigenvector associated with the eigenvalue λ_1^0 is

$$v_1^0 = \begin{bmatrix} 0 & -i & 1 & 0 & \cdots & 0 \end{bmatrix}'. \tag{4.34}$$

The eigenvector associated with the eigenvalue λ_1^1 is

$$v_1^1 = \begin{bmatrix} 0 & i & 1 & 0 & \cdots & 0 \end{bmatrix}'. \tag{4.35}$$

Both eigenvectors are equally sized $v_1^0, v_1^1 \in \mathbb{R}^m$.

Again, we find the values γ_1 in Equation (4.27) such that the equivalences

$$\begin{cases} e^{i\omega t}(a_1 - ib_1) & = \gamma_1 e^{i\omega t} v_1^0 \\ e^{-i\omega t}(a_1 + ib_1) & = \gamma_1 e^{i\omega t} v_1^1 \end{cases}, \tag{4.36}$$

hold. They hold for

$$\gamma_1 = \begin{bmatrix} 0 & b_1 & a_1 & 0 & \cdots & 0 \end{bmatrix}. \tag{4.37}$$

The proof for the remaining $r - 1$ blocks is equivalent.

The initial guess $\mathbf{q}_0$ is built such that the sum of the coefficients is the same in both the signals. In the output matrix, the frequency $1/T$ accounts for the period in Equation (4.17), Equation (4.27), and Equation (4.9). At time instant zero, the coefficients b_j are not present and the coefficients a_j are doubled for each $j = 1, 2, \ldots, r$ (thus we multiply by half the corresponding coefficients in $\mathbf{q}_0$). To match the outputs $h(t) = y(t)$, or equivalently

$$F\mathbf{q}(t)/T = C\mathbf{q}(t), \tag{4.38}$$

we have

$$C = (1/T)\begin{bmatrix} 1 & 1 & 0 & \cdots & 1 & 0 \end{bmatrix}. \tag{4.39}$$

We thus conclude that the signal and the output are equal and that the lemma holds.

■

We note for practical reasons that the signal would still be periodic with another linear combination of coefficients. For instance

$$C = d\begin{bmatrix}1 & 1 & 0 & \cdots & 1 & 0\end{bmatrix}, \tag{4.40}$$

equivalent to

$$C = d\begin{bmatrix}1 & 0 & 1 & \cdots & 0 & 1\end{bmatrix}, \tag{4.41}$$

or

$$C = d\begin{bmatrix}1 & \cdots & 1\end{bmatrix}, \tag{4.42}$$

for a given constant value $d \in \mathbb{R}$.

4.3.2 Nominal control of the energy signal

Let us suppose that at time instant t the plan in Definition 2.4.2 reached the ith stage Γ_i and the control contains the configuration of path and computations parameters

$$\begin{aligned} c_i(t) :&= \Big[\overbrace{c_{i,1}(t) \quad \cdots \quad c_{i,\rho}(t)}^{\rho} \; \overbrace{c_{i,\rho+1}(t) \quad \cdots \quad c_{i,\rho+\sigma}(t)}^{\sigma}\Big]' \\ &= \begin{bmatrix} c_i^{\rho}(t) & c_i^{\sigma}(t)\end{bmatrix}', \end{aligned} \tag{4.43}$$

where $c_i(t) \in \mathbb{R}^n$ with $n = \rho + \sigma$ differs from the nominal control $\mathbf{u}(t)$ in Equation (4.10). We include the control in the nominal control exploiting the following observation.

Observation: Relation between the control and energy

We observe that: *(a)* a change in path parameters affects the energy indirectly. It alters the time when the aerial robot reaches the final point $\mathbf{p}_{\Gamma_l}$ and enters the final stage Γ_l, *(b)* a change in computation parameters affects the energy directly. It alters the power as more computations require more power (and vice versa).

The second point in the observation is easily verified. The `powprofiler` profiling tool models the energy consumption of the heterogeneous computing hardware the mobile robot is carrying. A variation in the computations parameters affects the schedule (as the schedule is parametrized by the parameters in Definition 2.1.2), and hence results in more/less power required by the computing hardware.

The first point in the observation can be verified by inspection of the example in Section 2.6. It is clear that if we decrease the parameter $c_{4,1}$ relative to the circle radius, the flying time decreases. This is shown in Figure 5.12 and Figure 5.13. Figure 5.12 illustrates the trajectory of the aerial robot (composed of all the paths $\varphi_1, \varphi_2 \ldots$) flying at the highest configuration of the path parameter $c_{i,1} = \overline{c}_{4,1}$. Figure 5.13 then illustrate the trajectory flying at the lowest configuration $c_{i,1} = \underline{c}_{4,1}$. The flying time differs significantly, along with the quality of the coverage of the polygon (the agricultural field in Figure 1.6). In Figure 5.13, the parameter $c_{4,1}$ that alters the

radius and center of the upper circle (defined originally in Section 2.6) is replanned as, e.g., averse atmospheric conditions do not allow to terminate the original plan in Figure 5.12.

We use the observation later in Section 5.3.2 to check that the time to completely discharge the battery is greater than the flight time and replan the path parameters accordingly. We replan the computation parameters to maximize the instantaneous energy consumption against the maximum battery discharge rate.

The nominal control is

$$\mathbf{u}(t) := \hat{\mathbf{u}}(t) - \hat{\mathbf{u}}(t - \Delta t), \tag{4.44}$$

where $\hat{\mathbf{u}}(t)$ is defined as the energy estimate of a given control sequence at time instant t, $\hat{\mathbf{u}}(t-\Delta t)$ at the previous time instant $t - \Delta t$

$$\hat{\mathbf{u}}(t) := \mathtt{diag}(v_i)c_i(t) + \tau_i, \tag{4.45}$$

where $\mathtt{diag}(v_i)$ is a diagonal matrix with the parameters $v_{i,j} \in v_i$, $\forall j \in [n]_{>0}$.

The input matrix is then

$$B = \begin{bmatrix} 0^{1\times\rho} & \overbrace{1 & \cdots & 1}^{\sigma} \\ 0^{1\times\rho} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0^{1\times\rho} & 0 & \cdots & 0 \end{bmatrix} \Bigg\} 2r+1 \tag{4.46}$$

where $B \in \mathbb{R}^{m\times n}$ contains zeros except the first row where the first ρ columns are still zeros and the remaining σ are ones.

$\hat{\mathbf{u}}(t)$ is a stage-dependent scale transformation with

$$v_i = \begin{bmatrix} \overbrace{v_{i,1} & \cdots & v_{i,\rho}}^{\rho} & \overbrace{v_{i,\rho+1} & \cdots & v_{i,\rho+\sigma}}^{\sigma} \end{bmatrix}' = \begin{bmatrix} v_i^\rho & v_i^\sigma \end{bmatrix}', \tag{4.47a}$$

$$\tau_i = \begin{bmatrix} \tau_{i,1} & \cdots & \tau_{i,\rho} & \tau_{i,\rho+1} & \cdots & \tau_{i,\rho+\sigma} \end{bmatrix}' = \begin{bmatrix} \tau_i^\rho & \tau_i^\sigma \end{bmatrix}', \tag{4.47b}$$

scaling factors. They quantify the contribution to the plan of a given parameter in terms of time for the first ρ parameters, and instantaneous energy consumption for the remaining σ (we use the same notation for the path and computation scaling factors as for the parameters).

The nominal control $\mathbf{u}(t)$ is then the difference of these contributions of two consecutive controls $c_i(t - \Delta t)$, $c_i(t)$ applied to the system. $B\mathbf{u}(t)$ merely includes the difference in the instantaneous energy consumption into the model in Equation (4.10). Matrix B ignores the time contribution of the path parameters in c_i. We use them to verify that the flying time is lower than the battery time in Section 5.3.3.

4.3.3 Control scale transformation

To transform the control $c_i(t)$ at ith stage and time instant t, we use different approaches for the path and computation scaling factors. The scaling factors for the path parameters from

Equation (4.47) are derived empirically. For example, we can obtain the scaling factor $\nu_{4,1}$ relative to the alteration $c_{4,1}$ of the upper circle φ_4 from Section 2.6 by measuring the time needed to compute the path with the lowest configuration $\underline{c}_{4,1}$, $\underline{t}$ in Figure 5.13, and the highest $\overline{t}$ in Figure 5.12.

The variation of the control hence results in an approximate measure of the plan's time variation with factors

$$\nu_{i,j} = \left((\overline{t} - \underline{t}) / (\overline{c}_{i,j} - \underline{c}_{i,j}) \right) / \rho, \tag{4.48a}$$

$$\tau_{i,j} = \left(\underline{c}_{i,j} (\underline{t} - \overline{t}) / (\overline{c}_{i,j} - \underline{c}_{i,j}) + \underline{t} \right) / \rho, \tag{4.48b}$$

$\forall j \in [\rho]^+$. Moreover, let the factors be zero when the parameters $c_i^\rho = \emptyset$. We use the latter to initialize the algorithm in Section 5.3.3.

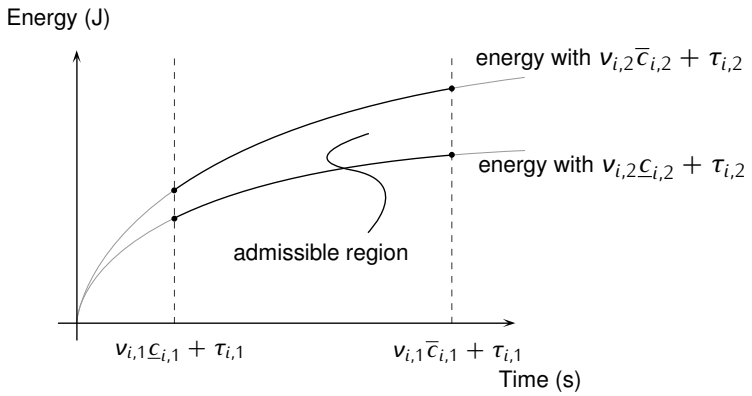


Figure 4.9. The concept of a path and computations parameters scale transformation. Without any battery constraints, the energy-aware coverage planning and scheduling select the highest configuration which respects the control constraint (admissible region) from Equation (2.6).

The scaling factors for the computations parameters from Equation (4.47) are derived using `powprofiler`, the open-source modeling tool from Section 4.1.4. We estimate the energy cost of a given schedule (a given computations configuration) with the function g from Definition 4.1.2. For instance, if the computation is the CNN ROS node, the computation parameter $c_{1,2}$ corresponds to the FPS rate. The tool then measures power according to the detection frequency.

The scaling factors add the computational energy component to the model in Equation (4.10). They are derived similarly to Equation (4.48)

$$\nu_{i,j} = (g(\overline{c}_{i,j}) - g(\underline{c}_{i,j})) / (\overline{c}_{i,j} - \underline{c}_{i,j}), \tag{4.49a}$$

$$\tau_{i,j} = \underline{c}_{i,j} (g(\underline{c}_{i,j}) - g(\overline{c}_{i,j})) / (\overline{c}_{i,j} - \underline{c}_{i,j}) + g(\underline{c}_{i,j}), \tag{4.49b}$$

$\forall j \in [\rho + 1, n]$. We then assume g only returns the power metric (so we do not specify an additional parameter to numerate the metric) and, for ease of notation, assume all the values from

$g(\underline{c}_{i,j})$ to $g(\underline{c}_{i,j})$ are distributed linearly. Moreover, let the factors be zero when the parameters $c_i^{\sigma} = \emptyset$.

The concept of a path and a computation parameter $(c_{i,1}, c_{i,2})$ scale transformation is illustrated in Figure 4.9. The energy domain is bounded by the output of the `powprofiler` tool, while the flight time domain is by the empirical data. The energy-aware coverage planning and scheduling select the highest possible configuration of parameters (control) in the admissible region (under the constraints). Currently, the highest control corresponds to $(\overline{c}_{i,1}, \overline{c}_{i,2})$. We will

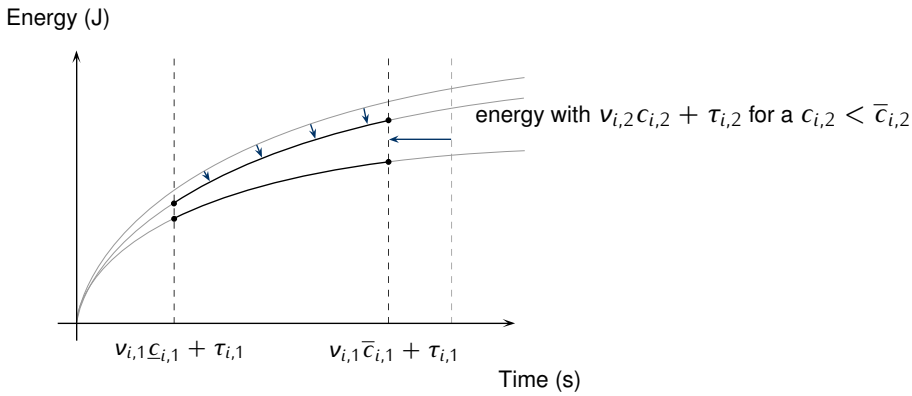


Figure 4.10. Change in the admissible region in Figure 4.9 due to a battery constraint.

see in Section 5.3.1 the optimal control derivation over a time horizon N under given battery constraints. In Figure 4.10 we briefly exemplify this latter case, where due to, e.g., a sudden battery drop, the energy and time domains are shrunk. The highest possible control is thus now different from the above scenario.

4.3.4 Aperiodic energy evolution

4.4 Results

4.5 Summary

Chapter 5

Coverage Planning and Scheduling

> *"[I]t may be possible to trade off reduced resource consumption for a slightly lower but still acceptable level of performance."*
>
> — Lahijanian et al., 2018

In the previous chapters, we introduced progressively the research questions we are interested in addressing. We then provided some preliminaries with basic terminology, formulated the problem formally, detailed the available literature, and derived various energy models. Once we detailed all these basic constructs, we are ready to describe their interaction to solve Problem 2.5.2 and Problem 2.5.1 and thus provide an energy-aware coverage planning and scheduling for autonomous aerial robots.

This chapter describes the main contribution of our work. Here we generate the coverage plan Γ that we defined in Definition 2.4.2 solving Problem 2.5.2, replan Γ energy-wise with the models from Chapter 4 solving Problem 2.5.1 in case of, e.g., sudden battery drops, and guide the aerial robot on Γ. In particular, we first detail how we guide the aerial robot on the plan in Section 5.1, recalling some constructs in Chapter 2. These include path functions, stages, triggering points, and primitive paths. In Section 5.2, we discuss the generation of the coverage plan with a union of path functions and triggering points in Sections 2.2–2.3 (it is on this coverage that we are interested in guiding the aerial robot). In Section 5.3, we then discuss how to replan the coverage energy-wise. Although we already described most of the concepts in preliminaries in Chapter A and literature in Chapter 3, we still need some additional notions. To guide the aerial robot, we use the theory of vector fields that point to the path functions. To generate the coverage path, a class of methods under the name of cellular decomposition, generating a coverage motion that respects the nonholonomic and other constraints of a fixed-wing aerial robot (such as the Opterra craft in Figure 1.1 that we have discussed extensively in this work), including requirements on the turning radius. To replan the coverage path, we use an optimal control approach termed model predictive control (MPC) along with the periodic model in Chapter 4 (which we proved formally

and motivated empirically in Section 4.3.1). We describe all these concepts and contextualize them in the solution to the problems in this chapter.

This chapter connects to the remainder of this work as follows. Here we provide the solution to the problems in Chapter 2. To this end, we use the available literature on planning in Chapter 3 and the energy models in Chapter 4. We provided the motivation and discussed why it is important to solve these problems in Chapter 1. We use the model in the cost and constraints of an optimal control problem (OCP) using the preliminaries in Chapter A. Although we provide an algorithm for energy-aware coverage planning and scheduling for autonomous aerial robots, some research questions remain open. We discuss these questions in Chapter 7.

5.1 Guidance on the coverage

In this section, we describe how we guide the aerial robot in space $\mathcal{Q}_v \subseteq \mathbb{R}^2$ for an inertial navigation frame $\mathcal{O}_W$ (we will see what we mean by v in Section 5.2.1). For this purpose, we briefly recall some concepts we introduced in Chapter 3. We describe the path the aerial robot flies in Section 2.2 with a mathematical function $\varphi_i : \mathbb{R}^2 \times \mathbb{R}^\rho \to \mathbb{R}$ that maps a point in 2D space and the path parameters to a given value on the z-axis in Definition 2.2.1. We saw two examples of such functions. In the first example, we proposed a line at an altitude $h \in \mathbb{R}$ in Figure 2.1. The value on the z-axis given a point $\mathbf{p}(t)$ at the time t is then the length, let's call it d, of a vertical segment parallel to the z-axis that goes from the plane $\varphi(x, y) = h$ and intersects the plane in Equation (2.2). In the second example, we proposed a circle (at the same altitude h) in Figure 2.2. The value on the z-axis is the length d_2 of a similar segment, going from the plane $\varphi(x, y) = h$ to the intersection of the paraboloid in Equation (2.3). We further recall from Chapter 3 that we store path functions in stages in Definition 2.3.1. The set of stages form the plan Γ in Definition 2.4.2. The aerial robot flies the ith a stage Γ_i traveling the ith path function φ_i up until it encounters the triggering point $\mathbf{p}_{\Gamma_i}$ in Definition 2.3.2; at the occurrence, the action depends on how we defined Γ. We can define Γ with all the stages explicitly so that the aerial robot switches to Γ_{i+1} up to reaching the final point $\mathbf{p}_{\Gamma_l}$ in Γ_l, the final stage. Alternatively, we can define Γ with n stages and a shift $\mathbf{d}$. When the aerial robot reaches the knth triggering point $\mathbf{p}_{\Gamma_{kn}}$ for some $k \in \mathbb{Z}_{>0}$, it advances the n stages of $\mathbf{d}$. It iterates the process up to reaching the final point $\mathbf{p}_{\Gamma_l}$.

In the remainder of this section, we detail how we guide the aerial robot on a path function φ_i from the stage Γ_i, $\forall i \in [l]_{>0}$ (or $\forall i \in [n]_{>0}$ with a consequent shift of $\mathbf{d}$ when we reach $\mathbf{p}_{\Gamma_{kn}}$ for a k) up to reaching $\mathbf{p}_{\Gamma_l}$. The path function can be, e.g., the circle and line in Figures 2.1–2.2. By guidance, we mean where to fly next with the aerial robot starting from an initial point in space $\mathbf{p}(t_0)$ at the first time instant t_0 up to the final triggering point $\mathbf{p}_{\Gamma_l}$ at $t_l > t_0$.

5.1.1 Vector fields for guidance

Let us briefly discuss the intuition behind vector fields for guidance with the concept of potential functions. These are differentiable real-valued functions $\varphi : \mathbb{R}^d \to \mathbb{R}$, of which the value we

can consider as energy, and hence their gradient as force (H. M. Choset et al., 2005). There are several pseudonyms for potential functions for different fields: e.g., the electrostatic potential for electrostatics, velocity potential for hydrodynamics, and temperature for flowing heat (Needham, 1998). We use the concept for exemplification; we don't deal with aerial robots' dynamics directly in our model and see gradients as velocity and not force vectors, being $\mathbf{p}$ the position. We note that the gradient of the potential function points where it maximally (locally) increases (H. M. Choset et al., 2005). We define the gradient of φ

$$\nabla\varphi_i(\mathbf{p}(t), c_i^\rho) := \begin{bmatrix} \partial\varphi_i(\mathbf{p}(t), c_i^\rho)/\partial\mathbf{p}_1(t) \\ \partial\varphi_i(\mathbf{p}(t), c_i^\rho)/\partial\mathbf{p}_2(t) \\ \vdots \\ \partial\varphi_i(\mathbf{p}(t), c_i^\rho)/\partial\mathbf{p}_d(t) \end{bmatrix}, \tag{5.1}$$

where $\partial\varphi/\partial\mathbf{p}_k$ for $k \in [d]_{>0}$ indicates the differential and $\mathbf{p}_1, \mathbf{p}_2, \dots$ are

$$\mathbf{p}(t) = \begin{bmatrix} \mathbf{p}_1(t) & \mathbf{p}_2(t) & \cdots & \mathbf{p}_d(t) \end{bmatrix}', \tag{5.2}$$

simply the components of the vector $\mathbf{p}$ (i.e., when we are dealing with 2D space, d is two, and the components are x and y).

We can then use the gradient in Equation (5.1) to define a vector field–a function that assigns a vector at each $\mathbf{p}$ in $\mathcal{Q}^v$ (LaValle, 2006), which will then point in the direction of the gradient

$$\Phi(t, \varphi_i, c_i^\rho) := \bigcup_{\mathbf{p}(t)\in\mathcal{Q}^v} \nabla\varphi_i(\mathbf{p}(t), c_i^\rho). \tag{5.3}$$

We note some analogies in the physical theory of potential functions with our path functions; indeed vector fields are a well-known concept in physics, with applications such as electrostatic, gravitational, and magnetic fields (Feynman et al., 2015). Imagine that the aerial robot is a positively charged particle in an electrostatics analysis, attracted by a negatively charged goal. Via the gradient, we can then direct the particle (the aerial robot) to the goal (where the function maximally locally decreases) (H. M. Choset et al., 2005). In the setting of Equation (2.2) and Equation (2.3), i.e.,

$$2y - x = h, \quad (x-3)^2 + (y-3)^2 - 2 = h, \tag{5.4}$$

their gradient then points away from the base of the plane in Equation (2.2) and the center of the circle in Equation (2.3) in Figure 5.1. If the goal is not to fly over the circle in Figure 2.2, but to its center (x_c, y_c) in Equation (2.3), the vector field Φ in Equation (5.3) direct us in the opposite direction; we can then use $-\nabla\varphi_i$ to direct the robot to the goal.

Vector fields are common in the motion planning literature (H. M. Choset et al., 2005; LaValle, 2006) and in studies (Garcia De Marina et al., 2017; Gonçalves et al., 2010; Kapitanyuk et al., 2017; Lindemann and LaValle, 2005; Panagou, 2014; Zhou and Schwager, 2014) for navigation and guidance of different mobile robots. A well-known intuitive method based on vector fields brought from optimization is the gradient descent algorithm (Bryson and Ho, 1975; H. M. Choset

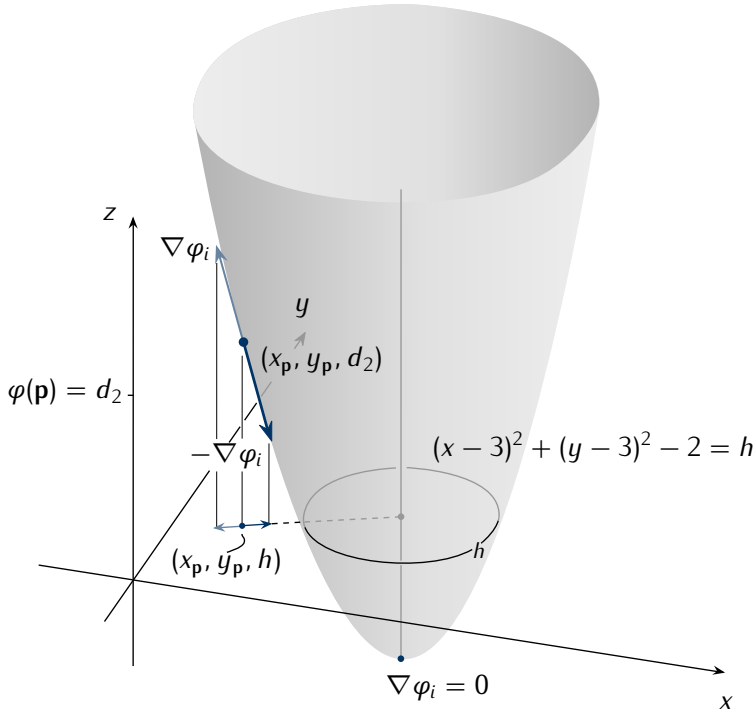


Figure 5.1. The direction of the gradient $\nabla\varphi_i$ in the point $(x_\mathbf{p}, y_\mathbf{p})$ at an altitude h w.r.t. $\mathcal{O}_W$ for the circle path function in Equation (2.3) and Figure 2.2. The gradient directs where the function locally maximally increases; the opposite thus to the center of the circle in Equation (5.4) (or the base of the paraboloid φ_i).

Input : t_0 initial time step
c_i^ρ value of the path parameters
j current stage
Output : $\mathbf{p}(\mathcal{K})$ trajectory

1 foreach $i \in \mathcal{K} := \{t_0, t_0 + h, t_0 + 2h, \dots\}$ do
2 if $\left\|\nabla\varphi_j(\mathbf{p}(t), c_i^\rho)\right\| \leq \varepsilon$ then
3 return $\mathbf{p}(\mathcal{K})$
4 $\mathbf{p}(i+h) \leftarrow \mathbf{p}(i) + \theta(i)\Delta\varphi_j(\mathbf{p}(t), c_i^\rho)$

Algorithm 5.1. Gradient descent

et al., 2005), where an intuitive choice of the search direction is the negative gradient $\Delta\varphi_i := -\nabla\varphi_i$ (Boyd et al., 2004). We detail the gradient descent algorithm in Algorithm 5.1, where we iterate at discrete time steps, meaning that at instant n, $\mathcal{K}$ contains $t_0, t_0 + h, \dots, t_0 + nh$. We discuss further the time step h ($h$is not to be confused with the altitude when used in the path functions) later in this chapter in Section 5.3.3. In the algorithm, $\mathbf{p}(t_0)$ is given (e.g., from sensors data), $\theta(i)$ is a scalar step size at time instant i (H. M. Choset et al., 2005)–there can be indeed

different step sizes at different instants–and $\varepsilon \in \mathbb{R}_{>0}$ is chosen based on the task requirements (it is unrealistic to assume we will reach $\nabla\varphi_i = 0$).

We discuss in the next section how to fix Φ so that it directs us to follow Equation (5.4) rather than, e.g., to $\nabla\varphi_i = 0$.

5.1.2 Derivation of a path following vector field

Algorithm 5.1 that we illustrate in Figure 5.2 directs to the center of the circle when we have a circle as a path function in Equation (5.4). However, we want to track (or follow) these functions.

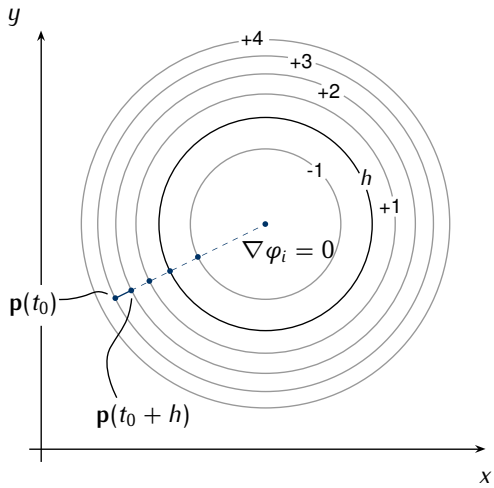


Figure 5.2. The gradient descent algorithm after the first two steps, with the following step being dashed and directing the aerial robot to the center of the circle where the gradient is zero $\nabla\varphi_i = 0$. The paraboloid φ_i representing the path function circle at altitude h is illustrated in the contour plot.

Concretely, in the path sub-plan in Equations (2.12–2.13) in Section 2.6.1, we want to follow the path function φ_{i+1} by flying over rather than heading the center when we follow a circle path function. In the example, we started following the circle described by φ_{i+1} after reaching $\mathbf{p}_{\Gamma_i}$ while tracking φ_i (for all the previous and following path functions). To this end, we use a vector field-based approach proposed in the literature specifically for aerial robots (Garcia De Marina et al., 2017), which points to the contours of the functions in Figures 2.1–2.2 and thus to Equation (5.4).

The expression for the search direction $\Delta\varphi_i$ in Algorithm 5.1 using the vector field in (Garcia De Marina et al., 2017) becomes

$$\Delta_d\varphi_i(\mathbf{p}(t), c_i^\rho) := E_i\nabla\varphi_i(\mathbf{p}(t), c_i^\rho) - k_e\varphi_i(\mathbf{p}(t), c_i^\rho)\nabla\varphi_i(\mathbf{p}(t), c_i^\rho), \tag{5.5}$$

where $E_i \nabla \varphi_i(\mathbf{p}(t), c_i^\rho)$ is a vector pointing perpendicularly to the gradient $\nabla \varphi_i(\mathbf{p}(t), c_i^\rho)$, E_i is the ith stage direction

$$E_i = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \tag{5.6}$$

with E_i being the counterclockwise, $-E_i$ the clockwise direction. The contribution of the component $-k_e \varphi_i(\mathbf{p}(t), c_i^\rho) \nabla \varphi_i(\mathbf{p}(t), c_i^\rho)$ in Equation (5.5) points in the direction of the path function. It depends on the coefficient $k_e \in \mathbb{R}_{>0}$–indicating the speed of convergence (Garcia De Marina et al., 2017)–and on the value of φ_i at the current point (of course also on the path parameters and the value of the gradient). We illustrate $\Delta_d \varphi_i$ in Figure 5.3. When we take a point within

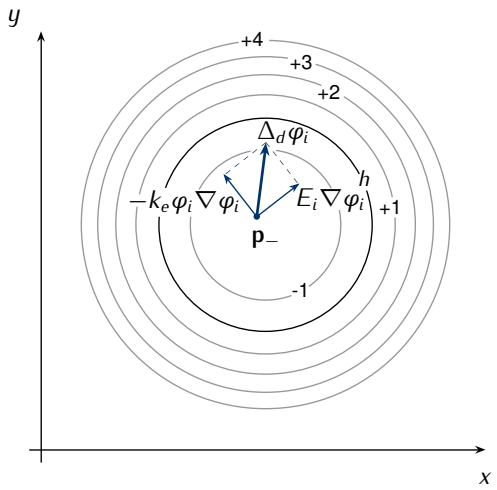


Figure 5.3. The direction of the vector field for guidance in (Garcia De Marina et al., 2017) with a point $\mathbf{p}_-$ inside the circle: Δ_d points to the path function opposite to the center of the circle.

the circle, let's call it $\mathbf{p}_-$, the value of φ_i is negative; $-k_e \varphi_i(\mathbf{p}_-, c_i^\rho) \nabla \varphi_i(\mathbf{p}_-, c_i^\rho)$ points outwards of the circle center, in the direction of the path function. $E_i \nabla \varphi_i(\mathbf{p}_-, c_i^\rho)$ points perpendicularly to the gradient $\nabla \varphi_i$ and to the path function itself. The resulting vector $\Delta_d \varphi_i(\mathbf{p}_-, c_i^\rho)$ then points to the direction of the path function.

Now we take a point $\mathbf{p}_+$ out of the circle, and not exactly over the path function. The value of φ_i is positive; $-k_e \varphi_i(\mathbf{p}_+, c_i^\rho) \nabla \varphi_i(\mathbf{p}_+, c_i^\rho)$ points inwards of the circle center and the resulting vector $\Delta_d \varphi_i(\mathbf{p}_-, c_i^\rho)$ in the direction of the path function. If we finally take a point $\mathbf{p}_0$ exactly over the path function, φ_i is zero; $-k_e \varphi_i(\mathbf{p}_0, c_i^\rho) \nabla \varphi_i(\mathbf{p}_0, c_i^\rho)$ is thus also zero, and $\Delta_d \varphi_i(\mathbf{p}_0, c_i^\rho)$ points perpendicularly to the path functions. We illustrate these two cases in Figure 5.4. Let us thus define the vector fields

$$\Phi_d(t, \varphi_i, c_i^\rho) := \bigcup_{\mathbf{p}(t) \in \mathcal{Q}^v} \Delta_d \varphi_i \mathbf{p}(t), c_i^\rho. \tag{5.7}$$

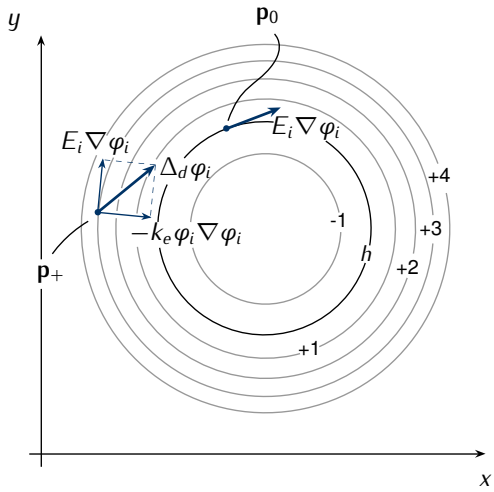


Figure 5.4. The direction of the vector field with $\mathbf{p}_+$ outside the circle: Δ_d points to the path function in the direction of the center of the circle. With $\mathbf{p}_0$, Δ_d points perpendicularly to the path function.

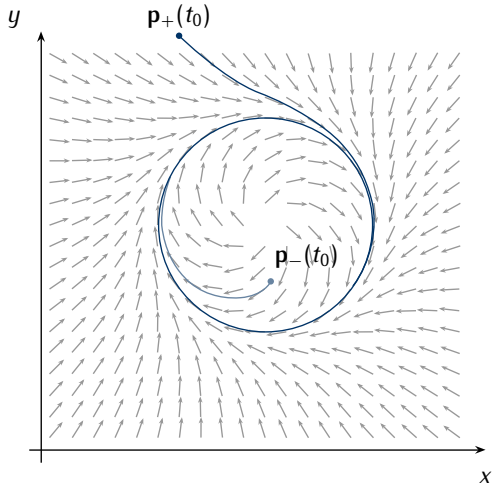


Figure 5.5. Path-following vector field Φ_d of a circle path function with the aerial robot starting inside $\mathbf{p}_-(t_0)$ and outside $\mathbf{p}_+(t_0)$ the circle.

This vector fields points to the path function such as the line and the circle in Equation (5.4) for every point in space $\mathcal{Q}^v$ as we illustrate in Figure 5.5 for the circle $(x-3)^2 + (y-3)^2 - 2 = h$.

Finally, let's reconsider Algorithm 5.1 with the vector field Φ_d and guide the aerial robot in space to track a specific path function φ_i in Algorithm 5.2. The algorithm iterates at discrete time steps on Line 1 as Algorithm 5.1. It stops tracking the jth path function at the occurrence of the triggering point $\mathbf{p}_{\Gamma_j}$ on Line 2 using the Euclidean distance with a small enough $\varepsilon \in \mathbb{R}_{>0}$ rather

Input : t_0 initial time step
c_i^ρ value of the path parameters
j current stage
Output : $\mathbf{p}(\mathcal{K})$ trajectory

1 foreach $i \in \mathcal{K} := \{t_0, t_0 + h, t_0 + 2h, \dots\}$ do
2 if $\left\|\mathbf{p}(i) - \mathbf{p}_{\Gamma_j}\right\| \leq \varepsilon$ then
3 return $\mathbf{p}(\mathcal{K})$
4 $\mathbf{p}(i + h) \leftarrow \mathbf{p}(i) + \theta(i)\Delta_d\varphi_j(\mathbf{p}(i), c_j^\rho)$

Algorithm 5.2. Path function tracking

than evaluating if the function reached a local minimum with $\nabla\varphi_j = 0$ in Algorithm 5.1. The algorithm then computes a simplified version of the next position on Line 4 using the current position $\mathbf{p}(i)$ and the gradient $\Delta_d\varphi_i$ without considering vehicles' velocity $v(i)$ and other parameters (such as external interferences, e.g., wind speed and direction). For simplicity, we can use the same value for the time step h that we used for the step size θ for all the time instants in $\mathcal{K}$.

5.1.3 Derivation of the guidance action

5.2 Coverage Path Planning

In this section, we solve Problem 2.5.2. Let us recall briefly our objective of providing a set of paths (a tour) in the plan from Definition 2.4.2 to cover each point in a given space. We summarize such space with a set of vertices $v := \{v_1, v_2, \dots\}$ that form a polygon. The robot is free to move within the polygon except for some obstacles described by other sets of vertices, one per each obstacle $o_1 := \{o_{1,1}, o_{1,2}, \dots\}, o_2 := \{o_{2,1}, o_{2,2}, \dots\}, \dots$. There are several different approaches in the literature to solve this problem. We have detailed the approaches in the literature in Sections 3.3.1–3.3.2 for mobile robots and in Sections 3.4.1–3.4.2 for aerial robots specifically. In summary, the sub-class of motion planning that finds the coverage tour of a given space is called coverage path planning (CPP) (H. Choset and Pignon, 1998). The algorithms for the coverage tour are NP-hard (E. M. Arkin, S. P. Fekete, et al., 2000) and use either implicitly or explicitly the cellular decomposition that divides the robot's free space into sub-regions that can be easily covered (H. Choset, 2001; Galceran and Carreras, 2013).

There are numerous methodologies for cellular decomposition itself. Some decompose the polygon into equally sized sub-regions that form a grid and then visits only the sub-regions where the robot is free to move (Galceran and Carreras, 2013). This methodology is termed grid decomposition in Figure 5.6. Another way is to sweep the polygon and divide it into sub-regions when the sweep line encounters a change in connectivity. We implement this latter class in Section 5.2.1. Once the algorithm divides the free space into sub-region, it builds an adjacency graph. The vertices contain the sub-regions and edges connect adjacent sub-regions (H. M. Choset et al., 2005). A covering order between the sub-region to derive the sequence of the coverage is then an exhaustive walkthrough of the adjacency graph with, e.g., depth-first search algorithm (H. M.

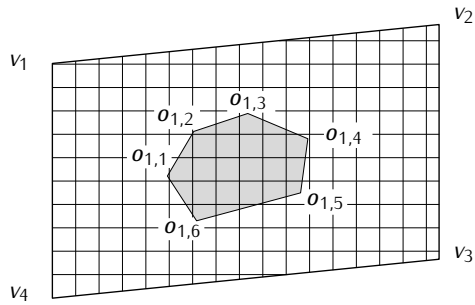


Figure 5.6. A polygonal space where we want to find a coverage tour and visit all the points delimited by $v := \{v_1, \ldots, v_4\}$ except for the obstacle $o_1 := \{o_{1,1}, \ldots, o_{1,6}\}$. A way to cover the space is the grid decomposition that divides the polygon into equally sized cells and visits each cell except the obstacle.

Choset et al., 2005). Once divided the space and the order of the coverage, we need to cover each sub-region. We recall that a method is the boustrophedon motion that we discussed in Chapter 3. However, a generic nonholonomic mobile robot, such as the Opterra fixed-wing craft in the precision agriculture scenario in Section 1.3, has limited maneuverability (Mannadiar and Rekleitis, 2010; Xu et al., 2011, 2014). For a generic aerial robot, it is preferred to have a large turning radius (X. Wang et al., 2017); to this end, we propose a Zamboni-like motion. We introduced both the Zamboni- and boustrophedon-like motions in Section 2.6.1, whereas we implement them later in this chapter in Section 5.2.2.

5.2.1 Cellular decomposition of the space

In detail, a cellular decomposition decomposes the coverage space into non-overlapping subregions called cells. Let us define the robot's free space or the coverage space as $\mathcal{Q}^v$ for an inertial navigation frame $\mathcal{O}_W$. Physically, the free space is where the robot is free to move without intersecting an obstacle (H. M. Choset et al., 2005). Let $\mathcal{Q}^{o_i} \subset \mathbb{R}^2$ be the space delimited by the obstacle o_i. $\mathcal{Q}^v \subseteq \mathbb{R}^2$ contains all the points delimited by the vertices of the polygon v except for i obstacles delimited by the vertices of i polygons o_i. The entire space in the polygon v, including all the obstacles o_i, is then $\mathcal{Q} := (\bigcup_{i \in |o|} \mathcal{Q}^{o_i}) \cup \mathcal{Q}^v$. In Figure 5.7, the polygon is delimited by $v := \{v_1, \ldots, v_4\}$ and forms $\mathcal{Q}^v$, whereas the obstacle by $o_1 := \{o_{1,1}, \ldots, o_{1,6}\}$ and forms $\mathcal{Q}^{o_1}$. The union of these two is then $\mathcal{Q}$.

An important approach in the polygonal environment is the boustrophedon decomposition (H. Choset, 2000). For non-polygonal environments where v and o_i are, e.g., elliptical functions, a significant result is the decomposition in terms of critical points of Morse functions (H. Choset, E. Acar, et al., 2000). Both the boustrophedon decomposition and decomposition in terms of critical points of Morse functions sweep $\mathcal{Q}$ with a line and decompose $\mathcal{Q}^v$ adding a cell in case of a change in connectivity or when they encounter a critical point (H. Choset, 2000, 2001; H. M. Choset et al., 2005). In Figure 5.8, the change in connectivity happens when the sweeping line encounters the obstacle o_1. This approach optimizes the neighboring cells that

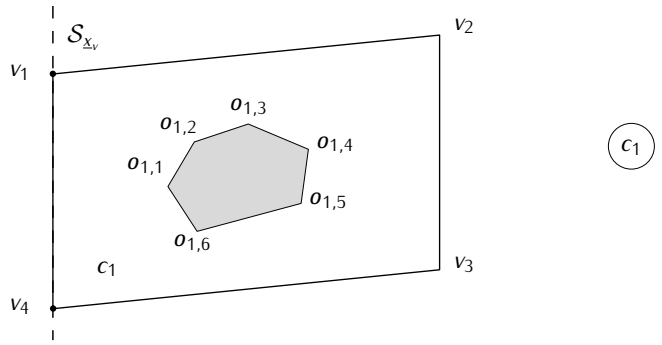


Figure 5.7. The boustrophedon decomposition for coverage path planning sweeps the space and adds cells in case the sweeping line encounters a change in connectivity. Figure shows an initial step with c_1 the first cell formed.

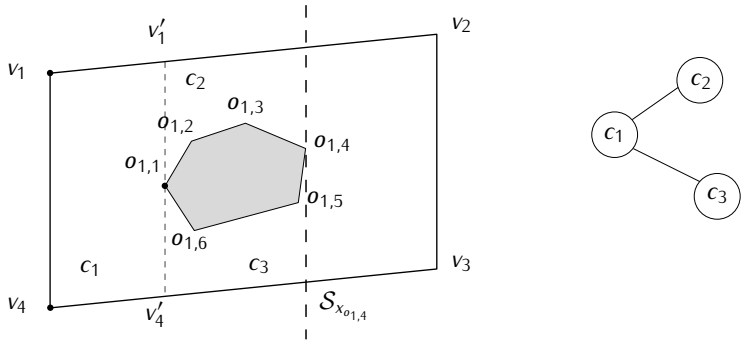


Figure 5.8. An intermediate step of the boustrophedon decomposition, with c_2, c_3 formed at the first encounter of the obstacle o_1. The black points indicate the critical points or changes in connectivity.

can be thus aggregated as opposed to, e.g., trapezoidal decomposition (Galceran and Carreras, 2013) in Figure 5.9, which splits the space into cells when it encounters a vertex (LaValle, 2006). For the decomposition in terms of critical points of Morse functions, the intuition of using critical points (H. Choset, E. Acar, et al., 2000) comes from some early studies on roadmaps (J. Canny, 1988a,b; J. F. Canny and M. C. Lin, 1993). Notably, these studies show that topology (i.e., connectivity) changes only at critical points of a sweeping function restricted to the boundaries of obstacles. We briefly summarize some findings (H. Choset, E. Acar, et al., 2000) for this latter method before discussing the coverage motion for the cells.

Let us define $\mathcal{S}_\lambda$ as the vertical sweeping function that sweeps $\mathcal{Q}$. A change in the value of λ moves the function in $\mathcal{Q}$. Let further $\overline{x}_v, \underline{x}_v$ be the highest and lowest coordinate x of all the vertices in v, i.e., $\lambda \in [\underline{x}_v, \overline{x}_v]$. If we refer to the sweeping function with at a specific point in space as a slice, we can express the entire space as the union of all the slices, i.e., $\mathcal{Q} = \cup_\lambda \mathcal{S}_\lambda$. Let us further define the slice contained in the free space $\mathcal{S}_\lambda^v := \mathcal{S}_\lambda \cap \mathcal{Q}^v$. At this point, a change in connectivity of $\mathcal{S}^v$ means that the original cell has to be closed and two more opened,

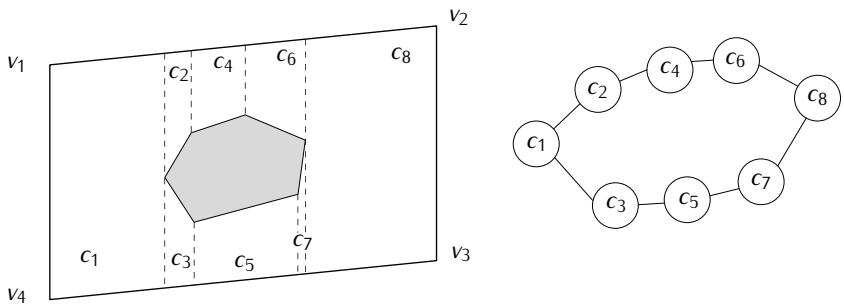


Figure 5.9. In the trapezoidal decomposition a lot of small cells are created (i.e., the cells c_2, c_3, c_7) that can be otherwise merged resulting in disconnected coverage. Boustrophedon decomposition solves the problem by splitting/merging the cells at critical points rather than at vertices. The resulting tour has eight cells as opposed to four cells with boustrophedon decomposition in Figure 5.10.

or that two cells are closed and one is opened respectively when the connectivity increases or decreases (H. Choset, E. Acar, et al., 2000). In Figure 5.8, $\mathcal{S}_\lambda$ sweeps the space from $\lambda = \underline{x}_v$ in

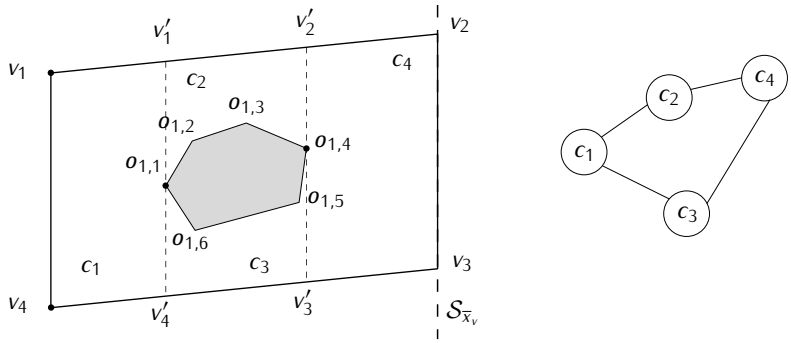


Figure 5.10. The final step of the boustrophedon decomposition, where the sweeping line $\mathcal{S}_\lambda$ encounters the final point of its domain $\overline{x}_v$. The decomposition results in four cells. To determine the order of the coverage tour, the methodology is to visit the adjacency graph.

Figure 5.7 up to $\lambda = x_{o_{1,1}}$. At this latter lambda, $\mathcal{S}_{x_{o_{1,1}}}$ encounters a change in connectivity ($\mathcal{S}^v_{x_{o_{1,1}}}$ forms two disconnected slices). The decomposition methodology builds two new cells c_2, c_3, and adds these cells to the adjacency graph. $\mathcal{S}_\lambda$ encounters another change in connectivity at $\lambda = x_{o_{1,4}}$ ($\mathcal{S}^v_{x_{o_{1,4}}}$ is again one connected slice). This latter is different from the previous: the cells are merged with forming a new cell c_4. The coverage tour is then a visit through the adjacency graph, resulting in the coverage order c_1, c_2, c_4, and finally c_3.

In summary, the methodology iterates through the environment with $\mathcal{S}^v_\lambda$ in Figure 5.7. When the connectivity of $\mathcal{S}^v_\lambda$ increases in Figure 5.8, it closes a cell and opens two new cells–the literature (H. Choset, E. Acar, et al., 2000; H. M. Choset et al., 2005) refers to these cells as ceiling and floor cells (for c_2 and c_3 in Figure 5.10 respectively). When the connectivity decreases back in Figure 5.10, the two opened cells are closed, and a new one is opened. The overall complexity is

$O(n \log n)$ with $n := |v| + \sum_{i=1}^{|o|} |o_i|$ the total number of vertices (H. Choset, E. Acar, et al., 2000). Indeed, for polygonal environments, it is enough to verify the change in connectivity by iterating the vertices and visit the constructed adjacency graph to find the coverage order.

5.2.2 Coverage motion generation

Once we delimited the coverage and obstacles spaces into appropriate cells, we need to define the tour that travels through all the points in the cells–the coverage motion. A classical approach for the coverage motion in the literature is to travel back and forth. We saw in Chapter 3 and discussed briefly at the beginning of Section 5.2 that this is termed the boustrophedon motion. We propose a slight variation of this motion for our problem, which we introduced with the intuitive path in Section 2.6.1 in Figure 2.6. The variation called the boustrophedon-like motion optimizes the turns of the aerial robot flying. The past literature analyzes broadly turns optimizations in the coverage for both mobile (Huang, 2001) and aerial robots (Artemenko et al., 2016; Y. Li et al., 2011). The problem is that mobile robots are often subject to various constraints, including the turning radius. The original boustrophedon motion has edges parallel to the polygon where a mobile robot might have to slow to perform the turn. For instance, a lawnmower mobile robot would have to drive outside the path to turn efficiently (Huang, 2001). An aerial robot traveling the boustrophedon motion might have to follow a greedy path planning algorithm instead or travel an additional turning maneuver such as a curlicue orbit (Xu et al., 2011, 2014). To this end, the boustrophedon-like motion in Section 2.6.1 considerably smooths the turns. Given two following back and forth parallel lines in the original version, ours connects these lines using a semi-circle of a given radius rather than connecting them with additional lines. We illustrate how we cover the cell c_1 in Figure 5.10 using the boustrophedon-like motion in Figure 5.11. The

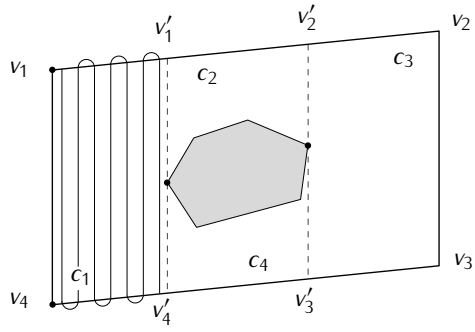


Figure 5.11. The boustrophedon-like motion covering the cell c_1, composed of back and forth parallel lines and circles connecting them of radius half the ideal coverage distance.

remaining cells are covered in the same manner. The overall coverage is achieved by visiting the cells in the appropriate order derived in the previous section (c_1, c_2, c_4, and c_3).

Let us assume for an instant that the turning radius in Problem 2.5.2 is not given. Let us further assume that the aerial robot can overfly the boundaries of the polygons v and o_i for the turns. The methodology that outputs the plan Γ that covers $\mathcal{Q}^v$ with boustrophedon-like motion is

to build four paths: *(a)* the line φ_1 from Figure 2.6 parallel to the edge formed by vertices v_1 and v_4, *(b)* the circle φ_2 with the center laying on the edge formed by vertices v_4 and v_3, *(c)* the line φ_3 parallel to φ_1 that connects the right side of φ_2 and extends up to the left side of φ_4, and *(d)* the circle φ_4 whose center is on the edge formed by vertices v_1 and v_2. The remaining paths $\varphi_5, \varphi_6, \ldots$ are formed similarly. To evaluate the radius of the circles, let us assume the ideal distance between the vertical lines in the motion (the lines $\varphi_1, \varphi_3, \varphi_5, \ldots$) is a given constant. Then the radius in the plan (the circles $\varphi_2, \varphi_4, \varphi_6, \ldots$) is half the ideal distance. We can change the radius of the circles and thus alter the quality of the coverage accordingly. Indeed our planning approach consists of generating an initial plan that can be changed in a replanning phase using an optimal control technique in Section 5.3.1 with the aerial robot being subject to uncertainty and external interferences in flight.

Although the turns are considerably smoothed with the plan containing the boustrophedon-like motion, they are still impractical for fixed-wing aerial robots with the turning radius exceeding the radius of these turns (Dille and Singh, 2013; Xu et al., 2011, 2014). For this latter class of robots, we utilize the Zamboni-like motion. The Zamboni motion is often in the literature for fixed-wing aerial robots (Ablavsky and Snorrason, 2000; Araújo et al., 2013; Majeed and S. Lee, 2019), and its name comes from the hockey arenas' ice maintenance machines (Ablavsky and Snorrason, 2000; Araújo et al., 2013; Dille and Singh, 2013). They have a large turning radius like the aerial robots we study; hence they resurface the ice by sweeping distant lines first instead of adjacent lines in a back and forth motion (the boustrophedon or boustrophedon-like motions). The Zamboni-like motion is similar to the original Zamboni motion in the literature but ap-

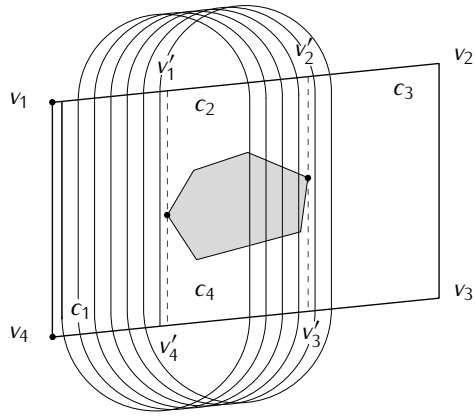


Figure 5.12. The Zamboni-like motion to cover the cell c_1. It is similar to the boustrophedon-like motion in Figure 5.11 but travels distant lines first, respecting the large turning radius constraint of the aerial robots we analyze in this work.

plied to our scenario. We illustrate the Zamboni-like motion for c_1 in Figure 5.12 (whereas the boustrophedon-like motion in Figure 5.11).

Let us assume that the aerial robot can completely overfly the obstacles; we discuss this assumption in Section 5.3.1. The intuition is that although the robot moves over the obstacle, it has a

further computations constraint specifying that it cannot perform any computation (similarly to the turns which we constrained in Section 2.6.2). Let us further adopt the notation ${}^{v_1}|_{v_{|v|}}$ for an edge connecting vertices v_1 and $v_{|v|}$ (in this latter case, the first and last vertex). To generate the plan Γ that covers $\mathcal{Q}^v$ with the Zamboni-like motion we build four paths: *(a)* the line φ_1 parallel to ${}^{v_1}|_{v_{|v|}}$ that extends from ${}^{v_{|v|}}|_{v_{|v-1|}}$ to ${}^{v_1}|_{v_2}$ (similarly to the boustrophedon-like motion), *(b)* the circle φ_2 of which the left side intersects the line that we just created and the right ${}^{v_x}|_{v_y}$ with $v_x, v_y \in v$ being two vertices of the polygon v at a point x_{Γ_2} (the name of the point is relative to the nomenclature in Figure 2.7). This latter point depends on the radius of the circle r_1. For the following path, let us call ${}^{v_x}|_{v_y}$ the floor edge and ${}^{v_w}|_{v_z}$ the corresponding ceiling edge constructed with the sweeping function $\mathcal{S}_\lambda$ intersecting ${}^{v_w}|_{v_z}$ at $\lambda = x_{\Gamma_2}$. *(c)* the line φ_3 parallel to φ_1 that intersects the right side of φ_2 and extends from the ceiling to the floor edge at x_{Γ_2}, and *(d)* the circle φ_4 of a given radius and a parameter introduced in Section 2.6.1. The left edge of the circle lays on φ_3, whereas the right intersects another edge of the polygon. In Figure 5.12, the circle intersects ${}^{v_1}|_{v_2}$. Once we built these four paths, let us assume the polygon is regular and composed of four edges. It is then enough to generate the coverage tour from the primitive paths $\varphi_1, \ldots, \varphi_4$ with a shift $\mathbf{d}$ in the same way as we did in Section 2.6.1. The corresponding plan Γ contains the stages and some additional obstacles dependent constraints: to perform the computations only in $\mathcal{Q}^v$, or analogously, cells $c_1, c_2, \ldots$ coming from the cellular decomposition in Section 5.2.1. We discuss the actual implementation of the aerial robot flying the plan Γ in the next section, where we execute the plan according to the constraints (of the plan and the cellular decomposition) and replan the original plan in case of unexpected and uncertainty-driven events. For the plan Γ, we thus use the Equations (2.12–2.13) for the paths, and Equation (2.14) for the triggering points (i.e., the points in Definition 2.3.2 where happens the change of the path).

If the polygon is not regular and composed of four edges, we still build the remaining paths in the plan Γ $\varphi_5, \varphi_6, \ldots$ starting from the primitive paths with slight variations. If, for example, the ceiling edge points higher than the floor edge, the line segments $\varphi_5, \varphi_9, \varphi_{13}, \ldots$ and $\varphi_7, \varphi_{11}, \varphi_{15}, \ldots$ are longer than φ_1 and φ_3 of a given rate. If, on the contrary, the ceiling edge points lower than the floor edge, the line segments are shorter. Complex shapes are equally possible by, e.g., generating the plan online at each period (i.e., the time needed to fly primitive paths in Definition 2.4.3).

The complexity of the coverage motion generation algorithm for a cell that returns Γ with the primitive paths is simply the complexity of building then the four primitive paths $\varphi_1, \ldots, \varphi_4$. We saw that this is enough to cover both a complex polygon or a regular one with four edges in Figure 5.12, and thus the running time is constant $O(1)$. The overall complexity of cellular decomposition of a polygon and the plan generation is thus $O(n \log n)$, where n is the number of vertices v and the vertices of i obstacles o_i (H. Choset, E. Acar, et al., 2000).

5.3 Energy-Aware Coverage Replanning

We have discussed various techniques for controlling a system optimally in Chapter A. In this section, we use these techniques on the aerial robot. We recall briefly that the control in our model

in Section ?? is the configuration of path and computations parameters in Section 2.3 specified in a given plan Γ; thus, the optimal control is the optimal configuration of both of the paths the aerial robot is flying and the computations it is computing. In our precision agriculture example, the configuration is relative to the coverage (we built the paths for a coverage tour in Section 5.2) and the hazard detection (we discussed the computations running on the robot in Section 2.6.2). To derive the optimal configuration, we derive the optimal control on a finite horizon rather than the entire time frame with MPC or receding horizon predictive control (RHPC), an optimal control technique on a finite horizon (Camacho and Alba, 2007). A problem of the generic optimal control is its difficulty and computational complexity. MPC overcomes this difficulty by optimizing for a bit of the time frame at each optimization step (Camacho and Alba, 2007). MPC forecasts the system behavior and optimizes the forecast to derive a control action (Rawlings et al., 2017). There have been many MPC developments in optimal control literature (Rawlings et al., 2017). These range from robust (where the model is subject to uncertainty), output (where noisy sensors' data estimates a model state), to distributed MPC (that splits the original MPC into sub-problems) (Camacho and Alba, 2007; Kwon and Han, 2005; Rawlings et al., 2017; Rossiter, 2004; L. Wang, 2009).

The models in Section ?? model the energy of computations and the overall energy evolution as a differential equation controlled with the configuration. We use the model in the cost and constraints of the MPC, which is then a convenient technique to derive a configuration that is energy-aware: we only have an empirical approximation of the time needed to cover a given space; instead of considering this approximation, with MPC, we optimize a given horizon. As further motivation, numerous recent studies apply MPC to aerial robots, such as studies for path following (Gavilan et al., 2015), trajectory tracking (Torrente et al., 2021), and involving fixed (Cavanini et al., 2021; Chao et al., 2011; Kang and Hedrick, 2009; Stastny and Siegwart, 2018) and rotary-wings (Bicego et al., 2020; Kostadinov and Scaramuzza, 2020; Song and Scaramuzza, 2020). More closely related to ours, some literature use optimization techniques on a finite horizon to plan the motion and schedule the computations energy-wise (Ondrúška et al., 2015; W. Zhang and J. Hu, 2007), and others use different optimization techniques in this regard (Brateman et al., 2006; Lahijanian et al., 2018). Our contribution extends some of these studies deriving further an energy-aware coverage path along with a power-saving schedule. In Section 3.5, we broadly discuss the available literature for simultaneous planning and scheduling of mobile robots. Here, in Section 5.3.1, we discuss output MPC that we use for replanning of Γ: a technique that uses estimates to refine the state of a model and thus to derive a control action robust to noise (Rawlings et al., 2017). We then provide a further construct to include the battery model from Chapter 4. In Section Section 5.3.2, we dig further into the implementation of output MPC, and finally, in Section 5.3.3, we discuss an algorithm for the coverage replanning and scheduling. This algorithm is central to our approach: it uses the model from Chapter 4 for energy-aware coverage planning and scheduling of the aerial robot flying under tight battery constraints and uncertainty.

5.3.1 Definition of replanning via optimal control

In this section, we derive the optimal control (the configuration of the path and computations parameters) over a finite time horizon N for an estimated state $\hat{\mathbf{q}}$ of the plan Γ. We use the estimated state $\hat{\mathbf{q}}$ in Section A.3, opposed to the ideal state in Chapter 4 due to the uncertainty. The literature commonly refers to this latter problem as output MPC; a variation where estimates refine the state of a perfect model for a system of which the state is not fully known (indeed, the name refers to the notion that some available outputs estimate the state) (Rawlings et al., 2017). For a differential model, such as the periodic model in Equation (4.10) in Section ??, state estimation uses filtering techniques that include the Kalman filter in Section A.3.3.

In our work, we do not know the state of our model (the coefficients of $\mathbf{q}$ that we presented in Section 4.3.2), which we estimate in this section from the energy sensors measurements. Furthermore, the control differs from the nominal control in the model; in the observation in Section 4.3.2, we presented a motivation for such control based on empirical data. The nominal control that we use inside the model maps the change in path and computations parameter to the change in time and power consumption. We use the latter to see how computations affect instantaneous energy (what happens to the power if we increase/decrease computations?), and the former to estimate the time needed to cover a given space and thus to the overall energy consumption (what happens to the energy if we travel more/fewer paths?). To this end, the algorithm that we propose in Section 5.3.3 uses the change in path parameters to verify whenever a given path configuration can be done with the current state of charge (SoC) of the battery, whereas it uses the change in computations parameters to check that the computations configuration is within the battery maximum instantaneous energy consumption. While the former constraint is critical (having less battery SoC than needed would result in an abrupt termination of the flight), the latter does not necessarily imply a plan failure. Indeed the maximum instantaneous energy consumption is an approximated value derived from the battery model in Section 4.2. However, exceeding such value might degrade battery performance since the capacity fade of Li-ion batteries is related to different discharging (and charging) strategies (Lv et al., 2020; Tian et al., 2019). Including this information allows future analysis on different energy-aware methodologies by, e.g., analyzing the cost of sudden spikes on the battery life as opposed to constant energy drain. We have done some earlier analysis in this direction (Seewald, Schultz, Ebeid, et al., 2021), concluding that the spikes that our scheduler generates are not enough to show a visible effect on the battery life. Nonetheless, there are multiple optimizations possible within battery energy awareness, most of which depend on different battery chemistries (Tian et al., 2019) that are not the scope of this work. We discuss different future directions including accurate battery optimizations in Chapter 7.

For the sole purpose of exemplification, we provide a detailed visual example of the observation in Section 4.3.2 relative to the coverage path, showing how a change in the coverage affects the energy. We already discussed the energy effect of the change of computations on instantaneous energy consumption in Section 4.3.2. It is due to different schedules on the computing hardware carried by the aerial robot. We observe a decrement in the power consumption intuitively by

running at lower frames per second (FPS) rate. The energy effect of the change of path is due to the length of the coverage path. We elucidate what we mean by this latter statement in Figures 5.12–5.13, where each figure represents the same coverage but different radiuses r_2 of the fourth circle φ_4 in the primitive paths $\varphi_1, \ldots, \varphi_4$ in Section 2.6.1. Figure 5.12 showed the coverage of the

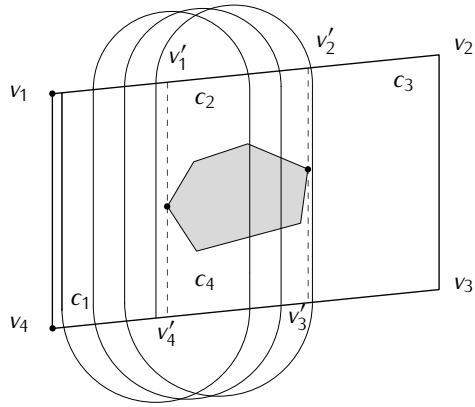


Figure 5.13. The Zamboni-like motion to cover the cell c_1 with the lowest configuration of parameter $c_{4,1}$ relative to the radius of the circle φ_4 in the primitive paths that form the coverage that we introduced in Section 2.6.1.

cell c_1 with the highest configuration of parameter $c_{4,1}$. If we assume that the time needed to perform the circle $\varphi_4(\overline{c}_{4,1})$ is t_3, the vertical lines φ_1, φ_3 is $2t_1$, and the circle φ_2 is t_2, then the overall time to cover c_1 with configuration $c_{4,1} = \overline{c}_{4,1}$ is $t_{\overline{c}_{4,1}} = 7(2t_1 + t_2 + t_3) + t_1$. Conversely, in Figure 5.13 we assume that the time needed to perform $\varphi_4(\underline{c}_{4,1})$ is t_4; then the time needed to cover c_1 with configuration $c_{4,1} = \underline{c}_{4,1}$ is $t_{\underline{c}_{4,1}} = 3(2t_1 + t_2 + t_4) + t_1$. It is clear that $t_4 < t_3$ (for how the parameter $c_{4,1}$ is constructed in Section 2.6.1) and thus $t_{\underline{c}_{4,1}} < t_{\overline{c}_{4,1}}$. If we further assume that traveling all the paths take a similar time $t_1 \approx t_2 \approx t_3 \approx t_4$, then we can observe a 45% time reduction in flying Figure 5.13 compared to flying Figure 5.12. Analogously, in the energy domain, we can expect a considerable reduction in the battery drain with $c_{4,1} = \underline{c}_{4,1}$ compared to $c_{4,1} = \overline{c}_{4,1}$. Our purpose in the remaining of this section is to find the configuration of the path (in the latter case of $c_{4,1}$) along with the computations parameters to maximize the coverage with SoC higher than zero–to find the optimal control on N w.r.t. a given energy cost.

The derivation of such optimal control involves the definition of an OCP and its transformation into a nonlinear program (NLP) (Grüne and Pannek, 2017; Rawlings et al., 2017). Before, however, we re-evaluate the output constraints to include the battery model in Section 4.2. The output of the model in Equation (4.10) is the instantaneous energy consumption y that we stated earlier evolves in $\mathbb{R}$. Nevertheless, there is a limit to the instantaneous energy consumption drainable from a battery at a given time instant. Moreover, aerial robots are bounded by strict energy budgets due to battery limitations, as we motivated in Section 1.3. Hence, we redefine the original output constraint ($\mathbb{R}$) to include the battery model in Section 4.2. We consider SoC b of the

mobile robot's battery with the simplistic differential model in Equations (??–??)

$$\dot{b}(y(t), t) = -k_b \left(V - \sqrt{V^2 - 4R_r y(t)} \right) / (2R_r Q_c), \tag{5.8}$$

where k_b is the battery coefficient determined experimentally, $V \in \mathbb{R}$ is the internal battery voltage measured in volts, $R_r \in \mathbb{R}$ the resistance measured in ohms, and $Q_c \in \mathbb{R}$ the constant nominal capacity measured in amperes per hour.

From the literature on the battery SoC (Deng et al., 2017; Kurzweil and Scheuerpflug, 2021; Kurzweil and Shamonin, 2018; Sundén, 2019), we know that this latter can be calculated as $b(y(t), t) = Q(y(t), t)/Q_c$ where $Q(y(t), t)$ is the available capacity at a given time t. For simplicity, we omit further details that interfere in the calculation, such as battery state of health, temperature, and C-rate. Indeed that are other methods in the literature (Espedal et al., 2021; L. Lu et al., 2013; R. Zhang et al., 2018) to estimate more accurately the SoC together with other battery parameters (these are, however, beyond the scopes of our work of coverage planning and scheduling). From the previous expression and the estimated SoC in Equation (5.8), we can calculate the maximum instantaneous energy consumption by multiplying the constant nominal capacity, the SoC, and the internal battery voltage. We assume the maximum energy consumption cannot be negative

$$0 \leq y(t) \leq b(y(t), t) Q_c V, \tag{5.9}$$

and therefore, we define a time-varying constraint for the output in Definition 5.3.1, being the maximum instantaneous energy consumption dependent on the SoC b from Equation (5.8), which is dependent on time and the instantaneous energy consumption (at the previous time step).

Definition 5.3.1: Output constraint

$$\mathcal{Y}(t) := \{ y \mid y \in [0, b(y(t), t) Q_c V] \subseteq \mathbb{R}_{\geq 0} \},$$

is the *output constraint*, where $b(y(t), t) Q_c V$ is the maximum instantaneous energy consumption.

We assume the mobile robot carries a battery energy sensor and obtain the initial SoC $b(y(t_0), t_0)$ in the output constraint using the output of such sensor. This is a realistic assumption: aerial robots are often equipped with a flight controller, which returns various metrics, including the battery SoC. Evaluating the constraint requires numerical simulation: the battery model in Equation (5.8) is differential, similarly to the energy dynamics of the periodic model in Equation (4.10). We can compute the numerical simulation using the Euler method in Section A.4.1 or the Runge-Kutta method in Section A.4.2.

To state the OCP on a finite horizon, we use a similar expression to Section A.2.2 with

$$\max_{\mathbf{q}(t),c_i(t)} l_f(\mathbf{q}(T),T) + \int_{t_0}^{T} l(\mathbf{q}(t),c_i(t),t)\,dt, \tag{5.10a}$$

$$\text{s.t. } \dot{\mathbf{q}} = f(\mathbf{q}(t),c_i(t),t), \tag{5.10b}$$

$$c_i(t) \in \mathcal{U}_i, \mathbf{q}(t) \in \mathbb{R}^m, \tag{5.10c}$$

$$y(t) \in \mathcal{Y}(t), \tag{5.10d}$$

$$\mathcal{S}_{i,j} := \{0\},\ \forall j \in [\sigma] \text{ when } \mathbf{p}(t) \notin \mathcal{Q}^v, \tag{5.10e}$$

$$\mathbf{q}(t_0) = \hat{\mathbf{q}}_0 \text{ given (last estimate state), and} \tag{5.10f}$$

$$b(y(t_0),t_0) = b_0 \text{ given,} \tag{5.10g}$$

where constraints in Equations (5.10b–5.10e) are evaluated on $t \in [t_0, T]$. $\mathbf{q}(t)$ and $c_i(t)$ are the state and control trajectories and $\mathbf{p}(t)$ is the aerial robot's position w.r.t. an inertial navigation frame $\mathcal{O}_W$. The sizes of the state and control (m and n) are defined in Section ?? and Section 4.3.2. By solving the OCP in Equation (5.10), we want to derive the trajectory $c_i^*(t) = \{c_{i,1}^*, \ldots, c_{i,\rho}^*, c_{i,\rho+1}^*, \ldots, c_{i,\rho+\sigma}^*\}$ in Section 2.3 for a given stage i in Definition 2.3.1. In the max term in Equation (5.10), we further derive the trajectory of the ideal state $\mathbf{q}(t)$, which we can use to evaluate the model's fidelity against future measured data. Indeed the solution to the OCP has to evolve the model from trained data using state estimation up to the initial time instant t_0. At the very beginning of the optimization (when $t_0 = 0$), we will see that the perfect model evolution in Equation (5.10b) does not correspond to the data despite converging later on: for successive horizons, there $\exists k$ s.t. $t_0 = kN$ and the model in Equation (5.10b) converges for $\mathbf{q}(kN) = \hat{\mathbf{q}}_{kN}$. Our constraints contain the control and state constraint in Equation (2.6), the output constraint in Definition 5.3.1, and the dynamics with the ideal state evolution in Equation (4.10) in Equations (5.10b–5.10d). The OCP further contains the coverage constraint from Section 2.6.1 and Section 5.2.2 in Equation (5.10e). Although the aerial robot can overfly the obstacles and the edges of the polygon, it cannot compute any computation. Formally, we inhibit the computations setting their constraint to $\{0\}$ when the aerial robot is flying over the ith obstacle o_i and out of the polygon v (it is thus out of the space $\mathcal{Q}^v$). We have similarly inhibited the computations out of the polygon v in Section 2.6.1.

The dynamic evolution in Equation (5.10b) is then the periodic model in Equation (4.10) together with the scale transformation from Section 4.3.3

$$f(\mathbf{q}(t),c_i(t),t) = A\mathbf{q}(t) + B\text{diag}(\nu_i)(c_i(t) - c_i(t-\Delta t)), \tag{5.11}$$

where $c_i(t-\Delta t)$ is the control at the time instant preceding t, A is the state transition matrix in Equation (4.12), B the input matrix in Equation (4.46), and ν_i is the scale transformation in Equation (4.47) with the scaling factors that for the first ρ path parameters are given in Equation (4.48) and for the remaining computations parameters in Equation (4.49).

The instantaneous cost function is the quadratic expression

$$l(\mathbf{q}(t),c_i(t),t) = \mathbf{q}'(t)Q\mathbf{q}(t) + c_i'(t)Rc_i(t), \tag{5.12}$$

where $Q \in \mathbb{R}^{m\times m}$, $R \in \mathbb{R}^{n\times n}$ are given positive semidefinite matrices, resulting in the convexity of the cost l (Nocedal and S. Wright, 2006) with some guarantees on the solution (Beck, 2014).

The final cost function is alike a quadratic expression but with no control

$$l_f(\mathbf{q}(T), T) = \mathbf{q}'(T)Q_f\mathbf{q}(T), \tag{5.13}$$

where $Q_f \in \mathbb{R}^{m\times m}$ is a given positive semidefinite matrix.

The horizon N is in seconds, and the controller selects the optimal control trajectory $c_i^*(t)$ over $[t_0, T]$, with $T = t_0 + N$. At each instant, the controller refines the control trajectory (replans the coverage and schedule) that respects the constraints. To evaluate the state trajectory–needed for the instantaneous cost function l and the final cost function l_f–the controller evaluates the battery trajectory $b(y(t), t)$ (it has to verify that the output is in $\mathcal{Y}(t)$). It then maximizes the instantaneous cost function l for all the time instants but T ($t_0 \leq t < t_0 + N$), and the final cost function l_f in Equation (5.13) for the time instant T ($t = t_0 + N$). The dynamics constraint satisfaction requires to evolve the perfect model f in Equation (5.11) over horizon $[t_0, t_0 + N]$ beginning from the last estimated state $\mathbf{q}_0 = \hat{\mathbf{q}}(t_0)$ at time instant t_0. Similarly, the output constraint satisfaction requires then to evolve the battery constraint on $[t_0, t_0 + N]$, from the last battery measurement b_0 from the battery energy sensor at instant t_0.

The OCP from Equation (5.10) is infinite-dimensional, being the system dynamics in Equation (5.10b) and the battery dynamics in Equation (5.8) given in continuous- and not discrete-time. Such an infinite dimensional OCP has an infinite-dimension of constraints and decision variables since there are infinitely many time instants between t_0 and $t_0 + N$. We discretize the OCP to finite dimensions in Section 5.3.2.

5.3.2 Replanning with output model predictive control

In this section, we solve the OCP in Equation (5.10), providing a solution to Problem 2.5.1 that we introduced along with Problem 2.5.2 in Chapter 3. We use the notions from the previous sections and Chapters 4–A: with the solution, we derive the configuration of paths and computations c_i^* for each stage i in an energy-aware fashion to replan the plan Γ. The solution to Problem 2.5.2 is the plan Γ, which contains the primitive paths for the coverage motion in Section 5.2.2 and Section 2.6.1, along with the computations in Section 2.6.2.

We are interested in providing this solution online, with a procedure running onboard the aerial robot subject to various atmospheric interferences. There are different techniques for this purpose (Grüne and Pannek, 2017; Rawlings et al., 2017). In Chapter A, we saw some relevant to our approach, which are the direct methods already employed in many MPC-related applications (Rawlings et al., 2017). In particular, we saw single and multiple shooting methods in Sections A.5.1–A.5.2. They parametrize the control and state trajectories with a finite-dimensional vector and derive an NLP (Rawlings et al., 2017). Both the methods perform this latter discretization, but the resulting NLP differs. The multiple shooting method keeps the states as decision variables at the boundary time points, allowing further optimizations (Rawlings et al., 2017). It combines some of the advantages of the other methods, such as the direct collocation method

that we discussed in Section A.5.3, together with the direct single shooting method (Diehl et al., 2006; Grüne and Pannek, 2017). Once we obtain the finite-dimensional NLP, we can solve it numerically, with the numerical solvers available in the literature (Diehl et al., 2006; Grüne and Pannek, 2017; Nocedal and S. Wright, 2006).

The NLP derived from the OCP in Equation (5.10) is

$$\max_{\mathbf{q}(k),c_i(k)} l_f(\mathbf{q}(T),T)+\sum_{k\in\mathcal{K}} l_d(\mathbf{q}(k),c_i(k),k), \tag{5.14a}$$

$$\text{s.t. } \mathbf{q}(k+h)=f_d(\mathbf{q}(k),c_i(k),k), \tag{5.14b}$$

$$c_i(k)\in\mathcal{U}_i,\mathbf{q}(k)\in\mathbb{R}^m, \tag{5.14c}$$

$$y(k)\in\mathcal{Y}(k), \tag{5.14d}$$

$$\mathcal{S}_{i,j}:=\{0\},\ \forall j\in[\sigma]\text{ when }\mathbf{p}(k)\notin\mathcal{Q}^v, \tag{5.14e}$$

$$\mathbf{q}(t_0)=\hat{\mathbf{q}}_0\ \text{ given (last estimated state), and} \tag{5.14f}$$

$$b(y(t_0),t_0)=b_0\ \text{ given,} \tag{5.14g}$$

where the constraints in Equations (5.14b–5.14e) are evaluated now on a finite interval $k\in\mathcal{K}=\{t_0,t_0+h,t_0+2h,\dots,T\}$, and h is a given time step or distance between two consecutive time instants; the smaller the distance, the more precise the simulation. The other expressions are analogous to Equation (5.10).

We use numerical simulation to transform Equation (5.10) into Equation (5.14). We can use either the Runge-Kutta methods in Section A.4.2 or the Euler method in Section A.4.1. For simplicity, we show the transformation with the Euler method. The instantaneous cost function

$$l_d(\mathbf{q}(k),c_i(k),k)=hl(\mathbf{q}(k),c_i(k),k), \tag{5.15}$$

where l is given in Equation (5.12).

The discrete dynamic evolution in Equation (5.14b)

$$f_d(\mathbf{q}(k),c_i(k),k)=A_d\mathbf{q}(k)+B\text{diag}(\nu_i)(c_i(k)-c_i(k-h)), \tag{5.16}$$

where A_d is the discretized version of the state transition matrix A in Equation (4.12) and for a small enough interval of h

$$A_d=(hA+\text{diag}(1,1,\dots,1)), \tag{5.17}$$

where $\text{diag}(1,1,\dots,1)\in\mathbb{R}^{m\times m}$ is a diagonal matrix of ones. B is then the input matrix in Equation (4.46), and ν_i is the scale transformation in Equation (4.47) with the scaling factors that for the first ρ path parameters are given in Equation (4.48) and for the remaining σ computations parameters in Equation (4.49), similarly to Equation (5.11).

To discretize the battery dynamics in Equation (5.8), we use

$$b_d(y(k+h),k+h)=b(y(k),k)+hb(y(k+h),k+h), \tag{5.18}$$

where b is given in Equation (5.8) and $y(k)$ is the output of the model in Equation (4.10b). The matrix C is to be found in Equation (4.14).

In Equation (5.14), we transformed the OCP in Equation (5.10) into an NLP by first discretizing and thus effectively implementing the direct multiple shooting method in Section A.5.2. We note that we can implement the single shooting method by keeping only the initial state as the decision variable $\hat{\mathbf{q}}_0$, conversely to using the interval boundary time points as a decision variable in the multiple shooting method (Rawlings et al., 2017).

5.3.3 Coverage planning and scheduling algorithm

Algorithm 5.3 derives the optimal configuration of path and computations parameters and thus computes the coverage (re)planning and scheduling. It inputs the initial plan Γ along with the initial time step and a guess for the energy model that we proposed in Section A.3.2 and outputs the revised plan Γ when the aerial robot reaches the final point $\mathbf{p}_{\Gamma_l}$. It optionally inputs then the initial stage that is within $[n]_{>0}$ when Γ compromise primitive stages, $[l]_{>0}$ otherwise. In the remainder of this chapter, we discuss in detail the algorithm.

The algorithm iterates at each time step $h \in \mathbb{R}_{>0}$ on Line 1, with the size of h related to the accuracy and the time needed for replanning: the higher the step, the lower the accuracy, but the shorter the necessary time to replan Γ. Nonetheless, there are further constraints on h due to the numerical simulation algorithm: h has to be small enough ($h \to 0$) so that the numerical simulation (on Line 10, 22, 14, and 15) does not diverge. To this end, a typical value for h that we adopted is 1/100 of a second; there are various techniques to estimate h more accurately, e.g., by running the numerical simulation and analyzing the error of two different guesses (Iserles, 2009); such techniques are, however, beyond the scopes of this work. With a small enough h, Euler method that we use in Equations (5.15–5.18) converges to the real solution–see (Atkinson et al., 2009; Iserles, 2009) for a formal proof–whereas for problems with higher accuracy we advise the Runge-Kutta methods in Section A.4.2 that are among the most popular methods for numerical simulation (Atkinson et al., 2009). These methods use quadrature: a procedure that replaces the integral with a finite sum (Iserles, 2009). We use the Runge-Kutta inside `powprofiler` in Section 4.1.4 and the Euler method in the algorithm for simplicity.

On Lines 2–3, the algorithm verifies whenever the aerial robot reached the final point $\mathbf{p}_{\Gamma_l}$ in Definition 2.3.2 and returns the replanned plan Γ in this latter eventuality, and on Lines 4–9, switches the stages when the aerial robot reaches a triggering point (also in Definition 2.3.2). If Γ contains n primitive paths rather than all the stages up to l explicitly, on Lines 8–9, it updates the paths and the triggering points with the shift $\mathbf{d}$. It then selects the energy-aware configuration of computations on Line 10 by solving the NLP in Equation (5.14) derived from the OCP in Equation (5.10) using the multiple shooting method in Section A.5.2. It thus obtains the trajectory of the controls and states for a given horizon N expressed in seconds. In particular, the control trajectory contains $|\mathcal{K}| - 1$ items, whereas the state trajectory $|\mathcal{K}|$ items. This discrepancy in the number of sets of the two trajectories is due to the construction of OCP. We provide an initial guess for the state $\mathbf{q}(i)$ at time instant i but derive the first control $c_j(i)$ already from the solution to the OCP. For the horizon N, we adopted the value of N equal to ten seconds, meaning the algorithm evolves the model in Equation (5.11) and Equation (4.10) for ten future seconds and

Input : Γ initial plan
t_0 initial time step
$\hat{\mathbf{q}}(t_0)$ initial guess for the energy model
j starting stage within the primitive stages $\in [n]_{>0}$ otherwise $j = 1$
Output : Γ revised plan

1 **foreach** $i \in \{t_0, t_0 + h, t_0 + 2h, \dots\}$ **do**
2 **if** $\mathbf{p}(i) \neq \mathbf{p}_{\Gamma_l}$ **then**
3 return Γ
4 **if** $\mathbf{p}(i) = \mathbf{p}_{\Gamma_j}$ **then**
5 $j \leftarrow j + 1$
6 **if** $j \notin [n]_{>0}$ **then**
7 $j \leftarrow 1$
8 $\varphi_1, \varphi_2, \dots, \varphi_n \leftarrow$ shift $\varphi_1, \varphi_2, \dots, \varphi_n$ of $\mathbf{d}$
9 $\mathbf{p}_{\Gamma_1}, \mathbf{p}_{\Gamma_2}, \dots, \mathbf{p}_{\Gamma_n} \leftarrow$ shift also $\mathbf{p}_{\Gamma_1}, \mathbf{p}_{\Gamma_2}, \dots, \mathbf{p}_{\Gamma_n}$ of $\mathbf{d}$
10 $\mathbf{q}(\mathcal{K} \setminus \{i + N\}), c_j(\mathcal{K}) \leftarrow$ solve NLP $\arg\max_{\mathbf{q}(k), c_j(k)} l_f(\mathbf{q}(i + N), i + N) + \sum_{k \in \mathcal{K}} l_d(\mathbf{q}(k), c_j(k), k)$ from Equation (5.14) on $\mathcal{K} = \{i, i + h, \dots, i + N\}$
11 $k \leftarrow i$
12 **while** $b_d(k) > 0$ **do**
13 **if** $k \notin \mathcal{K}$ **then**
14 $\mathbf{q}(k + h) \leftarrow A_d \mathbf{q}(k)$
15 $b_d(k + h) \leftarrow b_d(k) - h k_b \left(V - \sqrt{V^2 - 4 R_r C \mathbf{q}(k + h)} \right) / (2 R_r Q_c)$
16 $k \leftarrow k + h$
17 $t_b \leftarrow k - i$
18 $t_s \leftarrow (\mathrm{diag}(v_j^\rho) c_j^\rho(i) + \tau_j^\rho)[\overbrace{1 \quad 1 \quad \cdots \quad 1}^{\rho}]$
19 $t_r \leftarrow (t_s / \bar{t})(\bar{t} - i)$
20 **if** $t_r < t_b$ **then**
21 $c_j^\rho(i) \leftarrow$ find c_j^ρ with $t_c \in [0, t_b]$, otherwise take $\underline{c}_j^\rho$
22 $\hat{\mathbf{q}}(i + h) \leftarrow A_d \hat{\mathbf{q}}(i) + B \mathrm{diag}(v_j)(c_j(i) - c_j(i - h))$
23 $P(i + h)^- \leftarrow A_d P(i) A_d' + S(i)$
24 $K(i + h) \leftarrow (P(i + h)^- C') / (C P(i + h)^- C' + V(i))$
25 $\hat{\mathbf{q}}(i + h) \leftarrow$ compute $\hat{\mathbf{q}}(i + h) + K(i + h)(y(i) - C \hat{\mathbf{q}}(i + h))$ from energy sensor $y(i)$
26 $\hat{y}(i + h) + C \hat{\mathbf{q}}(i + h)$
27 $P(i + h) \leftarrow (I + K(i + h) C) P(i + h)^-$
28 $\mathbf{p}(i + h) \leftarrow$ compute from position $\mathbf{p}(i)$, velocity $v(i)$ with $\Delta_d \varphi_j$ in Algorith 5.2

Algorithm 5.3. Coverage (re)planning and scheduling algorithm

selects the control trajectory c_j^* that minimizes the costs in Equation (5.15) and Equation (5.13). The higher the horizon, the better the accuracy. A typical value in the aerial robotics literature that implements MPC (Chao et al., 2011; Gavilan et al., 2015; Kang and Hedrick, 2009; Stastny and Siegwart, 2018) is of dozens of seconds. For instance, it is fourteen in Gavilan et al., ten and forty in Kang and Hedrick, two to eight in Stastny and Siegwart, and five in Chao et al. To solve the NLP in Equation (5.14), the algorithm evaluates the constraints in Equations (5.14b–5.14e) (t_0 in the latest two constraints is equal to i), where it has to numerically simulate both the energy and the battery models from i to $i + N$ with a step size of h. A possible optimization that considerably speeds up the derivation of the optimal control is to use different horizons for numerical simulation and the constraints, i.e., $\mathcal{H} = i, i + nh, i + 2h, i + N$, where $n \in [1, 1/h]$. When n is equal to $1/h$, the algorithm adds the constraints at each second but numerically simulates the models on $\mathcal{K}$. Another optimization is to move the constraint in Equation (5.14e) out of the NLP and check whenever the aerial robot is flying the obstacle and thus require $c_j^\rho = 0$.

On Lines 22–27, the algorithm estimates the state $\mathbf{q}$ of the periodic model in Section ??, filtering the state with Kalman filter in Section A.3.3 from the energy sensor's measurements $y(i)$. The Kalman filter has several important properties over other methods for state estimation, and among the others, minimizes the variance of the estimation mean square error (MSE) (Jwo and Cho, 2007; Kalman, 1960; Simon, 2006). On Line 22, the algorithm computes the a priori, and on Line 25 the a posteriori state estimation. On Line 23, the a priori covariance state error with P the covariance of the state estimation error (we provide an initial guess $P(t_0) \in \mathbb{R}^{m \times m}$, the covariance error of $\mathbf{q}(t_0)$) and $S \in \mathbb{R}^{m \times m}$ the covariance of the state noise, and on Line 24, the gain with $V \in \mathbb{R}$ the covariance of the output noise (Simon, 2006). To ease the computations, we keep the covariances fixed in our experimental setup.

On Lines 11–17, the algorithm estimates the time needed to completely drain the battery with the differential model in Section 4.2, in Equation (5.8), and Equation (5.18), using the Euler method for numerical simulation (as on Line 22). A possible optimization in this set of lines is to utilize the state already from MPC on Line 10 for future energy prediction on the horizon N; at further time horizons, we don't have any trajectory for the control available, and thus we keep $c_j(k) = c_j(i + N - h)$, $\forall k > i + N - h$. The variable t_b on Line 17 contains the estimated available battery time. The algorithm then computes the scale transformation from the path parameters domain to the time domain in Section 4.3.3 on Line 18 and the estimated remaining time of the path parameters $c_j^\rho(i)$ on Line 19. It selects a different configuration of path parameters when the battery time is lower than the remaining time on Line 21. We can further assume that the battery information is incomplete or that the model does not account for all the eventualities and select the best configuration for the current total battery time. Finally, on Line 28, the algorithm computes the next position in space using the vector field for guidance in Section 5.1, where $v(i) \in \mathbb{R}_{\geq 0}$ is the velocity and $\mathbf{p}(i)$ the position at the current time step i.

5.4 Results

5.5 Summary

Chapter 6

Results

> *"[Computing energy included motion planning] shows improved performance over the baseline and looks to be promising solution to the low-power motion planning problem."*
>
> — Sudhakar et al., 2020

In this chapter, we report some results both published in our early studies (Seewald, Ebeid, et al., 2019; Seewald, Garcia de Marina, Midtiby, et al., 2020; Seewald, Schultz, Ebeid, et al., 2021; Seewald, Schultz, Roeder, et al., 2019; Zamanakos et al., 2020) and to appear in a forthcoming study (Seewald, Garcia de Marina, and Schultz, n.d.), validating our overall approach towards energy-aware dynamic planning and scheduling for autonomous aerial robots. The chapter thus connects to the remainder of the work by describing our experimental setup and results. We describe the latter for Chapters 4–5, which contains our most notable contribution: the energy models for both the computation and motion of an aerial robot and a coverage planning and scheduling technique that uses the models. Chapter 2 is limitedly involved as well: we extensively use the agricultural scenario we introduced in Section 2.6 to showcase our approach, along with other formalities we proposed in the chapter.

The remainder of this chapter is structured as follows. In Section 6.1, we describe our experimental setup and results for the computations energy model obtained with the `powprofiler` tool from Section 4.1.4. We then detail similarly the energy of the motion of an aerial robot flying a path similar to Section 2.6 independently and along with the computing hardware in Section 6.2. In both Sections 6.1–6.2, we describe our experimental setup for a batter of the computing hardware, aerial robot, and computing hardware with the aerial robot, detailing for each the results. In Section 6.3, we then describe the coverage planning and scheduling in MATLAB (R) and Paparazzi autopilots and thus, validate our work experimentally.

6.1 Computations Energy Modeling

In this section, we describe the results and the experimental setup to obtain computations energy models with the `powprofiler` tool. We report how we derive both the measurement and predictive layers from Sections 4.1.2–4.1.3 and the battery models from Section 4.2 to integrate the two layers with the battery energy contribution for the computing hardware.

6.1.1 The `darknet-gpu` computation

In Figure 6.1, we illustrate a concrete example of four different measurement layers from our early contribution (Seewald, Schultz, Ebeid, et al., 2021): the power evolution in the function of time for three measuring devices, CPU, GPU, and overall of the NVIDIA Jetson TX2 board. The

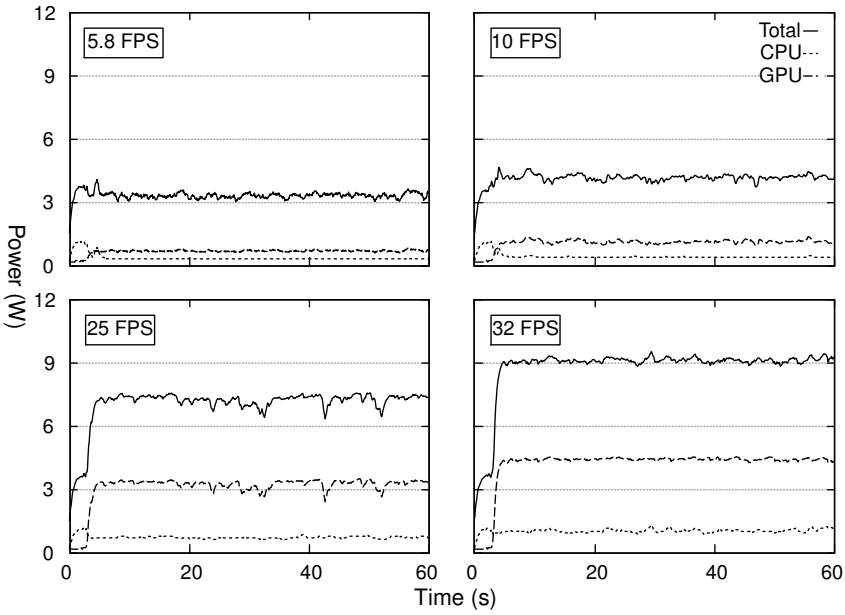


Figure 6.1. The `darknet-gpu` computation measurement layer models, featuring the GPU implementation of the YOLO (Redmon, Divvala, et al., 2016) deep neural network library modified to introduce delays between detections. From top left in horizontal order to bottom right, the figure shows the schedules from approximately six up to thirty-two FPS. The figure appeared in our early study (Seewald, Schultz, Ebeid, et al., 2021).

model is relative to one computation with four different schedules where we observed notable differences in instantaneous and overall energies. The computation consists of an object detection algorithm that we term `darknet-gpu`, a neural network-based pattern recognition utility. It is a standard computer vision computation, built upon the darknet (Redmon, 2013–2016; Redmon and Farhadi, 2017) GPU implementation of a deep neural network library YOLO (Redmon,

Divvala, et al., 2016), which detects the objects on some pre-trained as well as trained networks (the latter to detect a personalized set of objects).

In an initial iteration of our work (TeamPlay Consortium, 2019), we trained the network using images of different shapes and colors for a search and rescue aerial robotics scenario where the robot detects vessels on the sea. We benchmarked the corresponding computation on a video stream of vessels in an offshore area, simulating an aerial robot flying the scenario with a camera. We further modified the `darknet-gpu` computation to simulate different scheduling options, such that the computation can be altered with a given parameter, which we call $c_{i,1}$ in accordance to Definition 2.3.1 $\forall\, i \in [l]$ (with the plan lasting assigned l stages). The parameter indicates the delay between two invocations of the detection, simulating different frames per second (FPS) rates. We schedule the computation with the parameter $c_{i,1}$ to assess the predictive layer in

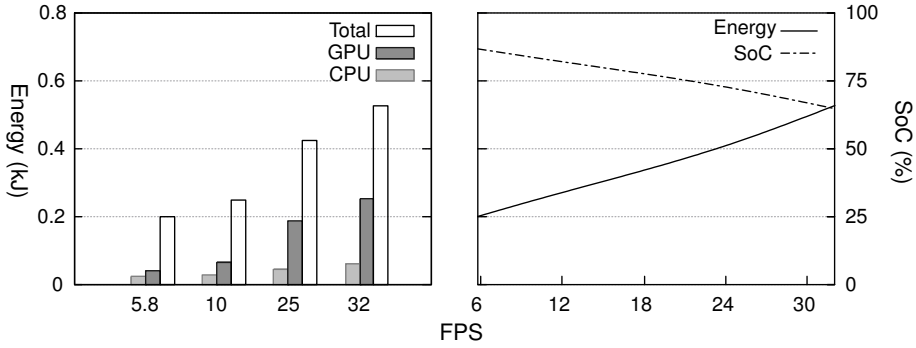


Figure 6.2. Per-minute energy consumption and SoC of the `darknet-gpu` computation in terms of CPU, GPU, and overall energies. On the right is the resulting predictive layer that shows the energy and battery SoC in the function of any possible configuration. The figure appeared in our early study (Seewald, Schultz, Ebeid, et al., 2021).

Figure 6.2. On the right of the figure, we further illustrate the battery state of charge (SoC) in the function of varying FPS for the overall power measuring device of the TX2 board (dashed line). Similarly, the predictive layer provides the evolution of the energy also in the function of varying FPS (continuous line).

We used a frequency of ten hertz and a running time of one minute to derive all the measurement layers. The energy measure in Figure 6.2 is then relative to the energy needed to run the computation for a minute and the correspondent remaining SoC. For the latter, we used an integration step of one hundredth, internal battery voltage V of 14.8 volts, internal resistance R_r of 1.2 milliohms, stabilized voltage V_s of twelve volts, and battery capacity Q_c of five amperes per hour in Equations (4.5–4.8) in Section 4.2.2. The parameters that correspond to such configurations are `frequency=10`, `h=0.01`, and `runtime=60000` in the specification in Section 4.1.5, whereas the battery values are specified while invoking the constructor of the `soc_1resistor` class. Both Figures 6.1–6.2 show the parameter $c_{i,1}$ within $\mathcal{S}_{i,1}$ constructed so that the resulting FPS is between 5.8 and thirty-two.

6.1.2 The `matrix-gpu` computation

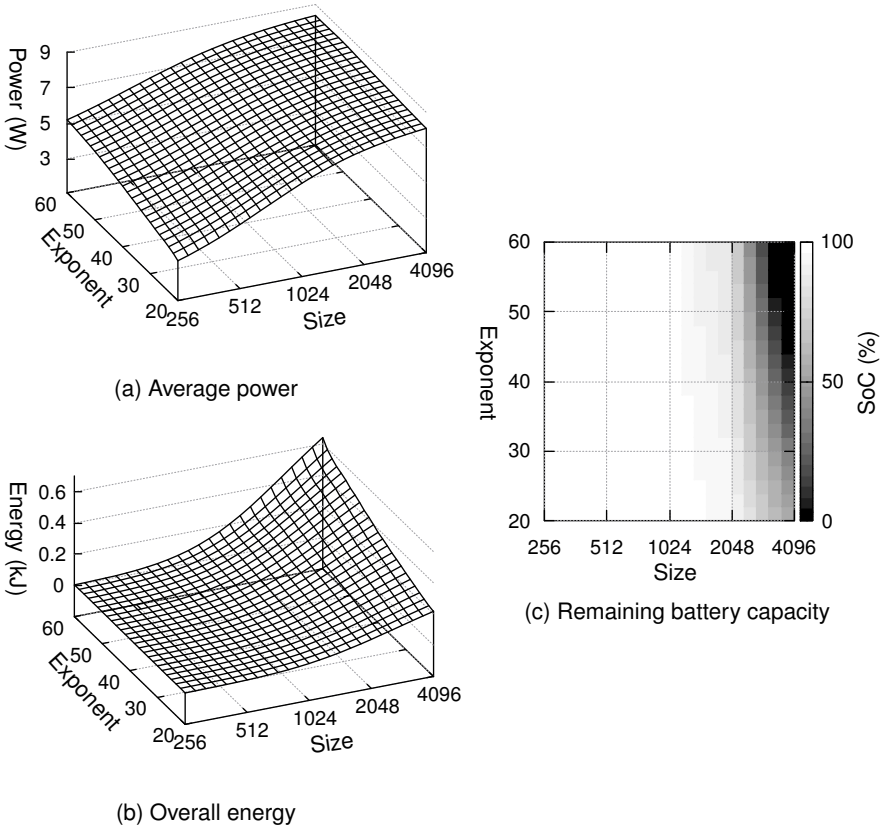
We then derived a set of measurement and predictive layers of another computation, `matrix-gpu`, also related to our early study (Seewald, Schultz, Ebeid, et al., 2021). Here we use again the NVIDIA Jetson TX2 computing hardware and its overall power measuring device (the computing hardware supports measurements of the power for CPU and GPU separately, as well as overall; see Figure 6.1 or Section 4.1.1). `matrix-gpu` computes the matrix exponentiation on the GPU of various matrix sizes using parameter $c_{i,1}$, with different exponents using $c_{i,2}$ and delaying the intermediate steps of the exponentiation with different times using $c_{i,3}$ (we assume that the parameters are the same for all the l stages). The exponentiation is meant to simulate the heavy computational load of, e.g., an algorithm related to computer vision. Indeed these algorithms often rely on various operations with matrix representations of images where, for instance, color balancing (i.e., incandescent lighting compensation) occurs by multiplication with a scale factor (Szeliski, 2011). We illustrate the predictive layers for the computation varying the parameter $c_{i,1}$ from 256 to 4096 and $c_{i,2}$ from twenty to sixty in Figure ?? (these values enclose the sets $\mathcal{S}_{i,1}$ and $\mathcal{S}_{i,2}$ respectively). Particularly, Figure 6.3a shows the average power, Figure 6.3b the overall energy, and Figure 6.3c the battery SoC as a function of matrix size and exponent. We report the average power consumption independently of the running time of the `matrix-gpu`, whereas the overall energy measures are measured up until a given configuration of the two parameters $c_i = \{c_{i,1}, c_{i,2}\}$ evaluated the exponentiation, as well as for the battery state of charge. The computation might pose no notable effect on the total power consumption for values of c_i close to $\underline{c}_{i,1}$ and $\underline{c}_{i,2}$ for parameters $c_{i,1}$, $c_{i,2}$. An effect indicating the computation terminated before reaching the maximal power level (Seewald, Schultz, Ebeid, et al., 2021), also visible in Figure 6.1. It is the reason why there is a notable difference in average power for two opposite configurations.

In Figure 6.4, we illustrate the predictive layers by varying the parameter $c_{i,1}$ and $c_{i,3}$ relative to the delay–or "sleep"–between iterations of the matrix exponentiation from none to ten seconds. Figure 6.4a shows the average power, Figure 6.4b the overall energy, and Figure 6.4c the battery SoC as a function of matrix size and the sleep. We observe that the latter directly affects the overall power consumption hence, the higher the delay, the lower the remaining battery SoC (conversely, the higher the overall energy). In both set of Figures 6.3b–6.3c and Figures 6.4b–6.4c, we thus observe a relation between overall energy and battery SoC.

We used the same configuration parameters as with the `darknet-gpu` computation but for the running time. In Figures ??–6.4, the runtime is unbounded: the average power, overall energy, and battery SoC are relative to the entire exponentiation. The battery parameters are likewise the same. To define the constraint sets $\mathcal{S}_{i,1}$, we used the configuration `pow`, e.g., `range=256,4096,pow(2)`, meaning $c_{i,1}$ utilize exponential sampling in Equation (4.2).

6.1.3 The `darknet/matrix -cpu`, `nvidia- matrix/quicks` computations

We deployed the computations energy model on additional computations and computing hardware, including `matrix-cpu` and `darknet-cpu`, similar to `matrix-gpu` and `darknet-gpu` compu-



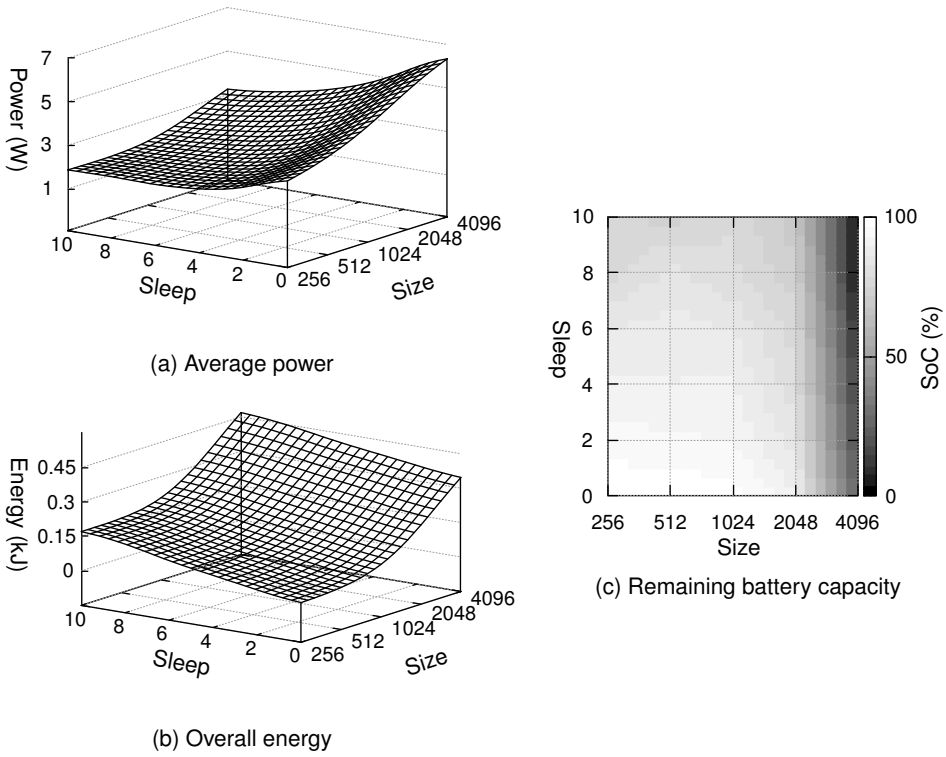
(a) Average power

(b) Overall energy

(c) Remaining battery capacity

Figure 6.3. Predictive layers with the power in Figure 6.3a, energy in Figure 6.3b, and battery SoC power in Figure 6.3c of the `matrix-gpu` component in the function of varying matrix size and exponent. The figure appeared in our early study (Seewald, Schultz, Ebeid, et al., 2021).

tations except that the operations run on the CPU rather than GPU. We used all the computing hardware described in Section 4.1.1 for `matrix-cpu`, and NVIDIA Jetson TX2 for `darknet-cpu`. On NVIDIA Jetson TX2, we further interfaced some already existing benchmarks with the `powprofiler` tool. These were modified to run for several instances over a time interval and implement a quicksort (`nvidia-quicks`) and a matrix multiplication (`nvidia-matrix`), both with a given problem size (parameter $c_{i,4}$). We summarize the overall energy contribution of all the computations from this section in Table 6.1. The table shows the performance while running on different computing hardware and heterogeneous elements. For instance, `matrix-gpu` requires considerably more energy compared to `matrix-cpu`; indeed, there is a large performance gap between GPUs and general-purpose multicore CPUs in terms of heavily parallelizable computations (D. B. Kirk and Wen-Mei, 2016): `matrix-gpu` requires only 4.5 joules, whereas `matrix-cpu` 241.3 joules for the same operation on the NVIDIA Jetson TX2 computing hardware (Seewald,



(a) Average power

(b) Overall energy

(c) Remaining battery capacity

Figure 6.4. Predictive layers with the power in Figure 6.3a, energy in Figure 6.3b, and battery SoC power in Figure 6.3c of the `matrix-gpu` component in the function of varying matrix size and delays between consecutive iterations, simulating different schedules. The figure appeared in our early study (Seewald, Schultz, Ebeid, et al., 2021).

Schultz, Ebeid, et al., 2021). The battery SoC evolves accordingly, with the difference between `matrix-gpu` and `matrix-cpu` in terms of battery SoC being 16%. In our early work (Seewald, Schultz, Ebeid, et al., 2021; Seewald, Schultz, Roeder, et al., 2019), we further observe the parallelization effect on the overall energy. We can conserve energy by running computations parallel on the CPU and GPU compared to a sequential schedule (e.g., scheduling computations sequentially on CPU and GPU in some order) even if we subtract the base power. This latter results in a 20% larger overall energy consumption than a parallel schedule (Seewald, Schultz, Ebeid, et al., 2021).

In Table 6.1 we additionally compare `darknet-gpu` to `darknet-cpu`, which is similar to its GPU equivalent but runs merely on the CPU. Although `darknet-cpu` requires less energy per minute compared to `darknet-gpu`, it runs for considerably longer; the energy cost per frame is then higher on the CPU than on GPU implementation (Seewald, Schultz, Ebeid, et al., 2021). We further show the energy effect of the `nvidia-matrix` and `nvidia-quicks` computations on the NVIDIA Jetson TX2 computing hardware. The `nvidia-matrix` computation runs a sig-

Computation	ODROID XU3	NVIDIA		
		TK1	TX2	Nano
`matrix-cpu`	528.4 J	406.7 J	241.3 J	273.6 J
`matrix-gpu`	-	8.1 J	4.5 J	3.9 J
`darknet-cpu`	(-)	(-)	240 J	(-)
`darknet-gpu`	-	-	525.5 J	(-)
`nvidia-matrix`	-	(-)	405.4 J	(-)
`nvidia-quicks`	-	(-)	199.5 J	(-)

Table 6.1. Overall energy per computation on different computing hardware. Unsupported hardware are indicated by "-", whereas possible future support by "(-)" as it appeared in our early study (Seewald, Schultz, Ebeid, et al., 2021).

nificant portion on the CPU to check whenever the result of the GPU matrix multiplication matches the one on the CPU. Nonetheless, we observe that the problem size affects the overall energy consumption and battery SoC in both the `nvidia-matrix` and `nvidia-quicks` computations (Seewald, Schultz, Ebeid, et al., 2021). One notable difference between the two latter computations is the nature of the problem they solve; `nvidia-quicks` uses random data that are being sorted and has thus lower predictability in terms of overall energy (Seewald, Schultz, Ebeid, et al., 2021).

6.1.4 Validation

To validate the computations energy model from Section 4.1 illustrated in this chapter via the output of the `powprofiler` tool, we demonstrated that the model and the tool function on numerous heterogeneous computing hardware and various computations. In Table 6.1, we empirically evaluate the `matrix-cpu` computation on the ODROID XU3, NVIDIA Jetson TK1, TX2, and Nano computing hardware. The `matrix-gpu` on the NVIDIA Jetson computing hardware, and the rest of the computations in this section on the NVIDIA Jetson TX2 computing hardware. A cross-platform comparison shows the energy efficiency of different computing hardware: the `matrix-cpu` computation is most efficient on NVIDIA Jetson TX2 computing hardware, followed by Nano, TK1, and ODROID XU3. Some computations not explicitly evaluated here can be potentially extended in future instances of our approach (hyphen in parenthesis in Table 6.1–future work is then in Chapter 7) conversely to others that are not supported (hyphen in Table 6.1).

We then evaluate the `powprofiler` tool through comparison with an external measuring device, i.e., a multimeter connected to the power source of the NVIDIA Jetson TX2 hardware. We observe a close co-relation between external and internal power measures. The error is less than 3% over one minute, with the external exceeding the internal measuring device, possibly due to the energy impact of the carrier board (Seewald, Schultz, Ebeid, et al., 2021). Indeed the NVIDIA Jetson TX2 computing hardware we use is mounted on the Jetson Developer Kit board in Figure 4.2. We further observe that the internal measurements of the overall power include the energy impact of the tool itself, and we thus conclude that `powprofiler` has a marginal effect

on power (Seewald, Schultz, Ebeid, et al., 2021). We later saw such a marginal effect with other auxiliary libraries, such as the Robot Operating System (ROS) middleware (Zamanakos et al., 2020).

We further validate our model against a more fine-grained model in the literature Nikov et al., 2015; Nunez-Yanez and Lore, 2013, using the ODROID XU3 computing hardware with the `matrix-cpu` computation. We described this further validation step in our early work (Seewald, Schultz, Ebeid, et al., 2021). The fine-grained model requires some apriori training, which we performed by evaluating the computation configuration with parameters $c_{i,1}$ 512, and $c_{i,2}$ 30 (meaning the thirtieth power of square matrix of size 512). The model returns an expected energy value, which we compared by subsequently running the computation, obtaining an error of 3.42% (Seewald, Schultz, Ebeid, et al., 2021). Similarly, we used the `powprofiler` tool by varying $c_{i,2}$ with a step that excludes the configuration $c_{i,2}$ 30. We then obtained an expected energy value for configuration 30, which we again evaluated against running the configuration subsequently, obtaining an error of 2.25% (Seewald, Schultz, Ebeid, et al., 2021), justifying ours against another model in the literature.

6.2 Motion Energy Modeling

In this section, we derive motions energy models from Section 4.3 of an aerial robot flying the agricultural scenario doing static coverage path planning (CPP). We use the computations energy and battery models from Sections 4.1–4.2 to integrate the static CPP with computing hardware and the effect of the battery. The coverage is static and decided offline, with no online replanning when unexpected events occur (e.g., sudden battery drops). We will then replan such coverage in Section 6.3 along with the scheduling of the computations. In detail, here we deploy two case studies of aerial robots flying the Zamboni-like motion: in Section 6.2.1, we derive an energy model using the expression in Equation (4.9), relying on the periodicity of the energy signal; we use NVIDIA Jetson TX2 as computing hardware (Seewald, Garcia de Marina, Midtiby, et al., 2020). In Section 6.2.2, we derive a differential energy model based on expressions in Equation (4.10) and Lemma 4.3.1; we use NVIDIA Jetson Nano and ROS middleware. The first case study uses a static model where we cannot predict, e.g., future energy consumption with varying schedules or flight time with varying coverage; whereas the second allows such operation. Indeed we will use a similar approach later in Section 6.3 for energy-aware coverage planning and scheduling.

6.2.1 Periodic modeling case study

The first case study is the aerial robot flying the agricultural scenario and doing static CPP, which we studied in our previous work (Seewald, Garcia de Marina, Midtiby, et al., 2020).

Experimental setup

The aerial robot is the Opterra fixed-wing that we first presented in Figure 1.1 in Chapter 1. It uses the Apogee v1.00 microcontroller with the popular Paparazzi flight controller (Paparazzi, 2016)

and has a 3.2 amperes per hour battery. The experimental setup then consists of the NVIDIA Jetson TX2 computing hardware evaluated separately of the aerial robot with two computations. The `darknet-gpu` computation that detects objects with the YOLO (Redmon, Divvala, et al., 2016) deep neural network library (which we encountered in Section 6.1.1), varying a parameter $c_{i,1}$ relative to the delay in two consecutive detections and thus the FPS rate. The `blowfish` computation then encrypts the data with a symmetric variable key algorithm named "Blowfish" (Schneier, 1993), varying a parameter $c_{i,2}$, the key-size. To model the motion energy, we analyzed the flight logs from the Paparazzi flight controller of the aerial robot flying the agricultural scenario using a static Zamboni-like motion in Section 2.6.1 implemented in the Paparazzi flight controller. We then derive the constants a, b, ω in Equation (4.9) from similar flights on the same day with similar atmospheric conditions (i.e., typical wind and temperature). We rely on the energy evolution periodicity; the data in Section 4.3 (concretely in Figures 4.7–4.8) refer to this case study and show a periodic energy evolution. We then model the energy with three frequencies, including the base frequency.

We assume static dependency on motion energy, with only computation energy varying with different computations (we will see in the next section how we merge both). Here again, the `darknet-gpu` varies in the same range as in Section 6.1.1, whereas `blowfish` varies between thirty-two and 448 bits, enclosing the computations constraint $\mathcal{S}_{i,2}$. For the latter, we used the OpenSSL (Viega et al., 2002) command-line tool for a heavy data file of approximately 150 megabytes in an iterated encryption manner. The two computations were in this use case analyzed separately. An approach we discussed in our previous work (Seewald, Schultz, Ebeid, et al., 2021), where per-component energy is modeled in a dataflow computational network, decreasing the modeling effort (Seewald, Garcia de Marina, Midtiby, et al., 2020).

Motion energy evaluation

The motion energy models are in Figure 6.5 and some corresponding paths in Figures 6.5a–6.5b. In this use case, we saw marked variability in energy signals for take-off, cruise, and landing phases per flight. To distinguish between the phases, we analyzed the motor torque, altitude, and throttle (Seewald, Garcia de Marina, Midtiby, et al., 2020). The Opterra fixed-wing aerial robot gains altitude during take-off before starting to fly the Zamboni-like motion for coverage. The modeled energy signal is in Figure 6.5c. It is evaluated from some test flights (the gray area in the figure), whereas MATLAB (R) aids the resulting regression (black line in the figure). Such a regressional analysis depicts little variability at the beginning of the take-off, likely justified by different controls necessary by various atmospheric conditions, and very little to no variability in the remaining. This latter part of the trajectory is where the flight controller guides the aerial robot on the Zamboni-like motion. Figure 6.9a is the modeled energy signal for the cruise phase. There is little variability between the test flights; the flight controller takes charge of the guidance. Finally, there is considerable variability in the landing phase energy signal illustrated in Figure 6.5e. Initially, the aerial robot flies in small circles (see the path in Figure 6.5b), lowering the altitude while descending to the ground under human control (Seewald, Garcia de Marina, Midtiby, et al.,

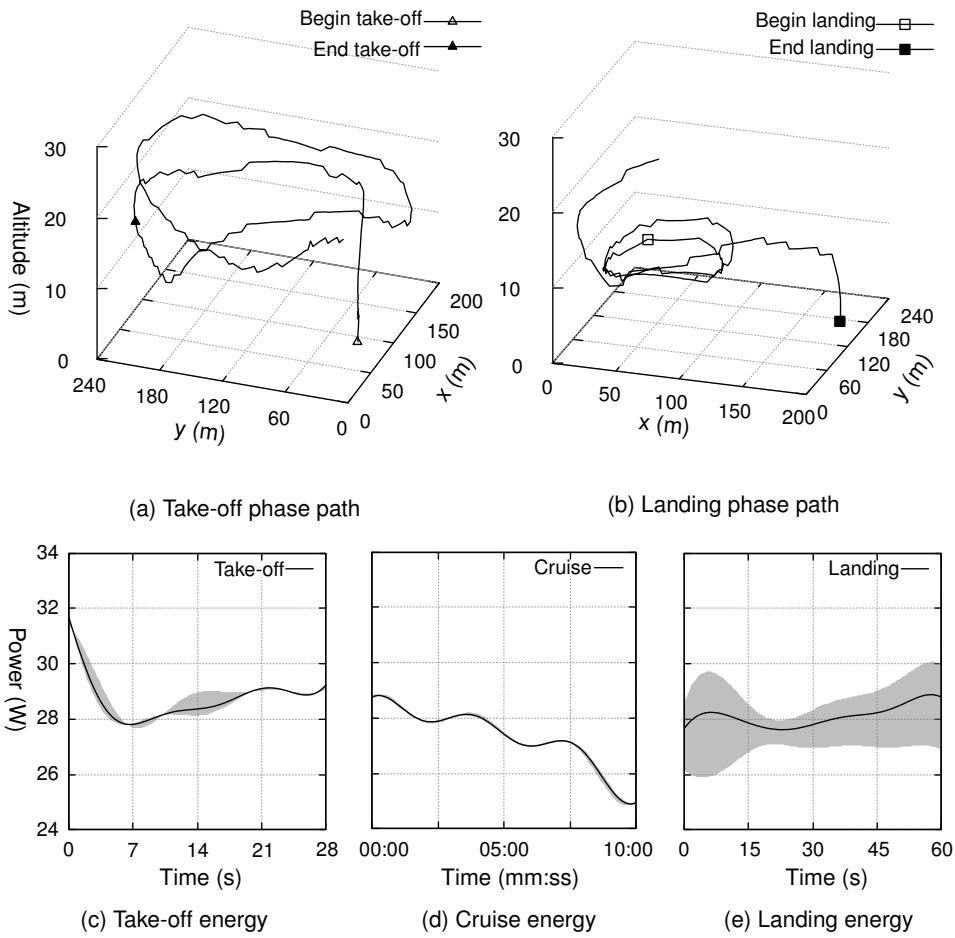
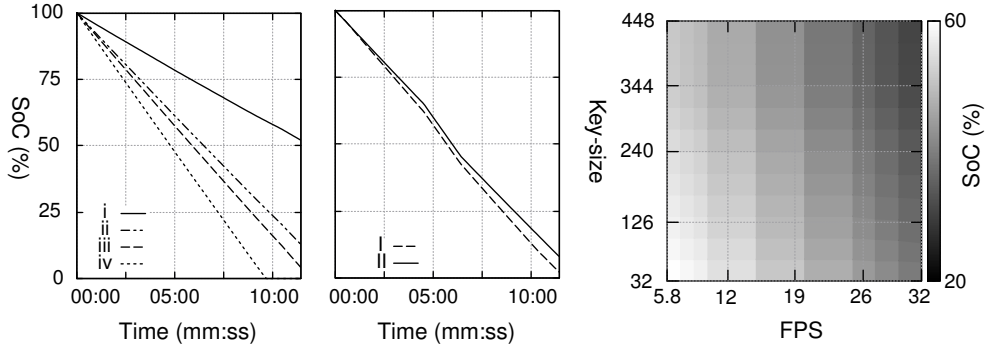


Figure 6.5. Paths and modeled energy evolutions in time for different flight phases as they appeared in our early study (Seewald, Garcia de Marina, Midtiby, et al., 2020).

2020). The phase depends on different conditions, including landing site geologic conformations, sudden wind gusts, and others, presenting thus high energy variability. We measured an average of 28 and 60 seconds for take-off and landing, whereas the cruise depends on the polygon to cover size (containing the agricultural field).

Computations energy evaluation

The computation energy model in terms of battery SoC for `darknet-gpu` and `blowfish` varying parameters $c_{i,1}$, $c_{i,2}$ is in Figure 6.6c. The figure then shows the energy impact: the higher the key size and FPS rate, the larger the impact on the battery. In Figures 6.6a–6.6b, we show the remaining battery after flying with different schedules. Particularly, Figure 6.6a illustrates various static schedules: i indicates the impact on the battery of merely flying the regressions from



(a) Static schedule (b) SoC-aware schedule (c) SoC in the function of parameters

Figure 6.6. The effect of different schedules on the battery SoC. The figure appeared in our early study (Seewald, Garcia de Marina, Midtiby, et al., 2020).

Figures 6.5c–6.5e, i.e., the energy impact of the motion on the battery; ii a static schedule of flying the configuration 5.8 FPS, and 32 bits (relative to $c_{i,1}$, $c_{i,2}$, $\forall i \in [l]$ with a fixed number of stages l); iii also a static schedule with 10 FPS and 240 bits; and finally iii with 32 FPS and 448 bits.

Finally, Figure 6.6b shows dynamic schedules. I has a configuration of 5.8 FPS and 32 bits at take-off and landing, of 32 FPS and 448 bits for the duration of two minutes in the middle of the cruise, and of 5.8 FPS and 32 bits otherwise, resulting in 2.05% remaining battery SoC at the end of the flight. II has the same configuration as I only for the two minutes in the middle of the cruise, and 5.8 FPS and 32 bits otherwise, resulting in 7.86% remaining SoC (Seewald, Garcia de Marina, Midtiby, et al., 2020). In both cases, we show that the dynamic scheduling allows completing the flight while draining the battery optimally w.r.t. the current battery SoC.

—

6.2.2 Differential modeling case study

This section details another case-study of the same aerial robot proposed in the previous section: the Opterra fixed-wing aerial robot for CPP in an agricultural scenario equipped with the Paparazzi flight controller. The computing hardware is this time the NVIDIA Jetson Nano, implementing the computations with the ROS middleware. In this use-case we evaluated some initial experiments of the differential model in Equation (4.10) in Section 4.3.1, opposed to the model in Equation (4.9). Here we did not consider the energy-aware coverage planning along various power-saving schedules (empirically demonstrated in the next section), but rather evaluated initial feasibility of the periodic differential model.

The computing hardware implements three computations. The `ssd-mobilenet` computation implements the SSD-MobileNet V2 convolutional neural network (CNN) (Sandler et al., 2018), whereas the `pednet` computation implements the PedNet fully convolutional network (FCN) (Ullah et al., 2018). The third computation is termed `sender`, as the name suggests, it sends the eventual detections on ground using the technical standard for wireless communica-

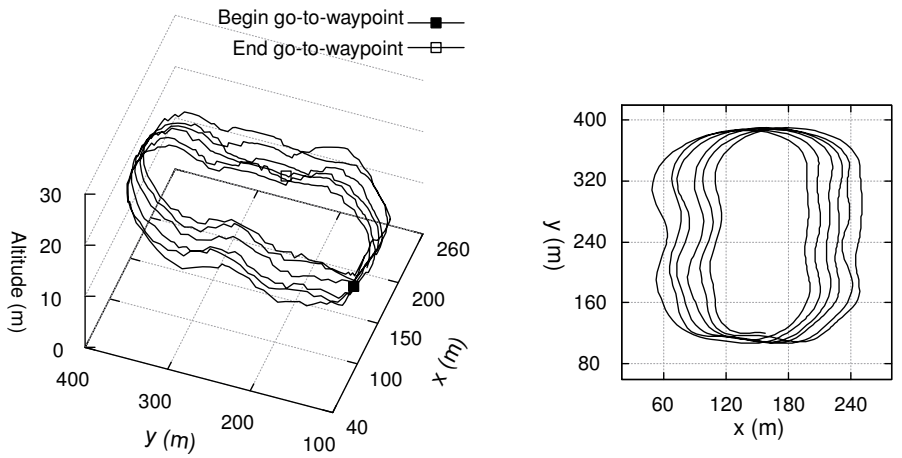


Figure 6.7. .

tion IEEE 802.11 (Crow et al., 1997). Alike the previous section, the computations and motion energy models are combined without needing to test the two together. The coverage path is to be seen in Figure 6.7; the aerial robot flies static CPP with the Zamboni-like motion in Section 2.6.1, implemented in the Paparazzi flight controller, in the same condition from the previous section. Indeed such a flight plan can be seen in Figure 6.7a, whereas in Figure 6.7b is the path from above. There is slight deviation mostly on the x-axis, which we attribute to the atmospheric conditions. Initially, we thought to count the period approximately at one fourth of the one we proposed in Section ?? (being and end of go-to-waypoint), supposing this would be optimal to model the periodicity of the energy signal, an assumption that we later correct to flying $\varphi_i, \varphi_{i+1}, \varphi_{i+2}, \varphi_{i+3}\ \forall i \in \lceil l/4 \rceil$ with l a given number of stages in the next section.

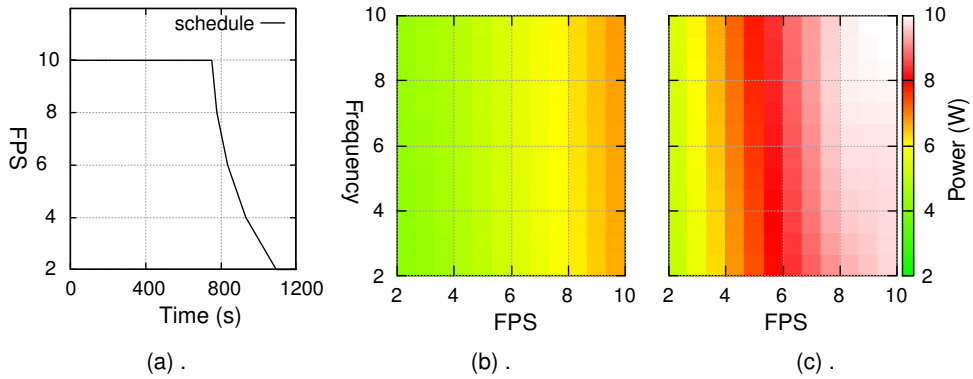


Figure 6.8. .

For the computations `ssd-mobilenet` and `pednet` the parameter $c_{i,1}$ allows to alter the rate hazard detection rate, from two to ten frames per second enclosed in the constraint set $\mathcal{S}_{i,1}$. These numbers are evaluated empirically to match our detection criteria opposed to, e.g., the computation `darknet-gpu` where we analyzed all the possible detection rates on the NVIDIA Jetson TX2 computing hardware. The computation `sender` can be then altered with the parameter $c_{i,2}$ indicating the rate at which the detections are sent to the ground, also from two to ten seconds (again enclosing the constraint set $\mathcal{S}_{i,2}$). All the computations are wrapped in ROS nodes, meaning that $c_{i,1}$ and $c_{i,2}$ can be changed or analyzed by subscribing to appropriate ROS topics. The predictive layers in temrs of battery SoC are in Figure 6.8c for the `ssd-mobilenet` computation, and in Figure 6.8b for the `pednet` computation. We use the colors here to underline the energy impact of some configurations, otherwise not visible in grayscale. The overall

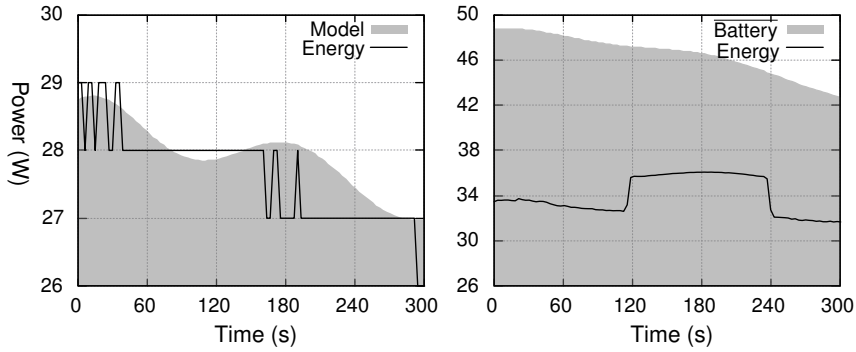


Figure 6.9. .

energy assessment for what concerns the motion is then in Figure 6.9a, and for the motion and computations energies in Figure 6.9b. For the latter, our early model consisted for an expression similar to Equation (4.10), but for the components A_j of matrix A

$$A_j := \begin{bmatrix} 0 & 1 \\ -j^2\omega^2 & 0 \end{bmatrix}, \tag{6.1}$$

where ω is the same from Equation (4.9). We note that the expression derived from using A_j in Equation (6.1) is equivalent to Equation (4.13) for periodic modeling purposes (the equivalence comes from the proof of Lemma 4.3.1, where we evaluate the determinant of A_j in Equation (4.32) and thus multiply the first row and second column with second row and first column with the negative unit, i.e., $j^2\omega^2 = j\omega j\omega$). In the model, we then used the coefficients of the fourier series as an initial guess for the model $\mathbf{q}(t_0)$ at a given time instant t_0. We derived in the coefficient via the analysis from Section 6.2.1. For modeling purposes, we limited the actual flying time of the cruise phase in Figure 6.7 to five minutes. The model output from the initial guess is then the gray area, whereas the energy data are the black solid line.

We used the computations energy model further along with the motion energy model to evaluate the maximum allowed configuration as a function of time against the overall energy

budget in Figure 6.8a and the battery SoC. The figure shows how different configurations range at different times to complete the flight using the PedNet FCN. We have further observed the cost of ROS bag recording; it is approximately 0.2 watts. The overall energy assessment for the flight is then in Figure 6.9b, coupling the motion and computations energies–the black line in the figure. The output constraint in Definition 5.3.1 is where how much energy can be drown in the function of time; the gray area in the figure. The schedule consists of $c_{i,1}$, $c_{i,2}$: *(a)* two FPS and ten hertz for the first two minutes (eventual cached images from the previous cruise are sent to the ground, as the frequency is greater than the FPS rate), *(b)* ten FPS and two hertz from the second minute to third, *(c)* ten FPS and eight hertz from the third minute to fourth, *(d)* and two FPS and ten hertz from the fourth minute to fifth.

6.3 Coverage Planning and Scheduling

In this section we provide the results for energy-aware coverage planning and scheduling proving experimentally our approach. We extensively use the construct that we derived in the previous chapter, including the energy models in Chapter 4, the guidance, coverage, and replanning from Chapter 5, thus solving both the Problems 2.5.2–2.5.1 in Chapter 2. In particular, in Section 6.3.1 we provide numerical simulations derived with MATLAB (R) , in Section ?? we integrate such simulations in the popular Paparazzi flight controller. In both the section we derive a coverage for a fixed-wing aerial robot with the Zamboni-like motion we introduced in Section 2.6.1, and finally, in Section 6.3.2, we integrate the previous results to the case of some obstacles in Section 6.3.3.

6.3.1 Numerical simulations

In this section, we discuss our experimental setup, the implementation strategy, and the results of the numerical simulation implemented in MATLAB (R) to match some realistic condition.

Experimental setup

The algorithm that we presented in this paper is motivated by a periodic behavior of empirical data on energy consumption (see the subfigures of Figure ??). We collected these data flying Opterra, a fixed-wing UAV that we adapted for an agricultural scenario. The UAV was flying in standard atmospheric conditions like the path 5.I in Figure ??, or the first "non-adapted" part of Figure ??. We later extended the UAV to carry a companion computer, NVIDIA Jetson Nano NVIDIA, 2020, running ROS. The companion computer has two ROS nodes; one detects hazards using PedNet, a Fully Convolutional Neural Network Ullah et al., 2018, and the other communicates with a ground station.

We implemented a realistic simulator to simulate the empirical data of a given plan and atmospheric conditions. We show the effects of different conditions in Figure ??. Both paths 5.I and 5.II are flying the same plan with different initial parameters values. Path 5.I is flying with a constant wind speed of 5 meters per second, wind direction of 0 degrees, and an initial path parameter $c_{1,1}$ value of 0. Path 5.II is flying under the same conditions but a wind direction of 90

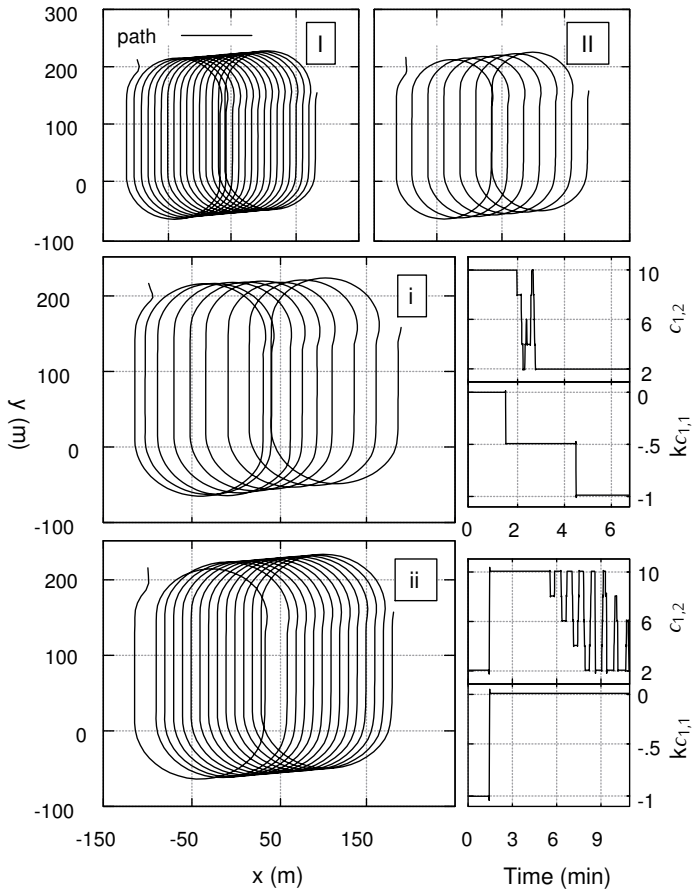


Figure 6.10. Path simulations with variations of wind speed and direction. In 5.I and 5.II the path is static. It is dynamically replanned with the algorithm in 5.i and 5.ii. The algorithm adapts path parameter radius of the circle $c_{1,1}$ and computation parameter fps rate $c_{1,2}$.

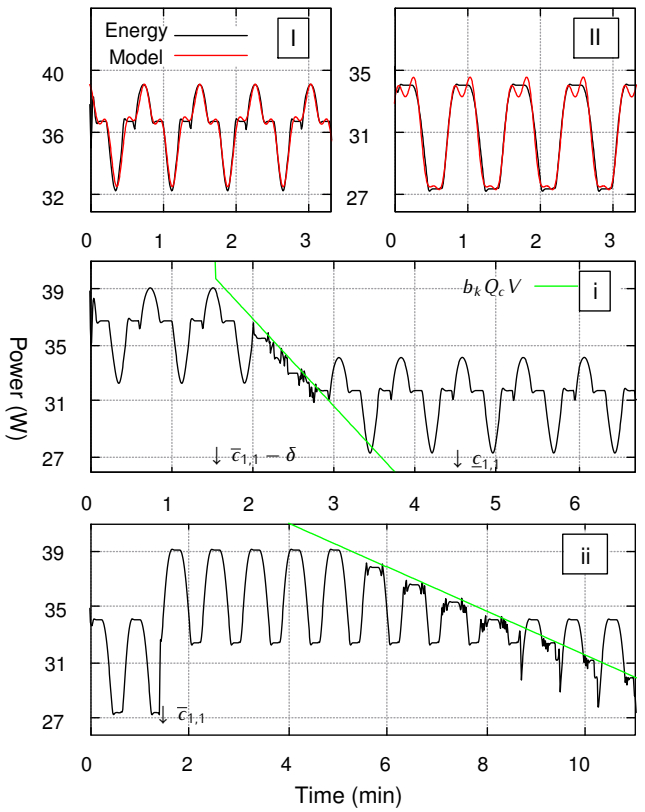


Figure 6.11. The energy models of the paths from Figure ?? for 200 seconds against the simulated data (6.I and 6.II). Below are the energy evolutions from the algorithm (6.i and 6.ii). It replans the path when the final time and battery time do not match, and the computation when the battery is discharged.

degrees and the initial path parameter value of -1000. The energy evolutions of these two paths are the top two subfigures 6.I and 6.II of Figure ??. The simulated energy data are in black, the model evolution in red.

Algorithm evaluation

We showcase the energy model from Section ?? in Figure ??. Left figures illustrate an initial slice of the model and period estimation (Definition 2.4.3). We used order r equal to 3. We motivate this choice again with Figure ??, where the power spectrum subfigure shows that 3 frequencies are adequate. On the right of Figure ??, we show the states $\alpha_0, \ldots, \beta_3$ in time, concluding that approximately 2 periods are sufficient to obtain a consistent state estimate. With non-periodic signals, we observed that the estimator estimates primarily the first state α_0 and it neglects the others. It hence approximates the non-periodicity with a linear model.

Replanning strategy

The practical implementation is based on observations of different variations of paths and computations. A variation of path alters the overall flying time, which we reflect in the factors $\nu_{1,1}, \tau_{1,1}$ from Equation (4.48). We compare the remaining flight time with the time needed to completely deplete the battery from Equation (5.8). We reduce or increase the parameter to optimize the battery time. The path parameter $c_{1,1}$ is equal for all the stages and it changes the radius of the first circle in the current period and therefore shifts the other paths accordingly (the change is illustrated in Figure ??). It results in a shorter or longer distance between the survey lines and in an increment or reduction of the flying time respectively. The path constraint set is set to $\underline{c}_{1,1} = -1000$ and $\overline{c}_{1,1} = 0$ equal for all the stages.

A variation of computations affects directly the power. We thus select the highest computation which satisfies the constraints from Definition 5.3.1 in the line ?? of the algorithm. We observed a low effect on the power of the communication ROS node. Nevertheless, the detection node varies between 5 and 10 watts for the lowest and highest fps. We implemented fps rate parameter $c_{1,2}$ with factors $\nu_{1,2}, \tau_{1,2}$ mapping $c_{1,2}$ to the data from `powprofiler`. The computation constraint set is set to $\underline{c}_{1,2} = 2$ and $\overline{c}_{1,2} = 10$ equal for all the stages.

Coverage planning and scheduling

We have tested the validity of the algorithm showing the dynamic adaptation in the subfigures 5.i and 5.ii of Figure ?? path-wise, and 6.i and 6.ii of Figure ?? energy-wise. For the first path (Subfigure 5.I) the plan starts at the highest configuration of parameters. We simulated two unexpected battery drops at approximately 1 minute and a half and 4 minutes and a half. The algorithm optimizes the path in the proximity of the drops to ensure that the flight is completed. Moreover, it maximizes the parameter $c_{1,2}$ when the battery is discharging (green line) respecting the output constraint (Definition 5.3.1). We simulated the opposite scenario for the second path (Subfigure 5.II). The plan starts at the lowest configuration of parameters and the battery behaves

linearly. We note that the path parameter is increased as soon as the algorithm estimated enough data (two periods T) and the computation parameter is optimized for the battery discharge rate. For both cases, we used reductions δ of 500 and 2 for $c_{1,1}$ and $c_{1,2}$ respectively, and the horizon N equal to 6 seconds.

6.3.2 Paparazzi flight controller

6.3.3 Coverage with obstacles

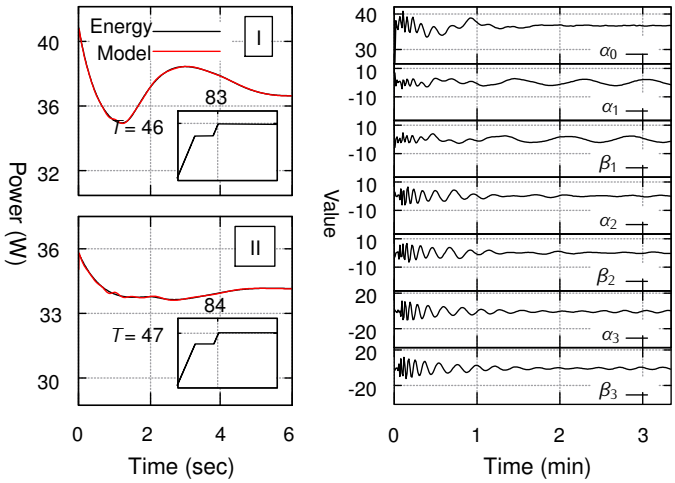


Figure 6.12. Energy estimation for the first 6 seconds on the left side, the evolution of the state $\mathbf{q}$ on the right.

Chapter 7

Summary and Future Directions

"The energy budget for sensing and computing is commensurate with that of actuation – such as is typically the case for planetary rover missions."

— Ondrúška et al., 2015

Appendix A

Optimal Control and State Estimation

This chapter provides essential theoretical background on optimal control theory necessary to derive an optimal configuration of the path and computations of the mobile robot. It solves the problem posed in Section 2 and illustrate an algorithm that generates the optimal configuration dynamically. The algorithm relies on a modern optimal control technique known as model predictive control (MPC), where the optimal control trajectory is evaluated on a receding horizon for each optimization step (Rawlings et al., 2017).

Optimal control deals with finding optimal ways to control a dynamic system (Sethi, 2019). It determines the control signal–the evolution in time of the decision variables–such that the model satisfies the dynamics and simultaneously optimizes a performance index (D. E. Kirk, 2004).

Many optimization problems originating in fields such as robotics, economics, and aeronautics can be formulated as optimal control problems (OCPs) (Von Stryk and Bulirsch, 1992). Optimization is often called mathematical programming (Nocedal and S. Wright, 2006) a term that means finding ways to solve the optimization problem. One can often find programming in this context in terms such as linear program (LP), quadratic program (QP), and nonlinear program (NLP). NLPs is the class of optimization problems that we use to derive the optimal configuration. OCPs can be seen as optimization problems with the added difficulty of continuous dynamics. The latter is to be integrated over the optimization horizon using numerical simulation. In the algorithm, we formulate the dynamic planning problem as an OCP that we solve with a numerical method: we transform the OCP in an NLP using numerical simulation and solve the NLP using numerical optimization, as proposed in (Rawlings et al., 2017). The process is illustrated in Fig. A.1.

A typical performance measure for an OCP is built such that the system: reaches a target set $\mathcal{Q}_f$ in minor time, reaches a given final state $\mathbf{q}_f$ with minimum deviation, maintains the state evolution as close as possible to a given desired evolution, or reaches the target set with the minimum control expenditure effort (D. E. Kirk, 2004). In energy planning, it is desired to focus on the latter performance measure.

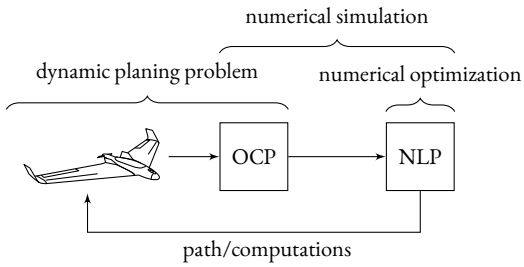


Figure A.1. Summary of the optimal control approach. The problem is formulated as an OCP, into finite-dimensional discrete NLP using numerical simulation. NLP is solved using numerical optimization and the optimal configuration for a given time horizon is returned to the aerial robot. The following horizon is evaluated again in a technique known as MPC.

The outline of the chapter is as follows. After a brief history of optimal control, we introduce formally the OCP subject to continuous dynamics. We then illustrate numerical simulation approaches to convert the infinite-dimensional continuous dynamics into finite-dimensional discrete dynamics. We formulate later in the chapter the dynamic planning problem for the optimal configuration of the path and computations with proper constraints. Finally, we illustrate MPC to solve OCP on a receding horizon. We propose an algorithm to solve such OCP using a numerical method on the horizon along with the analysis of its practical feasibility.

The chapter builds on the rest of the work as follows. In the OCP formulation, we use the estimated state from Chapter A.3 of a perfect model in Chapter 4 to solve the planning problem in Section 2 and guide the aerial robot with the obtained optimal configuration from this chapter with the technique in Chapter 5.

A.1 A Brief History of Optimal Control

Optimal control originates from the calculus of variations (Sethi, 2019), based on the work of Euler and Lagrange. Calculus of variations solves the problem of determining the arguments of an integral, such that its value is maximum (or minimum). The equivalent problem in calculus is to determine the argument of a function where the function is maximum (or minimum). The work by Euler and Lagrange was later extended by Legendre, Hamilton, and Weierstrass (Paulen and Fikar, 2016). It has gained a renewed interest in the mid-twentieth century, as modern calculators offered practical ways of solving some OCPs for nonlinear and time-varying systems that were earlier impracticable (Bryson and Ho, 1975).

The conversion of the calculus of variation problems in OCPs requires the addition of the control variable to the dynamics (Sethi, 2019).

There are numerous methods to analytically and numerically solve these continuous time OCPs, although analytical solution is often impracticable except for very limited state dimensions (Rawlings et al., 2017). In the early day of optimal control, some analytical solutions were

proposed with dynamic programming (Bellman, 1957), and with maximum (or minimum) principle (Pontryagin et al., 1962).

In computer science dynamic programming is fundamental to compute optimal solutions, yet it's original form was developed to solve optimal control problems (LaValle, 2006). Dynamic programming in optimal control theory is based on a partial differential equation of the performance index named Hamilton-Jacobi-Bellman (HJB) equation, which is solved either analytically for small dimensional state space problems, or numerically (Rawlings et al., 2017). Dynamic programming can be shown to be equivalent to the principle (Paulen and Fikar, 2016). The principle is related to HJB equation in that it provides optimality conditions an optimal trajectory must satisfy (LaValle, 2006). HJB offer sufficient conditions for optimality while the principle necessary; yet it is useful to find suitable candidates for optimality (LaValle, 2006).

All the numerical approaches discretize infinite-dimensional problems at a certain point (Rawlings et al., 2017).

A first class of these approaches solves the optimality conditions in continuous time using first-order necessary conditions of optimality from the principle (Böhme and Frank, 2017). This is done by algebraic manipulation using an expression that is similar to the HJB equation, and results in a boundary-value problem (BVP) (Rawlings et al., 2017). The class is commonly referred to as the indirect methods. The BVP is solved by discretization at the very end (Rawlings et al., 2017) and/or gradient-based resolution (Paulen and Fikar, 2016).

On the contrary, modern optimal control often first discretize an the control and state variables in the OCP to a finite dimensional optimization problem (usually NLP), which is then solved with numerical optimization (using gradient-based techniques). This other class of numerical approaches is referred to as direct methods. Some direct methods are single and multiple shooting and collocation methods. We employ direct methods in this chapter.

Modern OCPs are often solved on a finite and receding horizon using an approximation of the true dynamics using MPC techniques. It is a more systematic technique which allow to control a model by re-optimizing the OCP repeatedly (Paulen and Fikar, 2016; Poe and Mokhatab, 2017). It takes into account external interferences by re-estimating the model's state (with techniques that we introduced in Chapter A.3). MPC is extensively treated in modern optimal control literature (Camacho and Alba, 2007; Kwon and Han, 2005; Rawlings et al., 2017; Rossiter, 2004; L. Wang, 2009).

A.2 Optimization Problems with Dynamics

A.2.1 Continuous systems: unconstrained case

Given a state variable $\mathbf{q}$ composed of m states and a control variable $\mathbf{u}$ composed of n controls, the state variable dynamics at a given time instant t can be described by a differential model

$$\dot{\mathbf{q}}(t) = f(\mathbf{q}(t), \mathbf{u}(t), t), \quad \mathbf{q}(t_0) = \mathbf{q}_0 \text{ given}, \quad \forall t \in [t_0, T], \tag{1.1}$$

where $t_0 \in \mathbb{R}_{\geq 0}$ is a given initial time instant, and $\mathbf{q}_0 \in \mathbb{R}^m$ a given initial state guess. The latest can be derived empirically from a previous execution or using some initial sensor data.

$f : \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^m$ maps the current state, control and time to the next state. The notations for $\dot{\mathbf{q}}(t) := d\mathbf{q}(t)/dt$, $\mathbf{q}$, and $\mathbf{u}$ are the same from Chapter 4. The function f is assumed to be continuously differentiable. Physically, Equation (1.1) specifies the instantaneous change in state variable of a perfect model with no disturbances.

If the control trajectory $\mathbf{u}(t_0), \mathbf{u}(t_1), \ldots, \mathbf{u}(T - \Delta t)$ is known for a given time horizon $t_0 \leq t \leq T$, the model in Equation (1.1) can be derived to obtain the state trajectory $\mathbf{q}(t_0), \mathbf{q}(t_1), \ldots, \mathbf{q}(T)$, where Δt is the instantaneous change in time. The last state at the final time instant T is derived from the last control at the time instant $T - \Delta t$. The state trajectory has indeed one item more than the control trajectory.

Optimal control finds a control trajectory which maximizes (or minimizes) a performance index

$$L = l_f(\mathbf{q}(T), T) + \int_{t_0}^{T} l(\mathbf{q}(t), \mathbf{u}(t), t)\, dt, \tag{1.2}$$

where l, l_f are given instantaneous and final cost functions. The instantaneous cost function maps state, controls, and time to a value that quantifies the cost of a given instant $l : \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$. The final cost function maps the state and time to a value which quantifies the cost of the final instant $l_f : \mathbb{R}^m \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$. The performance index $L \in \mathbb{R}$ is then the sum of all the contribution on the time horizon.

Performance index found in (Bryson and Ho, 1975) is also found in literature as cost function in (Simon, 2006; Stengel, 1994), objective function in (S. S. Rao, 2019; Rawlings et al., 2017; Sethi, 2019), or performance measure (D. E. Kirk, 2004).

The control variable is usually constrained

$$\mathbf{u}(t) \in \mathcal{U}(t), \quad \forall t \in [t_0, T], \tag{1.3}$$

where $\mathcal{U}(t) \subseteq \mathbb{R}^m$ is the control constraint set. It delimits all the feasible values of the control for the horizon. There can be different control constraint sets for different instants.

The Equations (1.1–1.3) forms unconstrained OCPs. These problems are formalized

$$\begin{aligned} &\max_{\mathbf{q}(t),\mathbf{u}(t)} l_f(\mathbf{q}(T), T) + \int_{t_0}^{T} l(\mathbf{q}(t), \mathbf{u}(t), t)\, dt, \\ &\text{s.t. } \dot{\mathbf{q}}(t) = f(\mathbf{q}(t), \mathbf{u}(t), t), \\ &\quad \mathbf{u}(t) \in \mathcal{U}(t), \quad \mathbf{q}(t) \in \mathcal{Q}(t), \\ &\quad \mathbf{q}(t_0) = \mathbf{q}_0 \text{ given.} \end{aligned} \tag{1.4}$$

The evolution of the model is used to derive an optimal control trajectory $\mathbf{u}(t)$ from an initial guess of the state $\mathbf{q}_0$ and the horizon. This initial simplistic controller does not represent a realistic scenario. The controller implies that the horizon is known. However, it is often the case that only the initial time step of the horizon $[t_0, T]$ is known. In the model from Chapter 4 it is indeed unknown apriori when the aerial robot plan terminates. Moreover the controller does not include any constraint on the state $\mathbf{q}$, although mobile robots are often bounded by strict battery requirements. Lastly, the optimal control generated with such controller is static given the

initial state and the horizon. It is unrealistic to assume that the state of the aerial robot travelling the optimal control $\mathbf{u}$ does not change for instants $t_0 + \Delta t, t_0 + 2\Delta t, \ldots, T$.

All these initial assumption (known final time step, absence of state constraints, static optimal control law) will be eased in the remaining of the chapter.

A.2.2 Continuous systems: constrained case

A.2.3 Perturbed systems

A.2.4 Multistage systems

A.3 State Estimation

A.3.1 The curve fitting problem

A.3.2 Period estimation

A.3.3 Discrete time Kalman filter

As the environment uncertainty and measurement error evolve in a normal distribution, we use a Kalman filter (Simon, 2006; Stengel, 1994) for the purpose of state estimation.

The prediction is done using

$$\hat{\mathbf{q}}_{k+1}^{-} = A\hat{\mathbf{q}}_k + B\mathbf{u}_k, \tag{1.5a}$$

$$P_{k+1}^{-} = AP_kA' + Q, \tag{1.5b}$$

where $\hat{\mathbf{q}}_k^{-}, \hat{\mathbf{q}}_k \in \mathbb{R}^j$ depicts the estimate of the state before and after measurement (or simply estimate), and $P_k, P_k^{-} \in \mathbb{R}^{j\times j}$ the error covariance matrix (i.e., the variance of the estimate).

The estimation of the state and the update of the predicted output is done using

$$K_k = (CP_{k+1}^{-}C' + R)^{-1}(P_{k+1}^{-}C'), \tag{1.6a}$$

$$\hat{\mathbf{q}}_{k+1} = \hat{\mathbf{q}}_{k+1}^{-} + K_k(y_k^s + y_k^c - C\hat{\mathbf{q}}_{k+1}^{-}), \tag{1.6b}$$

$$P_{k+1} = (I - K_kC)P_{k+1}^{-}, \tag{1.6c}$$

$$\hat{y}_k = C\hat{\mathbf{q}}_{k+1}, \tag{1.6d}$$

where $K_k \in \mathbb{R}^j$ is the gain of the Kalman filter, and I the identity matrix. y_k^s, y_k^c are the instantaneous energy readings: $y_k^s \in \mathbb{R}_{\geq 0}$ the robot sensor, i.e., the energy due to the trajectory, and y_k^c the energy of a given software configuration described in …. The noise covariance matrices $Q \in \mathbb{R}^{j\times j}, R \in \mathbb{R}$ indicates the uncertainty and measurement error covariance respectively, and $\hat{y}_k \in \mathbb{R}_{\geq 0}$ is the estimated energy.

Equations (1.5–1.6) converge to the predicted energy evolution as follows. An initial guess of the values $\hat{\mathbf{q}}_0, P_0$ is derived empirically from collected data. It is worth considering that an appropriate guess of these parameters allows the algorithm to converge to the desired energy evolution in a shorter amount of time. The tuning parameters Q, R are also derived from the

collected data, and may differ due to i.e., different sensors used to measure the instantaneous energy consumption, or different atmospheric conditions accounting for the process noise.

At time $k = 0$, the initial estimate before measurement of the state and of the error covariance matrix is updated in Equations (1.5a–1.5b) respectively. The value of $\hat{\mathbf{q}}_1^-$ is then used in Equation (1.6b) to estimate the current state along with the data from the sensor y_0 (e.g., the energy sensor of the flight controller of the fixed-wing craft), where the sensor noise covariance matrix R accounts for the amount of uncertainty in the measurement. The estimated output $\hat{y}_0$ is then obtained from Equation (1.6d). The algorithm is iterative. At time $k = 1$ the values $\hat{\mathbf{q}}_1$, P_1 computed at previous step are used to estimate the values $\hat{\mathbf{q}}_2$, P_2, and y_1.

A.3.4 Continuous time Kalman filter

A.3.5 Nonlinear filtering

A.4 Numerical Simulation and Differentiation

A.4.1 Euler method

A.4.2 Runge-Kutta methods

A.4.3 Algorithmic differentiation

A.5 Direct Optimal Control Methods

A.5.1 Direct single shooting

A.5.2 Direct multiple shooting

A.5.3 Direct collocation

A.6 Numerical Optimization

A.6.1 Convexity

A.6.2 Optimality conditions

A.6.3 First order optimality conditions

A.6.4 Second order optimality conditions

A.6.5 Sequential quadratic programming

A.6.6 Nonlinear interior point methods

A.7 Results

A.8 Summary

Appendix B

Implementation

B.1 Energy Models

To build the model in MATLAB (R) one can use the function `build_model` from Listing B.1.

```
function [model] = build_model(dat)

    m = 2*dat.r+1;

    Aj = @(dat.omega,j) [0 dat.omega*j;-dat,omega*j 0];
    A  = zeros(m);
    A(1,1) = 0;

    B = [zeros(1,dat.rho) ones(1,dat.sigma)];

    C = [1];

    for i = 1:dat.r
        A(2*i:2*i+1,2*i:2*i+1) = Aj(dat.omega,i);
        C = [C 1 0];
        B = [B;zeros(1,dat.rho+dat.sigma)];
    end

    model.A = A;
    model.B = B;
    model.C = C;

    model.est_u = @(c) dat.nu*c+dat.tau;
    model.u = @(u1,u0) u1-u0;

end
```

Listing B.1. Function `build_model` that creates the energy model.

The function builds the matrices from Equation (4.10). In particular matrix A from Equation (4.12), B from Equation (4.46), and C from Equation (4.14). The term $1/T$ is missing in `C` as we motivated in the proof of Lemma 4.3.1 in Equation (4.40). The nominal control from Equation (4.44) is `u`, and the energy estimate of a given control sequence at time instant t and the previous time instant $t - \Delta t$ from Equation (4.45) is `est_u`.

The function requires the structure `dat` illustrated in Listing B.2. The structure contains some information about the model. In particular, `r` is the order r of the model from Subsection ??, `omega` is the angular frequency ω, `rho` the number of path parameters to alter the path ρ, and `sigma` the number of computation parameters to alter the computations σ. The path parameters are defined in Equation (??), and the computation parameters are defined in Equation (??). Furthermore, `delta_T` is the sampling step Δt, and `nu` and `tau` are scaling factors (ν and τ) from Equation (4.47).

```
dat.r = 3;
dat.omega = 2*pi/period;
dat.rho = 1;
dat.sigma = 1;
dat.delta_T = .01;
dat.nu = nu;
dat.tau = tau;
```

Listing B.2. Struct `dat` used by function `build_model` to build the model.

The function `build_model` and the structure `dat` can be used along an initial guess of parameters and a time horizon to model the future energy consumption using the Euler or Runge-Kutta methods for numerical integration. The Euler method is described in Subsection A.4.1 and implemented in Listing B.3, the Runge-Kutta method in Subsection A.4.2 and Listing B.5. For the Runge-Kutta method, we use the fourth-order fixed size standard method.

```
function [q,y] = evolve_model_euler(model,dat,init)

    m = 2*dat.r+1;
    Ad = model.A*dat.delta_T+eye();

    q0 = init.q0;

    q = q0;
    y = C*q0;

    est_u0 = init.est_u0;
    est_u1 = model.est_u(init.c_chain(1));

    i = 2;

    for init.t0+dat.delta_T:dat.delta_T:init.tf
        q0 = Ad*q0+model.B*model.u(est_u1,est_u0);
        est_u0 = est_u1;
```

```
20          if i <= size(init.c_chain,2)
21              est_u1 = model.est_u(init.c_chain(i));
22              i = i+1;
23          end

25          q = [q;q0.'];
26          y = [y;C*q0];
27      end
28  end
```

Listing B.3. Euler method for numerical integration of the model.

The function `evolve_model_euler` in Listing B.3 evolve the model in Equation (4.10) from an initial time t_0 to the final time t_f. This information is passed to function using the structure `init` from Listing B.4.

```
1 init.t0 = 0;
2 init.tf = 60;
3 init.q0 = q0;
4 init.est_u0 = est_u0;
5 init.c_chain = c_chain;
```

Listing B.4. Structure `init` with numerical integrator's initializations.

In Listing B.4, the field `t0` indicates the initial time t_0 (the mathematical simulation starts at $t_0 + \Delta t$ assuming the instant t_0 corresponds to the state $\mathbf{q}(t_0) = \mathbf{q}_0$). The field `tf` indicates the final time t_f when the simulation stops. This is usually $t_0 + N$ where N is the optimization horizon. Both fields are expressed in seconds. The field `q0` indicates the initial state guess at time instant t_0, $\mathbf{q}_0$, and finally `c_chain` the sequence of controls. If we assume that the horizon is $N := t_f - t_0$ with $(t_f - t_0) \in \mathbb{R}_{>0}$, then the user is expected to provide $N - 1$ controls. Finally, the field `est_u0` contains the control energy estimate at time t_0, the value $\hat{\mathbf{u}}(t_{-1})$. If t_0 is the initial time step, then `est_u0` is set to zero.

```
1 % TODO
```

Listing B.5. Runge-Kutta fourth order method for numerical integration of the model.

Before calling structure `dat`, it is necessary to define the scaling factors $\nu_1, \nu_2, \ldots, \nu_\rho$ and $\tau_1, \tau_2, \ldots, \tau_\rho$ for the path parameters, and $\nu_{\rho+1}, \nu_{\rho+2}, \ldots, \nu_{\rho+\sigma}$ and $\tau_{\rho+1}, \tau_{\rho+2}, \ldots, \tau_{\rho+\sigma}$ for the computation parameters. For example, we suppose there is one path and one computation parameters.

```
1 nu1 = -(max_t-min_t)/min_c1;
2 tau1 = -min_c1*(min_t-max_t)/min_c1+min_t;

4 nu2 = (g_max_c2-g_min_c2)/(max_c2-min_c2);
5 tau2 = min_c2*(g_min_c2-g_max_c2)/(max_c2-min_c2)+g_min_c2;

7 nu = [nu1;nu2];
```

```
8 tau = [tau1;tau2];
```

Listing B.6. Implementation of path and computation parameters scaling factors.

Listing B.6 has to run before Listing B.2.

The scaling factors can be defined according to Equation (4.48) for the path parameter on Line 1 and according to Equation (4.49) for the computation parameter on Line 4.

Let us assume that the one path parameter c_1 doesn't change for the stages $i = \{1, 2, \ldots, l\}$. `max_t` and `min_t` are the maximum and minimum times that correspond to the time needed to hypothetically execute the path with parameter $c_1 = \overline{c}_1$ and with parameter value $c_1 = \underline{c}_1$. A guess for these values can be obtained empirically; we obtained them by running the simulation. They correspond to $\overline{t}$ and $\underline{t}$ in Equation (4.48).

Let us further assume that the one computation parameter c_2 doesn't change for the stages $i = \{1, 2, \ldots, l\}$. `max_c2` and `min_c2` are the minimum and maximum configuration of the computation parameter c_2 defined in Definition 2.3.1. `g_max_c2` and `g_min_c2` are values retrieved from `powprofiler` and they correspond to $g(\overline{c}_2)$ and $g(\underline{c}_2)$ in Equation (4.49). Function g is formally defined in Definition 4.1.2.

A possible call to the functions `build_model` and `evolve_model_euler` or `evolve_model_rk4` from Listing B.1 and Listing B.3 or Listing B.5 is illustrated in Listing B.7.

```
1 model = build_model(dat);
2 [q,y] = evolve_model_euler(model,dat,init);
```

Listing B.7. An example to build a differential model and evolve it over a horizon N.

A generic routine to plot the resulting modeled energy and the coefficients $\mathbf{q}$ in Equation (4.10) is illustrated in Listing B.8.

```
1 time = dat.t0:model.delta_T:dat.tf;

3 figure;
4 plot(time,y)
5 title('energy evolution');
6 ylabel('power (W)');
7 xlabel('time (sec)');

9 figure;
10 m = 2*dat.r+1;
11 t = tiledlayout(m,1);
12 title_str = {'alpha','beta'};
13 nexttile,plot(time,q(1,1:end))
14 title(strcat(title_str(1),0));

16 for i=2:m
17     nexttile,plot(time,q(i,1:end))
18     title(strcat(title_str(mod(i,2)+1),i-1));
19 end
```

```
21 title(t,'coefs evolution');
22 xlabel(t,'time (sec)');
23 ylabel(t,'value');
```

Listing B.8. A generic polotting routine for them modeled energy and coefficients evolution.

We describe the estimation of T, the `period` in structure `dat`, in Subsection B.2.1. We generate the $N-1$ controls in Section B.4. For benchmarking, one can define the lowest (or similarly the highest) level of parameters with Listing B.9. It has to run before defining the structure `init` in Listing B.4.

```
1 c_chain = min_c1*ones(1,N-1);
2 c_chain = [c_chain;min_c2*ones(1,N-1)];
```

Listing B.9. Implementation of the lowest possible control for benchmarking.

B.2 State Estimation

B.2.1 Estimation of the period

```
1 % iterating trajectories in the plan to get the constant n (to measure
      the period)
2 d = [];
3 p = [0; 0];
4 n = 0;

6 for traj = transpose(path) % per each trajectory

8     traj = split(traj,';');
9     traj = str2sym(traj(3));

11    x = p(1);
12    y = p(1);
13    di = double(subs(traj));
14    x = p(1)-sp(1);
15    y = p(2)-sp(2);

17    if ismember(double(subs(traj)),d)
18        break;
19    else
20        d = [d di];
21    end

23    n = n+1;
24 end

26 fprintf('n is %d\n', n);
```

Listing B.10. Estimation of the period.

B.2.2 Estimation of the state

```
minus_q1 = Ad*q0+B*est_u();

minus_P1 = Ad*P0*Ad.'+Q;

K1 = (minus_P1*C.')*(C*minus_P1*C.'+R)^-1;
q1 = minus_q1+K1*(pow(end)-C*minus_q1);
P1 = (eye(size(q0,1))+K1*C)*minus_P1;

q0 = q1;
P0 = P1;

y = [y;C*q1];
q = [q q0];
```

Listing B.11. Estimation of the state using Kalman filter.

B.3 Guidance

```
function [u_theta] = gvf_control_2D(p,dot_p,ke,kd,path,grad,hess,dir)

    if (nargin(path) > 1) % function handle has two arguments (circle)
        e = path(p(:,1),p(:,2));
        n = grad(p(:,1),p(:,2));
    else % one argument (line)
        e = path(p(:,1));
        n = grad();
    end

    H = hess();
    E = [0 -1;1 0];
    tau = dir*E*n;

    dot_pd = tau-ke*e*n; % (7)
    ddot_pd = (E-ke*e*eye(2))*H*dot_p-ke*n'*dot_p*n; % (10)
    ddot_pdhat = -E*(dot_pd*dot_pd')*E*ddot_pd/norm(dot_pd)^3; % (9)

    dot_Xid = ddot_pdhat'*E*dot_pd/norm(dot_pd); % (13)

    u_theta = dot_Xid+kd*dot_p'*E*dot_pd/(norm(dot_p)*norm(dot_pd)); %
        (16)
end
```

Listing B.12. Guidance vector field.

```
u_theta = gvf_control_2D(log_p(:,end).',pdot,ke,kd,path,grad,hess,dir);

th_delta = kp*(hd-h)-kvv*vv;
```

```
wbx = dot(w,[cos(theta),sin(theta)]);
vs = cth*(th_nominal+th_delta)+wbx;
L = cl*vs*vs;
av = 1/m*(L-W);

vv = vv+av*delta_T;
h = h+vv*delta_T;

% Horizontal unicycle model
sh = cth*(th_nominal+th_delta);
pdot = sh*[cos(theta);sin(theta)]+w;

p = p+pdot*delta_T;
theta = theta+u_theta*delta_T;
theta = wrapToPi(theta); % normalizing between -pi and pi
```

Listing B.13. Call to the guidance function and dynamics evolution.

B.4 Optimal Control Generation

```
function [c_chain q_chain] = mpc(min_c1,max_c1,min_c2,max_c2,c1,c2,N,eu,
    b0,b,qc,int_v,q0,Ad,B,C,u,est_u,k,delta_T,r)
    import casadi.* % import casadi for optimal control

    interval = k+delta_T:delta_T:k+N; % no k+1 in the formula as we add
        the
    % initial condition implicitly
    hh = length(interval);

    opti = casadi.Opti(); % define opt problem
    Q = opti.variable(2*r+1,N);
    U = opti.variable(2,N-1);
    L = opti.variable(1,N);
    log_Q = opti.variable(2*r+1,hh);
    log_b = opti.variable(1,hh);

    opti.set_initial(Q(:,1),q0);

    opti.subject_to(C*Q(:,1)-b0*qc*int_v <= 0);
    opti.subject_to(U(1,:) == c1);
    opti.subject_to(min_c2 <= U(2,:) <= max_c2);
    opti.subject_to(min_c1 <= U(1,:) <= max_c1);

    eeu = eu; % control estimate (from model)
    eeeu =  est_u(c1,c2);

    jjj = 1;
    dQ = q0; % initial state
    log_Q(:,1) = dQ;
```

```
29      for jj=1:hh-1
30          dQ = Ad*dQ+B*u(eeeu,eeu);
31          eeu = eeeu;
32          b0 = b0+delta_T*b(C*dQ); % battery dynamics

34          log_Q(:,jj+1) = dQ;
35          log_b(jj+1) = b0;

37          if mod(jj,1/delta_T) == 0 % every sum in the MPC
38              L(jjj) = C*dQ;

40              if (jjj < N)
41                  eeeu = est_u(U(1,jjj),U(2,jjj));
42              end

44              opti.subject_to(Q(:,jjj+1)-dQ == 0); % dynamics const
45              opti.subject_to(C*Q(:,jjj+1)-b0*qc*int_v <= 0); % battery
                    const

47              jjj = jjj+1;
48          end
49      end

51      opti.minimize(-sum(L.^2));
52      opti.solver('ipopt');

54      try
55          sol = opti.solve();
56          c_chain = sol.value(U); % optimal control u on N
57      catch
58          c_chain = [ones(1,N-1)*min_c1;ones(1,N-1)*min_c2]; % there is no
                control which sattisfies consts
59      end

61      q_chain = opti.debug.value(Q);
62  end
```

Listing B.14. Model predictive control.

References

1. Ablavsky, V. and Snorrason, M. (2000). “Optimal search for a moving target - A geometric approach”. In: *AIAA Guidance, Navigation, and Control Conference and Exhibit* (cit. on p. 89).

2. Abramov, A., Pauwels, K., Papon, J., Worgotter, F., and Dellen, B. (2012). “Real-time segmentation of stereo videos on a portable system with a mobile gpu”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.9, pp. 1292–1305 (cit. on p. 1).

3. Acar, E. U., Choset, H., Rizzi, A. A., Atkar, P. N., and Hull, D. (2002). “Morse Decompositions for Coverage Tasks”. In: *The International Journal of Robotics Research* 21.4, pp. 331–344 (cit. on p. 44).

4. Acevedo, J. J., Arrue, B. C., Diaz-Banez, J. M., Ventura, I., Maza, I., and Ollero, A. (2014). “One-to-one coordination algorithm for decentralized area partition in surveillance missions with a team of aerial robots”. In: *Journal of Intelligent & Robotic Systems* 74.1, pp. 269–285 (cit. on p. 4).

5. Ahmadzadeh, A., Keller, J., Pappas, G., Jadbabaie, A., and Kumar, V. (2008). “An Optimization-Based Approach to Time-Critical Cooperative Surveillance and Coverage with UAVs”. In: *Experimental Robotics: The 10th International Symposium on Experimental Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 491–500 (cit. on p. 46).

6. Aldegheri, S., Bombieri, N., Bloisi, D. D., and Farinelli, A. (2019). “Data Flow ORB-SLAM for Real-time Performance on Embedded GPU Boards”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5370–5375 (cit. on p. 56).

7. Alexey, G., Klyachin, V., Eldar, K., and Driaba, A. (2021). “Autonomous Mobile Robot with AI Based on Jetson Nano”. In: *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 1*. Cham: Springer International Publishing, pp. 190–204 (cit. on p. 56).

8. Aljanobi, A., Al-Hamed, S., and Al-Suhaibani, S. (2010). “A setup of mobile robotic unit for fruit harvesting”. In: *19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010)*. IEEE, pp. 105–108 (cit. on p. 2).

9. De-An, Z., Jidong, L., Wei, J., Ying, Z., and Yu, C. (2011). “Design and control of an apple harvesting robot”. In: *Biosystems engineering* 110.2, pp. 112–122 (cit. on p. 2).

10. Anderson, J. D. (2005). *Introduction to flight*. McGraw-Hill Higher Education (cit. on pp. 3, 5).

11. Andrew, W., Greatwood, C., and Burghardt, T. (2019). “Aerial Animal Biometrics: Individual Friesian Cattle Recovery and Visual Identification via an Autonomous UAV with Onboard Deep Inference”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 237–243 (cit. on p. 56).

12. Anguelov, D., Koller, D., Parker, E., and Thrun, S. (2004). “Detecting and modeling doors with mobile robots”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 4, pp. 3777–3784 (cit. on p. 50).

13. Araújo, J., Sujit, P., and Sousa, J. (2013). “Multiple UAV area decomposition and coverage”. In: *2013 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pp. 30–37 (cit. on pp. 25, 46, 47, 89).

14. Arkin, E., Fekete, S., and Mitchell, J. (1993). “The lawnmower problem”. In: *Proceedings of the 5th Canadian Conference on Computational Geometry*, pp. 461–466 (cit. on p. 43).

15. Arkin, E. M., Bender, M. A., Demaine, E. D., Fekete, S. P., Mitchell, J. S. B., and Sethia, S. (2001). “Optimal Covering Tours with Turn Costs”. In: *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*. USA: Society for Industrial and Applied Mathematics, pp. 138–147 (cit. on pp. 15, 45, 48).

16. — (2005). “Optimal Covering Tours with Turn Costs”. In: *SIAM Journal on Computing* 35.3, pp. 531–566 (cit. on pp. 45, 48).

17. Arkin, E. M., Fekete, S. P., and Mitchell, J. S. (2000). “Approximation algorithms for lawn mowing and milling”. In: *Computational Geometry* 17.1, pp. 25–50 (cit. on pp. 43, 84).

18. Arkin, E. M. and Hassin, R. (1994). “Approximation algorithms for the geometric covering salesman problem”. In: *Discrete Applied Mathematics* 55.3, pp. 197–218 (cit. on p. 43).

19. Artemenko, O., Dominic, O. J., Andryeyev, O., and Mitschele-Thiel, A. (2016). “Energy-Aware Trajectory Planning for the Localization of Mobile Devices Using an Unmanned Aerial Vehicle”. In: *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9 (cit. on pp. 25, 26, 49, 88).

20. Atkinson, K., Han, W., and Stewart, D. (2009). “Euler’s method”. In: *Numerical Solution of Ordinary Differential Equations*. John Wiley & Sons. Chap. 2, pp. 15–36 (cit. on p. 98).

21. Bailey, P. E., Lowenthal, D. K., Ravi, V., Rountree, B., Schulz, M., and De Supinski, B. R. (2014). “Adaptive configuration selection for power-constrained heterogeneous systems”. In: *2014 43rd International Conference on Parallel Processing*. IEEE, pp. 371–380 (cit. on pp. 35, 59).

22. Barrientos, A., Colorado, J., Cerro, J. d., Martinez, A., Rossi, C., Sanz, D., and Valente, J. (2011). “Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots”. In: *Journal of Field Robotics* 28.5, pp. 667–689 (cit. on p. 46).

23. Basilico, N. and Carpin, S. (2015). “Deploying teams of heterogeneous UAVs in cooperative two-level surveillance missions”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 610–615 (cit. on p. 4).

24. Beck, A. (2014). *Introduction to nonlinear optimization: Theory, algorithms, and applications with MATLAB*. SIAM (cit. on p. 96).

25. Bellman, R. E. (1957). *Dynamic Programming*. Princeton: Princeton University Press (cit. on p. 127).

26. Benini, L., Castelli, G., Macii, A., Macii, E., Poncino, M., and Scarsi, R. (2001). “Discrete-time battery models for system-level low-power design”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9.5, pp. 630–640 (cit. on pp. 40, 41).

27. Bhat, G., Gumussoy, S., and Ogras, U. Y. (2019). “Power and Thermal Analysis of Commercial Mobile Platforms: Experiments and Case Studies”. In: *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 144–149 (cit. on p. 57).

28. Bicego, D., Mazzetto, J., Carli, R., Farina, M., and Franchi, A. (2020). “Nonlinear model predictive control with enhanced actuator model for multi-rotor aerial vehicles with generic designs”. In: *Journal of Intelligent & Robotic Systems* 100.3, pp. 1213–1247 (cit. on p. 91).

29. Böhme, T. J. and Frank, B. (2017). “Indirect Methods for Optimal Control”. In: *Hybrid Systems, Optimal Control and Hybrid Vehicles: Theory, Methods and Applications*. Cham: Springer International Publishing, pp. 215–231 (cit. on p. 127).

30. Bouzid, Y., Bestaoui, Y., and Siguerdidjane, H. (2017). “Quadrotor-UAV optimal coverage path planning in cluttered environment with a limited onboard energy”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 979–984 (cit. on p. 49).

31. Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press (cit. on p. 80).

32. Brateman, J., Xian, C., and Lu, Y.-h. (2006). “Energy-Effcient Scheduling for Autonomous Mobile Robots”. In: *2006 IFIP International Conference on Very Large Scale Integration*, pp. 361–366 (cit. on pp. 33, 50, 51, 91).

33. Bridges, R. A., Imam, N., and Mintz, T. M. (2016). “Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods”. In: *ACM Comput. Surv.* 49.3, pp. 1–27 (cit. on p. 38).

34. Bryson, A. E. and Ho, Y.-C. (1975). *Applied optimal control: optimization, estimation and control*. Hemisphere Publishing Corporation (cit. on pp. 79, 126, 128).

35. Bürkle, A. (2009). “Collaborating miniature drones for surveillance and reconnaissance”. In: *Unmanned/Unattended Sensors and Sensor Networks VI*. Vol. 7480. International Society for Optics and Photonics, 74800H (cit. on p. 4).

36. Burri, M., Gasser, L., Käch, M., Krebs, M., Laube, S., Ledergerber, A., Meier, D., Michaud, R., Mosimann, L., Müri, L., et al. (2013). “Design and control of a spherical omnidirectional blimp”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1873–1879 (cit. on p. 6).

37. Cabreira, T. M., Brisolara, L. B., and Ferreira Jr., P. R. (2019). “Survey on Coverage Path Planning with Unmanned Aerial Vehicles”. In: *Drones* 3.1 (cit. on pp. 7, 41, 47).

38. Cabreira, T. M., Franco, C. D., Ferreira, P. R., and Buttazzo, G. C. (2018). “Energy-Aware Spiral Coverage Path Planning for UAV Photogrammetric Applications”. In: *IEEE Robotics and Automation Letters* 3.4, pp. 3662–3668 (cit. on pp. 25, 42, 49).

39. Calore, E., Schifano, S. F., and Tripiccione, R. (2015). “Energy-performance tradeoffs for HPC applications on low power processors”. In: *European Conference on Parallel Processing*. Springer, pp. 737–748 (cit. on pp. 37, 60).

40. Camacho, E. F. and Alba, C. B. (2007). *Model predictive control*. London: Springer-Verlag (cit. on pp. 91, 127).

41. Canny, J. (1988a). “Constructing roadmaps of semi-algebraic sets I: Completeness”. In: *Artificial Intelligence* 37.1, pp. 203–222 (cit. on p. 86).

42. — (1988b). *The complexity of robot motion planning*. MIT press (cit. on p. 86).

43. Canny, J. F. and Lin, M. C. (1993). “An opportunistic global path planner”. In: *Algorithmica* 10.2, pp. 102–120 (cit. on p. 86).

44. Cao, Z. L., Huang, Y., and Hall, E. L. (1988). “Region filling operations with random obstacle avoidance for mobile robots”. In: *Journal of Robotic Systems* 5.2, pp. 87–102 (cit. on p. 43).

45. Cavanini, L., Ippoliti, G., and Camacho, E. F. (2021). “Model Predictive Control for a Linear Parameter Varying Model of an UAV”. In: *Journal of Intelligent & Robotic Systems* 101.3, pp. 1–18 (cit. on p. 91).

46. Chao, Z., Ming, L., Shaolei, Z., and Wenguang, Z. (2011). “Collision-free UAV formation flight control based on nonlinear MPC”. In: *2011 International Conference on Electronics, Communications and Control (ICECC)*, pp. 1951–1956 (cit. on pp. 91, 100).

47. Chen, M. and Rincon-Mora, G. (2006). “Accurate electrical battery model capable of predicting runtime and I-V performance”. In: *IEEE Transactions on Energy Conversion* 21.2, pp. 504–511 (cit. on p. 65).

48. Chen, X. and Touba, N. A. (2009). “Ch. 2 - Fundamentals of CMOS design”. In: *Electronic Design Automation*. Ed. by L.-T. Wang, Y.-W. Chang, and K.-T. (Cheng. Boston: Morgan Kaufmann, pp. 39–95 (cit. on p. 34).

49. Cheng, K. P., Mohan, R. E., Nhan, N. H. K., and Le, A. V. (2019). “Graph Theory-Based Approach to Accomplish Complete Coverage Path Planning Tasks for Reconfigurable Robots”. In: *IEEE Access* 7, pp. 94642–94657 (cit. on p. 44).

50. Choset, H., Acar, E., Rizzi, A., and Luntz, J. (2000). “Exact cellular decompositions in terms of critical points of Morse functions”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 3, pp. 2270–2277 (cit. on pp. 44, 85–88, 90).

51. Choset, H. (2000). “Coverage of known spaces: The boustrophedon cellular decomposition”. In: *Autonomous Robots* 9.3, pp. 247–253 (cit. on p. 85).

52. — (2001). “Coverage for robotics–A survey of recent results”. In: *Annals of Mathematics and Artificial Intelligence* 31, pp. 113–126 (cit. on pp. 24, 25, 41, 43, 44, 47, 84, 85).

53. Choset, H. M., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., Thrun, S., and Arkin, R. C. (2005). *Principles of robot motion: theory, algorithms, and implementation*. MIT press (cit. on pp. 25, 43, 79, 80, 84, 85, 87).

54. Choset, H. and Pignon, P. (1998). “Coverage Path Planning: The Boustrophedon Cellular Decomposition”. In: *Field and Service Robotics*. Springer London, pp. 203–209 (cit. on pp. 24, 44, 84).

55. Chowdhury, P. and Chakrabarti, C. (2005). “Static task-scheduling algorithms for battery-powered DVS systems”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13.2, pp. 226–237 (cit. on p. 39).

56. Collange, S., Defour, D., and Tisserand, A. (2009). “Power Consumption of GPUs from a Software Perspective”. In: *Computational Science – ICCS 2009*. Ed. by G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, and P. M. A. Sloot. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 914–923 (cit. on p. 38).

57. Colombatti, G., Aboudan, A., La Gloria, N., Debei, S., and Flamini, E. (2011). “Lighter-Than-Air UAV with slam capabilities for mapping applications and atmpsphere analysys.” In: *Memorie della Societa Astronomica Italiana Supplementi* 16, p. 42 (cit. on p. 6).

58. Colomina, I. and Molina, P. (2014). “Unmanned aerial systems for photogrammetry and remote sensing: A review”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 92, pp. 79–97 (cit. on p. 4).

59. Corke, P. (2017). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. 2nd. Springer Publishing Company, Incorporated (cit. on pp. 1, 6).

60. Crow, B. P., Widjaja, I., Kim, J. G., and Sakai, P. T. (1997). “IEEE 802.11 wireless local area networks”. In: *IEEE Communications magazine* 35.9, pp. 116–126 (cit. on pp. 28, 114).

61. Cui, J. Q., Phang, S. K., Ang, K. Z., Wang, F., Dong, X., Ke, Y., Lai, S., Li, K., Li, X., Lin, F., et al. (2015). “Drones for cooperative search and rescue in post-disaster situation”. In: *2015 IEEE 7th international conference on cybernetics and intelligent systems (CIS) and IEEE conference on robotics, automation and mechatronics (RAM)*. IEEE, pp. 167–174 (cit. on p. 4).

62. Czarnul, P., Proficz, J., and Krzywaniak, A. (2019). “Energy-aware high-performance computing: survey of state-of-the-art tools, techniques, and environments”. In: *Scientific Programming* 2019 (cit. on p. 35).

63. Dadkhah, N. and Mettler, B. (2012). “Survey of motion planning literature in the presence of uncertainty: Considerations for UAV guidance”. In: *Journal of Intelligent & Robotic Systems* 65.1, pp. 233–246 (cit. on p. 47).

64. Daponte, P., De Vito, L., Glielmo, L., Iannelli, L., Liuzza, D., Picariello, F., and Silano, G. (2019). “A review on the use of drones for precision agriculture”. In: *IOP Conference Series: Earth and Environmental Science*. Vol. 275. 1. IOP Publishing, p. 012022 (cit. on pp. 2, 4, 10).

65. Deng, Z., Yang, L., Cai, Y., and Deng, H. (2017). “Maximum Available Capacity and Energy Estimation Based on Support Vector Machine Regression for Lithium-ion Battery”. In: *Energy Procedia* 107. 3rd International Conference on Energy and Environment Research, ICEER 2016, pp. 68–75 (cit. on p. 94).

66. Dharmadhikari, M., Dang, T., Solanka, L., Loje, J., Nguyen, H., Khedekar, N., and Alexis, K. (2020). “Motion Primitives-based Path Planning for Fast and Agile Exploration using Aerial Robots”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 179–185 (cit. on p. 56).

67. Di Franco, C. and Buttazzo, G. (2015). “Energy-Aware Coverage Path Planning of UAVs”. In: *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, pp. 111–117 (cit. on pp. 25, 48).

68. — (2016). “Coverage path planning for UAVs photogrammetry with energy and resolution constraints”. In: *Journal of Intelligent & Robotic Systems* 83.3, pp. 445–462 (cit. on p. 48).

69. Diehl, M., Bock, H., Diedam, H., and Wieber, P.-B. (2006). “Fast Direct Multiple Shooting Algorithms for Optimal Robot Control”. In: *Fast Motions in Biomechanics and Robotics: Optimization and Feedback Control*. Springer Berlin Heidelberg, pp. 65–93 (cit. on p. 97).

70. Dille, M. and Singh, S. (2013). “Efficient Aerial Coverage Search in Road Networks”. In: *AIAA Guidance, Navigation, and Control (GNC) Conference*, pp. 1–20 (cit. on pp. 25, 49, 89).

71. Dong, F., Heinemann, W., and Kasper, R. (2011). “Development of a row guidance system for an autonomous robot for white asparagus harvesting”. In: *Computers and Electronics in Agriculture* 79.2, pp. 216–225 (cit. on p. 2).

72. Doyle, M., Fuller, T. F., and Newman, J. (1993). “Modeling of Galvanostatic Charge and Discharge of the Lithium/Polymer/Insertion Cell”. In: *Journal of The Electrochemical Society* 140.6, pp. 1526–1533 (cit. on p. 40).

73. Dressler, F. and Dietrich, I. (2006). “Lifetime Analysis in Heterogeneous Sensor Networks”. In: *9th EUROMICRO Conference on Digital System Design (DSD’06)*, pp. 606–616 (cit. on p. 43).

74. Dressler, F. and Fuchs, G. (2005). “Energy-aware operation and task allocation of autonomous robots”. In: *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo’05.* IEEE, pp. 163–168 (cit. on p. 42).

75. Edan, Y., Rogozin, D., Flash, T., and Miles, G. E. (2000). “Robotic melon harvesting”. In: *IEEE Transactions on Robotics and Automation* 16.6, pp. 831–835 (cit. on p. 2).

76. Eiselt, H. A. and Laporte, G. (2000). “A Historical Perspective on Arc Routing”. In: *Arc Routing: Theory, Solutions and Applications*. Boston, MA: Springer US, pp. 1–16 (cit. on p. 49).

77. Erickson, D. (2003). “Non-learning artificial neural network approach to motion planning for the Pioneer robot”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*. Vol. 1, pp. 112–117 (cit. on p. 50).

78. Ersal, T., Kim, Y., Broderick, J., Guo, T., Sadrpour, A., Stefanopoulou, A., Siegel, J., Tilbury, D., Atkins, E., Peng, H., et al. (2014). “Keeping ground robots on the move through battery & mission management”. In: *Mechanical Engineering* 136.06, pp. 1–6 (cit. on p. 51).

79. Espedal, I. B., Jinasena, A., Burheim, O. S., and Lamb, J. J. (2021). In: *Energies* 14.11 (cit. on p. 94).

80. Fekete, S., Arkin, E., and Mitchell, J. (1994). “The lawnmower problem and other geometric path covering problems”. In: *15th International Symposium on Mathematical Programming* (cit. on p. 43).

81. Feynman, R., Leighton, R., and Sands, M. (2015). *The Feynman Lectures on Physics, Vol. II: The New Millennium Edition: Mainly Electromagnetism and Matter*. v. 2. Basic Books (cit. on p. 79).

82. Fisher, M., Dennis, L., and Webster, M. (2013). “Verifying autonomous systems”. In: *Communications of the ACM* 56.9, pp. 84–93 (cit. on p. 1).

83. Flautner, K., Reinhardt, S., and Mudge, T. (2001). “Automatic Performance Setting for Dynamic Voltage Scaling”. In: *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*. MobiCom ’01. Rome, Italy: Association for Computing Machinery, pp. 260–271 (cit. on p. 34).

84. Floreano, D. and Wood, R. J. (2015). “Science, technology and the future of small autonomous drones”. In: *Nature* 521.7553, pp. 460–466 (cit. on p. 5).

85. Fomenko, A. T. and Kunii, T. L. (1997). *Topological modeling for visualization*. Springer Japan (cit. on p. 49).

86. Forejt, V., Kwiatkowska, M., and Parker, D. (2012). “Pareto Curves for Probabilistic Model Checking”. In: *Automated Technology for Verification and Analysis*. Springer Berlin Heidelberg, pp. 317–332 (cit. on p. 52).

87. Fuchs, G., Truchat, S., and Dressler, F. (2006). “Distributed Software Management in Sensor Networks using Profiling Techniques”. In: *2006 1st International Conference on Communication Systems Software Middleware*, pp. 1–6 (cit. on p. 43).

88. Fui Liew, C., DeLatte, D., Takeishi, N., and Yairi, T. (2017). “Recent Developments in Aerial Robotics: A Survey and Prototypes Overview”. In: *arXiv e-prints*, arXiv–1711 (cit. on p. 6).

89. Gabriely, Y. and Rimon, E. (2002). “Spiral-STC: an on-line coverage algorithm of grid environments by a mobile robot”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. Vol. 1, pp. 954–960 (cit. on p. 44).

90. Galceran, E. and Carreras, M. (2013). “A survey on coverage path planning for robotics”. In: *Robotics and Autonomous Systems* 61.12, pp. 1258–1276 (cit. on pp. 24, 43, 44, 46, 47, 84, 86).

91. Garcia De Marina, H., Kapitanyuk, Y. A., Bronz, M., Hattenberger, G., and Cao, M. (2017). “Guidance algorithm for smooth trajectory tracking of a fixed wing UAV flying in wind flows”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5740–5745 (cit. on pp. 10, 79, 81, 82).

92. García-Martín, E., Rodrigues, C. F., Riley, G., and Grahn, H. (2019). “Estimation of energy consumption in machine learning”. In: *Journal of Parallel and Distributed Computing* 134, pp. 75–88 (cit. on pp. 37, 38).

93. Gavilan, F., Vazquez, R., and Camacho, E. F. (2015). “An iterative model predictive control algorithm for UAV guidance”. In: *IEEE Transactions on Aerospace and Electronic Systems* 51.3, pp. 2406–2419 (cit. on pp. 91, 100).

94. Geem, Z. W. (2009). *Music-inspired harmony search algorithm: theory and applications*. Vol. 191. Springer (cit. on p. 49).

95. Giusti, A., Guzzi, J., Cireşan, D. C., He, F.-L., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Caro, G. D., Scaramuzza, D., and Gambardella, L. M. (2016). “A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots”. In: *IEEE Robotics and Automation Letters* 1.2, pp. 661–667 (cit. on p. 57).

96. Goerzen, C., Kong, Z., and Mettler, B. (2010). “A survey of motion planning algorithms from the perspective of autonomous UAV guidance”. In: *Journal of Intelligent and Robotic Systems* 57.1, pp. 65–100 (cit. on p. 47).

97. Gold, S. (1997). “A PSPICE macromodel for lithium-ion batteries”. In: *The Twelfth Annual Battery Conference on Applications and Advances*, pp. 215–222 (cit. on p. 40).

98. Gonçalves, V. M., Pimenta, L. C., Maia, C. A., Dutra, B. C., and Pereira, G. A. (2010). “Vector fields for robot navigation along time-varying curves in n-dimensions”. In: *IEEE Transactions on Robotics* 26.4, pp. 647–659 (cit. on p. 79).

99. Gong, Z., Li, J., and Li, W. (2016). “A low cost indoor mapping robot based on TinySLAM algorithm”. In: *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pp. 4549–4552 (cit. on p. 57).

100. González-Jorge, H., Martínez-Sánchez, J., Bueno, M., et al. (2017). “Unmanned aerial systems for civil applications: A review”. In: *Drones* 1.1, p. 2 (cit. on p. 4).

101. Goraczko, M., Liu, J., Lymberopoulos, D., Matic, S., Priyantha, B., and Zhao, F. (2008). “Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems”. In: *2008 45th ACM/IEEE Design Automation Conference*. IEEE, pp. 191–196 (cit. on p. 35).

102. Grüne, L. and Pannek, J. (2017). “Numerical optimal control of nonlinear systems”. In: *Nonlinear model predictive control*. Springer Cham, pp. 275–339 (cit. on pp. 93, 96, 97).

103. Hajjaj, S. S. H. and Sahari, K. S. M. (2014). “Review of research in the area of agriculture mobile robots”. In: *The 8th International Conference on Robotic, Vision, Signal Processing & Power Applications*. Springer, pp. 107–117 (cit. on p. 2).

104. Hamza, A. and Ayanian, N. (2017). “Forecasting Battery State of Charge for Robot Missions”. In: *Proceedings of the Symposium on Applied Computing*. New York, NY, USA: Association for Computing Machinery, pp. 249–255 (cit. on p. 63).

105. Hasan, A., Skriver, M., and Johansen, T. A. (2018). “Exogenous kalman filter for state-of-charge estimation in lithium-ion batteries”. In: *2018 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, pp. 1403–1408 (cit. on pp. 40, 41, 63–65).

106. Haugen, J. and Imsland, L. (2016). “Monitoring Moving Objects Using Aerial Mobile Sensors”. In: *IEEE Transactions on Control Systems Technology* 24.2, pp. 475–486 (cit. on p. 6).

107. Hayat, S., Yanmaz, E., Brown, T. X., and Bettstetter, C. (2017). “Multi-objective UAV path planning for search and rescue”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5569–5574 (cit. on pp. 4, 46).

108. He, H., Xiong, R., and Fan, J. (2011). “Evaluation of Lithium-Ion Battery Equivalent Circuit Models for State of Charge Estimation by an Experimental Approach”. In: *Energies* 4.4, pp. 582–598 (cit. on pp. 64, 65).

109. Hinz, H. (2019). “Comparison of Lithium-Ion Battery Models for Simulating Storage Systems in Distributed Power Generation”. In: *Inventions* 4 (cit. on pp. 63–65).

110. Hobby, H. (2020). *Opterra 2m Wing BNF Basic*. URL: https://www.horizonhobby.com/opterra-2m-wing-bnf-basic-p-efl11150 (visited on 02/02/2020) (cit. on p. 2).

111. Hoffer, N. V., Coopmans, C., Jensen, A. M., and Chen, Y. (2014). “A survey and categorization of small low-cost unmanned aerial vehicle system identification”. In: *Journal of Intelligent & Robotic Systems* 74.1, pp. 129–145 (cit. on p. 7).

112. Holper, J. W., Henry, K. V., and Atkins, E. M. (2017). “Cyber-physical thermal modeling for a small UAS”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1757–1766 (cit. on p. 57).

113. Hong, I., Kirovski, D., Qu, G., Potkonjak, M., and B, S. M. (1999). “Power optimization of variable-voltage core-based systems”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18.12, pp. 1702–1714 (cit. on p. 39).

114. Hong, S. and Kim, H. (2010). “An integrated GPU power and performance model”. In: *ACM SIGARCH Computer Architecture News*. Vol. 38. 3. ACM, pp. 280–289 (cit. on p. 38).

115. Horowitz, M. (2014). “Computing’s energy problem (and what we can do about it)”. In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, pp. 10–14 (cit. on pp. 34, 53).

116. Hu, X., Li, S., and Peng, H. (2012). “A comparative study of equivalent circuit models for Li-ion batteries”. In: *Journal of Power Sources* 198, pp. 359–367 (cit. on p. 40).

117. Huang, W. (2001). “Optimal line-sweep-based decompositions for coverage algorithms”. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. Vol. 1, pp. 27–32 (cit. on pp. 44, 45, 48, 88).

118. Iserles, A. (2009). *A first course in the numerical analysis of differential equations*. 44. Cambridge university press (cit. on p. 98).

119. Jaramillo-Avila, U., Aitken, J. M., and Anderson, S. R. (2019). “Visual saliency with foveated images for fast object detection and recognition in mobile robots using low-power embedded GPUs”. In: *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, pp. 773–778 (cit. on p. 1).

120. Johansen, T. A. and Fossen, T. I. (2017). “The exogenous kalman filter (xkf)”. In: *International Journal of Control* 90.2, pp. 161–167 (cit. on p. 40).

121. Jwo, D.-J. and Cho, T.-S. (2007). “A practical note on evaluating Kalman filter performance optimality and degradation”. In: *Applied Mathematics and Computation* 193.2, pp. 482–505 (cit. on p. 100).

122. Kalman, R. E. (Mar. 1960). “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1, pp. 35–45 (cit. on p. 100).

123. Kang, Y. and Hedrick, J. K. (2009). “Linear Tracking for a Fixed-Wing UAV Using Nonlinear Model Predictive Control”. In: *IEEE Transactions on Control Systems Technology* 17.5, pp. 1202–1210 (cit. on pp. 91, 100).

124. Kapitanyuk, Y. A., Proskurnikov, A. V., and Cao, M. (2017). “A guiding vector-field algorithm for path-following control of nonholonomic mobile robots”. In: *IEEE Transactions on Control Systems Technology* 26.4, pp. 1372–1385 (cit. on p. 79).

125. Karaca, Y., Cicek, M., Tatli, O., Sahin, A., Pasli, S., Beser, M. F., and Turedi, S. (2018). “The potential use of unmanned aircraft systems (drones) in mountain search and rescue operations”. In: *The American journal of emergency medicine* 36.4, pp. 583–588 (cit. on p. 4).

126. Karaman, S. and Frazzoli, E. (2011). “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30.7, pp. 846–894 (cit. on p. 50).

127. Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S. (2011). “Anytime Motion Planning using the RRT*”. In: *2011 IEEE International Conference on Robotics and Automation*, pp. 1478–1483 (cit. on p. 50).

128. Kasichayanula, K., Terpstra, D., Luszczek, P., Tomov, S., Moore, S., and Peterson, G. D. (2012). “Power Aware Computing on GPUs”. In: *2012 Symposium on Application Accelerators in High Performance Computing*, pp. 64–73 (cit. on p. 37).

129. Keane, J. F. and Carr, S. S. (2013). “A brief history of early unmanned aircraft”. In: *Johns Hopkins APL Technical Digest* 32.3, pp. 558–571 (cit. on p. 3).

130. Kellermann, R., Biehle, T., and Fischer, L. (2020). “Drones for parcel and passenger transportation: A literature review”. In: *Transportation Research Interdisciplinary Perspectives* 4, p. 100088 (cit. on p. 4).

131. Kim, C. H. and Kim, B. K. (2005). “Energy-saving 3-step velocity control algorithm for battery-powered wheeled mobile robots”. In: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, pp. 2375–2380 (cit. on p. 41).

132. Kim, H. and Kim, B.-K. (2008). “Minimum-energy translational trajectory planning for battery-powered three-wheeled omni-directional mobile robots”. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. IEEE, pp. 1730–1735 (cit. on p. 41).

133. Kim, T. and Qiao, W. (2011). “A Hybrid Battery Model Capable of Capturing Dynamic Circuit Characteristics and Nonlinear Capacity Effects”. In: *IEEE Transactions on Energy Conversion* 26.4, pp. 1172–1180 (cit. on p. 40).

134. Kim, T., Qiao, W., and Qu, L. (2019). “An Enhanced Hybrid Battery Model”. In: *IEEE Transactions on Energy Conversion* 34.4, pp. 1848–1858 (cit. on p. 40).

135. Kim, Y. G., Kong, J., and Chung, S. W. (2018). “A Survey on Recent OS-Level Energy Management Techniques for Mobile Processing Units”. In: *IEEE Transactions on Parallel and Distributed Systems* 29.10, pp. 2388–2401 (cit. on p. 37).

136. Kirk, D. B. and Wen-Mei, W. H. (2016). *Programming massively parallel processors: a hands-on approach*. 3rd ed. Morgan Kaufmann (cit. on p. 107).

137. Kirk, D. E. (2004). *Optimal control theory: an introduction*. Courier Corporation (cit. on pp. 125, 128).

138. Kostadinov, D. and Scaramuzza, D. (2020). “Online Weight-adaptive Nonlinear Model Predictive Control”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1180–1185 (cit. on p. 91).

139. Kreciglowa, N., Karydis, K., and Kumar, V. (2017). “Energy efficiency of trajectory generation methods for stop-and-go aerial robot navigation”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 656–662 (cit. on p. 46).

140. Kuo, B. (1967). *Automatic Control Systems*. Electrical engineering series. Prentice-Hall (cit. on p. 69).

141. Kurzweil, P. and Scheuerpflug, W. (2021). “State-of-Charge Monitoring and Battery Diagnosis of Different Lithium Ion Chemistries Using Impedance Spectroscopy”. In: *Batteries* 7.1 (cit. on p. 94).

142. Kurzweil, P. and Shamonin, M. (2018). “State-of-Charge Monitoring by Impedance Spectroscopy during Long-Term Self-Discharge of Supercapacitors and Lithium-Ion Batteries”. In: *Batteries* 4.3 (cit. on p. 94).

143. Kwon, W. H. and Han, S. H. (2005). *Receding horizon control: model predictive control for state models*. London: Springer-Verlag (cit. on pp. 91, 127).

144. Lahijanian, M., Svorenova, M., Morye, A. A., Yeomans, B., Rao, D., Posner, I., Newman, P., Kress-Gazit, H., and Kwiatkowska, M. (2018). “Resource-Performance Tradeoff Analysis for Mobile Robots”. In: *IEEE Robotics and Automation Letters* 3.3, pp. 1840–1847 (cit. on pp. 50–52, 77, 91).

145. LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press (cit. on pp. 25, 33, 42, 43, 47, 50, 79, 86, 127).

146. Lee, B. C. and Brooks, D. M. (2006a). “Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction”. In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, pp. 185–194 (cit. on pp. 39, 59).

147. — (2006b). “Statistically rigorous regression modeling for the microprocessor design space”. In: *ISCA-33: Workshop on Modeling, Benchmarking, and Simulation* (cit. on pp. 39, 59).

148. Lee, S., Kim, J., Lee, J., and Cho, B. (2008). “State-of-charge and capacity estimation of lithium-ion battery using a new open-circuit voltage versus state-of-charge”. In: *Journal of Power Sources* 185.2, pp. 1367–1373 (cit. on p. 40).

149. Lee, T.-K., Baek, S.-H., Choi, Y.-H., and Oh, S.-Y. (2011). “Smooth coverage path planning and control of mobile robots based on high-resolution grid map representation”. In: *Robotics and Autonomous Systems* 59.10, pp. 801–812 (cit. on p. 44).

150. Lemay, M., Michaud, F., Letourneau, D., and Valin, J.-M. (2004). “Autonomous initialization of robot formations”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004*. Vol. 3, pp. 3018–3023 (cit. on p. 50).

151. Leng, J., Hetherington, T., ElTantawy, A., Gilani, S., Kim, N. S., Aamodt, T. M., and Reddi, V. J. (2013). “GPUWattch: Enabling Energy Optimizations in GPGPUs”. In: *SIGARCH Comput. Archit. News* 41.3, pp. 487–498 (cit. on p. 38).

152. Li, Y., Chen, H., Joo Er, M., and Wang, X. (2011). “Coverage path planning for UAVs based on enhanced exact cellular decomposition method”. In: *Mechatronics* 21.5. Special Issue on Development of Autonomous Unmanned Aerial Vehicles, pp. 876–885 (cit. on pp. 26, 44, 48, 88).

153. Li, Z., Liu, J., Li, P., and Li, W. (2008). “Analysis of workspace and kinematics for a tomato harvesting robot”. In: *2008 International Conference on Intelligent Computation Technology and Automation (ICICTA)*. Vol. 1. IEEE, pp. 823–827 (cit. on p. 2).

154. Lindemann, S. R. and LaValle, S. M. (2005). “Smoothly blending vector fields for global robot navigation”. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pp. 3553–3559 (cit. on p. 79).

155. Lotfi, N., Landers, R. G., Li, J., and Park, J. (2017). “Reduced-Order Electrochemical Model-Based SOC Observer With Output Model Uncertainty Estimation”. In: *IEEE Transactions on Control Systems Technology* 25.4, pp. 1217–1230 (cit. on p. 40).

156. Lottes, P., Khanna, R., Pfeifer, J., Siegwart, R., and Stachniss, C. (2017). “UAV-based crop and weed classification for smart farming”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3024–3031 (cit. on p. 4).

157. Lu, L., Han, X., Li, J., Hua, J., and Ouyang, M. (2013). “A review on the key issues for lithium-ion battery management in electric vehicles”. In: *Journal of Power Sources* 226, pp. 272–288 (cit. on p. 94).

158. Luo, C. and Suda, R. (2011). “A Performance and Energy Consumption Analytical Model for GPU”. In: *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 658–665 (cit. on p. 38).

159. Luo, J. and Jha, N. K. (2001). “Battery-aware static scheduling for distributed real-time embedded systems”. In: *Proceedings of the 38th annual Design Automation Conference*. ACM, pp. 444–449 (cit. on p. 39).

160. Lv, H., Huang, X., and Liu, Y. (2020). “Analysis on pulse charging–discharging strategies for improving capacity retention rates of lithium-ion batteries”. In: *Ionics* 26.4, pp. 1749–1770 (cit. on p. 92).

161. Ma, K., Li, X., Chen, W., Zhang, C., and Wang, X. (2012). “GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures”. In: *2012 41st International Conference on Parallel Processing*. IEEE, pp. 48–57 (cit. on p. 37).

162. Madani, S. S., Schaltz, E., and Knudsen Kær, S. (2019). “An Electrical Equivalent Circuit Model of a Lithium Titanate Oxide Battery”. In: *Batteries* 5.1 (cit. on p. 63).

163. Majeed, A. and Lee, S. (2019). “A New Coverage Flight Path Planning Algorithm Based on Footprint Sweep Fitting for Unmanned Aerial Vehicle Navigation in Urban Environments”. In: *Applied Sciences* 9.7, pp. 1–17 (cit. on p. 89).

164. Mannadiar, R. and Rekleitis, I. (2010). “Optimal coverage of a known arbitrary environment”. In: *2010 IEEE International Conference on Robotics and Automation*, pp. 5525–5530 (cit. on pp. 25, 49, 85).

165. Marcicki, J., Canova, M., Conlisk, A. T., and Rizzoni, G. (2013). “Design and parametrization analysis of a reduced-order electrochemical model of graphite/LiFePO4 cells for SOC/SOH estimation”. In: *Journal of Power Sources* 237, pp. 310–324 (cit. on p. 40).

166. Marowka, A. (2017). “Energy-aware modeling of scaled heterogeneous systems”. In: *International Journal of Parallel Programming* 45.5, pp. 1026–1045 (cit. on p. 35).

167. Maza, I. and Ollero, A. (2007). “Multiple UAV cooperative searching operation using polygon area decomposition and efficient coverage algorithms”. In: *Distributed Autonomous Robotic Systems 6*. Tokyo: Springer Japan, pp. 221–230 (cit. on p. 46).

168. Mei, Y., Lu, Y.-H., Hu, Y. C., and Lee, C. G. (2004). “Energy-efficient motion planning for mobile robots”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*. Vol. 5. IEEE, pp. 4344–4349 (cit. on pp. 1, 41, 42).

169. — (2005). “A case study of mobile robot's energy consumption and conservation techniques”. In: *ICAR'05. Proceedings., 12th International Conference on Advanced Robotics, 2005.* IEEE, pp. 492–497 (cit. on pp. 1, 41, 42, 50, 53).

170. Mei, Y., Lu, Y.-H., Hu, Y., and Lee, C. (2005a). “Deployment Strategy for Mobile Robots with Energy and Timing Constraints”. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2816–2821 (cit. on p. 50).

171. — (2005b). “Reducing the number of mobile sensors for coverage tasks”. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1426–1431 (cit. on p. 50).

172. Mei, Y., Lu, Y.-H., Hu, Y. C., and Lee, C. G. (2006). “Deployment of mobile robots with energy and timing constraints”. In: *IEEE Transactions on robotics* 22.3, pp. 507–522 (cit. on pp. 1, 50).

173. Mei, Y., Lu, Y.-H., Lee, C., and Hu, Y. (2006). “Energy-efficient mobile robot exploration”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* Pp. 505–511 (cit. on p. 51).

174. Milas, A. S., Cracknell, A. P., and Warner, T. A. (2018). “Drones - the third generation source of remote sensing data”. In: *International Journal of Remote Sensing* 39.21, pp. 7125–7137 (cit. on p. 4).

175. Mittal, S. (2019). “A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform”. In: *Journal of Systems Architecture* 97, pp. 428–442 (cit. on p. 37).

176. Mittal, S. and Vetter, J. S. (2014). “A Survey of Methods for Analyzing and Improving GPU Energy Efficiency”. In: *ACM Comput. Surv.* 47.2, pp. 1–23 (cit. on pp. 37, 38).

177. Moler, C. and Van Loan, C. (2003). “Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later”. In: *SIAM review* 45.1, pp. 3–49 (cit. on p. 69).

178. Morbidi, F., Cano, R., and Lara, D. (2016). “Minimum-energy path generation for a quadrotor UAV”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1492–1498 (cit. on p. 46).

179. Moura, S. J., Argomedo, F. B., Klein, R., Mirtabatabaei, A., and Krstic, M. (2017). “Battery State Estimation for a Single Particle Model With Electrolyte Dynamics”. In: *IEEE Transactions on Control Systems Technology* 25.2, pp. 453–468 (cit. on p. 40).

180. Mousavi G., S. and Nikdel, M. (2014). “Various battery models for various simulation studies and applications”. In: *Renewable and Sustainable Energy Reviews* 32, pp. 477–485 (cit. on pp. 64, 65).

181. Nam, L. H., Huang, L., Li, X. J., and Xu, J. F. (2016). “An approach for coverage path planning for UAVs”. In: *2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)*, pp. 411–416 (cit. on p. 47).

182. Needham, T. (1998). *Visual complex analysis*. Oxford University Press (cit. on p. 79).

183. Nikov, K., Nunez-Yanez, J. L., and Horsnell, M. (2015). “Evaluation of hybrid run-time power models for the ARM big. LITTLE architecture”. In: *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*. IEEE, pp. 205–210 (cit. on pp. 40, 110).

184. Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media (cit. on pp. 96, 97, 125).

185. Noor, N. M., Abdullah, A., and Hashim, M. (2018). “Remote sensing UAV/drones and its applications for urban areas: a review”. In: *IOP conference series: Earth and environmental science*. Vol. 169. 1. IOP Publishing, p. 012003 (cit. on p. 4).

186. Nunez-Yanez, J. and Lore, G. (2013). “Enabling accurate modeling of power and energy consumption in an ARM-based System-on-Chip”. In: *Microprocessors and Microsystems* 37.3, pp. 319–332 (cit. on pp. 40, 110).

187. NVIDIA (2020). *NVIDIA Jetson Nano developer kit*. URL: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (visited on 02/02/2020) (cit. on p. 116).

188. O’Brien, K., Pietri, I., Reddy, R., Lastovetsky, A., and Sakellariou, R. (2017). “A Survey of Power and Energy Predictive Models in HPC Systems and Applications”. In: *ACM Comput. Surv.* 50.3, pp. 1–38 (cit. on p. 35).

189. O’Neal, K. and Brisk, P. (2018). “Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey”. In: *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 763–768 (cit. on p. 34).

190. Ogata, K. (2002). *Modern Control Engineering*. Prentice Hall (cit. on p. 69).

191. Ondrúška, P., Gurău, C., Marchegiani, L., Tong, C. H., and Posner, I. (2015). “Scheduled perception for energy-efficient path following”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4799–4806 (cit. on pp. 33, 41, 50–53, 91, 123).

192. Panagou, D. (2014). “Motion planning and collision avoidance using navigation vector fields”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2513–2518 (cit. on p. 79).

193. Panigrahi, D., Chiasserini, C., Dey, S., Rao, R., Raghunathan, A., and Lahiri, K. (2001). “Battery life estimation of mobile embedded systems”. In: *VLSI Design 2001. Fourteenth International Conference on VLSI Design*, pp. 57–63 (cit. on p. 41).

194. Papachristos, C., Tzoumanikas, D., and Tzes, A. (2015). “Aerial robotic tracking of a generalized mobile target employing visual and spatio-temporal dynamic subject perception”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4319–4324 (cit. on p. 57).

195. Paparazzi (2016). *UAV open-source project*. URL: http://wiki.paparazziuav.org/ (visited on 09/01/2016) (cit. on pp. 10, 110).

196. Paucar, C., Morales, L., Pinto, K., Sánchez, M., Rodríguez, R., Gutierrez, M., and Palacios, L. (2018). “Use of drones for surveillance and reconnaissance of military areas”. In: *International Conference of Research Applied to Defense and Security*. Springer, pp. 119–132 (cit. on p. 4).

197. Paulen, R. and Fikar, M. (2016). “Solution of Optimal Control Problems”. In: *Optimal Operation of Batch Membrane Processes*. Cham: Springer International Publishing, pp. 37–56 (cit. on pp. 126, 127).

198. Pedram, M. and Wu, Q. (1999). “Design Considerations for Battery-Powered Electronics”. In: *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, pp. 861–866 (cit. on p. 40).

199. Peng, T., Zhang, D., Liu, R., Asari, V. K., and Loomis, J. S. (2019). “Evaluating the Power Efficiency of Visual SLAM on Embedded GPU Systems”. In: *2019 IEEE National Aerospace and Electronics Conference (NAECON)*, pp. 117–121 (cit. on p. 56).

200. Pensieri, M. G., Garau, M., and Barone, P. M. (2020). “Drones as an Integral Part of Remote Sensing Technologies to Help Missing People”. In: *Drones* 4.2, p. 15 (cit. on p. 4).

201. Percin, M., Eisma, J., Van Oudheusden, B., Remes, B., Ruijsink, R., and De Wagter, C. (2012). “Flow Visualization in the Wake of the Flapping-Wing MAV ’DelFly II’ in Forward Flight”. In: *30th AIAA Applied Aerodynamics Conference*. AIAA (cit. on p. 6).

202. Poe, W. A. and Mokhatab, S. (2017). “Process Control”. In: *Modeling, Control, and Optimization of Natural Gas Processing Plants*. Ed. by W. A. Poe and S. Mokhatab. Boston: Gulf Professional Publishing, pp. 97–172 (cit. on p. 127).

203. Pontryagin, L. S., Mishchenko, E., Boltyanskii, V., and Gamkrelidze, R. (1962). *The mathematical theory of optimal processes*. John Wiley & Sons (cit. on p. 127).

204. Popović, M., Hitz, G., Nieto, J., Sa, I., Siegwart, R., and Galceran, E. (2017). “Online informative path planning for active classification using UAVs”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5753–5758 (cit. on pp. 4, 46).

205. Puri, V., Nayyar, A., and Raja, L. (2017). “Agriculture drones: A modern breakthrough in precision agriculture”. In: *Journal of Statistics and Management Systems* 20.4, pp. 507–518 (cit. on pp. 2, 4).

206. PX4 (2016). *PX4 open-source autopilot*. URL: https://px4.io/ (visited on 09/01/2016) (cit. on p. 10).

207. Qingchun, F., Wengang, Z., Quan, Q., Kai, J., and Rui, G. (2012). “Study on strawberry robotic harvesting system”. In: *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*. Vol. 1. IEEE, pp. 320–324 (cit. on p. 2).

208. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2, p. 5 (cit. on pp. 12, 60).

209. Rakhmatov, D. and Vrudhula, S. (2001). “An analytical high-level battery model for use in energy management of portable electronic systems”. In: *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, pp. 488–493 (cit. on p. 40).

210. Ramasamy, M. and Ghose, D. (2017). “A heuristic learning algorithm for preferential area surveillance by unmanned aerial vehicles”. In: *Journal of Intelligent & Robotic Systems* 88.2, pp. 655–681 (cit. on p. 4).

211. Rao, R., Vrudhula, S., and Rakhmatov, D. N. (2003). “Battery modeling for energy aware system design”. In: *Computer* 36.12, pp. 77–87 (cit. on pp. 40, 62, 63).

212. Rao, S. S. (2019). *Engineering optimization: theory and practice*. John Wiley & Sons (cit. on p. 128).

213. Rao, V., Singhal, G., Kumar, A., and Navet, N. (2005). “Battery model for embedded systems”. In: *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, pp. 105–110 (cit. on pp. 33, 34, 41).

214. Rawlings, J. B., Mayne, D. Q., and Diehl, M. (2017). *Model predictive control: theory, computation, and design*. Vol. 2. Madison, Wisconsin: Nob Hill Publishing (cit. on pp. 11, 91–93, 96, 98, 125–128).

215. Reddy, B. K., Walker, M. J., Balsamo, D., Diestelhorst, S., Al-Hashimi, B. M., and Merrett, G. V. (2017). “Empirical CPU power modelling and estimation in the gem5 simulator”. In: *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 1–8 (cit. on p. 39).

216. Redmon, J. (2013–2016). *Darknet: Open Source Neural Networks in C*. `http://pjreddie.com/darknet/` (cit. on p. 104).

217. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788 (cit. on pp. 104, 111).

218. Redmon, J. and Farhadi, A. (2017). “YOLO9000: Better, Faster, Stronger”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, USA, pp. 6517–6525 (cit. on p. 104).

219. Renzaglia, A., Reymann, C., and Lacroix, S. (2016). “Monitoring the evolution of clouds with UAVs”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 278–283 (cit. on p. 4).

220. Rizvi, S. T. H., Cabodi, G., Patti, D., and Gulzar, M. M. (2017). “A general-purpose graphics processing unit (gpgpu)-accelerated robotic controller using a low power mobile platform”. In: *Journal of Low Power Electronics and Applications* 7.2, p. 10 (cit. on p. 1).

221. Rossiter, J. A. (2004). *Model-based predictive control: a practical approach*. Boca Raton, Florida: CRC press (cit. on pp. 91, 127).

222. Ryou, G., Sim, Y., Yeon, S. H., and Seok, S. (2018). “Applying Asynchronous Deep Classification Networks and Gaming Reinforcement Learning-Based Motion Planners to Mobile Robots”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6268–6275 (cit. on p. 56).

223. Sa, I., Chen, Z., Popović, M., Khanna, R., Liebisch, F., Nieto, J., and Siegwart, R. (2018). “weedNet: Dense Semantic Weed Classification Using Multispectral Images and MAV for Smart Farming”. In: *IEEE Robotics and Automation Letters* 3.1, pp. 588–595 (cit. on p. 4).

224. Sadat, S. A., Wawerla, J., and Vaughan, R. T. (2014). “Recursive non-uniform coverage of unknown terrains for UAVs”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1742–1747 (cit. on p. 47).

225. Sadrpour, A., Jin, J., and Ulsoy, A. G. (2013a). “Mission Energy Prediction for Unmanned Ground Vehicles Using Real-time Measurements and Prior Knowledge”. In: *Journal of Field Robotics* 30.3, pp. 399–414 (cit. on pp. 50, 51).

226. *Real-Time Energy-Efficient Path Planning for Unmanned Ground Vehicles Using Mission Prior Knowledge* (2013b). ASME, pp. 1–10 (cit. on p. 51).

227. Sadrpour, A., Jin, J., and Ulsoy, A. G. (2013c). “Experimental validation of mission energy prediction model for unmanned ground vehicles”. In: *2013 American Control Conference*. IEEE, pp. 5960–5965 (cit. on pp. 50, 51).

228. Salameh, Z., Casacca, M., and Lynch, W. (1992). “A mathematical model for lead-acid batteries”. In: *IEEE Transactions on Energy Conversion* 7.1, pp. 93–98 (cit. on p. 65).

229. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). “Mobilenetv2: Inverted residuals and linear bottlenecks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520 (cit. on p. 113).

230. Satria, M. T., Gurumani, S., Zheng, W., Tee, K. P., Koh, A., Yu, P., Rupnow, K., and Chen, D. (2016). “Real-time system-level implementation of a telepresence robot using an embedded GPU platform”. In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 1445–1448 (cit. on p. 1).

231. Schneier, B. (1993). “Description of a new variable-length key, 64-bit block cipher (Blowfish)”. In: *International Workshop on Fast Software Encryption*. Springer, pp. 191–204 (cit. on p. 111).

232. Schuyler, T. J., Gohari, S., Pundsack, G., Berchoff, D., and Guzman, M. I. (2019). “Using a balloon-launched unmanned glider to validate real-time WRF modeling”. In: *Sensors* 19.8, p. 1914 (cit. on p. 4).

233. Seewald, A. (2020). “Beyond Traditional Energy Planning: the Weight of Computations in Planetary Exploration”. In: *IROS Workshop on Planetary Exploration Robots: Challenges and Opportunities (PLANROBO20)*. ETH Zurich, Department of Mechanical and Process Engineering, p. 3 (cit. on pp. 11, 12).

234. Seewald, A., Ebeid, E., and Schultz, U. P. (2019). “Dynamic Energy Modelling for SoC Boards: Initial Experiments”. In: *HLPGPU 2019: High-Level Programming for Heterogeneous and Hierarchical Parallel Systems*, p. 4 (cit. on pp. 37, 60, 103).

235. Seewald, A., Garcia de Marina, H., Midtiby, H. S., and Schultz, U. P. (2020). “Mechanical and Computational Energy Estimation of a Fixed-Wing Drone”. In: *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, pp. 135–142 (cit. on pp. 1, 11, 12, 66, 103, 110–113).

236. Seewald, A., Garcia de Marina, H., and Schultz, U. P. (n.d.). *Energy-Aware Dynamic Planning Algorithm for Autonomous UAVs*. In preparation (available online: `https://github.com/adamseew/iros-2021`) (cit. on pp. 11, 12, 103).

237. Seewald, A., Schultz, U. P., Ebeid, E., and Midtiby, H. S. (2021). “Coarse-Grained Computation-Oriented Energy Modeling for Heterogeneous Parallel Embedded Systems”. In: *International Journal of Parallel Programming* 49.2, pp. 136–157 (cit. on pp. 12, 39, 40, 55, 58–61, 64, 92, 103–111).

238. Seewald, A., Schultz, U. P., Roeder, J., Rouxel, B., and Grelck, C. (2019). “Component-based computation-energy modeling for embedded systems”. In: *Proceedings Companion of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. ACM, pp. 5–6 (cit. on pp. 12, 60, 103, 108).

239. Seguin, C., Blaquière, G., Loundou, A., Michelet, P., and Markarian, T. (2018). “Unmanned aerial vehicles (drones) to prevent drowning”. In: *Resuscitation* 127, pp. 63–67 (cit. on p. 4).

240. Sethi, S. P. (2019). *Optimal Control Theory: Applications to Management Science and Economics*. Cham: Springer International Publishing (cit. on pp. 125, 126, 128).

241. Shi, P. and Zhao, Y. (2006). “Application of Unscented Kalman Filter in the SOC Estimation of Li-ion Battery for Autonomous Mobile Robot”. In: *2006 IEEE International Conference on Information Acquisition*, pp. 1279–1283 (cit. on p. 63).

242. Shnaps, I. and Rimon, E. (2016). “Online Coverage of Planar Environments by a Battery Powered Autonomous Mobile Robot”. In: *IEEE Transactions on Automation Science and Engineering* 13.2, pp. 425–436 (cit. on pp. 44, 45).

243. Siciliano, B. and Khatib, O. (2016). *Springer handbook of robotics*. Springer. Chap. 44, pp. 1012–1014 (cit. on pp. 3–5, 7).

244. Simon, D. (2006). *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons (cit. on pp. 100, 128, 129).

245. Song, Y. and Scaramuzza, D. (2020). “Learning High-Level Policies for Model Predictive Control”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7629–7636 (cit. on p. 91).

246. Stastny, T. and Siegwart, R. (2018). “Nonlinear Model Predictive Guidance for Fixed-wing UAVs Using Identified Control Augmented Dynamics”. In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 432–442 (cit. on pp. 91, 100).

247. Stengel, R. F. (1994). *Optimal control and estimation*. Courier Corporation (cit. on pp. 128, 129).

248. Stroustrup, B. (1988). “What is object-oriented programming?” In: *IEEE Software* 5.3, pp. 10–20 (cit. on p. 60).

249. Sudhakar, S., Karaman, S., and Sze, V. (2020). “Balancing Actuation and Computing Energy in Motion Planning”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4259–4265 (cit. on pp. 1, 33, 50, 51, 103).

250. Sundén, B. (2019). “Chapter 6 - Thermal management of batteries”. In: *Hydrogen, Batteries and Fuel Cells*. Ed. by B. Sundén. Academic Press, pp. 93–110 (cit. on p. 94).

251. Syracuse, K. and Clark, W. (1997). “A statistical approach to domain performance modeling for oxyhalide primary lithium batteries”. In: *The Twelfth Annual Battery Conference on Applications and Advances*, pp. 163–170 (cit. on p. 40).

252. Szeliski, R. (2011). *Computer vision algorithms and applications*. London: Springer (cit. on p. 106).

253. Takouna, I., Dawoud, W., and Meinel, C. (2011). “Accurate Mutlicore Processor Power Models for Power-Aware Resource Management”. In: *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 419–426 (cit. on p. 39).

254. Tang, L. and Shao, G. (2015). “Drone remote sensing for forestry research and practices”. In: *Journal of Forestry Research* 26.4, pp. 791–797 (cit. on p. 4).

255. TeamPlay Consortium (2019). *Deliverable D4.3 - Report on Energy, Timing and Security Modeling of Complex Architectures - Version 2.0* (cit. on pp. 60, 105).

256. Tian, R., Park, S.-H., King, P. J., Cunningham, G., Coelho, J., Nicolosi, V., and Coleman, J. N. (2019). “Quantifying the factors limiting rate performance in battery electrodes”. In: *Nature communications* 10.1, pp. 1–11 (cit. on p. 92).

257. Torrente, G., Kaufmann, E., Föhn, P., and Scaramuzza, D. (2021). “Data-Driven MPC for Quadrotors”. In: *IEEE Robotics and Automation Letters* 6.2, pp. 3769–3776 (cit. on p. 91).

258. Ullah, M., Mohammed, A., and Alaya Cheikh, F. (2018). “Pednet: A spatio-temporal deep convolutional neural network for pedestrian segmentation”. In: *Journal of Imaging* 4.9, p. 107 (cit. on pp. 113, 116).

259. Valavanis, K. P. and Vachtsevanos, G. J. (2015). *Handbook of unmanned aerial vehicles*. Vol. 1. Springer (cit. on p. 3).

260. Valente, J., Del Cerro, J., Barrientos, A., and Sanz, D. (2013). “Aerial coverage optimization in precision agriculture management: A musical harmony inspired approach”. In: *Computers and Electronics in Agriculture* 99, pp. 153–159 (cit. on p. 49).

261. Viega, J., Messier, M., and Chandra, P. (2002). *Network security with openSSL: cryptography for secure communications*. O’Reilly (cit. on p. 111).

262. Von Stryk, O. and Bulirsch, R. (1992). “Direct and indirect methods for trajectory optimization”. In: *Annals of operations research* 37.1, pp. 357–373 (cit. on p. 125).

263. Voosen, P. (2019). *NASA to fly drone on Titan* (cit. on p. 5).

264. Wahab, M., Rios-Gutierrez, F., and El Shahat, A. (2015). *Energy modeling of differential drive robots*. IEEE (cit. on p. 41).

265. Walker, M. J., Diestelhorst, S., Hansson, A., Das, A. K., Yang, S., Al-Hashimi, B. M., and Merrett, G. V. (2017). “Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.1, pp. 106–119 (cit. on pp. 39, 40).

266. Wang, L., Ye, X., Xing, H., Wang, Z., and Li, P. (2020). “YOLO Nano Underwater: A Fast and Compact Object Detector for Embedded Device”. In: *Global Oceans 2020: Singapore – U.S. Gulf Coast*, pp. 1–4 (cit. on p. 56).

267. Wang, L. (2009). *Model predictive control system design and implementation using Matlab*. London: Springer Science & Business Media (cit. on pp. 91, 127).

268. Wang, X., Jiang, P., Li, D., and Sun, T. (2017). “Curvature Continuous and Bounded Path Planning for Fixed-Wing UAVs”. In: *Sensors* 17.9 (cit. on pp. 1, 25, 46, 85).

269. Watts, A. C., Ambrosia, V. G., and Hinkley, E. A. (2012). “Unmanned Aircraft Systems in Remote Sensing and Scientific Research: Classification and Considerations of Use”. In: *Remote Sensing* 4.6, pp. 1671–1692 (cit. on p. 7).

270. Wegner, P. (1990). “Concepts and Paradigms of Object-Oriented Programming”. In: *SIGPLAN OOPS Mess.* 1.1, pp. 7–87 (cit. on p. 60).

271. Wei, M. and Isler, V. (2018). “Coverage Path Planning Under the Energy Constraint”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 368–373 (cit. on pp. 42, 44, 46).

272. Wright, O. (Dec. 1913). “How We Made The First Flight”. In: *Flying* (cit. on p. iii).

273. Wu, G., Greathouse, J. L., Lyashevsky, A., Jayasena, N., and Chiou, D. (2015). “GPGPU performance and power estimation using machine learning”. In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 564–576 (cit. on p. 38).

274. Xia, B., Chen, C., Tian, Y., Wang, M., Sun, W., and Xu, Z. (2015). “State of charge estimation of lithium-ion batteries based on an improved parameter identification method”. In: *Energy* 90, pp. 1426–1434 (cit. on pp. 62, 63).

275. Xing, Y., He, W., Pecht, M., and Tsui, K. L. (2014). “State of charge estimation of lithium-ion batteries using the open-circuit voltage at various ambient temperatures”. In: *Applied Energy* 113, pp. 106–115 (cit. on p. 40).

276. Xu, A., Viriyasuthee, C., and Rekleitis, I. (2011). “Optimal complete terrain coverage using an Unmanned Aerial Vehicle”. In: *2011 IEEE International Conference on Robotics and Automation*, pp. 2513–2519 (cit. on pp. 25, 26, 44, 46, 49, 85, 88, 89).

277. — (2014). “Efficient complete coverage of a known arbitrary environment with applications to aerial operations”. In: *Autonomous Robots* 36.4, pp. 365–381 (cit. on pp. 25, 26, 49, 85, 88, 89).

278. Yamauchi, B. M. (2004). “PackBot: a versatile platform for military robotics”. In: *Unmanned Ground Vehicle Technology VI*. Ed. by G. R. Gerhart, C. M. Shoemaker, and D. W. Gage. Vol. 5422. International Society for Optics and Photonics. SPIE, pp. 228–237 (cit. on p. 51).

279. Yang, T.-J., Chen, Y.-H., Emer, J., and Sze, V. (2017). “A method to estimate the energy consumption of deep neural networks”. In: *2017 51st asilomar conference on signals, systems, and computers*. IEEE, pp. 1916–1920 (cit. on pp. 37, 38).

280. Yang, T.-J., Chen, Y.-H., and Sze, V. (2017). “Designing energy-efficient convolutional neural networks using energy-aware pruning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5687–5695 (cit. on p. 37).

281. Yeomans, B., Porav, H., Gadd, M., Barnes, D., Dequaire, J., Wilcox, T., Kyberd, S., Venn, S., and Newman, P. (2017). “MURFI 2016-From Cars to Mars: Applying autonomous vehicle navigation methods to a space rover mission”. In: *Proceedings of the 14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA), Leiden, Netherlands*, pp. 1–8 (cit. on p. 52).

282. Zamanakos, G., Seewald, A., Midtiby, H. S., and Schultz, U. P. (2020). “Energy-Aware Design of Vision-Based Autonomous Tracking and Landing of a UAV”. In: *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, pp. 294–297 (cit. on pp. 2, 12, 60, 103, 110).

283. Zelinsky, A., Jarvis, R., Byrne, J. C., and Yuta, S. (1993). “Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot”. In: *In Proceedings of International Conference on Advanced Robotics*, pp. 533–538 (cit. on p. 44).

284. Zhang, C., Allafi, W., Dinh, Q., Ascencio, P., and Marco, J. (2018). “Online estimation of battery equivalent circuit model parameters and state of charge using decoupled least squares technique”. In: *Energy* 142, pp. 678–688 (cit. on pp. 63–65).

285. Zhang, C., Li, K., Mcloone, S., and Yang, Z. (2014). “Battery modelling methods for electric vehicles - A review”. In: *2014 European Control Conference (ECC)*, pp. 2673–2678 (cit. on p. 63).

286. Zhang, F., Liu, G., and Fang, L. (2009). “Battery state estimation using Unscented Kalman Filter”. In: *2009 IEEE International Conference on Robotics and Automation*, pp. 1863–1868 (cit. on p. 63).

287. Zhang, F., Liu, G., Fang, L., and Wang, H. (2012). “Estimation of Battery State of Charge With H_∞ Observer: Applied to a Robot for Inspecting Power Transmission Lines”. In: *IEEE Transactions on Industrial Electronics* 59.2, pp. 1086–1095 (cit. on p. 63).

288. Zhang, L., Tiwana, B., Qian, Z., Wang, Z., Dick, R. P., Mao, Z. M., and Yang, L. (2010). “Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones”. In: *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. New York, NY, USA: Association for Computing Machinery, pp. 105–114 (cit. on p. 41).

289. Zhang, R., Xia, B., Li, B., Cao, L., Lai, Y., Zheng, W., Wang, H., and Wang, W. (2018). “State of the Art of Lithium-Ion Battery SOC Estimation for Electrical Vehicles”. In: *Energies* 11.7 (cit. on p. 94).

290. Zhang, W. and Hu, J. (2007). “Low power management for autonomous mobile robots using optimal control”. In: *2007 46th IEEE Conference on Decision and Control*, pp. 5364–5369 (cit. on pp. 50, 51, 91).

291. Zhou, D. and Schwager, M. (2014). “Vector field following for quadrotors using differential flatness”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6567–6572 (cit. on p. 79).

Index