Dynamic Energy Planning for Autonomous Aerial Robots

University of Southern Denmark

Dynamic Energy Planning for Autonomous Aerial Robots

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

by

Adam Seewald

Approved by:

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

Approved on:

2021

https://doi.org/10.

The typesetting is done using LaTeX. Figures are generated with PGF/TikZ. 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

To …

Acknowledgements

A[A]

Contents

Figures

Notation

$\exists$	there exists
$\in$	is an element of the set
$:=$	is defined
$f(\cdot)$	is function f
$f : \mathbb{C} \to \mathbb{D}$	f maps set $\mathbb{C}$ to set $\mathbb{D}$
$\mathbb{Z}$	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
$[x]$	set of positive integers up to a: $\{0, 1, \ldots, a\}$, with $a \in \mathbb{Z}_{\geq 0}$
$[x]_{>0}$	set of strictly positive integers up to a: $\{1, 2, \ldots, x\}$, with $x \in \mathbb{Z}_{>0}$
	vector
x_i	i-th set of parameters
$x_{i,j}$	j-th parameter of the i-th set of parameters
$\underline{x}_{i,j}$	lower bound of the parameter $x_{i,j}$
$\overline{x}_{i,j}$	upper bound of the parameter $x_{i,j}$

Abbreviations

LP	linear program
QP	quadratic program
MPC	model predictive control
NLP	non linear program
UAV	unmanned aerial vehicle
UAS	unmanned aerial system
OCP	optimal control problem
BVP	boundary-value problem
RSTA	reconnaissance, surveillance, and target acquisition
GNSS	global navigation satellite system
IMU	inertial measurement unit
RPV	remotely piloted vehicle
GPS	global positioning system
SoC	state of charge
M&CE	difference of motion and computation energy

Chapter 1

Introduction

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

To move, mobile robots combine different components which sense and interact with the surrounding environment (Mei et al., 2006). The analysis and interpretation of data originating from these components require energy-demanding heterogeneous computing hardware. Planning an energy-aware path with a power-saving scheduling policy on such hardware is an underrepresented topic in the literature. In fact, many past approaches focus almost exclusively on one of these topics. For instance, some approaches generate an energy-optimized path, despite the energy needed for the computations almost equaling the actuation in some instances of low-energy mobile robots (Sudhakar et al., 2020). Due to the recent advancements in the computational capabilities of mobile hardware, such as the introduction of powerful portable GPUs (Rizvi et al., 2017), the use of computations is expected to increase (Abramov et al., 2012; Jaramillo-Avila et al., 2019; Satria et al., 2016). Other approaches provide a power-saving scheduling policy. Yet, actuating a mobile robot requires considerable power over mere calculations (Mei et al., 2004, 2005).

In this work, we focus on energy optimal planning for the path and scheduling for the computational tasks. We plan these two aspects simultaneously, exploring their tradeoffs. We demonstrate the approach both in simulation and empirically focusing on aerial mobile robots. These systems share with the broader class of mobile robots very stringent battery limitations. Recharging the battery during normal operation

Fig. 1.1. Opterra fixed-wing drone employed as an aerial robotics platform for precision agriculture (photo credit: Amit Ferencz Appel).

is, however, rarely an option for aerial robots. It would require to land to replace or recharge the battery (Zamanakos et al., 2020). They are hence a perfect instance of energy-constrained systems and they apply to an energy optimization planning approach.

We build an energy model to predict the future energy consumption of some autonomous operations. We estimate the coefficients of the model using robust estimation techniques. Later, we derive an optimal configuration of the path and computations using modern optimal control in a data-driven control fashion. Here some estimates are made from sensors' data and used to derive an optimal control action. We use such optimal control for guidance and scheduling of the aerial robot. The guidance moves physically the robot; the scheduling defines the granularity of the tasks being executed in flight.

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

1.1 From UAVs to Modern Aerial Robots

Modern aerial robots are a valuable tool in robotic research and aerospace. Historically, they can be found with different names in the literature. These include un-

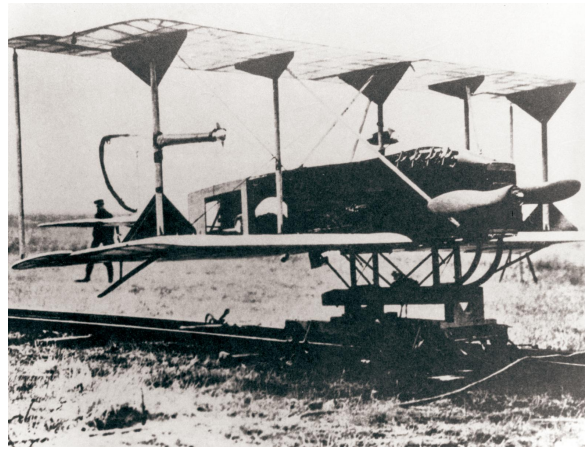

Fig. 1.2. The Hewitt-Sperry Automatic Airplane also denominated "flying bomb" was developed in WWI and represents the first instance of a UAV (photo credit: United States Naval Institute).

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

Uninhabited or unmanned flight has more than a century of developments (Siciliano and Khatib, 2016). The origin of the field of aerial robotics, which deals with the design and development of aerial robots, dates back to the first guided missiles (Siciliano and Khatib, 2016), although unmanned flying machines have more than a century of developments. One of the first such unmanned flying machines is the Hewitt-Sperry Automatic Airplane developed in 1917 during World War I (WWI) (Keane and Carr, 2013; Valavanis and Vachtsevanos, 2015); a milestone reached relatively soon if we consider that the first heavier-than-air flight in history was demonstrated 14 years earlier with the Wright Flyer I by Wilbur and Orville Wright (commonly referred to as the Wright brothers). The Hewitt-Sperry Automatic Airplane used a gyroscope, invented by Elmer Sperry (Keane and Carr, 2013), mechanically connected to the control sur-

[1]The term unmanned is sometimes replaced by uninhabited if the vehicle is operated remotely by human operator

faces and thus successfully integrating a control feedback loop (Siciliano and Khatib, 2016). It is shown in Figure 1.2.

In the early days, these flying machines were labeled remotely piloted vehicles (RPVs) (Anderson, 2005). Many instances from WWI on of such early UAVs were also designed for military purposes. In the 1950s, United States used a remotely controlled vehicle Ryan Firebee for reconnaissance in Vietnam, and Israel used first a RPV in a combat situation (Anderson, 2005). Some other instances of these vehicles include the V-1 flying bomb from 1944 (deployed by the unified armed forces of nazi Germany) and the Lockheed D-21 from 1962 (deployed by the United States Air Force).

Global positioning system (GPS) at the end of 1970 opened the possibility for using the UAVs in autonomous surveillance, and later integrate them with cameras (Siciliano and Khatib, 2016), paving for the introduction of modern UAVs. In recent years unmanned flight has been applied in many civilian applications (González-Jorge et al., 2017).

Aerial robots are used increasingly in applications such as remote sensing (Milas et al., 2018; Noor et al., 2018; Tang and Shao, 2015), surveillance (Bürkle, 2009; Paucar et al., 2018), meteorology (Schuyler et al., 2019), search and rescue (Cui et al., 2015; Karaca et al., 2018; Pensieri et al., 2020; Seguin et al., 2018), precision agriculture (Daponte et al., 2019; Puri et al., 2017), transportation, and payload delivery (Kellermann et al., 2020). Although the former four categories fall into the area of reconnaissance, surveillance, and target acquisition (RSTA), agriculture, transportation, and payload delivery are often realized using to a greater or lesser extent some advanced levels of computational intelligence (Siciliano and Khatib, 2016).

Modern aerial robots are designed to handle unexplored terrain with little interaction as opposed to past UAVs that were mostly operated by a human operator (Siciliano and Khatib, 2016). Instances of these systems are in fact expected to autonomously adapt and possibly interact in a broad variety of environmental conditions.

1.2 Common Classes of Aerial Robots

Numerous different types of aerial robots have emerged ever since their first introduction. We briefly investigate the most studied classes in the robotics literature and relate them to the dynamic energy planning that we focus on in this work. The two most generic classes are heavier-than-air and lighter-than-air aerial robots.

Heavier-than-air aerial robots are further divided into fixed- and rotary-wings (Siciliano and Khatib, 2016). Some recent developments in bio-inspired robotics study flapping-wings (Floreano and Wood, 2015). We discuss in detail heavier-than-air aerial robots in this work, as they are a common platform for robotics research.

Rotary-wing aerial robots are highly maneuverable and can perform stationary vertical flight (commonly referred to as hovering) (Siciliano and Khatib, 2016). These

systems can be classified into further categories, which include multirotors (such as quadrotors or quadcopters, hexacopters, or octocopters), conventional helicopters (these have one main and one tail rotor), and a coax (these have counter-rotating coaxial rotors) (Corke, 2017). In a fixed-wing aerial robot, wings are providing the lift. Some control surfaces are used for maneuvering, and a propeller for forwarding thrust; a shared principle with a common passenger aircraft (Corke, 2017). An example constitutes the Opterra adapted for precision agriculture in Figure 1.1.

We discuss the dynamic forces which govern the flight of rotary and fixed-wing aerial robots in Chapter 5.

Common instances of lighter-than-air aerial robots are blimps (or non-rigid airships). They usually rely on a gas–such as helium enclosed in a protected envelope (Burri et al., 2013)–to generate the lifting force (Fui Liew et al., 2017). An omnidirectional spherical blimp is shown in Figure 1.3. Blimps are similar to balloons but provide basic maneuverability whereas in a balloon only the altitude can be controlled (Colombatti et al., 2011).

Fig. 1.3. Skye, an omnidirectional spherical blimp developed by ETH Zurich. Developed 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 Zurich).

Some other classifications found in aerial robotics literature often rely on size and maneuverability and include micro aerial vehicles (MAVs). These are aerial robots with all dimensions lower than 15 cm. Vertical take-off and landing (VTOLs) are aerial robots flying in a fixed-wing configuration except if taking-off and landing where they use thrust from rotors rather than lift from wings.

While rotary-wings are the most maneuverable, less-than-air aerial robots are the least. These, however, have the highest endurance followed by fixed-wing aerial-robots. Flapping-wings and mixed configurations, such as VTOLs, fall into the intersection

of rotary and fixed-wings for what concerns maneuverability and endurance (Siciliano and Khatib, 2016). The energy requirements are critical for all aerial robots, but the difference of motion and computation energy (M&CE) varies greatly. The energy–inversely proportional to endurance–is highest in rotary-wings and lowest in lighten-than-air aerial robots. The energy planning approach accounts for both power-saving task scheduling and energy-efficient path planning the fixed-wings robots while accounting almost exclusively for energy-efficient path planning for rotary-wing aerial robots. In the former M&CE is close to zero, in the latter is usually high except in energy optimized designs such as some rotary-wing MAVs. Hypothetically, in lighter-than-air aerial robots, M&CE might be negative; the energy planning approach would rely heavily on power-saving task scheduling.

1.3 Motivation

Many applications that involve mobile robots have strict battery limitations. Although this is a common problem to almost any mobile robot, aerial robots are particularly affected. To overcome physical limitations they generally would have to land to recharge or replace the battery (Zamanakos et al., 2020). The autonomy of these systems is consequently affected by the available of the power source.

A typical example of an aerial robotics scenario in this situation is a robot following a path and performing some on-board computational tasks. For instance, the robot might detect ground patterns and notify other ground-based actors with little human interaction in a precision agriculture scenario.

Autonomous mobile robots usually carry energy-demanding heterogeneous computing hardware along a microcontroller used for basic motion. Computing hardware is involved in path planning and enables autonomous functions, opposed to the microcontroller. The latter executes motion primitives by directly interfacing actuators (such as servos for the control surfaces and motors for the thrust of a fixed-wing aerial robot). The former often computer vision and other complex algorithms.

For certain classes of aerial robots with M&CE close or lower than zero, the autonomy can directly influence the battery state. Indeed many autonomous tasks rely on energy-demanding mobile hardware. It is uncommon to find a ready-to-use solution for planning both the path and scheduling the computations of these systems in an energy-aware fashion (we discuss in detail the state-of-the-art in this regard in Chapter 2). Planning these two aspects simultaneously would pose an advantage in terms of autonomy by optimizing both the path and computations in function of the battery state of charge (SoC).

Moreover, it is often desirable to reschedule the computations in an energy-aware fashion in-flight phases when the energy needed for motion is different from expected. For instance, a fixed-wing aerial robot might be flying headwinds (with the wind

vector parallel and opposite to the direction of motion) and utilizing more energy than planned. It would be of advantage to reschedule the tasks accordingly to save energy needed for motion. During the same flight, the wind direction might suddenly change. The fixed-wing craft, now flying tailwinds, utilizes less energy. It could then potentially increase the level of computations by rescheduling the tasks. In the same flight, the battery might be subject to sudden drops (due to e.g., temperature changes) requiring again replanning by for instance shortening the path. It is clear that planning the path and scheduling the computations simultaneously in this scenario is the most desirable course of action. Yet, this is rarely the case.

1.3.1 Motion and computation energy

In the remainder, we refer to computational tasks that can be scheduled in an energy-aware fashion as *computations*, opposed to the other tasks that have no significant effect on energy consumption.

Definition 1.3.1 (Computations)
Computations are computational tasks executed on energy-demanding heterogeneous computing hardware carried by a mobile robot along microcontrollers, sensors, and actuators.

For instance, detecting ground patterns usually involves heterogeneous computing elements (GPUs and/or multiple CPU cores) and consumes a significant amount of energy opposed to the log of the position. Detection is a computation, while a log script is (an energy-inexpensive) task.

We use the computation and motion energy and M&CE several times in the remained. We thus define these terms formally.

Definition 1.3.2 (Computation/motion energy, M&CE)
Given a schedule composed of i computations $s := \{s_1, s_2, \ldots, s_i\}$, the computation energy $E_s(s)$ is the energy spent by heterogeneous computing hardware while executing s.
The motion energy is the remaining E_m energy the mobile robot utilize for the physical motion, for sensing, and for actuating.

Computation and motion energy are measured in watts if instantaneous or joules if over a given time interval.

Definition 1.3.3 (Difference of motion and computation energy M&CE)
The difference of motion and computation energy (M&CE) is then $E_s - E_m$.

1.3.2 Objective

We are interested in the energy optimization of the path and computations under uncertainty (atmospheric interferences) in-flight and refer to it as energy dynamic planning.

Such planning would find optimal tradeoffs between the path, computations, and energy requirements. Current generic planning solutions for aerial robots do not plan the path and computations dynamically, nor are they energy-aware. They are often semi-autonomous: the path and computations are static and usually defined using planning software (Daponte et al., 2019) (for instance (Paparazzi, 2016) and (PX4, 2016)). Such a state of practice has prompted us to pursue the presented work that culminates in the formal derivation of an energy-aware dynamic planning algorithm for mobile robots. The algorithm combines and generalizes some of the past body of knowledge on mobile robot planning problems and addresses the increasing computational demands and their relation to energy consumption, path, and autonomy for the mobile robot planning problem.

1.4 Outline of the Approach

Unlike most of the past mobile robots planning literature, the approach in this work plans the path and computations simultaneously.

To model the path we use multiple mathematical functions $\varphi_1, \varphi_2, \ldots$. If the path is expressed in 2D at a fixed altitude h, the function $\varphi : \mathbb{R}^2 \to \mathbb{R}$ quantifies the distance of a generic point $\mathbf{p}$ in the 2D space from the path. For instance one can define the path line

$$\varphi(x, y) := ax + by + c, \tag{1.1}$$

where a, b, c are given constants. A generic point $\mathbf{p} := (x_p, y_p)$ quantifies the distance on the z axis from $ax + by + c = 0$.

The concept is shown in Figure 1.4 for c zero and a, b two and minus one respectively. A generic point $\mathbf{p} := (x_\mathbf{p}, y_\mathbf{p})$ intersects the plane

$$\varphi(x, y) := 2y - x, \tag{1.2}$$

at $z = \varphi(\mathbf{p})$. The path at a fixed altitude is then $\varphi(x, y) = 0$, or more generically $\varphi(x, y) = h$ with $h \in \mathbb{R}$.

We use this concept to model complex paths. For instance, the path might contain multiple circles and lines, as shown in .... The aerial robot travels the path using vector field for guidance (we describe in detail such vector field in Chapter 5) and switches between the trajectories as soon as it reaches specific triggering points.

To model the computations we use `powprofiler`, a profiling tool presented in previous work (Seewald, Schultz, Ebeid, et al., 2021). To guide the UAV we use a vector

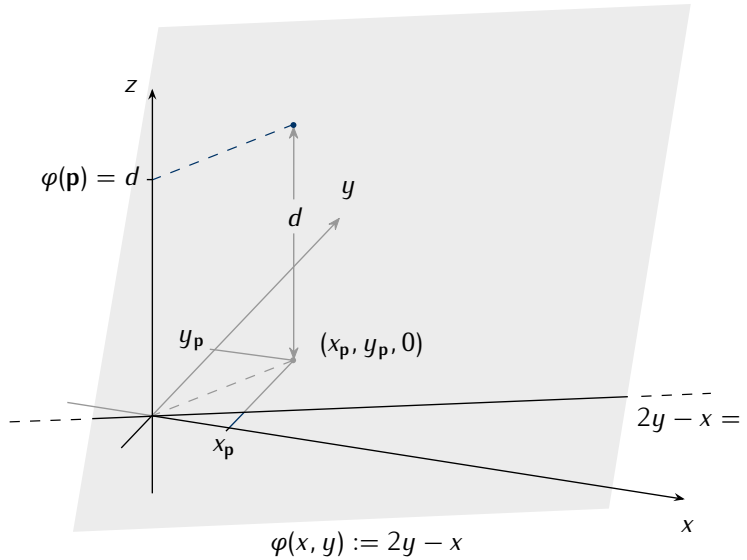

Fig. 1.4. If we define the path as mathematical function φ at a fixed altitude h, a generic point p in 2D space quantifies the distance d on the z axis from h.

field (De Marina et al., 2017) that converges to the path. The use of vector fields for guidance is widely discussed in the literature (De Marina et al., 2017; Gonçalves et al., 2010; Kapitanyuk et al., 2017; Lindemann and LaValle, 2005; Panagou, 2014; Zhou and Schwager, 2014).

To achieve the energy-aware dynamic planning, we further introduce and formally prove a periodic energy model that accounts for the uncertainty. We use Fourier analysis to derive the model, and state estimation to address the uncertainty. Periodicity is often present due to repetitive patterns in the plan (Seewald, Garcia de Marina, et al., 2020). Indeed, mobile robots often iterate over a set of tasks and paths. Given that the plan is periodic, we expect the energy consumption to approximately evolve periodically. We describe in detail such model in Chapter 3.

Some collected energy data from the Opterra (Figure 1.1) adapted for precision agriculture along its power spectrum motivates our choice in … Figure ?? and ….

The approach necessitates a user-defined initial plan that is replanned energy wise using the model for future prediction. In the remainder we progressively build an algorithm that inputs a user-specified initial plan, estimates the model, evaluates future energy consumption and eventually replans the parameters.

The initial plan consists of different stages. At each stage the mobile robot travels a path and executes some computations. The plan then contains some parameters that allow to alter the path and computations along an energy budget.

The alterations are bounded. There is one path constraint set which bounds the path and multiple computation constraint sets, one per each computation parameter, that bound computations.

We replans the path and computations with model predictive control (MPC) Rawlings et al., 2017. The control is data-driven. Energy sensor data estimates some coefficients of an energy model used to predict future energy consumption in presence of uncertainty. The energy budget is the battery capacity and other battery parameters. These are fixed values that are not replanned by the algorithm. Our goal is to complete the plan with the highest possible parameters configuration as the mobile robot executes the plan and its batteries drain.

1.4.1 Applications

In the spirit of reducing costs and resources, we showcase the approach using dynamic planning of the Opterra for a precision agriculture use-case. Precision agriculture is often put into practice (Hajjaj and Sahari, 2014) with ground mobile robots used for harvesting (Aljanobi et al., 2010; De-An et al., 2011; F. Dong et al., 2011; Edan et al., 2000; Z. Li et al., 2008; Qingchun et al., 2012), and aerial robots for preventing damage and ensuring better crop quality (Daponte et al., 2019; Puri et al., 2017).

The plan is structured as follows. Path-wise, the Opterra flies in circles and lines covering a polygon. Computation-wise, it detects hazards using a neural network and notifies grounded mobile robots employed for, e.g., harvesting. The approach alters the plan; it controls the processing rate and the radius of the circles affecting the distance between the lines.

We observe that not only the path but also the computations significantly impact the energy, with a potential extension of up to 13 minutes over an hour by switching from the highest to the lowest level of computations in presence of a standard battery.

1.5 Problem Formulation

To formulate the problem and in the remainder of this work we stick to the following mathematical notation.

Given an integer $x \in \mathbb{Z}_{\geq 0}$, we use the notation $[x]$ for the set $\{0, 1, \ldots, x\} \subseteq \mathbb{Z}_{\geq 0}$. The notation $[x]_{>0}$ for the set $\{1, 2, \ldots, x\} \subseteq \mathbb{Z}_{>0}$ or simply $[x]/\{0\}$ with $x \in \mathbb{Z}_{>0}$. We use bold lower-case letters for the vectors.

We use extensively the term parameters. These are variable values that we use to dynamically plan path and computations. $c_{i,j}$ denotes the j-th parameter of the i-th parameters set $\boldsymbol{c}_i$. $\underline{c}_{i,j}$ is the lower bound of the parameter $c_{i,j}$. $\overline{c}_{i,j}$ is the upper bound of the parameter $c_{i,j}$, i.e., $\underline{c}_{i,j} \leq c_{i,j} \leq \overline{c}_{i,j}$.

Let us assume that the path at stage i can be altered with ρ path parameters

$$c_i^{\rho} := \{c_{i,1}, c_{i,2}, \dots, c_{i,\rho}\}, \tag{1.3}$$

and the computations with σ computation parameters

$$c_i^{\sigma} := \{c_{i,\rho+1}, c_{i,\rho+2}, \dots, c_{i,\rho+\sigma}\}. \tag{1.4}$$

We then express the path as a continuous twice differentiable function $\varphi_i : \mathbb{R}^2 \times \mathbb{R}^{\rho} \to \mathbb{R}$ of a point and the path parameters. The function returns a metric of the distance between the point and the nominal trajectory. We express the computations as the value of the computation parameters. We discuss the concrete meaning of the value of path parameters in Chapter 3.

Definition 1.5.1 (Stage)
The i-th *stage* Γ_i at time instant t of a plan Γ is

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

where $\mathcal{C}_i := [\underline{c}_i, \overline{c}_i] \subseteq \mathbb{R}$ is the path constraint set, and $\mathcal{S}_{i,j} := [\underline{c}_{i,\rho+j}, \overline{c}_{i,\rho+j}] \subseteq \mathbb{Z}_{\geq 0}$ the j-th computation constraint set.
$\mathbf{p}(t)$ is a point of a mobile robot travelling at an altitude $h \in \mathbb{R}_{\geq 0}$ with respect to some inertial navigation frame $\mathcal{O}_W$.

Definition 1.5.2 (Plan)
The *plan* is a finite state machine (FSM) Γ where the state-transition function $s : \bigcup_i \Gamma_i \times \mathbb{R}^2 \to \bigcup_i \Gamma_i$ maps a stage and a point to the next stage

$$s(\Gamma_i, \mathbf{p}_k) := \begin{cases} \Gamma_{i+1} & \text{if } \mathbf{p}_k = \mathbf{p}_{\Gamma_i} \\ \Gamma_i & \text{otherwise} \end{cases}.$$

Definition 1.5.3 (Triggering, and final point)
The point $\mathbf{p}_{\Gamma_i}$ that allows the transition between Γ_i and Γ_{i+1} is called *triggering point*. The last triggering point $\mathbf{p}_{\Gamma_l}$ relative to the last stage Γ_l is called *final point*.

The altitude h from Definition 1.5.3 might change at different flying phases and under different atmospheric conditions

In Figure 1.6, $\varphi_1, \dots, \varphi_6$ are paths. φ_1 and φ_5 are circles, while φ_2, φ_4, and φ_6 are lines. They are all relative to different stages $\Gamma_1, \Gamma_2, \dots$. The constraints set $\mathcal{C}_1, \mathcal{C}_2, \dots$ forms the area where the paths $\varphi_1, \varphi_2, \dots$ can be altered with the parameters $c_{i,1}, \dots, c_{i,\rho}$ (gray area in the figure). This area is bounded by $\underline{c}_i$, $\overline{c}_i$, and can be different per each stage Figure 1.6, the area relative to Γ_4 is bounded by $\underline{c}_4$, $\overline{c}_4$).

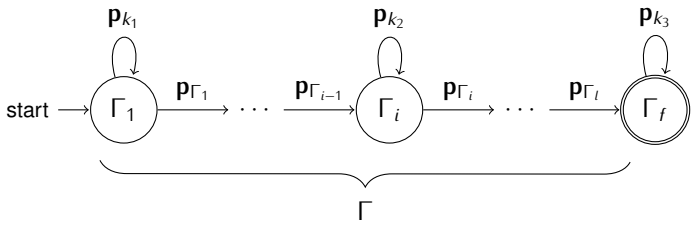

Fig. 1.5. The plan defined as a FSM

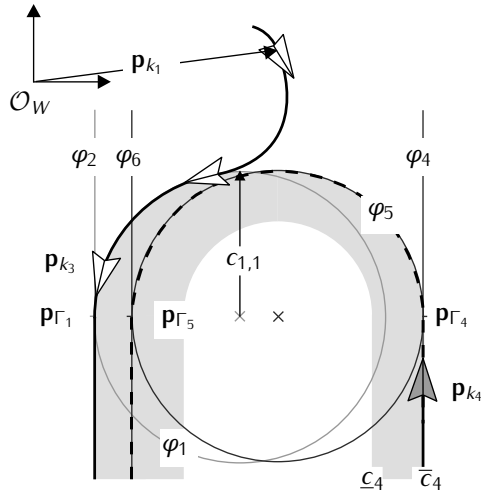

Fig. 1.6. Definition notation shown on an slice of the plan.

In Figure 1.6, $\mathbf{p}_{\Gamma_1}$ allows the transition between Γ_1 and Γ_2, $\mathbf{p}_{\Gamma_4}$ between Γ_4 and Γ_5, and $\mathbf{p}_{\Gamma_5}$ between Γ_5 and Γ_6.

A slice of the plan in Figure 1.5 shows the transition between the stages with the FSM. 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 $\mathbf{p}_{k_2}$. It eventually enters the stage Γ_{i+1} with the triggering point $\mathbf{p}_{\Gamma_i}$ and so on, until it reaches the final point. The stage Γ_f is the accepting stage (it indicates that the robot has completed the plan).

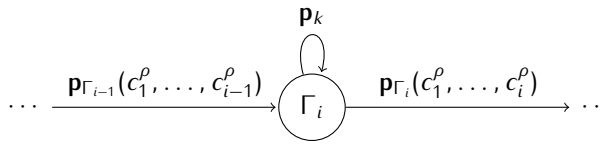

Fig. 1.7. Detail of the stage Γ_i in the FSM

Generally, one can express the triggering points in function of the i-th trajectory parameters c_i^ρ, or any previous trajectory parameters, propagating the information therein if necessary (see Figure 1.7).

1.5.1 Problem formulation

In order to simplify the problem formulation, we consider some primitive paths. All the other paths are built from these paths with a shift $\mathbf{d} := (x_d, y_d)$.

Given $n \in \mathbb{Z}_{>0}$ $(n < l, l/n \in \mathbb{Z})$ primitive paths $\varphi_1, \dots \varphi_n$, a generic starting point $\mathbf{p}$ and the current levels of the path parameters c_1^ρ, all the other paths $\varphi_{n+1}, \dots, \varphi_l$ are built

$$\begin{aligned} &\varphi_{(i-1)n+j}(\mathbf{p} + (i-1)\mathbf{d}, c_1^\rho) - \\ &\varphi_{in+j}(\mathbf{p} + i\mathbf{d}, c_1^\rho) = e_j, \end{aligned} \tag{1.5}$$

$\forall i \in [l/n - 1]^+, j \in [n]^+$, where $e_j \in \mathbb{R}$ is the j-th constant difference.

Definition 1.5.4 (Period)
The period $T \in \mathbb{R}_{>0}$ is the time between $\varphi_{(i-1)n+j}$ and φ_{in+j} in Equation (1.5).

The algorithm measures the time between the paths and assumes the initial period is one. The periods might be different for different js due to atmospheric interferences.

One can define the plan using primitive paths or define all the stages explicitly and find n searching the value which satisfies the Equation (1.5). If there is no such value, (e.g., when the plan is composed of only one stage), the period T from Definition 1.5.4 can be determined empirically from energy data.

Problem 1.5.1 (Aerila robot planning problem)
Consider an initial plan Γ from Definition 1.5.3. We are interested in the planning of the parameters c_i, $\forall i \in [l]^+$ and energy constraints and in the guidance of the aerial robot to the path resulting from such plan.

1.6 Structure

Chapter 2

State of the Art

A[A]

2.1 Energy Modeling

Marowka developed a power-metrics based analytical model, to exploit the possibility of considerably increasing energy efficiency by choosing an optimal chip configuration, which we extended to evaluate the impact of different architectural design choices on energy efficiency (Marowka, 2017). Marowka's theoretical contribution shows three processing schemes for heterogeneous computing with a comparison of their respective energy efficiency. Variations in chip configurations are done to investigate the impact on energy, power, and performance. Symmetric processor scheme consists of only a multicore CPU. Asymmetric CPU-GPU processor scheme consists of both CPU and GPU on the hardware side, and of a program running on CPU or GPU, but not on both in the same time interval, on the software side. CPU-GPU simultaneous processing scheme consists of a program running on CPU and GPU simultaneously. Our work extends and starts from Marowka's approach by building an experimental method to the CPU-GPU simultaneous processing scheme.

For evaluating the effects of a battery as an energy source, we used the work done by (R. Rao et al., 2003). Their work summarizes state-of-the-art battery modeling into four classes of models that capture the battery state and its non-linearities. The lowest class contains the physical models that are accurate and model battery state evolution through a set of ordinary and partial differential equations. However, they suffer from a significant level of complexity that reflects on the time needed to produce predictions. The work proceeds by showing empirical models, that predict battery state from empirical trials. The third class consists of abstract models that we incorporated into our

approach, in particular, by deriving the equation from the model developed by (Hasan et al., 2018) (they model battery state through an equivalent electrical circuit and its evolution in time). The fourth class consists of mixed models where experimental data are collected and subsequently refined with analytical expressions to determine the models' parameters.

System-level optimization techniques, such as dynamic voltage scaling, have been used for lowering the power consumption (Chowdhury and Chakrabarti, 2005; I. Hong et al., 1999; Luo and Jha, 2001). These techniques are available for hardware featuring dynamic voltage scaling and aim to achieve higher energy efficiency by including information about configuration parameters into the scheduler. They however focus on homogeneous systems, unlike our work which is designed to work for heterogeneous systems. This approach to modeling has nevertheless been extended to include GPU features (S. Hong and H. Kim, 2010), to heterogeneous systems (Bailey et al., 2014), to optimal software partitioning (Goraczko et al., 2008), and by Wu et al. to machine learning techniques (Wu et al., 2015). The work by Wu et al. mostly relies on neural networks and has been introduced only recently in the field of power estimation and modeling for heterogeneous systems. In particular, for a collection of applications, Wu et al. train a neural network by measuring a number of performance counters for different configurations. Even if these techniques perform well for defining static optimization strategies, they are generally not suitable for heterogeneous parallel systems in aerial robotics. In these cases, systems suffer from a considerable level of uncertainty, for which reason a statically defined energy model often would not model the real energy behavior. An overview of energy estimation in the context of machine learning approaches has recently been presented by (García-Martín et al., 2019). They present a literature review motivated by a belief that the machine learning community is unfamiliar with energy models, but do not relate to GPU-featured devices nor in general heterogeneous devices. In contrast, in our work we aim at automating the generation of application-level power estimation models that can be adapted for machine learning algorithms. Our approach does not yet take detailed scheduling decisions into account, unlike the black-box approach for CPU-GPU energy-aware scheduling presented by (Barik et al., 2016): they model the power by relating execution time to power consumption but otherwise do not focus on energy models.

Our approach shares the same principle of differentiating the microcontroller from the companion computer with (Mei et al., 2004, 2005). The controller acts on the actuators and reads the sensors, while the companion computer (can be found with different names in literature, such as secondary or embedded computer), performs computationally heavy operations. A similar approach for mobile robots is presented by (Dressler and Fuchs, 2005). However, both contributions neither elaborate further on computational elements of a heterogeneous platform, such as GPU, nor focus on different robots except the one under analysis.

The majority of other contributions in the literature focus on optimizing motion planning to increase power efficiency. For instance, approaches to minimize UAV power consumption, such as the work by (Kreciglowa et al., 2017), aim to determine the best trajectory generation method for an aerial vehicle to travel from one configuration to another. Uragun suggests the use of power-efficient components (Uragun, 2011): an energy-efficient UAV system can either be built using conceptual product development with emerging technologies or using energy-efficient components. Kanellakis et al. affirm that integrating visual sensors in the UAV ecosystem still lacks solid experimental evaluation (Kanellakis and Nikolakopoulos, 2017). They suggest that to save energy, the available payload for sensing and computing has to be restricted. Our approach towards energy modeling shares a similar principle as the one presented by (Sadrpour et al., 2013a,b) for Unmanned Ground Vehicles or UGVs. They propose a linear regression-based technique in the absence of real measurements and a Bayesian networks-based one in their presence. We used a simplified approximation technique to limit the number of computations needed while focusing rather on an accurate battery prediction.

To validate our approach and quantify its outcomes, we used the models previously developed for fine-grained energy modeling by (Nunez-Yanez and Lore, 2013), and (Nikov et al., 2015) respectively. In summary, fine-grained energy modeling uses hardware event registers to capture the CPU state under a representative workload. The energy-modeling consists of three stages. In data collection, the first stage, a benchmark runs on the platform and data are collected. The second and third stage, data processing and model generation, are performed offline on a different architecture. In these two stages, data are analyzed and a model that predicts possible future usage is generated.

Calore et al. develop an approach for measuring power efficiency for High-Performance Computing or HPC systems (Calore et al., 2015). An external board is used to measure the power consumption, while the data are collected from NVIDIA Jetson TK1 board running one benchmark. Our initial analysis presented at HLPGPU 2019 was made using a similar technique (Seewald, Ebeid, et al., 2019). A shunt resistor and digital multimeter integrated into the external board was used to evaluate the power efficiency. In this paper we extend our experiments to use internal power monitors and address a broader range of platforms. We now build a proper energy model that reflects the computational behavior of the device under study and shows the energy evolution. An early report on our work has been presented in the TeamPlay project's deliverable D4.3 (*Public Deliverables of the TeamPlay Horizon2020 Project* 2019) "Report on Energy, Timing and Security Modeling of Complex Architectures".

2.2 Motion Planning

Planning algorithms literature for mobile robots includes topics such as trajectory generation and path planning. Generally, the algorithms select an energy-optimized trajectory Mei et al., 2004, e.g., by maximizing the operational time Wahab et al., 2015. However, they apply to a small 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 et al., 2005. Given the availability of powerful GPU-equipped mobile hardware Rizvi et al., 2017, the use of computations is expected to increase in the near future Abramov et al., 2012; Jaramillo-Avila et al., 2019; Satria et al., 2016. More complex planning, which includes a broader concept of the plan being a set of tasks and a path, all focus on the trajectory Mei et al., 2005, 2006 and apply to a small number of robots Sadrpour et al., 2013a,b. For UAVs specifically, rotorcrafts have also gained interest in terms of algorithms for energy-optimized trajectory generation Kreciglowa et al., 2017; Morbidi et al., 2016.

2.3 Planning with Dynamics

2.4 Planning for Autonomous UAVs

2.4.1 Flight controllers

2.4.2 Energy models in aerial robotics

2.4.3 State estimation in aerial robotics

2.4.4 Optimal control in aerial robotics

2.5 Planning Computations with Motion

2.6 Summary

Chapter 3

Energy Models

A[A]

3.1 Models Classification

3.2 Energy Model of the Computations

Numerous new challenges have been introduced in embedded systems through evolution from small and predictable to large and complex architectures featuring different computational units. Limited energy availability, security considerations, and time requirements are among the most common challenges and significantly impact the level of autonomy of modern heterogeneous systems. These systems use many-core architectures, where a CPU executes along a general-purpose GPU or GPGPU with energy-efficient small cores. A parallel CPU-GPU execution is a core aspect to achieve the best possible energy efficiency and speed-up (Woo and Lee, 2008).

In this paper, we focus on limited energy availability for mobile heterogeneous devices powered by a battery and present a coarse-grained computation-oriented energy modeling approach.

The model *describes energy usage as a function of component configuration* and is based on physical measurements of power consumption of the on-board computing elements.

The term *computing* is used here to denote the energy consumed by the software components of a complex embedded system. Estimates indicate that more than 50% of power consumption of complex devices featuring mechanical and computational units can occur due to computational operations, with the motion accounting for the remaining (Mei et al., 2004, 2005). Energy efficiency concerns are not limited to

battery-powered devices, but also to the high-end systems (Mudge, 2001), and thus it is becoming more important for computer architects as power consumption is growing with the introduction of new processors.

Complex software applications are often composed of many components interacting concurrently to achieve a specific functionality. Our approach aims to predict the energy consumption of a provided set of software components, in a specific configuration, executed according to a given scheduling policy. Statistical methods are used to derive a predictive model allowing prediction of the components' total power consumption as scheduling and configuration parameters are varied. High-level modeling of using a battery as energy-source is used to model the energy that would be drained from the system during computation. The energy model is segmented into two layers: the measurement layer, which describes energy consumption as a function of time, and the predictive layer, which describes energy consumption as a function of component configuration and scheduling. The model generation process starts from the developer, who specifies the configuration space within which software components can run on a given system. A profiling process collects power samples for every component configuration (measurement layer). A model that maps configuration parameters to an energy metric, such as overall energy or average power, is generated from the data, showing the energy evolution for every possible configuration (predictive layer).

3.2.1 Computational energy of the UAV

We use software for drones as a case study for energy efficiency. Drones (known also as aerial robots, Unmanned Aerial Vehicles, or UAVs) are often composed of two computational units. A controller that manages real-time control of the physical aspects of the drone, and a companion computer that handles heavy computations. The latter is often a heterogeneous parallel system. A coarse-grained model of these systems can be used for energy-aware planning and optimization from a computational point of view. Moreover, energy constraints are of particular interest for drones, since their level of autonomy is directly proportional to the power-to-compute (Fabiani et al., 2007). Unlike their grounded counterparts, drones must trade-off size, weight, and power, resulting in strict limits to their energy source that may not accommodate their computational needs. Operating from a limited energy source, typically a lithium-ion battery, can significantly reduce the amount of autonomy that the drone can carry on. One of the reasons why this happens is the high energy demand required by complex parallel algorithms used in several use cases, from computer vision to data processing. To this extent, many tasks in aerial robotics are still not fully autonomous. Advanced operations that require a significant level of autonomy, such as path planning, obstacle avoidance, environment mapping, and object detection, often involve

human interaction. Thus there is still little difference between flying robots and their manned counterparts, except that the pilot operates from a ground station rather than onboard (Siciliano and Khatib, 2016).

As a concrete example, we present the drone use case that recognize an object while performing some computationally heavy operations. The use case is composed of two components, object detection algorithm (`darknet-gpu`) that recognize a pattern using a neural network, and matrix exponentiation (`matrix-gpu`) that simulates heavy operations with varying scheduling patterns using sleep of different durations between consecutive operations. Both the components execute mostly on GPU. In a configuration file, the developer specifies these parameters per component:

a) for `darknet-gpu` a frame-per-seconds rate [5.8, 32] from the lowest acceptable, to the highest achievable, and

b) for `matrix-gpu` two sets of rates, the matrix size [256, 4096] from smallest to largest, and sleep interval [0, 10] from no sleep to 10 seconds.

A profiling process generates models that map time to power, one per combination in the configuration file. Based on this a high-level model is generated which combines component parameters to energy consumption. The energy consumption is estimated for any parameter combinations not profiled. This information is useful to define a proper trade-off between parameters and decide eventually, what configuration has to be used to optimize energy consumption. The information about the battery state, included in the analysis, enables the developer to select a specific runtime configuration (for instance, one that will allow completing the mission without recharging the battery by just adjusting the combinations of components and their parameters).

In our experiments, the resulting model shows an almost linear behavior for the `darknet-gpu` component, highlighting the energy behavior for any configuration in the interval [5.8, 32]. The `matrix-gpu` model shows the energy evolution with different scheduling options and addresses the impact of different scheduling to the battery depletion. Results for both components are presented and further discussed in Section ??.

With this work we have:

1. designed a computation-oriented energy modeling approach for heterogeneous platforms,
2. evaluated the outcomes of the model on several benchmarks and devices,
3. assessed the model's validity against external measurements and a fine-grained approach, and
4. developed a profiling tool for computation-oriented energy modeling.

The tool, named `powprofiler`, is written in C++ and distributed under MIT license.[1] The tool was used to perform all of the experiments reported in this paper and is designed to be used, among others, in mobile robotics scenarios. It can profile a set of components being executed in a number of possible configurations and from this, build an energy model. The model allows the system under analysis to define an appropriate tradeoff between consumed energy and computations performed. This information can be used to increase energy efficiency, often linked to the level of autonomy in many modern scenarios, and meet the power requirements. For example, a drone can dynamically adjust autonomous detection capabilities to fit the current battery state of charge (referred to in this paper as SoC). Similarly, an autonomous vehicle can increase the amount of computation only when the collision detection system is required, and a mobile robot can automatically schedule tasks to compute in an energy-aware fashion. In these examples, awareness of precise computational energy cost can potentially lead to significant energy savings. Key to the approach is flexibility in terms of easy support for different heterogeneous platforms and extensibility to other classes of systems outside the mobile robotics domain.

3.2.2 Energy modeling for heterogeneous hardware

In this paper, we aim to provide a tool for automatic generation of hardware-specific energy models of a given application. The energy model is generated through a profiling process which collects several power samples and builds an energy abstraction upon them. Computational energy is addressed by mapping a component configuration to an energy metric, such as average power or overall energy. Our approach includes several on-board computing elements, such as CPU and GPU, and analyzes their role in the tradeoff related power decisions.

Figure 3.1 shows an overview of our approach. For a specific application on the system under study, the developer describes how components behave in a configuration file. For each component, any parameter range or value can be specified. By varying the parameters and scheduling policy, the tool generates n combinations of components called configurations $\mathcal{D} = \{d_1, d_2, \cdots, d_n\}$. Once the set $\mathcal{D}$ is generated, the profiling process starts by collecting a set of samples and by generating a layer 1 model for every configuration d_k (with $1 \leq k \leq n$). The layer 1 model maps a time interval t to a power measure $f(t)$ and estimates the power-to-compute value for a given component. Later, the model is integrated with battery state evolution and quantified in the amount of SoC left in the battery. Finally, both $f(t)$ and SoC are used to generate a layer 2 model, that maps a set of configurations $\mathcal{D}$ to the energy metric (i.e., average power consumption or overall energy usage) and SoC $g(\mathcal{D})$. This information can be

[1] `https://bitbucket.org/adamseew/powprofiler`

used to select the best configuration and define a better scheduling policy to optimize the average computational energy.

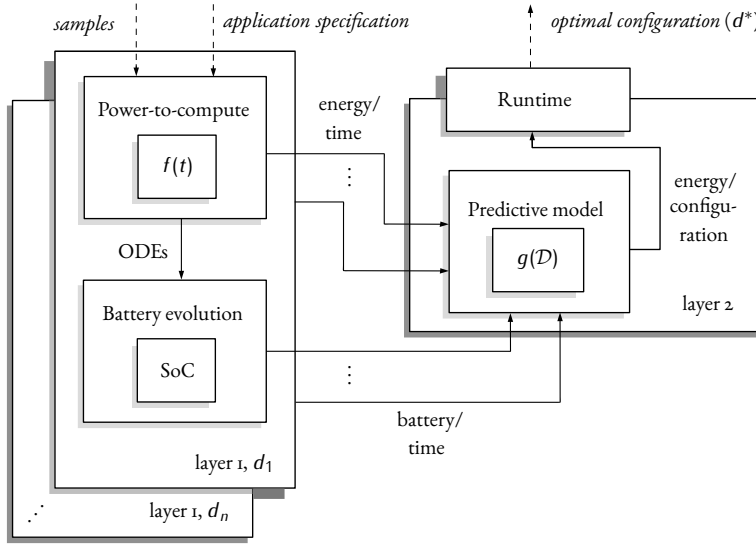

Fig. 3.1. Overview of energy modeling for heterogeneous hardware.

The modeling approach is tested across four different platforms, ODROID XU3, NVIDIA TK1, TX2, and Nano (see Section ?? for details), and a selection of benchmark components is used. Each component has a set of configuration parameters varied inside defined boundaries, that represent the acceptable rate of quality of service or QoS. We use the following benchmark components: a random matrix of a predefined size that is multiplied by another or exponentiated by a given exponent on CPU (`matrix-cpu`) or GPU (`matrix-gpu`). A computer vision component built upon the darknet (Redmon, 2013–2016; Redmon and Farhadi, 2017) implementation of the YOLO library (Redmon, Divvala, et al., 2016), that uses a deep neural network to detect an object from a video stream on CPU (`darknet-cpu`) or GPU (`darknet-gpu`), and NVIDIA custom-made benchmarks, modified so they can be iterated several times over a time interval, that perform matrix multiplication (`nvidia-matrix`) and quicksort (`nvidia-quicks`) of a given size on GPU.

3.2.3 Measurement layer

The measurement layer describes an energy sample for a given configuration and scheduling of a component executed on specific hardware as a function of time. It provides average power or overall energy over a time interval for every computational unit that allows power measurement.

The developer provides for every application a number of components in the configuration file along with their parameters. An application in this layer is structured as several configured components that execute in parallel according to a specific scheduling policy. First, immediate power consumption is measured. This information is used to model the power into the power-to-compute and battery drain. An *application specification*, described later in this section, is used to define all the possible configurations of an application in the configuration file. The power is measured throughout a time interval set by the developer. Multiple metrics can be captured at this level and sampled at a fixed rate specified in the application specification, including CPU, GPU, and total power. Other metrics, as the system's load and sensors' evolution, can be added by extending the approach. Based on the information obtained at this layer, the energy that the computation would draw from a battery can be estimated using the *battery model*, also described later in this section. As a concrete example, Figure ?? shows the total energy usage as a function of time for CPU and GPU of the NVIDIA TX2 board executing the `darknet-gpu` component in four different QoS configurations.

3.2.4 Application specification

The application specification is used to define all the computations to run for a specific energy model. It depends on the schedule that is used when performing measurements, in the sense that scheduling policies will be specific to the kind of scheduling supported by this scheduler. Currently, only a basic application specification is supported that corresponds to the capabilities of the `powprofiler` tool described later in Subsection ??. Support for more advanced models is considered future work. In particular, a dynamic measurement-oriented scheduler should be adopted in the future to define the optimal scheduling policy during computation.

The basic application specification is defined as follows. The choice between component implementations (if needed) is implicitly supported by requiring the developer to select the specific component implementation to use for the measurement. Components are configured by providing concrete, fixed values for all of their configuration parameters. A scheduling policy is implicitly supported by requiring each component to provide parameters that explicitly control fixed scheduling patterns in time (e.g., what rate to execute, when to execute). Components are assumed to implicitly control scheduling in space, for example by having different implementations for CPU and GPU (and hence being explicitly selected as part of the application configuration).

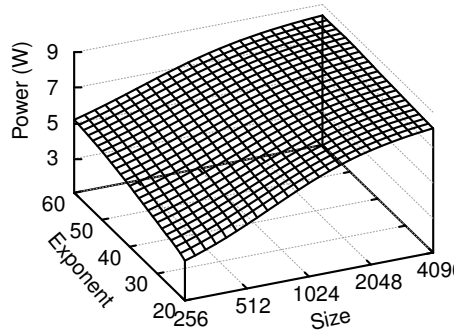

Fig. 3.2. Average power

3.2.5 Predictive layer

The predictive model is expressed in terms of the measurement layer and describes "average power over time frame" and other similar coarse-grained metrics as a function of component configuration parameters and scheduling policies.

The predictive model concerns the average power over a time frame as a function of varying application models (i.e., configuration parameters, scheduling policy, …). The average power can be any of the outputs of the measuring layer, such as average power used by the embedded board, or average power drained from a battery. First-layer measurements are used to generate functions that map application models to energy metrics. Any aspect of the application model can be varied, as defined by the *sampling strategy* described later in this section. Varying selected aspects of the application model corresponds to changes in component configuration parameters and scheduling policies, thus making it possible to determine the average power as a function of specific variations of selected component configuration parameters and aspects of the scheduling policy. Due to the potentially large configuration space, it is critical that a model covering all relevant parameter/scheduling variations can be generated from a subset of samples. This is handled by the *approximation method*, also described later in this section.

As a concrete example, Figure 3.3 shows the total average energy usage of the TX2 board executing the `matrix-gpu` component as a function of the size and exponent parameters. Results are discussed in further detail in the context of experimental results, see Section ??.

3.2.6 Hardware platforms and benchmarks

In the following section, we present hardware platforms under study, the two measuring technique to obtain power metrics, and a summary of the profiling tool. Finally,

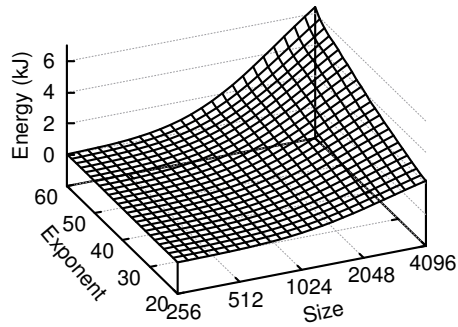

Fig. 3.3. Overall energy

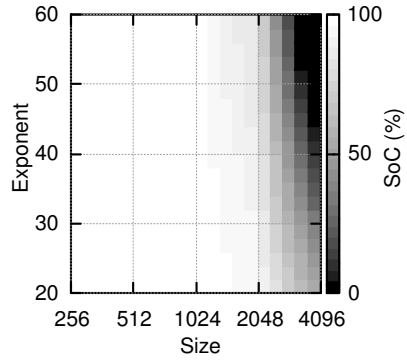

Fig. 3.4. Remaining battery capacity

Experimental Methodology outlines our experimental approach before introducing results in the next Section.

We studied four different hardware platforms, ODROID XU3, NVIDIA Jetson TK1, TX2, and Nano that are shown in Figure ??

ODROID XU3 Provides an ARM Cortex-A15 and -A7 CPU, 2 GBytes of LPDDR3 RAM, a MALI GPU, and storage via microSD. It includes built-in sensors that enable accurate power measurements of the two CPUs, RAM, and GPU.

NVIDIA TK1 Provides an ARM Cortex-A15 CPU, 2 GBytes of DDR3L RAM, 16 GB of non-volatile storage, and an NVIDIA Kepler GPU with 192 CUDA cores. The TK1 does not provide any built-in sensor for power measurement, so an external sensor is used to measure the total power consumed.

NVIDIA TX2 Provides an ARM Cortex-A57 CPU, 8 GBytes of LPDDR4 RAM, 32 GB of non-volatile storage, and an NVIDIA Pascal GPU with 256 CUDA cores. The TX2 comes with on-board power measurement functionality that can measure the power of different components, and was used in our model to

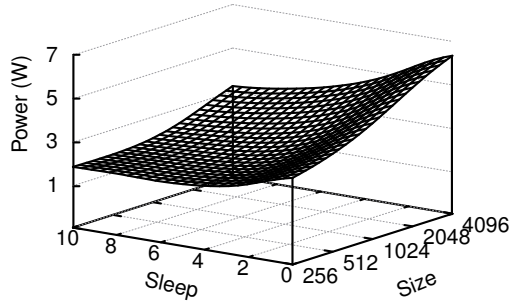

Fig. 3.5. Average power

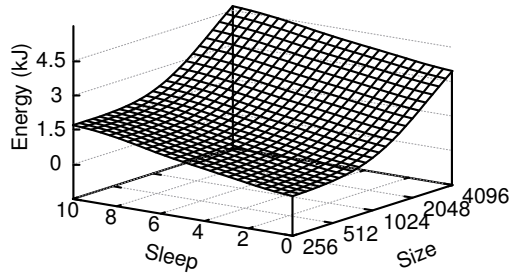

Fig. 3.6. Overall energy

gather independently the instantaneous power usage of the CPU, GPU, and the whole board.

NVIDIA Nano Provides an ARM Cortex-A57 CPU, 4 GBytes of LPDDR4 RAM, an NVIDIA Maxwell GPU with 128 CUDA cores, and storage via microSD. Similarly to the TX2, the Nano comes with on-board power measurement functionality that is able to measure the power of different components and can be used within our model to gather independently the instantaneous power usage of the CPU, GPU, and the whole board.

Platform	CPU	GPU	RAM	Memory	Board	Sensors
ODROID	A15, A7	MALI	2 GB	microSD	94x70	✓
TK1	A15	Kepler	2 GB	16 GB	125x95	-
TX2	A57	Pascal	8 GB	32 GB	170x170	✓
Nano	A57	Maxwell	4 GB	microSD	100x80	✓

Table 1. Summary of hardware platforms under analysis.

Several benchmarks (see Section ??) have been performed on the four hardware platforms under study. (Not all benchmarks components are compatible with every

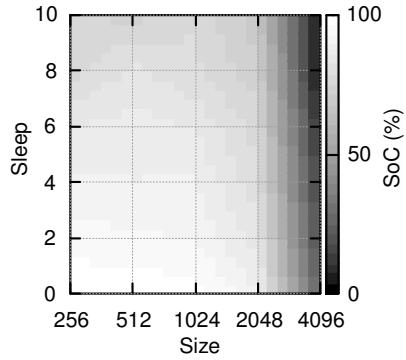

Fig. 3.7. Remaining battery capacity

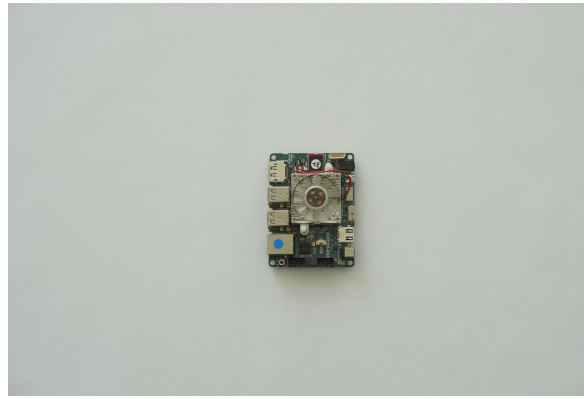

Fig. 3.8. ODROID XU3

platform, e.g., GPU-based benchmarks are implemented in CUDA which only runs on NVIDIA platforms, limiting the set of experiments that can be performed.)

3.2.7 Power measurements

We use two different power measurements techniques for our experiments: the current shunt and current probe. With the current shunt method, an internal circuit on the embedded board composed of a shunt resistor is used, along with a digital acquisition unit. It is the preferred approach as it can precisely sample the power consumption of specific computing elements as CPU and GPU. Such circuits are present on the ODROID XU_3 and NVIDIA TX_2 or Nano boards and can be sampled directly by the `powprofiler` tool described in Subection ??.

As an alternative to the current shunt method, we have also implemented the current probe method, where an external circuit measures the power on the connection

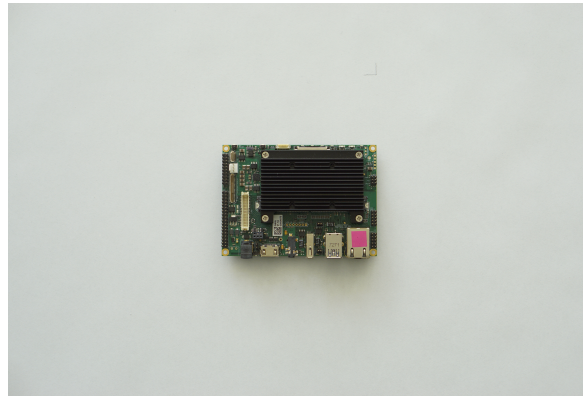

Fig. 3.9. NVIDIA Jetson TK1

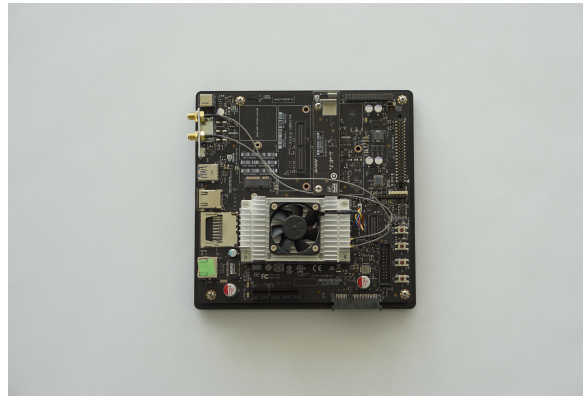

Fig. 3.10. NVIDIA Jetson TX2

to the main embedded board (i.e., the carrier board). We adopted the current probe method to measure the overall power consumption of the NVIDIA TK1. This measurement setup consists of three hardware units, henceforth referred to as nodes. The first node is the board under analysis, the second node a multimeter that performs the sampling of the power consumption at a specific sampling frequency, and the third node is an external computer that collects the data for subsequent processing by `powprofiler`. The whole setup is described in detail in our prior work (Seewald, Ebeid, et al., 2019). Data and metrics are stored for use with `powprofiler`, essentially allowing an arbitrary power measurement technique for the system under analysis. This means that a layer 2 model can be generated from the data with no distinction of what type of measuring device was used to obtain the layer 1 model.

All experiments reported in this section are performed using the standard performance governor present in each of the systems under observation. In particular, dy-

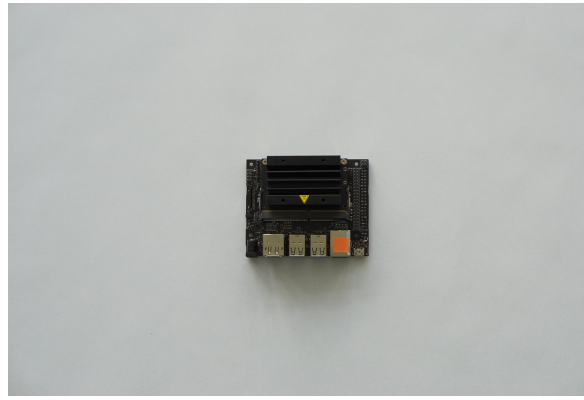

Fig. 3.11. NVIDIA Jetson Nano

Component \ Platform	Odroid-XU3	NVIDIA		
		TK1	TX2	Nano
CPU	A15+A7	A15	A57	A57
GPU	MALI	Kepler	Pascal	Maxwell
`matrix-cpu`	✓	✓	✓	✓
`matrix-gpu`	-	✓	✓	✓
`darknet-cpu`	(✓)	(✓)	✓	(✓)
`darknet-gpu`	-	-	✓	(✓)
`nvidia-matrix`	-	(✓)	✓	(✓)
`nvidia-quicks`	-	(✓)	✓	(✓)

Table 2. Benchmark components: platforms and modeling approach. ✓: supported and included in this paper, (✓): supported but not included in this paper, -: not supported.

namic effects such as DVFS have not been disabled. Our experiments have not revealed any effects on performance due to, e.g., ambient temperature or the temperature of the embedded systems increasing.

3.2.8 The `powprofiler` tool

The `powprofiler` tool implements all the features described in this paper and is designed to be useable for heterogeneous parallel embedded systems. The tool supports all the platforms described in Section ??. It is designed for extensibility and can, in principle, support any Linux-based embedded device that provides power measurement metrics. Concretely, to support a new embedded device, a subclass defining the device-specific features can be derived from a virtual class, and only requires the implementation of a function that returns the current measurements in the form of

a weighted vector where every element can represent a different metric. In a first iteration, the tool profiles a set of configurations of a component and builds a layer 1 model as described in Section ??. Once `powprofiler` obtains power profiles and other metrics that are considered useful for the purpose of energy modeling, it builds a predictive model as described in Section ??. The tool relies on `gnuplot` utility to visualize energy evolution if the configuration search space allows it. Automatically generated output includes CSV files containing measured data as well as the 2D- and 3D-plots shown in this paper.

3.2.9 Energy-aware design of algorithms

3.2.10 ROS middleware

3.2.11 Experimental methodology

The power modeling experiments described in this paper are, unless otherwise mentioned, done using `powprofiler` tool. The tool uses two temporal schemes to profile data. A component can be profiled for a specific amount of time with the first, or until it terminates its computation with the second. The second scheme is used for all the components that end in a specific amount of time, i.e., matrix multiplication. The first for components that run continuously, i.e., an image detection algorithm operating on an image stream. For both, the longer the profiling, the better the accuracy.

Ideally, any component combination can be profiled by specifying possible sets of parameters in the configuration file. Each entry corresponds to a component and defines how its execution is varied as described in application specification described in Subsection 3.2.4. In summary, a layer 1 model is evaluated and stored for every possible combination. The layer 1 model is then integrated with the battery model. Data containing energy consumption, battery drain, and other information as system load, are evaluated into the layer 2 model. This model builds an n-dimensional function and provides energy consumption data for every possible combination. The missing data are integrated using estimation with the approximation technique describe in Subsection ??. The n-dimensional space is derived by the use of the battery model to map each value with a weight describing the SoC.

As an example of energy modeling, we implemented a combination of three different parameters and generated a surface that shows approximately where the minimum power consumption is expected. Figure 3.3 and Figure 3.7 show how the output of the predictive model includes, but is not limited to, overall energy in Figure 3.3 and Figure 3.6 and average power consumption in Figure 3.2 and Figure 3.5 for all the combinations.

3.3 Battery Model

The battery model is a mathematical abstraction that models draining energy from a battery used to power the computation being measured. It was derived from the state-space problem representation of the empirical battery model through an equivalent electrical circuit, presented by Hasan et al. Hasan et al., 2018. The battery evolution model can be represented in this way using an ordinary differential equation that is a function of the power drained from the system, and represents the SoC of the battery at a given instant

$$\frac{d}{dt}\mathrm{SoC}(t) = -\frac{I_{\mathrm{int}}(t)}{Q_c}, \tag{3.1}$$

$$I_{\mathrm{int}}(t) = \frac{U_{\mathrm{int}} - \sqrt{U_{\mathrm{int}}^2 - 4 \cdot R_{\mathrm{int}} \cdot U_{\mathrm{sta}} \cdot I_{\mathrm{load}}(t)}}{2 \cdot R_{\mathrm{int}}}, \tag{3.2}$$

where Q_c is the constant nominal capacity, U_{int} is the internal battery voltage, I_{int} is the current load that depends on the power requirements, R_{int} is the internal resistance of the battery, U_{ext} is the external battery voltage, U_{sta} is the stabilized voltage (a fixed value determined by the load of the system), and I_{load} is the current required by the load. The constants are chosen to describe a small drone battery. The external battery voltage U_{ext} can be also expressed as follows:

$$U_{\mathrm{ext}}(t) = U_{\mathrm{int}} - R_{\mathrm{int}} \cdot I_{\mathrm{int}}(t) \tag{3.3}$$

The approach allows modeling the computational system not only in terms of the overall power consumption but also in terms of the actual amount of power drained from the battery. This is especially important with a mobile robot dependent on a limited and time-dependent energy supply, running computationally heavy parallel algorithms. Critically, although this mathematical model is a simple abstraction, it models how a stable power consumption over time can induce less drain from the battery compared to using the same amount of energy distributed in a number of spikes. The battery model allows this information to be determined for specific usage scenarios and thus provides a useful way to determine what configuration corresponds to the best power usage combination for a mobile use case.

3.3.1 UAV batteries

3.3.2 Derivation of differential battery model

3.4 Energy Model of the Motion

3.4.1 Mechanical energy of the UAV

3.4.2 Experimental methodology

3.5 Periodic Energy Model

3.5.1 Furier series of empirical data

3.5.2 Derivation of differential periodic model

We refer to the instantaneous energy consumption evolution simply as the energy signal. We model the energy using energy coefficients $\mathbf{q} \in \mathbb{R}^m$ that characterize such energy signal. The coefficients are derived from Fourier analysis (the size of the energy coefficients vector m is related to the order of a Fourier series) and estimated using a state estimator.

We prove a relation between the energy signal and the energy coefficients in Lemma 3.5.1. We show after the main results how this approach allows us variability in terms of non-periodic signals.

After having illustrated the energy model, we enhance it with the energy contribution of the path in and of the computations in Subsection 3.5.4.

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

$$h(t) = a_0/T + (2/T)\sum_{j=1}^{r}\left(a_j \cos \omega j t + b_j \sin \omega j t\right), \tag{3.4}$$

where $h : \mathbb{R}_{\geq 0} \to \mathbb{R}$ maps time to the instantaneous energy consumption, $\omega := 2\pi/T$ is the angular frequency, and $a, b \in \mathbb{R}$ the Fourier series coefficients.

The energy signal can be modeled by Equation (3.4) and by the output of a linear model

$$\dot{\mathbf{q}}(t) = A\mathbf{q}(t) + B\mathbf{u}(t), \tag{3.5a}$$
$$y(t) = C\mathbf{q}(t), \tag{3.5b}$$

where $y(t) \in \mathbb{R}$ is the instantaneous energy consumption.

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

$$\mathbf{q}(t) = \begin{bmatrix} \alpha_0(t) & \alpha_1(t) & \beta_1(t) & \cdots & \alpha_r(t) & \beta_r(t) \end{bmatrix}^T, \tag{3.6}$$

where $\mathbf{q}(t) \in \mathbb{R}^m$ with $m = 2r + 1$. The state transition matrix

$$A = \begin{bmatrix} 0 & 0^{1\times 2} & 0^{1\times 2} & \dots & 0^{1\times 2} \\ 0^{2\times 1} & A_1 & 0^{2\times 2} & \dots & 0^{2\times 2} \\ 0^{2\times 1} & 0^{2\times 2} & A_2 & \dots & 0^{2\times 2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0^{2\times 1} & 0^{2\times 2} & 0^{2\times 2} & \dots & A_r \end{bmatrix}, \tag{3.7}$$

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

$$A_j := \begin{bmatrix} 0 & \omega j \\ -\omega j & 0 \end{bmatrix}, \tag{3.8}$$

The output matrix

$$C = (1/T) \begin{bmatrix} 1 & 1 & 0 & \cdots & 1 & 0 \end{bmatrix}, \tag{3.9}$$

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

The linear model in Equation (3.5) allows us to include the control in the model of Equation (3.4).

Lemma 3.5.1 (Signal, output equality)
Suppose control $\mathbf{u}$ is a zero vector, matrices A, C are described by Equation (3.6), and the initial guess $\mathbf{q}_0$ is

$$\mathbf{q}_0 = \begin{bmatrix} a_0 & a_1/2 & b_1/2 & \cdots & a_r/2 & b_r/2 \end{bmatrix}^T.$$

Then, the signal h in Equation (3.4) is equal to the output y in Equation (3.5).

Proof. We propose a formal proof of the lemma. The proof justifies the choice of the items of the matrices A, C and of the initial guess $\mathbf{q}_0$ in Equation (3.6). 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 (3.4) in its complex form with the well-known Euler's formula

$$e^{it} = \cos t + i \sin t. \tag{3.10}$$

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

$$\cos \omega j t = (e^{i\omega j t} + e^{-i\omega j t})/2 \tag{3.11a}$$
$$\sin \omega j t = (e^{i\omega j t} - e^{-i\omega j t})/(2i) \tag{3.11b}$$

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

$$\begin{aligned} h(t) = a_0/T + (1/T) \sum_{j=1}^{r} e^{i\omega j t}(a_j - i b_j) + \\ (1/T) \sum_{j=1}^{r} e^{-i\omega j t}(a_j + i b_j), \end{aligned} \tag{3.12}$$

where i is the imaginary unit.

The solution at time t can be expressed

$$\mathbf{q}(t) = e^{At}\mathbf{q}_0. \tag{3.13}$$

Both the solution and the system in Equation (3.5) are well established expressions derived using standard textbooks Kuo, 1967; Ogata, 2002. To solve the matrix exponential e^{At}, we use the eigenvectors matrix decomposition method Moler and Van Loan, 2003.

The method works on the similarity transformation

$$A = VDV^{-1}. \tag{3.14}$$

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

$$e^{At} = Ve^{Dt}V^{-1}. \tag{3.15}$$

We consider the non-singular matrix V, whose columns are eigenvectors of A

$$V := \begin{bmatrix} v_0 & v_1^0 & v_1^1 & \dots & v_r^0 & v_r^1 \end{bmatrix}. \tag{3.16}$$

We then consider the diagonal matrix of eigenvalues

$$D = \text{diag}(\lambda_0, \lambda_1^0, \lambda_1^1, \dots, \lambda_r^0, \lambda_r^1). \tag{3.17}$$

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

$$Av_j = \lambda_j v_j \;\; \forall j = \{1, \dots, m\}, \tag{3.18}$$

and

$$AV = VD. \tag{3.19}$$

We apply the approach in terms of Equation (3.5), under the assumptions made in the lemma (the control is a zero vector)

$$\dot{\mathbf{q}}(t) = A\mathbf{q}(t). \tag{3.20}$$

The linear combination of the initial guess and the generic solution

$$F\mathbf{q}(0) = \gamma_0 v_0 + \sum_{k=0}^{1}\sum_{j=1}^{r} \gamma_j v_j^k, \tag{3.21a}$$

$$F\mathbf{q}(t) = \gamma_0 e^{\lambda_0 t} v_0 + \sum_{k=0}^{1}\sum_{j=1}^{r} \gamma_j e^{\lambda_j t} v_j^k, \tag{3.21b}$$

where

$$F = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \tag{3.22}$$

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

Let us consider the second expression in Equation (3.21). It represents the linear combination of all the coefficients of the state at time t. It can also be expressed in the following form

$$\begin{aligned} F\mathbf{q}(t)/T = \gamma_0 e^{\lambda_0 t} v_0/T + (1/T) \sum_{j=1}^{r} \gamma_j e^{\lambda_j^0 t} v_j^0 + \\ (1/T) \sum_{j=1}^{r} \gamma_j e^{\lambda_j^1 t} v_j^1 . \end{aligned} \tag{3.23}$$

We proof that the eigenvalues λ and eigenvectors V are such that Equation (3.23) is equivalent to Equation (3.12).

The matrix A is a block diagonal matrix, so we can express its determinant as the multiplication of the determinants of its blocks

$$\det(A) = \det(0) \det(A_1) \det(A_2) \cdots \det(A_r). \tag{3.24}$$

We proof the first determinant and the others separately.

Thereby we start by proofing that the first terms of the Equations (3.12–3.23) match. We find the eigenvalue from $\det(0) = 0$, which is $\lambda_0 = 0$. The corresponding eigenvector can be chosen arbitrarily

$$(0 - \lambda_0) v_0 = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}, \tag{3.25}$$

$\forall v_0$, thus we choose

$$v_0 = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}. \tag{3.26}$$

We find the value γ_0 of the vector γ so that the terms are equal

$$\gamma_0 = \begin{bmatrix} a_0 & 0 & \cdots & 0 \end{bmatrix}. \tag{3.27}$$

Then, we proof that all the terms in the sum of both the Equations (3.12–3.23) match.

For the first block A_1, we find the eigenvalues from

$$\det(A_1 - \lambda I) = 0. \tag{3.28}$$

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

$$\lambda_1^0 = i\omega, \tag{3.29a}$$

$$\lambda_1^1 = -i\omega. \tag{3.29b}$$

The eigenvector associated with the eigenvalue λ_1^0 is

$$v_1^0 = \begin{bmatrix} 0 & -i & 1 & 0 & \cdots & 0 \end{bmatrix}^T. \tag{3.30}$$

The eigenvector associated with the eigenvalue λ_1^1 is

$$v_1^1 = \begin{bmatrix} 0 & i & 1 & 0 & \cdots & 0 \end{bmatrix}^T. \tag{3.31}$$

Again, we find the values γ_1 of the vector γ such that the equivalences

$$\begin{cases} e^{i\omega t}(a_1 - ib_1) & = \gamma_1 e^{i\omega t} v_1^0 \\ e^{-i\omega t}(a_1 + ib_1) & = \gamma_1 e^{i\omega t} v_1^1 \end{cases} \tag{3.32}$$

hold. They hold for

$$\gamma_1 = \begin{bmatrix} b_1 & a_1 \end{bmatrix}. \tag{3.33}$$

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

The initial guess is build 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 Equations (3.12–3.23) and Equation (3.4). 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 a half the corresponding coefficients in $\mathbf{q}_0$). To match the outputs $h(t) = y(t)$, or equivalently

$$F\mathbf{q}(t)/T = C\mathbf{q}(t), \tag{3.34}$$

we define

$$C := (1/T)\begin{bmatrix} 1 & 1 & 0 & \cdots & 1 & 0 \end{bmatrix}. \tag{3.35}$$

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

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

$$C := d\begin{bmatrix} 1 & 1 & 0 & \cdots & 1 & 0 \end{bmatrix}, \tag{3.36}$$

equivalent to

$$C := d\begin{bmatrix} 1 & 0 & 1 & \cdots & 0 & 1 \end{bmatrix}, \tag{3.37}$$

or

$$C := d\begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}, \tag{3.38}$$

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

3.5.3 Derivation of the nominal control

Let us suppose that at time instant t the plan reached the i-th stage Γ_i and the control

$$\mathbf{c}_i(t) = \begin{bmatrix} c_i^{\rho}(t) & c_i^{\sigma}(t) \end{bmatrix}^T, \tag{3.39}$$

where $\mathbf{c}_i(t) \in \mathbb{R}^n$ with $n := \rho + \sigma$ differs from the nominal control $\mathbf{u}(t)$ in Equation (3.5). We include the control in the nominal control exploiting the following observation.

Observation
We observe that:

- A change in path parameters affects the energy indirectly. It alters the time when the UAV reaches the final point $\mathbf{p}_{\Gamma_i}$.
- A change in computation parameters affects the energy directly. It alters the instantaneous energy consumption as more computations require more power (and vice versa).

We use this information later in the algorithm to check that the battery discharge time is greater and replan the path parameters accordingly. We replan the computation parameters to maximize the instantaneous energy consumption against the maximum battery discharge rate.

The nominal control is

$$\mathbf{u}(t) := \hat{\mathbf{u}}(t) - \hat{\mathbf{u}}(t-1), \tag{3.40}$$

where $\hat{\mathbf{u}}(t)$ is defined as the energy estimate of a given control sequence at time instant t, $\hat{\mathbf{u}}(t-1)$ at the previous time instant $t-1$

$$\hat{\mathbf{u}}(t) := \mathrm{diag}(v_i)\mathbf{c}_i(t) + \tau_i. \tag{3.41}$$

The input matrix is then

$$B = \begin{bmatrix} 0^{1\times\rho} & 1 & \cdots & 1 \\ 0^{1\times\rho} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0^{1\times\rho} & 0 & \cdots & 0 \end{bmatrix}, \tag{3.42}$$

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

$$\nu_i = \begin{bmatrix} \nu_i^\rho & \nu_i^\sigma \end{bmatrix}^T, \tag{3.43a}$$

$$\tau_i = \begin{bmatrix} \tau_i^\rho & \tau_i^\sigma \end{bmatrix}^T, \tag{3.43b}$$

scaling factors quantifying the contribution to the plan of a given parameter in terms of time for the first ρ parameters, and power 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 $\mathbf{u}(t-1), \mathbf{u}(t)$ applied to the system.

$B\mathbf{u}(t)$ merely includes the difference in power into the model in Equation (3.5).

3.5.4 Merging computations and motion

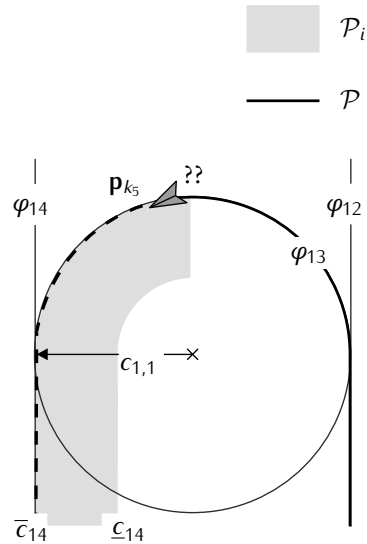

Fig. 3.12. .

The set

$$\mathcal{P}_i := \{\mathbf{p}_k \mid \varphi_i(\mathbf{p}_k, c_i^\rho) \in \mathcal{C}_i\}, \tag{3.44}$$

delimits the area where the i-th path φ_i is free to evolve using the path parameters c_i^ρ (the gray area in Figure 3.14). φ_i is a function of the two coordinates and the path parameters, and is equal to zero when a point $\mathbf{p}_k$ is on the path. Physically, this means the UAV is flying exactly over the nominal trajectory.

The path parameters allows to change the path. They are a way to alter the nominal trajectory in the initial plan and thus alter the energy by changing the flying time in

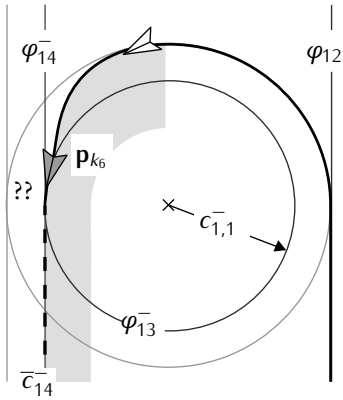

Fig. 3.13. The alteration of the path parameter $c_{1,1}$, the radius of the circle.

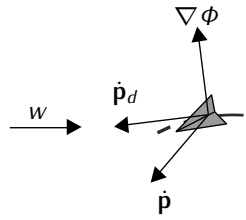

Fig. 3.14. .

the example in Figure ??. In fact, the algorithm uses the set from Equation (3.44) to find the path parameters such that the plan consisting of flying φ_i has the highest energy, while still respecting the constraints. In Figure 3.14, the parameter radius of the circle $c_{1,1}$ is replanned as, e.g., averse atmospheric conditions do not allow to terminate the plan.

We derive the new position $\mathbf{p}_{k+1}$ computing the vector field

$$\nabla\varphi_i := \begin{bmatrix} \partial\varphi_i/\partial x \\ \partial\varphi_i/\partial y \end{bmatrix}, \tag{3.45}$$

and the direction to follow in the form of velocity vector De Marina et al., 2017

$$\dot{\mathbf{p}}_d(\mathbf{p}_k) := E\nabla\varphi_i - k_e\varphi_i\nabla\varphi_i, \tag{3.46}$$

where E specifies the rotation (it influence the tracking direction). For instance

$$E = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \tag{3.47}$$

is the counter clockwise direction, $-E$ the clockwise direction.

$k_e \in \mathbb{R}_{\geq 0}$ is the gain to adjusts the speed of convergence. The direction the velocity vector $\dot{\mathbf{p}}_d$ is pointing at is generally different from the course heading $\dot{\mathbf{p}}$ due to the atmospheric interferences (wind $w \in \mathbb{R}$ in the top of Figure 3.14).

The scaling factors for the path parameters from Equation (3.40) are derived empirically. For the example in Figure 3.14, we can obtain the scaling factor $\nu_{1,1}$ measuring the time needed to compute the path with the lowest configuration $\underline{c}_1$, $\underline{t}$ and the highest $\overline{t}$.

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

$$\nu_{i,j} = \left((\overline{t} - \underline{t}) / (\overline{c}_{i,j} - \underline{c}_{i,j}) \right) / \rho, \tag{3.48a}$$

$$\tau_{i,j} = \left(\underline{c}_{i,j} (\underline{t} - \overline{t}) / (\overline{c}_{i,j} - \underline{c}_{i,j}) + \underline{t} \right) / \rho, \tag{3.48b}$$

$\forall j \in [\rho]^+$. Moreover, let the factors be zero when the parameters set $c_i^\rho = \{\emptyset\}$.

Whenever the trajectory parameters are not equally distributed, one can define $(y_{\overline{c}_i} - y_{\underline{c}_i})$ as a the highest (and lowest) levels of specific trajectory parameters.

Let us recall from Definition 1.5.3 that the i-th stage Γ_i of the plan Γ contains the computation parameters which characterize the computations. We estimate the energy cost of these computations using `powprofiler`, the open-source modeling tool adapted from earlier work on computational energy analysis (Seewald, Schultz, Ebeid, et al., 2021; Seewald, Schultz, Roeder, et al., 2019), and energy estimation of a fixed-wing UAV (Seewald, Garcia de Marina, et al., 2020).

For this purpose, we assume the UAV carries an embedded board that runs the computations. Our tool measures the instantaneous energy consumption of a subset of possible computation parameters within the computation constraint sets and builds an energy model: a linear interpolation, one per each computation.

The computations are implemented by software components, e.g., Robot Operating System (ROS) nodes in a ROS-based system (Quigley et al., 2009). The user implements these nodes such that they change the computational load according to node-specific ROS parameters–the computation parameters. In a generic software component system, the user maps the computational load to the arguments (Seewald, Schultz, Roeder, et al., 2019). In both cases, with ROS (Zamanakos et al., 2020) or with generic software components system (Seewald, Schultz, Roeder, et al., 2019), the tool performs automatic modeling. For instance, if the computation is an object detector, a computation parameter $c_{1,2}$ might correspond to frames-per-second (fps) rate. The tool then measures power according to the detection frequency.

We note that while the path can differ for each stage, the tasks remain the same. However, the user can inhibit or enable a computation varying its computation constraint set.

Let us define the output from `powprofiler` formally.

Definition 3.5.1 (Instantaneous computational energy)
$g : \mathbb{Z}_{\geq 0} \to \mathbb{R}_{\geq 0}$ as the instantaneous computational energy consumption value obtained using the `powprofiler` tool.

The scaling factors add the computational energy component to the model in Equation (3.5). They are derived similarly to Equation (3.48)

$$\nu_{i,j} = (g(\overline{c}_{i,j}) - g(\underline{c}_{i,j}))/(\overline{c}_{i,j} - \underline{c}_{i,j}), \tag{3.49a}$$

$$\tau_{i,j} = \underline{c}_{i,j}(g(\underline{c}_{i,j}) - g(\overline{c}_{i,j}))/(\overline{c}_{i,j} - \underline{c}_{i,j}) + g(\underline{c}_{i,j}), \tag{3.49b}$$

$\forall j \in [\rho + 1, \rho + \sigma]$. Moreover, let the factors be zero when the parameters set $c_i^{\sigma} = \{\emptyset\}$.

3.6 Contribution

3.7 Results

This Section shows and assesses experimental results for the benchmarks, and validates the presented approach.

We now describe the experimental results of the benchmarks previously introduced. A summary of these results is outlined in Table 3.

Component	ODROID	NVIDIA		
	XU3	TK1	TX2	Nano
`matrix-cpu`	5284 J	4067 J	2413 J	2736 J
`matrix-gpu`	-	81 J	45 J	39 J
`darknet-cpu`	(-)	(-)	2400 J	(-)
`darknet-gpu`	-	-	5255 J	(-)
`nvidia-matrix`	-	(-)	4054 J	(-)
`nvidia-quicks`	-	(-)	1995 J	(-)

Table 3. The overall energy consumption for each benchmark. Unsupported platforms are indicated by '-' and '(-)' indicates supported but not included in this paper.

Matrix Exponentiation

Execution of GPU matrix exponentiation, while varying size and exponent parameters, was previously shown in Figure 3.3. The figure shows the average power as well as overall energy consumption along with the battery depletion as a function of size and exponent parameters. Average power consumption is reported independently of

Fig. 3.15. Layer 2 model of `darknet-gpu` component running under four configurations, respectively 5.8, 10, 25 and 32 frames-per-second. The figure shows the per-minute energy consumption in terms of CPU, GPU, and overall. On the right side, energy consumption for any possible frame per second rate is shown along with SoC (the y2-axis, the y-axis is shared with the left side).

the running time of the component and thus does not reflect the total power consumption. For small problem sizes, the computation terminates before reaching the maximal power level. This effect is visible in Figure ?? (also previously shown), where power consumption is low at the beginning and then reaches the maximum, for which reason the average power consumption is low for small problem sizes. Battery depletion is reported in terms of the total amount of energy consumed by the computation for the duration of the execution. The effect of introducing "scheduling" in the form of sleep of various durations in between iterations of the matrix computations can be seen in Figure 3.7. Here, the duration of the sleep affects the total power consumption: the higher the sleep value in between the iterations, the greater the battery depletion.

CPU vs GPU comparison shows, expectedly, that `matrix-gpu` is very performant compared to `matrix-cpu`. While the `matrix-cpu` requires 2 413 J, `matrix-gpu` requires only 45 J for the same operation on the TX2 board. Therefore, running the benchmark on GPU results in 16% more SoC against the same trial on CPU. Such a high speed-up is observed due to the highly parallelizable nature of the matrix exponentiation and hence cannot be used as a general rule. From our experiments, we additionally observe the power-related effect of running components sequentially versus in parallel on different computational units. Although the CPU and GPU are different computational units, the energy consumed by running components independently (i.e., sequentially in some order) on CPU and GPU is 20% larger compared to running them in parallel, even when subtracting the base power consumed by the board. Thus energy can be conserved by running computations in parallel on the CPU and GPU, compared to scheduling them sequentially.

Darknet

We modified the `darknet` component to simulate different scheduling options and evaluated the outcome on a video stream: `darknet` now accepts an argument that indicates the amount of sleep between two invocations of the image recognition algorithm. In this way, we were able to simulate different frames-per-second options and assess the power evolution using the model.

Our experimental data depicted in Figure 3.15 shows that an increment in frames-per-second corresponds to an increased power consumption along with a higher battery depletion. The resulting model can be used to define an appropriate trade-off that

represents an acceptable rate of QoS, by i.e., correlating the FPS rate to the battery depletion and hence highlighting the dynamic behavior of a mobile scenario. Moreover, we observe that `darknet-cpu` consumes less energy per minute compared to the `darknet-gpu` component, but is considerably slower: while running `darknet-cpu` for one minute on TX2 board requires 2 400 J, `darknet-gpu` requires 5 255 J. When considering the energy cost per frame the computation is however considerably more efficient on GPU, where it requires just 1.3 J per frame, against the 175 J on GPU.

NVIDIA

The `nvidia-matrix` benchmark differs from `matrix-gpu`, even if both perform matrix multiplication on GPU, since `nvidia-matrix` includes a significant CPU computation after GPU matrix multiplication to check whether the two match. The `matrix-gpu` benchmark was similarly tested during development, but does not perform this test at runtime. We observe that the `nvidia-matrix` benchmark has a similar energy behavior to `matrix-gpu`, with the problem size affecting the overall energy and henceforth battery depletion. Average power consumption on GPU is however higher for `matrix-gpu` while it is approximately the same on CPU for both benchmarks (presumably none of the two benchmarks focus on energy efficiency on CPU). For the quicksort benchmark, as expected energy consumption increases with the problem size, as more operations are performed. Nevertheless quicksort differs from matrix multiplication: there is more noise due to a higher dependence on the random data that is being sorted.

3.7.1 Validation

We validate our approach by: a) demonstrating that `powprofiler` can be used on a number of heterogeneous platforms, b) comparing model generated with internal metrics to external physical measurements on the TX2, and c) comparing the model to a fine-grained one.

A cross-platform comparison shows energy-related behavior of running the same benchmark on different boards. We observed, for instance, that the most energy-efficient board for `matrix-cpu` benchmark is TX2, which consumes 2 413 J to perform the operation, followed by Nano with 2 736 J, TK1 with 4 067 J, and ODROID with 5 284 J.

A measurement analysis is used to compare internal against external power metrics on the TX2 board. We observe that both externally and internally measured power models are close one to each other, with an error of less than 3% measured over one minute. The externally measured power exceeds the internally measured power — this is natural as the externally measured power also includes the carrier board, which requires additional energy to operate. Moreover, the measurements performed using

`powprofiler` include the power consumed by `powprofiler` itself, while the multimeter setup excludes `powprofiler`'s energy from the evaluation. We therefore assume that the tool has a marginal effect on power consumption and that the model is showing realistic behavior.

In a last validation step, we compared our approach to fine-grained energy model of Nunez et al. (Nunez-Yanez and Lore, 2013) and Nikov et al. (Nikov et al., 2015) for the ODROID-XU3 embedded board. To relate the two approaches for energy-modeling, we used the `matrix-cpu` benchmark component as follows. First, we evaluated the matrix exponentiation benchmark by raising a 512 by 512 matrix to the 30th power (512^{30}) to train the fine-grained model. We obtained a value of expected energy per operation that we compared to the measured one and measured a relative error of 3.42%. Second, we applied an equivalent approach to our model. We sampled some configurations that were distant from the expected one, concretely we ran configurations 512^x with $x \in \{5, 15, 25, 35, \dots\}$, and we used the approximation method described in Section ?? to evaluate the energy of the configuration 512^{30}. This value was then compared to the measured one and we obtained a relative error of 2.25%. We can conclude that both models produce similar results on this benchmark, and have a similar relative error even if they adopt a different approach towards energy-modeling.

3.7.2 Assessment

We performed a number of experiments on different boards. Most of the data in this paper were obtained from NVIDIA TX2 due to the similarity with TK1 and Nano and the easy accessibility of the internal power measurement units. These data allowed us to validate our model, and to show that different scheduling options can correspond to different energy models. The model can describe energy consumption and instantaneous power, together with the battery depletion.

Coarse-grained whole-system energy modeling can take into account some behaviors that cannot otherwise easily be observed. As an example, consider the effect of simple scheduling previously shown in Figure 3.7. Here, we expected that introducing a higher sleep period between executions would result in an energy cost. We used the model to investigate this assumption, and we found that it is not always happening. For instance, a period of 2.5s between two consecutive schedules of 512^{12} matrix exponentiation iterated for 12 times is more energy-efficient than executing the whole 512^{144} operation. The model can relate these observations, provide information about the battery depletion, and predict the total time the system can operate on a given energy source. It can also highlight battery-specific behaviors since different scheduling options drain battery differently. In particular, Using the `powprofiler` tool on the NVIDIA TX2, we experimentally observe that:

- Running a set of components separately, and simply adding their energy consumption (while excluding base energy), leads to a different model versus running them in parallel (Section 3.7). This behavior was observed for matrix exponentiation components running on separate computational units (CPU, GPU).
- Scheduling of computations directly impacts the battery drain: processing a computation in its entirety with a steady power load drains the battery less than scheduling that same computation into smaller steps with resulting spikes in the power load (Section 3.7.2). This behavior was observed for a matrix exponentiation component running on CPU.

3.8 Summary

Chapter 4

State Estimation

A[A,]

4.1 A Brief History of State Estimation

4.2 A Necessary Background on Probability Theory

4.3 The Curve Fitting Problem

4.3.1 Least square fitting

4.3.2 Levenberg-Marquardt algorithm

4.4 Periodic Model Estimation

4.4.1 Period estimation

4.4.2 Minimization of the estiamte error

4.4.3 Period and horizon in continuous estimation

4.5 Filtering

4.5.1 Discrete time Kalman filter

4.5.2 Continuous time Kalman filter

4.5.3 Nonlinear filtering

4.6 Results

4.7 Summary

Chapter 5

Guidance

A[A]

5.1 Vector Fields for Guidance

5.2 Derivation of the Guidance Action

5.2.1 Motion simulations

5.2.2 Energy simulations

5.3 Alteration of the Path

5.4 Results

5.5 Summary

Chapter 6

Optimal Control Generation

This chapter provides essential theoretical background on optimal control theory necessary to derive an optimal configuration of the path and computations of the flying UAV. It solves the problem posed in Section 1.5 and illustrate an algorithm that generates the optimal configuration dynamically. The algorithm relies on a modern optimal control technique known as model predictive control (MPC), where the optimal control trajectory is evaluated on a receding horizon for each optimization step (Rawlings et al., 2017).

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

Many optimization problems originating in fields such as robotics, economics, and aeronautics can be formulated as optimal control problems (OCPs) (Von Stryk and Bulirsch, 1992). Optimization is often called mathematical programming (Nocedal and Wright, 2006) a term that means finding ways to solve the optimization problem. One can often find programming in this context in terms such as linear program (LP), quadratic program (QP), and nonlinear program (NLP). NLPs is the class of optimization problems that we use to derive the optimal configuration. OCPs can be seen as optimization problems with the added difficulty of continuous dynamics. The latter is to be integrated over the optimization horizon using numerical simulation. In the algorithm, we formulate the dynamic planning problem as an OCP that we solve with a numerical method: we transform the OCP in an NLP using numerical simulation and solve the NLP using numerical optimization, as proposed in (Rawlings et al., 2017). The process is illustrated in Fig. 6.1.

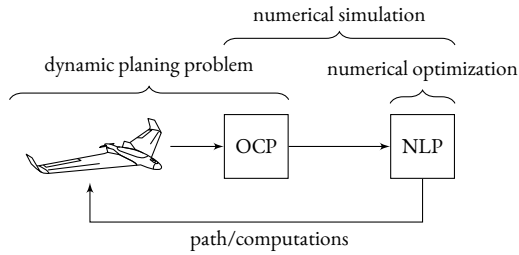

Fig. 6.1. Summary of the optimal control approach. The problem is formulated as an OCP, into finite-dimensional discrete NLP using numerical simulation. NLP is solved using numerical optimization and the optimal configuration for a given time horizon is returned to the UAV. The following horizon is evaluated again in a technique known as MPC.

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

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

The chapter builds on the rest of the work as follows. In the OCP formulation, we use the estimated state from Chapter 4 of a perfect model in Chapter 3 to solve the planning problem in Section 1.5 and guide the UAV with the obtained optimal configuration from this chapter with the technique in Chapter 5.

6.1 A Brief History of Optimal Control

Optimal control originates from the calculus of variations (Sethi, 2019), based on the work of Euler and Lagrange. Calculus of variations solves the problem of determining the arguments of an integral, such that its value is maximum (or minimum). The equivalent problem in calculus is to determine the argument of a function where the function is maximum (or minimum). The work by Euler and Lagrange was later extended by Legendre, Hamilton, and Weierstrass (Paulen and Fikar, 2016). It has gained a renewed interest in the mid-twentieth century, as modern calculators offered prac-

tical ways of solving some OCPs for nonlinear and time-varying systems that were earlier impracticable (Bryson and Ho, 1975).

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

There are numerous methods to analytically and numerically solve these continuous time OCPs, although analytical solution is often impracticable except for very limited state dimensions (Rawlings et al., 2017). In the early day of optimal control, some analytical solutions were proposed with dynamic programming (Bellman, 1957), and with maximum (or minimum) principle (Pontryagin et al., 1962).

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

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

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

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

Modern OCPs are often solved on a finite and receding horizon using an approximation of the true dynamics using MPC techniques. It is a more systematic technique which allow to control a model by re-optimizing the OCP repeatedly (Paulen and Fikar, 2016; Poe and Mokhatab, 2017). It takes into account external interferences by re-estimating the model's state (with techniques that we introduced in Chapter 4).

MPC is extensively treated in modern optimal control literature (Camacho and Alba, 2007; Kwon and Han, 2005; Rawlings et al., 2017; Rossiter, 2004; L. Wang, 2009).

6.2 Optimization Problems with Dynamics

6.2.1 Continuous systems: unconstrained case

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

$$\dot{\mathbf{q}}(t) = f(\mathbf{q}(t), \mathbf{u}(t), t), \quad \mathbf{q}(t_0) = \mathbf{q}_0 \text{ given}, \quad \forall t \in [t_0, T], \tag{6.1}$$

where $t_0 \in \mathbb{R}_{\geq 0}$ is a given initial time instant, and $\mathbf{q}_0 \in \mathbb{R}^m$ a given initial state guess. The latest can be derived empirically from a previous execution or using some initial sensor data. $f : \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^m$ maps the current state, control and time to the next state. The notations for $\dot{\mathbf{q}}(t) := d\mathbf{q}(t)/dt$, $\mathbf{q}$, and $\mathbf{u}$ are the same from Chapter 3. The function f is assumed to be continuously differentiable. Physically, Equation (6.1) specifies the instantaneous change in state variable of a perfect model with no disturbances.

If the control trajectory $\mathbf{u}(t_0), \mathbf{u}(t_1), \ldots, \mathbf{u}(T - \Delta t)$ is known for a given time horizon $t_0 \leq t \leq T$, the model in Equation (6.1) can be derived to obtain the state trajectory $\mathbf{q}(t_0), \mathbf{q}(t_1), \ldots, \mathbf{q}(T)$, where Δt is the instantaneous change in time. The last state at the final time instant T is derived from the last control at the time instant $T - \Delta t$. The state trajectory has indeed one item more than the control trajectory.

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

$$L = l_f(\mathbf{q}(T), T) + \int_{t_0}^{T} l(\mathbf{q}(t), \mathbf{u}(t), t)\, dt, \tag{6.2}$$

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

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

The control variable is usually constrained

$$\mathbf{u}(t) \in \mathcal{U}(t), \quad \forall t \in [t_0, T], \tag{6.3}$$

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

The Equations (6.1–6.3) forms unconstrained OCPs. These problems are formalized

$$\begin{aligned} &\max_{\mathbf{u}(\mathbf{t})\in\mathcal{U}(\mathbf{t})} l_f(\mathbf{q}(T), T) + \int_{t_0}^{T} l(\mathbf{q}(t), \mathbf{u}(t), t)\, dt, \\ &\text{s.t. } \dot{\mathbf{q}}(t) = f(\mathbf{q}(t), \mathbf{u}(t), t) \\ &\mathbf{q}(t_0) = \mathbf{q}_0 \text{ given.} \end{aligned} \tag{6.4}$$

The evolution of the model is used to derive an optimal control trajectory $\mathbf{u}(t)$ from an initial guess of the state $\mathbf{q}_0$ and the horizon. This initial simplistic controller does not represent a realistic scenario. The controller implies that the horizon is known. However, it is often the case that only the initial time step of the horizon $[t_0, T]$ is known. In the model from Chapter 3 it is indeed unknown apriori when the UAV plan terminates. Moreover the controller does not include any constraint on the state $\mathbf{q}$, although UAVs are often bounded by strict battery requirements. Lastly, the optimal control generated with such controller is static given the initial state and the horizon. It is unrealistic to assume that the state of the UAV travelling the optimal control $\mathbf{u}$ does not change for instants $t_0 + \Delta t, t_0 + 2\Delta t, \ldots, T$.

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

6.2.2 Continuous systems: constrained case

6.2.3 Perturbed systems

6.2.4 Multistage systems

6.3 Numerical Simulation and Differentiation

6.3.1 Euler method

6.3.2 Runge-Kutta methods

6.3.3 Algorithmic differentiation

6.4 Direct Optimal Control Methods

6.4.1 Direct single shooting

6.4.2 Direct multiple shooting

6.4.3 Direct collocation

6.5 Numerical Optimization

6.5.1 Convexity

6.5.2 Optimality conditions

6.5.3 First order optimality conditions

6.5.4 Second order optimality conditions

6.5.5 Sequential quadratic programming

6.5.6 Nonlinear interior point methods

6.6 Model Predictive Control

6.6.1 Output model predictive control

6.6.2 Optimal control generation with model predictive control

6.7 Results

6.8 Summary

Appendix A

Implementation

A.1 Energy Models

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

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

    m = 2*dat.r+1;

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

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

    C = [1];

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

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

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

```
26 end
```

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

The function builds the matrices from Equation (3.5). In particular matrix A from Equation (3.7), B from Equation (3.42), and C from Equation (3.9). The term $1/T$ is missing in `C` as we motivated in the proof of Lemma 3.5.1 in Equation (3.36). The nominal control from Equation (3.40) is `u`, and the energy estimate of a given control sequence at time instant t and the previous time instant $t-1$ from Equation (3.41) is `est_u`.

The function requires the structure `dat` illustrated in Listing A.2. The structure contains some information about the model. In particular, `r` is the order r of the model from Subsection 3.5.2, `omega` is the angular frequency ω, `rho` the number of path parameters to alter the path ρ, and `sigma` the number of computation parameters to alter the computations σ. The path parameters are defined in Equation (1.3), and the computation parameters are defined in Equation (1.4). Furthermore, `delta_T` is the sampling step Δt, and `nu` and `tau` are scaling factors from Equation (3.43).

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

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

The function `build_model` and the structure `dat` can be used along an initial guess of parameters and a time horizon to model the future energy consumption using the Euler or Runge-Kutta methods for numerical integration. The Euler method is described in Subsection 6.3.1 and implemented in Listing A.3, the Runge-Kutta method in Subsection 6.3.2 and Listing A.5. For the Runge-Kutta method we use the fourth order fixed size standard algorithm.

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

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

6     q0 = init.q0;

8     q = q0;
9     y = C*q0;

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

```
14      i = 2;

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

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

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

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

The function `evolve_model_euler` in Listing A.3 evolve the model in Equation (3.5) from an initial time t_0 to the final time t_f. This information is passed to function using the structure `init` from Listing A.4.

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

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

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

```
1 % TODO
```

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

Before calling structure `dat`, it is necessary to define the scaling factors $\nu_1, \nu_2, \ldots, \nu_\rho$ and $\tau_1, \tau_2, \ldots, \tau_\rho$ for the path parameters, and $\nu_{\rho+1}, \nu_{\rho+2}, \ldots, \nu_{\rho+\sigma}$ and $\tau_{\rho+1}, \tau_{\rho+2}, \ldots, \tau_{\rho+\sigma}$ for the computation parameters. If we use Listing A.4 there is one path and one computation parameters.

```
1 nu1 = -(max_t-min_t)/min_c1;
```

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

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

7 nu = [nu1;nu2];
8 tau = [tau1;tau2];
```

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

The scaling factors thus can be defined according to Equation (3.48) for the path parameter on Line 1 and according to Equation (3.49) for the computation parameter on Line 4.

Let us assume there is one path parameter c_1 which doesn't change for the stages $i = \{1, 2, \ldots, l\}$. `max_t` and `min_t` are the maximum and minimum times that correspond to the time needed to hypothetically execute the path with parameter $c_1 = \overline{c}_1$ and with parameter value $c_1 = \underline{c}_1$. A guess for these values can be obtained empirically; we obtained them by running the simulator. They correspond to $\overline{t}$ and $\underline{t}$ in Equation (3.48).

Let us further assume there is one computation parameter c_2 which doesn't change for the stages $i = \{1, 2, \ldots, l\}$. `max_c2` and `min_c2` are the minimum and maximum configuration of the computation parameter c_2 defined in Definition 1.5.3. `g_max_c2` and `g_min_c2` are values retrieved from `powprofiler` and they correspond to $g(\overline{c}_2)$ and $g(\underline{c}_2)$ in Equation (3.49). Function g is formally defined in Definition 3.5.1.

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

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

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

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

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

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

9 figure;
10 m = 2*dat.r+1;
11 t = tiledlayout(m,1);
12 title_str = {'alpha','beta'};
```

```
nexttile,plot(time,q(1,1:end))
title(strcat(title_str(1),0));

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

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

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

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

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

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

A.2 State Estimation

A.2.1 Estimation of the period

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

for traj = transpose(path) % per each trajectory

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

    x = p(1);
    y = p(1);
    di = double(subs(traj));
    x = p(1)-sp(1);
    y = p(2)-sp(2);

    if ismember(double(subs(traj)),d)
        break;
    else
```

```
20          d = [d di];
21      end

23      n = n+1;
24  end

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

Listing A.10. Estimation of the period.

A.2.2 Estimation of the state

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

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

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

9  q0 = q1;
10 P0 = P1;

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

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

A.3 Guidance

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

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

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

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

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

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

Listing A.12. Guidance vector field.

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

3  th_delta = kp*(hd-h)-kvv*vv;
4  wbx = dot(w,[cos(theta),sin(theta)]);
5  vs = cth*(th_nominal+th_delta)+wbx;
6  L = cl*vs*vs;
7  av = 1/m*(L-W);

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

12 % Horizontal unicycle model
13 sh = cth*(th_nominal+th_delta);
14 pdot = sh*[cos(theta);sin(theta)]+w;

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

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

A.4 Optimal Control Generation

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

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

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

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

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

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

25      jjj = 1;
26      dQ = q0; % initial state
27      log_Q(:,1) = dQ;

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

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

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

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

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

47              jjj = jjj+1;
48          end
49      end

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

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

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

```
62 end
```

Listing A.14. Model predictive control.

References

1. 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 pp. 1, 18).
2. Aljanobi, A., Al-Hamed, S., and Al-Suhaibani, S. (2010). “A setup of mobile robotic unit for fruit harvesting”. In: *19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010)*. IEEE, pp. 105–108 (cit. on pp. 2, 10).
3. 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 pp. 2, 10).
4. Anderson, J. D. (2005). *Introduction to flight*. McGraw-Hill Higher Education (cit. on p. 4).
5. Bailey, P. E., Lowenthal, D. K., Ravi, V., Rountree, B., Schulz, M., and De Supinski, B. R. (2014). “Adaptive configuration selection for power-constrained heterogeneous systems”. In: *2014 43rd International Conference on Parallel Processing*. IEEE, pp. 371–380 (cit. on p. 16).
6. Barik, R., Farooqui, N., Lewis, B. T., Hu, C., and Shpeisman, T. (2016). “A black-box approach to energy-aware scheduling on integrated CPU-GPU systems”. In: *Proceedings of the 2016 International Symposium on Code Generation and Optimization*. ACM, pp. 70–81 (cit. on p. 16).
7. Bellman, R. E. (1957). *Dynamic Programming*. Princeton: Princeton University Press (cit. on p. 53).
8. Böhme, T. J. and Frank, B. (2017). “Indirect Methods for Optimal Control”. In: *Hybrid Systems, Optimal Control and Hybrid Vehicles: Theory, Methods and Applications*. Cham: Springer International Publishing, pp. 215–231 (cit. on p. 53).
9. Bryson, A. E. and Ho, Y.-C. (1975). *Applied optimal control: optimization, estimation and control*. Hemisphere Publishing Corporation (cit. on pp. 53, 54).
10. 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).

11. Burri, M., Gasser, L., Käch, M., Krebs, M., Laube, S., Ledergerber, A., Meier, D., Michaud, R., Mosimann, L., Müri, L., et al. (2013). “Design and control of a spherical omnidirectional blimp”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1873–1879 (cit. on p. 5).

12. 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 p. 17).

13. Camacho, E. F. and Alba, C. B. (2007). *Model predictive control*. London: Springer-Verlag (cit. on p. 54).

14. 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. 16).

15. Colombatti, G., Aboudan, A., La Gloria, N., Debei, S., and Flamini, E. (2011). “Lighter-Than-Air UAV with slam capabilities for mapping applications and atmpsphere analysys.” In: *Memorie della Societa Astronomica Italiana Supplementi* 16, p. 42 (cit. on p. 5).

16. Corke, P. (2017). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. 2nd. Springer Publishing Company, Incorporated (cit. on pp. 1, 5).

17. Cui, J. Q., Phang, S. K., Ang, K. Z., Wang, F., Dong, X., Ke, Y., Lai, S., Li, K., Li, X., Lin, F., et al. (2015). “Drones for cooperative search and rescue in post-disaster situation”. In: *2015 IEEE 7th international conference on cybernetics and intelligent systems (CIS) and IEEE conference on robotics, automation and mechatronics (RAM)*. IEEE, pp. 167–174 (cit. on p. 4).

18. Daponte, P., De Vito, L., Glielmo, L., Iannelli, L., Liuzza, D., Picariello, F., and Silano, G. (2019). “A review on the use of drones for precision agriculture”. In: *IOP Conference Series: Earth and Environmental Science*. Vol. 275. 1. IOP Publishing, p. 012022 (cit. on pp. 2, 4, 8, 10).

19. De Marina, H. G., Kapitanyuk, Y. A., Bronz, M., Hattenberger, G., and Cao, M. (2017). “Guidance algorithm for smooth trajectory tracking of a fixed wing UAV flying in wind flows”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 5740–5745 (cit. on pp. 9, 40).

20. 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 pp. 2, 10).

21. Dressler, F. and Fuchs, G. (2005). “Energy-aware operation and task allocation of autonomous robots”. In: *Proceedings of the Fifth International Workshop on Robot Motion and Control, 2005. RoMoCo’05.* IEEE, pp. 163–168 (cit. on p. 16).

22. 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 pp. 2, 10).

23. Fabiani, P., Fuertes, V., Piquereau, A., Mampey, R., and Teichteil-Königsbuch, F. (2007). “Autonomous flight and navigation of VTOL UAVs: from autonomy demonstrations to out-of-sight flights”. In: *Aerospace Science and Technology* 11.2-3, pp. 183–193 (cit. on p. 20).

24. 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).

25. 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. 4).
26. 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. 5).
27. 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 p. 16).
28. 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. 9).
29. 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).
30. Goraczko, M., Liu, J., Lymberopoulos, D., Matic, S., Priyantha, B., and Zhao, F. (2008). “Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems”. In: *2008 45th ACM/IEEE Design Automation Conference*. IEEE, pp. 191–196 (cit. on p. 16).
31. Hajjaj, S. S. H. and Sahari, K. S. M. (2014). “Review of research in the area of agriculture mobile robots”. In: *The 8th International Conference on Robotic, Vision, Signal Processing & Power Applications*. Springer, pp. 107–117 (cit. on pp. 2, 10).
32. Hasan, A., Skriver, M., and Johansen, T. A. (2018). “Exogenous kalman filter for state-of-charge estimation in lithium-ion batteries”. In: *2018 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, pp. 1403–1408 (cit. on pp. 16, 32).
33. Hobby, H. (2020). *Opterra 2m Wing BNF Basic*. URL: https://www.horizonhobby.com/opterra-2m-wing-bnf-basic-p-efl11150 (visited on 02/02/2020) (cit. on p. 2).
34. 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. 16).
35. 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. 16).
36. Jaramillo-Avila, U., Aitken, J. M., and Anderson, S. R. (2019). “Visual saliency with foveated images for fast object detection and recognition in mobile robots using low-power embedded GPUs”. In: *2019 19th International Conference on Advanced Robotics (ICAR)*. IEEE, pp. 773–778 (cit. on pp. 1, 18).
37. Kanellakis, C. and Nikolakopoulos, G. (2017). “Survey on computer vision for UAVs: Current developments and trends”. In: *Journal of Intelligent & Robotic Systems* 87.1, pp. 141–168 (cit. on p. 17).
38. 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. 9).
39. 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).

40. 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).

41. 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).

42. Kim, C. H. and Kim, B. K. (2005). “Energy-saving 3-step velocity control algorithm for battery-powered wheeled mobile robots”. In: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE, pp. 2375–2380 (cit. on p. 18).

43. Kim, H. and Kim, B.-K. (2008). “Minimum-energy translational trajectory planning for battery-powered three-wheeled omni-directional mobile robots”. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. IEEE, pp. 1730–1735 (cit. on p. 18).

44. Kirk, D. E. (2004). *Optimal control theory: an introduction*. Courier Corporation (cit. on pp. 51, 52, 54).

45. Kreciglowa, N., Karydis, K., and Kumar, V. (2017). “Energy efficiency of trajectory generation methods for stop-and-go aerial robot navigation”. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, pp. 656–662 (cit. on pp. 17, 18).

46. Kuo, B. (1967). *Automatic Control Systems*. Electrical engineering series. Prentice-Hall (cit. on pp. 34, 35).

47. Kwon, W. H. and Han, S. H. (2005). *Receding horizon control: model predictive control for state models*. London: Springer-Verlag (cit. on p. 54).

48. LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press (cit. on p. 53).

49. 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 pp. 2, 10).

50. Lindemann, S. R. and LaValle, S. M. (2005). “Smoothly blending vector fields for global robot navigation”. In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, pp. 3553–3559 (cit. on p. 9).

51. Luo, J. and Jha, N. K. (2001). “Battery-aware static scheduling for distributed real-time embedded systems”. In: *Proceedings of the 38th annual Design Automation Conference*. ACM, pp. 444–449 (cit. on p. 16).

52. Marowka, A. (2017). “Energy-aware modeling of scaled heterogeneous systems”. In: *International Journal of Parallel Programming* 45.5, pp. 1026–1045 (cit. on p. 15).

53. Mei, Y., Lu, Y.-H., Hu, Y. C., and Lee, C. G. (2004). “Energy-efficient motion planning for mobile robots”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*. Vol. 5. IEEE, pp. 4344–4349 (cit. on pp. 1, 16, 18, 19).

54. — (2005). “A case study of mobile robot’s energy consumption and conservation techniques”. In: *ICAR’05. Proceedings., 12th International Conference on Advanced Robotics, 2005.* IEEE, pp. 492–497 (cit. on pp. 1, 16, 18, 19).

55. 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, 18).

56. 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).

57. 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. 35).

58. Morbidi, F., Cano, R., and Lara, D. (2016). “Minimum-energy path generation for a quadrotor UAV”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1492–1498 (cit. on p. 18).

59. Mudge, T. (2001). “Power: A first-class architectural design constraint”. In: *Computer* 34.4, pp. 52–58 (cit. on p. 20).

60. Nikov, K., Nunez-Yanez, J. L., and Horsnell, M. (2015). “Evaluation of hybrid run-time power models for the ARM big. LITTLE architecture”. In: *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing*. IEEE, pp. 205–210 (cit. on pp. 17, 45).

61. Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media (cit. on p. 51).

62. Noor, N. M., Abdullah, A., and Hashim, M. (2018). “Remote sensing UAV/drones and its applications for urban areas: a review”. In: *IOP conference series: Earth and environmental science*. Vol. 169. 1. IOP Publishing, p. 012003 (cit. on p. 4).

63. 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. 17, 45).

64. Ogata, K. (2002). *Modern Control Engineering*. Prentice Hall (cit. on p. 35).

65. Panagou, D. (2014). “Motion planning and collision avoidance using navigation vector fields”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2513–2518 (cit. on p. 9).

66. Paparazzi (2016). *UAV open-source project*. URL: http://wiki.paparazziuav.org/ (visited on 09/01/2016) (cit. on p. 8).

67. 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).

68. Paulen, R. and Fikar, M. (2016). “Solution of Optimal Control Problems”. In: *Optimal Operation of Batch Membrane Processes*. Cham: Springer International Publishing, pp. 37–56 (cit. on pp. 52, 53).

69. 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).

70. Poe, W. A. and Mokhatab, S. (2017). “Process Control”. In: *Modeling, Control, and Optimization of Natural Gas Processing Plants*. Ed. by W. A. Poe and S. Mokhatab. Boston: Gulf Professional Publishing, pp. 97–172 (cit. on p. 53).

71. Pontryagin, L. S., Mishchenko, E., Boltyanskii, V., and Gamkrelidze, R. (1962). *The mathematical theory of optimal processes*. Wiley (cit. on p. 53).

72. *Public Deliverables of the TeamPlay Horizon2020 Project* (2019). `https://teamplay-h2020.eu/index.php?page=deliverables`. Accessed: 2019-08-25 (cit. on p. 17).

73. Puri, V., Nayyar, A., and Raja, L. (2017). "Agriculture drones: A modern breakthrough in precision agriculture". In: *Journal of Statistics and Management Systems* 20.4, pp. 507–518 (cit. on pp. 2, 4, 10).

74. PX4 (2016). *PX4 open-source autopilot*. URL: `https://px4.io/` (visited on 09/01/2016) (cit. on p. 8).

75. Qingchun, F., Wengang, Z., Quan, Q., Kai, J., and Rui, G. (2012). "Study on strawberry robotic harvesting system". In: *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*. Vol. 1. IEEE, pp. 320–324 (cit. on pp. 2, 10).

76. 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 p. 41).

77. 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 p. 15).

78. Rao, S. S. (2019). *Engineering optimization: theory and practice*. John Wiley & Sons (cit. on p. 54).

79. Rawlings, J. B., Mayne, D. Q., and Diehl, M. (2017). *Model predictive control: theory, computation, and design*. Vol. 2. Madison, Wisconsin: Nob Hill Publishing (cit. on pp. 10, 51, 53, 54).

80. Redmon, J. (2013–2016). *Darknet: Open Source Neural Networks in C*. `http://pjreddie.com/darknet/` (cit. on p. 23).

81. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788 (cit. on p. 23).

82. Redmon, J. and Farhadi, A. (2017). "YOLO9000: Better, Faster, Stronger". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, USA, pp. 6517–6525 (cit. on p. 23).

83. 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 pp. 1, 18).

84. Rossiter, J. A. (2004). *Model-based predictive control: a practical approach*. Boca Raton, Florida: CRC press (cit. on p. 54).

85. 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. 17, 18).

86. Sadrpour, A., Jin, J., and Ulsoy, A. G. (2013b). "Experimental validation of mission energy prediction model for unmanned ground vehicles". In: *2013 American Control Conference*. IEEE, pp. 5960–5965 (cit. on pp. 17, 18).

87. Satria, M. T., Gurumani, S., Zheng, W., Tee, K. P., Koh, A., Yu, P., Rupnow, K., and Chen, D. (2016). “Real-time system-level implementation of a telepresence robot using an embedded GPU platform”. In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 1445–1448 (cit. on pp. 1, 18).

88. 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).

89. Seewald, A., Ebeid, E., and Schultz, U. P. (2019). “Dynamic Energy Modelling for SoC Boards: Initial Experiments”. In: *HLPGPU 2019: High-Level Programming for Heterogeneous and Hierarchical Parallel Systems*, p. 4 (cit. on pp. 17, 29).

90. Seewald, A., Garcia de Marina, H., Midtiby, H. S., and Schultz, U. P. (2020). “Mechanical and Computational Energy Estimation of a Fixed-Wing Drone”. In: *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, pp. 135–142 (cit. on pp. 1, 9, 41).

91. Seewald, A., Schultz, U. P., Ebeid, E., and Midtiby, H. S. (2021). “Coarse-Grained Computation-Oriented Energy Modeling for Heterogeneous Parallel Embedded Systems”. In: *International Journal of Parallel Programming* 49.2, pp. 136–157 (cit. on pp. 8, 41).

92. Seewald, A., Schultz, U. P., Roeder, J., Rouxel, B., and Grelck, C. (2019). “Component-based computation-energy modeling for embedded systems”. In: *Proceedings Companion of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. ACM, pp. 5–6 (cit. on p. 41).

93. 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).

94. Sethi, S. P. (2019). *Optimal Control Theory: Applications to Management Science and Economics*. Cham: Springer International Publishing (cit. on pp. 51–54).

95. Siciliano, B. and Khatib, O. (2016). *Springer handbook of robotics*. Springer. Chap. 44, pp. 1012–1014 (cit. on pp. 3, 4, 6, 21).

96. Simon, D. (2006). *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons (cit. on p. 54).

97. Stengel, R. F. (1994). *Optimal control and estimation*. Courier Corporation (cit. on p. 54).

98. Sudhakar, S., Karaman, S., and Sze, V. (2020). “Balancing Actuation and Computing Energy in Motion Planning”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4259–4265 (cit. on p. 1).

99. 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).

100. Uragun, B. (2011). “Energy efficiency for unmanned aerial vehicles”. In: *2011 10th International Conference on Machine Learning and Applications and Workshops*. Vol. 2. IEEE, pp. 316–320 (cit. on p. 17).

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

102. Von Stryk, O. and Bulirsch, R. (1992). “Direct and indirect methods for trajectory optimization”. In: *Annals of operations research* 37.1, pp. 357–373 (cit. on p. 51).

103. Wahab, M., Rios-Gutierrez, F., and El Shahat, A. (2015). *Energy modeling of differential drive robots*. IEEE (cit. on p. 18).

104. Wang, L. (2009). *Model predictive control system design and implementation using Matlab*. London: Springer Science & Business Media (cit. on p. 54).

105. Woo, D. H. and Lee, H.-H. S. (2008). "Extending Amdahl's law for energy-efficient computing in the many-core era". In: *Computer* 41.12, pp. 24–31 (cit. on p. 19).

106. Wu, G., Greathouse, J. L., Lyashevsky, A., Jayasena, N., and Chiou, D. (2015). "GPGPU performance and power estimation using machine learning". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, pp. 564–576 (cit. on p. 16).

107. Zamanakos, G., Seewald, A., Midtiby, H. S., and Schultz, U. P. (2020). "Energy-Aware Design of Vision-Based Autonomous Tracking and Landing of a UAV". In: *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, pp. 294–297 (cit. on pp. 2, 6, 41).

108. Zhou, D. and Schwager, M. (2014). "Vector field following for quadrotors using differential flatness". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6567–6572 (cit. on p. 9).

Index