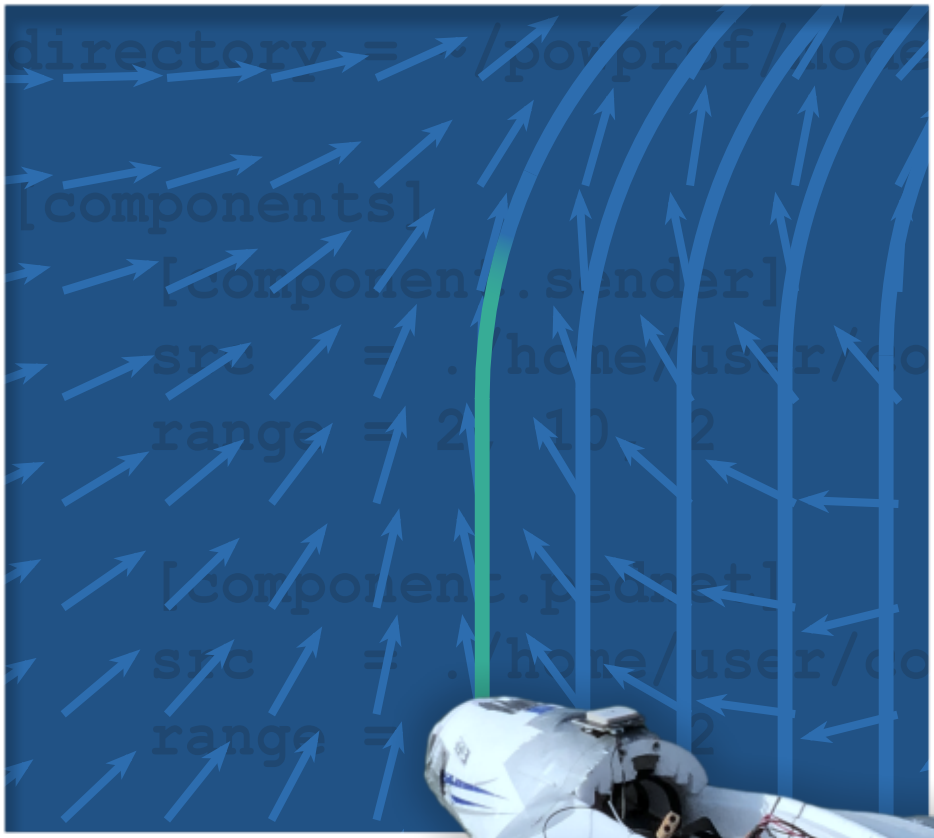
Energy-Aware Coverage Planning and Scheduling for Autonomous Aerial Robots

Adam Seewald



University of Southern Denmark

Energy-Aware Coverage Planning and Scheduling for Autonomous Aerial Robots

Adam Seewald

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Engineering Science

2021

https://doi.org/10.21996/022y-wk34

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

Abstract

This work aims to derive an energy-optimal path and a power-saving schedule for an aerial robot simultaneously, inflight, and under strict energy constraints. Although energy conservations techniques for mobile robots' motion planning and the scheduling for heterogeneous computing hardware carried by these robots have been studied, the close interaction between the two remains mostly unexplored. It is regarded in the available literature that there are classes of mobile robots where it could be advantageous to trade-off reduced resources for a still acceptable level of performance.

Within mobile robots, aerial robots are particularly affected by various energy considerations. Generally, it would be required to land and recharge the battery in case of adverse energy-related events. Therefore, this work emphasizes aerial robots. It derives planning-scheduling energy awareness using optimal control techniques, regression analysis, and differential periodic energy modeling. The future computations energy prediction further derives an automatic profiling and modeling utility that generates overall energy, average power, and battery state of charge models in the function of a software configuration, allowing the integration in a data-flow computational network. The work is demonstrated on the problem of planning coverage in an autonomous precision agriculture use case, where a fixed-wing aerial robot flies over an agricultural field, detects hazards, and communicates the detections with other ground-based actors. The guidance on the coverage path relies on the theory of vector fields, and the overall approach is an algorithm that incorporates the battery, motion, and computations energy modeling along with gradient descent and optimal control.

Planning-scheduling exhibits improved performance mitigating the effect of uncertainty in battery-powered aerial robots against the baseline of an aerial robot flying full coverage, requiring landing in the eventuality of sudden battery defect. Although specific, the approach can be generalized. The computations energy and battery models applied to different domains, whereas the differential periodic energy model, the guidance, and the overall planning-scheduling approach to a broad class of potential autonomous mobile robotics use cases.

Acknowledgements

I am obliged to my advisor Prof. Ulrik Pagh Schultz for his encouragement, patience, and helpful discussions. He was always ready with a very thorough list of insightful suggestions and improvements. My gratitude goes also to my co-advisors, Dr. Héctor García de Marina, who proposed the direction of this work, and Dr. Henrik Skov Midtiby, who shaped the outcomes.

Parts of this work are a result of collaborative efforts. I am indebted to Julius Roeder, Dr. Benjamin Rouxel, and Dr. Clemens Grelck from the Parallel Computing Systems group at the University of Amsterdam and to my colleagues, past and present, from the Unmanned Aerial Systems group at the University of Southern Denmark. I am additionally grateful to TeamPlay project consortium members for stimulating debates on low power computing during various occasions.

Finally, I wish to thank my mother and maternal grandparents who were of great support and indulgence.

Publications

The work is partially based on one journal article.

1. Seewald, A., Schultz, U. P., Ebeid, E., and Midtiby, H. S. (2021a). "Coarse-grained computation-oriented energy modeling for heterogeneous parallel embedded systems". In: *International Journal of Parallel Programming* 49.2. https://adamseewald.cc/short/coarse2019, pp. 136–157 (cit. on pp. 11, 12, 37, 38, 55, 58, 59, 61, 64, 92, 101–109, 128, 129).

Three conference articles.

2. Seewald, A., Schultz, U. P., Roeder, J., Rouxel, B., and Grelck, C. (2019). "Component-based computation-energy modeling for embedded systems". In: *SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH)*. https://adamseewald.cc/short/component2019. ACM, pp. 5–6 (cit. on pp. 11, 12, 59, 101, 106, 129).

3. Seewald, A., García de Marina, H., Midtiby, H. S., and Schultz, U. P. (2020). "Mechanical and computational energy estimation of a fixed-wing drone". In: *4th International Conference on Robotic Computing (IRC)*. https://adamseewald.cc/short/mechanical2020. IEEE, pp. 135–142 (cit. on pp. 11, 13, 66, 92, 101, 108–111, 129).

4. 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: *4th International Conference on Robotic Computing (IRC)*. https://adamseewald.cc/short/energy2020. IEEE, pp. 294–297 (cit. on pp. 9, 11, 12, 59, 92, 101, 107, 129).

Two workshop articles.

5. Seewald, A., Ebeid, E., and Schultz, U. P. (2019). "Dynamic energy modelling for SoC boards: Initial experiments". In: *High-Level Programming for Heterogeneous and Hierarchical Parallel Systems (HLPGPU)*. https://adamseewald.cc/short/dynamic2019, p. 4 (cit. on pp. 35, 60, 101).

6. Seewald, A. (2020). "Beyond traditional energy planning: The weight of computations in planetary exploration". In: *IROS Workshop on Planetary Exploration Robots: Challenges and Opportunities (PlanRobo)*. https://adamseewald.cc/short/beyond2020. ETH Zürich, Department of Mechanical and Process Engineering, p. 3 (cit. on pp. 11, 13, 129, 130).

Additionally, it includes a software.

7. Seewald, A., Schultz, U. P., Ebeid, E., and Midtiby, H. S. (Oct. 2021b). *PowProfiler computations energy modeling tool*. `https://doi.org/10.5281/zenodo.5562457`. Version v1.0.2 (cit. on pp. 11, 12, 59, 128, 129).

A manuscript based on the contents in this work is in preparation at the time of submission.

8. Seewald, A., García de Marina, H., and Schultz, U. P. (n.d.). *Energy-aware dynamic planning algorithm for autonomous UAVs*. `https://adamseewald.cc/short/energy2021`. In preparation (cit. on pp. 11, 13, 101, 115–120, 122–125, 129).

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.
$v_1\big\|_{v_2}$	edge connecting vertices v_1 and v_2.

Abbreviations

BVP	boundary-value problem.
CNN	convolutional neural network.
CPP	coverage path planning.
CSV	comma-separated values.
DFS	dynamic frequency scaling.
DNN	deep neural network.
DVS	dynamic voltage scaling.
ECM	equivalent circuit model.
FCN	fully convolutional network.
FDM	flight dynamics model.
FPS	frames per second.
FSM	finite state machine.
GNSS	global navigation satellite system.
GPGPU	general-purpose GPU.
GPS	global positioning system.
IMU	inertial measurement unit.
ILP	integer linear programming.
MAV	micro aerial vehicle.
MPC	model predictive control.
MSE	mean square error.
M&CE	difference of motion and computations energy.
NFZ	no-flight zone.
NIZ	no-interest zone.
NLP	non linear program.
NPS	new paparazzi simulator.
OCP	optimal control problem.
PMC	performance monitoring counter.
QoS	quality of service.

RL	reinforcement learning.
ROS	robot operating system.
RPV	remotely piloted vehicle.
RSTA	reconnaissance, surveillance, and target acquisition.
SLAM	simultaneous localization and mapping.
SoC	state of charge.
UAS	unmanned aerial system.
UAV	unmanned aerial vehicle.
UGV	unmanned ground vehicle.
VTOL	vertical take-off and landing.

Chapter 1

Introduction

> *"Fixed-wing [aerial robots] [...] 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 have the ability to move (Corke, 2017) and eventually sense and interact with the surrounding environment. They combine and interpret data from multiple components (Mei, Y.-H. Lu, Y. C. Hu, et al., 2006), using various algorithms for perception and planning. For instance, with a motion planning algorithm, a mobile robot converts a human-level plan with sensing into machine-level motion primitives (LaValle, 2006). There are numerous planning algorithms applied to a variety of robots. Some, including motion planning and obstacle avoidance, are mature enough to be transferred onto real-world use cases (Siciliano and Khatib, 2016a). These algorithms often run on energy-demanding heterogeneous computing hardware onboard the mobile robots. They share computational capabilities with other algorithms, enabling various autonomous features. In this context, energy consumption is a crucial aspect (Jaiem et al., 2016). Most mobile robots have strict battery limitations, affecting the computing hardware and influencing the autonomous features, in turn, expected to increase in the foreseeable future (Fisher et al., 2013).

While there are planning approaches that are well covered, others have received little attention from the research community. Planning-scheduling energy awareness is one of such underrepresented approaches in the robotics research literature (Brateman et al., 2006; Lahijanian et al., 2018; Ondrúška et al., 2015; Sudhakar et al., 2020). Here, by planning-scheduling energy awareness, we mean the derivation of an energy-aware path with a power-saving scheduling policy for the computing hardware. Past studies cover extensively one of these topics, but their interactions are largely unexplored (Brateman et al., 2006). Some generate an energy-optimized path, but the computations energy–the energy required by the power demanding tasks running on the computing

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

hardware–might equal the motion in some instances of low-energy mobile robots (Sudhakar et al., 2020). 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 then further 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).

We trade both planning-scheduling altogether and under strict energy constraints. Within motion planning itself, by focusing on a specific category. In autonomous scenarios, it is often required to explore every location in space (E. M. Arkin, Bender, et al., 2001, 2005; Cao et al., 1988; H. Choset, E. Acar, et al., 2000). A problem solved by coverage planning (H. Choset, 2001; Galceran and Carreras, 2013), and so, we show the interactions between planning-scheduling on a coverage planning algorithm, covering a space scheduling high-level task for, e.g., perception, in an elemental region. As a mobile robotics platform, we evaluate the approach with aerial mobile robots in simulations and empirical trials. Although sharing with the broader class of mobile robots stringent battery limitations, they have additional impediments. A conventional setup would require landing to replace or recharge the battery. *Aerial robots are an ideal instance of energy-constrained systems, benefitting the simultaneous planning-scheduling energy awareness*. We will see that we can back such a statement with observations of actual flights failure reductions later with this work in Chapter 6.

There are numerous autonomous use cases involving aerial robots, such as precision agriculture, search and rescue, payload delivery, transportation, and many others. Here, we focus on a precision agriculture use case of an autonomous aerial robot flying over an agricultural field

with little human input. Precision agriculture is indeed 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 aerial robots for preventing damage and ensuring better crop quality (Daponte et al., 2019; Puri et al., 2017). We investigate different physical aerial robotics platforms in this work but derive most results with the Opterra fixed-wing (Hobby, n.d.) adapted for precision agriculture. The aerial robot is in Figure 1.1.

The scope of this chapter is to introduce and motivate the planning-scheduling energy awareness for aerial robots. To this end, Section 1.1 investigates the evolution of aerial robots, and Section 1.2 classifies the modern aerial robotics literature w.r.t. this work. Section 1.3 provides further motivation, objective, and methodology, and Section 1.4 outlines the approach. Sections 1.5–1.6 contain discussions of possible application and overall contribution. Finally, Section 1.7 structures the remaining chapters.

1.1 From UAVs to Modern Aerial Robots

Modern aerial robots are a valuable tool in robotic research and aerospace and have different names in the literature, including unmanned aerial vehicles (UAVs)[i], unmanned aerial systems (UASs)[ii], flying robots, or drones. Usually, UAVs and UASs indicate that these systems are semi-autonomous or operated from the ground, whereas aerial or flying robots have advanced levels of autonomy (Siciliano and Khatib, 2016b). Nevertheless, all these systems have basic autonomous features such as position holding implemented with, e.g., the global navigation satellite system (GNSS), altitude holding with a barometer, and leveling with the inertial measurement unit (IMU).

Unmanned (or uninhabited) flight has more than a century of developments (Siciliano and Khatib, 2016b). The origin of the field, which deals with the design and development of aerial robots, dates back to the first guided missiles. Hewitt-Sperry Automatic Airplane in Figure 1.2, also denominated "flying bomb" and deployed 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. 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)–it used a gyroscope, invented shortly before by Elmer Sperry (Keane and Carr, 2013), like modern aerial robots. The device was mechanically connected to the control surfaces, successfully implementing a control feedback loop (Siciliano and Khatib, 2016b).

In their early days, these first flying machines were referred to as remotely piloted vehicles (RPVs) (Anderson, 2005). Many instances from WWI served military purposes. In the 1950s, the United States used an RPV, the Ryan Firebee, for reconnaissance in Vietnam, and Israel was the first to use an RPV in a combat situation (Anderson, 2005). Other instances include the V-1 flying bomb from 1944 (deployed by the unified armed forces of nazi Germany) and

[i] The term unmanned is sometimes replaced by uninhabited.
[ii] UAS often denotes the entire infrastructure of unmanned flight in the aerospace jargon.

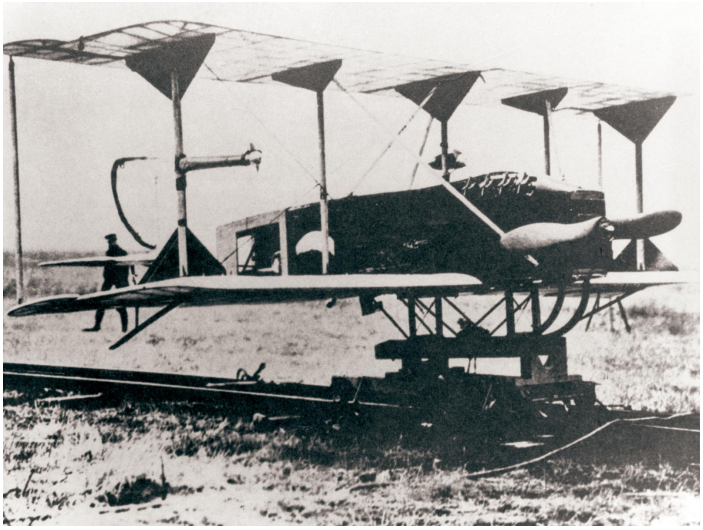


Figure 1.2. The Hewitt-Sperry Automatic Airplane, denominated "flying bomb" and developed during WWI, represents the first instance of a UAV (photo credit: United States Naval Institute).

the Lockheed D-21 from 1962 (deployed by the United States Air Force). The introduction of the global positioning system (GPS) at the end of 1970 increased recent developments with applications such as surveillance. Integration with cameras and other sensors grounded further applications (Siciliano and Khatib, 2016b), introducing modern aerial robots.

Recent developments include many civilian applications (González-Jorge et al., 2017). Aerial robots are used increasingly 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 reconnaissance, surveillance, and target acquisition (RSTA) and do not require advanced autonomy; precision agriculture, transportation, and payload delivery utilize a certain extent of computational intelligence (Siciliano and Khatib, 2016b). They handle unexplored terrain with little interaction, contrary to the past human-operated UAVs (Siciliano and Khatib, 2016b). Instances autonomously adapt and possibly interact in a wide variety of environmental conditions.

In summary, aerial robots have a recent past. Some initial experiments came shortly after the first heavier-than-air manned and powered flight. These often served military purposes, whereas modern aerial robots fly in a broad range of civilian applications. Aerial robots are to grow significantly in numerous areas in and out 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 in the first powered, controlled flight on another planet on April 19, 2021. Aerial

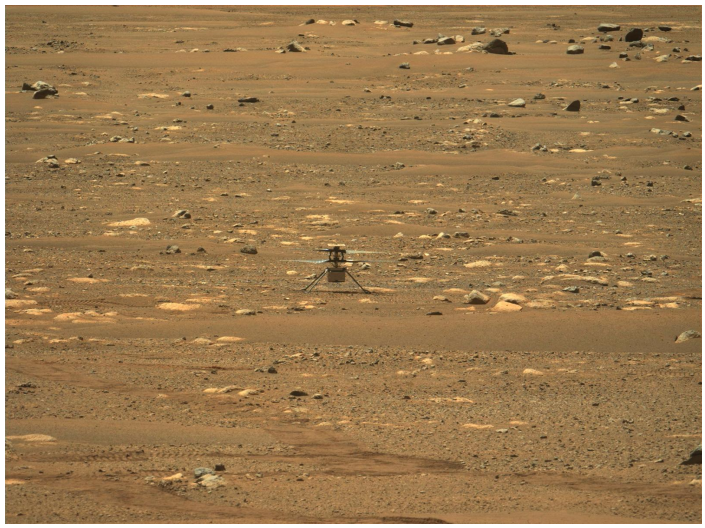


Figure 1.3. NASA's Ingenuity Mars Helicopter. A rotary-wing coaxial aerial robot, achieving the first powered, controlled flight on another planet on April 19, 2021 (photo credit: NASA Jet Propulsion Laboratory).

robots for planetary exploration are to be deployed in future explorations endeavors, for instance, to study Saturn's moon Titan (Voosen, 2019).

1.2 Common Classes of Aerial Robots

There are different types of aerial robots in the robotics literature. We briefly investigate the most studied classes w.r.t. the planning-scheduling energy awareness in this work. The two most generic classes are heavier- and lighter-than-air aerial robots. Heavier-than-air compromise fixed- and rotary-wings (Siciliano and Khatib, 2016b), and some recent contributions in bio-inspired robotics investigate flapping-wings aerial robots (Floreano and Wood, 2015).

Rotary-wing aerial robots are highly maneuverable and suited for stationary vertical flight or hovering (Siciliano and Khatib, 2016b) where rotors provide the lift and maneuvering. They compromise multirotors (equipped with multiple rotors such as quadrotors or quadcopters, hexacopters, and octocopters), conventional helicopters (with one main and one tail rotor), and coaxes (with counter-rotating coaxial rotors) (Corke, 2017). Examples of quadrotors are DJI Mavic Mini in (h) and DJI Phantom 4 in (j) in Figure 1.5. DJI Agras T16 in (g) and DJI Matrice 600 in (f) are hexacopters.

Fixed-wing aerial robots share the same motion principle with an aircraft where wings provide the lift, control surfaces maneuvering, and propellers the forward thrust (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

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).

flapping-wings are DelFly II in (k) (Clercq et al., 2009; Groen et al., 2010; Percin et al., 2012) and Nano-Hummingbird in (l).

Instances of lighter-than-air aerial robotss are blimps (or non-rigid airships). They usually rely on gas, e.g., 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 in Figure 1.4. Blimps are similar to balloons but provide some maneuverability against controlling merely the altitude (Colombatti et al., 2011).

Other classifications are, e.g., micro aerial vehicles (MAVs), vertical take-off and landing (VTOLs) aerial robots, and others. The former are aerial robots with all dimensions lower than 15 centimeters. The latter fly in a fixed-wing configuration except for 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, lighter-than-air aerial robots the least. Nonetheless, these 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, 2016b). The energy requirements are critical for all aerial robots, but the difference of motion and computations energy (M&CE) varies greatly. It is highest in rotary-wings, lowest in lighter-than-air aerial robots in Figure 1.5, and relatively comparable for fixed-wings. Planning-scheduling would thus rely on both for this latter group energy-wise while relying almost exclusively on planning for rotary-wing aerial robots. In the former, M&CE is close to zero. In the latter, it is usually high except for energy-optimized designs such as some rotary-wing MAVs. Hypothetically, in lighter-than-

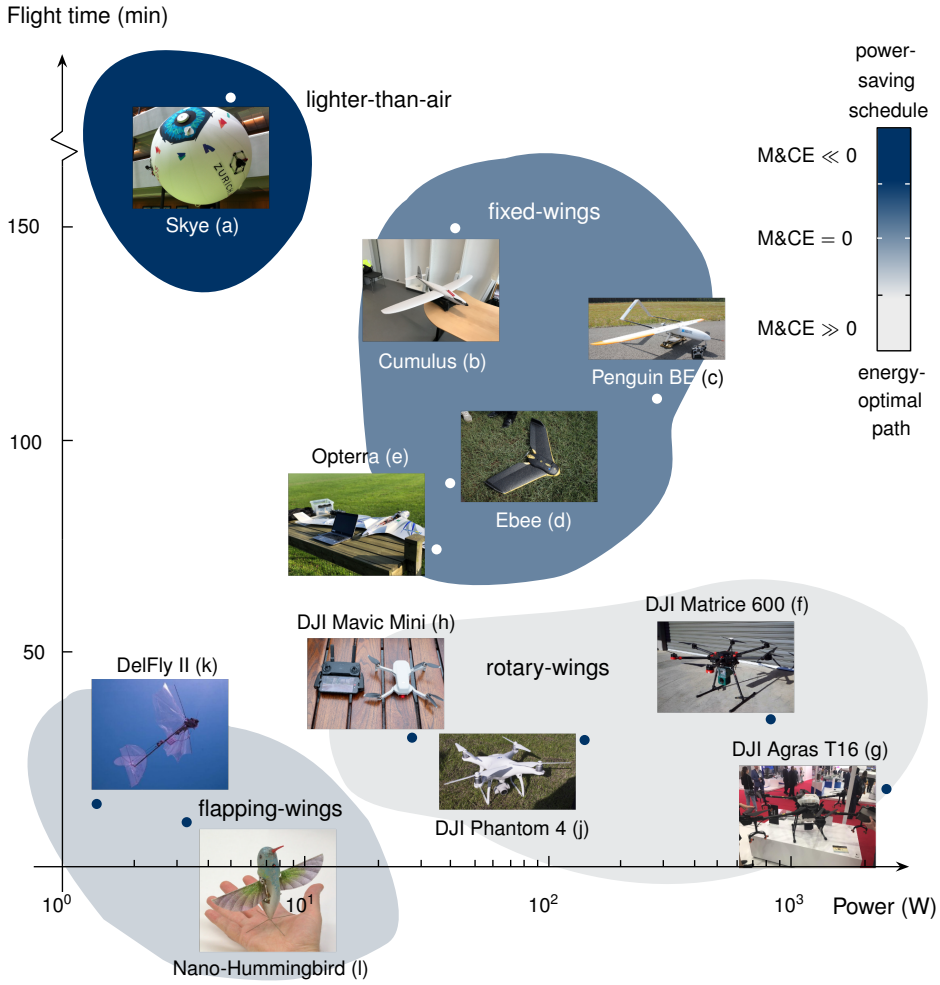


Figure 1.5. Different aerial robots in relation to the power, flight time, and M&CE with a hypothetical fixed costs for computations energy. The power is expressed using a logarithmic scale. Heavier-than-air aerial robots (b)–(l) include flapping-wings (k), (l) with a negative M&CE (computations energy is predominant), and rotary-wings with a positive M&CE (f), (g) (motion energy is predominant). Smaller rotary-wings have an M&CE closer to zero (h), (j). In general, rotary-wings have a short flight time. Fixed-wings (b)–(e) have longer flight time and M&CE close to zero (computations and motion energies are comparable). Lighter-than-air aerial robots (a) have a hypothetically 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).

air aerial robots, M&CE might be negative. The planning-scheduling would rely heavily on scheduling.

The classification with M&CE serves the scope of our work. There are similar efforts to group aerial robots by size and propulsion technology (Cabreira, Brisolara, et al., 2019; Hoffer et al., 2014), but other classifications are possible, e.g., categorization by altitude (Watts et al., 2012).

1.3 Motivation

Use cases involving aerial robots ranging from remote sensing to payload delivery have strict battery constraints. It is a common problem of most mobile robots (Mei, Y.-H. Lu, C. Lee, et al., 2006), yet, aerial robots are particularly affected. Indeed, the availability of the power source might influence their autonomy. An instance is an aerial robot autonomously inspecting a given space by, e.g., detecting ground patterns and notifying other ground-based actors in a precision agriculture use case. For such and many others, aerial robots often rely on computing hardware along with a microcontroller (Andrew et al., 2019; Dharmadhikari et al., 2020; Holper et al., 2017; Papachristos et al., 2015). Computing hardware provides autonomous capabilities and planning, whereas a microcontroller runs motion primitives by directly interfacing with actuators (such as servos for the control surfaces) and motors (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005).

1.3.1 Planning-scheduling energy awareness

It is uncommon to find a ready-to-use solution for simultaneous planning the path and scheduling the computations of these systems in an energy-aware fashion (Brateman et al., 2006; Sudhakar et al., 2020). It might be potentially advantageous to schedule the computations on the computing hardware and simultaneously to plan the path rather than solving these two separately (Lahijanian et al., 2018; Ondrúška et al., 2015) by, e.g., optimizations in terms of the battery state of charge (SoC).

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 headwind (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 and compensate for the increment in motion energy. During the same flight, the wind direction might suddenly change. The fixed-wing craft, now flying downwind, 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, for instance, shortening the path. Planning-scheduling altogether in all these cases is, perhaps, 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 the most from simultaneous planning-scheduling. They have an M&CE close to zero and a long flight time. Although some

rotary-wings have M&CE also close to zero, their flight time is generally short. Flapping-wings and lighter-than-air aerial robots have a negative M&CE, requiring less energy for the motion. Furthermore, practical implementations of these latter categories might be carrying only a microcontroller (Groen et al., 2010).

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 others 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 use case, doing coverage planning while detecting ground hazards. These include livestock, humans, and vehicles (Zamanakos et al., 2020), potentially obstructing operations of a ground-based mobile robot for, e.g., harvesting. In abstract terms, the field is a polygon in Figure 1.6. The aerial robot detects patterns using a convolutional neural network (CNN), flying over the polygon. It further communicates the position of the detections to other, ground-based actors. Detecting the patterns usually involves heterogeneous computing elements (GPU and/or multi-core CPU) and consumes a significant amount of energy contrary to, e.g., communication.

Figure 1.6. The coverage problem in the precision agriculture use case. The aerial robot covers an agricultural field that forms a polygon (blue/transparent area in the frame), executing tasks as part of its autonomous operations (photo credit: Amit Ferencz Appel).

We are interested in the energy optimization of the path and schedule in-flight and under uncertainty (such as from atmospheric interferences). Such planning-scheduling finds optimal trade-offs between the path, computations, and energy requirements. Current generic solutions for, e.g., aerial robots coverage planning, do not investigate the two aspects simultaneously, 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) from existing algorithms (H. Choset, 2001; Galceran and Carreras, 2013), with instances including popular flight controllers (Paparazzi,

n.d.[b]; PX4, n.d.). This state of practice has prompted us to investigate the possible interaction between planning-scheduling applied to coverage path planning (CPP) for autonomous aerial robots. In the remainder, we will gradually build an approach that plans and schedules altogether while the aerial robot flies and its batteries drain under various conditions.

1.3.3 Methodology

Initial iterations of our work relied on case study research, a widely used qualitative research method (Darke et al., 1998). Here, we qualitatively investigated collected data and observed phenomena of the aerial robot flying early instances of the agricultural use case to derive methodologies and implications at subsequent stages. We focused on an in-depth analysis of the planning-scheduling interactions, analyzing its energy implications and the available literature.

The grounding of further research directions involved research questions formulation: *(a)* how to model the computations energy to predict the impact of any possible schedule on the computing hardware's power? *(b)* How to model the motion energy to predict the impact of any possible variation of motion on the aerial robot's energy? *(c)* How to merge these two points with a cohesive model? *(d)* How to derive an optimal configuration of the schedule and motion?

We addressed these questions using the active research method cycle (Susman and Evered, 1978): beginning from each question, we derived a research action, deployed the action with a rigorous methodology, evaluated the results, and under unsatisfactory outcomes, adjusted the methodology. Multiple iterations of the cycle led to the energy-aware coverage planning and scheduling for autonomous robots in this work. We addressed the research question *(a)* in Sections 4.1–4.2, *(b)* in Sections 4.2–4.3, *(c)* in Section 4.3, and *(d)* in Chapter 5.

1.4 Outline of the Approach

For planning-scheduling together, we need some information on the intended use for both the path and computations. Path-wise, these include data on the coverage area, such as a description of an equivalent polygon. Chapter 2 details ways of defining the problem with its building constructs (plan, stages, parameters, paths, constraints, triggering points). From the polygon, a coverage algorithm in Section 5.2 generates a plan composed of multiple stages, later eventually replanned by a replanning algorithm in case of, e.g., sudden battery drops in Section 5.3. At each stage, the aerial robot flies a path and executes some computations. The concept of different stages serves the purposes of modeling complex paths, e.g., multiple circles and lines form the overall coverage. The robot switches between the paths in the proximity of specific triggering points. The plan further 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. The approach guides the aerial robot with a vector field-based gradient descent algorithm in Section 5.1.

Computation-wise, we need an approach to specify the computations and quantify their energy contribution. Chapter 4 covers these aspects, as well as the motion and battery energy models. The approach relies on `powprofiler` (Seewald, Schultz, Ebeid, et al., 2021b), a tool that we introduce in Section 4.1, part of our early studies (Seewald, Schultz, Ebeid, et al., 2021a; Seewald, Schultz, Roeder, et al., 2019). The tool models the power, energy, and battery SoC of the computations, and is embedded in the future energy estimations of the aerial robot. To this end, we empirically derive and formally prove a periodic differential energy model that accounts for the uncertainty. Based on Fourier analysis, the model exploits empirical observations to include path and computations alterations. Periodicity is due to the periodic patterns in the coverage plan. Indeed, in the coverage, as well as some other autonomous scenarios, the mobile robot often iterates over a set of tasks and paths (Seewald, García de Marina, Midtiby, et al., 2020; Seewald, García de Marina, and Schultz, n.d.). Given that the plan is periodic, we expect the energy consumption to evolve (approximately) periodically. Chapter 6 will back the statement with both experimental and realistic simulated findings.

Once we have a plan and all the components to model the energy and battery, in Chapter 5, we can (re)plan-schedule the coverage online in-flight, aided by modern optimal control techniques, i.e., state estimation and model predictive control (MPC) (Rawlings et al., 2017; Simon, 2006). The control is data-driven: energy sensor data estimates some coefficients of the model to predict the future energy consumption with uncertainty while obeying the 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

In the remainder of this work, we focus on the precision agriculture use case where we plan the coverage and schedule hazards detection. Nonetheless, the approach works with other potential use cases. Earlier collaborations between consortium members of the TeamPlay project (TeamPlay Consortium, 2019b), funding this work has led to a search and rescue use case, where an aerial robot detects vessels in an offshore area, eventually planning-scheduling of the search pattern and the detection rates. We briefly detailed the scenario in our earlier study (Seewald, Schultz, Ebeid, et al., 2021a), which has been since extended with a deadline guarantee scheduling policy (Rouxel et al., 2020). We have then attempted other use cases, e.g., mapping in the planetary exploration context, where we hypothesized the possibility of scheduling navigation (Seewald, 2020). Indeed an earlier study proposes a similar technique, scheduling perception (Ondrúška et al., 2015), perhaps further motivating our analysis. Simulated use cases are also possible. In an early study (Zamanakos et al., 2020) of a use case in agricultural safety, we followed a simulated agricultural vehicle by varying the tracking algorithm.

There are multiple possibilities to apply our approach to a broad range of real-world and simulated use cases, arising in many different fields in and outside of basic robotics research. Generally, the planning-scheduling energy awareness in this work applies to modern aerial robots with a certain degree of autonomy, whereby the robot performs at least a predefined set of tasks

over a given space. Indeed most of the guidance in Chapter 5 works well for aerial robots, yet, one can adapt our work to other mobile robots with energy constraints. We discuss applications out of the aerial robotics domain further in Chapter 7. We expect the approach to be most relevant to energy-efficient mobile robots, with the best outcomes for M&CE close to zero (the motion and computations energy contributions are similar). Such conclusion is shared with other studies researching planning-scheduling energy awareness (Brateman et al., 2006; Lahijanian et al., 2018; Mei, Y.-H. Lu, Y. C. Hu, et al., 2005; Ondrúška et al., 2015; Sudhakar et al., 2020), which we discuss further in Chapter 3.

1.6 Contribution

There are some approaches to merging planning-scheduling in different robotics use cases in the literature (Brateman et al., 2006; Lahijanian et al., 2018; Mei, Y.-H. Lu, Y. C. Hu, et al., 2005; Mei, Y.-H. Lu, Y. C. Hu, et al., 2006; Ondrúška et al., 2015; Sadrpour et al., 2013a,c; Sudhakar et al., 2020; W. Zhang and J. Hu, 2007), yet the research area remains mostly unexplored (Brateman et al., 2006; Sudhakar et al., 2020). Our work contributes with the past relevant literature to this existing research gap, proposing coverage planning and scheduling for autonomous aerial robots under stringent energy constraints. Specifically, we derive an energy model for the heterogeneous computing hardware, alongside modeling the energy contribution of the motion. We use the model in an optimal control technique similarly to past literature (Brateman et al., 2006; Lahijanian et al., 2018; Ondrúška et al., 2015; W. Zhang and J. Hu, 2007), but fill the gap further with accurate energy modeling of computations and propose an actual implementation of an MPC-based algorithm. We provide a power-saving scheduling policy as opposed to simply varying the frequency of the computing hardware (Brateman et al., 2006; W. Zhang and J. Hu, 2007), and notably, our approach runs online and is dynamic, planning-scheduling in flight rather than deriving static plans-schedules (Lahijanian et al., 2018). It provides additional modeling rigor and incorporates battery-aware optimization as opposed to merely considering the overall energy expenditure (Sudhakar et al., 2020). We vary both the path and the schedule the aerial robot flies and runs, contrary to others that vary just one of the aspects while analyzing the energy implications of the remaining (Ondrúška et al., 2015).

Aside from past relevant studies, literature on topics related to computations energy modeling, battery modeling, motion planning, and aerial planning, our contribution builds from some of our past and forthcoming studies. Our computational energy modeling relies on the methodology in our early study (Seewald, Schultz, Ebeid, et al., 2021a), which presented the `powprofiler` tool (Seewald, Schultz, Ebeid, et al., 2021b) for future energy predictions of heterogeneous computing hardware. We proposed an extension of `powprofiler` with a component-based energy modeling approach to abstract per-component energy in a dataflow computational network in another study (Seewald, Schultz, Roeder, et al., 2019) that we later integrated into a Robot Operating System (ROS) (Quigley et al., 2009) set-up in the following work (Zamanakos et al., 2020). Here we scheduled the simulated tracking of agricultural vehicles. The motion energy model we propose in Chapter 4 relies on empirical observations of the Opterra fixed-wing aerial robot flying

the agricultural scenario we introduced earlier in our work (Seewald, García de Marina, Midtiby, et al., 2020). We hypothesized the approach on different robots in our brief early study (Seewald, 2020). Finally, we plan to detail the planning-scheduling energy-awareness for CPP in precision agriculture in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.), relying on all the concepts in this work.

1.7 Reading Guide

In this chapter, we introduced energy-aware coverage planning and scheduling for autonomous aerial robots, the field of aerial robotics, motivated our work, and provided its objective and methodology. We then outlined the approach, proposed the applications, and discussed the contribution.

The remaining chapters have the following structure. Formal definitions of the problem and the basic constructs are subjects of Chapter 2. These include the plan, stages, parameters, paths, constraints, and triggering points. The chapter then contains a plan example extended in the remainder of the work. Chapter 3 discusses the relevant literature for computations energy and battery modeling, motion and aerial planning, and planning of computations with motion, relating each study to our approach. Planning-scheduling energy awareness requires accurate future energy predictions, and so, Chapter 4 derives computations and motion energies and battery models. It details the `powprofiler` tool for computations energy modeling and the differential periodic energy and battery models. Chapter 5 proposes a set of algorithms for guidance, CPP, and energy-aware coverage replanning and scheduling. The guidance directs the aerial robot physically on the coverage, CPP provides a plan, covering each point in space, and energy-aware replanning replans the plan energy-wise in-flight. Chapter 6 describes the experimental setup and results. Finally, Chapter 7 summarizes the work and concludes with the outcomes and future perspectives.

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

The scope of this chapter is to provide building blocks and formal definitions paving the foundation of the remaining chapters. Here, we derive the planning and coverage problems we solve with this work. The coverage problem is the problem of finding a path, covering all the points in a given space (H. Choset, 2001; Galceran and Carreras, 2013), for instance, the agricultural field in Section 1.3. The coverage path with computations forms the plan. The planning problem is the problem of replanning the plan. In the context of this work, it is replanned energy-wise in the eventuality of energy constraints dissatisfaction and whenever the uncertainty affects the flight unexpectedly. Both the problems are formulated in Section 2.5. The formulations rely on plan-specific constructs in Sections 2.1–2.4, with the concepts of computations, motion, and computations and motion energies. Further formalities include the difference between computations and motion energies (M&CE) encountered in Section 1.2. We illustrate the problem with an example of the precision agriculture use case in Section 2.6.

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. Chapter 3 discusses past approaches to solve the problems. Chapter 4 exploits the plan characteristics to derive computations and motion energy and battery models. Chapter 5 provides a solution to the planning problem with modern optimal control techniques and the coverage problem using a coverage path planning (CPP) algorithm. It further provides ways to guide the aerial robot with an algorithm based on vector fields, using the plan's building blocks. Chapter 6 shows the problems in realistic and simulated experiments.

2.1 Definitions of computations and motion

This section contains definitions of computations and motion, their respective energies, and M&CE.

Definition 2.1.1: Computations/motion. *Computations* are energy-demanding computational tasks. The aerial robot runs the computations on heterogeneous computing hardware that interfaces to microcontrollers.

Motion is the act of the aerial robot moving in the surrounding environment. For this purpose, it runs primitives on microcontrollers that interface to actuators, motors, and other components.

Autonomous capabilities are often achieved by interconnecting heterogeneous computing hardware and microcontrollers. For the computations, we assume that the heterogeneous computing hardware runs a parametrized schedule. For instance, for the detections in the precision agriculture use case in Section 1.3, a parameter is the frames per second (FPS) rate. Similarly, for the motion, we assume that the robot travels parametrized paths. In the use case, a parameter changes the coverage quality.

Definition 2.1.2: Computations/motion and overall energy. Given a path parametrized by ρ values $c_i^{\rho} := \{c_{i,1}, c_{i,2}, \ldots, c_{i,\rho}\}$, the *motion energy* is the energy spent by the aerial robot while moving on the path.

Given a schedule parametrized by σ values $c_i^{\sigma} := \{c_{i,\rho+1}, c_{i,\rho+2}, \ldots, c_{i,\rho+\sigma}\}$, 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. It uses measures in watts for instantaneous or average, joules for overall energies. Section 2.3 details the meaning of *parametrized paths and computations*.

Definition 2.1.3: Difference of motion and computations energy (M&CE). Given measures of average motion and computations energies, $\mathcal{M\&CE}$ is the measure of their difference.

M&CE is measured in watts and quantifies which of the two energy components is predominant. For M&CE greater than zero, the motion energy dominates over the computations. The planning-scheduling energy awareness accentuates on an energy-efficient path (e.g., rotary-wing aerial robots (f) and (g) in Figure 1.5). On the contrary, for M&CE lower than zero, the computations energy dominates. The planning-scheduling energy awareness focus on a power-saving schedule (e.g., lighter-than-air aerial robot (a)). For M&CE close to zero, both energy components are important energy-wise. The planning-scheduling energy awareness trades both an energy-efficient path and a power-saving schedule to a similar extent (e.g., fixed-wing aerial robots (b)–(e)).

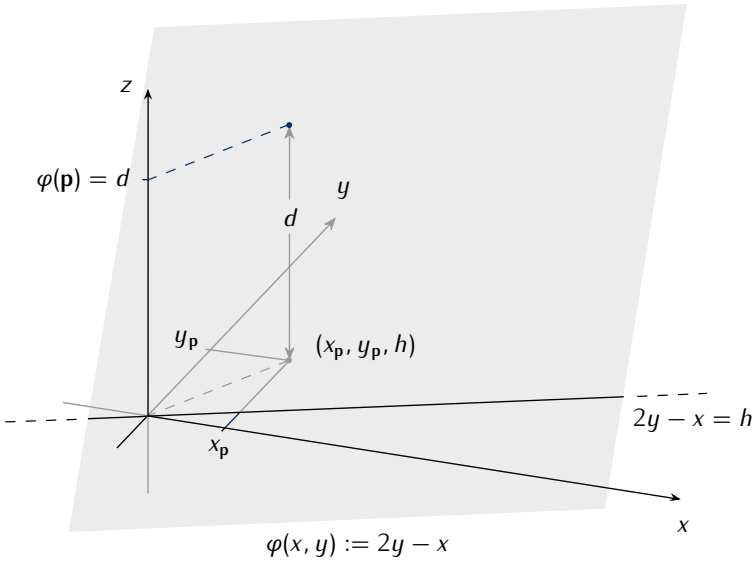


Figure 2.1. The path function is a mathematical function $\varphi(\mathbf{p}(t)) = h$ that represents a line at an altitude h. A generic point $\mathbf{p}$ in 2D space intersects the plane formed by φ at a value d of the z-axis.

2.2 Definition of path functions

A succession of multiple mathematical functions $\varphi_1, \varphi_2, \dots$ termed path functions forms the coverage path that the aerial robot follows. They are expressed in 2D space at an altitude $h \in \mathbb{R}$ for an inertial navigation frame $\mathcal{O}_W$. Here, we assume the aerial robot flies over a given elemental region for the coverage without altering the altitude[i].

Definition 2.2.1: Path functions. $\varphi_i : \mathbb{R}^2 \to \mathbb{R}$, $\forall i \in \{1, 2, \dots\}$ are *path functions*, forming 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 at an altitude h and are continuous and twice differentiable.

We use this notation to guide the aerial robot with the theory of vector fields. For instance, one can define a line as a path function with

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

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

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

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.

[i] We discuss later in terms of future directions a generalization of the path functions in 3D space for, e.g., urban monitoring use cases.

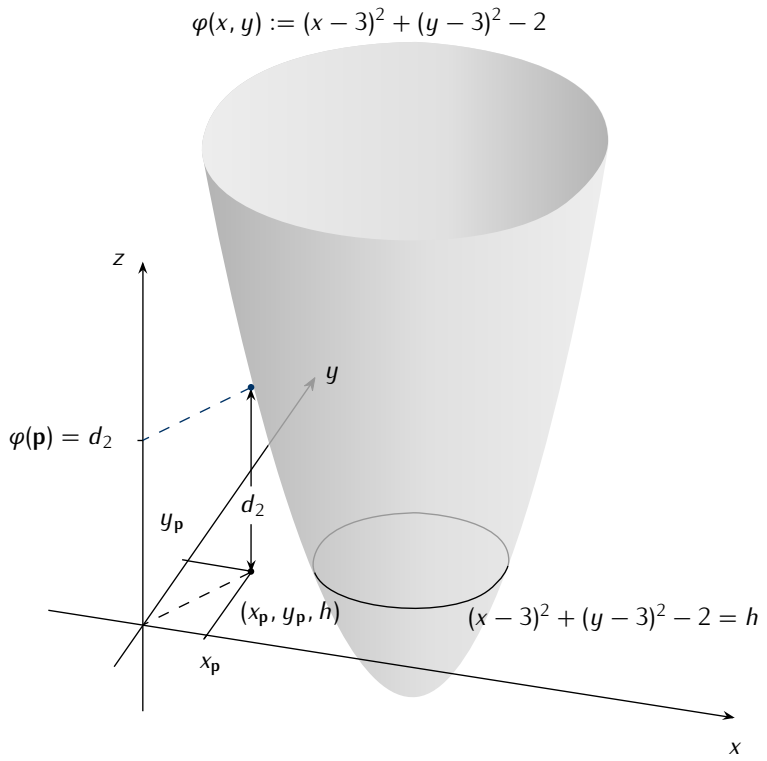


Figure 2.2. The path function now represents a circle at an altitude h. $\mathbf{p}$ intersects the cone formed by φ at a value d_2 of the z-axis.

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. Figure 2.2 illustrates the circle path function for x_c, y_c both three, $\sqrt{r}$ 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–to form the coverage path. However, one can define any mathematical function, with the only requirement being continuity and twice differentiability. The first derivative is a requirement for the vector field, the second derivative of the control action. Chapter 5 details the concept further.

2.3 Definitions of stages and triggering points

The plan has several stages $i = \{1, 2, \dots\}$, and we assume that at each stage, the aerial robot runs a schedule and travels a path function φ_i using a parameters set c_i. Parameters are variable values to replan the path and computations, influencing the computations/motion energy in Definition 2.1.2. The path parameters are real-, the computations parameters integer-valued ($\mathbb{R}$, $\mathbb{Z}$).

The notation $c_{i,j}$ denotes the jth parameter of the ith parameters set

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

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

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

expressing physical bounds of the computing hardware and the aerial robot (e.g., it is not possible to compute more than the capabilities of the computing hardware, turn narrower than the minimum turning radius of the aerial robot, and so on).

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 $c_i^\rho := \{c_{i,1}, c_{i,2}, \ldots, c_{i,\rho}\}$, and the computations scheduled with σ computations parameters $c_i^\sigma := \{c_{i,\rho+1}, c_{i,\rho+2}, \ldots, c_{i,\rho+\sigma}\}$.

Returning to Section 2.1, "*path parametrized by ρ values*" indicates that φ_i is enhanced with c_i^ρ. The function $\varphi_i : \mathbb{R}^2 \times \mathbb{R}^\rho \to \mathbb{R}$ 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, "*computations parametrized by σ values*" indicates that c_i^σ is the computations schedule. These parameters also serve for energy alteration (e.g., decreasing the granularity of a given computation lowers the power). We discuss the alteration of the energy with path and computations parameters further in Sections 4.3.2–4.3.3.

The stage is a set that contains the path and path and computations parameters.

Definition 2.3.1: Stage. For a generic point $\mathbf{p}(t)$, the ith *stage* Γ_i at time instant t of a plan Γ is

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

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.

The next section clarifies why the stage contains the generic point $\mathbf{p}(t)$.

It is possible to 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 simplified

$$\Gamma_i := \{\varphi_i(\mathbf{p}(t), c_i^\rho), c_i^\sigma \mid \forall j \in [\rho + \sigma]_{>0},\ c_{i,j} \in \mathcal{U}_i(c_{i,j})\}. \tag{2.7}$$

Specific points termed triggering points $\mathbf{p}_{\Gamma_i}$ allow the transition between stages.

Definition 2.3.2: Triggering and final points. The *triggering point* is the point $\mathbf{p}_{\Gamma_i}$ that allows the transition between stages. The *final point* is the last triggering point $\mathbf{p}_{\Gamma_l}$ relative to the last stage Γ_l.

As soon as the aerial robot reaches the proximity of these points, it switches to the next stage

$$\left\|\mathbf{p}(t) - \mathbf{p}_{\Gamma_i}\right\| < \varepsilon_i, \tag{2.8}$$

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

2.4 Definition of plan

The energy model in Section 4.3 exploits 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, which one can observe in the precision agriculture use case in Section 1.3. An exhaustive 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. In literature, the basic pattern is often termed "motion" and the most common is, e.g., boustrophedon motion (Cabreira, Brisolara, et al., 2019; H. Choset, 2001; H. M. Choset et al., 2005; Galceran and Carreras, 2013). One can then define the plan just as a set of stages, triggering points, a final point, and a shift that specifies how the constructs (except the final point) shifts in space every period. The concept of primitive paths in the following simplifies the planning to this latter case.

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 $l \in \mathbb{Z}_{>0}$. If the plan is built from the primitive paths, it means that n in Definition 2.4.1 respects

$$n < l,\ l/n \in \mathbb{Z}, \tag{2.9}$$

where n is a multiplier of l.

One can then write the remaining paths from the n primitive paths as φ_{n+1}, $\varphi_{n+2}, \ldots, \varphi_{n+n}, \ldots, \varphi_l$, or more generally $\varphi_{(i-1)n+1}, \varphi_{(i-1)n+2}, \ldots, \varphi_{(i-1)n+n}$, $\forall i \in [l/n - 1]_{>0}$. Or similarly

$$\varphi_{(i-1)n+j}(\mathbf{p} + (i-1)\mathbf{d}, c_1^\rho) - \varphi_{in+j}(\mathbf{p} + i\mathbf{d}, c_1^\rho) = e_j, \tag{2.10}$$

for a given shift $\mathbf{d}$, initial point $\mathbf{p}$, and initial value of path parameters c_1^ρ. Equation (2.10) 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 $\mathbf{p}_{\Gamma_l}$. 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.

Formally, the plan is a finite state machine that exploits the constructs of path functions in Definition 2.2.1 and stages and triggering points in Definitions 2.3.1–2.3.2.

Definition 2.4.2: Plan. For a generic point $\mathbf{p}(t)$, the *plan* is a finite state machine (FSM) Γ, where the state-transition function $s : \bigcup_i \Gamma_i \times \mathbb{R}^2 \rightarrow \bigcup_i \Gamma_i$ maps a stage and a point to the next stage

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

The value ε_i in Definition 2.4.2 is the same ε_i in Equation (2.8).

A concept that we use in the remainder of this work, and particularly in energy modeling in Chapter 4, is the 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 T 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 or replanning. 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 (2.10). If there is no such value (e.g., when the plan is composed of only one stage or the plan is aperiodic), the period can be determined empirically from energy data.

We illustrate the plan, stage, triggering, and final points definitions in Figures 2.3–2.5.

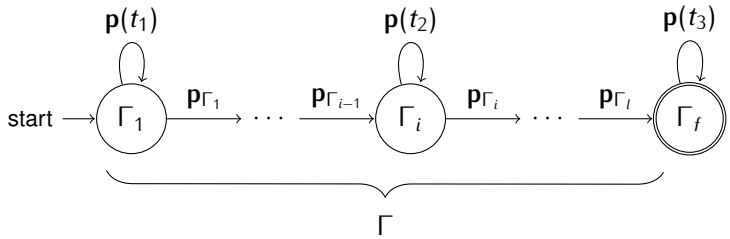


Figure 2.3. Definition of a plan Γ as an FSM. Each state is a stage Γ_i, the transition happens in the proximity of specific points called triggering points $\mathbf{p}_{\Gamma_i}$. The accepting stage Γ_f indicates the termination of the plan.

Figure 2.3 illustrates 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

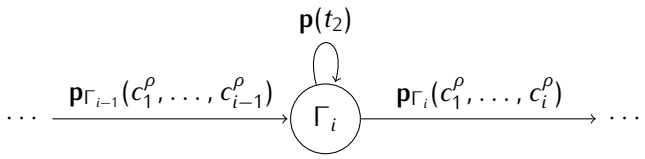


Figure 2.4. Detail of the stage Γ_i in the FSM. The triggering points used to transition between states are expressed in the function of the last and/or previous path parameters.

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). Figure 2.4 illustrates that it is possible to express the basic constructs–such as path functions and triggering points–in the function of the ith path parameters c_i^ρ, or any previous path 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.

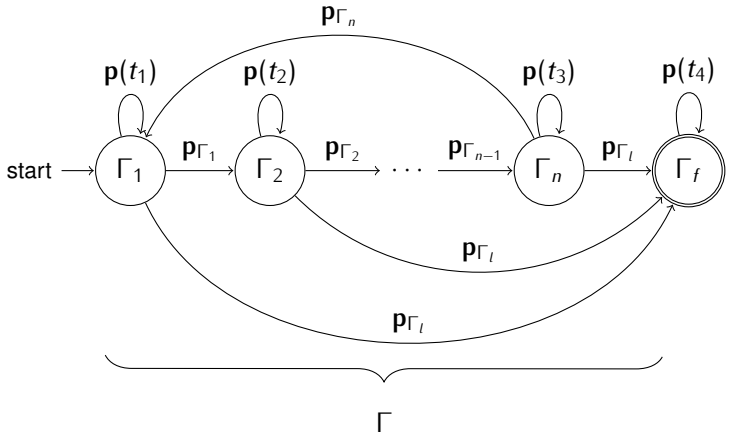


Figure 2.5. Definition of a plan Γ with periodic patterns. Stages $\Gamma_1, \Gamma_2, \ldots, \Gamma_n$ containing primitive paths $\varphi_1, \varphi_2, \ldots, \varphi_n$ are iterated with a shift $\mathbf{d}$.

Figure 2.5 illustrates 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.

2.5 Problem Statement

The overall strategy for energy-aware coverage planning and scheduling for autonomous aerial robots is split into two sub-problems: the planning and coverage problems.

2.5.1 Planning problem

The planning problem is the problem of finding the optimal configurations of the parameters using some criteria. Within planning-scheduling energy awareness in the robotics research literature, it is often solved with optimal control techniques (Brateman et al., 2006; Lahijanian et al., 2018; Ondrúška et al., 2015; W. Zhang and J. Hu, 2007). Here, we focus on energy criteria, such as the battery constraints. Solving the planning problem by finding the optimal configurations with 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. Chapter 7 discusses the eventuality of solving the planning problem with different costs.

Problem 2.5.1: Planning problem. Consider an initial plan Γ in Definition 2.4.2. It is either composed of l stages or n stages and a shift $\mathbf{d}$. The *planning problem* is the problem of finding the optimal configuration of *path* and *computations parameters* $c_i\, \forall i \in [l]_{>0}$ or $\forall i \in \{1, 2, \dots\}$ under energy constraints and uncertainty at each time step t.

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 $\mathbf{p}_l$. Given the optimal configuration, we are further interested in the guidance to the path and the scheduling of the computations.

2.5.2 Coverage problem

The coverage problem is the problem of finding a route that covers all the points in a given space, avoiding the obstacles. In the literature, it is under a branch of motion planning termed CPP (H. Choset, 2001; H. Choset and Pignon, 1998; Galceran and Carreras, 2013). Some approaches ensure the completeness of the coverage and include algorithms optimized for mobile robots. Here, we focus on these algorithms, including specific constraints such as turning radius and the variability of the coverage. Furthermore, in the aerial robotics context, obstacles can be seen as areas that the aerial robot should not consider (Cabreira, Brisolara, et al., 2019). Since there are no obstacles in our use case to avoid physically–e.g., airports, tall buildings, etc.–we use the term no-interest zones (NIZs). It depicts areas on the ground where the aerial robot can potentially fly but should not perform any computation, e.g., adjacent agricultural lots or supporting infrastructure laying on the ground. We assume that sets of vertices describe the space and NIZs. In this context, the coverage problem is the problem of covering the space while, e.g., overflying NIZs without computing. Chapter 7 discusses a generalization of NIZs to physical obstacles.

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 NIZs $o := \{o_1 := \{o_{1,1}, o_{1,2}, \dots\}, o_2 := \{o_{2,1}, o_{2,2}, \dots\}, \dots\}$ where each vertex $v_i := (x_{v_i}, y_{v_i}), \forall i \in |v|$, $o_{j,k} := (x_{o_{j,k}}, y_{o_{j,k}})$, $\forall j \in |o|, k \in |o_j|$ is a point w.r.t. $\mathcal{O}_W$. Let $\underline{r} \in \mathbb{R}_{\geq 0}$, the minimum turning radius, and $\mathbf{p}(t_0)$, the starting point at the time instant t_0, be given. The *coverage problem* is the problem of finding a plan Γ that covers v while

avoiding computations over o[ii], starting from $\mathbf{p}(t_0)$ with a turning radius greater or equal to $\underline{r} \in \mathbb{R}_{\geq 0}$.

Chapter 5 proposes two algorithms for the problems in this section. A first algorithm generates the coverage path, and another algorithm replans the plan in time. The replanning is energy-aware. The algorithm outputs the best trajectory of the path and computations alterations for an aerial robot with varying battery and atmospheric conditions.

2.6 Precision agriculture use case

In this section, we discuss an example of a plan for the Opterra fixed-wing aerial robot in the precision agriculture use case. Here, we provide one possible solution to Problem 2.5.2, which we will generalize in Chapter 5. 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 the paths and the other the computations. We discuss the first sub-plan in Section 2.6.1 and the second in Section 2.6.2.

2.6.1 Paths sub-plan

The paths sub-plan defines the paths that form the plan. 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 NIZs $o := \emptyset$. An intuitive static plan Γ can be composed of lines connected by circles. The distance between the lines can then be a measure of the quality of the coverage. Such an intuitive static plan is illustrated in Figure 2.6.

We can use the intuitive plan to solve the coverage problem with a rotary-wing aerial robot with a small turning radius. A similar pattern, termed the boustrophedon motion, is abundant in the aerial robotics literature relative to CPP (Araújo et al., 2013; Artemenko et al., 2016; Cabreira, Franco, et al., 2018; Di Franco and Buttazzo, 2015) and in the broader mobile robotics literature (H. Choset, 2001; H. M. Choset et al., 2005; LaValle, 2006) where the circles are instead straight lines parallel to the segments connecting the corresponding vertices. However, this pattern is unsuitable for a fixed-wing aerial robot such as the Opterra. Fixed-wing aerial robots have reduced maneuverability compared to rotary-wings (Dille and Singh, 2013; Mannadiar and Rekleitis, 2010; Xu et al., 2011, 2014) and are generally unable to fly quick turns (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 coverage variation 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 robotics literature but optimized to our use case and potentially a broad class of aerial robots with turning constraints. Indeed such constraints are commonly treated in the aerial robotics literature (Artemenko et al., 2016; Y. Li et al., 2011; Xu et al., 2011,

[ii] There might be a further strict requirement to consider obstacles as no-flight zones (NFZs) rather than NIZs, which we discuss in the context of possible future directions.

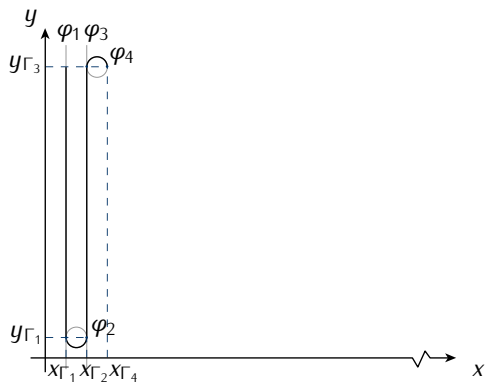


Figure 2.6. An intuitive plan to cover a regular polygon with four sides composed of circles φ_2, φ_4 and lines φ_1, φ_3. To switch between the paths, the aerial robot reaches the proximity of triggering points $\mathbf{p}_{\Gamma_1} := (x_{\Gamma_1}, y_{\Gamma_1}), \mathbf{p}_{\Gamma_2}, \mathbf{p}_{\Gamma_3},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

2014), and similar patterns are already in other studies (Huang, 2001; Xu et al., 2014). We discuss further the boustrophedon, Zamboni, and other motions for the coverage in Chapter 3.

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

$$\varphi_1(\mathbf{p}(t)) := x - 10, \tag{2.11a}$$
$$\varphi_2(\mathbf{p}(t)) := (x - 85)^2 + (y - 10)^2 - 5625, \tag{2.11b}$$
$$\varphi_3(\mathbf{p}(t)) := x - 160, \tag{2.11c}$$
$$\varphi_4(\mathbf{p}(t)) := (x - 90)^2 + (y - 140)^2 - 4900, \tag{2.11d}$$

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

$$\mathbf{p}_{\Gamma_1} := (10, 10),\ \mathbf{p}_{\Gamma_2} := (160, 10),\ \mathbf{p}_{\Gamma_3} := (160, 140),\ \mathbf{p}_{\Gamma_4} := (20, 140). \tag{2.12}$$

The coverage problem can be solved using the paths in Equation (2.11), the triggering points in Equation (2.12), 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 (2.11–2.12). A generic solution for the coverage problem defined as an iterated pattern contains the primitive paths

$$\varphi_i(\mathbf{p}(t)) := x - x_{\Gamma_1} - \lfloor i/4 \rfloor x_{\mathbf{d}}, \tag{2.13a}$$
$$\varphi_{i+2}(\mathbf{p}(t)) := x - x_{\Gamma_2} - \lfloor i/4 \rfloor x_{\mathbf{d}}, \tag{2.13b}$$

$\forall i \in \{1, 5, 9, \ldots\}$. $x_{\mathbf{d}}$ is a shift on the x-axis, $\lfloor i/4 \rfloor$ is the integer division. The expressions in Equation (2.13) correspond to the generalizations of the lines in Equation (2.11a) and Equation (2.11c). The generalizations of the circles in Equation (2.11b) and Equation (2.11d) are

$$\varphi_{i+1}(\mathbf{p}(t)) := (x - x_{\Gamma_1} - r_1 - \lfloor i/4 \rfloor x_{\mathbf{d}})^2 + (y - y_{\Gamma_1} - \lfloor i/4 \rfloor y_{\mathbf{d}})^2 - r_1^2, \tag{2.14a}$$
$$\varphi_{i+3}(\mathbf{p}(t)) := (x - x_{\Gamma_2} + r_2 - \lfloor i/4 \rfloor x_{\mathbf{d}})^2 + (y - y_{\Gamma_3} - \lfloor i/4 \rfloor y_{\mathbf{d}})^2 - r_2^2, \tag{2.14b}$$

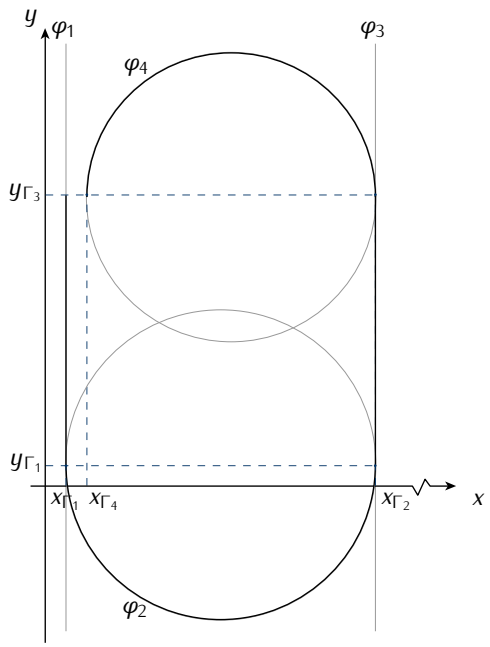


Figure 2.7. Fixed-wing aerial robot plan to cover a regular polygon with four sides. The plan covers the polygon with the same principle in Figure 2.6 but respecting turning constraints. The polygon is covered by iterating the primitive paths (gray lines) and triggering points (dashed blue line). The black line is the planned flight. The height and length of the polygon are $y_{\Gamma_3} - y_{\Gamma_1}$ and $lx_\mathbf{d}/4$.

where index i is defined the same way as in Equation (2.13), along with 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

$$\mathbf{p}_{\Gamma_i} := (x_{\Gamma_1} + \lfloor i/4 \rfloor x_\mathbf{d}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_\mathbf{d}), \tag{2.15a}$$

$$\mathbf{p}_{\Gamma_{i+1}} := (x_{\Gamma_1} + 2r_1 + \lfloor i/4 \rfloor x_\mathbf{d}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_\mathbf{d}), \tag{2.15b}$$

$$\mathbf{p}_{\Gamma_{i+2}} := (x_{\Gamma_1} + 2r_1 + \lfloor i/4 \rfloor x_\mathbf{d}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_\mathbf{d}), \tag{2.15c}$$

$$\mathbf{p}_{\Gamma_{i+3}} := (x_{\Gamma_1} + 2r_1 - 2r_2 + \lfloor i/4 \rfloor x_\mathbf{d}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_\mathbf{d}). \tag{2.15d}$$

We note from Figure 2.7 and Equations (2.13–2.15) 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 $lx_\mathbf{d}/4$ is the length of the polygon if the number of stages l is known.

The plan in Equations (2.13–2.15) is static: no path parameter allows altering the plan. We can transform such a 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

$$r_2(c_{4,1}) := \sqrt{r^2 + c_{4,1}}, \tag{2.16}$$

where $\underline{r} < r < r_1$ is a given positive constant initial radius and $c_{4,1} \in (\underline{r}^2 - r^2, 0]$. 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 squared $\underline{c}_{4,1} > \underline{r}^2 - r^2$ (to respect the turning radius).

We note from the expression in Equation (2.16) that Equation (2.15d) and Equation (2.14b) depend on the parameter $c_{4,1}$ (they contain r_2, which depends 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

$$c_{i,1} \in [\underline{c}_{4,1}, \overline{c}_{4,1}] =: \mathcal{C}_{4,1} \subseteq (\underline{r}^2 - r^2, 0], \ \forall i, \tag{2.17}$$

where we assume for ease of notation that we can change the parameter in advance at any stage (thus we use $\forall i$ in the equation). We will see in Chapter 6 that $\underline{c}_{4,1}$ is usually chosen from empirical observations to alter the flight time in case of, e.g., sudden battery drop, while still respecting the turning radius.

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 influences the energy consumption. Since the last path φ_4 is a function of the parameter $c_{4,1}$, the correct expression for Equation (2.14b) is

$$\begin{aligned} \varphi_{i+3}(\mathbf{p}(t), c_{4,1}) := (x - x_{\Gamma_2} + r_2(c_{4,1}) - \lfloor i/4 \rfloor x_{\mathbf{d}})^2 + \\ (y - y_{\Gamma_3} - \lfloor i/4 \rfloor y_{\mathbf{d}})^2 - r_2(c_{4,1})^2. \end{aligned} \tag{2.18}$$

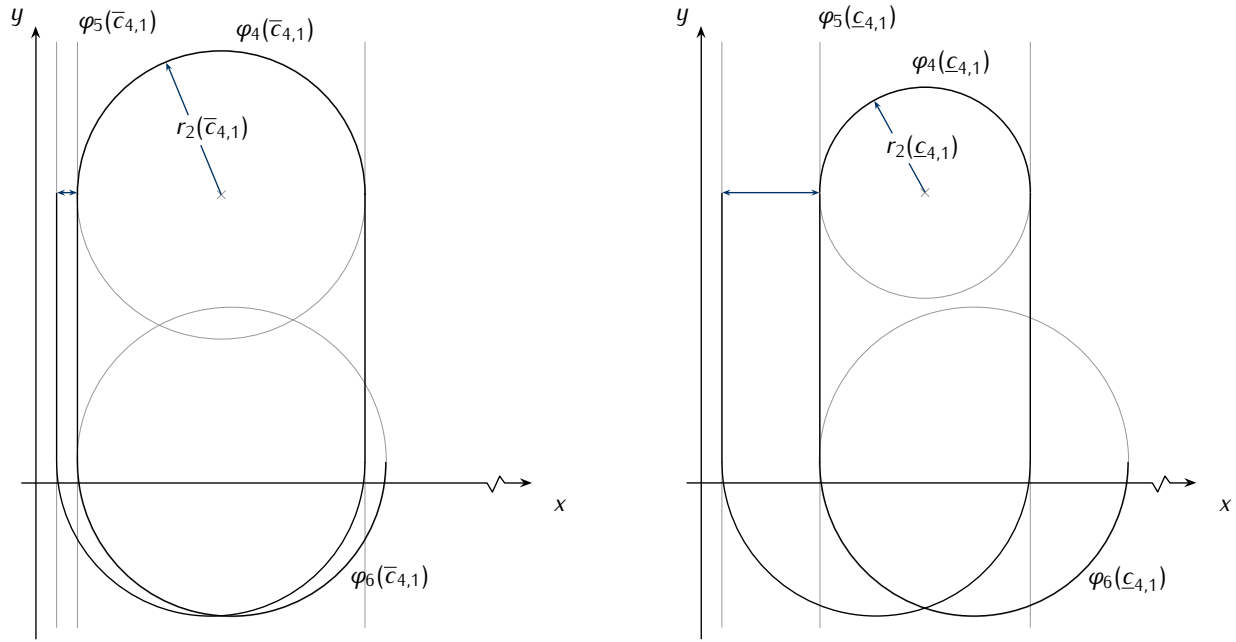
The last triggering point $\mathbf{p}_{\Gamma_4}$ in Equation (2.15d) is likewise a function of the parameter $c_{4,1}$

$$\mathbf{p}_{\Gamma_{i+3}}(c_{4,1}) := (x_{\Gamma_1} + 2r_1 - 2r_2(c_{4,1}) + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.19}$$

as it depends on the radius r_2 of the circle φ_4. The same applies to all the following functions $\varphi_5, \varphi_6, \varphi_7$ 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 $\varphi_9, \varphi_{10}, \varphi_{11}$ 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. In the precision agriculture use case, we are interested in monitoring the field in Figure 1.6, detecting ground hazards, and communicating with other ground-based actors. We use a convolutional neural network (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 and that the communication between the aerial robot and the centralized workstation 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



(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 $2(r_1 - r_2(\underline{c}_{4,1}))$. The same principle applies to path parameter $c_{8,1}$ for the next two survey lines, to $c_{12,1}$, and so on.

ROS nodes here and discuss the implementation in Chapter 6. The schedule for the CNN ROS node is parametrized by the FPS rate at which the frames are sent from the camera and for the encryption ROS node 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 is the encryption binary value. The CNN ROS node's computation parameter is $c_{i,2}$, and the encryption ROS node's computation parameter is $c_{i,3}$.

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

$$c_{i,3} \in \mathcal{S}_{i,3} := \{0, 1\}, \forall i, \tag{2.20}$$

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 (2.23b) and Equation (2.23d), the computations parameters constraint sets are

$$c_{i+1,2}, c_{i+3,2} \in \mathcal{S}_{i+1,2} = \mathcal{S}_{i+3,2} := \{0\}, \forall i \in \{1, 5, 9, \dots\}, \tag{2.21}$$

and thus the CNN ROS node is not searching for any hazard when it flies out of the polygon. On the contrary, for the lines in Equation (2.23a) and Equation (2.23c)

$$c_{i,2}, c_{i+2,2} \in \mathcal{S}_{i,2} = \mathcal{S}_{i+2,2} := [2, 10], \forall i \in \{1, 5, 9, \dots\}, \tag{2.22}$$

the CNN ROS node is detecting hazards on the ground with an 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 introduce in Section 4.1, to measure and model the future energy consumption of different computations' configurations.

2.6.3 Coverage plan with paths and computations sub-plans

The plan for the precision agriculture use case is composed of stages, containing the primitive paths in Equations (2.13–2.14) and the parameter-dependent version in Equation (2.18). It further contains the path parameter $c_{i,1}$, $\forall i \in \{4, 8, \dots\}$ and 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

$$\Gamma_i := \{\varphi_i(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}), c_{i,2}, c_{i,3}\}, \tag{2.23a}$$

$$\Gamma_{i+1} := \{\varphi_{i+1}(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}), c_{i+1,2}, c_{i+1,3}\}, \tag{2.23b}$$

$$\Gamma_{i+2} := \{\varphi_{i+2}(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}), c_{i+2,2}, c_{i+2,3}\}, \tag{2.23c}$$

$$\Gamma_{i+3} := \{\varphi_{i+3}(\mathbf{p}(t), c_{4,1}, c_{8,1}, \dots, c_{i-1,1}, c_{i+3,1}), c_{i+3,2}, c_{i+3,3}\}, \tag{2.23d}$$

$\forall i \in \{1, 5, 9, \dots\}$. The path constraint set for the path parameter $c_{i,1}$ is then in Equation (2.17), the computation constraint set for the computation parameter $c_{i,2}$ in Equation (2.22), and for

the computation parameter $c_{i,3}$ in Equation (2.20). The triggering points

$$\mathbf{p}_{\Gamma_i}(c_{4,1},\dots,c_{i-1,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.24a}$$

$$\mathbf{p}_{\Gamma_{i+1}}(c_{4,1},\dots,c_{i-1,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + 2r_1 + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_1} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.24b}$$

$$\mathbf{p}_{\Gamma_{i+2}}(c_{4,1},\dots,c_{i-1,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + 2r_1 + \lfloor i/4 \rfloor x_{\mathbf{d}}, y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}), \tag{2.24c}$$

$$\mathbf{p}_{\Gamma_{i+3}}(c_{4,1},\dots,c_{i+3,1}) := (x_{\Gamma_{i-4}}(c_{4,1},\dots,c_{i-1,1}) + 2r_1 - 2r_2(c_{i+3,1}) + \lfloor i/4 \rfloor x_{\mathbf{d}}, \; y_{\Gamma_3} + \lfloor i/4 \rfloor y_{\mathbf{d}}). \tag{2.24d}$$

$\forall i \in \{5, 9, \dots\}$. 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 (2.16). It contains r, which is likewise given (we can estimate r from the desired distance of the covering lines $r = r_1 - x_{\mathbf{d}}/2$).

The solutions 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. Since each quadruple of stages in Equation (2.23) and triggering points in Equation (2.24) 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

$$c_{i,1} = c_{i+1,1} = c_{i+2,1} = c_{i+3,1}, \; \forall i \in \{1, 5, 9, \dots\}. \tag{2.25}$$

Chapter 5 generalizes the plan in this section with coverage of an arbitrary polygon and derives the optimal configuration c_i^*. Chapter 6 revisits the assumption in Equation (2.25) and the two sub-plans in an experimental implementation of the agricultural use case.

2.7 Summary

In this chapter, we defined the planning and coverage problems. The solution to the coverage problem is a cover tour; to the planning problem, the optimal configurations of the path and computations parameters. Both are interconnected and require some basic concepts, including plan and path functions. The plan is composed of stages, triggering, and final points and parameters for the replanning itself. At each stage, the robot flies a path function and computes a schedule. Some parameters parametrize the path, and some parameters parametrize the computations. Bounds limit these parameters due to physical impediments of the computing hardware and the aerial robot. Alternatively to defining all the stages manually, it is possible defining an autonomous use case with some periodicity, using primitive paths and a shift. We illustrated the concept with the precision agriculture use case. Here, we provided two sub-plans. The first contains the paths constructed with the Zamboni-like motion for fixed-wing aerial robots with strict turning radius constraints. The second contains the computations with the hazards detection and communication with other ground-based actors. There is one parameter to alter the quality of the coverage with the radius in the Zamboni-like motion and two parameters to alter the computations of the ground hazards detection and communication.

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, including battery-powered. 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, plan the path and schedule 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 planning-scheduling energy awareness. 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 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 (CPP) and the derivation of optimal coverage. Although simultaneous planning-scheduling 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 work. We report these in detail in Section 3.5.

This chapter connects to the remainder of this work as follows. Here we discuss the other literature that allowed us to derive an energy-aware coverage planning and scheduling approach for autonomous aerial robots. Based on these findings, we propose computations and motion energy and battery models in Chapter 4. Planning in the context of mobile robots, for aerial robots, of the coverage and the 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 problem in Chapter 2. In Chapters 6–7, we report the findings and conclude the work relating to the state-of-the-art in this chapter.

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 model: a selection of hardware parameters or a configuration of 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. We further state if a given study use DVS and/or DFS, summarize the accuracy, and detail 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 power metrics: the scaled speedup, performance per watt, and 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 simultaneously on CPU and GPU. Our work extends the model and builds a computations model of 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 (in Marowka and Bailey et al.) and a CPU with a microcontroller as a heterogeneous platform instead of a CPU with 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. It is involved in a software partitioning problem to derive the optimal mode via integer linear programming (ILP).

		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

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. We expand further by merging the path with the planning-scheduling energy awareness.

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–has no built-in power monitors. 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 without built-in power monitors, as opposed to other computing hardware 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's bitwidth, sparsity, and architecture. The methodology applies exclusively to DNNs but has been extended to 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 scheduling and details aspects including Quality of Service (QoS) we use to define computations execution boundaries.

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). None is, however, focused on 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. They use a collection of applications, deriving the optimal values of the counters. The approach works across multiple hardware; data gathered from one hardware estimate the power and performance of 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), deterring similar approaches in our work, where we aim to preserve the energy to the greatest extent.

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 further observes the energy implications of a given benchmark but does not propose an energy optimization technique. The accuracy analysis shows a strong correlation between observations and the analytical model. We also derive an analytical expression further motivated with empirical observations of energy data and employ 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 specifically, 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 GPU. In our approach, we similarly input configuration to the model but specify the computing hardware computations rather than architectural parameters. We use the model output 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., 2021a) and focus on homogeneous hardware.

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 processor and gem5–a full-system architectural performance simulator. Reddy et al. evaluate the model against sixty workloads reporting high accuracy. To collect the measurements, the model uses built-in power monitors in the ODROID-XU3 platform. 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., 2021a). The model further deals with ARM big.LITTLE architecture. We use these models to evaluate the performance of our computations energy modeling in Section 6.1.4.

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 both Reddy et al. and 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

This section describes battery modeling approaches and focuses on battery models to be used for both the computing hardware and the aerial robot. There are several models for this purpose, each with its advantages and disadvantages. Usually, battery models derive the battery state of charge (SoC) (Xia et al., 2015), at various levels (Deng et al., 2017; Kurzweil and Scheuerpflug, 2021; Kurzweil and Shamonin, 2018; Sundén, 2019). 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 in Section 3.1. Mixed models use empirical data to derive 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. Sometimes referred to as equivalent circuit models (ECMs), they have compelling trade-offs concerning analytical insight, accuracy, complexity, and ease of implementation (R. Rao et al., 2003). They are the models of our choice for battery modeling. ECMs use different constructs, including 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). 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) based on an equivalent electrical circuit. The first model (Xing et al., 2014) utilizes an unscented Kalman filter to tune parameters at each sampling step, and the second (Hasan et al., 2018) estimates the SoC with an eXogenous Kalman filter (Johansen and Fossen, 2017). There

are multiple others who focus on equivalent electrical circuits for battery modeling (He et al., 2011; Hinz, 2019; Madani et al., 2019; Mousavi G. and Nikdel, 2014; C. Zhang, Allafi, et al., 2018; F. Zhang, Liu, and Fang, 2009; F. Zhang, Liu, Fang, and H. Wang, 2012). We will see a concrete implementation of a circuit termed "Rint" (He et al., 2011; Hinz, 2019; Mousavi G. and Nikdel, 2014) in Section 4.2.1 for a Li-ion aerial robot's battery.

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 physical battery characteristics. The approach is insightful in terms of evaluating models for computing hardware. Indeed our approach relies similarly on an abstract model, 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 of what battery the aerial robot mounts.

Zhang et al. propose a modeling approach for smartphones (L. Zhang et al., 2010) that consists of an automated 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 electrical circuit. Although the analysis is limited to smartphones, it provides insights into automated modeling techniques for computing hardware. We have similarly derived an automated technique for computations energy and battery models in Chapter 4.

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

3.3 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 energies altogether, we detail planning in the literature. We are interested in planning and scheduling, and specifically, in coverage planning of a given space. 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 3.3.3 and Section 3.4.2, we discuss these studies respectively for mobile and aerial robots.

3.3.1 Motion planning

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 energies. In (Mei, Y.-H. Lu, Y. C. Hu, et al., 2004), they focus on motion energy optimizations and analyze the computations energy in the following iteration of their work (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005), discussed in Section 3.5. In the former, a path plan and a velocity schedule form a "motion plan". The path plan is similar to ours in Definition 2.4.2, and the velocity schedule contains velocities, accelerations, and decelerations along the route. The study analyzes the most energy-efficient motion plan by evaluating the energy cost of turns and accelerations and differentiating between the travel and motors velocities. The experimental setup consists of Palm Pilot Robot Kit with polyurethane omnidirectional wheels and three MS492MH DC servo motors. It relies on external power monitors and five batteries: one 9 and four 1.5 volt batteries for the control circuit and motors. The energy efficiency is measured in terms of squared meters per jouls. Here, Mei et al. report an improvement of 50% by using an energy-optimized path plan.

The study has encouraged our initial analysis in many respects. It is the first we encountered analyzing motion and computations energies (Mei, Y.-H. Lu, Y. C. Hu, et al., 2005). However, Mei et al. do not consider optimizations in terms of path variations. They focus on straight lines, spirals, and the energy cost of turns, whereas we vary CPP within given limits. They do not explicitly address planning-scheduling energy awareness but propose foundations for future studies. We have optimized the coverage plan in Section 2.6.1 (and Section 5.2.2) compared to the traditional coverage planning that uses, e.g., boustrophedon motion (LaValle, 2006). We do not consider a velocity schedule due to the physical constraints of airborne systems. Nevertheless, we also optimize the plan against sudden accelerations. Indeed the boustrophedon-like motion requires sudden decelerations in contrast to the Zamboni-like motion for fixed-wing crafts.

Dressler and Fuchs undertake a different approach (Dressler and Fuchs, 2005) to energy-efficient motion planning. The approach models the energy of motion plans for specific 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. The study models energy-consuming parts of the robot with corresponding characteristic curves. Dressler and Fuchs observe a linearly approximable curve for the SoC for sub-tasks of a given task, such as movement and wireless communication. Although we do not explicitly model the energy of a given task, we similarly formulate the plan as a set of paths and computations in Definition 2.4.2. The study has thus inadvertently introduced the differentiation between computations and motion energies. The reported sub-tasks energy models are relative to one of the two components: wireless communication to a computation that transfers data airborne; the

movement to a path to reach a 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, LaValle's textbook (LaValle, 2006) groups planning algorithms literature and focus on robot motion planning. Juliá et al. propose an extensive survey (Juliá et al., 2012) of the most important methods for motion planning for autonomous exploration and mapping of unknown environments. The survey presents different techniques, classifying for multi-robot coordination level and integration with simultaneous localization and mapping (SLAM) algorithm.

3.3.2 Coverage path planning

In autonomous mobile robotics use cases, it is often required to explore every location in a given elemental region (Cao et al., 1988); this problem is defined in the literature as the 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 sub-topic of motion planning in Section 3.4.1. Our work uses CPP to cover a space in Problem 2.5.2.

CPP is related to the covering salesman problem, a variation of the famous traveling salesman problem (TSP) (E. M. Arkin and Hassin, 1994). The salesman visits a neighborhood of each city, minimizing the travel length (E. M. Arkin and Hassin, 1994), and passes over all the 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 the CPP problem 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 (H. Choset, 2001). The overall solution to the CPP 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 CPP to motion planning between the cells (H. Choset, 2001). The cells are geometric structures that 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 cell in case of an obstacle (change in connectivity) (H. Choset, E. Acar, et al., 2000). Choset et al. propose exact cellular decomposition methods (H. Choset and Pignon, 1998), complemented with different

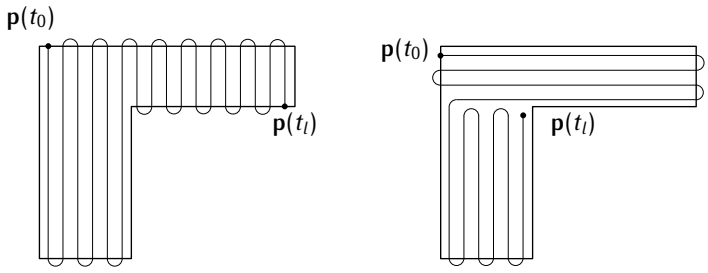
motions in the following instance (H. Choset, E. Acar, et al., 2000) (e.g., spiral, spike, diamond pattern, and others). Within the cellular decomposition method, some approaches (E. U. Acar et al., 2002; H. Choset, E. Acar, et al., 2000) work with the concept of Morse function to indicate the cell boundaries location, dealing with generalized obstacles. We adapt the boustrophedon decomposition proposed by Choset et al. with the Zamboni-like motion 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), which 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 variations 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 CPP and further detail relevant studies on optimal coverage later on.

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). Choset reports that many algorithms for CPP 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.

It is the case in Galceran and Carreras' survey (Galceran and Carreras, 2013) of CPP approaches. The survey classifies the algorithms among the others in online and offline classes. The former relies 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. 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, graphs, and grids algorithms. The work further surveys some approaches for natural landmarks detection and robots with contact sensors. It covers methodologies operating in rectilinear and 3D environments, for optimal coverage, for multiple robots, and based on reduction of the localization error.

3.3.3 Optimal coverage

Huang analyzes an optimal coverage methodology based on cellular decomposition. The study (Huang, 2001) emphasizes the cost of performing turns, utilizing round turns similar to the boustrophedon-like motion. To derive the coverage, Huang minimizes the sum of sub-region altitudes–a metric related to the sweep direction of a sub-region. Figure 3.1 illustrates the principle, where different sweeping directions produce a different number of turns. Huang proposes the methodology for both convex and non-convex shapes and uses dynamic programming for



(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).

the optimal direction in the coverage. The approach performs better than other approaches and is insightful in terms of analyzing the coverage optimality. We similarly try to optimize the turns.

Arkin et al. focus (E. M. Arkin, Bender, et al., 2001, 2005) on optimal coverage and, similarly to Huang, consider 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, with the robot being required to decelerate (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 just to minimize the turns. They propose various approximation algorithms to compute nearly optimal covering tours. However, 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 nor replanning in case of, e.g., external interferences affecting the feasibility of the original plan. The latter scenario can happen when an aerial robot suffers a sudden battery drop flying a given optimal tour.

Shnaps and Rimon propose a solution to this problem, focusing on robotics scenarios explicitly. In the study (Shnaps and Rimon, 2016), a robot covers 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, it returns to the starting point to recharge the battery. Shnaps and Rimon model the energy cost with the path length. 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. They propose an algorithm (Wei and Isler, 2018) restricted to axis-parallel motion generalized for arbitrary polygons. The approach is tested on aerial robots but is limited to rotary-wing crafts. In both Shnaps and Rimons's and Wei and Isler's studies, the methodology divides the space into equally sized cells in a grid, and the energy optimality is considered within CPP rather than planning-scheduling energy awareness.

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 uncertainty with a noise-dependent model, including a time budget. Popović et al. plan the path online, combining evolutionary optimization and global viewpoint selection. They validate the approach with a precision agricultural use case to detect weeds. Although the proposed use case has similarities with ours, it does not account for different aerial robots nor involves planning-scheduling energy awareness. Furthermore, the use case does not require complete coverage, making it unsuitable for some applications we consider, such as hazards detection. Hayat et al. derive an approach (Hayat et al., 2017) based similarly on evolutionary optimization. It plans tasks of multiple aerial robots (the notion of task refers to a specific action rather than computation) in a search and rescue use case. The approach has similar 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 CPP nor disturbances. Kreciglowa et al. focus on the best trajectory between two hovering configurations in terms of energy (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 fixed-wing aerial robots. The approach passes through a set of waypoints by satisfying a given constraint on curvature.

Two surveys (Dadkhah and Mettler, 2012; Goerzen et al., 2010) summarize approaches for aerial robotics planning. The earlier 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 later survey (Dadkhah and Mettler, 2012) analyzes these explicitly. It groups the motion planning approaches by the class of uncertainty, e.g., vehicles' dynamics uncertainty, environmental uncertainty, etc. In the former, the survey proposes studies based on optimal control and artificial intelligence, and in the latter, on mapping.

3.4.1 Aerial coverage path planning

In the context of aerial robots, the survey (Galceran and Carreras, 2013) proposed by Galceran and Carreras that we discussed in Section 3.3 focuses 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 comes into the intersection of optimal coverage and coverage with aerial robots. Indeed in our coverage planning, we refine a plan in-flight 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. It fo-

cuses 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 optimizations and focus on different covering strategies. They describe the Zamboni motion.

Motions are similarly analyzed in Araújo et al.'s work (Araújo et al., 2013). It details the boustrophedon (that appears under the name "lawnmower"), Zamboni, spiral, spiral-like, modified boustrophedon, modified Zamboni (these are different from boustrophedon-like and Zamboni-like motions), and Dubins path motions.

Nam et al. propose an approach (Nam et al., 2016) for CPP for aerial robots that generates the covering tour offline. The methodology consists of a grid decomposition method, which visits the cells with a wavefront algorithm–a specialized version of the Dijkstra algorithm that optimizes the number of stages to reach the goal (LaValle, 2006). Nam et al. generate the optimal path between given points in space but do not focus on optimal criteria. Indeed experimentally, the results are not optimized w.r.t., e.g., the number of turns. Furthermore, Nam et al. do not deal with replanning nor account for energy explicitly, and the planning happens before the flight. Moreover, the approach focuses on rotary-wing aerial robots.

Sadat et al. propose an approach (Sadat et al., 2014) for non uniformly shaped areas distributed into clusters. The methodology is to cover adaptively, 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 discusses 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 aerial robot. It outputs the energy consumption as a function of velocity and operating conditions, using an interpolating curve of the power measurements. These curves are derived with the robot flying in several phases (during maximum acceleration and deceleration, horizontal flight, climbing, descending, hovering, and turns). In an extension (Di Franco and Buttazzo, 2016) of the original study (Di Franco and Buttazzo, 2015), they cover a polygon with the boustrophedon motion and consider the quality of the coverage with varying distances between the lines. Furthermore, the energy model is plan-specific–different paths have different analytical expressions. Our model in Section 4.3 is generic, once trained with enough measurements. Nonetheless, the study is insightful for defining the energy cost of a given coverage and for analyzing the energy effect of variations in the coverage quality. The experimental setup compromises an IRIS rotary-wing aerial robot, a GoPro camera

mounted on a gimbal stabilizer, and a PX4 flight controller (PX4, n.d.). 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. The approach minimizes the number of turns, showing that they are less efficient from energy, duration, and tour length points of view. The turns are already considered in the generic CPP by Arkin et al. and Huang (E. M. Arkin, Bender, et al., 2001, 2005; Huang, 2001). For complex polygons, Li et al. propose a convex decomposition algorithm, based on a greedy recursive method. To connect the decomposed subregions, 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 the optimality in terms of turns, nonetheless, they do not consider further requirements for different aerial robots, such as the turning radius.

Mannadiar and Rekleitis and Xu et al. propose studies (Mannadiar and Rekleitis, 2010; Xu et al., 2011, 2014) on a near-optimal complete coverage methodology, using a sequence of waypoints. The methodology (Mannadiar and Rekleitis, 2010) was first presented by Mannadiar and Rekleitis and extended to nonholonomic robots (Xu et al., 2011, 2014) by Xu et al. It uses the cellular decomposition with an arbitrary number of obstacles. To derive the coverage order, the cells are stored in a Reeb graph–an encoding for cyclic paths where critical points are vertices and cells are edges (Fomenko and Kunii, 1997)–that is visited via the Chinese postman problem (Eiselt and G. Laporte, 2000). The resulting path ensures that cells are not traversed more than twice and that the robot returns to the starting point. Xu et al. analyze possible planning strategies for fixed-wing aerial robots and, notably, affirm that this latter class of robots lack the maneuverability needed to follow a sharp-turned path. To this end, Xu et al. append a turning segment to the path by adding curlicue orbits at turns (Xu et al., 2014); whereas we incorporate the turning maneuver in our coverage planning rather than embedding external segments.

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 use cases. Cabreira et al. propose spiral coverage for some polygon shapes and alter the velocity to achieve energy savings. Artemenko et al. propose a boustrophedon motion with smoothed turns using Bézier curves. 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, the studies in this section do not deal with planning-scheduling energy awareness nor in-flight replanning. We take these aspects into account in our coverage planning.

3.5 Planning Computations with Motion

There are some approaches that deal with planning-scheduling energy awareness in the robotics research literature. Table 3.2 summarizes the most relevant to ours. We discuss these approaches in the remainder of this chapter.

Sudhakar et al. examine the trade-off between motion and computation energies for mobile robots in their recent study (Sudhakar et al., 2020), focusing on robots with similar motion and computations energies. 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 (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 energies. 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 energies (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 studies (Lahijanian et al., 2018; Ondrúška et al., 2015) carried by Ondrúška et al. and Lahijanian et al.

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 energies. 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 mobile robot–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 schedules 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. Mei et al. do not implement

	Replan.	Class	Robot	Planning-scheduling	Energy model	Bat.	Approach
Mei et al.	(✓)	ground	Pioneer 3DX	DVS, DFS, peripherals selection	regression	✗	greedy
Brateman et al.	✗	ground	-	DVS, DFS, velocity	analytical	✗	optimization
Zhang et al.	✓	ground	-	hardware frequency, velocity	analytical	✗	optimization
Sadrpour et al.	(✓)	ground	Pack-Bot UGV	peripherals selection	regression, Bayesian	✗	greedy
Ondrúška et al.	✓	ground	ARC Q14	perception (path following)	analytical	✗	greedy, optimization
Lahijanian et al.	✗	ground	ARC Q14	localization	analytical	(✓)[a]	Pareto front
Ho et al.	✓	ground	Pioneer 3DX	navigation (waypoints)	analytical	✗	RL
Sudhakar et al.	✓	ground	-	anytime planning	analytical	✗	greedy
Ours	✓	aerial	Opterra fixed-wing	coverage path planning	regression, differential	✓	otpimization

Table 3.2. Different methodologies in the literature for planning-scheduling energy awareness against ours (last line). The studies are ordered by ascending date. Replan. indicates that the planning-scheduling runs online (e.g., in-flight). Bat. that the approach accounts for the battery. (✓) that the approach is "hypothesized".

[a] The approach does not account explicitly for the battery. However, it derives some implications of different trade-offs w.r.t. the battery SoC.

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 coverage replanning. 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 5 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 replanning 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 computations energy (Sudhakar et al., 2020).

More recently, Ondrúška et al. proposed a study (Ondrúška et al., 2015) to reduce energy consumption by scheduling perception 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). The robot follows a predetermined path and accounts directly for computations energy by scheduling the perception so that it remains within a margin from the 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. The methodology consists of finding a schedule for mobile robots under a given resource budget using quantitative multi-objective verification and controller synthesis. This latter is a technique that starts from constraints such as time and energy and produces a set of optimal achievable trade-offs via a Pareto front (Forejt et al., 2012). Although insightful in selecting the best schedule, the approach does not account for replanning, whereas in our work, an initial plan accounts for the highest achievable performance. It is then replanned both path- and computations-wise in an energy-aware fashion. Lahijanian et al. also use the ARC Q14 robot.

Recent contributions in planning-scheduling energy awareness include the work (D.-K. Ho et al., 2018, 2019a,b) by Ho et al. The methodology varies the navigation on a set of waypoints and the schedule using reinforcement learning (RL). Concerning the motion energy, Ho et al. derive variations of controller frequency and travel velocity of a ground-based mobile robot. Whereas for the computations energy, they derive variations of schedules frames and resolution rates within specific QoS. Similar to Mei et al., the experimental setup consists of Pioneer 3DX ActivMedia mobile robot. Although the methodology derives plans-schedules online, the approach does not model the battery explicitly nor focuses on aerial robots.

3.6 Summary

This chapter discussed the state of the art in topics ranging from heterogeneous computing hardware energy modeling to planning-scheduling energy awareness. Initial sections are dedicated to past approaches in energy modeling, focusing on heterogeneous elements such as CPUs and GPUs. Follows the literature for battery modeling of both the computing hardware and the

aerial robot, and studies for the planning of mobile robots with an emphasis on approaches for CPP and optimal coverage. Within aerial robots, the chapter details relevant contributions for both motion planning and CPP and the derivation of an optimal aerial cover. Finally, the chapter compromises studies that, similarly to ours, deal with planning-scheduling energy awareness by accounting for computations and motion energies.

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 the high computing performance requirements of autonomous aerial robots is the motivation for the planning-scheduling energy awareness 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 flying a coverage plan. 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 empirical observations. The energy output of the motion model is the input of the battery model, which predicts the battery evolution.

We saw in Chapter 3 some computations energy and battery models and discussed energy-aware approaches for aerial (and mobile) robots 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 use case, 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 analysis. In Section 4.2, we provide a battery model based on an equivalent electrical circuit, and in Section 4.3, 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 energies. 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 results, including the outcomes of the models and future implications, are in Chapters 6–7.

4.1 Energy Model of the Computations

This section describes the computations energy model, which predicts the energy consumption of a given configuration of computations. In Section 2.1, 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 (see Definition 2.1.1). For instance, in the agricultural scenario, the computation object detection. Parametrization indicates these computations that can be scheduled via our planning-scheduling energy awareness, and run on, e.g., a lower/higher rate requiring lower/higher power. Practically, this means that if the 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. Chapter 5 details ways to derive the optimal schedule, whereas this section predicts the energy of any possible configuration. In the remainder, 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 3 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 3.1 for an extensive discussion of energy modeling approaches in the literature for CPUs and GPUs powered and heterogeneous computing hardware.

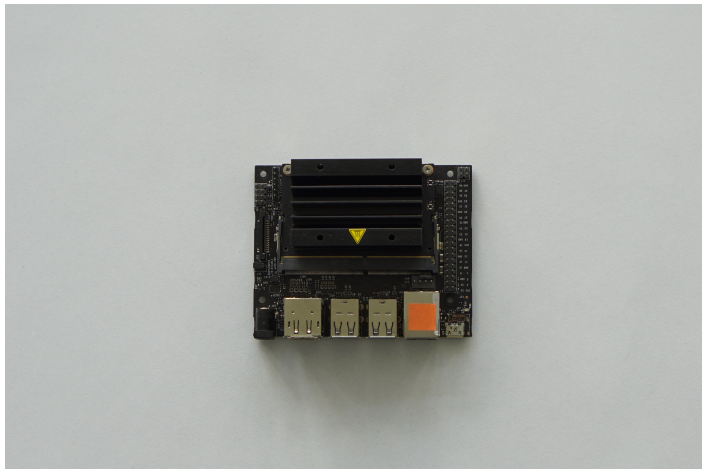


Figure 4.1. NVIDIA Jetson Nano heterogeneous computing hardware for computations energy modeling. The hardware in the image includes the Jetson Developer Kit board that the Nano heterogeneous board is mounted on and has a total size of 100x80 millimeters and a weight of approx. 140 grams.

The approach in this section is based on our work (Seewald, Schultz, Ebeid, et al., 2021a) on heterogeneous computing devices' energy modeling. It is generic, modeling virtually a wide range of computing hardware 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 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

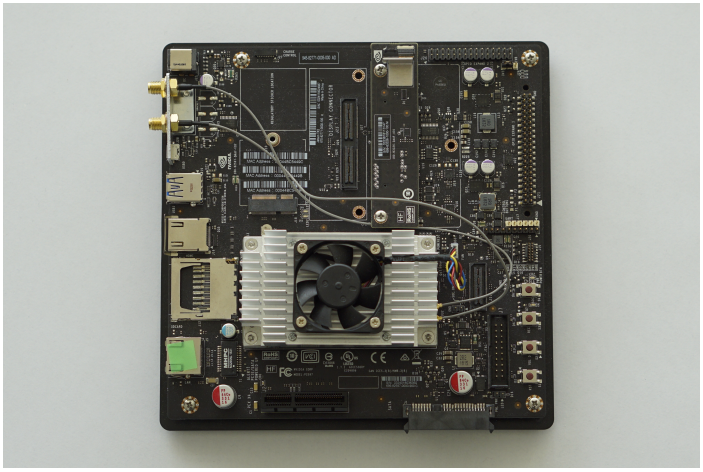


Figure 4.2. NVIDIA Jetson TX2 heterogeneous computing hardware mounted on a Jetson Developer Kit board as Nano in Figure 4.1 with a total size of 170x170 millimeters and a weight of approx. 512 grams.

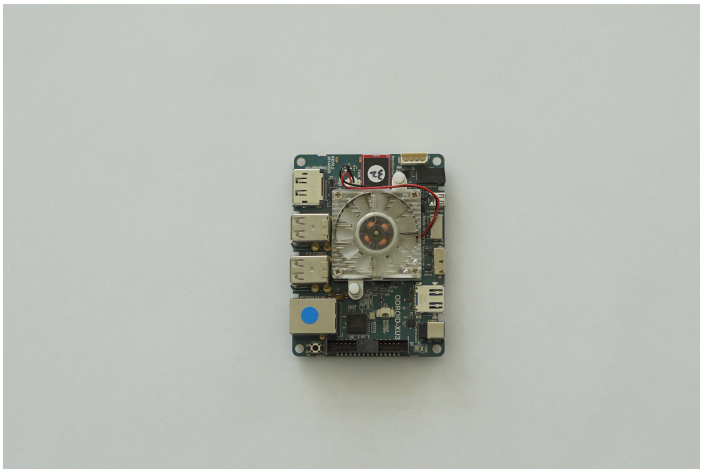


Figure 4.3. ODROID XU3 heterogeneous computing hardware with a total size of 94x70 millimeters and a weight of 70 grams.

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), 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.

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. All 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.

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 instants 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. 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 (2.20–2.22). 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 $c_{i,3}$, where δs are the sampling step. 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, $\delta_1 = 2, \delta_2 = 1$, so that there are ten configurations and consequently ten measurement layers (there are two possible configurations of parameter $c_{i,3}$ with δ_2 and five of $c_{i,2}$ with δ_1). 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. These include random sampling with the condition merely $c_{i,j} \in \mathbb{Z}_{>0}$ and exponential sampling with

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

where δ_1, δ_2 are now bases. Currently, the `powprofiler` tool supports automated linear and exponential sampling and complex sampling formed by different sampling strategies (Seewald, Schultz, Ebeid, et al., 2021a), 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., 2021a).

With the measurement layer described in this section, we know the power and other energy metrics of the sampled 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 $c_i^\sigma(t)$ the *predictive layer* is the function $g : \mathbb{Z}_{\geq 0} \times \mathbb{Z}^\sigma \to \mathbb{R}^3$. 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., 2021a). 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., 2021a).

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.

4.1.4 The `powprofiler` tool

The `powprofiler`[i] tool (Seewald, Schultz, Ebeid, et al., 2021b) 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., 2021a; TeamPlay Consortium, 2019a), 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

[i] The latest version of the tool can be retrieved from `https://github.com/adamseew/powprofiler`, whereas the version 1.0.2 released on October 2021 in this work from `https://doi.org/10.5281/zenodo.5562457`.

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.

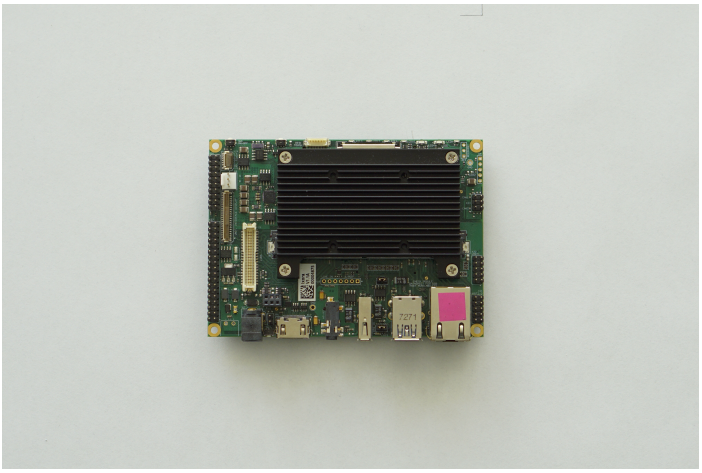


Figure 4.4. NVIDIA Jetson TK1 heterogeneous computing hardware mounted on a Toradex Ixora carrier board with a total size of 125x95 millimeters and a weight of approx. 160 grams.

The 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 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., 2021a).

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 generate 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 in 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. 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 discussed the most common ones in the literature in Section 3.2 . 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 equivalent electrical circuit. 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 (R. Rao et al., 2003).

We detail the equivalent electrical circuit that we adapted from the literature in Section 4.2.1; 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 5.3.2 to define the output constraint in Definition 5.3.1

4.2.1 Equivalent electrical circuit

In summary of Section 4.2.1, 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). 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). We omit further

details that interfere in the calculation, such as battery state of health, temperature, and C-rate. Indeed there 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).

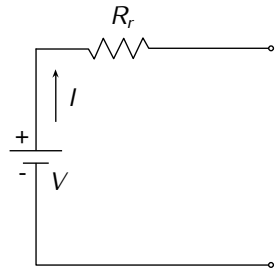


Figure 4.5. Equivalent electrical circuit for battery modeling with one resistor, representing the internal battery resistance.

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 require a current to be drawn from the battery–according to the equation (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

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

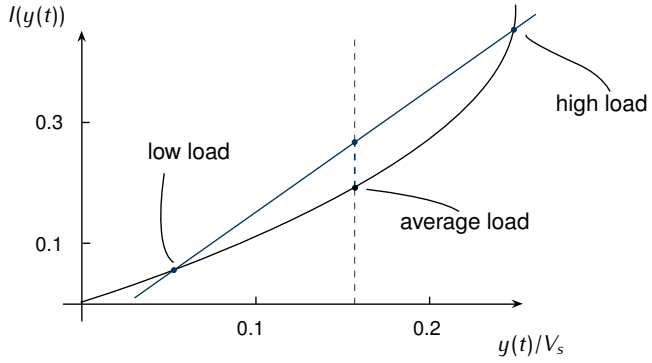


Figure 4.6. Evolution of I for a given linear load, showing that a constant load is to be preferred compared to one that repeatedly changes.

using simply Kirchhoff's circuit laws (the power into the load should exactly match the power out). Combining the Equations (4.6–4.7)[ii], we obtain the quadratic expression $R_r I^2 - V I + V_s I_l = 0$, 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., 2021a), 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 4.6, we show the evolution of I in Equation (4.8) for a given linear load $y(t)/v$ 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 ECM in Figure 4.7. 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).

[ii] Practical implementations would then require a voltage regulator.

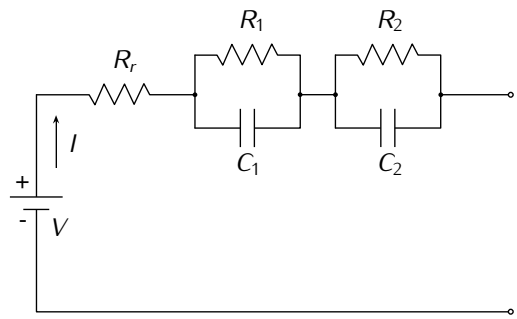


Figure 4.7. Thevenin-based equivalent electrical circuit for battery modeling with one resistor and two RC internal elements. The two elements add some complexity, making the model able to account for changes in the load current.

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 $\mathbf{g}$ 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 4.2.1. 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 4.7, 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 (Iserles, 2009). 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 serves 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.

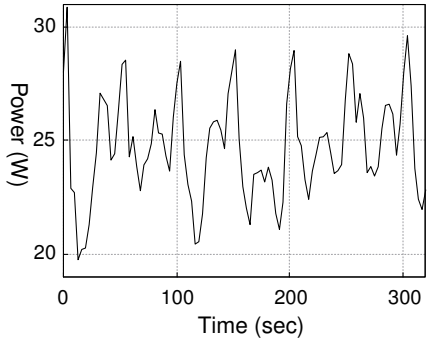


Figure 4.8. Empirical energy data of the aerial robot in Figure 1.6 in coverage planning. The data shows that the energy signal is periodic over time as the fixed-wing aerial robot reiterates a set of paths.

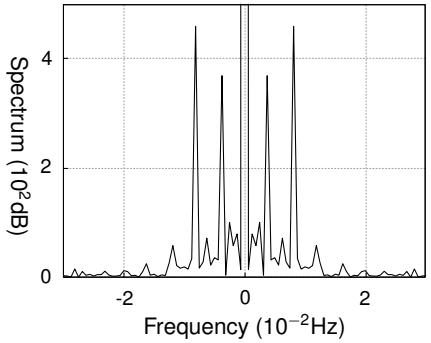


Figure 4.9. Frequency spectrum of the empirical energy data in Figure 4.8 where the aerial robot does the coverage planning.

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 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, García 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 in Chapter 6. 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 4.8–4.9. 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 4.8 (and further backs the choice of the Fourier series of third order in our

earlier work, illustrating that the power evolution needs approximately 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 Algorithm 5.3.

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

$$C = (1/T)\Big[1 \overbrace{\;1 \quad 0 \quad \cdots \quad 1 \quad 0\;}^{2r}\Big], \tag{4.14}$$

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 $a_0, \ldots, b_r$ are the same as the coefficients of the state $\alpha_0, \ldots, \beta_r$.

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 $t = \omega j t$, we find the expression for

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

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

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

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

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

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

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

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

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

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 computations 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 5.3.4 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 computations 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 $t - \Delta t$

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

where $\mathrm{diag}(v_i)$ 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 5.3.5.

4.3.3 Control scale transformation

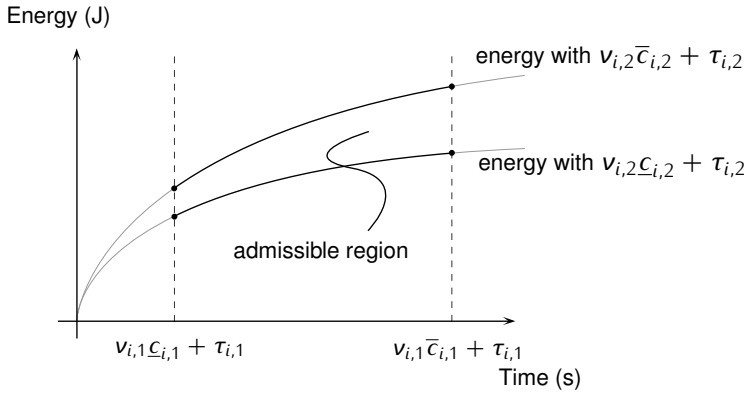


Figure 4.10. The concept of a path and computations parameters scale transformation. Without any battery constraints, the energy-aware coverage planning and scheduling select the highest configuration which respects the control constraint (admissible region) from Equation (2.6).

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 5.3.5.

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.

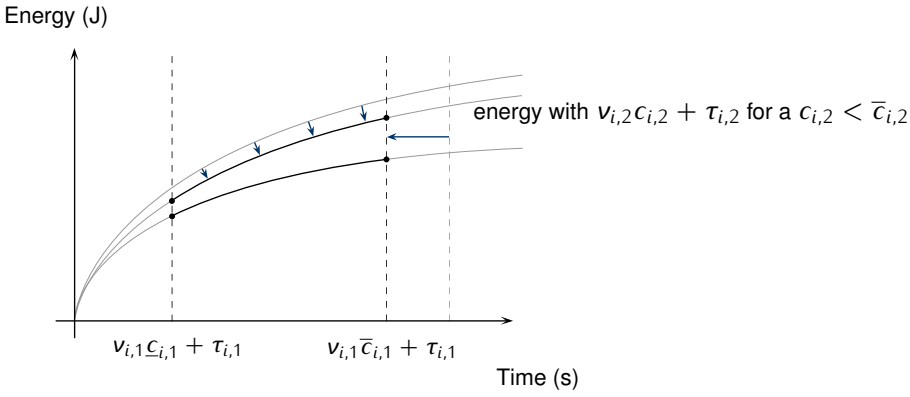


Figure 4.11. Change in the admissible region in Figure 4.10 due to a battery constraint.

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 4.10. 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 see in Section 5.3.1 the optimal control derivation over a time horizon N under given battery constraints. In Figure 4.11 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.

4.4 Summary

This chapter derives energy models to predict future instances of the energy for both the computations and motion. It details the `powprofiler` profiling and modeling tool to derive the power, energy, and battery SoC of computations configurations on the heterogeneous computing hardware within specific boundaries. The computations modeling methodology relies on a two-layer architecture. The search space is here sampled discretely for several possible configurations in the measurement layer. The union of multiple measurement layers forms the predictive layer, which returns the energy metrics in the function of any configuration in the search space. The motion modeling methodology relies on a differential periodic energy model that predicts future energy instances, relying on the characteristics of CPP and empirical observation of the aerial robot flying the precision agriculture use case. It includes the predictive layer from the `powprofiler` tool, quantifying the energy effect of schedules variations. The chapter further derives a battery model to accurately assess the battery SoC evolution. The model utilizes an equivalent electrical circuit in the literature termed "Rint". In the next chapter, we exploit the two energy and battery models for planning-scheduling energy awareness, replanning a plan energy-wise, and w.r.t. the current battery SoC.

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 discussed 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 one of the main contributions 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. 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 discuss the result of our planning-scheduling in Chapter 6. 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 $\mathcal{Q} \subseteq \mathbb{R}^2$ for an inertial navigation frame $\mathcal{O}_W$. For this purpose, we briefly recall some concepts we introduced in Chapter 2. 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 and Definition 2.3.1. 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 Definition 2.3.1 in Chapter 2 that we store path functions in stages. 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 $\mathbf{p}_{\Gamma_l}$ with the theory of vector fields. The path function can be, e.g., the line and circle 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 $\mathbf{p}(t_l) = \mathbf{p}_{\Gamma_l}$ 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 $\mathcal{Q}$ (LaValle, 2006), which will then point in the direction of the gradient

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

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.,

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

the 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 (García 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 Y.-C. Ho, 1975; H. M.

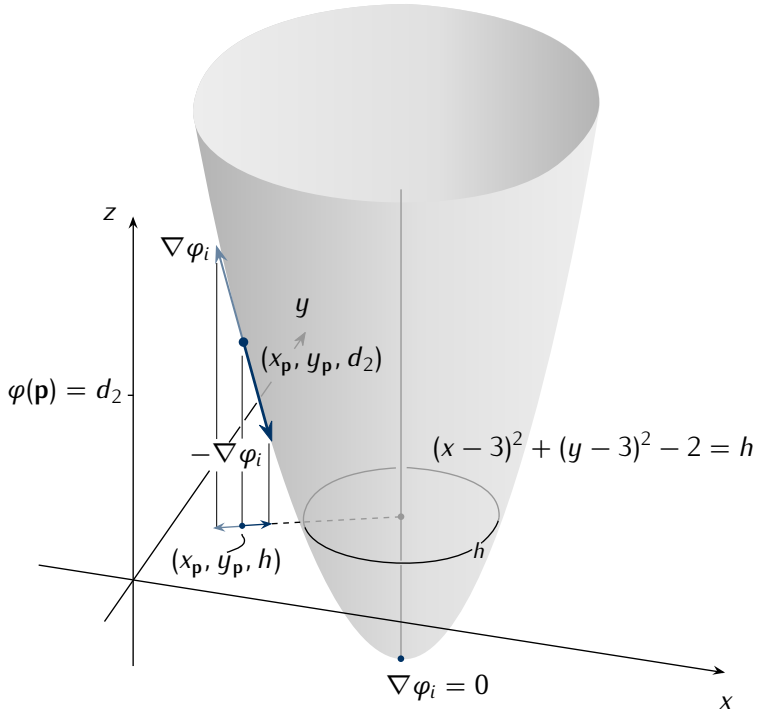


Figure 5.1. The direction of the gradient $\nabla\varphi_i$ in the point $(x_\mathbf{p}, y_\mathbf{p})$ at an altitude h w.r.t. $\mathcal{O}_W$ for the circle path function in Equation (2.3) and Figure 2.2. The gradient directs where the function locally maximally increases; the opposite thus to the 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 $\left\|\nabla\varphi_j(\mathbf{p}(i), c_i^\rho)\right\| \leq \varepsilon$ then
3 return $\mathbf{p}(\mathcal{K})$
4 $\mathbf{p}(i + h) \leftarrow \mathbf{p}(i) + \theta(i)\Delta\varphi_j(\mathbf{p}(i), c_i^\rho)$

Algorithm 5.1. Gradient descent

Choset 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 method in Algorithm 5.1, where we iterate at discrete time steps, meaning that at instant $n \in \mathbb{Z}_{>0}$, $\mathcal{K}$ contains t_0, $t_0 + h$, ..., $t_0 + nh$. 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 5.3.5. 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

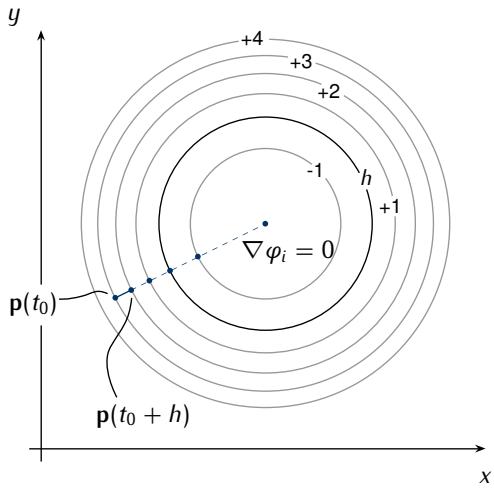


Figure 5.2. The gradient descent algorithm after the first two steps, with the following step being dashed and directing the aerial robot to the center of the circle where the gradient is zero $\nabla\varphi_i = 0$. The paraboloid φ_i representing the path function circle at altitude h is illustrated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

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. Concretely, in the path sub-plan in Equations (2.13–2.14) 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 (García 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 (García 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 (García 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

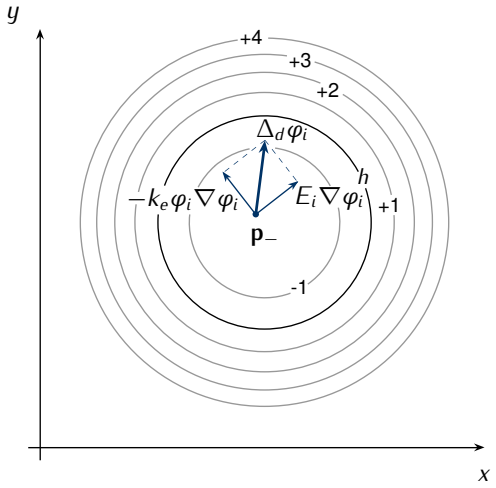


Figure 5.3. The direction of the vector field for guidance in (García de Marina et al., 2017) with a point $\mathbf{p}_-$ inside the circle: Δ_d points to the path function opposite to the center of the circle.

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

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

This vector fields points to the path function such as the line and the circle in Equation (5.4) for every point in space $\mathcal{Q}$ 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

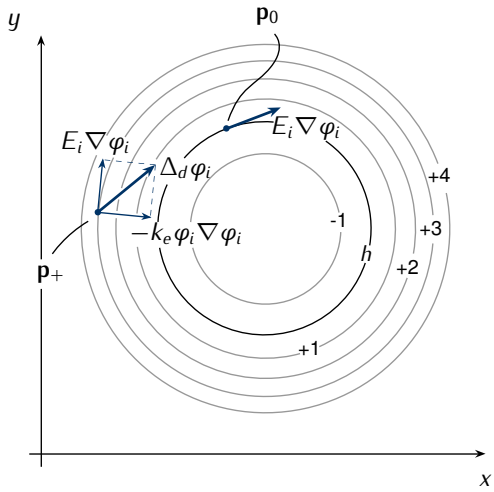


Figure 5.4. The direction of the vector field with $\mathbf{p}_+$ outside the circle: Δ_d points to the path function in the direction of the center of the circle. With $\mathbf{p}_0$, Δ_d points perpendicularly to the path function.

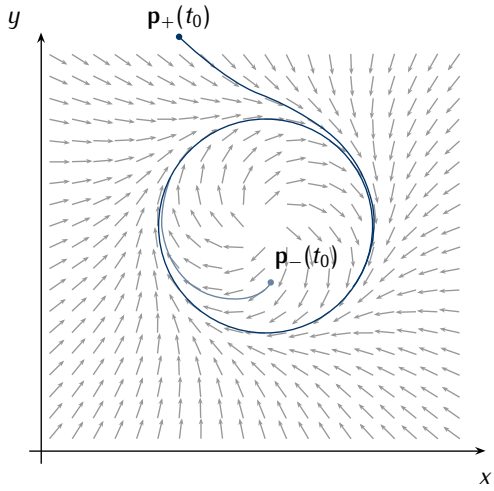


Figure 5.5. Path-following vector field Φ_d of a circle path function with the aerial robot starting inside $\mathbf{p}_-(t_0)$ and outside $\mathbf{p}_+(t_0)$ the circle.

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). We can further simplify Algorithm 5.2 and use the same value for the time step h that we used for the step size θ for all the time instants in $\mathcal{K}$.

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

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

Algorithm 5.2. Path function tracking

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 that covers each point in a given space. We describe the space with a set of vertices $v := \{v_1, v_2, \dots\}$ that form a polygon. The aerial robot is free to fly and compute anywhere within the polygon except for some no-interest zones (NIZs), where it should avoid computations. These are described by other sets of vertices, one per each NIZ $o_1 := \{o_{1,1}, o_{1,2}, \dots\}, o_2 := \{o_{2,1}, o_{2,2}, \dots\}, \dots$. There are several different approaches in the literature to solve this problem. We have detailed some in Sections 3.3.2–3.3.3 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 CPP algorithms are NP-hard (E. M. Arkin, S. P. Fekete, et al., 2000) and use (either implicitly or explicitly) the cellular decomposition, dividing the robot's free space into sub-regions that can be easily covered (H. Choset, 2001; Galceran and Carreras, 2013). Our approach extends one cellular decomposition method with the Zamboni-like motion we introduced in Section 2.6.1, respecting the turning radius constraint of, e.g., the Opterra in the precision agriculture use case. Fixed-wing aerial robots have limited maneuverability (Mannadiar and Rekleitis, 2010; Xu et al., 2011, 2014), hence a large turning radius (X. Wang et al., 2017). The decomposition is then between interest zones and NIZs; we compute over the former and solely fly over the latter. We provided a partial solution in Section 2.6, which here we generalize. A strict interpretation of NIZs as no-flight zones (NFZs), e.g., tall buildings in an urban monitoring use case, is also possible. We consider this latter case in the future directions in Chapter 7.

There are numerous methodologies for cellular decomposition itself. Some decompose the polygon into equally sized sub-regions that form a grid (see Figure 5.6) and then visits only the sub-regions where the robot is free to move (Galceran and Carreras, 2013). This methodology is termed grid decomposition. Another way is to sweep the polygon with a line 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, whereas the edges connect adjacent sub-regions (H. M. Choset et al., 2005). A covering order between the sub-region to derive the

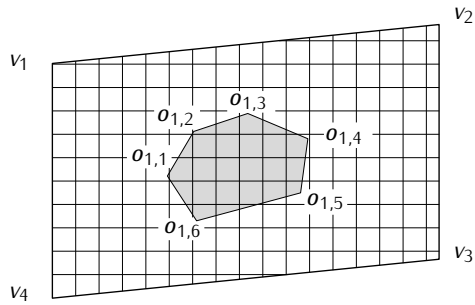


Figure 5.6. A polygonal space where we want to find a coverage tour and visit all the points delimited by $v := \{v_1, \ldots, v_4\}$. A way to cover the space is the grid decomposition that divides the polygon into equally sized cells and visits each cell.

sequence of the coverage in the literature is then an exhaustive walkthrough of the adjacency graph with, e.g., depth-first search algorithm (H. M. Choset et al., 2005). We use the covering order in terms of future directions, analyzing the energy implications of different sub-coverages against overall coverage avoiding computations over NIZs.

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 full coverage space as $\mathcal{Q}^v$ for an inertial navigation frame $\mathcal{O}_W$. Let $\mathcal{Q}^{o_i} \subset \mathbb{R}^2$ be the space delimited by the NIZ o_i. Physically, the free space is where the robot is free to fly the coverage and compute the computations, $\mathcal{Q}^{o_i}$ where it avoids computations. $\mathcal{Q}^v \subseteq \mathbb{R}^2$ contains all the points delimited by the vertices of the polygon v except for NIZs delimited by the vertices of polygons in o. The entire space in the polygon v, including all the NIZs o, is then $\mathcal{Q} := (\bigcup_{i \in |o|} \mathcal{Q}^{o_i}) \cup \mathcal{Q}^v$. In Figure 5.7, the polygon

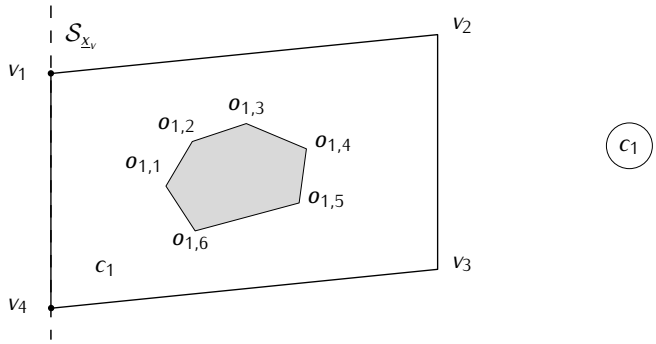


Figure 5.7. The boustrophedon decomposition for coverage path planning sweeps the space and adds cells in case the sweeping line encounters a change in connectivity. Figure shows an initial step with c_1 the first cell formed.

is delimited by $v := \{v_1, \ldots, v_4\}$ and forms $\mathcal{Q}$, whereas the NIZ by $o_1 := \{o_{1,1}, \ldots, o_{1,6}\}$

and forms $\mathcal{Q}^{o_1}$. The aerial robot in our implementation flies over $\mathcal{Q}$ but avoids computations over $\mathcal{Q}^{o_1}$.

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

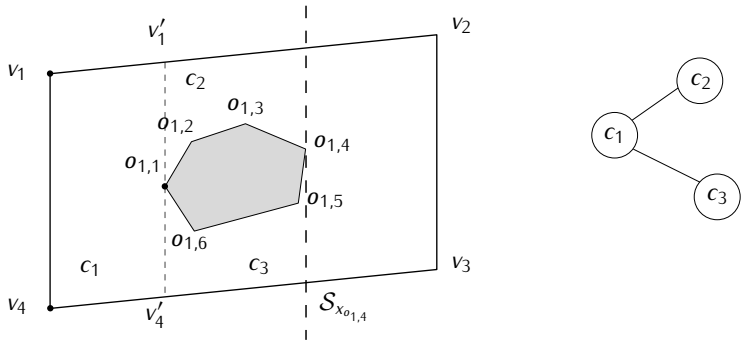


Figure 5.8. An intermediate step of the boustrophedon decomposition, with c_2, c_3 formed at the first encounter of the NIZ o_1. The black points indicate the critical points or changes in connectivity.

sweeping line encounters the NIZ o_1. This approach optimizes the neighboring cells that 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

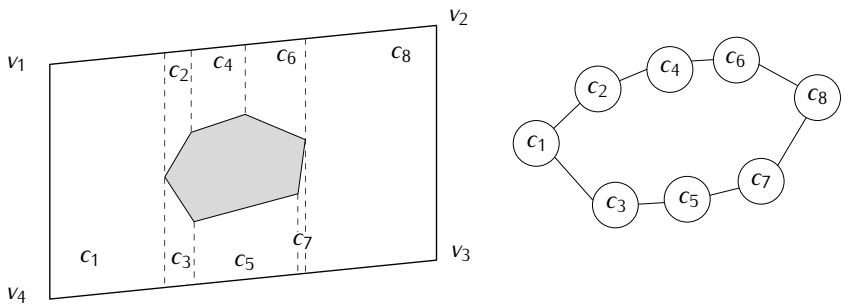


Figure 5.9. In the trapezoidal decomposition a lot of small cells are created (i.e., the cells c_2, c_3, c_7) that can be otherwise merged resulting in disconnected coverage. Boustrophedon decomposition solves the problem by splitting/merging the cells at critical points rather than at vertices. The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

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 obstacle (or NIZs in our case). 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 $\mathcal{S}^v_\lambda := \mathcal{S}_\lambda \cap \mathcal{Q}^v$. At this point, a change in connectivity of $\mathcal{S}^v$ means that the original cell has to be closed and two more opened, 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 5.7 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 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 which goes through $\mathcal{Q}^v$ is then a visit through the adjacency graph, resulting in the coverage order c_1, c_2, c_4, and finally c_3.

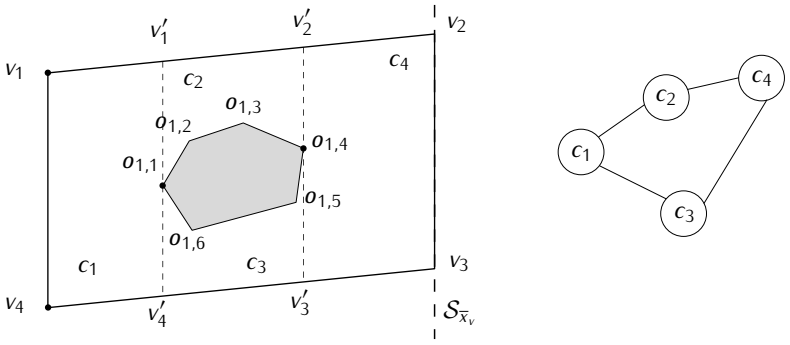


Figure 5.10. The final step of the boustrophedon decomposition, where the sweeping line $\mathcal{S}_\lambda$ encounters the final point of its domain $\overline{x}_v$. The decomposition results in four cells. To determine the order of the coverage tour, the methodology is to visit the adjacency graph.

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 space to cover, the tour that travels through such space is termed the coverage motion. A classical approach 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 merely the cell c_1 in $\mathcal{Q}$ in Figure 5.10 using the boustrophedon-like motion in Figure 5.11. The remaining cells are covered in the same manner. The overall coverage

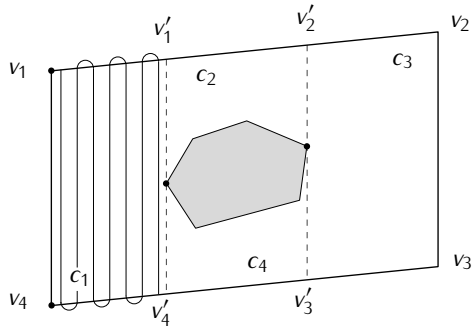


Figure 5.11. The boustrophedon-like motion covering the cell c_1, composed of back and forth parallel lines and circles connecting of radius half the ideal coverage distance.

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 $\mathcal{Q}$ 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 φ_5, φ_6, ... 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 $\varphi_1, \varphi_3, \varphi_5, \dots$) is a given constant. Then the radius in the plan (the circles $\varphi_2, \varphi_4, \varphi_6, \dots$) 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 propose the Zamboni-like motion. The Zamboni motion is in some literature for fixed-wing aerial robots (Ablavsky and Snorrason, 2000; Araújo et al., 2013; Majeed and S. Lee, 2019). The name of this coverage motion comes from the hockey arenas' ice maintenance machines (Ablavsky and Snorrason, 2000; Araújo et al., 2013; Dille and Singh, 2013), which have a large turning radius like the aerial robots we study. 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 applied to our scenario with variable size of the coverage, enabling the possibility of future coverage replanning in Section 5.3.1. We illustrate the Zamboni-like motion for c_1 in Figure 5.12 (whereas the boustrophedon-like motion in Figure 5.11).

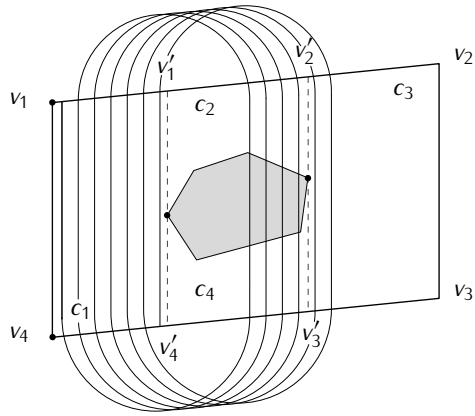


Figure 5.12. The Zamboni-like motion to cover the cell c_1. It is similar to the boustrophedon-like motion in Figure 5.11 but travels distant lines first, respecting the large turning radius constraint of, e.g., the fixed-wing aerial robots.

In Problem 2.5.2, we assumed the aerial robot can overfly the NIZs. The intuition is that although the robot has a further computations constraint specifying that it cannot perform any computation. For ease of notation, let us 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 NIZs 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 (2.13–2.14) for the paths, and Equation (2.15) 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 NIZs o (H. Choset, E. Acar, et al., 2000).

5.3 Energy-Aware Coverage Replanning and Scheduling

In this section, we solve Problem 2.5.1, finding the optimal control in the model in Section 4.3.2, the configuration of the path the aerial robot is flying c_i^ρ, and computations it is computing c_i^σ. The path configuration is relative to the coverage (we built the paths for a coverage tour in Section 5.2); the computations configuration in the precision agriculture use case to the hazard detection (we discussed the computations running on the robot in the use case in Section 2.6.2).

Generally, the exact final time instant t_l, when the aerial robot reaches the final point $\mathbf{p}(t_l) = \mathbf{p}_{\Gamma_l}$ in the last stage Γ_l, is unknown. We hence derive the optimal control on a finite horizon rather than the entire time frame $[t_0, t_l]$ with MPC or receding horizon predictive control (RHPC), an optimal control technique 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 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 with no a priori knowledge of t_l (Camacho and Alba, 2007). It forecasts the system behavior–the energy evolution–and optimizes the forecast to derive a control action (Rawlings et al., 2017). There have been many MPC developments in optimal control literature. 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).

In detail, we derive an approach that uses the model in Section 4.3 in the cost and constraints of an optimal control problem (OCP) subsequently solved with MPC. Similar approaches are being studied in modern aerial robotics literature. These include studies that apply MPC in 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). Planning-scheduling energy awareness is also the object of study in the context of optimal control on a finite horizon (Ondrúška et al., 2015; W. Zhang and J. Hu, 2007), along with various optimization techniques (Brateman et al., 2006; Lahijanian et al., 2018). A detailed discussion of the state-of-the-art is in Chapter 3; for planning-scheduling in Section 3.5.

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 Section 4.2 and derive the OCP in Sections 5.3.2–5.3.3. In Section 5.3.4, we dig further into the implementation discretizing the OCP, and finally, in Section 5.3.5, we discuss an algorithm for the coverage replanning and scheduling.

5.3.1 Output model predictive control

The literature commonly refers to the problem of applying MPC to a system with an unknown state $\mathbf{q}$ as output MPC. A variation where estimates $\hat{\mathbf{q}}$ refine the state of a perfect model via state estimation 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 Equation (4.10) in Section 4.3, state estimation uses filtering techniques that include Kalman filtering (Kalman, 1960; Simon, 2006).

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 Sec-

tion 4.3.2, we presented a motivation for such control based on empirical data. The nominal control 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 increase/decrease the quality of the coverage?). To this end, the algorithm that we propose in Section 5.3.5 uses the change in : *(a)* path parameters to verify whenever a given path configuration can be done with the current state of charge (SoC) of the battery, *(b)* 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 flight termination), 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 conducted an early analysis in this direction (Seewald, Schultz, Ebeid, et al., 2021a), 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 discuss the observation in Section 4.3.2 relative to the coverage path visually, showing how a change in the coverage affects the energy. We discussed the energy effect of the change of computations on power in Section 4.3.2. It is due to different schedules on the computing hardware. We observe a decrement in the power consumption intuitively by running at lower frames per second (FPS) rate (Seewald, García de Marina, Midtiby, et al., 2020; Seewald, Schultz, Ebeid, et al., 2021a; Zamanakos et al., 2020). 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 $\varphi_1, \ldots, \varphi_4$ of the Zamboni-like motion in Section 5.2.2. Figure 5.12 showed the coverage with the highest configuration of parameter $c_{4,1}$. If we assume that the time needed to travel 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

$$t_{\overline{c}_{4,1}} = 7(2t_1 + t_2 + t_3) + t_1. \tag{5.8}$$

Conversely, in Figure 5.13, we assume that the time needed to travel $\varphi_4(\underline{c}_{4,1})$ is t_4; then the time needed to cover c_1 with configuration $c_{4,1} = \underline{c}_{4,1}$ is

$$t_{\underline{c}_{4,1}} = 3(2t_1 + t_2 + t_4) + t_1. \tag{5.9}$$

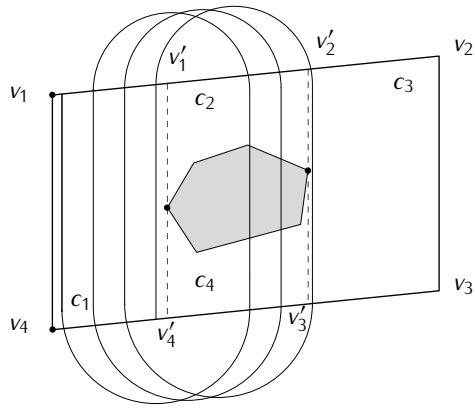


Figure 5.13. The Zamboni-like motion to cover the cell c_1 with the lowest configuration of parameter $c_{4,1}$ relative to the radius of the circle φ_4.

It is clear that $t_4 < t_3$ 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 remainder 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.

5.3.2 Re-evaluation of the output constraint

The model output in Equation (4.10) is the power (instantaneous energy consumption) that we stated earlier evolves in $y \in \mathbb{R}$. Nevertheless, the power drainable from a battery at any time instant is limited. Hence, we redefine the original output constraint to include the battery model in Section 4.2. To this end, we consider SoC b of the mobile robot's battery with the model in Equations (4.5–4.8) using an equivalent electrical circuit

$$\dot{b}(y(t)) = -k_b \left(V - \sqrt{V^2 - 4R_r y(t)} \right) / (2R_r Q_c), \tag{5.10}$$

where k_b is an additional battery coefficient determined experimentally, and the values of V, R_r, and Q_c are detailed in Equations (4.5–4.8). In summary, we derived Equation (5.10) in Section 4.2 from the literature on the battery SoC in Section 3.2, using the "Rint" equivalent electrical circuit (He et al., 2011; Hinz, 2019; Mousavi G. and Nikdel, 2014).

Using the expression to estimate SoC in Equation (5.10), 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

$$0 \leq y(t) \leq b(y(t)) Q_c V, \tag{5.11}$$

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 (5.10).

Definition 5.3.1: Output constraint.

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

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

We assume the aerial robot carries a sensor to obtain the initial SoC $b(y(t_0))$ in the output constraint. This is a realistic assumption. Aerial robots often have a flight controller, which returns various metrics. The current SoC can be estimated from, e.g., the voltage. Evaluating the constraint requires numerical simulation: the battery model in Equation (5.10) is differential, similarly to the energy dynamics of the periodic model in Equation (4.10). We compute the numerical simulation using, e.g., the Euler or the Runge-Kutta methods (Iserles, 2009). In Algorithm 5.3, we use the Euler method.

We use the output constraint set in an OCP formulation of Problem 2.5.1 in the next section.

5.3.3 Derivation of an optimal control problem

An OCP that selects the highest configuration of parameters (control) at each time step while respecting all the constraints we defined so far can be expressed

$$\max_{\mathbf{q}(t), c_i(t)} l_f(\mathbf{q}(T), T) + \int_{t_0}^{T} l(\mathbf{q}(t), c_i(t), t)\, dt, \tag{5.12a}$$

$$\text{s.t. } \dot{\mathbf{q}} = f(\mathbf{q}(t), c_i(t), t), \tag{5.12b}$$

$$c_i(t) \in \mathcal{U}_i, \mathbf{q}(t) \in \mathbb{R}^m, \tag{5.12c}$$

$$y(t) \in \mathcal{Y}(t), \tag{5.12d}$$

$$\mathcal{S}_{i,j} := \{0\}, \forall j \in [\sigma] \text{ when } \mathbf{p}(t) \notin \mathcal{Q}^v, \tag{5.12e}$$

$$\mathbf{q}(t_0) = \hat{\mathbf{q}}_0 \text{ given (last estimated state), and} \tag{5.12f}$$

$$b(y(t_0)) = b_0 \text{ given,} \tag{5.12g}$$

where constraints in Equations (5.12b–5.12e) 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 trajectory w.r.t. an inertial navigation frame $\mathcal{O}_W$. The sizes of the state and control (m and n) are defined in Sections 4.3.1–4.3.2. By solving the OCP in Equation (5.12), we want to derive the trajectory

$$c_i^*(t) = \{c_{i,1}^*(t), \ldots, c_{i,\rho}^*(t), c_{i,\rho+1}^*(t), \ldots, c_{i,\rho+\sigma}^*(t)\}, \tag{5.13}$$

for a given stage i in Definition 2.3.1. In the max term in Equation (5.12), we further derive the trajectory of the ideal state $\mathbf{q}(t)$, which we use to derive the optimal control and, e.g., to evaluate the model's fidelity against future measured data. Indeed the solution to the OCP evolves the model from an estimate of the state at the initial time instant t_0 in Equation (5.12f). At the very

beginning of the optimization (when t_0 corresponds to the initial time instant of the flight), we will see in Section 6.3.1 that the perfect model evolution in Equation (5.12b) does not correspond to the data despite converging later on. The constraints contain the control and state constraint from Equation (2.6), the output constraint from Definition 5.3.1, and the dynamics with the ideal state evolution from Equation (4.10) in Equations (5.12b–5.12d). The OCP further contains the coverage constraint from Section 2.6.1 and Section 5.2.2 in Equation (5.12e). Although the aerial robot can overfly the NIZs and the polygon edges, it cannot compute any computation. Formally, we inhibit the computations setting their constraint to $\{0\}$ when the aerial robot is flying over the ith NIZ 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 (5.12b) is then the periodic model in Equation (4.10) together with the scale transformation from Section 4.3.3

$$f(\mathbf{q}(t), c_i(t), t) = A\mathbf{q}(t) + B\mathrm{diag}(\nu_i)(c_i(t) - c_i(t - \Delta t)), \tag{5.14}$$

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). The scaling factors for the first ρ path parameters are given in Equation (4.48) and for the remaining σ computations parameters in Equation (4.49).

The instantaneous cost function is the quadratic expression

$$l(\mathbf{q}(t), c_i(t), t) = \mathbf{q}'(t)Q\mathbf{q}(t) + c_i'(t)Rc_i(t), \tag{5.15}$$

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) and some solution guarantees (Beck, 2014).

The final cost function is also a quadratic expression but with no control (Rawlings et al., 2017)

$$l_f(\mathbf{q}(T), T) = \mathbf{q}'(T)Q_f\mathbf{q}(T), \tag{5.16}$$

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 derives the 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 $b(y(t))$ (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 (5.16) for the time instant T ($t = t_0 + N$). The dynamics constraint satisfaction requires to evolve the perfect model f in Equation (5.14) over horizon $[t_0, t_0 + N]$ beginning from the last estimated state $\mathbf{q}(t_0) = \hat{\mathbf{q}}_0$ at time instant t_0. Similarly, the output constraint satisfaction requires then to evolve the battery model b in Equation (5.10) over $[t_0, t_0 + N]$, from $b(y(t_0)) = b_0$, the last battery measurement at instant t_0.

The OCP in Equation (5.12) is infinite-dimensional, being the system dynamics in Equation (5.12b) and the battery dynamics in Equation (5.10) given in continuous- and not discrete-time, and the cost integrated over the interval. Such an infinite-dimensional OCP has infinite

dimensions 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 5.3.4.

5.3.4 Coverage replanning and scheduling: Discretization

In this section, we discretize the infinite-dimensional OCP in Equation (5.12), which result we can subsequently solve in Section 5.3.5. There are different techniques for this purpose (Grüne and Pannek, 2017; Rawlings et al., 2017); some relevant to our approach are the direct methods already employed in many MPC-related applications (Rawlings et al., 2017), which include the single and multiple shooting methods. They parametrize the control and state trajectories with a finite-dimensional vector and derive a nonlinear program (NLP) (Rawlings et al., 2017). Both the methods compute the 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, including a popular method called the direct collocation method, 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).

A possible NLP derived from discretizing the OCP in Equation (5.12) is then

$$\max_{\mathbf{q}(k), c_i(k)} \; l_f(\mathbf{q}(T), T) + \sum_{k \in \mathcal{K}} l_d(\mathbf{q}(k), c_i(k), k), \tag{5.17a}$$

$$\text{s.t. } \mathbf{q}(k+h) = f_d(\mathbf{q}(k), c_i(k), k), \tag{5.17b}$$

$$c_i(k) \in \mathcal{U}_i, \mathbf{q}(k) \in \mathbb{R}^m, \tag{5.17c}$$

$$y(k) \in \mathcal{Y}(k), \tag{5.17d}$$

$$\mathcal{S}_{i,j} := \{0\}, \; \forall j \in [\sigma] \text{ when } \mathbf{p}(k) \notin \mathcal{Q}^v, \tag{5.17e}$$

$$\mathbf{q}(t_0) = \hat{\mathbf{q}}_0 \text{ given (last estimated state), and} \tag{5.17f}$$

$$b(y(t_0)) = b_0 \text{ given}, \tag{5.17g}$$

where the constraints in Equations (5.17b–5.17e) are evaluated now on a finite interval $k \in \mathcal{K} = \{t_0, t_0 + h, t_0 + 2h, \ldots, T\}$, 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 (5.12).

We use numerical simulation to transform Equation (5.12) into Equation (5.17) (Iserles, 2009) the instantaneous cost function

$$l_d(\mathbf{q}(k), c_i(k), k) = h\, l(\mathbf{q}(k), c_i(k), k), \tag{5.18}$$

where l is given in Equation (5.15).

The discrete dynamic evolution in Equation (5.17b)

$$f_d(\mathbf{q}(k), c_i(k), k) = A_d \mathbf{q}(k) + B \text{diag}(\nu_i)(c_i(k) - c_i(k-h)), \tag{5.19}$$

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

$$A_d = (hA + \mathrm{diag}(1, 1, \ldots, 1)), \tag{5.20}$$

where $\mathrm{diag}(1, 1, \ldots, 1) \in \mathbb{R}^{m \times m}$ 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 (5.14).

To discretize the battery dynamics in Equation (5.10), we use

$$b_d(y(k + h)) = b(y(k)) + hb(y(k + h)), \tag{5.21}$$

where b is given in Equation (5.10) 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 (5.17), we transformed the OCP in Equation (5.12) into an NLP by first discretizing and thus effectively implementing the direct multiple shooting method (Rawlings et al., 2017). 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).

Practical implementation requires a rational choice of h in Equations (5.18–5.21), weighting precision with execution time. For instance, to speed the derivation of the configuration in Section 6.3, we use different hs for Equations (5.19–5.21) and the set $\mathcal{K}$ along with the cost in Equation (5.18).

5.3.5 Coverage replanning and scheduling: Algorithm

Finally, Algorithm 5.3 provides the optimal configuration of path and computations parameters and thus computes the coverage replanning and scheduling. It inputs the initial plan Γ along with the initial time step and a guess for the energy model $\hat{\mathbf{q}}(t_0)$, and outputs the revised plan, deriving the configuration of parameters c_i^* in Equation (5.13). 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.

It 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 time necessary 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, 14, 15, and 22) 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 scope of this work. With a small enough h, Euler method that we use in Equations (5.18–5.21) 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 that are among the most popular methods for numerical simulation (Atkinson

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

foreach $i \in \{t_0, t_0 + h, t_0 + 2h, \dots\}$ do
  if $\mathbf{p}(i) \neq \mathbf{p}_{\Gamma_l}$ then
    return Γ
  if $\mathbf{p}(i) = \mathbf{p}_{\Gamma_j}$ then
    $j \leftarrow j + 1$
    if $j \notin [n]_{>0}$ then
      $j \leftarrow 1$
      $\varphi_1, \varphi_2, \dots, \varphi_n \leftarrow$ shift $\varphi_1, \varphi_2, \dots, \varphi_n$ of $\mathbf{d}$
      $\mathbf{p}_{\Gamma_1}, \mathbf{p}_{\Gamma_2}, \dots, \mathbf{p}_{\Gamma_n} \leftarrow$ shift also $\mathbf{p}_{\Gamma_1}, \mathbf{p}_{\Gamma_2}, \dots, \mathbf{p}_{\Gamma_n}$ of $\mathbf{d}$
  $\mathbf{q}(\mathcal{K} \setminus \{i + N\}), c_j(\mathcal{K}) \leftarrow$ 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 (5.17) on $\mathcal{K} = \{i, i + h, \dots, i + N\}$
  $k \leftarrow i$
  while $b_d(C\mathbf{q}(k)) > 0$ do
    if $k \notin \mathcal{K}$ then
      $\mathbf{q}(k + h) \leftarrow A_d \mathbf{q}(k)$
    $b_d(C\mathbf{q}(k+h)) \leftarrow b_d(C\mathbf{q}(k)) - h k_b \left(V - \sqrt{V^2 - 4 R_r C \mathbf{q}(k + h)} \right) / (2 R_r Q_c)$
    $k \leftarrow k + h$
  $t_b \leftarrow k - i$
  $t_s \leftarrow (\mathrm{diag}(v_j^\rho) c_j^\rho(i) + \tau_j^\rho)[\overbrace{1 \quad 1 \quad \cdots \quad 1}^{\rho}]$
  $t_r \leftarrow (t_s / \bar{t})(\bar{t} - i)$
  if $t_r < t_b$ then
    $c_j^\rho(i) \leftarrow$ find c_j^ρ with $t_c \in [0, t_b]$, otherwise take $\underline{c}_j^\rho$
  $\hat{\mathbf{q}}(i + h) \leftarrow A_d \hat{\mathbf{q}}(i) + B \mathrm{diag}(v_j)(c_j(i) - c_j(i - h))$
  $P(i + h)^- \leftarrow A_d P(i) A_d' + S(i)$
  $K(i + h) \leftarrow (P(i + h)^- C') / (C P(i + h)^- C' + V(i))$
  $\hat{\mathbf{q}}(i + h) \leftarrow$ compute $\hat{\mathbf{q}}(i + h) + K(i + h)(y(i) - C\hat{\mathbf{q}}(i + h))$ from energy sensor $y(i)$
  $\hat{y}(i + h) + C\hat{\mathbf{q}}(i + h)$
  $P(i + h) \leftarrow (I + K(i + h)C) P(i + h)^-$
  $\mathbf{p}(i + h) \leftarrow$ compute from position $\mathbf{p}(i)$, velocity $v(i)$ with $\Delta_d \varphi_j$ in Algorith 5.2

Algorithm 5.3. Coverage replanning and scheduling algorithm

et al., 2009). These methods use quadrature, a procedure that replaces the integral with a finite sum (Iserles, 2009). We use a Runge-Kutta method inside the `powprofiler` tool in Section 4.2.2 and the Euler method in Algorithm 5.3.

On Lines 2–3, the algorithm verifies if 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. On Lines 4–9, it 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 $\mathbf{d}$ when it reaches $\mathbf{p}_{\Gamma_n}$ (otherwise it ignores Lines 8–9; the shift is zero). It then selects the energy-aware configuration of computations on Line 10 by solving the NLP in Equation (5.17) derived from the OCP in Equation (5.12) using the multiple shooting method. 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 the 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, e.g., six and ten seconds in Section 6.3, meaning the algorithm evolves the models in Equation (5.14) and Equation (4.10) for ten future seconds and selects the control trajectory c_j^* that minimizes the costs in Equation (5.18) and Equation (5.16). 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 (5.17), the algorithm evaluates the constraints in Equations (5.17b–5.17e) (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 we mentioned in Section 5.3.4 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 $\mathcal{K}$. Practically, we implement the MPC on Line 10 using a software framework for nonlinear optimization termed CasADi (J. A. Andersson et al., 2019; J. Andersson et al., 2012a,b), and a popular NLP solver called IPOPT (Wächter and Biegler, 2006).

On Lines 22–27, the algorithm estimates the state $\mathbf{q}$ of the periodic model in Section 4.3, with Kalman filtering 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 of $P(t_0) \in \mathbb{R}^{m \times m}$, the covariance error of $\mathbf{q}(t_0)$, and of the trajectories of $S \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}$, the covariances of the state and output noise.

On Line 24, the algorithm computes the gain with V (Simon, 2006). To ease the computations, we keep the covariances fixed in our experimental setup in Section 6.3.

On Lines 11–17, the algorithm estimates the time needed to completely drain the battery with the differential model in Equation (5.21), using the Euler method for numerical simulation . 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 do not have any trajectory for the control available and thus 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. On Line 18, the algorithm then computes the scale transformation from the path parameters domain to the time domain from Section 4.3.3, and on Line 19, the estimated remaining time. It selects a different configuration of path parameters when the battery time is lower than the remaining time on Line 21. 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 Summary

This chapter detailed the derivation of a guidance action for a coverage plan, coverage plan itself, and energy-aware configuration of both the path and computations, solving the coverage and planning problems. The guidance uses a path-following gradient descent algorithm in the literature, further adapted to the coverage planning and scheduling. The plan is indeed composed of multiple path functions that switch at specific triggering points. The algorithm exploits the theory of vector fields. It points to the path corresponding to the current coverage stage at each location in space. The coverage integrates a known method named boustrophedon decomposition, deriving a fixed-wing friendly covering tour with the Zamboni-like motion. Both the coverage and the schedule are varied within specific limits to alter the energy and power by, e.g., changing the distance between the coverage lines or the detection rate. The chapter thus proposes an optimal control-based algorithm for such replanning. The algorithm uses MPC and Kalman filtering and exploits the computations, motion, and battery models from the previous chapter to derive the best trajectory of both path and computations variations. In the next chapter, we show the algorithms both using numerical and Paparazzi flight controller-based simulations.

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, García de Marina, Midtiby, et al., 2020; Seewald, Schultz, Ebeid, et al., 2021a; Seewald, Schultz, Roeder, et al., 2019; Zamanakos et al., 2020) and to appear in a forthcoming study (Seewald, García 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 battery 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.

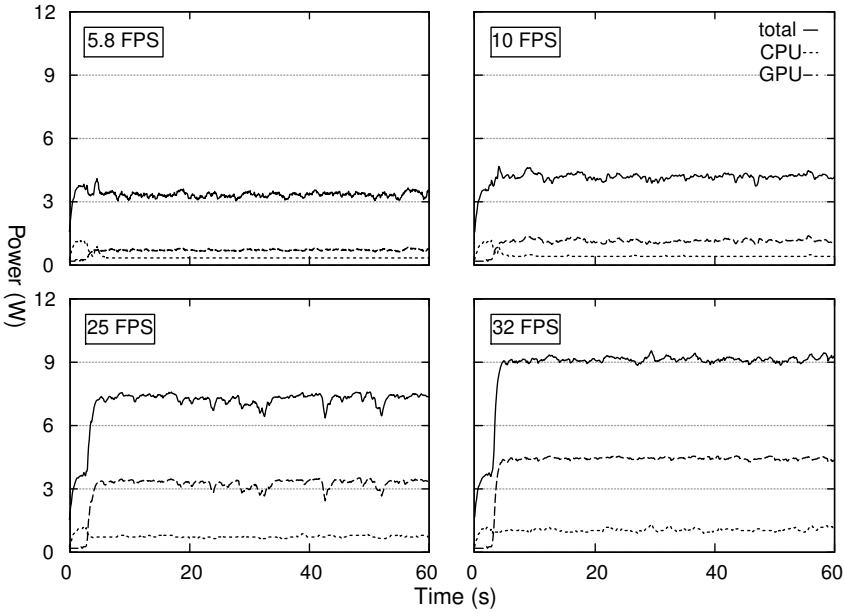


Figure 6.1. The `darknet-gpu` computation measurement layer models, featuring the GPU implementation of the YOLO (Redmon, Divvala, et al., 2016) deep neural network library modified to introduce delays between detections. From top left in horizontal order to bottom right, the figure shows the schedules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

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., 2021a): the power evolution in the function of time for three measuring devices, CPU, GPU, and overall of the NVIDIA Jetson TX2 board. The 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).

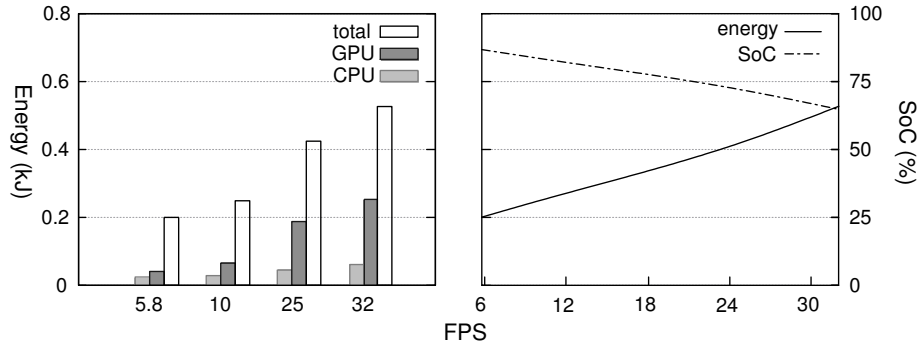
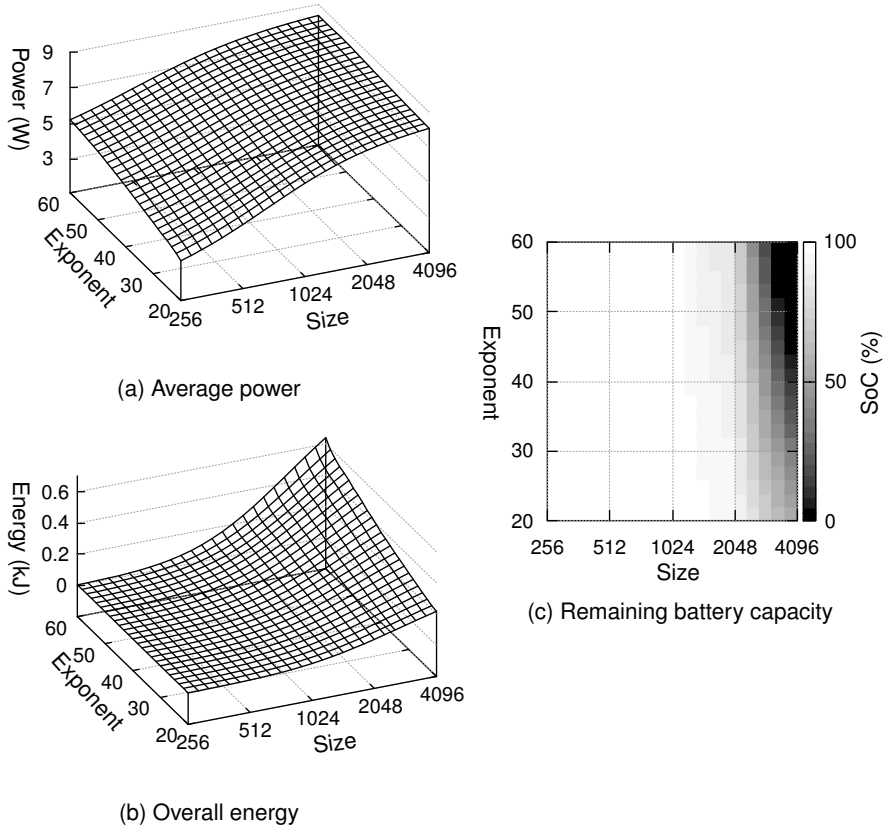


Figure 6.2. Per-minute energy consumption and SoC of the `darknet-gpu` computation in terms of CPU, GPU, and overall energies. On the right is the resulting predictive layer that shows the energy (same scale as on the left) and battery SoC in the function of any possible configuration. The figure appeared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

In an initial iteration of our work (TeamPlay Consortium, 2019a), 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. 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 4.2.1. 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.



(a) Average power

(b) Overall energy

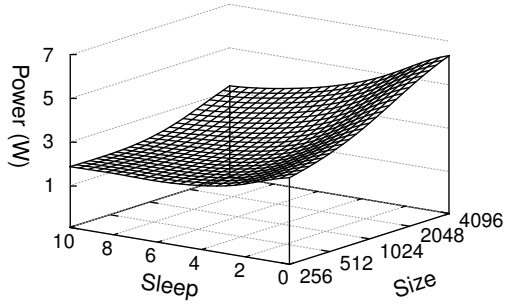
(c) Remaining battery capacity

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., 2021a).

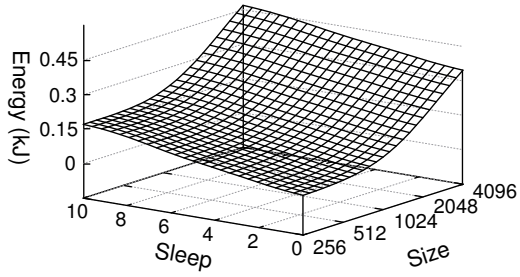
6.1.2 The `matrix-gpu` computation

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., 2021a). 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 6.3 (these values enclose

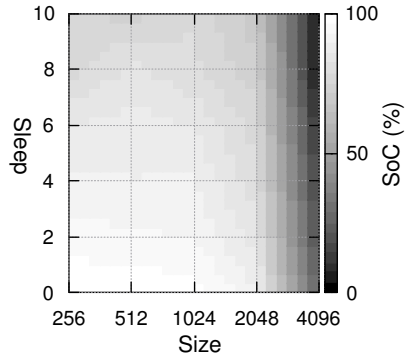
the sets $\mathcal{S}_{i,1}$ and $\mathcal{S}_{i,2}$ respectively). Particularly, Figure 6.3a shows the average power, Figure 6.3b the overall energy, and Figure 6.3c the battery SoC as a function of matrix size and exponent. We report the average power consumption independently of the running time of 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., 2021a), also visible in Figure 6.1. It is the reason why there is a notable difference in average power for two opposite configurations.



(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., 2021a).

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,

Computation	ODROID XU3	NVIDIA TK1	NVIDIA TX2	NVIDIA 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., 2021a).

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 6.3–6.4, 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` computations 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 less 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 (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, Schultz, Ebeid, et al., 2021a). 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., 2021a; 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., 2021a).

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., 2021a). 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 significant 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., 2021a). 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., 2021a).

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., 2021a). 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 conclude that the tool's effect on the power is marginal (Seewald, Schultz, Ebeid, et al., 2021a). We later saw the same effect with other 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., 2021a). 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., 2021a). 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., 2021a), justifying ours against another model in the literature.

6.2 Case Studies in Motion Energy Modeling

In this section, we discuss case studies of 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, García 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 while exploiting the model in Equation (4.9) for the energy signal, which we studied in our previous work (Seewald, García 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, n.d.[b]) 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 4.8–4.9) 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., 2021a), where per-component energy is modeled in a dataflow computational network, decreasing the modeling effort (Seewald, García 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, García 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 6.8a 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, García de Marina, Midtiby, et al., 2020). The phase depends on different conditions, including landing site geologic conformations,

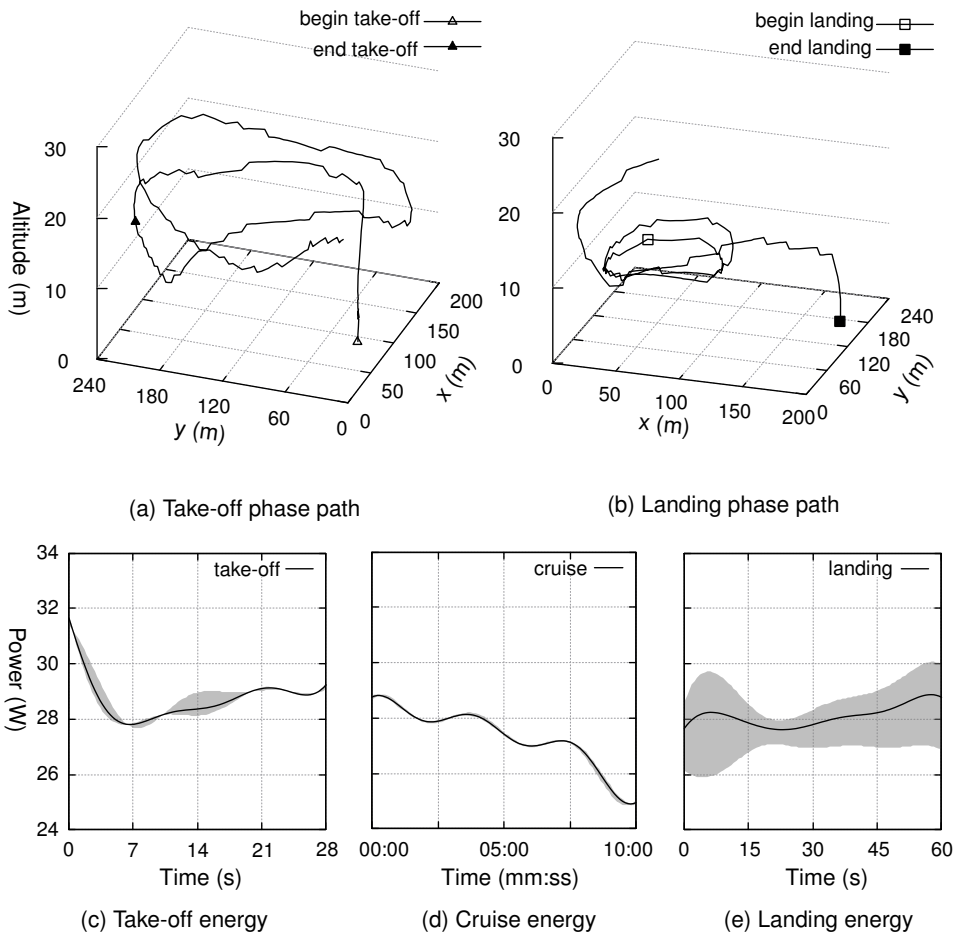
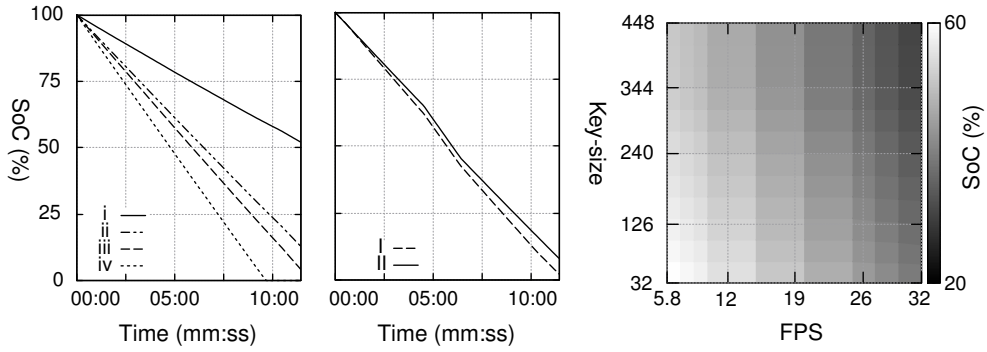


Figure 6.5. Paths and modeled energy evolutions in time for different flight phases as they appeared in our early study (Seewald, García de Marina, Midtiby, et al., 2020).

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 Figures 6.5c–6.5e, i.e., the energy impact of the motion on the battery; ii a static schedule of flying



(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, García de Marina, Midtiby, et al., 2020).

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, García 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

The second case study is the same aerial robot and agricultural scenario from Section 6.2.1, but now exploiting the model in Equation (4.10) along with different computing hardware and computations.

Experimental setup

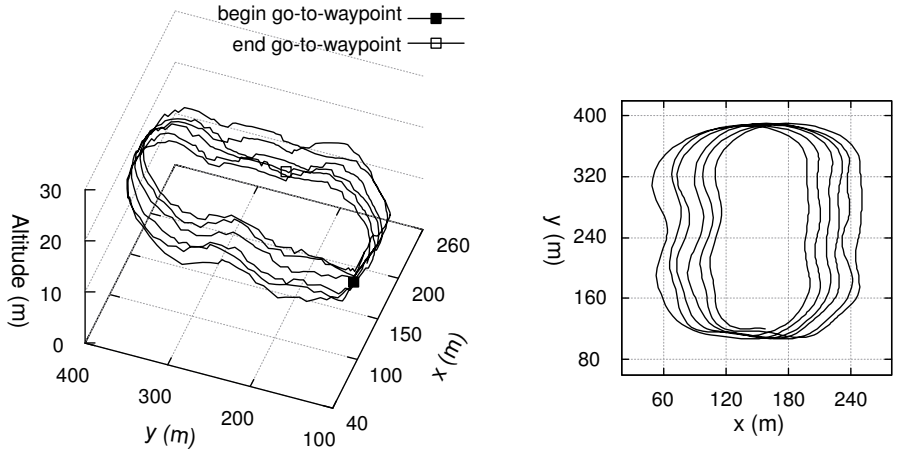
The aerial robot is the Opterra fixed-wing for CPP in an agricultural scenario equipped with the Paparazzi flight controller. The computing hardware is now the NVIDIA Jetson Nano, implementing the computations with the ROS middleware. In this use-case, we evaluated some initial experiments of the differential 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 and various power-saving schedules (empirically demonstrated in the next section) but instead evaluated the initial feasibility of the periodic differential model.

The computing hardware implements three computations: `ssd-mobilenet` computation SSD-MobileNet V2 convolutional neural network (CNN) (Sandler et al., 2018), whereas `pednet` computation PedNet fully convolutional network (FCN) (Ullah et al., 2018). The third compu-

tation is termed `sender`, which sends the eventual detections on the ground using the technical standard for wireless communication 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, simplifying the modeling effort.

Motion energy evaluation

The coverage path is to be seen in Figure 6.7a; 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. Figure 6.7b is then the path from above. There is a slight deviation



(a) 3D path

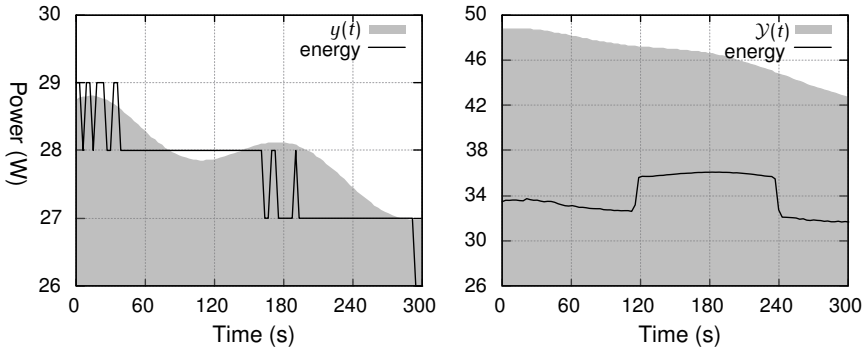
(b) 2D path with altitude zero

Figure 6.7. Paths for the cruise phase in the second case study with the Opterra fixed-wing aerial robot flying the Zamboni-like motion.

mostly on the x-axis, which we attribute to the atmospheric conditions. Initially, we counted the period approximately at one-fourth of the one we proposed in Section 2.6.1 (begin and end of go-to-waypoint), supposing this would be optimal to model the periodicity of the energy signal. We later corrected this assumption 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, and we will see the results in the next section. The overall energy assessment for what concerns the motion is then in Figure 6.8a, and for the motion and computations energies in Figure 6.8b. For the latter, our early model consisted of 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 as in 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)



(a) Modeled motion energy (b) Coupled motion and computations energy

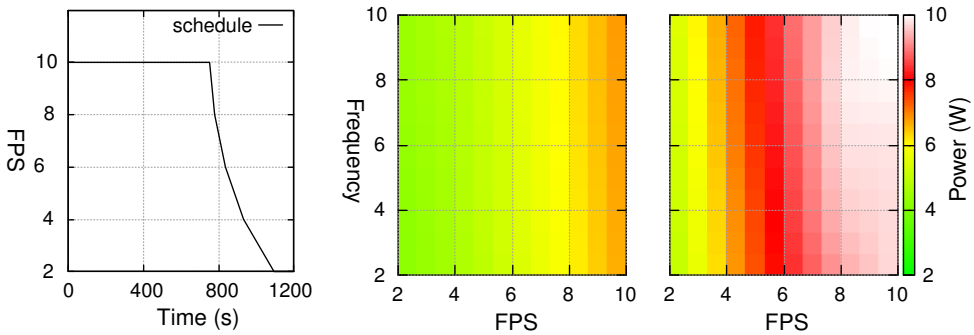
Figure 6.8. Modeled energy evolution with the differential model of the second case study, corresponding to the cruise path in Figure 6.7.

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 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.

Computations energy evaluation

For the computations `ssd-mobilenet` and `pednet`, the parameter $c_{i,1}$ alters the 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, as opposed to 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 frequency at which the detections are sent to the ground, also from two to ten hertz (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 terms of battery SoC are in Figure 6.9b for the `ssd-mobilenet` computation, and in Figure 6.9c for the `pednet` computation (both the layers vary then the frequency $c_{i,2}$ for the `sender` computation). We use the colors here to underline the energy impact of some configurations, otherwise not visible in grayscale.

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 6.9a and the battery SoC. The figure shows how different the detection rate ranges, 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 6.8b, coupling the motion and computations energies–the black line in the figure. The



(a) Schedule over time (b) `sender,ssd-mobilenet` (c) `sender,pednet`

Figure 6.9. Computations energy models in term of SoC as well as the schedule over time for the second case study.

output constraint in Definition 5.3.1 is how much energy is available from the battery 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.

6.2.3 Assessment

In this section, we discussed two case studies of the aerial robot flying a static coverage without replanning. We propose computations models corresponding to the agricultural use case for detections and ground communication, reporting their energy efficiency in terms of power and battery SoC. The first case study exploits the model in Equation (4.9) to approximate the energy of different flight phases, motivating the periodic analysis. The second case study then proposes initial experiments of the differential periodic energy model in Equation (4.10) with some additional computations implemented in the ROS middleware we use in the remainder. We expand the model using variations of paths and computations with replanning in the next section.

6.3 Coverage Planning and Scheduling

In this section, we report the results for energy-aware coverage planning and scheduling; and thus prove our approach experimentally. We extensively use the constructs from the previous chapters, including the energy models in Chapter 4, the guidance, coverage, and replanning from Chapter 5, and demonstrate both the solutions to Problems 2.5.2–2.5.1 in Chapter 2. Indeed, in the previous sections, we justified the various energy models and discussed merely the solution to Problem 2.5.2 with the Zamboni-like motion. Specifically, in Section 6.3.1, we provide results for numerical simulations derived with MATLAB (R) . In Section 6.3.2, we integrate such simulations with the

popular Paparazzi flight controller (Paparazzi, n.d.[b]), and finally, in Section 6.3.3, we extend the previous results to the case of avoiding the computations over the no-interest zones (NIZs) and out of the polygon.

6.3.1 Numerical simulations

The first results report is that of a realistic simulation implemented in MATLAB (R) relative to our forthcoming work (Seewald, García de Marina, and Schultz, n.d.), where we implement the craft's flight dynamics. It involves the Opterra fixed-wing aerial robot in the precision agriculture scenario we introduced in Section 1.3. It does both the CPP and scheduling and online in-flight replanning energy-wise. The agricultural field is delimited by vertices v and forms $\mathcal{Q}^v$ (see Section 5.2.1 for the notation). There are no NIZs, i.e., $\mathcal{Q}^v = \mathcal{Q}$.

Experimental setup

The aerial robot in this result is the simulated Opterra fixed-wing for CPP in an agricultural scenario, independent of the specific flight controller. In the remainder, we discuss two subresults, each with different atmospheric conditions, I and II, in Figure 6.10. Path-wise, we replan the coverage embedding a parameter $c_{i,1}$ (same for all the stages) relative to the radius of the fourth circle φ_4 in a set of four primitive paths $\varphi_i, \ldots, \varphi_{i+3}\, \forall i \in \lfloor l/4 \rfloor$ iterated over time up to reaching a final point $\mathbf{p}_{\Gamma_l}$ that we proposed for the Zamboni-like motion in Section 2.6.1. The parameter influences the radius r_2 of the circle φ_4 in Equation (2.16); we discussed further the precise meaning of how such parameter affects the quality of the coverage in Section 5.3. The result I is that of the aerial robot flying with a constant wind speed of 5 meters per second, the wind direction of 0 degrees, and an initial path parameter $c_{i,1}$ value of 0. Result II is flying under the same conditions but a wind direction of 90 degrees and the initial path parameter value of -1000. We simulate sudden battery drops in I, expecting the replanning to occur with both scheduling and coverage shortening (i.e., lesser quality of the coverage) and an ideal battery behavior in II, expecting coverage largening (i.e., higher quality of the coverage).

Identically to the two case studies from the previous section, the simulation is extended with computing hardware: NVIDIA Jetson Nano (NVIDIA, n.d.), which runs ROS middleware. The computing hardware implements two computations via ROS nodes: the `pednet` computation detects hazards using PedNet FCN (Ullah et al., 2018) and can be scheduled with parameter $c_{i,2}$ the detection rate in FPS. The `sender` computation communicates to a ground station and can be scheduled with parameter $c_{i,3}$ the sending frequency in hertz. Both are similar to the case study in Section 6.2.2, and here we further demonstrate the scheduling with simultaneous online replanning. For the computations, we expect the scheduler to schedule the computations to respect the output constraint set in Definition 5.3.1 (i.e., the maximum battery instantaneous power).

For battery modeling, the 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 3.2 amperes per hour in Equations (4.5–4.8) in Section 4.2.1. Finally, the battery coefficient k_b in Equation (5.10)

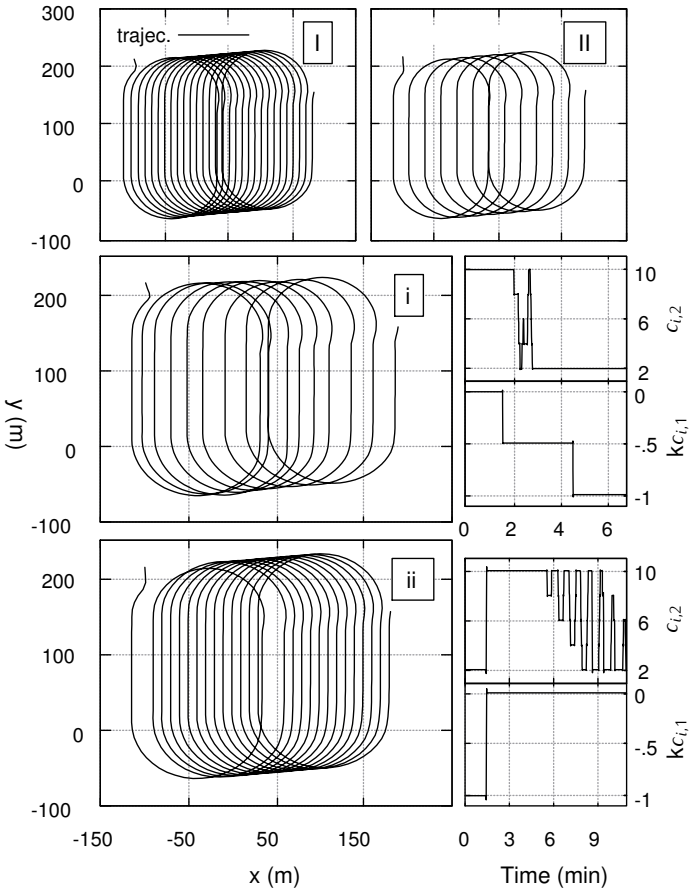


Figure 6.10. Trajectory simulations with variations of wind speed and direction. In I and II plan is static. It is replanned online with the algorithm in i and ii. The algorithm adapts the path parameter relative to r_2 $c_{i,1}$ and computation parameter FPS rate $c_{i,2}$. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

is set to 1.83/1000 and determined experimentally. The model's parameter r in Equation (4.10) is equal to 3. We motivate such a choice with the spectral analysis in Figure 4.9, showing that three frequencies are adequate to model the energy signal.

Planning evaluation

We show the effects of different conditions on the CPP in Figure 6.10. Both trajectories I and II are flying the same plan for the coverage with the Zamboni-like motion but with different initial parameters values, the highest configuration $\overline{c}_{i,1}$ and the lowest configuration $\underline{c}_{i,1}$, along slightly different atmospheric conditions. The energy models of these two trajectories are the top two sub-figures I and I in Figure 6.11. The simulated energy data are in black the model evolution is in red. The model evolves for two hundred seconds with no information about the real energy: the

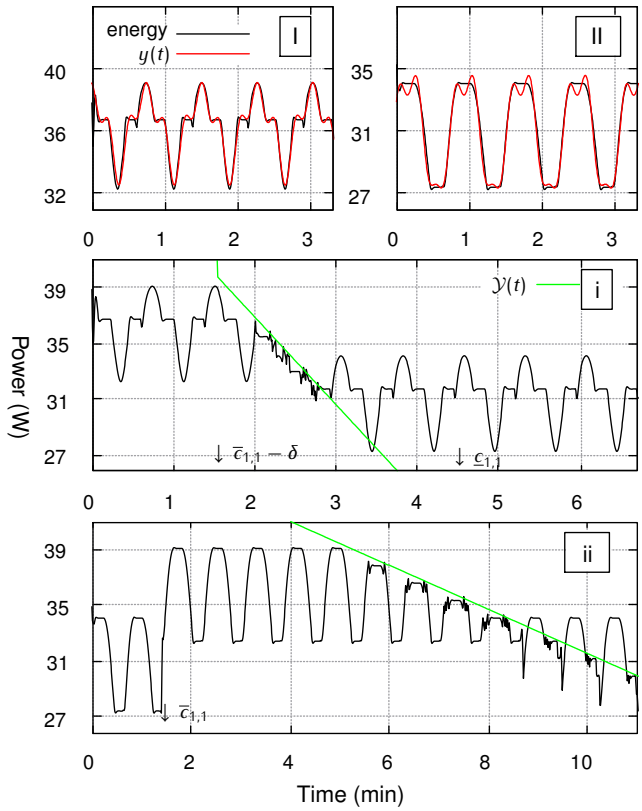


Figure 6.11. The energy models of the trajectories from Figure 6.10 for 200 seconds against the simulated data (I and II). Below are the energy evolutions with replanning (i and ii). Here, the algorithm replans the plan when the final time and battery time do not match and the computation when the battery is discharged. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

model is estimated earlier with, e.g., Kalman filter on Lines 22–27 in Algorithm 5.3, whereas now it merely evolves without further estimation on Line 14. The figure proves the model accuracy experimentally, i.e., the red line copies the periodic evolution of the energy signal.

The components of the energy model from Section 4.3.1 are illustrated in Figure 6.12. Left sub-figures are the initial slices of the models and period estimation (the latter we defined in Definition 2.4.3). On the right of Figure 6.12, we show the states $\alpha_0, \ldots, \beta_3$ evolution in time, concluding that approximately two periods are sufficient to obtain consistent state estimates. With non-periodic signals, we observed that the estimator on Lines 22–27 in Algorithm 5.3 estimates primarily the first state α_0 and it neglects the others. It hence approximates the non-periodicity with a linear model.

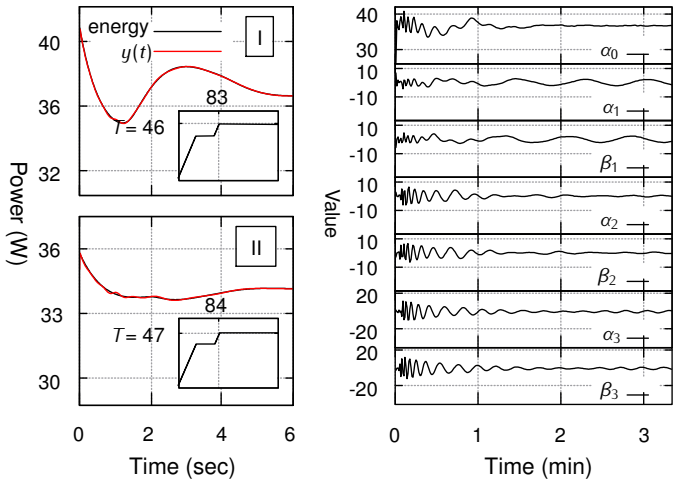


Figure 6.12. Energy estimation for the first 6 seconds on the left side, the evolution of the state $\mathbf{q}$ on the right. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

Replanning evaluation

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 $\nu_{i,1}$, $\tau_{i,1}$ from Equation (4.48). We compare the remaining flight time with the time needed to completely deplete the battery from Equation (5.10) on Lines 11–17 in Algorithm 5.3 and reduce or increase the parameter to optimize the battery time. The path parameter $c_{i,1}$ is equal for all the stages. It changes the radius of the first circle in the current period and therefore shifts the other paths accordingly. The change is in Figure 2.8, resulting in a shorter distance between the coverage lines in the Zamboni-like motion and increment or reduction of the flying time. The path constraint set is set to $\underline{c}_{i,1} =$ -1000 and $\overline{c}_{i,1} =$ zero (enclosing the constraint set $\mathcal{C}_{i,1}$), equal for all the stages.

A variation of computations directly affects the power. We thus select the highest computation which satisfies the constraints in Definition 5.3.1 on Line 10 in Algorithm 5.3, using model predictive control (MPC). We observed a low effect on the power of the communication ROS node (see Figure 6.9c); we thus used a static frequency of ten hertz for the parameter $c_{i,3}$. Nevertheless, the detection node varies between 5 and 10 watts for the lowest and highest FPS rate. We implemented the FPS rate parameter $c_{i,2}$ with factors $\nu_{i,2}$, $\tau_{i,2}$ mapping $c_{i,2}$ to the data from `powprofiler` as we described in Equation (4.49) in Section 4.3.3. The computation constraint set is set to $\underline{c}_{i,2} =$ two and $\overline{c}_{i,2} =$ ten (enclosing the constraint set $\mathcal{S}_{i,1}$), equal for all the stages.

Coverage planning and scheduling

We have tested the validity of our approach via a numerical simulation implemented in MATLAB (R) , showing the replanning in Figure 6.10 in i and ii path-wise and in Figure 6.11 in i and ii energy-wise. For the first sub-result I, the plan starts at the highest configuration of

parameters. We simulated two unexpected battery drops at approximately one minute and a half and four minutes and a half. The algorithm optimizes the path (downward facing arrows in Figure 6.11) in the proximity of the drops to ensure that the flight is completed. Moreover, it maximizes the parameter $c_{i,2}$ when the battery is discharging (green line), respecting the output constraint in Definition 5.3.1. We simulated the opposite scenario for sub-result II. The plan starts at the lowest configuration of parameters while the battery behaves ideally. We note that the path parameter increases when the algorithm estimates enough data (two periods T). The algorithm further optimizes the computation parameter w.r.t. to the battery discharge rate. For both cases, we used discrete reductions δ of five hundred and two for $c_{i,1}$ and $c_{i,2}$ respectively, and the horizon N equal to six seconds. We observed a slight increment in overall replanning time (order of seconds) by using a horizon of ten seconds–attributable to the MPC component on Line 10 of Algorithm 5.3–but did not detect a notable increment in precision.

6.3.2 Paparazzi flight controller

This section extends the previous, describing the experimental setup and the result for the implementation of our approach in the New Paparazzi Simulator (NPS), an advanced simulator with sensors models along with JSBSim (Berndt, 2004), an open-source flight dynamics model (FDM) (Paparazzi, n.d.[a]). It is also relative to our forthcoming work (Seewald, García de Marina, and Schultz, n.d.). It involves the simulated craft in CPP and hazards detection from the previous section under more diverse battery conditions, with a further complication of asynchronous communication between the planner-scheduler and the flight controller. Again, there are no NIZs $\mathcal{Q}^v = \mathcal{Q}$.

Experimental setup

The polygon v is described by the set of vertices

$$v_1 := (5, 139.5),\ v_2 := (384.4, 162.4),\ v_3 := (384.4, 32.4),\ v_4 := (5, 9.5), \tag{6.2}$$

and is regular. On the contrary to the previous result, the outcome depends on the specific flight controller, meaning we simulate the actual interactions between the Paparazzi flight controller (Paparazzi, n.d.[b]) and the computing hardware on-board the aerial robot in some simulated flights using the NPS with JSBSim. We implement planning-scheduling in Python open source programming language and the communication between NPS and planner-scheduler with a software bus termed Ivy (*Ivy lightweight software bus* n.d.). The resulting NPS graphical user interface for one of the flights is in Figure 6.14. The path and computations are analogous to these that we proposed in the previous result in Section 6.3.1. The battery model parameters, the constraint sets $\mathcal{C}_{i,1}$ and $\mathcal{S}_{i,2}$, and the factors ν_1, ν_2, τ_1, τ_2 all have the same values in these two results. We replan the coverage with varying parameter $c_{i,1}$ that alters the distance between the survey lines and schedule the computations with the parameter $c_{i,2}$ relative to the FPS rate of `pednet`. We keep the parameter $c_{i,3}$, the communication frequency of the `sender` computation, static, motivated by the early results showing low energy effect of different schedules (see

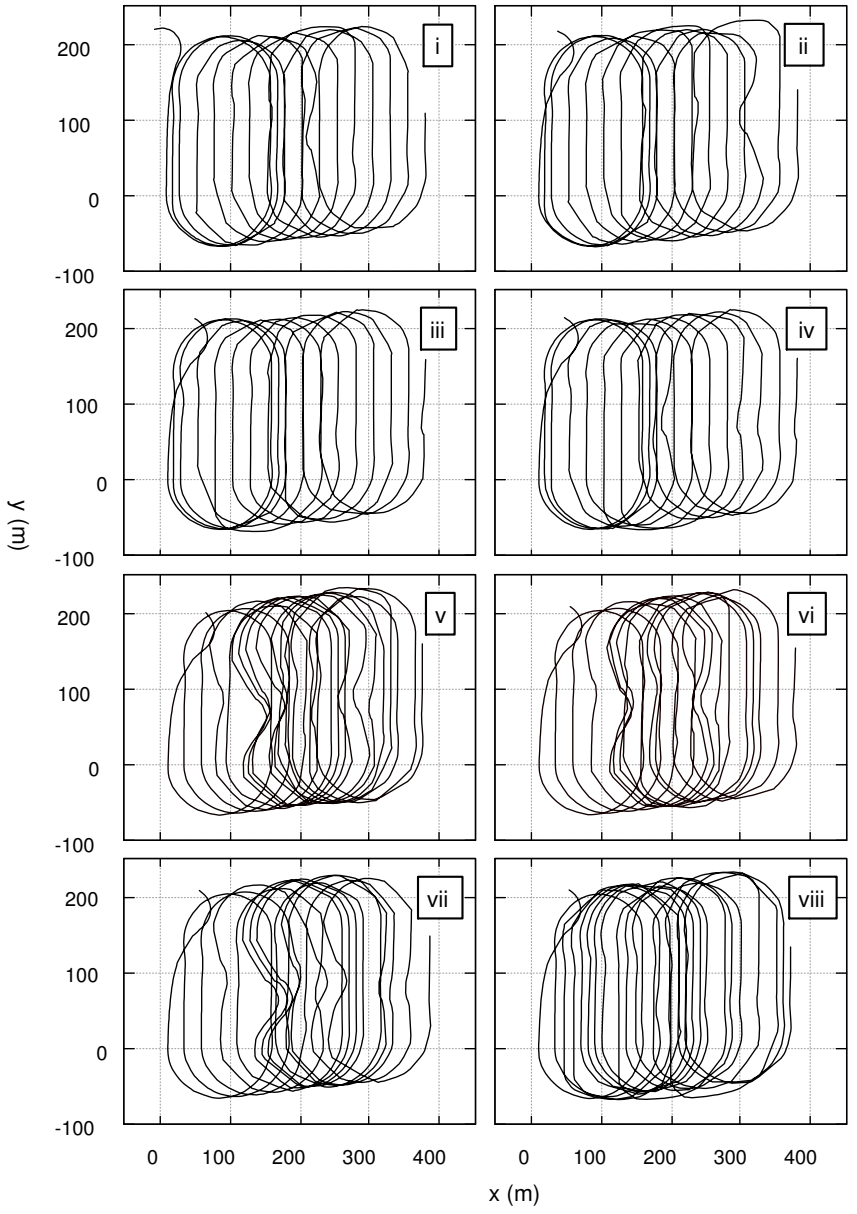


Figure 6.13. Trajectories of flight under various conditions and initial configurations implemented in New Paparazzi Simulator. The first cluster of flights i–iv starts with the highest configuration of parameters, the second cluster v–viii with the lowest. Similarly to Figure 6.13, different battery conditions apply to each flight resulting in different replannings. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

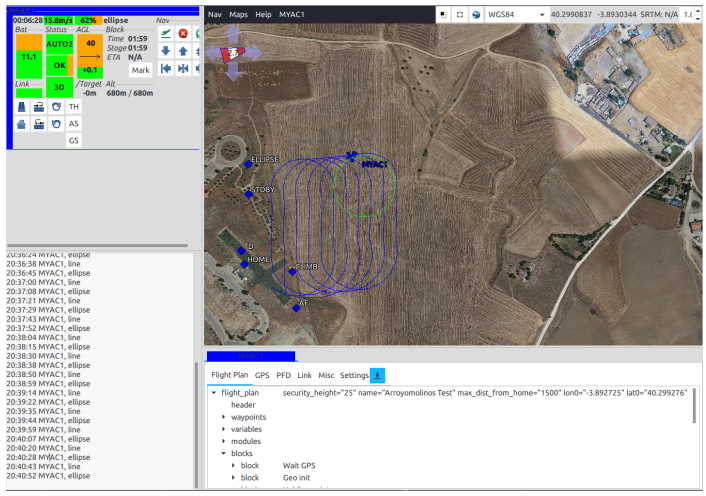


Figure 6.14. New Paparazzi Simulator flying the coverage planning with energy-aware replanning, implementing the JSBSim flight dynamics model.

Figure 6.9c). The two computations run on the NVIDIA Jetson Nano computing hardware using ROS middleware, extending the simulation. The schedule for $c_{i,2}$ is varied via a ROS topic, whereas the computations are ROS nodes, one implementing PedNet FCN (Ullah et al., 2018), the other simply communicating with the ground.

Planning evaluation

We performed eight test flights in two clusters. The trajectories of these flights are in Figure 6.13. A first cluster i–iv compromises flights starting with the highest configurations of parameter $c_{i,1} = \underline{c}_{i,1}$ and $c_{i,2} = \underline{c}_{i,2}$, requiring then replanning due to various simulated battery profiles. A second cluster v–viii starts on the contrary with the lowest configuration of $c_{i,1} = \underline{c}_{i,1}$ and $c_{i,2} = \underline{c}_{i,2}$. The evolution of the parameters is then in Figure 6.15. For coverage planning, we further observe some unexpected flight patterns in ii, v–vii, which are, in part, caused by the delay between the flight controller and planner-scheduler. Indeed these often happen when the planner-scheduler evaluates the final battery time t_b on Lines 11–17 in Algorithm 5.3, being these lines computationally expensive. Nonetheless, we observe a minor incidence in a multi-thread implementation, with each computationally expensive block (i.e., MPC on Line 10 and battery estimator on Lines 11–17) computing in a separate thread, compared to simple sequential execution.

In Figure 6.10, the reference frame $\mathcal{O}_W$ is different along with the flight time. The latter is due to the different velocities of the craft. Indeed we solely rely on the values from FDM rather than our implementation of the dynamics. The former is due to different initial conditions. Although we generated the same coverage, $\mathbf{p}(t_0)$ are different: practically, we shift the paths that form the plan.

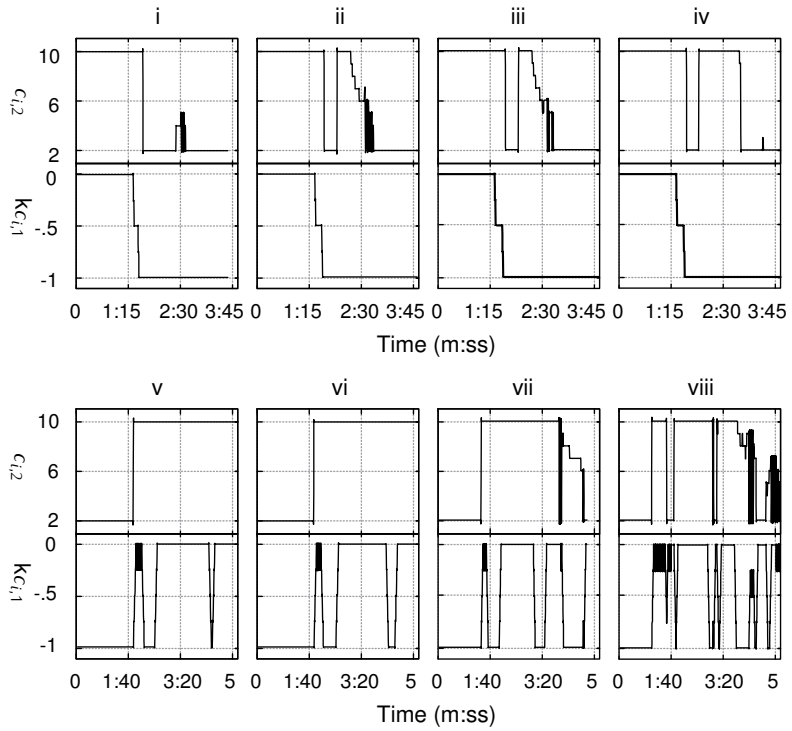


Figure 6.15. Evolution of the parameters configurations for two clusters of flights implemented in New Paparazzi Simulator. Each has a similar overall flight time due to the initial conditions. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

Replanning evaluation

All the flights start with a charged battery but have different battery evolutions and initial conditions. For the first cluster of flights, flight i starts with the highest configuration of $c_{i,1}$ and $c_{i,2}$, and the battery behaves linearly up to 90%. At approx. 95 seconds we simulate a sudden battery drop of 4%. The battery then again behaves linearly, and the final battery state is approx. 57%. The parameter $c_{i,1}$ is thus replanned by Algorithm 5.3 in the proximity of the drop and the computation parameter $c_{i,2}$ is scheduled to respect the output constraint in Definition 5.3.1 visible in Figure 6.16 (the green line indicates the constraint). Flight ii is similar, but the battery drop happens at approx. 122 seconds. The evolution of the parameter $c_{i,1}$ is indeed mostly the same, whereas the parameter $c_{i,2}$ is replanned with a higher schedule. Flight iii has slightly different atmospheric conditions and eventual additional uncertainty due to the communication lag between the flight controller and scheduler. We can see that indeed ii–iii are very similar, conversely to flight iv, where the battery drop happens at approx. 163 seconds.

The second cluster of flights exhibits the same tendency for flights v–vi and, indeed, the energy in Figure 6.16, the configurations in Figure 6.15, and the trajectories in Figure 6.13 are almost identical. There are no simulated sudden battery drops and enough battery SoC to respect the

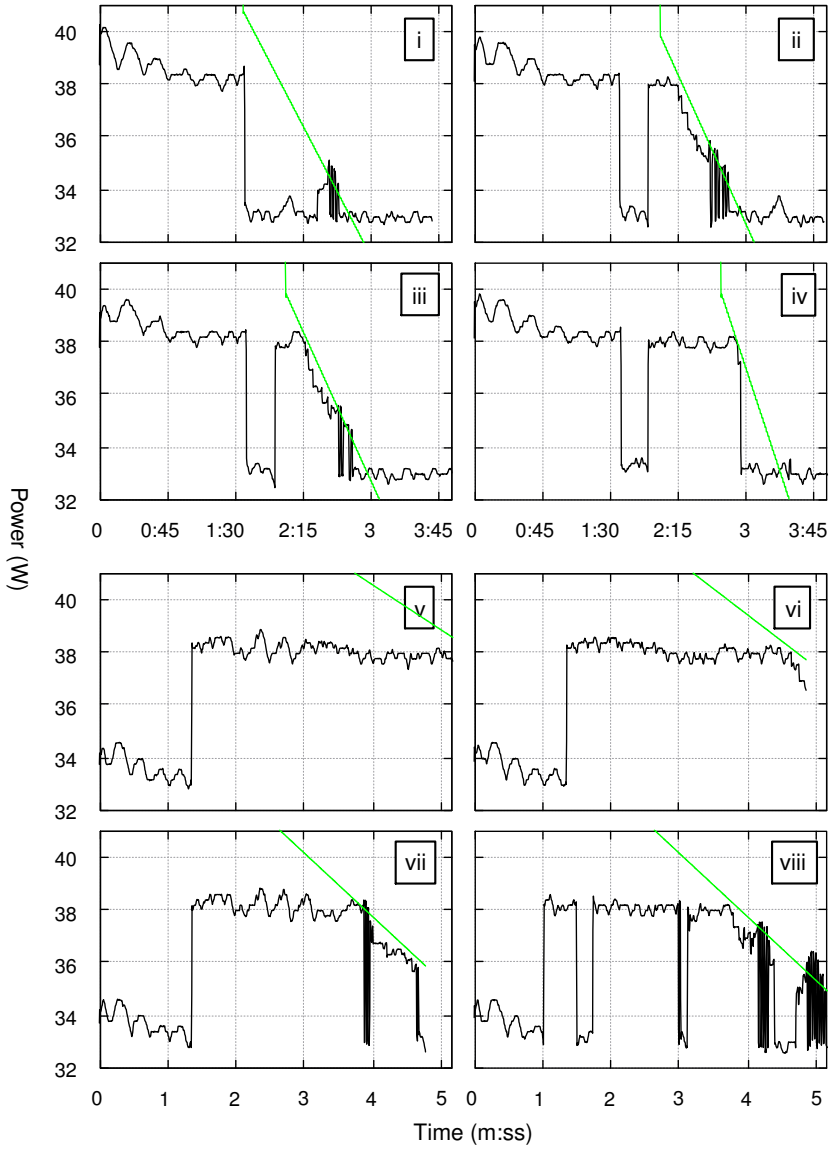


Figure 6.16. The energy models for the flights implemented in New Paparazzi Simulator with different initial and simulated battery conditions. The green line indicates the output constraint. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

output constraint in Definition 5.3.1. Flights vii–viii are similar to the previous couple, except for a more steep simulated battery evolution, lower of approx. 4%. There is then a simulated 2% points battery difference at the end of the flight between vii and viii, requiring additional replanning.

Coverage planning and scheduling

We have further validated our approach utilizing NPS with JSBSim as FDM. We simulated various battery evolutions and drops, corresponding to, e.g., imperfect batteries of fixed-wing aerial robots operating in real-world conditions. We have simulated the interaction between the flight controller and the planner-scheduler, interconnecting the two via Ivy bus, showing possible complications such as delays. We will discuss future possibilities of additional optimizations to reduce such delays in Chapter 7. In this result, we use a lower reduction δ for the path parameters of two hundred fifty, with no notable difference in path replanning. We thus conclude that δ of five hundred from Section 6.3.1 is adequate for path replanning. Conversely to the previous result, we use a fixed horizon N of ten seconds.

6.3.3 Coverage with no-interest zones

In this final result, we complement our approach, interrupting the computations over a NIZ on the ground as we hypothesized in Problem 2.5.2 and Section 5.2, and additionally out of the polygon v, thus computing only within the space $\mathcal{Q}^v$ in Section 2.6. The result is relative to our forthcoming work (Seewald, García de Marina, and Schultz, n.d.). The NIZ resulting polygon o_1 mimics conveniently the shape and location in Figures 5.6–5.13 in Sections Sections 5.2–5.3, whereas v is the same as in Section 6.3.2. The polygon o_1, representing the NIZ, has six vertices

$$\begin{aligned} o_{1,1} &:= (110, 85), \quad o_{1,2} := (125, 106.5), \; o_{1,3} := (155, 114.5), \\ o_{1,4} &:= (195, 93.5), \; o_{1,5} := (175, 76.5), \quad o_{1,6} := (137, 57.7), \end{aligned} \tag{6.3}$$

and the resulting trajectory of the aerial robot flying the coverage while avoiding the computations over NIZ and out of the polygon is in Figure 6.17. To simulate the energy effect of the configuration $c_{i,2} = \{0\}$, we evaluated the average power of the NVIDIA Jetson Nano computing hardware with the `powprofiler` tool, solely running ROS core with the remaining computations (`pednet`, `sender`) switched off. The rest of the experimental setup is shared with Section 6.3.2, and the resulting initial and battery conditions are these of flight ii in the first cluster in Figure 6.13, i.e., the highest configuration of $c_{i,1}$ and $c_{i,2}$ with the simulated battery behaving ideally but for a drop at approx. 122 seconds.

The trajectory is indeed similar to ii. Flight ii.a in Figure 6.17 is that of the coverage while avoiding computations over $\mathcal{Q}^{o_1}$, whereas flight ii.b avoids the computations everywhere but over $\mathcal{Q}^v$ (i.e., additionally to ii.a, it avoids the computations outside of the polygon v). There is an observable difference in the configuration trajectories for the parameter $c_{i,2}$, visible in Figure 6.19. Indeed ii.a in Figure 6.19 is similar to ii in Figure 6.15, but for the instants when the aerial robot flies over the NIZ o_1 and the computation parameter is inhibited $c_{i,2} = \{0\}$. The corresponding energy consumption is then in ii.a in Figure 6.18.

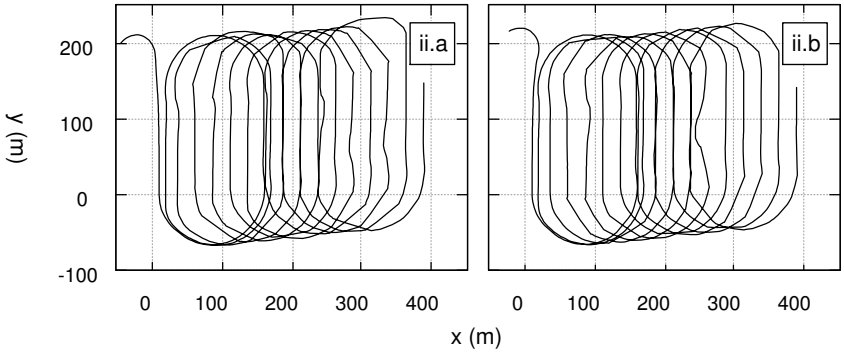


Figure 6.17. Trajectories of two flights flying the same condition as ii in 6.13 but inhibiting computations over NIZ in ii.a, and over NIZ and out of the polygon v in ii.b. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

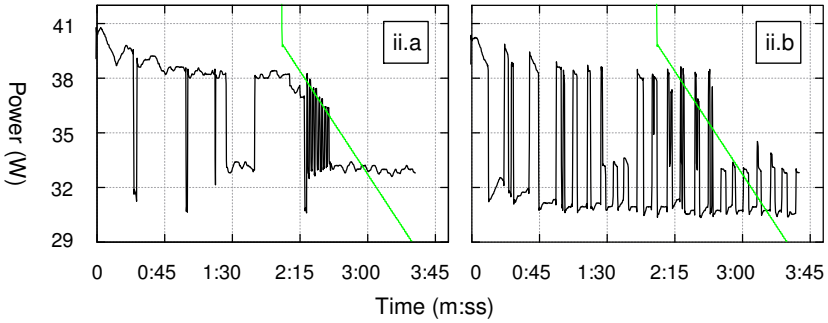


Figure 6.18. Energy evolution of a schedule that additionally inhibits the computations over NIZ in ii.a, and over NIZ and out of the polygon v in ii.b, similarly to Figures 6.17–6.19. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

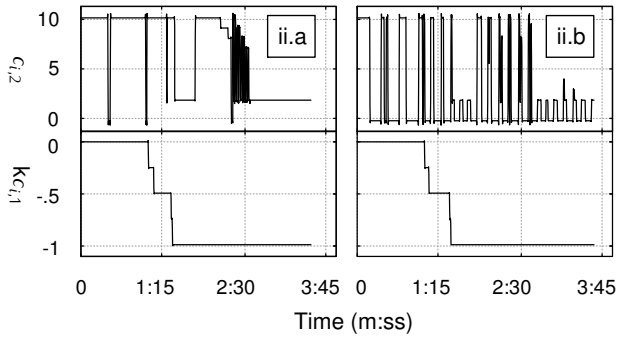


Figure 6.19. Configurations trajectories of a schedule that additionally inhibits the computations over NIZ in ii.a, and over NIZ and out of the polygon v in ii.b, similarly to Figure 6.17. The figure is to appear in our forthcoming study (Seewald, García de Marina, and Schultz, n.d.).

Finally, there is a notable difference between the two sub-results ii.a and ii.b in Figures 6.19–6.18, contrary to Figure 6.17, showing the energy evolution (and configuration trajectory) effect of avoiding computations on o_1 and avoiding computations everywhere except over the interest space $\mathcal{Q}^v$. One can further note the impact of the NIZ polygon o_1 shape on the energy and configuration trajectories. Depending on the position in space, it takes longer or shorter to fly over the NIZ due to the different lengths of a line segment parallel to the edge ${}^{v_4}|_{v_1}$ intersecting the NIZ o_1 (see Figure 5.8).

6.4 Summary

This chapter described the experimental setup and results for computations and motion energy and battery models, two case studies of the aerial robot flying a static coverage, and coverage planning and scheduling. The computations energy models were derived via the `powprofiler` modeling tool and reported the overall energy, average power, and battery SoC per each computations configuration within a bounded search space. The chapter detailed computations energy models for several computations ranging from simple matrix exponentiation to object detection. It then provided two case studies of aerial robots flying physical experiments with a static coverage, showing the periodic energy evolution and motivating the differential energy model from the previous chapters. The guidance, coverage planning, and online replanning are then proposed via numerical simulations with various realistic conditions, showing the energy effect of online replanning from the lowest and the highest initial configurations of the coverage and detection. These results are then further extended with the popular Paparazzi flight controller and various battery conditions and additionally extended to the case of flight with NIZs. Here the aerial robot inhibits the computation when flying over a NIZ of a given size and out of the survey polygon.

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

PLANNING-SCHEDULING energy awareness demonstrates improved robustness to in-flight failures in the autonomous use case, mitigating the adverse effect of environmental interferences on battery life and overall flight span. In the previous chapters, we progressively built and showcased our approach using the use case both in simulation and some early flight of the Opterra fixed-wing autonomous aerial robot. The approach varies both the path and the schedule while exploiting differential periodic, computations, and battery models. In this chapter, we summarize our work, discuss the outcomes, and propose future directions and conclusions in Sections 7.1–7.4.

7.1 Summary

The approach in this work provides an energy-optimal path and a power-saving schedule altogether. Past studies often derive one of these aspects, whereas the analysis of the interactions of the two is less common (Brateman et al., 2006; Sudhakar et al., 2020). To this end, our work focus on coverage path planning (CPP)–a common problem in the planning literature where it is required to visit each point in a given space (H. Choset, 2001; Galceran and Carreras, 2013)–in a precision agriculture use case. Here, a fixed-wing aerial robot covers a given agricultural field, detects ground hazards, and communicates with other ground-based actors. Although use-case specific, the approach is generic in terms of computations and battery modeling, and CPP. The guidance and differential periodic energy modeling can be applied to a broader class of energy-constrained autonomous mobile robotics use cases.

To derive energy-aware coverage planning and scheduling for autonomous aerial robots, we defined some basic constructs in Chapter 2. These included the concepts of computations and motion along with their energies in Section 2.1. Indeed their close interaction is the very basis of this work. Computations are some energy-expensive computational tasks we aim to schedule energy-wise along with the motion on a coverage path. The latter is a succession of continuous and twice differentiable path functions in Section 2.2. These functions are wrapped with computations in stages in Section 2.3, along with parameters for scheduling and coverage replanning. The succession of stages forms a plan in Section 2.4, and the progression from one stage to another happens in the proximity of triggering points. The plan can be expressed as a mere succession of stages or using primitive paths with a shift. In Section 2.5, we formally defined the problems of coverage planning and energy-aware coverage replanning and scheduling.

We discussed the state of the art spanning broadly in Chapter 3. In Sections 3.1–3.2, we saw the literature for computations energy and battery modeling. We focused on energy models for heterogeneous elements, involving CPUs and GPUs separately, and on abstract battery models. We then examined motion planning for mobile and aerial robots in Sections 3.3–3.4, whereas the planning-scheduling interactions in the robotics literature in Section 3.5.

In Chapter 4, we provided energy models for future energy estimations of computations, motions, and batteries before digging into the technicalities of coverage planning and energy-aware replanning. The computations energy model in Section 4.1 models heterogeneous elements (i.e., CPU and GPUs) of the computing hardware and provides overall energy, average power, and battery state of charge (SoC) in the function of a software configuration. It utilizes a two layers architecture, where the top layer incorporates multiple bottom layers. It derives an automatic modeling and profiling tool named `powprofiler` (Seewald, Schultz, Ebeid, et al., 2021a,b), which derives a battery model in Section 4.2 from an abstract model in the literature termed "Rint". We proposed a differential periodic energy model for the motion in Section 4.3, exploiting energy characteristics of the coverage from some empirical observations. The motion energy model includes the computations energy via the control, modeling the energy effect of different schedules.

We discussed the coverage planning and scheduling, energy-aware replanning, and the guidance on a coverage path are in Chapter 5, providing the solution to the original problems in Chapter 2. First, in Section 5.1, we discussed how to guide the aerial robot given a set of paths in the plan using the theory of vector fields. We derived a coverage plan that visits all the points in a given space in Section 5.2, using cellular decomposition and a modified coverage motion for fixed-wings with variable coverage that we termed Zamboni-like motion. We included all the past constructs in Section 5.3 for energy-aware coverage replanning and scheduling. Here, we replanned the original path using the computations and motion energy along with the battery models. We used a modern optimal control and state estimation technique robust to uncertainty.

Finally, in Chapter 6, we discussed the experimental setup and results. In Section 6.1, we provided the `powprofiler` computations energy models. We saw case studies of motion and periodic energy models in Section 6.2, and finally, discussed the results for energy-aware coverage planning and scheduling in Section 6.3.

7.2 Outcomes

This works contributes to the existing robotics literature on planning-scheduling energy awareness (Brateman et al., 2006; Lahijanian et al., 2018; Mei, Y.-H. Lu, Y. C. Hu, et al., 2005; Mei, Y.-H. Lu, Y. C. Hu, et al., 2006; Ondrúška et al., 2015; Sadrpour et al., 2013a,c; Sudhakar et al., 2020; W. Zhang and J. Hu, 2007) by proposing an approach for energy-aware coverage planning for autonomous aerial robots.

Specific contributions in order as they appear in the previous chapters include: *(a)* the introduction of the concept of replanning a variable CPP along with schedule (Seewald, García de Marina, and Schultz, n.d.), *(b)* the formal definition of no-interest zones (NIZs) -aware schedules for further energy savings by selectively running computations only when necessary (Seewald, García de Marina, and Schultz, n.d.), *(c)* the derivation of an automatic modeling and profiling tool named `powprofiler` (Seewald, Schultz, Ebeid, et al., 2021a) for various computing hardwares (Seewald, Schultz, Ebeid, et al., 2021b) that derives power, energy, and battery SoC in the function of software configuration, *(d)* the extension of the `powprofiler` tool to dataflow computational networks (Seewald, Schultz, Roeder, et al., 2019) and to the robot operating system (ROS) middleware (Zamanakos et al., 2020), *(e)* the adaptation of the "Rint" equivalent electrical circuit in the literature into the `powprofiler` tool (Seewald, Schultz, Ebeid, et al., 2021a) and the coverage replanning problem for future SoC estimations (Seewald, García de Marina, and Schultz, n.d.), *(f)* the derivation of a differential periodic energy model (Seewald, García de Marina, and Schultz, n.d.) based on empirical observations (Seewald, García de Marina, Midtiby, et al., 2020) for future energy predictions and energy predictions under variable schedules for autonomous coverage, potentially applying to a broader class of mobile robotics use cases (Seewald, 2020), *(g)* the adaptation (Seewald, García de Marina, and Schultz, n.d.) of an existing approach for guidance (García de Marina et al., 2017) based on vector fields to the case of coverage plans composed of multiple paths, *(h)* the introduction of the Zamboni-like motion for variable coverage (Seewald, García de Marina, and Schultz, n.d.) for aerial robots with strict turning radius constraints, and *(i) the derivation of an algorithm for replanning of the coverage and scheduling inflight and under strict energy constraints* (Seewald, García de Marina, and Schultz, n.d.).

7.3 Future Directions

Multiple observations and open questions regarding possible future studies were made in the previous chapters. These include further developments spanning from coverage planning to broad planning-scheduling considerations.

We hypothesized a possible generalization to other mobile robots. Indeed CPP in the literature applies to diverse use cases. These include decontamination, mine clearance, oceanographic mapping, cleanup, and others (H. Choset and Pignon, 1998) and involve various mobile robots (Galceran and Carreras, 2013). Although the Zamboni-like motion in Section 5.2.2 suits best aerial robots with strict turning radius constraints, the concept of variable coverage can be applied

broadly. For instance, the boustrophedon-like motion for variable coverage in Section 2.6.1, to rotary-wings aerial robots and other mobile robots.

In our past brief work (Seewald, 2020), we discussed the approach's applicability to the planetary exploration context. An unexplored terrain can be covered with variable coverage by trading a greedy technique with more complex planning. Schedules can vary by, e.g., pattern detections of features of interest or perception (Ondrúška et al., 2015). Recent contributions are increasingly proposing onboard autonomy with artificial neural networks (Gankidi and Thangavelautham, 2017) and other machine learning techniques (Ono et al., 2020) for future space exploration missions such as Sample Retrieval and Lander mission (Muirhead and Karp, 2019). In the latter, the robot is to be solar-powered (Higa et al., 2019). Planning-scheduling energy awareness might pose an advantage for these systems, despite past instances caring limited onboard resources due to radiation levels and temperature changes (Bajracharya et al., 2008; Gankidi and Thangavelautham, 2017). Indeed the average motion energy reported in a recent study (Ishigami et al., 2011) decreased considerably compared to the past literature (Krotkov and Simmons, 1992).

The differential periodic energy model in Section 4.3 is generic, modeling systems energy with some degree of periodicity. However, other energy models for aperiodic use cases are equally possible. The only requirement is embedding the controls as the configuration of parameters for future energy prediction in Section 5.3. A possible model is a differential linear aperiodic model built via regressional analysis, mapping a change in the schedule to, e.g., the slope of the regression. Other examples are models that incorporate robots dynamics, that map the energy to turns, etc.

Section 2.5 mentions the optimal configuration in Equation (5.17) with costs other than the energy. For instance, it might be required to cover a given space with the fastest tour possible. Or, similarly, selectively prioritize some computations over the others. These considerations were not covered by Algorithm 5.3, which merely selects the best possible schedule w.r.t. to the given cost in Equation (5.15). In a prioritized setting, it might be then possible to utilize a stage-dependent priority list: let's call it $\mathbf{r}_i \in (0, 1] \subset \mathbb{R}^n$ where n is given in Section 4.3.2. The list can be used in the cost with the parameters, prioritizing specific computations for particular stages. In this setting, it might be possible to trade-off other non-functional properties, such as security involved in the TeamPlay project (TeamPlay Consortium, 2019b) that funded a considerable part of this work, by, e.g., increasing the latter at given "security demanding" stages.

The construct of path functions in Section 2.2 involves aerial coverage flying at a given altitude h. A more complex coverage planning with, for example, the aerial robot covering a building requires further investigation. In this latter case, Definition 2.2.1 might be extended to functions $\mathbb{R}^3 \to \mathbb{R}$, expressing simply a 3D shape of, e.g., the building to cover with a cylinder. The guidance in Section 5.1 would then adopt different strategies. For example, the introduction of multiple path functions in 2D discretely "slicing" the cylinder, or the development of the theory of vector fields further with, e.g., the magnitude of the gradient in Equation (5.1). The varying quality of the coverage can be achieved by changing the distance between the "slicing" functions.

Future directions then involve the extension of the coverage motion, the study of additional motions and their trade-offs, and the shortcomings of Algorithm 5.3. The derivation of the opti-

mal parameters configuration happens for the current stage i and assumes future stages evolve according to the energy model and the observation in Section 4.3.2. The energy of complex plans might, however, change completely at each stage and require a stage-dependent energy model. Algorithm 5.3 would necessitate a trajectory to map the time to the stage for energy predictions in this latter stage-dependent energy model.

The derivation of a coverage plan along with no-flight zones (NFZs) requires further analysis. Other than for the physical obstacles in form of, e.g., tall buildings, it might be indeed more convenient to avoid flying over a NIZ energy-wise. For instance, if the NIZ is attached to one of the vertices v. For some cases, it might be then advantageous to consider complex battery models, including details such as different battery chemistries, state of health, temperature, or C-rate. Additional rigor might lead to better modeling with other equivalent electrical circuits models such as 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) outlined in Section 5.2.1.

Further possible development is that of the derivation of an approach for planning-scheduling energy awareness with multiple robots. Here a swarm of aerial robots might be covering the agricultural field with a centralized or distributed system handling overall multi-agent planning-scheduling. The system might then integrate Algortihm 5.3 for coverage replanning by deciding a piece-wise coverage. It might even distribute the computations to optimize the battery utilization of each agent while preserving given use case constraints. Integration of various approaches might lead to additional new developments; for instance, considering the frequency and voltage from other planning-scheduling literature (Brateman et al., 2006; W. Zhang and J. Hu, 2007) in Algortihm 5.3.

These additional studies will possibly further provide insights into planning-scheduling energy awareness and expedite the research foundations of varying both aspects altogether, bridging low power computing and mobile robotics domains.

7.4 Conclusion

This section concludes our work. Even though planning-scheduling energy awareness is a relatively recent research direction, there are some contributions in the wider mobile robotics research literature. Initial studies date back to 2000–2010 and focus on varying computational aspects such as frequency and voltage, along with motional aspects such as velocity or selection of peripherals. Studies in the following decade 2010–2020, especially in the second half, dig further into computational aspects proposing various techniques for simultaneous planning-scheduling, emphasizing the possible savings. Past approaches, however, do not account for aerial robots. With this work, on the contrary, we emphasize planning-scheduling energy awareness for this latter class of mobile robotics platforms. We further exploit the battery as an energy source and the possibility of in-flight online replanning.

We derive an optimal configuration of both the path and computations, running on the heterogeneous computing hardware mounted onto the aerial robot. We use a sub-class of motion

planning where the aerial robot covers each point in a given space, varying the quality of the coverage and ground hazards detections in case of, e.g., sudden battery drops, adverse atmospheric conditions, etc. We observe the effect of replanning, with the aerial robot completing the coverage under adversities or exploiting the energy budget with ideal conditions in the function of the available battery SoC. We can back the statement in the introduction: aerial robot *are an ideal instance* of an energy-constrained system benefitting from planning-scheduling trade-offs, answering research question *(d)* in Section 1.3.3.

The derivation of such a conclusion required multiple steps. Firstly, it was necessary to predict the energy of future instances of computations, which we addressed with the `powprofiler` modeling and profiling tool, answering research question *(a)*. Knowing only the energy evolutions of computations is not enough. We thus derived a differential periodic energy model that exploits coverage characteristics and empirical observations, answering research question *(b)*, and predicts energy of variations of schedules and paths, answering *(c)*.

References

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

10. 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. 2).

11. 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. 42).

12. 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).

13. 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: *10th International Symposium on Experimental Robotics*. Springer, pp. 491–500 (cit. on p. 44).

14. Aldegheri, S., Bombieri, N., Bloisi, D. D., and Farinelli, A. (2019). “Data flow ORB-SLAM for real-time performance on embedded GPU boards”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5370–5375 (cit. on p. 56).

15. Alexey, G., Klyachin, V., Eldar, K., and Driaba, A. (2021). “Autonomous mobile robot with AI based on Jetson Nano”. In: *Future Technologies Conference (FTC)*. Springer, pp. 190–204 (cit. on p. 56).

16. 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)*. IEEE, pp. 105–108 (cit. on p. 3).

17. 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. 3).

18. Anderson, J. D. (2005). *Introduction to flight*. McGraw-Hill Higher Education (cit. on p. 3).

19. Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). “CasADi: A software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* 11.1, pp. 1–36 (cit. on p. 99).

20. Andersson, J., Åkesson, J., and Diehl, M. (2012a). “CasADi: A symbolic package for automatic differentiation and optimal control”. In: *Recent Advances in Algorithmic Differentiation*. Springer, pp. 297–307 (cit. on p. 99).

21. — (2012b). “Dynamic optimization with CasADi”. In: *51st Conference on Decision and Control (CDC)*. IEEE, pp. 681–686 (cit. on p. 99).

22. 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: *International Conference on Intelligent Robots and Systems (IROS)*, pp. 237–243 (cit. on pp. 8, 56).

23. Anguelov, D., Koller, D., Parker, E., and Thrun, S. (2004). “Detecting and modeling doors with mobile robots”. In: *International Conference on Robotics and Automation (ICRA)*. Vol. 4. IEEE, pp. 3777–3784 (cit. on p. 47).

24. Araújo, J., Sujit, P., and Sousa, J. (2013). “Multiple UAV area decomposition and coverage”. In: *Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. IEEE, pp. 30–37 (cit. on pp. 24, 44, 45, 89).

25. Arkin, E., Fekete, S., and Mitchell, J. (1993). “The lawnmower problem”. In: *5th Canadian Conference on Computational Geometry*, pp. 461–466 (cit. on p. 41).

26. 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: *12th Annual Symposium on Discrete Algorithms*. SIAM, pp. 138–147 (cit. on pp. 2, 15, 43, 46).

27. — (2005). “Optimal covering tours with turn costs”. In: *SIAM Journal on Computing* 35.3, pp. 531–566 (cit. on pp. 2, 43, 46).

28. 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. 41, 84).

29. 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. 41).

30. 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: *25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, pp. 1–9 (cit. on pp. 24, 46, 88).

31. Atkinson, K., Han, W., and Stewart, D. (2009). “Euler’s method”. In: *Numerical solution of ordinary differential equations*. John Wiley & Sons. Chap. Chapter 2, pp. 15–36 (cit. on p. 97).

32. 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: *43rd International Conference on Parallel Processing*. IEEE, pp. 371–380 (cit. on pp. 33, 59).

33. Bajracharya, M., Maimone, M. W., and Helmick, D. (2008). “Autonomy for Mars rovers: Past, present, and future”. In: *Computer* 41.12, pp. 44–50 (cit. on p. 130).

34. 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. 44).

35. Basilico, N. and Carpin, S. (2015). “Deploying teams of heterogeneous UAVs in cooperative two-level surveillance missions”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 610–615 (cit. on p. 4).

36. Beck, A. (2014). *Introduction to nonlinear optimization: Theory, algorithms, and applications with Matlab*. SIAM (cit. on p. 95).

37. 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. 38, 39).

38. Berndt, J. (2004). “JSBSim: An open source flight dynamics model in C++”. In: *Modeling and Simulation Technologies Conference and Exhibit*. AIAA, p. 4923 (cit. on p. 119).

39. Bhat, G., Gumussoy, S., and Ogras, U. Y. (2019). “Power and thermal analysis of commercial mobile platforms: Experiments and case studies”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 144–149 (cit. on p. 56).

40. 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).

41. Bouzid, Y., Bestaoui, Y., and Siguerdidjane, H. (2017). “Quadrotor-UAV optimal coverage path planning in cluttered environment with a limited onboard energy”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 979–984 (cit. on p. 46).

42. Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press (cit. on p. 80).

43. Brateman, J., Xian, C., and Lu, Y.-h. (2006). “Energy-effcient scheduling for autonomous mobile robots”. In: *IFIP International Conference on Very Large Scale Integration*. IEEE, pp. 361–366 (cit. on pp. 1, 8, 12, 23, 31, 47, 49, 91, 127, 129, 131).

44. 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. 35).

45. Bryson, A. E. and Ho, Y.-C. (1975). *Applied optimal control: Optimization, estimation and control*. Hemisphere Publishing Corporation (cit. on p. 79).

46. 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).

47. 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: *International Conference on Intelligent Robots and Systems (IROS)*, pp. 1873–1879 (cit. on p. 6).

48. 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. 8, 20, 23, 39, 44).

49. 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. 24, 40, 46).

50. 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. 35, 60).

51. Camacho, E. F. and Alba, C. B. (2007). *Model predictive control*. Springer (cit. on p. 91).

52. Canny, J. (1988a). “Constructing roadmaps of semi-algebraic sets I: Completeness”. In: *Artificial Intelligence* 37.1, pp. 203–222 (cit. on p. 86).

53. — (1988b). *The complexity of robot motion planning*. MIT press (cit. on p. 87).

54. Canny, J. F. and Lin, M. C. (1993). “An opportunistic global path planner”. In: *Algorithmica* 10.2, pp. 102–120 (cit. on p. 87).

55. 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 pp. 2, 41).

56. 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).

57. Chao, Z., Ming, L., Shaolei, Z., and Wenguang, Z. (2011). “Collision-free UAV formation flight control based on nonlinear MPC”. In: *International Conference on Electronics, Communications and Control (ICECC)*. IEEE, pp. 1951–1956 (cit. on pp. 91, 99).

58. 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 pp. 64, 131).

59. Chen, X. and Touba, N. A. (2009). “Ch. 2 - Fundamentals of CMOS design”. In: *Electronic design automation*. Morgan Kaufmann, pp. 39–95 (cit. on p. 32).

60. 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. 42).

61. Choset, H., Acar, E., Rizzi, A., and Luntz, J. (2000). “Exact cellular decompositions in terms of critical points of Morse functions”. In: *International Conference on Robotics and Automation (ICRA)*. Vol. 3. IEEE, pp. 2270–2277 (cit. on pp. 2, 41, 42, 86, 87, 90).

62. Choset, H. (2000). “Coverage of known spaces: The boustrophedon cellular decomposition”. In: *Autonomous Robots* 9.3, pp. 247–253 (cit. on p. 86).

63. — (2001). “Coverage for robotics–A survey of recent results”. In: *Annals of Mathematics and Artificial Intelligence* 31, pp. 113–126 (cit. on pp. 2, 9, 15, 20, 23, 24, 39, 41, 42, 44, 84, 86, 127).

64. 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. 20, 24, 41, 79, 80, 84–87).

65. Choset, H. and Pignon, P. (1998). "Coverage path planning: The boustrophedon cellular decomposition". In: *Field and Service Robotics*. Springer, pp. 203–209 (cit. on pp. 23, 41, 84, 129).

66. 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. 37).

67. Clercq, K. M. D., Kat, R. de, Remes, B., Oudheusden, B. W. van, and Bijl, H. (2009). "Aerodynamic experiments on DelFly II: Unsteady lift enhancement". In: *International Journal of Micro Air Vehicles* 1.4, pp. 255–262 (cit. on p. 6).

68. Collange, S., Defour, D., and Tisserand, A. (2009). "Power consumption of GPUs from a software perspective". In: *Computational Science (ICCS)*. Ed. by G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, and P. M. A. Sloot. Springer, pp. 914–923 (cit. on p. 36).

69. 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 Società Astronomica Italiana Supplementi* 16, p. 42 (cit. on p. 6).

70. 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).

71. Corke, P. (2017). *Robotics, vision and control: Fundamental algorithms in Matlab*. 2nd. Springer (cit. on pp. 1, 5).

72. 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. 27, 112).

73. 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: *7th International Conference on Cybernetics and Intelligent Systems (CIS) and Conference on Robotics, Automation and Mechatronics (RAM)*. IEEE, pp. 167–174 (cit. on p. 4).

74. 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. 32).

75. 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. 44).

76. 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: *Conference Series: Earth and Environmental Science*. Vol. 275. 1. IOP Publishing, p. 012022 (cit. on pp. 3, 4, 9).

77. Darke, P., Shanks, G., and Broadbent, M. (1998). "Successfully completing case study research: Combining rigour, relevance and pragmatism". In: *Information Systems Journal* 8.4, pp. 273–289 (cit. on p. 10).

78. 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), pp. 68–75 (cit. on p. 38).

79. 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: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 179–185 (cit. on pp. 8, 56).

80. Di Franco, C. and Buttazzo, G. (2015). “Energy-aware coverage path planning of UAVs”. In: *International Conference on Autonomous Robot Systems and Competitions*. IEEE, pp. 111–117 (cit. on pp. 24, 45).

81. — (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. 45).

82. 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, pp. 65–93 (cit. on p. 96).

83. Dille, M. and Singh, S. (2013). “Efficient aerial coverage search in road networks”. In: *Guidance, Navigation, and Control (GNC) Conference*. AIAA, pp. 1–20 (cit. on pp. 24, 46, 89).

84. 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. 3).

85. 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. 38).

86. Dressler, F. and Dietrich, I. (2006). “Lifetime analysis in heterogeneous sensor networks”. In: *9th EUROMICRO Conference on Digital System Design (DSD)*. IEEE, pp. 606–616 (cit. on p. 41).

87. Dressler, F. and Fuchs, G. (2005). “Energy-aware operation and task allocation of autonomous robots”. In: *5th International Workshop on Robot Motion and Control (RoMoCo)*. IEEE, pp. 163–168 (cit. on p. 40).

88. 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. 3).

89. Eiselt, H. A. and Laporte, G. (2000). “A historical perspective on arc routing”. In: *Arc routing: Theory, solutions and applications*. Springer, pp. 1–16 (cit. on p. 46).

90. Erickson, D. (2003). “Non-learning artificial neural network approach to motion planning for the Pioneer robot”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 112–117 (cit. on p. 47).

91. 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. 49).

92. Espedal, I. B., Jinasena, A., Burheim, O. S., and Lamb, J. J. (2021). In: *Energies* 14.11 (cit. on p. 63).

93. 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. 41).

94. Feynman, R., Leighton, R., and Sands, M. (2015). *The Feynman lectures on physics, Vol. II: The new millennium edition: Mainly electromagnetism and matter*. Vol. 2. Basic Books (cit. on p. 79).

95. 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).

96. Flautner, K., Reinhardt, S., and Mudge, T. (2001). “Automatic performance setting for dynamic voltage scaling”. In: *7th Annual International Conference on Mobile Computing and Networking (MobiCom)*. ACM, pp. 260–271 (cit. on p. 32).

97. 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).

98. Fomenko, A. T. and Kunii, T. L. (1997). *Topological modeling for visualization*. Springer (cit. on p. 46).

99. Forejt, V., Kwiatkowska, M., and Parker, D. (2012). “Pareto curves for probabilistic model checking”. In: *Automated Technology for Verification and Analysis*. Springer, pp. 317–332 (cit. on p. 50).

100. Fuchs, G., Truchat, S., and Dressler, F. (2006). “Distributed software management in sensor networks using profiling techniques”. In: *1st International Conference on Communication Systems Software Middleware*. IEEE, pp. 1–6 (cit. on p. 41).

101. 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).

102. Gabriely, Y. and Rimon, E. (2002). “Spiral-STC: An on-line coverage algorithm of grid environments by a mobile robot”. In: *International Conference on Robotics and Automation (ICRA)*. Vol. 1. IEEE, pp. 954–960 (cit. on p. 42).

103. 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. 2, 9, 15, 20, 23, 41, 42, 44, 84, 86, 127, 129).

104. Gankidi, P. R. and Thangavelautham, J. (2017). “FPGA architecture for deep learning and its application to planetary robotics”. In: *Aerospace Conference*. IEEE, pp. 1–9 (cit. on p. 130).

105. García 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: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5740–5745 (cit. on pp. 79, 81, 82, 129).

106. 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. 35, 36).

107. 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, 99).

108. Geem, Z. W. (2009). *Music-inspired harmony search algorithm: Theory and applications*. Vol. 191. Springer (cit. on p. 46).

109. 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. 56).

110. 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. 44).

111. Gold, S. (1997). “A PSPICE macromodel for lithium-ion batteries”. In: *12th Annual Battery Conference on Applications and Advances*. IEEE, pp. 215–222 (cit. on p. 38).

112. 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).

113. Gong, Z., Li, J., and Li, W. (2016). “A low cost indoor mapping robot based on TinySLAM algorithm”. In: *International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, pp. 4549–4552 (cit. on p. 56).

114. 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).

115. Goraczko, M., Liu, J., Lymberopoulos, D., Matic, S., Priyantha, B., and Zhao, F. (2008). “Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems”. In: *45th ACM/IEEE Design Automation Conference*. IEEE, pp. 191–196 (cit. on p. 33).

116. Groen, M., Bruggeman, B., Remes, B., Ruijsink, R., van Oudheusden, B., and Bijl, H. (2010). “Improving flight performance of the flapping wing MAV DelFly II”. In: *International Micro Air Vehicle Conference and Flight Competition (IMAV)*. Geman Institue of Navigation, pp. 1–17 (cit. on pp. 6, 9).

117. Grüne, L. and Pannek, J. (2017). “Numerical optimal control of nonlinear systems”. In: *Nonlinear model predictive control*. Springer, pp. 275–339 (cit. on p. 96).

118. Hajjaj, S. S. H. and Sahari, K. S. M. (2014). “Review of research in the area of agriculture mobile robots”. In: *8th International Conference on Robotic, Vision, Signal Processing & Power Applications*. Springer, pp. 107–117 (cit. on p. 3).

119. Hamza, A. and Ayanian, N. (2017). “Forecasting battery state of charge for robot missions”. In: *Proceedings of the Symposium on Applied Computing*. ACM, pp. 249–255 (cit. on p. 38).

120. Hasan, A., Skriver, M., and Johansen, T. A. (2018). “Exogenous Kalman filter for state-of-charge estimation in lithium-ion batteries”. In: *Conference on Control Technology and Applications (CCTA)*. IEEE, pp. 1403–1408 (cit. on pp. 38, 62–64, 131).

121. 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. 5).

122. Hayat, S., Yanmaz, E., Brown, T. X., and Bettstetter, C. (2017). “Multi-objective UAV path planning for search and rescue”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5569–5574 (cit. on pp. 4, 44).

123. 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. 39, 63–65, 93).

124. Higa, S., Iwashita, Y., Otsu, K., Ono, M., Lamarre, O., Didier, A., and Hoffmann, M. (2019). “Vision-based estimation of driving energy for planetary rovers using deep learning and terramechanics”. In: *IEEE Robotics and Automation Letters* 4.4, pp. 3876–3883 (cit. on p. 130).

125. Hinz, H. (2019). “Comparison of lithium-ion battery models for simulating storage systems in distributed power generation”. In: *Inventions* 4 (cit. on pp. 39, 62–65, 93, 131).

126. Ho, D.-K., Ben Chehida, K., Miramond, B., and Auguin, M. (2018). “Towards a multi-mission QoS and energy manager for autonomous mobile robots”. In: *2nd International Conference on Robotic Computing (IRC)*. IEEE, pp. 270–273 (cit. on p. 50).

127. — (2019a). “Learning-based adaptive management of QoS and energy for mobile robotic missions”. In: *International Journal of Semantic Computing* 13.04, pp. 513–539 (cit. on p. 50).

128. — (2019b). “QoS and energy-aware run-time adaptation for mobile robotic missions: A learning approach”. In: *3rd International Conference on Robotic Computing (IRC)*. IEEE, pp. 212–219 (cit. on p. 50).

129. Hobby, H. (n.d.). *Opterra 2m wing BNF basic*. `https://www.horizonhobby.com/opterra-2m-wing-bnf-basic-p-efl11150`. Accessed: 2020-02-02 (cit. on p. 3).

130. 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. 8).

131. Holper, J. W., Henry, K. V., and Atkins, E. M. (2017). “Cyber-physical thermal modeling for a small UAS”. In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 1757–1766 (cit. on pp. 8, 56).

132. 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. 37).

133. 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. 36).

134. Horowitz, M. (2014). “Computing’s energy problem (and what we can do about it)”. In: *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, pp. 10–14 (cit. on pp. 32, 53).

135. 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. 38).

136. Huang, W. (2001). “Optimal line-sweep-based decompositions for coverage algorithms”. In: *International Conference on Robotics and Automation (ICRA)*. Vol. 1. IEEE, pp. 27–32 (cit. on pp. 25, 42, 43, 46, 88).

137. Iserles, A. (2009). *A first course in the numerical analysis of differential equations*. 44. Cambridge University Press (cit. on pp. 65, 94, 96, 97, 99).

138. Ishigami, G., Nagatani, K., and Yoshida, K. (2011). “Path planning and evaluation for planetary rovers based on dynamic mobility index”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 601–606 (cit. on p. 130).

139. *Ivy lightweight software bus* (n.d.). `https://ivy-python.readthedocs.io/en/latest/index.html`. Accessed: 2021-10-12 (cit. on p. 119).

140. Jaiem, L., Druon, S., Lapierre, L., and Crestani, D. (2016). “A step toward mobile robots autonomy: Energy estimation models”. In: *Towards Autonomous Robotic Systems*. Springer, pp. 177–188 (cit. on p. 1).

141. 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: *19th International Conference on Advanced Robotics (ICAR)*. IEEE, pp. 773–778 (cit. on p. 2).

142. 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. 38).

143. Juliá, M., Gil, A., and Reinoso, O. (2012). “A comparison of path planning strategies for autonomous exploration and mapping of unknown environments”. In: *Autonomous Robots* 33.4, pp. 427–444 (cit. on p. 41).

144. 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. 99).

145. 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 pp. 91, 99).

146. 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, 99).

147. 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).

148. 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).

149. 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. 47).

150. Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S. (2011). “Anytime motion planning using the RRT*”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1478–1483 (cit. on p. 47).

151. Kasichayanula, K., Terpstra, D., Luszczek, P., Tomov, S., Moore, S., and Peterson, G. D. (2012). “Power aware computing on GPUs”. In: *Symposium on Application Accelerators in High Performance Computing*, pp. 64–73 (cit. on p. 35).

152. 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).

153. 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).

154. Kim, C. H. and Kim, B. K. (2005). “Energy-saving 3-step velocity control algorithm for battery-powered wheeled mobile robots”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2375–2380 (cit. on p. 39).

155. Kim, H. and Kim, B.-K. (2008). “Minimum-energy translational trajectory planning for battery-powered three-wheeled omni-directional mobile robots”. In: *10th International Conference on Control, Automation, Robotics and Vision*. IEEE, pp. 1730–1735 (cit. on p. 39).

156. 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. 38).

157. 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. 38).

158. 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. 35).

159. Kirk, D. B. and Wen-Mei, W. H. (2016). *Programming massively parallel processors: A hands-on approach*. 3rd ed. Morgan Kaufmann (cit. on p. 106).

160. Kostadinov, D. and Scaramuzza, D. (2020). “Online weight-adaptive nonlinear model predictive control”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1180–1185 (cit. on p. 91).

161. Kreciglowa, N., Karydis, K., and Kumar, V. (2017). “Energy efficiency of trajectory generation methods for stop-and-go aerial robot navigation”. In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 656–662 (cit. on p. 44).

162. Krotkov, E. and Simmons, R. G. (1992). “Performance of a six-legged planetary rover: Power, positioning, and autonomous walking”. In: *1992 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 169–174 (cit. on p. 130).

163. Kuo, B. (1967). *Automatic control systems*. Electrical engineering series. Prentice-Hall (cit. on pp. 68, 69).

164. 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. 38).

165. 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. 38).

166. Kwon, W. H. and Han, S. H. (2005). *Receding horizon control: Model predictive control for state models*. Springer (cit. on p. 91).

167. 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. 1, 8, 12, 23, 47, 49, 50, 77, 91, 129).

168. LaValle, S. M. (2006). *Planning algorithms*. Cambridge University Press (cit. on pp. 1, 24, 31, 40, 41, 45, 47, 79, 86).

169. Lee, B. C. and Brooks, D. M. (2006a). “Accurate and efficient regression modeling for microarchitectural performance and power prediction”. In: *12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, pp. 185–194 (cit. on pp. 37, 59).

170. — (2006b). “Statistically rigorous regression modeling for the microprocessor design space”. In: *ISCA-33: Workshop on Modeling, Benchmarking, and Simulation* (cit. on pp. 37, 59).

171. 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. 38).

172. 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. 42).

173. Lemay, M., Michaud, F., Letourneau, D., and Valin, J.-M. (2004). “Autonomous initialization of robot formations”. In: *International Conference on Robotics and Automation (ICRA)*. Vol. 3, pp. 3018–3023 (cit. on p. 47).

174. 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. 36).

175. 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. 24, 42, 46, 88).

176. 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. 3).

177. Lindemann, S. R. and LaValle, S. M. (2005). “Smoothly blending vector fields for global robot navigation”. In: *44th IEEE Conference on Decision and Control (CDC)*. IEEE, pp. 3553–3559 (cit. on p. 79).

178. 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. 38).

179. Lottes, P., Khanna, R., Pfeifer, J., Siegwart, R., and Stachniss, C. (2017). “UAV-based crop and weed classification for smart farming”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3024–3031 (cit. on p. 4).

180. 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. 63).

181. Luo, C. and Suda, R. (2011). “A performance and energy consumption analytical model for GPU”. In: *9th International Conference on Dependable, Autonomic and Secure Computing*. IEEE, pp. 658–665 (cit. on p. 36).

182. Luo, J. and Jha, N. K. (2001). “Battery-aware static scheduling for distributed real-time embedded systems”. In: *38th annual Design Automation Conference*. ACM, pp. 444–449 (cit. on p. 37).

183. 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).

184. 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: *41st International Conference on Parallel Processing*. IEEE, pp. 48–57 (cit. on p. 35).

185. 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. 39).

186. 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).

187. Mannadiar, R. and Rekleitis, I. (2010). “Optimal coverage of a known arbitrary environment”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5525–5530 (cit. on pp. 24, 46, 84).

188. 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. 38).

189. Marowka, A. (2017). “Energy-aware modeling of scaled heterogeneous systems”. In: *International Journal of Parallel Programming* 45.5, pp. 1026–1045 (cit. on p. 33).

190. 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*. Springer, pp. 221–230 (cit. on p. 44).

191. Mei, Y., Lu, Y.-H., Hu, Y. C., and Lee, C. G. (2004). “Energy-efficient motion planning for mobile robots”. In: *International Conference on Robotics and Automation (ICRA)*. Vol. 5. IEEE, pp. 4344–4349 (cit. on pp. 2, 39, 40).

192. — (2005). “A case study of mobile robot’s energy consumption and conservation techniques”. In: *12th International Conference on Advanced Robotics (ICAR)*. IEEE, pp. 492–497 (cit. on pp. 2, 8, 12, 39, 40, 47, 53, 129).

193. Mei, Y., Lu, Y.-H., Hu, Y., and Lee, C. (2005a). “Deployment strategy for mobile robots with energy and timing constraints”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2816–2821 (cit. on p. 49).

194. — (2005b). “Reducing the number of mobile sensors for coverage tasks”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1426–1431 (cit. on p. 49).

195. 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, 12, 47, 49, 129).

196. Mei, Y., Lu, Y.-H., Lee, C., and Hu, Y. (2006). “Energy-efficient mobile robot exploration”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 505–511 (cit. on pp. 8, 49).

197. 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).

198. 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. 35).

199. 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 p. 35).

200. 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).

201. Morbidi, F., Cano, R., and Lara, D. (2016). “Minimum-energy path generation for a quadrotor UAV”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1492–1498 (cit. on p. 44).

202. 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. 38).

203. 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. 39, 63–65, 93, 131).

204. Muirhead, B. K. and Karp, A. (2019). “Mars sample return lander mission concepts”. In: *Aerospace Conference*. IEEE, pp. 1–9 (cit. on p. 130).

205. Nam, L. H., Huang, L., Li, X. J., and Xu, J. F. (2016). “An approach for coverage path planning for UAVs”. In: *14th International Workshop on Advanced Motion Control (AMC)*. IEEE, pp. 411–416 (cit. on p. 45).

206. Needham, T. (1998). *Visual complex analysis*. Oxford University Press (cit. on p. 79).

207. Nikov, K., Nunez-Yanez, J. L., and Horsnell, M. (2015). “Evaluation of hybrid run-time power models for the ARM big.LITTLE architecture”. In: *13th International Conference on Embedded and Ubiquitous Computing*. IEEE, pp. 205–210 (cit. on pp. 37, 108).

208. Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer (cit. on pp. 95, 96).

209. Noor, N. M., Abdullah, A., and Hashim, M. (2018). “Remote sensing UAV/drones and its applications for urban areas: A review”. In: *Conference Series: Earth and Environmental Science*. Vol. 169. 1. IOP Publishing, p. 012003 (cit. on p. 4).

210. 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. 37, 108).

211. NVIDIA (n.d.). *NVIDIA Jetson Nano developer kit*. `https://developer.nvidia.com/embedded/jetson-nano-developer-kit`. Accessed: 2020-02-02 (cit. on p. 115).

212. 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. 32).

213. O’Neal, K. and Brisk, P. (2018). “Predictive modeling for CPU, GPU, and FPGA performance and power consumption: A survey”. In: *Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, pp. 763–768 (cit. on p. 32).

214. Ogata, K. (2002). *Modern control engineering*. Prentice-Hall (cit. on p. 69).

215. Ondrúška, P., Gurău, C., Marchegiani, L., Tong, C. H., and Posner, I. (2015). “Scheduled perception for energy-efficient path following”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4799–4806 (cit. on pp. 1, 8, 11, 12, 23, 31, 39, 47, 49, 50, 53, 91, 127, 129, 130).

216. Ono, M., Rothrock, B., Otsu, K., Higa, S., Iwashita, Y., Didier, A., Islam, T., Laporte, C., Sun, V., Stack, K., et al. (2020). “MAARS: Machine learning-based analytics for automated rover systems”. In: *Aerospace Conference*. IEEE, pp. 1–17 (cit. on p. 130).

217. Panagou, D. (2014). “Motion planning and collision avoidance using navigation vector fields”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2513–2518 (cit. on p. 79).

218. Panigrahi, D., Chiasserini, C., Dey, S., Rao, R., Raghunathan, A., and Lahiri, K. (2001). “Battery life estimation of mobile embedded systems”. In: *14th International Conference on VLSI Design*. IEEE, pp. 57–63 (cit. on p. 39).

219. 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: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4319–4324 (cit. on pp. 8, 56).

220. Paparazzi (n.d.[a]). *NPS new paparazzi simulator*. `https://wiki.paparazziuav.org/wiki/NPS`. Accessed: 2021-10-12 (cit. on p. 119).

221. Paparazzi (n.d.[b]). *UAV open-source project*. `http://wiki.paparazziuav.org/`. Accessed: 2021-10-12 (cit. on pp. 9, 108, 115, 119).

222. 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).

223. Pedram, M. and Wu, Q. (1999). "Design considerations for battery-powered electronics". In: *36th Annual Design Automation Conference*. ACM, pp. 861–866 (cit. on p. 38).

224. 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: *National Aerospace and Electronics Conference (NAECON)*. IEEE, pp. 117–121 (cit. on p. 56).

225. 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).

226. 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 Applied Aerodynamics Conference*. AIAA (cit. on p. 6).

227. Popović, M., Hitz, G., Nieto, J., Sa, I., Siegwart, R., and Galceran, E. (2017). "Online informative path planning for active classification using UAVs". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5753–5758 (cit. on pp. 4, 44).

228. 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. 3, 4).

229. PX4 (n.d.). *PX4 open-source autopilot*. `https://px4.io/`. Accessed: 2021-10-13 (cit. on pp. 10, 46).

230. Qingchun, F., Wengang, Z., Quan, Q., Kai, J., and Rui, G. (2012). "Study on strawberry robotic harvesting system". In: *International Conference on Computer Science and Automation Engineering (CSAE)*. Vol. 1. IEEE, pp. 320–324 (cit. on p. 3).

231. 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, 59).

232. Rakhmatov, D. and Vrudhula, S. (2001). "An analytical high-level battery model for use in energy management of portable electronic systems". In: *International Conference on Computer Aided Design (ICCAD) Digest of Technical Papers*. IEEE, pp. 488–493 (cit. on p. 38).

233. 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).

234. 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. 38, 62).

235. 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*. IEEE, pp. 105–110 (cit. on pp. 32, 39).

236. Rawlings, J. B., Mayne, D. Q., and Diehl, M. (2017). *Model predictive control: Theory, computation, and design*. Vol. 2. Nob Hill Publishing (cit. on pp. 11, 91, 95–97).

237. 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: *27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 1–8 (cit. on p. 37).

238. Redmon, J. (2013–2016). *Darknet: Open Source Neural Networks in C*. `http://pjreddie.com/darknet/` (cit. on p. 102).

239. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). "You only look once: Unified, real-time object detection". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 779–788 (cit. on pp. 102, 109).

240. Redmon, J. and Farhadi, A. (2017). "YOLO9000: Better, Faster, Stronger". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 6517–6525 (cit. on p. 102).

241. Renzaglia, A., Reymann, C., and Lacroix, S. (2016). "Monitoring the evolution of clouds with UAVs". In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 278–283 (cit. on p. 4).

242. 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. 2).

243. Rossiter, J. A. (2004). *Model-based predictive control: A practical approach*. CRC press (cit. on p. 91).

244. Rouxel, B., Schultz, U. P., Akesson, B., Holst, J., Jorgensen, O., and Grelck, C. (2020). "PReGO: A generative methodology for satisfying real-time requirements on COTS-based systems: Definition and experience report". In: *SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE)*. ACM, pp. 70–83 (cit. on p. 11).

245. 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: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6268–6275 (cit. on p. 56).

246. 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).

247. Sadat, S. A., Wawerla, J., and Vaughan, R. T. (2014). "Recursive non-uniform coverage of unknown terrains for UAVs". In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1742–1747 (cit. on p. 45).

248. 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. 12, 47, 49, 129).

249. *Real-time energy-efficient path planning for unmanned ground vehicles using mission prior knowledge* (2013b). ASME, pp. 1–10 (cit. on p. 49).

250. Sadrpour, A., Jin, J., and Ulsoy, A. G. (2013c). “Experimental validation of mission energy prediction model for unmanned ground vehicles”. In: *American Control Conference*. IEEE, pp. 5960–5965 (cit. on pp. 12, 47, 49, 129).

251. 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 pp. 64, 131).

252. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). “MobileNetV2: Inverted residuals and linear bottlenecks”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 4510–4520 (cit. on p. 111).

253. 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: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 1445–1448 (cit. on p. 2).

254. 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. 109).

255. 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).

256. 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).

257. Shi, P. and Zhao, Y. (2006). “Application of unscented Kalman filter in the SoC estimation of li-ion battery for autonomous mobile robot”. In: *International Conference on Information Acquisition*. IEEE, pp. 1279–1283 (cit. on p. 62).

258. 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. 42, 43).

259. Siciliano, B. and Khatib, O. (2016a). *Handbook of robotics*. Springer. Chap. Chapter 47, pp. 1177–1201 (cit. on p. 1).

260. — (2016b). *Handbook of robotics*. Springer. Chap. Chapter 44, pp. 1012–1014 (cit. on pp. 3–6).

261. Simon, D. (2006). *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons (cit. on pp. 11, 91, 99, 100).

262. Song, Y. and Scaramuzza, D. (2020). “Learning high-level policies for model predictive control”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 7629–7636 (cit. on p. 91).

263. Stastny, T. and Siegwart, R. (2018). “Nonlinear Model Predictive Guidance for Fixed-wing UAVs Using Identified Control Augmented Dynamics”. In: *International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 432–442 (cit. on pp. 91, 99).

264. Stroustrup, B. (1988). “What is object-oriented programming?” In: *IEEE Software* 5.3, pp. 10–20 (cit. on p. 59).

265. Sudhakar, S., Karaman, S., and Sze, V. (2020). “Balancing actuation and computing energy in motion planning”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4259–4265 (cit. on pp. 1, 2, 8, 12, 31, 47, 50, 101, 127, 129).

266. Sundén, B. (2019). “Thermal management of batteries”. In: *Hydrogen, Batteries and Fuel Cells*. Ed. by B. Sundén. Academic Press, pp. 93–110 (cit. on p. 38).

267. Susman, G. I. and Evered, R. D. (1978). “An assessment of the scientific merits of action research”. In: *Administrative Science Quarterly* 23.4, pp. 582–603 (cit. on p. 10).

268. Syracuse, K. and Clark, W. (1997). “A statistical approach to domain performance modeling for oxyhalide primary lithium batteries”. In: *12th Annual Battery Conference on Applications and Advances*. IEEE, pp. 163–170 (cit. on p. 38).

269. Szeliski, R. (2011). *Computer vision algorithms and applications*. London: Springer (cit. on p. 104).

270. Takouna, I., Dawoud, W., and Meinel, C. (2011). “Accurate mutlicore processor power models for power-aware resource management”. In: *9th International Conference on Dependable, Autonomic and Secure Computing*. IEEE, pp. 419–426 (cit. on p. 37).

271. 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).

272. TeamPlay Consortium (2019a). *Deliverable D4.3: Report on energy, timing and security modeling of complex architectures (v2.0)*. `https://gitlab.inria.fr/TeamPlay_Public/TeamPlay_Public_Deliverables/-/blob/master/D4.3.pdf`. Accessed: 2021-10-04 (cit. on pp. 59, 103).

273. — (2019b). *Deliverables of the TeamPlay H2020 project*. `https://teamplay-h2020.eu/index.php?page=deliverables`. Accessed: 2021-10-04 (cit. on pp. 11, 130).

274. 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).

275. 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).

276. 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. 111, 115, 121).

277. Valavanis, K. P. and Vachtsevanos, G. J. (2015). *Handbook of unmanned aerial vehicles*. Vol. 1. Springer (cit. on p. 3).

278. 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. 46).

279. Viega, J., Messier, M., and Chandra, P. (2002). *Network security with OpenSSL: cryptography for secure communications*. O’Reilly (cit. on p. 109).

280. Voosen, P. (2019). “NASA to fly drone on Titan”. In: AAAS (cit. on p. 5).

281. Wächter, A. and Biegler, L. T. (2006). “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106.1, pp. 25–57 (cit. on p. 99).

282. Wahab, M., Rios-Gutierrez, F., and El Shahat, A. (2015). “Energy modeling of differential drive robots”. In: *Southeast Conference (SoutheastCon)*. IEEE (cit. on p. 39).

283. 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. 37, 38).

284. 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 Conference*. IEEE, pp. 1–4 (cit. on p. 56).

285. Wang, L. (2009). *Model predictive control system design and implementation using Matlab*. Springer (cit. on p. 91).

286. 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, 24, 44, 84).

287. 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. 8).

288. Wegner, P. (1990). “Concepts and paradigms of object-oriented programming”. In: *SIGPLAN OOPS Mess.* 1.1, pp. 7–87 (cit. on p. 59).

289. Wei, M. and Isler, V. (2018). “Coverage path planning under the energy constraint”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 368–373 (cit. on pp. 40, 42, 43).

290. Wright, O. (Dec. 1913). “How we made the first flight”. In: *Flying* (cit. on p. iii).

291. Wu, G., Greathouse, J. L., Lyashevsky, A., Jayasena, N., and Chiou, D. (2015). “GPGPU performance and power estimation using machine learning”. In: *21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 564–576 (cit. on p. 36).

292. 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. 38, 62).

293. 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. 38).

294. Xu, A., Viriyasuthee, C., and Rekleitis, I. (2011). “Optimal complete terrain coverage using an unmanned aerial vehicle”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2513–2519 (cit. on pp. 24, 42, 44, 46, 84, 88, 89).

295. — (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. 24, 25, 46, 84, 88, 89).

296. 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. 49).

297. Yang, T.-J., Chen, Y.-H., Emer, J., and Sze, V. (2017). “A method to estimate the energy consumption of deep neural networks”. In: *51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, pp. 1916–1920 (cit. on pp. 35, 36).

298. Yang, T.-J., Chen, Y.-H., and Sze, V. (2017). “Designing energy-efficient convolutional neural networks using energy-aware pruning”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 5687–5695 (cit. on p. 35).

299. 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: *14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, pp. 1–8 (cit. on p. 50).

300. 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*. IEEE, pp. 533–538 (cit. on p. 42).

301. 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. 39, 62–64, 131).

302. Zhang, C., Li, K., Mcloone, S., and Yang, Z. (2014). “Battery modelling methods for electric vehicles - A review”. In: *European Control Conference (ECC)*. IEEE, pp. 2673–2678 (cit. on pp. 38, 62).

303. Zhang, F., Liu, G., and Fang, L. (2009). “Battery state estimation using unscented Kalman filter”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1863–1868 (cit. on p. 39).

304. Zhang, F., Liu, G., Fang, L., and Wang, H. (2012). “Estimation of battery state of charge with H infinity observer: Applied to a robot for inspecting power transmission lines”. In: *IEEE Transactions on Industrial Electronics* 59.2, pp. 1086–1095 (cit. on p. 39).

305. 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: *8th International Conference on Hardware/Software Codesign and System Synthesis*. ACM, pp. 105–114 (cit. on p. 39).

306. 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. 63).

307. Zhang, W. and Hu, J. (2007). “Low power management for autonomous mobile robots using optimal control”. In: *46th Conference on Decision and Control (CDC)*. IEEE, pp. 5364–5369 (cit. on pp. 12, 23, 47, 49, 91, 129, 131).

308. Zhou, D. and Schwager, M. (2014). “Vector field following for quadrotors using differential flatness”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6567–6572 (cit. on p. 79).

Index