ENSTA | INSTITUT POLYTECHNIQUE DE PARIS

Event-driven Cycle Navigation for Mobile Robots

Quentin Brateau

Submission Date July 15, 2025
Supervisors Luc Jaulin & Fabrice Le Bars
Funding AID 22850 - Followed by Jean-Daniel Masson
Version v1.0

Abstract

This thesis proposes a new paradigm for autonomous underwater navigation in environments where traditional localization systems—such as GNSS, inertial navigation, or acoustic positioning—are unavailable or unreliable. Drawing inspiration from biological systems that exhibit cycle navigation patterns, this work introduces the concept of stable cycles as a core principle for robotic navigation. Rather than relying on absolute positioning, the robot starts by navigating along periodic trajectories defined by a timed automaton. These cyclic trajectories are controlled using a set of low-dimensional sensory inputs, which are enough to reach a desired state and ta stay stable around it. Theoretical results establish local and global stability conditions for these cycles using tools from set methods, positive invariant sets, and capture basins. A transition mechanism between cycles enables long-range navigation across complex, unstructured environments without the need for global localization. This approach is validated through numerical simulations and field experiments using the BlueBoat platform, demonstrating robust, low-power navigation capabilities. Overall, the work opens new pathways for energy-efficient, sensor-minimal navigation strategies in autonomous marine systems, with potential applications ranging from environmental monitoring to military surveillance.

Abstract

Cette thèse propose un nouveau paradigme pour la navigation autonome sous-marine dans des environnements où les systèmes classiques de localisation — tels que le GNSS, la navigation inertielle ou le positionnement acoustique — sont indisponibles ou peu fiables. En s'inspirant des systèmes biologiques qui présentent des schémas de navigation cycliques, ce travail introduit le concept de cycles stables comme principe fondamental pour la navigation robotique. Plutôt que de s'appuyer sur une localisation absolue, le robot commence par suivre des trajectoires périodiques définies par un automate temporisé. Ces trajectoires cycliques sont contrôlées à l'aide d'un ensemble réduit de capteurs, suffisant pour atteindre un état souhaité et rester stable autour de celui-ci. Des résultats théoriques établissent des conditions de stabilité locale et globale de ces cycles en s'appuyant sur des outils issus des méthodes ensemblistes, des ensembles invariants positifs et des bassins d'attraction. Un mécanisme de transition entre cycles permet une navigation à longue portée dans des environnements complexes et non structurés, sans besoin de localisation globale. Cette approche est validée par des simulations numériques et des expériences en mer avec la plateforme BlueBoat, démontrant des capacités de navigation robustes et à faible consommation d'énergie. Dans l'ensemble, ce travail ouvre de nouvelles perspectives pour des stratégies de navigation sobres en capteurs et en énergie, applicables aux systèmes marins autonomes, avec des applications potentielles allant de la surveillance environnementale à la reconnaissance militaire.

Contents

Notations

Intervals

$[x]$	Interval scalar
$[\mathbf{x}]$	Interval vector
$\mathbb{IR}$	Set of interval scalar of $\mathbb{R}$
$\mathbb{IR}^n$	Set of interval vectors of $\mathbb{R}^n$
x^+	Lower bound of $[x]$
x^-	Upper bound of $[x]$
$C([\mathbf{x}])$	Contractor
$\mathcal{S}([\mathbf{x}])$	Separator

Robotics

$\mathbf{x}$	State vector, $\mathbf{x} \in \mathbb{R}^n$
$\mathbf{u}$	Input vector, $\mathbf{u} \in \mathbb{R}^m$
$\mathbf{y}$	Observation vector, $\mathbf{y} \in \mathbb{R}^p$
$\mathbf{f}$	Evolution function, $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$
$\mathbf{g}$	Observation function, $\mathbf{g} : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^p$

Sets

$\emptyset$	Empty set
$\mathbb{S}$	Set
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}$	Set of integers
$\mathbb{R}$	Set of real numbers
$\partial\mathbb{S}$	Boundary of $\mathbb{S}$
$\mathbb{X}^-$	Inner approximation of $\mathbb{S}$
$\mathbb{X}^+$	Outer approximation of $\mathbb{S}$

List of Figures

1 Introduction

1.1 The Challenge of Marine Robot Navigation

The exploration and monitoring of marine environments represent one of the most significant frontiers in robotics research. Oceans cover more than 70% of Earth's surface, yet remain largely unexplored, with estimates suggesting that less than 20% of the ocean floor has been mapped at high resolution [63]. The development of autonomous marine robots has emerged as a promising solution to this exploration challenge, offering capabilities for extended missions in harsh environments where human presence is limited or impossible. However, despite significant technological advances, current marine robotics face persistent challenges that limit their effectiveness, particularly in the domain of navigation.

Modern approaches to marine robot navigation have relied heavily on Global Navigation Satellite System (GNSS) technology, and inertial navigation systems (INS). While GNSS provides excellent positional accuracy in surface operations, its signals cannot penetrate water beyond shallow depths, rendering it ineffective for submerged operations [53, 47]. Alternative solutions such as acoustic positioning systems require infrastructure deployment and offer limited range [62]. Meanwhile, INS systems suffer from drift over time, necessitating periodic position corrections [72]. To contain the drift due to the integration of linear accelerations and angular velocities given by the INS, a DVL is typically used to measure the velocity relative to the ground. The most advanced current approaches combine multiple sensing modalities with sophisticated algorithms, including Simultaneous Localization and Mapping (SLAM) [105, 79, 35], particle filters [108], and increasingly, deep neural networks for environmental feature recognition and localization [12]. However, these techniques mainly rely on the availability of landmarks [96]. A landmark is a punctual feature of known position that can be detected using a sensor, such as a camera or a sonar, and that can be used to correct the position estimation of the robot. In this work, we will refer to these approaches as *conventional approaches*. In contrast to these methods, we will attempt to find navigation strategies in areas without landmarks, where conventional methods would be challenging to use.

These conventional approaches share common limitations that restrict their practical deployment, particularly for long-duration missions or in resource-constrained scenarios. First, they typically demand substantial computational resources, limiting operational duration due to power constraints. Second, they often require expensive, high-precision sensors that increase system complexity and cost. Third, many rely on detailed prior environmental knowledge or infrastructure that may be unavailable in remote or unexplored regions. Finally, these systems frequently lack the robustness and adaptability demonstrated by biological organisms navigating in similar environments.

The limitations of existing approaches create a compelling case for alternative navigation paradigms that emphasize efficiency, robustness, and simplicity—characteristics abundantly demonstrated in biological navigation systems that have evolved over millions of years.

1.2 Biological Navigation Using Cyclical Patterns

Nature has developed remarkably efficient solutions to the challenge of navigation across vast distances without the benefit of precise positioning systems or extensive computational resources. Numerous species demonstrate the ability to navigate reliably using minimal sensory inputs and relatively simple behavioral patterns, often structured around cyclical movements and responses. These biological systems provide valuable inspiration for developing more efficient robotic navigation approaches.

1.2.1 Sea Turtle Navigation

Sea turtles represent one of the most impressive examples of long-distance marine navigation. Loggerhead sea turtles (*Caretta caretta*) shown in Figure 1.1, undertake transoceanic migrations spanning thousands of kilometers between feeding and nesting grounds, often returning with remarkable precision to their natal beaches after years at sea [58]. Research suggests that these turtles employ a multi-modal navigation strategy that includes geomagnetic field detection to determine latitude and longitude positions [57].

Figure 1.1 Loggerhead sea turtle (*Caretta caretta*)

Particularly relevant to this research is the turtles use of cyclical swimming patterns during migration. In [31], Hays et al. observed that migrating turtles often engage in characteristic looping behaviors when encountering specific oceanographic features, such as frontal zones or current boundaries. These repeating movement patterns appear to serve both as a search strategy and as a means of maintaining position relative to dynamic environmental features. The cyclical behaviors seem to function as a form of ”embodied navigation,” where the physical interaction between the animal’s movement pattern and the environment itself facilitates orientation and progress toward a goal.

1.2.2 Migratory Bird Navigation

Migratory birds demonstrate perhaps the most studied examples of long-distance navigation using cyclical patterns. Species such as the Arctic tern (*Sterna paradisaea*) complete annual migrations of over 70,000 kilometers, navigating between the Arctic and Antarctic with remarkable precision [23]. These birds integrate multiple navigational cues, including solar and stellar compasses, geomagnetic sensors, and visual landmarks.

Of particular interest is the discovery that many migratory birds use a cyclical oscillation pattern when maintaining course over long distances. Radar tracking studies by Cochran et al. [16] revealed that thrushes engaged in nocturnal migration maintain their heading through a series of corrective maneuvers, creating a regular oscillation around their intended course rather than flying in a perfectly straight line. This approach appears

to provide robustness against navigational errors and environmental perturbations. Furthermore, birds often use regular spiraling flight patterns when seeking thermal updrafts or when descending to investigate potential stopover sites, demonstrating how cyclical movement patterns can efficiently extract environmental information with minimal sensory input [104].

1.2.3 Fish Schooling and Migration

Many fish species employ cyclical patterns both in schooling behaviors and during migrations. Atlantic bluefin tuna (*Thunnus thynnus*), for example, undertake precise annual migrations across the Atlantic Ocean, following consistent routes that correlate with oceanographic features [70]. Recent research by Papastamatiou et al. [70] demonstrated that these migrations involve regular, stereotyped diving behaviors forming diel vertical migration cycles that appear linked to both thermoregulation and navigation.

At a smaller scale, schooling fish demonstrate remarkably stable group movement patterns despite each individual following relatively simple behavioral rules. These emergent cyclical patterns—oscillations between expansion and contraction, coordinated turns, and polarized movement—emerge from local interactions rather than global coordination [18]. This represents a powerful example of how complex navigation can emerge from simple rules when coupled with environmental interactions and social dynamics.

1.2.4 Insect Navigation Using Path Integration

Even organisms with relatively simple nervous systems demonstrate sophisticated cyclical navigation strategies. Desert ants (*Cataglyphis sp.*) navigate across featureless terrain using path integration, continuously calculating their position relative to their nest by integrating the distance and direction traveled [98]. When uncertainty in their position estimate increases, these ants engage in characteristic search patterns consisting of loops of increasing size centered on their estimated nest location. This systematic search pattern demonstrates how simple cyclical behaviors can efficiently resolve navigational uncertainty with minimal computational resources.

Similarly, honeybees use the famous "waggle dance" to communicate the location of food sources relative to the hive. This communication system encodes distance and direction information through the duration and orientation of a figure-eight movement pattern [25]. Beyond communication, bees also use systematic scanning flights characterized by regular oscillations to map novel environments and calibrate their visual navigation systems [20].

These biological examples collectively illustrate how cyclical patterns—whether in movement, sensor sampling, or internal state transitions—can serve as the foundation for robust navigation strategies that require minimal sensory input and computational resources. The ubiquity of such patterns across diverse species suggests their fundamental utility as a navigation paradigm, particularly in scenarios where efficiency and resilience are prioritized over absolute precision. Without turning it into bio-inspired robotics, in which the goal is to faithfully imitate animal behaviors by implementing it into robots [26, 109], this work instead applies general concepts of animal navigation to robots.

1.3 Benefits of Cyclic Navigation for Marine Robotics

The application of biologically-inspired cyclical navigation patterns to marine robotics offers several compelling advantages over conventional approaches. These benefits address many of the core challenges facing current marine robotic systems.

1.3.1 Energy Efficiency

Energy constraints represent one of the most significant limitations for autonomous marine robots, particularly for long-duration missions. Biologically-inspired cyclical navigation approaches offer substantial energy efficiency improvements through multiple mechanisms. First, by reducing computational demands through simpler algorithmic approaches, power consumption from onboard processing can be dramatically reduced. Second, cyclical movement patterns can be designed to exploit environmental dynamics, such as currents, wave action, or thermal gradients, similar to how birds utilize thermals or fish leverage ocean currents. In [99], Weihs demonstrated theoretically that appropriately designed oscillatory swimming patterns can reduce energy consumption by up to 25% compared to constant-speed straight-line travel. Furthermore, Webb and Keyes [97] showed empirically that fish-inspired undulatory propulsion coupled with optimized swimming cycles could achieve propulsive efficiencies exceeding conventional marine propulsion systems.

1.3.2 Robustness and Resilience

Biological navigation systems demonstrate remarkable robustness in the face of environmental variability, sensor limitations, and even physical damage. This resilience stems largely from their reliance on redundant, overlapping systems and adaptive behavioral patterns rather than brittle, high-precision approaches. By implementing cyclical navigation patterns that continuously interact with and sample the environment, robotic systems can achieve similar resilience properties.

Cyclical approaches naturally incorporate continuous error correction, as demonstrated by the path integration and systematic search patterns of desert ants [98]. When position estimates become uncertain, expanding the search pattern can efficiently relocalize the system without requiring absolute position information. Additionally, the repetitive nature of cyclical patterns facilitates the detection of environmental changes through comparison across cycles, enabling adaptation to dynamic conditions without requiring comprehensive environmental models.

1.3.3 Simplicity and Scalability

Perhaps the most significant advantage of biologically-inspired cyclical navigation is its inherent simplicity. Rather than requiring complex world models, detailed maps, or sophisticated sensor fusion algorithms, cyclical approaches can operate effectively using relatively simple state machines and minimal sensor inputs. This simplicity translates directly to improved reliability, reduced development costs, and greater ease of deployment across diverse platforms.

The scalability of cyclical navigation approaches is particularly relevant for marine robotics applications that may require the deployment of multiple coordinated vehicles. Simple, rule-based navigation systems enable more effective scaling to multi-robot systems, as demonstrated by the emergent coordination seen in fish schools and bird flocks [18]. This scalability advantage becomes increasingly important as marine research and monitoring applications evolve toward distributed sensing paradigms requiring coordinated swarms of autonomous vehicles.

1.4 Research Approach: Stable Cycles for Marine Navigation

This thesis focuses on the navigation of robots inspired by the cyclical patterns that can be found in nature [55], without actually adopting bio-inspired robotics [8]. Rather than attempting to mimic specific biological mechanisms, the research abstracts the fundamental

principles of cyclical navigation into a computational framework suitable for implementation on autonomous marine robots. In this navigation paradigm, measurements trigger events to control the shape of the cycle, and the robot behavior. This event-driven approach is not new [75, 61] but far from being widespread in the field of marine robotics.

At its core, the approach leverages stable limit cycles—self-sustaining oscillatory trajectories—as the fundamental building blocks of navigation behavior. These limit cycles are generated through carefully designed nonlinear dynamical systems expressed as state machines with minimal sensory inputs. The state machines process raw data from basic exteroceptive sensors (e.g., light, magnetic field, pressure, water flow) [68, 106] to modulate the parameters of the limit cycles, creating environmentally responsive yet inherently stable navigation behaviors.

Unlike conventional approaches that separate perception, planning, and control into distinct systems, this research integrates these functions through the direct coupling of sensory inputs to the parameters of the limit cycle generators. This tight coupling creates an embodied navigation system where the robot's physical interaction with the environment becomes an integral part of the navigation process itself, similar to how sea turtles' swimming patterns interact with ocean currents to maintain heading.

The implementation approach emphasizes computational frugality, requiring only modest onboard processing capabilities comparable to microcontroller-class systems rather than the powerful embedded computers typically associated with autonomous marine robots. This computational efficiency stems from the use of simple state transition rules rather than complex optimization algorithms or neural network computations. The resulting system maintains stable navigation behavior while continuously adapting to environmental conditions through the modulation of key limit cycle parameters based on sensor inputs. Preliminary experiments conducted in controlled tank environments and limited field trials demonstrate that these biologically-inspired cyclical navigation patterns can achieve reliable goal-directed navigation using significantly fewer computational resources than conventional approaches. Particularly promising results have been observed in scenarios involving gradient following, boundary tracking, and homing behaviors—tasks that mirror the ecological navigation challenges faced by the biological systems that inspired this work.

1.5 Potential Impact and Applications

The development of biologically-inspired cyclical navigation systems for marine robots has the potential to transform multiple fields that rely on autonomous marine exploration and monitoring. By enabling longer missions with simpler, more energy-efficient systems, this approach directly addresses several key challenges in marine robotics applications.

1.5.1 Oceanographic Research

Oceanographic research increasingly depends on autonomous platforms to collect data across vast spatial and temporal scales that would be impossible to cover with surface vehicle based methods alone. Current autonomous oceanographic systems face significant limitations in deployment duration and operational range due to energy constraints and reliability issues. The cyclical navigation approach developed in this research could enable substantially longer deployments of autonomous oceanographic sensors, particularly for applications such as long-term monitoring of dynamic oceanographic features, tracking of marine biological phenomena, or persistent sampling of environmentally sensitive regions [81].

1.5.2 Environmental Monitoring

Environmental monitoring applications, from water quality assessment to harmful algal bloom tracking, require increasingly responsive and adaptive sampling strategies. The cyclical navigation approach is particularly well-suited to adaptive environmental monitoring scenarios, as it naturally implements behaviors such as boundary tracking, gradient following, and source localization—all critical capabilities for effective environmental monitoring [110]. By enabling these capabilities with minimal computational overhead, the approach could dramatically expand the deployment scale of environmental monitoring systems while reducing their cost and complexity.

1.5.3 Autonomous Marine Systems

Beyond scientific applications, the principles developed in this research have significant implications for commercial and industrial autonomous marine systems. From autonomous surface vessels for shipping and logistics to subsea inspection and maintenance robots, the maritime industry increasingly seeks more reliable, efficient autonomous capabilities. The robust, energy-efficient navigation approach presented here addresses key industry challenges related to operational endurance, reliability in GPS-denied environments, and resilience to sensor failures or environmental disturbances.

1.5.4 Biologically-Inspired Robotics

This research contributes to the broader field of biologically-inspired robotics by demonstrating how fundamental principles extracted from biological navigation systems can be effectively translated into engineered systems. Rather than attempting to directly replicate biological mechanisms, which often proves challenging due to the significant differences between biological and engineered substrates, this work focuses on abstracting functional principles at a higher level. This approach to bio-inspiration—focusing on behavioral patterns and system-level organization rather than mechanism-level mimicry—provides a template for future bio-inspired robotics research across domains.

1.5.5 Surveillance and Security

The cycle navigation approach also has potential applications in surveillance and security operations, particularly in scenarios where persistent monitoring of large areas is required. This method allows a deployment of autonomous underwater vehicles without any external positioning system setup, such as Long BaseLine (LBL) or Ultra-Short BaseLine (USBL) which require extensive infrastructure, impossible to deploy in hostile areas. Furthermore, this method is based on a few exteroceptive measurements, which guarantees stealth and low detectability. This is well suited for military operations, such as reconnaissance, mine warfare, port or bay surveillance, and enemy area mapping.

1.6 Thesis Statement and Dissertation Structure

This dissertation advances the thesis that biologically-inspired navigation based on stable limit cycles can enable robust, energy-efficient autonomous navigation for marine robots using minimal sensor data and computational resources. Through theoretical development, simulation studies, and experimental validation, the work demonstrates that appropriately designed cyclical navigation patterns can achieve performance comparable to conventional approaches while significantly reducing computational complexity and energy requirements. The remainder of this dissertation is organized as follows:

Figure 1.2 shows the organization of the manuscript. This manuscript is divided into two parts: Part I Tools and Part II Contributions. Part I presents the theoretical tools which are prerequisites for the contributions presented in Part II.

In this first part, Chapter 2 introduces the mathematical tools used to model dynamical systems, Chapter 3 presents the formalism of automata theory, particularly timed automata, that plays a crucial role in the cycle navigation, and then Chapter 4 introduces the set methods that will be used to prove the stability, and to solve the state estimation problem.

In the contributions part, Chapter 5 presents the formalism of the cycle navigation, and the premises of the cycle control, which is the main contribution of this thesis. Chapter 6 then presents the stability analysis of the cycle control, that will be done using the set methods introduced in Chapter 4. Chapter 7 introduces strategies that can be used to navigate using the cycle paradigm, and Chapter 8 solves the state estimation problem using set methods, and estimate the state of the cycle.

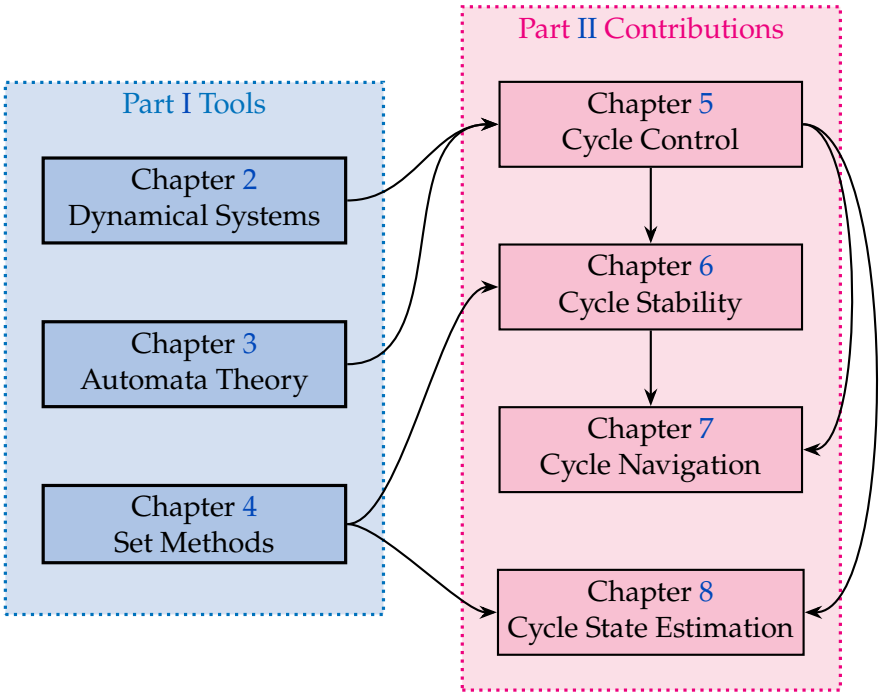


Figure 1.2 Organization of the manuscript

Part I:

Tools and Formalism

2 Dynamical systems modelling

2.1 Introduction

The study of dynamical systems and their control represents a cornerstone of modern robotics, providing the theoretical foundation for systems that can navigate complex environments with precision and reliability. This section introduces the fundamental concepts of control theory as applied to dynamical systems, which forms the basis for our research on navigation using stable cycles.

This chapter aims to provide a way to model the dynamics of robotic systems that can be continuous or discrete, and to understand how these systems evolve over time. Such models will be useful in Chapter 5 to define the cycle navigation designed for a robot. This navigation concept is based on the coupling of a timed automaton with a robot, which allows the robot to navigate along a cycle, that is continuous on each segment in the state space of the robot. This is why both continuous and discrete dynamical systems are presented in this chapter.

Control theory examines how the behavior of dynamical systems can be modified through feedback mechanisms to achieve desired outcomes. We present this theoretical framework by distinguishing between two principal mathematical formulations: continuous dynamical systems and discrete dynamical systems. This classification reflects not only different mathematical representations but also distinct approaches to control design and analysis.

Continuous dynamical systems model phenomena where state variables evolve smoothly over time, typically described through differential equations. These systems capture the physics of motion and force interactions that govern robotic platforms in their environments. In contrast, discrete dynamical systems represent processes where state updates occur at specific time instances, described by difference equations. This formulation proves particularly valuable for digital control implementations and for analyzing systems at strategic sampling points.

These two approaches allow us to model a wide range of systems, from simple mechanical systems to complex robotic platforms. They both play an important role in the cycle navigation.

The following sections detail the mathematical foundations of both continuous and discrete dynamical systems, exploring their stability properties, control strategies, and limitations. This theoretical groundwork establishes the context for our novel approach to navigation using stable cyclic behaviors, which addresses key challenges in modern robotics applications.

2.2 Dynamical Systems

2.2.1 General definition

Definition 1. *A **dynamical system** describes the evolution of a system according to a set of differential equations [89]. In its most general form, we consider a state variable $\mathbf{x}(t) \in \mathcal{S}$ that evolves according to*

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t), \tag{2.1}$$

where,

- *(i) $\mathcal{T}$ is the time set containing the evolution parameter t,*
- *(ii) $\mathcal{S}$ is the state space containing the system state $\mathbf{x}$,*
- *(iii) $\mathbf{f} : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ is the evolution function that describes the dynamics of the system.*

Remark. *Most robotic systems have either $\mathbb{Z}$ or $\mathbb{R}$ as time set $\mathcal{T}$, and $\mathbb{R}^n$ as state space $\mathcal{S}$.*

Definition 2. *An **autonomous dynamical system** is a special case of a dynamical system where the evolution function does not explicitly depend on the independent variables such as time [89]. In this case, the system can be expressed as:*

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)). \tag{2.2}$$

In the context of robotic systems, an input $\mathbf{u}(t)$ is used to control the system.

Definition 3. *A **controlled dynamical system** is a dynamical system that includes control inputs [89]. It can be expressed as:*

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.3}$$

where

(i) $\mathcal{U}$ is the input space containing the control inputs $\mathbf{u}$.

This model of controlled dynamical system is well suited to model a robot, which has a state $\mathbf{x}(t)$ that evolves over time according to the input $\mathbf{u}(t)$.

Example 1. *Consider the unicycle model [84] of a mobile robot, which is a common model used in robotics. The state of the robot is $\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$, where $(x, y) \in \mathbb{R}^2$ is the position of the robot in the plane and $\theta \in \mathbb{S}^1$ is the orientation of the robot. The input vector is $\mathbf{u} = \begin{bmatrix} v & \omega \end{bmatrix}^T$, where v is the linear velocity, and ω is the angular velocity of the robot. The evolution of the robot is given by*

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix}. \tag{2.4}$$

2.2.2 Initial Value Problem

To determine the behavior of a dynamical system, we need to solve the Initial Value Problem (IVP), which combines the differential equation with an initial condition.

Definition 4. *An **Initial Value Problem (IVP)** is a problem defined by a differential equation and an initial condition [89]. It follows*

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \end{cases}, \tag{2.5}$$

where

(i) $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ is the evolution equation of an autonomous system,
(ii) t_0 is the initial time,
(iii) $\mathbf{x}_0$ is the initial state of the system.

Under appropriate conditions on the evolution function $\mathbf{f}$, specifically when $\mathbf{f}$ is Lipschitz continuous, the Picard-Lindelöf theorem guarantees the existence and uniqueness of a solution to the IVP for some time space $\mathcal{T}$ containing t_0 [89].

This solution is the trajectory of the system in the state space $\mathbf{x}(t)$, which represents the state of the system at time t given the initial condition $\mathbf{x}_0$ at time t_0.

2.2.3 The Flow Function

The solution to the IVP is the flow function, denoted as $\phi(t, t_0, \mathbf{x}_0)$, which represents the state of the system at time t when starting from the initial state $\mathbf{x}_0$ at time t_0. The flow function is a fundamental concept in dynamical systems, as it describes how the state of the system evolves over time.

Definition 5. *The **flow function** $\phi(t, t_0, \mathbf{x}_0)$ is defined as the solution to Equation (2.5)*

$$\mathbf{x}(t) = \phi(t, t_0, \mathbf{x}_0) \tag{2.6}$$

Remark. *For autonomous systems, where the dynamics do not explicitly depend on time, the flow function simplifies to $\phi(t - t_0, \mathbf{x}_0)$, reflecting time-invariance of the system [85, 45]*

Example 2. *Recalling the unicycle model, the flow function can be derived from the evolution equation Equation (2.4). The solution of this evolution function can be expressed analytically from a starting state $\mathbf{x}_0 = \begin{bmatrix} x_0 & y_0 & \theta_0 \end{bmatrix}^T$ if the input vector $\mathbf{u}$ is constant. As the unicycle model is time-invariant, the initial time t_0 is irrelevant, and we can set $t_0 = 0$ without loss of generality.*

In the case of a constant linear velocity v and a null angular velocity $\omega = 0$, the trajectory of the robot is a straight line, and the flow function is given by

$$\phi(t, \mathbf{x}_0) = \begin{bmatrix} x_0 + vt\cos(\theta_0) \\ y_0 + vt\sin(\theta_0) \\ \theta_0 \end{bmatrix}. \tag{2.7}$$

If the linear velocity is constant and the angular velocity is constant $\omega \neq 0$, the trajectory of the robot is a circular arc, and the flow function is given by

$$\phi(t, \mathbf{x}_0) = \begin{bmatrix} x_0 + \frac{v}{w}(\sin(\theta_0 + \omega t) - \sin(\theta_0)) \\ y_0 - \frac{v}{w}(\cos(\theta_0 + \omega t) - \cos(\theta_0)) \\ \theta_0 + \omega t \end{bmatrix}. \tag{2.8}$$

The flow function possesses several important properties which will be used in the following sections.

Property 1. *The flow function $\phi(t, t_0, \mathbf{x}_0)$ satisfies the following properties:*

(i) $\phi(t_0, t_0, \mathbf{x}_0) = \mathbf{x}_0$ *(initial condition)*
(ii) $\phi(t_2, t_1, \phi(t_1, t_0, \mathbf{x}_0)) = \phi(t_2, t_0, \mathbf{x}_0)$ *(group property)*
(iii) $\frac{\partial}{\partial t}\phi(t, t_0, \mathbf{x}_0) = \mathbf{f}(\phi(t, t_0, \mathbf{x}_0), t)$ *(differentiability)*

This flow function plays a crucial role to define the trajectory of the robot in the state space, which is a key point of the navigation using stable cycles presented in this manuscript. By knowing the flow function of the robot, we can determine the shape of the cycle, which is mandatory to design the state automaton that will allow the cycle to be stable in the environment.

2.2.4 Trajectory in State Space

Definition 6. *The **trajectory** or **orbit** [74, 102, 73] of a dynamical system from an initial state $\mathbf{x}_0$ is the set of points in the state space visited by the system as time evolves:*

$$\gamma(\mathbf{x}_0) = \{\phi(t, t_0, \mathbf{x}_0) : t \in \mathcal{T}\}$$

The collection of all possible trajectories forms the phase portrait of the system, providing a geometric representation of the system's behavior across the entire state space. This geometric interpretation is particularly valuable for understanding the qualitative behavior of nonlinear systems where closed-form solutions often cannot be obtained.

In the context of navigation using stable cycles, these trajectories become especially significant as we seek to identify and utilize cycle-based patterns that exhibit stability properties advantageous for robotic control strategies.

Definition 7. *A **closed trajectory** or **periodic orbit** is a special type of trajectory in state space that repeats itself after a fixed time interval. For an autonomous system following Equation (2.2), an orbit $\gamma(\mathbf{x}_0)$ is periodic if*

$$\exists T > 0, \forall t \geq t_0, \phi(t+T, t, \mathbf{x}_0) = \phi(t, t, \mathbf{x}_0), \tag{2.9}$$

where the smallest positive value of T satisfying this condition is called the period of the periodic orbit.

Definition 8. *A **periodic point** $\mathbf{x}_p$ of period T is a point in the state space that lies on a periodic orbit. It satisfies the condition*

$$\phi(T, t_0, \mathbf{x}_p) = \mathbf{x}_p. \tag{2.10}$$

The set of all points on this periodic orbit forms a closed curve in the state space.

Definition 9. *The **periodic orbit** Γ is defined as the set of all periodic points in the state space. It can be expressed as:*

$$\Gamma = \{\mathbf{x}_p = \phi(t, t_0, \mathbf{x}_p) \in \mathcal{S}, 0 \leq t < T\} \tag{2.11}$$

where $\mathbf{x}_p$ is a periodic point of period T.

Example 3. *Consider the two-dimensional autonomous system of the undamped harmonic oscillator. The system is described by*

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix} \mathbf{x}(t). \tag{2.12}$$

where $\omega > 0$ is a constant parameter representing the angular frequency of the system. The unique solution to this system with initial condition $\mathbf{x}_0 = \begin{bmatrix} x_0 & v_0 \end{bmatrix}^T$ at time t_0 is given by

$$\mathbf{x}(t) = \begin{bmatrix} x_0 \cos(\omega t) + \frac{v_0}{\omega} \sin(\omega t) \\ -\omega x_0 \sin(\omega t) + v_0 \cos(\omega t) \end{bmatrix}. \tag{2.13}$$

The flow function is derived from Equation (2.13) and is given by

$$\phi(t, \mathbf{x}_0) = \begin{bmatrix} \cos(\omega t) & \frac{1}{\omega} sin(\omega t) \\ -\omega \sin(\omega t) & cos(\omega t) \end{bmatrix} \mathbf{x}_0. \tag{2.14}$$

For any non-zero initial condition $\mathbf{x}_0 \neq 0$, the trajectory traces an ellipse in the phase plane with period $T = \frac{2\pi}{\omega}$. This can be verified by observing that

$$\phi(T, \mathbf{x}_0) = \begin{bmatrix} \cos(2\pi) & \frac{1}{\omega} \sin(2\pi) \\ -\omega \sin(2\pi) & \cos(2\pi) \end{bmatrix} \mathbf{x}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}_0 = \mathbf{x}_0. \tag{2.15}$$

In the phase plane (x_1, x_2), these trajectories form concentric ellipses around the origin, which is an equilibrium point of the system. The shape of these ellipses depends on the value of ω, and each closed trajectory represents a periodic orbit with the same period $T = \frac{2\pi}{\omega}$.

This harmonic oscillator example illustrates how periodic orbits manifest in a simple yet fundamental two-dimensional system, providing insight into the geometric structure of the flow in phase

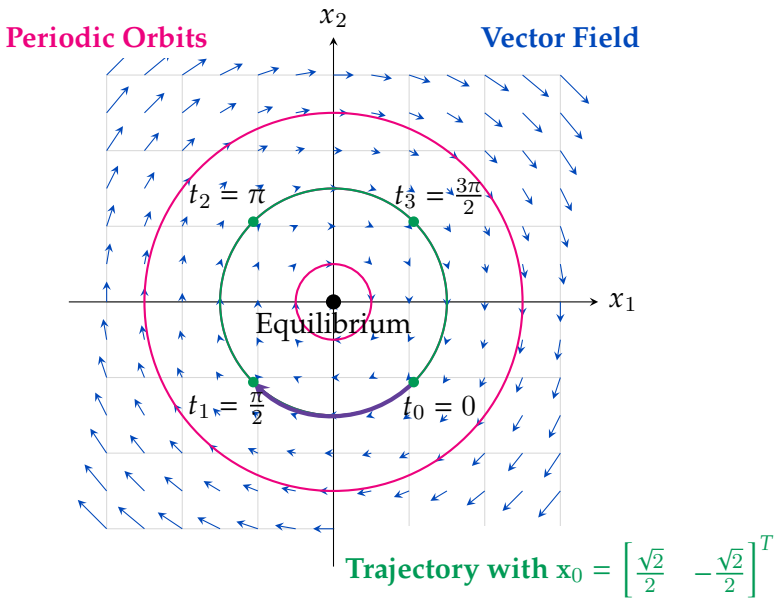


Figure 2.1 Phase portrait of the harmonic oscillator system Equation (2.12) with $\omega = 1\,\mathrm{rad\,s^{-1}}$.

space. Such oscillatory behavior forms the basis for numerous natural and engineered periodic phenomena, from pendulum movements to electrical circuits, and serves as a building block for more complex periodic behaviors in robotic systems.

This example illustrates how periodic orbits manifest in a simple yet fundamental two-dimensional system, providing insight into the geometric structure of the flow in phase space. Such oscillatory behavior forms the basis for numerous natural and engineered periodic phenomena, from pendulum movements to electrical circuits, and serves as a building block for more complex periodic behaviors in robotic systems.

However, this example highlights a continuum of stable cycles, in the sense that it exists another periodic orbit in the neighborhood of each periodic orbit of the pendulum. There are also examples of isolated periodic cycles. These cycles are called limit cycles [85, 45].

The Van der Pol oscillator is a well-known example of a system with a limit cycle. It is described by the following differential equation:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} x_2 \\ -\mu(x_1^2 - 1)x_2 - x_1 \end{bmatrix}. \tag{2.16}$$

The Van der Pol oscillator exhibits a stable limit cycle for $\mu > 0$, which attracts nearby trajectories in the state space. This means that regardless of the initial conditions, the system will converge to this limit cycle over time.

Figure 2.2 illustrates the phase portrait of the Van der Pol oscillator with $\mu = 1$. The limit cycle is shown as a closed curve in the state space, and nearby trajectories converge to this limit cycle.

The navigation using stable cycles presented in this manuscript is based on the concept of limit cycles. The goal is to design a cycle that is stable in the environment, meaning that the robot will converge to this cycle regardless of its initial state. The nature of these cycles is different from the harmonic oscillator, as they are isolated in the state space. This means that there is no continuum of stable cycles around them, and the robot will converge to the limit cycle regardless of its initial state.

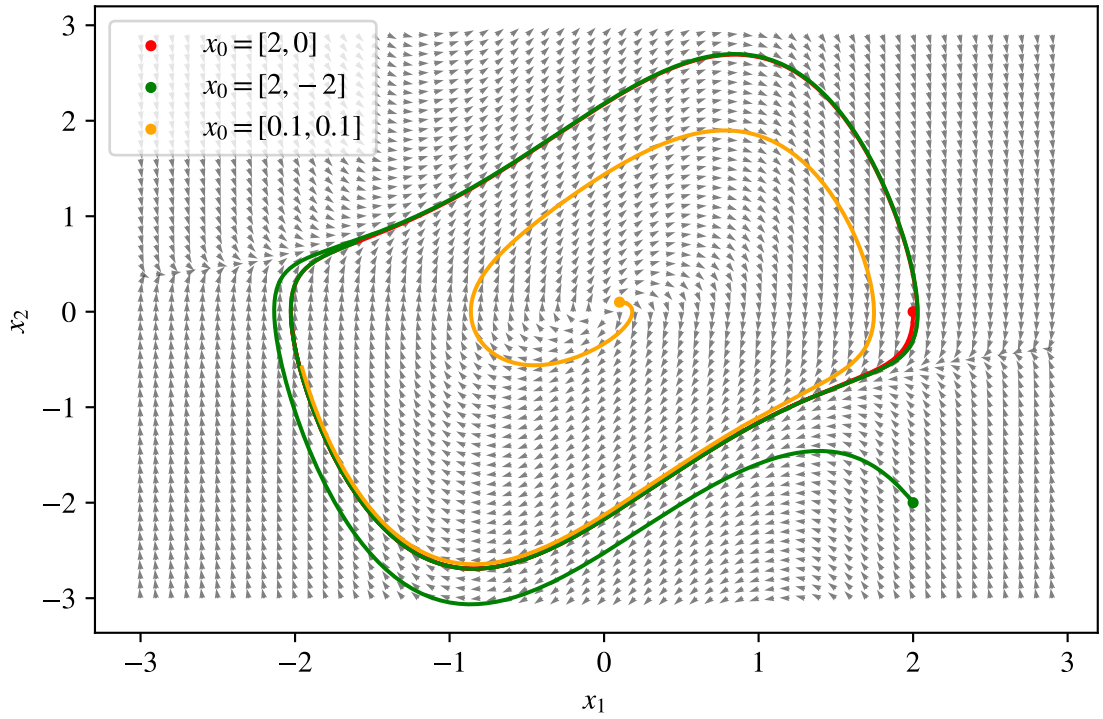


Figure 2.2 Phase portrait of the Van der Pol oscillator Equation (2.16) with $\mu = 1$. Arrows indicate the direction of the flow in the state space, and three trajectories are shown converging to the limit cycle.

2.3 Continuous dynamical systems

This section is dedicated to the presentation of continuous dynamical systems. We are now interested in controlled dynamical systems, which are system that admit an input $\mathbf{u}(t)$ that can be used to control the system. This model is well suited to model a robot, which has a state $\mathbf{x}(t)$ that evolves over time according to the input $\mathbf{u}(t)$. These inputs are typically the commands sent to the actuators, which are mainly thruster velocities and control surface angles in marine robotics.

As the robot needs to estimate its state to be controlled, the measurement equation will also be introduced to model the measurements $\mathbf{y}(t)$ of the system relative to its state $\mathbf{x}(t)$. This model will be fundamental in Chapter 5 to model the robot.

These systems are defined by a set of differential equations that describe the evolution of the state of the system over time. As the system is continuous, the time parameter t is defined in the set of real numbers $\mathbb{R}$.

2.3.1 Nonlinear continuous systems

Definition 10. *A continuous dynamical system of state $\mathbf{x}(t) \in \mathcal{S}$, of input $\mathbf{u}(t) \in \mathcal{U}$, and of output $\mathbf{y}(t) \in \mathcal{Y}$ is a mathematical model of a system defined by*

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)) \\ \mathbf{y}(t) = \mathbf{g}(\mathbf{x}(t)) \end{cases}, \tag{2.17}$$

where $t \in \mathbb{R}$, $\mathbf{f} : \mathcal{S} \times \mathcal{U} \to \mathcal{S}$ and $\mathbf{g} : \mathcal{S} \to \mathcal{Y}$ are two non-linear functions [54].

As $\mathbf{f}$ and $\mathbf{g}$ do not depend on the time t, the system is time-invariant [85, 45], meaning that the system does not change over time. Time-dependent systems will not be consid-

ered in this manuscript, as they are not suitable to model the cycle navigation presented in Chapter 5.

Remark. *Sensors can be of two types: proprioceptive sensors, which measure internal physical quantities of the robot (e.g., linear acceleration, angular velocity, etc.), and exteroceptive sensors, which measure external physical quantities of the robot (e.g., distance to obstacles, position of the robot, etc.). For example, an Inertial Measurement Unit (IMU) measures linear acceleration and angular velocity of the robot. This is then a proprioceptive sensor. A sonar is an exteroceptive sensor, as it measures the distance to obstacles in the environment.*

Example 4. *Suppose a pendulum of length l is swinging under the effect of gravity of intensity g and a torque u generated by a motor at the rotation point, as shown in Figure 2.3.*

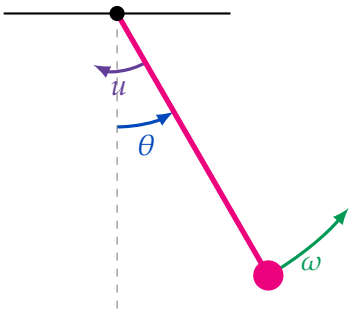


Figure 2.3 Simple pendulum

The state of the system $\mathbf{x} = \begin{bmatrix}\theta & \omega\end{bmatrix}^T$ is defined by the angle θ between the pendulum and the vertical axis and the angular velocity ω. By defining by θ the output of the system, the dynamics of the pendulum follows

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = -\dfrac{g}{l}\sin(\theta) + u \\ y = \theta \end{cases}. \tag{2.18}$$

This differential equation can be put into the form of the continuous dynamical system

$$\begin{cases} \mathbf{f}(\mathbf{x}, u) = \begin{bmatrix} \omega \\ -\frac{g}{l}\sin(\theta) + u \end{bmatrix} \\ g(\mathbf{x}) = \theta \end{cases}. \tag{2.19}$$

2.3.2 Linear continuous systems

In the case where the evolution equation $\mathbf{f}(\mathbf{x}, \mathbf{u})$ is linear, the system is called linear. Linear systems are defined by a set of linear differential equations that describe the evolution of the state of the system over time.

Definition 11. *A linear system [54], see Figure 2.4, of state $\mathbf{x}(t)$, of input $\mathbf{u}(t)$, and of output $\mathbf{y}(t)$ is a mathematical model of a system defined by*

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} = \mathbf{C}\mathbf{x} \end{cases}. \tag{2.20}$$

In most cases, evolution equations $\mathbf{f}$ of systems are non-linear. However, some systems are linear. For instance, this is the case for the harmonic oscillator.

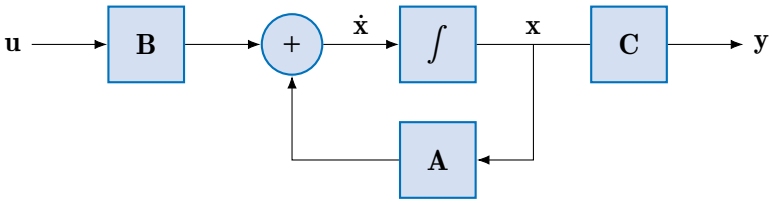


Figure 2.4 Block diagram of a continuous linear system

Example 5. *The harmonic oscillator is a continuous dynamical system that, when displaced from its equilibrium position, experiences a restoring force proportional to the displacement. This system is modelled by*

$$\begin{cases} \ddot{x} = -\omega^2 x + u \\ y = x \end{cases}, \tag{2.21}$$

where ω is the angular frequency of the oscillator. This system can be put into the form of a continuous linear system defined by

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\omega^2 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{C} = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{2.22}$$

A common approach to deal with non-linear systems is to linearize them around an equilibrium point $(\overline{\mathbf{x}}, \overline{\mathbf{u}})$ to obtain a linear approximation of the system.

Consider the continuous dynamical system defined by $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$. Around an equilibrium point $\overline{\mathbf{x}}$, the state can be expressed as a deviation $\tilde{\mathbf{x}} = \mathbf{x} - \overline{\mathbf{x}}$ from this point. The input of the system is also expressed as a deviation $\tilde{\mathbf{u}} = \mathbf{u} - \overline{\mathbf{u}}$ from the equilibrium input. The idea is then to write the differential equation of the system in terms of $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{u}}$ by using Taylor expansion of the evolution equation, and by neglecting terms of order two and higher.

$$\dot{\mathbf{x}} = \mathbf{f}(\overline{\mathbf{x}}, \overline{\mathbf{u}}) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})(\mathbf{x} - \overline{\mathbf{x}}) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})(\mathbf{u} - \overline{\mathbf{u}}). \tag{2.23}$$

Since $\overline{\mathbf{x}}$ is an equilibrium point, $\mathbf{f}(\overline{\mathbf{x}}, \overline{\mathbf{u}}) = \mathbf{0}$. Therefore, with $\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})$ and $\mathbf{B} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})$

$$\dot{\tilde{\mathbf{x}}} = \mathbf{A}\tilde{\mathbf{x}} + \mathbf{B}\tilde{\mathbf{u}}. \tag{2.24}$$

Example 6. *The pendulum system can be linearized around its equilibrium state $\overline{\mathbf{x}} = \begin{bmatrix} 0 & 0 \end{bmatrix}$, and on this state the input $u = 0$. By using Taylor expansion of the pendulum dynamics of Equation (2.18) around this equilibrium state, the linearized system follows*

$$\ddot{\theta} = -\frac{g}{l}\theta + u. \tag{2.25}$$

This equation models the evolution of a continuous linear system with

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{l} & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}. \tag{2.26}$$

2.4 Discrete dynamical systems

There are dynamical systems that are described by discrete evolution equations. This is the case for instance of financial transactions, traffic lights, digital electronics, communication systems, and so on. In these cases, the state of the system is updated at discrete time

steps. The discrete systems are defined by a set of difference equations that describe the evolution of the state of the system over time. As the system is discrete, the time parameter t is defined in the set of integers $\mathbb{Z}$. This discretization can also be a way to model a continuous system which is sampled at discrete time intervals, as in the case of a robot that is controlled at a fixed frequency. This will be the case in Chapter 5, in which the robot modelled by continuous dynamical system is discretized along its trajectory constant per segment. In this case, the discretization is asynchronous and event driven, as the duration of segment is not fixed, and can be driven by measurements in the environment.

2.4.1 Non-linear discrete systems

Definition 12. *A discrete dynamical system of state $\mathbf{x}_k \in \mathcal{S}$, of input $\mathbf{u}_k \in \mathcal{U}$, and of output $\mathbf{y}_k \in \mathcal{Y}$ is a mathematical model of a system defined by*

$$\begin{cases} \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{y}_k = \mathbf{g}(\mathbf{x}_k) \end{cases}, \tag{2.27}$$

where $t \in \mathbb{Z}$, $\mathbf{f} : \mathcal{S} \to \mathcal{S}$ and $\mathbf{g} : \mathcal{S} \to \mathcal{Y}$ are two non-linear functions [54].

Example 7. *The pendulum system can be discretized by using the Euler method, by approximating the derivative of the state by the tangent equation $\dot{\mathbf{x}} = \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{\Delta t}$, then*

$$\mathbf{f}(\mathbf{x}_k, u_k) = \begin{bmatrix} \theta_k + \Delta t \dot{\theta}_k \\ \dot{\theta}_k + \Delta t \left(u_k - \frac{g}{l} \sin(\theta_k) \right) \end{bmatrix}. \tag{2.28}$$

2.4.2 Linear discrete systems

Definition 13. *A linear system of state $\mathbf{x}_k \in \mathcal{S}$, of input $\mathbf{u}_k \in \mathcal{U}$, and of output $\mathbf{y}_k \in \mathcal{Y}$ is a mathematical model of a system defined by*

$$\begin{cases} \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k = \mathbf{C}\mathbf{x}_k \end{cases}. \tag{2.29}$$

Figure 2.5 shows the block diagram representation of a discrete linear system. This representation uses the z-transform to express the delay between $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$.

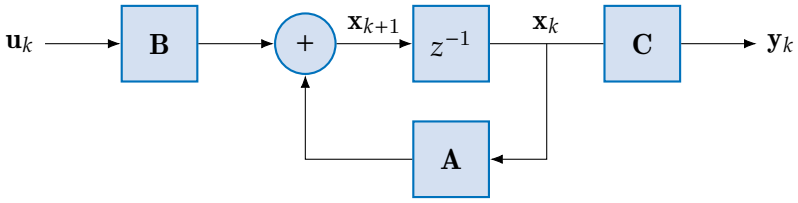


Figure 2.5 Block diagram of a discrete linear system

Remark. *Sometimes discrete systems are modeled by*

$$\begin{cases} \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \\ \mathbf{y}_k = \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \end{cases} \tag{2.30}$$

In this equation, the output $\mathbf{y}_k$ depends on the input $\mathbf{u}_k$. In this case, it is possible to return to a system verifying Equation (2.29) by defining a new output $\mathbf{z}_k = \mathbf{y}_k - \mathbf{D}\mathbf{u}_k$.

Example 8. *Consider a bouncing ball of mass m falling under gravity g and bouncing on the ground. The state of the ball at time t is denoted by $\mathbf{x} = \begin{bmatrix} z & v \end{bmatrix}^T$ where z is the height of the ball and v is its velocity. The initial state of the ball is $\mathbf{x}_0 = \begin{bmatrix} z_0 & 0 \end{bmatrix}^T$. The ball dynamics follows the free fall evolution equation*

$$\dot{\mathbf{x}} = \begin{cases} v \\ -g \end{cases}. \tag{2.31}$$

A bounce occurs when $z = 0$. By denoting by t_k the time of the k^{th} bounce, by $\mathbf{x}_k$ the state of the ball, and by v_k the velocity of the ball at this time, the bounce effect on the velocity follows

$$v(t_k^+) = -ev(t_k^-), \tag{2.32}$$

where e is the coefficient of restitution ($0 \leq e \leq 1$), $v(t_k^-)$ is the velocity of the ball just before the bounce, and $v(t_k^+)$ is the velocity just after the bounce. If $e = 1$ the bounce is perfectly elastic and the velocity will not decrease over bounces, while if $e = 0$ the bounce is perfectly inelastic and the velocity will be null after the first bounce.

The ball will bounce asynchronously. This means that the time between two successive bounces is not constant and depends on the velocity of the ball. The bouncing time t_k follows

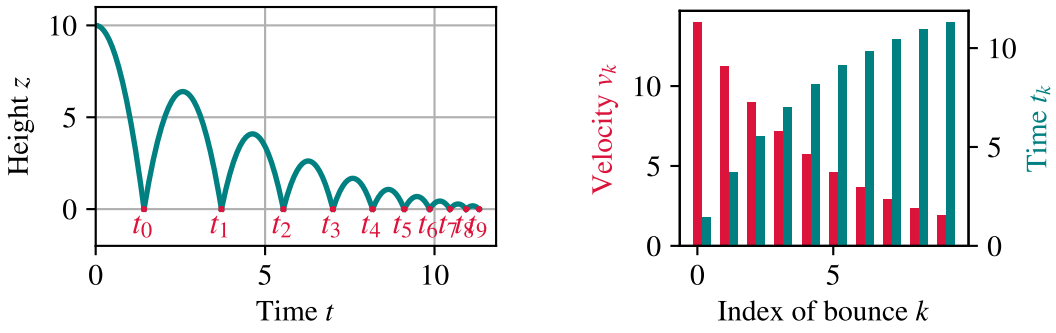
$$\begin{cases} t_0 = \sqrt{2gz_0} \\ t_{k+1} = t_k + \dfrac{2|v_k|}{g} \end{cases} \tag{2.33}$$

Supposing the output of the system y_k is the velocity of the ball at the k^{th} bounce. The system follows

$$\begin{cases} v_{k+1} = ev_k \\ y_k = v_k \end{cases}, \tag{2.34}$$

with the coefficients of the discrete dynamical system that can be identified by $A = e$, $B = 0$, and $C = 1$.

Figure 2.6 shows the simulation of the bouncing ball system with an initial height of $z_0 = 10\,\mathrm{m}$, and a restitution coefficient $e = 0.8$. The height of the ball over time is shown in Figure 2.6a, while the velocity v_k and the bounce time t_k over bounces are shown in Figure 2.6b. This simulation shows the asynchronous behavior of the considered discrete system of output the velocity of the ball at each bounce.



(a) Height of the ball over time **(b)** Velocity v_k and bounce time t_k over bounces

Figure 2.6 Bouncing ball simulation

As for continuous systems, it is possible to linearize discrete systems around an equilibrium point $(\overline{\mathbf{x}}, \overline{\mathbf{u}})$ to obtain a linear approximation of the system. This is usefull to apply results available for linear systems.

Consider the discrete nonlinear dynamical system defined by $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$. Around an equilibrium point $\overline{\mathbf{x}}$, the state can be expressed as a deviation $\tilde{\mathbf{x}} = \mathbf{x}_k - \overline{\mathbf{x}}$ from this point. The input of the system is also expressed as a deviation $\tilde{\mathbf{u}} = \mathbf{u}_k - \overline{\mathbf{u}}$ from the equilibrium input. The idea is then to write the difference equation of the system in terms of $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{u}}$ by using Taylor expansion of the evolution equation, and by neglecting terms of order two and higher.

$$\mathbf{x}_{k+1} = \mathbf{f}(\overline{\mathbf{x}}, \overline{\mathbf{u}}) + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})(\mathbf{x}_k - \overline{\mathbf{x}}) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})(\mathbf{u}_k - \overline{\mathbf{u}}). \tag{2.35}$$

Since $\overline{\mathbf{x}}$ is an equilibrium point, $\mathbf{f}(\overline{\mathbf{x}}, \overline{\mathbf{u}}) = \overline{\mathbf{x}}$, and it can be combined with $\mathbf{x}_{k+1}$ to form $\tilde{\mathbf{x}}_{k+1}$. Therefore, with $\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})$ and $\mathbf{B} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\overline{\mathbf{x}}, \overline{\mathbf{u}})$

$$\tilde{\mathbf{x}}_{k+1} = \mathbf{A}\tilde{\mathbf{x}} + \mathbf{B}\tilde{\mathbf{u}}. \tag{2.36}$$

Example 9. *The linearized pendulum could also be discretized by using the Euler method. The discrete system follows*

$$\mathbf{f}(\mathbf{x}_k, u_k) = \begin{bmatrix} 1 & \Delta t \\ -\frac{\Delta t g}{l} & 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ \Delta t \end{bmatrix} u_k. \tag{2.37}$$

2.5 Controllability and Observability

Controllability and observability were introduced in [41]. These criteria are particularly used to classify systems in terms of their ability to be controlled to any state, and whether from the outputs the state of the system can be reconstructed. Moreover, they are only valid for linear systems. These criteria are only applicable to linear systems, but for both continuous and discrete systems.

In this work, these concepts will play a crucial role, as the robot is equipped with a few sensors, and needs to evolve in a reduced event-driven environment. We should then ensure that the system is controllable and sufficiently observable to use the cycle navigation, but the system will not be fully observable as the localization of the robot will remain unknown. All these concepts will be used in Chapter 5.

Remark. *In the non-linear case, the controllability and observability can be studied around an operating point by linearizing the system around this point. This gives an insight on the controllability and the observability of the system, but no results can be guaranteed in this case.*

2.5.1 Controllability

The controllability is a criterion which expresses the possibility to change the state of the system from any initial value to any final value within a finite time window [41, 54].

Definition 14. *A linear system in dimension n defined by Equation (2.29) is **controllable** if and only if the controllability matrix is $C = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \dots & \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix}$ has a full row rank. In other words, the rank of the controllability matrix is equal to n.*

2.5.2 Observability

The observability is a criterion which expresses the ability to retrieve the state of the system from its outputs [41, 54].

Definition 15. *A linear system in dimension n defined by Equation (2.29) is **observable** if and only if the observability matrix* $O = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix}$ *has a full column rank. In other words, the rank of the observability matrix is equal to n.*

2.6 State observer

In robotics, the state of a dynamical system often includes variables that are not directly measurable. For instance, while we can measure the position of a robot, we might not be able to measure its velocity or internal states directly. State observers are designed to estimate the state of the dynamical system from the measurable outputs and inputs of the system.

The primary goal of a state observer is to provide an estimate of the system's state vector $\mathbf{x}(t)$, denoted as $\hat{\mathbf{x}}(t)$, using the system's input $\mathbf{u}(t)$ and output $\mathbf{y}(t)$. This estimation is crucial for implementing advanced control strategies, such as state feedback control, where the full state vector is required.

2.6.1 Need for a state observer

In the linear case, and in the case of sufficiently well-chosen measurements, the matrix $\mathbf{C}$ can be inverted to obtain the state of the system from measurements [54] using

$$\hat{\mathbf{x}} = \mathbf{C}^{-1}\mathbf{y}. \tag{2.38}$$

However, when the dimension of the output vector $\mathbf{y}$ is not the same as the dimension of the state vector $\mathbf{x}$, it becomes impossible to directly infer the state from the output. This discrepancy is common in robotics, where sensors provide limited information about the system. State observers bridge this gap by using a model of the system dynamics to estimate the full state vector.

Example 10. *For example, consider a robotic arm with two rotational joints. The state vector of this system is the two joint angles* $\mathbf{x} = \begin{bmatrix} \theta_1 & \theta_2 \end{bmatrix}^T$. *If high quality encoders are measuring angles, the output vector is* $\mathbf{y} = \mathbf{C}\mathbf{x} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$. *The estimated state is* $\hat{\mathbf{x}} = \mathbf{C}^{-1}\mathbf{y}$.

2.6.2 Luenberger observer design

Continuous Luenberger observer

The Luenberger observer is a state observer that estimates the state of a linear system using the system's input and output [59]. The estimated state is denoted by $\hat{\mathbf{x}}(t)$, and is adjusted at each iteration by comparing predicted outputs $\hat{\mathbf{y}}(t) = C\hat{\mathbf{x}}(t)$ with real outputs $\mathbf{y}(t)$. The estimated state $\hat{\mathbf{x}}(t)$ follows

$$\begin{cases} \dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{y} - \hat{\mathbf{y}}) \\ \hat{\mathbf{y}} = \mathbf{C}\hat{\mathbf{x}} \end{cases}. \tag{2.39}$$

By denoting by $\boldsymbol{\epsilon} = \hat{\mathbf{x}} - \mathbf{x}$, and by using Equation (2.39), then

$$\dot{\boldsymbol{\epsilon}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u} + \mathbf{L}(\mathbf{C}\hat{\mathbf{x}} - \mathbf{C}\mathbf{x}) - \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{u} = (\mathbf{A} - \mathbf{L}\mathbf{C})\boldsymbol{\epsilon}. \tag{2.40}$$

This expression gives a condition for correctly tuning the matrix $\mathbf{L}$. The goal is to ensure that the error $\boldsymbol{\epsilon}$ converges to zero over time, meaning that the estimated state $\hat{\mathbf{x}}$ converges toward the true state $\mathbf{x}$. The concept of Hurwitz matrices need to be introduced now.

Definition 16. *A **Hurwitz matrix** is a square matrix whose eigenvalues all have strictly negative real parts. In other words, a matrix $\mathbf{A}$ is Hurwitz if and only if all eigenvalues λ of $\mathbf{A}$ satisfy $\mathfrak{Re}(\lambda) < 0$.*

By choosing $\mathbf{L}$ such that $(\mathbf{A} - \mathbf{LC})$ is a Hurwitz matrix, $\boldsymbol{\epsilon}$ is asymptotically stable, so $\lim_{t \to +\infty} \boldsymbol{\epsilon} = 0$, and then $\hat{\mathbf{x}}$ converges toward $\mathbf{x}$. This concept of stability is widely developed in the next section about the stability of dynamical systems.

Remark. *Luenberger state observer is tuned ensuring the real part of the poles of the matrix $(A - LC)$ are negative to guarantee the stability of the observer. The closer the poles are to zero, the faster is the observer. However, this can lead to a peaking phenomenon due to high-gains, which can lead to an unsafe implementation [45].*

Discrete Luenberger observer

Luenberger observers can also be designed for discrete systems [59]. The estimated state is denoted by $\hat{\mathbf{x}}_k$, and is adjusted at each iteration by comparing predicted outputs $\hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{y}}_k$ with real outputs $\mathbf{y}_k$. The estimated state $\hat{\mathbf{x}}_k$ follows

$$\begin{cases} \hat{\mathbf{x}}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k - \mathbf{L}(\hat{\mathbf{y}}_k - \mathbf{y}_k) \\ \hat{\mathbf{y}}_k = \mathbf{C}\hat{\mathbf{x}}_k \end{cases}. \tag{2.41}$$

The same derivation can be led for the discrete Luenberger observer, to find a tuning condition. By denoting by $\boldsymbol{\epsilon}_k = \hat{\mathbf{x}}_k - \mathbf{x}_k$, and by using Equation (2.41), then

$$\boldsymbol{\epsilon}_{k+1} = \mathbf{A}\hat{\mathbf{x}}_k + \mathbf{B}\mathbf{u}_k - \mathbf{L}(\mathbf{C}\hat{\mathbf{x}}_k - \mathbf{C}\mathbf{x}_k) - \mathbf{A}\mathbf{x}_k - \mathbf{B}\mathbf{u}_k = (\mathbf{A} - \mathbf{LC})\boldsymbol{\epsilon}_k. \tag{2.42}$$

This expression gives a condition to tune correctly the matrix $\mathbf{L}$. To ensure that the error $\boldsymbol{\epsilon}_k$ converges to zero over time, meaning that the estimated state $\hat{\mathbf{x}}_k$ converges toward the true state $\mathbf{x}_k$, the concept of Schur matrices need to be introduced.

Definition 17. *A Schur matrix is a square matrix whose eigenvalues all lie within the open unit disk in the complex plane. In other words, a matrix $\mathbf{A}$ is Schur if and only if all eigenvalues λ of $\mathbf{A}$ satisfy $|\lambda| < 1$.*

By choosing $\mathbf{L}$ such that $(\mathbf{A} - \mathbf{LC})$ is a Schur matrix, $\boldsymbol{\epsilon}_k$ is asymptotically stable, so $\lim_{k \to +\infty} \boldsymbol{\epsilon}_k = 0$, and then $\hat{\mathbf{x}}_k$ converges toward $\mathbf{x}_k$.

Example 11. *Suppose the system of state $x_k \in \mathbb{R}$, of input $u_k \in \mathbb{R}$ and of output $y_k \in \mathbb{R}$ defined by the following dynamics*

$$\begin{cases} x_{k+1} = 0.99x_k + 0.07u_k \\ y_k = 0.5x_k \end{cases}.$$

Introduce the estimated state $\hat{x}_k$ defined by Equation (2.41). By choosing $L = 0.08$, $|A - LC| = 0.95 < 1$ and meets the stability criterion. The system is initialized with $x_0 = 42$ and $\hat{x} = 0$, and the simulation of the system with an input $u_k = sin(k/25)$ shows a convergence of the estimated state $\hat{x}_k$ toward the state of the system x_k, as shown in Figure 2.7.

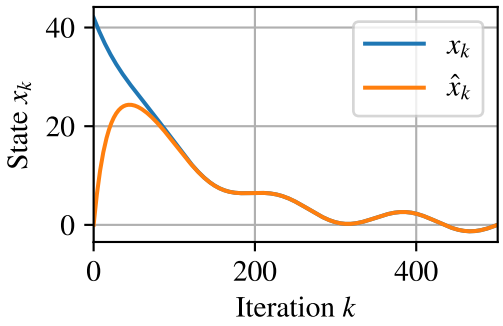


Figure 2.7 Luenberger state observer example

2.7 Stability of dynamical systems

This section details the tools and methods used to analyze and prove the stability of dynamical systems. Stability analysis is a crucial step in control system design, as it ensures that the system behaves as expected under various conditions. Stability guarantees that the system's state converges to a desired equilibrium point or trajectory, even in the presence of disturbances or uncertainties. These tools are fundamental to prove the stability of the cycle navigation, which is detailed in Chapter 6

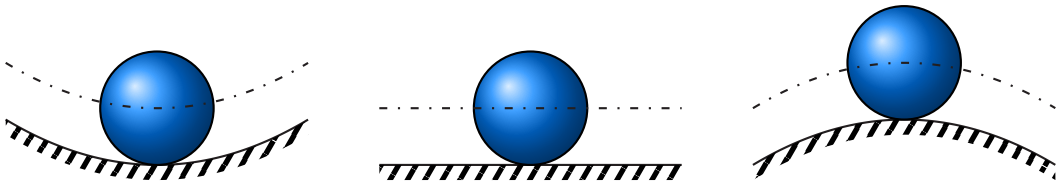
There are different tools which can be drawn from literature to prove the stability of a system. A classical tool for proving the stability of a dynamical system is the use of Lyapunov methods. This method is widely used to prove the stability of a system in the neighborhood of an equilibrium state. Another way to prove the stability of a dynamical system is to use set methods to find a positive invariant set around the equilibrium state. This is a set of states that will be captured forever if the system reaches it. This stability differs from the Lyapunov stability, as the system state could still evolve, but it will remain in the positive invariant set.

System stability is the study of a system's ability to remain stable under small disturbances around an equilibrium state [54, 45, 85].

2.7.1 Concepts of equilibrium point and stability

The equilibrium points can be of different natures. Figure 2.8 shows an example of three balls placed on a support. These three cases are equilibrium positions for the ball. However, the study of the trajectory of these three balls under a small perturbation will not lead to the same results. In Figure 2.8a, the ball will return to its equilibrium position under a small perturbation, while in Figure 2.8c the position of the ball will leave the equilibrium state. Figure 2.8b shows a case of marginal stability as under a bounded perturbation the ball will end up in another marginally stable position if some energy is lost in the friction between the ball and the support. This marginally stable case is the limit case between the stability and the instability of this system.

The nature of the stability can then be different depending on the behavior of the system. The stability can be defined as the evolution of the distance between the equilibrium state $\mathbf{x}_e$ and the state $\mathbf{x}$ [54]



(a) Stable case **(b)** Marginally stable case **(c)** Unstable case

Figure 2.8 Stability types on the ball example

2.7.2 Stability of continuous systems

Stability of a continuous dynamical system is defined by its behavior around an equilibrium state $\mathbf{x}_e$ under a zero input $\mathbf{u}(t) = \mathbf{0}_n$ [54]. To have a stable behavior, the state should remain bounded around the equilibrium state.

Definition 18. *A continuous system is **stable** on an equilibrium state $\mathbf{x}_e$ under a zero input ($\forall t \in \mathbb{R}, \mathbf{u}(t) = \mathbf{0}_n$), if and only if*

$$\forall \epsilon \in \mathbb{R}, \exists \delta \in \mathbb{R}, ||\mathbf{x}(0) - \mathbf{x}_e|| < \delta \Leftrightarrow \forall t \in \mathbb{R}, ||\mathbf{x}(t) - \mathbf{x}_e|| < \epsilon. \tag{2.43}$$

In the case the state of the system converges to the equilibrium state, the system is asymptotically stable.

Definition 19. *A continuous system is **asymptotically stable** on an equilibrium state $\mathbf{x}_e$ under a zero input ($\forall t \in \mathbb{R}, \mathbf{u}(t) = 0$), if and only if*

$$\lim_{t \to +\infty} ||\mathbf{x}_e - \mathbf{x}(t)|| = 0. \tag{2.44}$$

For linear continuous dynamical systems, it is possible to determine the stability of the system by studying the eigenvalues of the matrix $\mathbf{A}$ [54].

Theorem 1. *A continuous linear system defined by Equation (2.20) is asymptotically stable under a zero input, if and only if the matrix $\mathbf{A}$ is Hurwitz.*

Proof. Suppose the system at the finite initial state $\mathbf{x}_0 \in \mathbb{R}^n$. After a duration t, $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0$. As $\mathbf{A}$ is Hurwitz, its eigenvalues have negative real parts, and $\lim_{t \to +\infty} e^{\mathbf{A}t} = 0$. It comes

$$\lim_{t \to +\infty} e^{\mathbf{A}t} = 0 \Leftrightarrow \lim_{t \to +\infty} \mathbf{x}(t) = 0. \tag{2.45}$$

□

Example 12. *Consider the linear continuous system defined by the following state-space representation*

$$\begin{cases} \dot{\mathbf{x}} = \begin{bmatrix} -2 & 1 \\ -1 & -3 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{u} \\ \mathbf{y} = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x} \end{cases}. \tag{2.46}$$

The matrix $\mathbf{A} = \begin{bmatrix} -2 & 1 \\ -1 & -3 \end{bmatrix}$ has eigenvalues $\lambda_1 = \frac{-5-i\sqrt{3}}{2}$ and $\lambda_2 = \frac{-5+i\sqrt{3}}{2}$, both of which have negative real parts. Therefore, the system is asymptotically stable.

Figure 2.9 shows the simulation of the system with an initial condition $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$ under a zero input. The state converges toward the equilibrium state $\mathbf{x}_e = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$, which highlights the stability of the system.

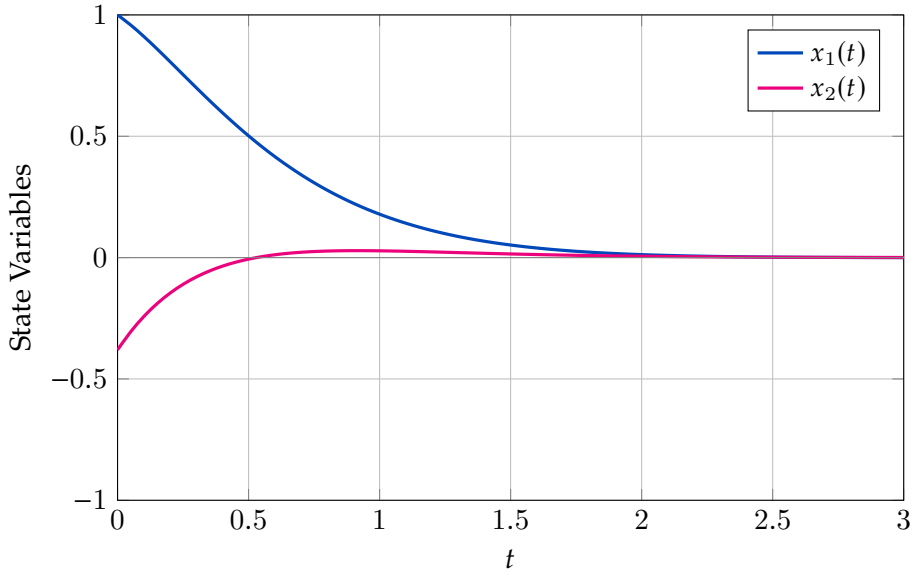


Figure 2.9 Simulation of the stable continuous system

2.7.3 Stability of discrete systems

Stability of a discrete dynamical system is defined by its behavior around an equilibrium state $\mathbf{x}_e$ under a zero input $\mathbf{u}_k = \mathbf{0}_n$ [54]. To have a stable behavior, the state of the cycle should be bounded around the equilibrium state.

Definition 20. *A discrete system is stable on an equilibrium state $\mathbf{x}_e$ under a zero input ($\forall k \in \mathbb{N}, \mathbf{u}_k = 0$), if and only if*

$$\forall \epsilon \in \mathbb{R}, \exists \delta \in \mathbb{R}, ||\mathbf{x}_0 - \mathbf{x}_e|| < \delta \Leftrightarrow \forall k \in \mathbb{N}, ||\mathbf{x}_k - \mathbf{x}_e|| < \epsilon. \tag{2.47}$$

In the case the state of the system converges to the equilibrium state, the system is asymptotically stable.

Definition 21. *A discrete system is asymptotically stable on an equilibrium state $\mathbf{x}_e$ under a zero input ($\forall k \in \mathbb{N}, \mathbf{u}_k = 0$), if and only if*

$$\lim_{k \to +\infty} ||\mathbf{x}_e - \mathbf{x}_k|| = 0. \tag{2.48}$$

For linear discrete dynamical systems, it is possible to determine the stability of the system by studying the eigenvalues of the matrix $\mathbf{A}$ [54].

Theorem 2. *A discrete linear system defined by Equation (2.29) is asymptotically internally stable if and only if the matrix $\mathbf{A}$ is Schur*

Proof. Suppose the system at the finite initial state $\mathbf{x}_0 \in \mathbb{R}^n$. After k iterations of the system under a null input $\mathbf{u}_k = 0$, $\mathbf{x}_k = \mathbf{A}^k \mathbf{x}_0$. As $\mathbf{A}$ is Schur, then its eigenvalues belong to the unit circle, and $\lim_{k \to +\infty} \mathbf{A}^k = 0$. It comes

$$\lim_{k \to +\infty} \mathbf{A}^k = 0 \Leftrightarrow \lim_{k \to +\infty} \mathbf{x}_k = 0. \tag{2.49}$$

□

Example 13. *Consider the linear discrete system defined by the following state-space representation*

$$\begin{cases} \mathbf{x}_{k+1} = \begin{bmatrix} 0.6 & 0.2 \\ 0.1 & 0.7 \end{bmatrix} \mathbf{x}_k + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{u}_k \\ \mathbf{y}_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}_k \end{cases}. \tag{2.50}$$

The matrix $\mathbf{A}$ *has eigenvalues* $\lambda_1 = 0.8$ *and* $\lambda_2 = 0.5$*, both of which lie within the open unit disk* ($|\lambda_i| < 1$)*. Therefore, the system is asymptotically stable.*

Figure 2.10 shows the simulation of the system with an initial condition $\mathbf{x}_0 = \begin{bmatrix} 1 & -0.5 \end{bmatrix}^T$ *under a zero input. The state converges toward the equilibrium state* $\mathbf{x}_e = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$*, which highlights the stability of the system.*

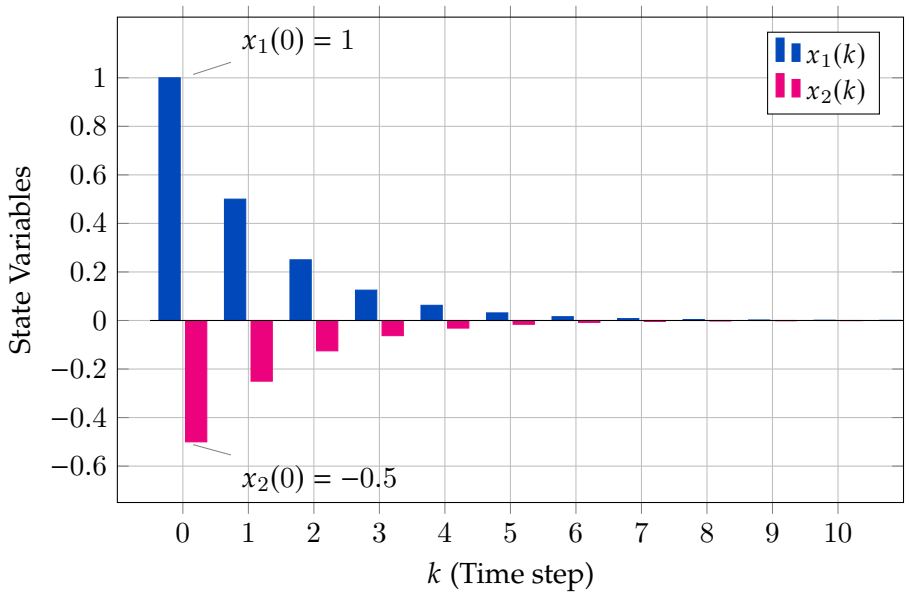


Figure 2.10 Simulation of the stable discrete system

These concepts of stability are fundamental to understand the behavior of a dynamical system, and will be used in the next chapters to first design a stable system composed of a timed automaton and a robot in Chapter 5, and then to prove the stability of the cycle navigation in Chapter 6.

2.7.4 Lyapunov methods for nonlinear systems stability

Lyapunov methods are powerful tools for proving the stability of non-linear dynamical systems [45, 54]. The main idea is to find a Lyapunov function which acts as an energy function of the system, and to study its evolution over time. This function should be positive, and its derivative should be strictly negative everywhere except at the equilibrium point. In this way, the system dynamics leads the state of the system to the equilibrium point, and the system is stable.

Definition 22. *A Lyapunov function* $V : \mathcal{S} \to \mathbb{R}$ *for a system of equilibrium point* $\mathbf{x}_e$ *is a continuously differentiable function that satisfies*

(i) $V(\mathbf{x}_e) = 0$,
(ii) $\forall \mathbf{x} \in \mathcal{S} \setminus \{\mathbf{x}_e\}, V(\mathbf{x}) > 0$,
(iii) $\forall \mathbf{x} \in \mathcal{S}, \dot{V}(\mathbf{x}) \leq 0$.

If the three conditions of Definition 22 are satisfied, then the system is stable around the equilibrium point $\mathbf{x}_e$. Moreover, if $\dot{V}(\mathbf{x}) < 0$ for all $\mathbf{x} \neq \mathbf{x}_e$, then the system is asymptotically stable.

Example 14. *Consider a non-linear pendulum of angle θ and of angular velocity $\dot{\theta}$, subject to gravity of intensity g and with a length l. The dynamics of the system is given by*

$$\ddot{\theta} = -\frac{g}{l}\sin(\theta). \tag{2.51}$$

A good candidate for the Lyapunov function is the mechanical energy of the pendulum. The energy of the pendulum $V(\theta, \dot{\theta}) = \frac{1}{2}\dot{\theta}^2 + \frac{g}{l}(1 - \cos(\theta))$ is a function which is always positive, and of negative derivative. Indeed

(i) $V(0, 0) = 0$,
(ii) For all $(\theta, \dot{\theta}) \neq (0, 0)$, $V(\theta, \dot{\theta}) > 0$,
(iii) $\dot{V}(\theta, \dot{\theta}) = \dot{\theta}\ddot{\theta} + \frac{g}{l}\sin(\theta)\dot{\theta} = 0$.

Therefore, the pendulum is stable around its equilibrium point $(0, 0)$.

Remark. *As the derivative of the Lyapunov function is equal to zero, the system is marginally stable. This means that the system will not converge to the equilibrium point, but will stay in its neighborhood. This is explained as there is no dissipation in the system, and the energy of the system is conserved, and the pendulum will oscillate indefinitely around* $\mathbf{x}_e$.

2.7.5 Set methods for non-linear systems stability

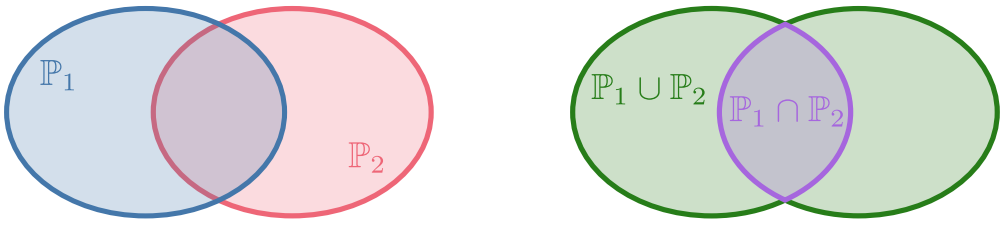
Lyapunov methods are able to prove the stability of a dynamical system around an equilibrium point in the neighborhood of this point. However, these methods are not able to prove the stability of a system far from the equilibrium point, or in presence of disturbances. To prove the stability of a system in these conditions, set-based methods are commonly used.

Lattice structure of sets

The set of all subsets of a given set $\mathcal{S}$ is denoted by $\mathcal{P}(\mathcal{S})$. This set has a lattice structure, which means that it is possible to define a partial order relation on this set. The inclusion relation $\subseteq$ is a partial order relation on $\mathcal{P}(\mathcal{S})$, and the largest element is $\mathcal{S}$, while the smallest element is the empty set $\emptyset$.

Definition 23. *A **lattice** is a partially ordered set in which every two elements have a unique supremum (least upper bound) and an infimum (greatest lower bound).*

Figure 2.11 shows the lattice structure of two positive invariant sets $\mathbb{P}_1$ and $\mathbb{P}_2$. The meet (infimum) of these two sets is their intersection $\mathbb{P}_1 \cap \mathbb{P}_2$, while the join (supremum) is their union $\mathbb{P}_1 \cup \mathbb{P}_2$.



(a) Two positive invariant sets **(b)** Meet and join

Figure 2.11 Lattice structure of positive invariant sets

The notion of positive invariant set will be introduced hereafter. This lattice structure of sets prove the existence of a largest positive invariant set contained in a given set $\mathbb{P}_0$, and

the smallest positive invariant set containing a given set $\mathbb{P}_0$. This concept will be usefull to characterize the largest positive invariant set contained in a set $\mathbb{P}_0$.

Positive invariant set

A positive invariant set $\mathbb{P}$ is a set stable by application of the evolution equation of the dynamical system $\mathbf{f}$ [5, 82, 9].

Definition 24. *A **positive invariant set** $\mathbb{P}$ for a dynamical system is a set of states $\mathbf{x}$ stable under the evolution function $\mathbf{f}$ of the system. In other words, $\mathbf{x} \in \mathbb{P} \Rightarrow \mathbf{f}(\mathbf{x}) \in \mathbb{P}$, or $\mathbf{f}(\mathbb{P}) \subseteq \mathbb{P}$.*

For a given dynamical system, there could be many positive invariant sets. However, as the positive invariant set has a lattice structure, the largest positive invariant set $\mathbb{P}$ contained in an initial set $\mathbb{P}_0$ can be characterized.

Theorem 3. *In $\mathbb{R}^n$, the largest positive invariant set is $\mathbb{R}^n$, and the smallest one is $\emptyset$.*

Proof. Whatever the evolution function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, the two following assertions holds:

(i) $\forall \mathbf{x} \in \mathbb{R}^n, \mathbf{f}(\mathbf{x}) \in \mathbb{R}^n \implies \mathbf{f}(\mathbb{R}^n) \subseteq \mathbb{R}^n$

(ii) $\mathbf{f}(\emptyset) = \emptyset \implies \mathbf{f}(\emptyset) \subseteq \emptyset$.

This prove that $\mathbb{R}^n$ and $\emptyset$ are positive invariant sets. As nothing is larger than $\mathbb{R}^n$ and nothing is smaller than $\emptyset$ respective to the inclusion relationship, these two sets are the largest and the smallest positive invariant sets. □

To compute a positive invariant set for a dynamical system [4, 5, 82], a sequence of sets $\mathbb{P}_k$ which will converge towards $\mathbb{P}$ is built. $\mathbb{P}_0$ is initialized to a set $\mathbb{X}_0$ around a supposed stable state, and $\mathbb{P}_{k+1}$ is computed as the intersection of $\mathbb{P}_k$ and $\mathbf{f}(\mathbb{P}_k)$ as follows

$$\begin{cases} \mathbb{P}_0 = \mathbb{X}_0 \\ \mathbb{P}_{k+1} = \mathbb{P}_k \cap \mathbf{f}(\mathbb{P}_k) \end{cases} \tag{2.52}$$

Therefore, each state which belongs to $\mathbb{P}_k$ and which is moved out of this set by the application of the system dynamics is removed of the solution for $\mathbb{P}_{k+1}$. Thus, the set $\mathbb{P}_k$ is iteratively contracted. Then, $\exists n \in \mathbb{N}$, such that $\forall k \in \mathbb{N}$, $\mathbf{f}^k(\mathbb{P}_n) \subseteq \mathbb{P}_n$. $\mathbb{P}_n$ is the largest capture basin contained in $\mathbb{P}_0$. The construction of this sequence is presented by Algorithm 1.

Algorithm 1 Positive invariant set computation

Input: $\mathbb{P}_0, \mathbf{f}$
Output: $\mathbb{P}_k$
$\mathbb{P}_k \leftarrow \mathbb{P}_0$
repeat
  $\mathbb{P}_k \leftarrow \mathbb{P}_k \cap \mathbf{f}(\mathbb{P}_k)$
until $\mathbb{P}_k \subseteq \mathbf{f}(\mathbb{P}_k)$

Figure 2.12 shows an example of a positive invariant set $\mathbb{P}$ for a dynamical system governed by the evolution function $\mathbf{f}$. The set $\mathbb{P}$ is stable by application of the evolution function $\mathbf{f}$, as the image of the set $\mathbb{P}$ by the function $\mathbf{f}$ is included in the set $\mathbb{P}$.

Remark. *As shown in Figure 2.12, the set $\mathbf{f}(\mathbb{P})$ is not positive invariant itself as $\mathbf{f}^2(\mathbb{P}) \not\subseteq \mathbf{f}(\mathbb{P})$. Therefore a subset of a positive invariant set is not necessarily a positive invariant set.*

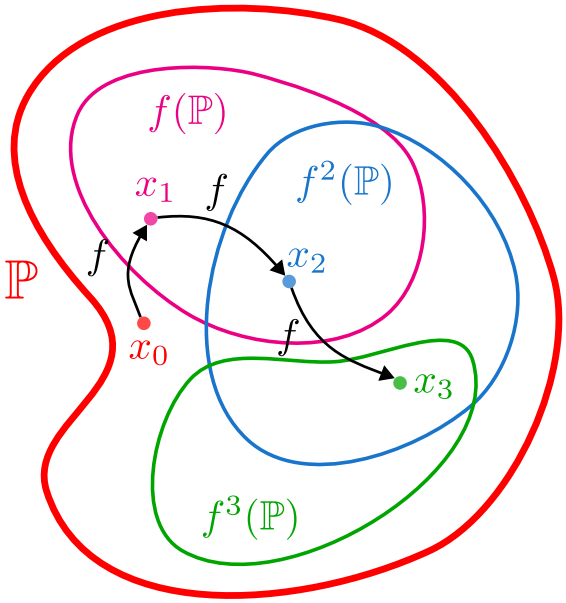


Figure 2.12 Positive invariant set

Capture basin

The capture basin of a dynamical system is the set of states which will converge to a positive invariant set $\mathbb{P}$ under the application of the system dynamics $\mathbf{f}$. The characterization of this capture basin plays a significant role in the stability analysis, as it allows to determine the set of states which will lead to a stable behavior.

The capture basin is a positive invariant set itself, as it is stable by application of the system dynamics $\mathbf{f}$. A possible definition of a capture basin, associated with a positive invariant set $\mathbb{P}$, is the largest positive invariant set $\mathbb{B}$ which enclose the positive invariant set $\mathbb{P}$, and which is not enclosing any other positive invariant set $\mathbb{P}'$ such that $\mathbb{P} \cap \mathbb{P}' = \emptyset$ [5, 82, 9]. This definition has the advantage to avoid the introduction of the trajectory of the dynamical system. However, as the robot trajectory is the topic of our study, we will use a more intuitive definition of the capture basin.

Definition 25. *A **capture basin** $\mathbb{B}$ of a dynamical system is a set of states $\mathbf{x}$ which will converge to a positive invariant set $\mathbb{P}$ under the application of the system dynamics $\mathbf{f}$. In other words,* $\forall \mathbf{x} \in \mathbb{B}, \exists k \in \mathbb{N}, \mathbf{f}^k(\mathbf{x}) \in \mathbb{P}$.

This definition seems more convenient to prove the stability of the cycle navigation, and will be an important element for the Chapter 6.

To characterize the capture basin of a dynamical system, a sequence of sets $\mathbb{B}_k$ which will converge towards the capture basin $\mathbb{B}$ is built [4, 5, 82]. This sequence is initialized to a positive invariant set $\mathbb{P}$, and $\mathbb{B}_{k+1}$ is computed as the union of $\mathbb{B}_k$ and the pre-image of $\mathbb{B}_k$ by the application of the system dynamics $\mathbf{f}$ as follows

$$\begin{cases} \mathbb{B}_0 = \mathbb{P} \\ \mathbb{B}_{k+1} = \mathbb{B}_k \cup \mathbf{f}^{-1}(\mathbb{B}_k) \end{cases} \tag{2.53}$$

Computing in this way this sequence ensures that all the states that reach the set $\mathbb{B}_{k-1}$ in one step by following the system dynamic are added to the set $\mathbb{B}_k$. This process is iterated until the pre-image of the set $\mathbb{B}_k$ by the application of the system dynamics $\mathbf{f}$ is included in the set $\mathbb{B}_k$. At this point, no new state can be added to the set $\mathbb{B}_k$, and the

capture basin is fully characterized. The construction of this sequence is presented by Algorithm 2.

Algorithm 2 Capture basin computation

Input: $\mathbb{P}, \mathbf{f}$
Output: $\mathbb{B}_k$
$\mathbb{B}_k \leftarrow \mathbb{P}$
repeat
  $\mathbb{B}_k \leftarrow \mathbb{B}_k \cup \mathbf{f}^{-1}(\mathbb{B}_k)$
until $\mathbf{f}^{-1}(\mathbb{B}_k) \subseteq \mathbb{B}_k$

Remark. *In contrast to the computation of the positive invariant set, the computation of the capture basin could be stopped at any iteration. The result will remain guaranteed, but there will be states which lead to the provided positive invariant set $\mathbb{P}$, and then present a stable behavior, which will not be included in the computed capture basin.*

Figure 6.7 illustrate the computation of a capture basin related to a positive invariant set $\mathbb{P}$ presented in Algorithm 2. Starting from the positive invariant set $\mathbb{P}$, the pre-image of this set by the application of the system dynamics $\mathbf{f}$ is computed. The union of this pre-image and the positive invariant set $\mathbb{P}$ gives a new set $\mathbb{B}_1$, and so on. Then, a state in the set $\mathbb{B}_k$ will reach the positive invariant set $\mathbb{P}$ in at most k iterations.

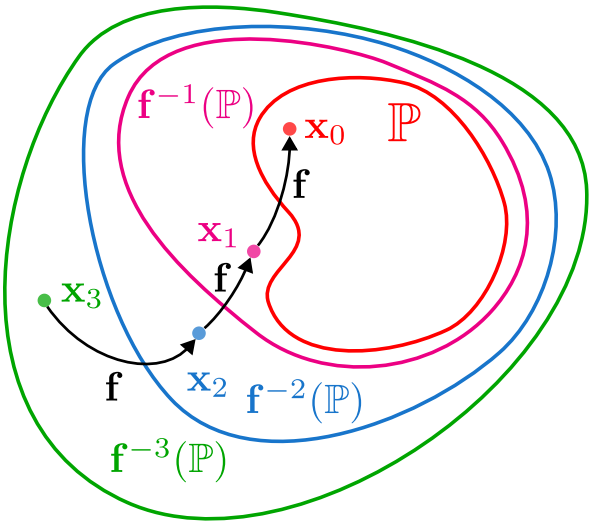


Figure 2.13 Capture basin

2.8 Conclusion

This chapter has established the mathematical foundations of dynamical systems theory that underpin modern robotic modeling and control. The framework presented here serves as the conceptual backbone for understanding how robotic systems evolve in time and how we can influence their behavior through inputs.

The distinction between continuous-time and discrete-time formulations reflects a fundamental duality in robotics. Continuous-time models naturally capture the physics

of mechanical systems, where forces, torques, and accelerations govern motion through differential equations. These models provide intuitive understanding of system behavior and enable elegant analytical techniques rooted in classical mechanics. Conversely, discrete-time formulations align with the digital nature of modern control implementations, where sensors sample at finite rates and actuators receive updated commands at discrete times. The choice between these representations depends on the specific application context, computational constraints, and desired fidelity to either the underlying physics or the implementation reality.

The concepts of controllability and observability emerge as fundamental system properties that determine what is theoretically possible in terms of control design and state estimation. Controllability establishes whether a system can be steered to arbitrary states through appropriate control inputs, while observability determines whether the complete system state can be reconstructed from available measurements. These dual properties provide the theoretical foundation for assessing system design choices, sensor placement strategies, and the feasibility of control objectives. In robotics, where systems often exhibit complex dynamics and operate under various constraints, these concepts guide the selection of actuator configurations and sensing architectures.

When complete state information is not directly measurable, state observers provide a principled approach to state estimation. The development of observer-based control strategies acknowledges the practical reality that many critical system variables cannot be directly sensed, whether due to physical limitations, cost constraints, or measurement noise. The theoretical guarantees provided by observer design ensure that estimated states converge to true states under appropriate conditions, enabling the implementation of sophisticated control strategies even with limited sensing capabilities.

System stability analysis represents perhaps the most critical aspect of dynamical systems theory for robotics applications. Classical Lyapunov theory provides powerful tools for establishing stability through energy-like functions, offering both local and global stability guarantees under appropriate conditions. The geometric interpretation of Lyapunov functions as generalized energy surfaces provides intuitive understanding of system behavior and guides controller design. Modern set-based approaches extend these classical results by explicitly handling uncertainties, disturbances, and constraints that are inherent in real robotic systems. These methods acknowledge that practical systems operate within bounded regions of state space and must satisfy physical limitations on actuator capabilities and environmental constraints.

The theoretical tools developed in this chapter find direct application in the navigation strategies explored in subsequent chapters. The concept of stable cycles, rooted in dynamical systems theory, provides a natural framework for generating periodic motions in robotic systems. Understanding how dynamical systems can exhibit stable oscillatory behavior enables the design of robust navigation algorithms that maintain desired trajectories while adapting to environmental variations and system uncertainties. The stability analysis techniques presented here ensure that such cyclic behaviors remain bounded and converge to desired patterns despite perturbations.

3 Automata theory

3.1 Introduction

A timed automaton is a mathematical model used to describe and analyze systems where timing constraints are crucial. It extends classical finite-state automata by introducing clocks—continuous variables that track the passage of time. These clocks can be reset at transitions, and states or transitions can be constrained by timing conditions called clock guards. This framework allows for precise modeling of time-dependent behaviors in various systems.

Timed automata were introduced by Alur and Dill in the early 1990s as a formalism for verifying properties of real-time systems [2]. They are particularly useful in expressing temporal requirements such as deadlines, delays, and synchronization constraints, which are often critical in software and hardware verification.

In this manuscript, timed automata play a crucial role in the design of the cycle navigation. A timed automaton will be used to generate inputs that the robot will follow to achieve a cyclic trajectory. In Chapter 5, transition durations between states of the timed automaton will even be tuned to control the movement of the cycle described by the robot trajectory in the state space.

Timed automata were formalized as an extension of finite-state automata [48] with a set of real-valued clocks. These clocks represent the elapsed time since their reset, and transitions between states are triggered by clock constraints [7, 2]. This framework allows for precise modeling of time-dependent behaviors in various systems.

Real-time systems, such as embedded controllers, industrial automation, and communication protocols, must operate under strict timing constraints. Any deviation from expected timing can lead to failures or degraded performance. Timed automata provide a rigorous way to analyze these systems and ensure they meet required specifications.

For example, in scheduling problems, timed automata can verify whether a system meets deadlines by modeling the execution of tasks and their timing constraints. Similarly, they are used in protocol verification to ensure that communication messages are sent and received within predefined time windows, avoiding synchronization issues.

Timed automata have found significant applications in robotics, particularly in motion planning [95], task scheduling [83], and control synthesis [3]. In autonomous systems, precise timing is essential for coordination, decision-making, and interaction with dynamic environments.

In multi-robot systems, timed automata can model the synchronization of robots performing cooperative tasks, ensuring that actions are executed within correct time bounds. For example, in warehouse automation, robots transporting goods must coordinate their movements to avoid collisions and optimize delivery times.

3.2 Finite state automaton

3.2.1 General definition

A finite-state automaton is a mathematical model used to represent systems with a finite number of states and transitions triggered by input symbols. It is a fundamental concept in automata theory and formal languages, providing a simple yet powerful framework for modeling various systems. Finite state automata are widely used in computer science, engineering, and robotics to describe discrete behaviors and control logic.

Finite state automata can be used in robotics to control a mission through a sequence of actions triggered by environmental events or user commands. For example, a robot can navigate a maze by following a Finite State Automaton describing its behavior to move between different locations based on measurements.

Definition 26. *A **finite state automaton** [2] is a tuple* $\mathcal{A} = (\Sigma, Q, Q_0, F, E)$ *where:*

- Σ *is a finite set of symbols called an alphabet.*
- Q *is a finite set of states.*
- $Q_0 \subseteq Q$ *is the set of initial states.*
- $F \subseteq Q$ *is the set of accepting states.*
- $E \subseteq Q^2 \times \Sigma$ *is the set of named edges between two states called transitions.*

Remark. *Accepting states are used to determine whether a sequence of input symbols leads to a valid state. This is particularly useful for recognizing patterns or sequences when processing languages or signals. For most of the robotics examples where the system follows a Finite State Automaton, the set of accepting states is empty, as the robot repeats indefinitely the same sequence of actions. Some accepting states could be used for error handling or to end a mission.*

Example 15. *An acceptor [48] is a finite state machine able to check if an input verifies a condition. Suppose an acceptor that detects whether an input binary number has an even number of* 1*s. The automaton has two states:* q_0 *and* q_1*. The initial state is* q_0*, and this state is also an accepting state. The alphabet is* $\Sigma = \{0, 1\}$*, and the set of transitions is defined as follows:*

- $\langle q_0, q_0, 0\rangle$*: stay in the same state on input 0.*
- $\langle q_0, q_1, 1\rangle$*: transition to state* q_1 *on input 1.*
- $\langle q_1, q_1, 0\rangle$*: stay in the same state on input 0.*
- $\langle q_1, q_0, 1\rangle$*: transition to state* q_0 *on input 1.*

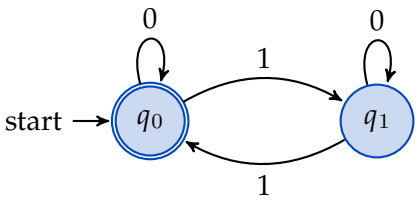


Figure 3.1 Finite state automaton for even number of 1s

If the input is $b_0 = 1010$*, the automaton will follow the transitions* $q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_0$ *and will end in the accepting state* q_0*. The input* $b_1 = 1011$ *will follow the transitions* $q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_0 \xrightarrow{1} q_1$ *and will end in the non-accepting state* q_1*. This automaton correctly accepts inputs with an even number of* 1*s.*

3.2.2 Deterministic Finite Automaton

Deterministic finite automata are a special case of Finite State Automata where each state has exactly one transition for each symbol in the alphabet. This property ensures that the automaton is deterministic, meaning that it can only be in one state at a time for a given input symbol. DFAs are particularly useful for modeling systems with well-defined behaviors and unambiguous transitions.

Definition 27. *A* ***deterministic finite automaton*** *[29] is a Finite State Automaton* $\mathcal{A} = (\Sigma, Q, Q_0, F, E)$ *that is deterministic, and meets the following conditions:*

- $Q_0 = \{q_0\}$ *is the single initial state.*
- $\forall (q_0, q_1, \sigma) \in Q^2 \times \Sigma, \exists! \langle q_0, q_1, \sigma\rangle \in E.$

Example 16. *Suppose a turnstile is modeled as a finite state automaton with two states: locked and unlocked. The turnstile accepts a coin as input to transition from the locked to the unlocked state. If no coin is inserted, the turnstile remains locked. This simple model can be represented as a deterministic finite automaton with the following components:*

- $\Sigma = \{coin, push\}$*: input alphabet.*

- $Q = \{closed, opened\}$: *states of the turnstile.*
- $Q_0 = \{closed\}$: *the initial state.*
- $F = \emptyset$: *accepting state.*
- $\langle closed, opened, coin \rangle$: *transition from closed to opened state.*
- $\langle opened, closed, push \rangle$: *transition from opened to closed state.*
- $\langle closed, closed, push \rangle$: *self-loop on closed state.*
- $\langle opened, opened, coin \rangle$: *self-loop on opened state*

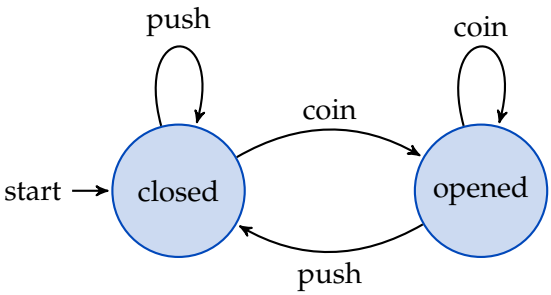


Figure 3.2 Finite state automaton for a coin-turnstile

3.3 Timed automaton

3.3.1 General definition

To deal with systems where timing constraints are crucial, transitions need to be triggered by clocks, in contrast to classical automata. This is the main difference between timed automata and classic automata. Clocks are real-valued variables that continuously increase over time and can be reset to zero during transitions.

Definition 28. *A **clock constraint** [2] is a condition expressed in the form $c \diamond n$, where:*

- *c is a clock*
- *$\diamond$ is a comparison operator ($<, \leq, =, \geq, >$)*
- *$n \in \mathbb{N}$ is a constant*

Using the definition of clock constraints, it is possible to formalize the definition of a timed automaton by knowing that transitions are triggered by clock constraints.

Definition 29. *A **timed automaton** [2] is a tuple $\mathcal{A} = \langle \Sigma, Q, Q_0, C, F, E \rangle$, where:*

- *Σ is a finite set of symbols called an alphabet*
- *Q is a finite set of states*
- *$Q_0 \subseteq Q$ is the set of initial states*
- *C is a finite set of clocks*
- *$F \subseteq Q$ is the set of accepting states*
- *$E \subseteq Q \times Q \times \Sigma \times \mathcal{B}(C) \times \mathcal{P}(C)$ is the set of edges called transitions where:*
  - *$\mathcal{B}(C)$ is the set of clock constraints*
  - *$\mathcal{P}(C)$ is the powerset of clocks*

An edge $e = \langle q_0, q_1, \sigma, c, r \rangle$ from E is a transition from states q_0 to q_1, with a label σ, a set of clock constraints c, and a set r of clocks to be reset.

Example 17. *Figure 3.3 shows an example of a simple timed automaton over the unary alphabet $\Sigma = \{\sigma\}$. The clock c is ticking continuously, and is automatically reset when its value reaches 3.*

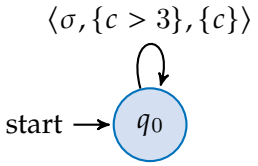


Figure 3.3 Example of timed automaton counting up to 3

3.3.2 Deterministic timed automaton

Timed automata can also have additional properties. For instance, the execution of the timed automaton can be deterministic if it meets some conditions [2]. This kind of timed automaton is relevant for robotics applications, as it is essential for robots to behave in a deterministic way.

Definition 30. *A timed automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, C, F, E \rangle$ is **deterministic** if and only if*

- *$Q = \{q_0\}$ is the only initial state*
- *$\forall (q, \sigma) \in Q \times \Sigma$, edges $e_0 = \langle q, -, \sigma, -, c_0 \rangle$ and $e_1 = \langle q, -, \sigma, -, c_1 \rangle$, the clock constraints are mutually exclusive.*

Example 18. *Figure 3.4 shows an example of a timed automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, C, F, E \rangle$.*
The timed automaton is defined over the alphabet $\Sigma = \{\sigma_0, \sigma_1\}$, and uses one clock $C = \{c\}$. The set of states is $Q = \{q_0\}$, and the initial state is q_0. The set of edges is $E = \{\langle q_0, q_0, \sigma_0, \{c > 5\}, \{c\} \rangle, \langle q_0, q_0, \sigma_1, \{c \leq 5\}, \emptyset \rangle\}$.
The set of clock valuations is well disjoint as the conditions $(c > 5)$ and $(c \leq 5)$ are never true simultaneously. Furthermore, the timed automaton has only one initial state. As a result this timed automaton is deterministic.

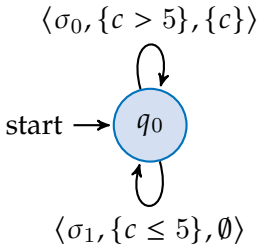


Figure 3.4 Example of deterministic timed automaton

3.3.3 Cyclic timed automaton

Cyclic timed automaton are introduced in this work. It is a subset of deterministic timed automaton which is appropriate to robotics applications. Robots following a cyclic timed automaton are able to repeatedly follow a succession of actions, always in the same order and in a deterministic way. This is particularly true for industrial robots.

Definition 31. *A timed automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, C, F, E \rangle$ is cyclic if and only if there is exactly one incoming and one outgoing edge for each state. In other words, if the timed automaton satisfies the condition*

$$\forall q_i \in Q, \exists! (q_j, \sigma, b, p) \in Q \times \Sigma \times \mathcal{B}(C) \times \mathcal{P}(C), \quad e = \langle q_i, q_j, \sigma, b, c \rangle \in E. \tag{3.1}$$

Example 19. *Consider a traffic light system. The system has three states: red, green, and yellow. The system starts in the red state, then transitions to green after a fixed time, and finally to yellow before returning to red. The timed automaton for this system is cyclic, as it follows a fixed sequence of states and transitions. The automaton can be defined as follows:*

- *$\Sigma = \{to_{green}, to_{yellow}, to_{red}\}$: the input alphabet.*
- *$Q = \{red, green, yellow\}$: the states of the traffic light.*

- $Q_0 = \{red\}$: *the initial state.*
- $F = \emptyset$: *no accepting states.*
- $\langle red, green, to_{green}, \{c \geq 30\}, \{c\}\rangle$: *transition from red to green after* 30 *seconds.*
- $\langle green, yellow, to_{yellow}, \{c \geq 30\}, \{c\}\rangle$: *transition from green to yellow after* 30 *seconds.*
- $\langle yellow, red, to_{red}, \{c \geq 5\}, \{c\}\rangle$: *transition from yellow to red after* 5 *seconds.*

The traffic light system follows a cyclic pattern of red, green, yellow states, with fixed time intervals between transitions. This timed automaton ensures that the traffic light operates in a deterministic and repeatable manner.

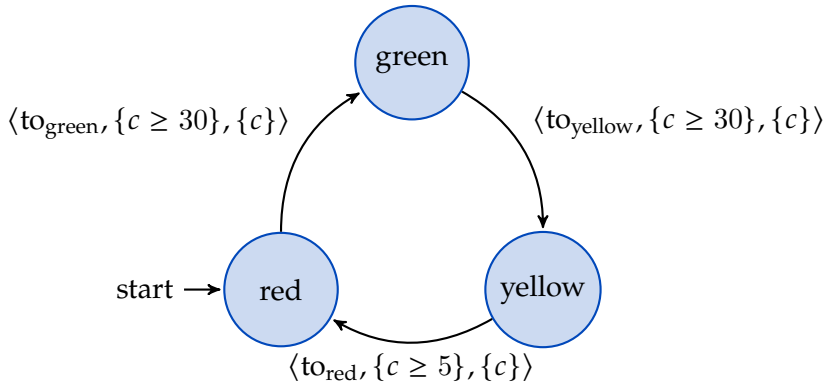


Figure 3.5 Finite state automaton for traffic light system

This cyclic timed automaton can be relevant for controlling a robot through a sequence of actions in a deterministic way. This will happen in the next chapters, particularly in Chapter 5 where the timed automaton will be used to generate cyclic trajectories for the robot.

Example 20. *Consider a cyclic timed automaton defined as follows:*

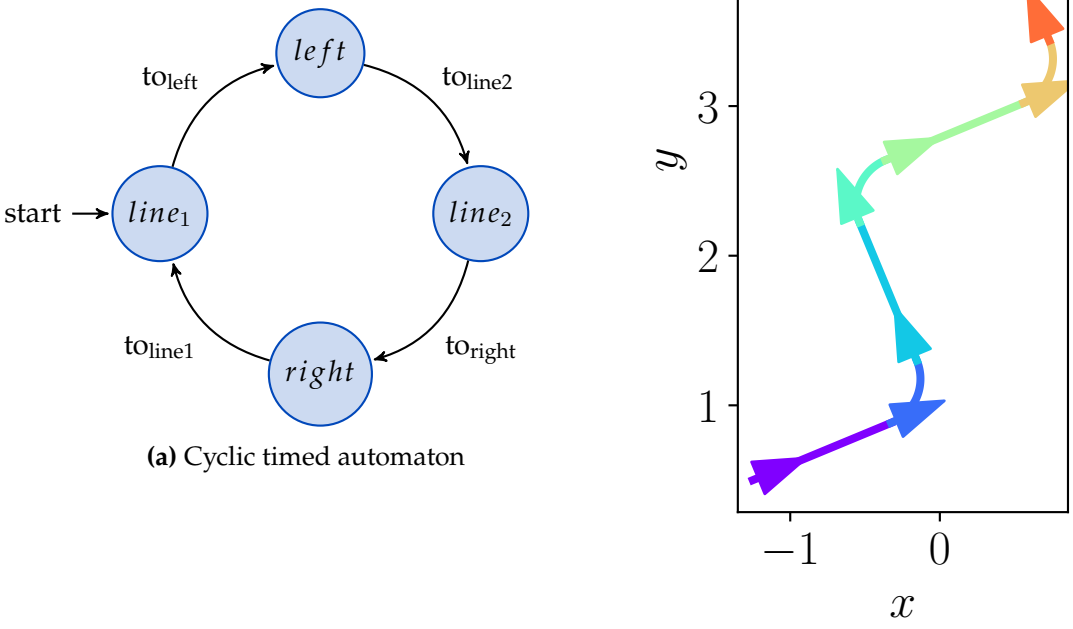
- $\Sigma = \{to_{turn}, to_{line}\}$ *the input alphabet.*
- $Q = \{line_1, left, line_2, right\}$ *the states of the automaton.*
- $Q_0 = \{line_1\}$ *the initial state.*
- $F = \emptyset$ *no accepting states.*
- $\langle line_1, left, to_{left}, \{c \geq 1\}, \{c\}\rangle$: *transition from* $line_1$ *to left after* 1 *second.*
- $\langle left, line_2, to_{line2}, \{c \geq \frac{\pi}{2}\}, \{c\}\rangle$: *transition from left to* $line_2$ *after* $\frac{\pi}{2}$ *seconds.*
- $\langle line_2, right, to_{right}, \{c \geq 1\}, \{c\}\rangle$: *transition from* $line_2$ *to right after* 1 *second.*
- $\langle right, line_1, to_{line1}, \{c \geq \frac{\pi}{2}\}, \{c\}\rangle$: *transition from right to* $line_1$ *after* $\frac{\pi}{2}$ *seconds.*

In the state line, the automaton will wait for 1 *second before transitioning to the state turn. In the state turn, the automaton will wait for* $\frac{\pi}{2}$ *seconds before transitioning back to the state line. This cyclic timed automaton will generate inputs for the robot. The velocity is constant and set to* $v = 1$*. The angular velocity is set to* $\omega = 0$ *in the state* $line_1$ *and* $line_2$*,* $\omega = 1$ *in the state left, and* $\omega = -1$ *in the state right. The robot will move* 1 *meter in the state line and turn* $\frac{\pi}{2}$ *radians in the state turn.*

Figure 3.6a shows the cyclic timed automaton. The robot will follow the trajectory shown in Figure 3.6b.

3.4 Conclusion

This chapter has established the mathematical foundations of timed automata and their application to robotics. We began with classical finite state automata and extended them to handle timing constraints by introducing clocks and timing conditions. This framework allows us to model systems where time plays a crucial role. This special case of automata is particularly relevant in robotics, where robots follow a sequence of actions deterministically, often repeating the same cyclic behaviors.



(a) Cyclic timed automaton

(b) Unicycle trajectory

Figure 3.6 Trajectory of the unicycle controlled by the cyclic timed automaton

We have shown that timed automata can effectively model time-dependent behaviors in various systems, from simple binary checkers to complex traffic light controllers. The examples demonstrate how these mathematical tools can represent both discrete events and continuous time processes, making them versatile for system design.

Deterministic timed automata presented here ensure predictable robot behavior, which is critical for cycle navigation, as the trajectory of the robot guided by the timed automaton should be repeatable and reliable. This theoretical foundation directly supports the cycle navigation approach developed in the following chapters, especially the formalism of the cycle navigation presented in Chapter 5. The properties of cyclic timed automata will be used to generate cyclic robot trajectories. The next chapters will build on this foundation to explore practical applications and implementations of timed automata in robotics.

4 Set methods

4.1 Introduction

Set methods have their origins in the work of R. E. Moore, who introduced interval analysis in the 1960s [64]. Interval analysis provides a way to handle uncertainties and rounding errors in numerical computations by representing numbers as intervals rather than single values. This approach allows for the computation of guaranteed bounds on the results, as long as the inputs are bounded, which is particularly useful in fields where precision and reliability are critical.

The development of interval analysis laid the groundwork for the broader field of set methods, which extend the concept of intervals to more complex set representations such as zonotopes, polytopes, and ellipsoids. These methods have found applications in various domains, including the numerical resolution of physical problems. For example, set methods can be used to solve differential equations with guaranteed bounds on the solutions, ensuring that the computed results are reliable even in the presence of uncertainties.

In recent years, set methods have gained significant attention in the field of robotics. Robots often operate in dynamic and uncertain environments, where traditional methods may struggle to provide reliable solutions. Set methods offer a powerful tool for addressing these challenges by providing a way to model and compute with uncertainties. Applications in robotics include motion planning [34], state estimation [50], control [76], and even Simultaneous Localization and Mapping (SLAM) [6], where set methods can be used to ensure safety and reliability in the robot's operations.

Overall, set methods represent a robust approach to handling uncertainties and providing guarantees in numerical computations, with applications ranging from physical problem-solving to advanced robotics.

4.2 Set operations

Set operations are fundamental in set methods, as they allow for the manipulation and combination of sets. The most common set operations include union, intersection, and complement, which are defined as follows:

Definition 32. *The **union** of $\mathbb{A}$ and $\mathbb{B}$, two subsets of $\mathbb{E}$, is denoted $A \cup B$ and is the set containing all elements that are in $\mathbb{A}$ or $\mathbb{B}$ (or both).*

$$\mathbb{A} \cup \mathbb{B} = \{x \in \mathbb{E} \mid x \in \mathbb{A} \vee x \in \mathbb{B}\} \tag{4.1}$$

Definition 33. *The **intersection** of $\mathbb{A}$ and $\mathbb{B}$, two subsets of $\mathbb{E}$, is denoted $\mathbb{A} \cap \mathbb{B}$ and is the set containing all elements that are in both $\mathbb{A}$ and $\mathbb{B}$.*

$$\mathbb{A} \cap \mathbb{B} = \{x \in \mathbb{E} \mid x \in \mathbb{A} \wedge x \in \mathbb{B}\} \tag{4.2}$$

Definition 34. *The **complement** of $\mathbb{A}$, a subset of $\mathbb{E}$, in $\mathbb{E}$ is denoted $\overline{\mathbb{A}}$ and is the set containing all elements that are not in $\mathbb{A}$.*

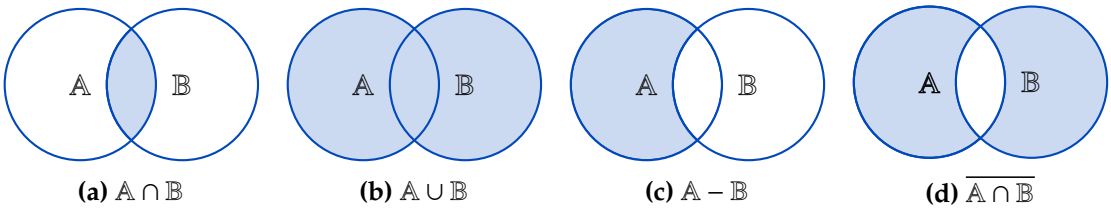
$$\overline{\mathbb{A}} = \{x \in \mathbb{E} \mid x \notin \mathbb{A}\} \tag{4.3}$$

Example 21. *Here is an example of set operations applied to sets immersed in the space $\mathbb{E} = \{1, 2, 3, 4, 5\}$, with $\mathbb{A} = \{1, 2, 3\}$ and $\mathbb{B} = \{3, 4, 5\}$:*

(i) $\mathbb{A} \cup \mathbb{B} = \{1, 2, 3, 4, 5\}$
(ii) $\mathbb{A} \cap \mathbb{B} = \{3\}$
(iii) $\overline{\mathbb{A}} = \{4, 5\}$
(iv) $\overline{\mathbb{B}} = \{1, 2\}$
(v) $\overline{\mathbb{A} \cup \mathbb{B}} = \emptyset$

(vi) $\overline{\mathbb{A} \cap \mathbb{B}} = \{1, 2, 4, 5\}$

Figure 4.1 illustrates these set operations using Venn diagrams. Each diagram shows the sets $\mathbb{A}$ and $\mathbb{B}$, with shaded areas representing the results of the operations.



(a) $\mathbb{A} \cap \mathbb{B}$ **(b)** $\mathbb{A} \cup \mathbb{B}$ **(c)** $\mathbb{A} - \mathbb{B}$ **(d)** $\overline{\mathbb{A} \cap \mathbb{B}}$

Figure 4.1 Example of set operations on two sets $\mathbb{A}$ and $\mathbb{B}$

4.3 Set Representation with Intervals

To use set methods, a set representation must be defined. There are many ways to represent sets, such as intervals [64], zonotopes [27, 49], polytopes [111], and ellipsoids [15]. Without going into the details of the different representations, this work focuses on the use of intervals for their simplicity of implementation, at the cost of a potentially greater pessimism than with other representations.

The formalism of interval analysis is attributed to R. E. Moore in [64]. This section presents the formalism of intervals and interval arithmetic by following the Moore formalism.

Definition 35. *An **interval** $[x] \in \mathbb{IR}$ is the set of bounded real numbers between its lower bound x^- and its upper bound x^+.*

$$[x] = [x^-, x^+] = \{x \in \mathbb{R} \mid x^- \leq x \leq x^+\} \tag{4.4}$$

Remark. *Lower bound x^- and upper bound x^+ may be infinite. For instance, the set of all positive reals $[x] = [0, +\infty]$ is an interval.*

As intervals are used to represent sets, set operators can be applied on intervals [64].

Example 22. *Here is an example of set operators applied on intervals:*
(i) $[1, 5] \cap [3, 7] = [3, 5]$
(ii) $[1, 2] \sqcup [5, 7] = [1, 7]$
(iii) $[-2, -1] \cap [2, 3] = \emptyset$

Remark. *The classical set union $\cup$ is not defined for intervals, as for disjoint intervals $[x]$ and $[y]$, $[x] \cap [y] = \emptyset$ and $[x] \cup [y]$ is not an interval as it is not connected. The operator $\sqcup$ is introduced as the interval hull operator to extend the union to intervals. $[x] \sqcup [y] = [[x] \cup [y]]$.*

Moreover, interval arithmetic is defined [64] and binary operators can be used on intervals.

Definition 36. ***Binary operators** $\diamond = \{+, -, \times, /\}$ are defined for intervals by:*

$$\forall ([x], [y]) \in \mathbb{IR}^2, \quad [x] \diamond [y] = \{x \diamond y \mid x \in [x] \wedge y \in [y]\} \tag{4.5}$$

Example 23. *Here is an example of binary operators applied to intervals:*
(i) $[-1, 2] + [5, 6] = [4, 8]$
(ii) $[3, 6] - [2, 3] = [0, 4]$
(iii) $[-2, 3] \times [4, 5] = [-10, 15]$

(iv) $[-10,-5]/[-1,1] = [-\infty,+\infty]$

Definition 37. *An **interval vector** $[\mathbf{x}] \in \mathbb{IR}^n$ is the cartesian product of intervals $[x_i] \in \mathbb{IR}$ forming the box.*

$$[\mathbf{x}] = [x_0] \times \ldots \times [x_{n-1}] \tag{4.6}$$

A binary operator is applied element-wise on interval vectors, such that the operator acts on each interval composing the interval vector, as defined above.

4.4 Set Representation with Pavings

4.4.1 Introduction to Pavings

In interval analysis, a paving is a fundamental concept used to represent sets through collections of non-overlapping boxes (intervals in higher dimensions). A paving $\mathbb{X}$ is defined as a finite union of boxes:

$$\mathbb{X} = \bigcup_{i=1}^{k} [\mathbf{x}_i] \tag{4.7}$$

where each $[\mathbf{x}_i] = [x_{i1}] \times [x_{i2}] \times \cdots \times [x_{in}]$ is an n-dimensional interval vector (box), and the boxes are non-overlapping in the sense that their interiors are disjoint.

The key advantage of pavings lies in their ability to provide guaranteed approximations of complex sets that may have irregular boundaries, disconnected components, or non-convex shapes. Unlike parametric representations, pavings can naturally handle sets with arbitrary topology without requiring prior knowledge of the set's structure.

To represent such sets, paver algorithms are used. These algorithms iteratively refine the representation of the set by subdividing boxes and testing their inclusion with respect to the target set. **SIVIA** (Set Inversion Via Interval Analysis) is the classical paver algorithm introduced by Jaulin and Walter [33]. It is designed to compute the set:

$$\mathbb{S} = \{\mathbf{x} \in \mathbb{X} \mid \mathbf{f}(\mathbf{x}) \in \mathbb{Y}\}, \tag{4.8}$$

where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ is a vector function, $\mathbb{X} \subseteq \mathbb{R}^n$ the initial set, $\mathbb{Y} \subseteq \mathbb{R}^m$ is a target set.

Algorithm 3 is the recursive version of the SIVIA algorithm that computes the paving of a set inversion problem as defined in Equation (4.8). Starting from an initial box $[\mathbf{x}_0] \in \mathbb{IR}^n$, a target set $[\mathbf{y}] \in \mathbb{IR}^m$, a function $\mathbf{f}$, and a termination criterion $\varepsilon > 0$, the SIVIA algorithm will compute subpavings $\mathbb{X}^-$, and $\mathbb{X}^+$ such that $\mathbb{X}^- \subseteq \mathbb{S} \subseteq \mathbb{X}^+$.

Example 24. *To characterize the set of points that are at a distance $d = [2,3]$ for the origin $(0,0)$, the distance function can be used to define the set:*

$$\mathbb{S} = \{(x,y) \in \mathbb{R}^2 \mid \sqrt{x^2+y^2} \in [2,3]\} \tag{4.9}$$

To represent this set using a paving, the SIVIA algorithm can be applied to compute the inner and outer approximations of $\mathbb{S}$. The resulting pavings will consist of boxes that cover the area between the circles of radius 2 and 3 centered at the origin, effectively capturing the annular region defined by the distance constraint.

Figure 4.2 illustrates the inner and outer approximations of the set $\mathbb{S}$, where the inner approximation consists of the pink boxes entirely contained within the annulus, and the outer approximation consists of the union of pink and yellow boxes that cover the entire annulus.

Algorithm 3 Recursive SIVIA for function

Input: $[\mathbf{x}] \in \mathbb{IR}^n$, $[\mathbf{y}] \in \mathbb{IR}^m$, $\varepsilon > 0$, $\mathbf{f}$
Output: $(\mathbb{X}^-, \mathbb{X}^+)$

function SIVIA($[\mathbf{x}], [\mathbf{y}], \mathbf{f}, \varepsilon$)
    if width($[\mathbf{x}]$) $< \varepsilon$ **then** ▷ Termination condition
        return $(\emptyset, \{[\mathbf{x}]\})$
    else if $\mathbf{f}([\mathbf{x}]) \subseteq [\mathbf{y}]$ **then** ▷ $[\mathbf{x}] \subseteq \mathbb{S}$
        return $(\{[\mathbf{x}]\}, \{[\mathbf{x}]\})$
    else if $\mathbf{f}([\mathbf{x}]) \cap [\mathbf{y}] = \emptyset$ **then** ▷ $[\mathbf{x}] \cap \mathbb{S} = \emptyset$
        return $(\emptyset, \emptyset)$
    else
        $([\mathbf{x}_1], [\mathbf{x}_2]) \leftarrow$ bisect($[\mathbf{x}]$) ▷ Bisect $[\mathbf{x}]$
        $(\mathbb{X}_1^-, \mathbb{X}_1^+) \leftarrow$ SIVIA($[\mathbf{x}_1], [\mathbf{y}], \mathbf{f}, \varepsilon$) ▷ Call SIVIA on $[\mathbf{x}_1]$
        $(\mathbb{X}_2^-, \mathbb{X}_2^+) \leftarrow$ SIVIA($[\mathbf{x}_2], [\mathbf{y}], \mathbf{f}, \varepsilon$) ▷ Call SIVIA on $[\mathbf{x}_2]$
        return $(\mathbb{X}_1^- \cup \mathbb{X}_2^-, \mathbb{X}_1^+ \cup \mathbb{X}_2^+)$
    end if
end function

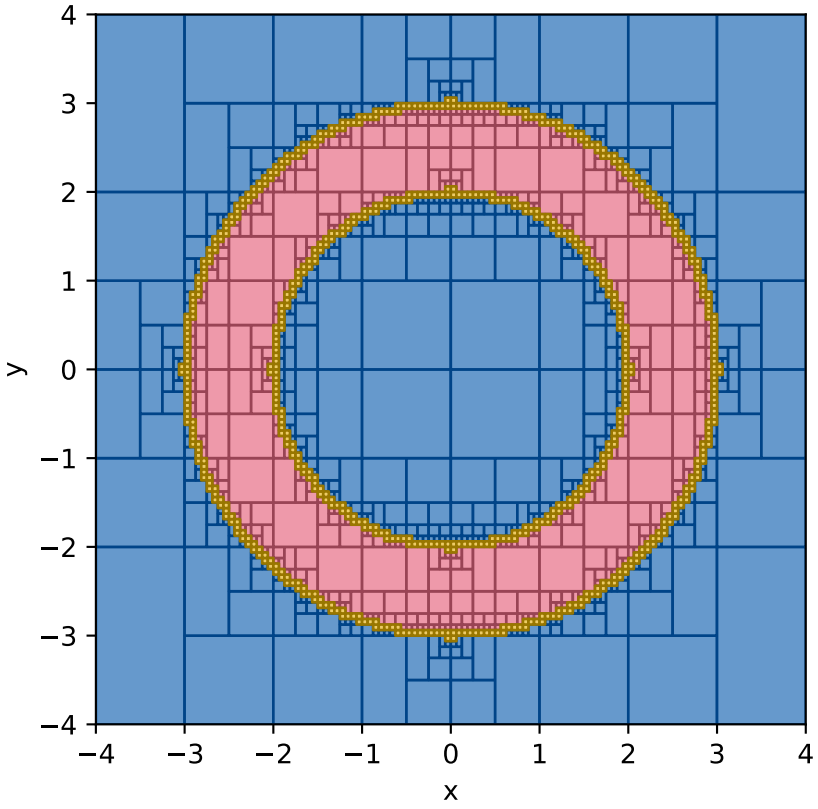


Figure 4.2 Inner and outer approximations of the annular set $\mathbb{S}$ defined by the distance constraint $d = [2, 3]$ from the origin

4.4.2 Inner and Outer Approximations

Given a target set $\mathbb{S}$, we can construct two pavings to represent it:

Definition 38. *An **inner approximation** (or **inner paving**) $\mathbb{X}^-$ of a set $\mathbb{S}$ is a paving such that:*

$$\mathbb{X}^- \subseteq \mathbb{S} \tag{4.10}$$

Every box in $\mathbb{X}^-$ is entirely contained within $\mathbb{S}$. In Example 24, the inner approximation consists of the pink boxes that are completely inside the annulus.

Definition 39. *An **outer approximation** (or **outer paving**) $\mathbb{X}^+$ of a set $\mathbb{S}$ is a paving such that:*

$$\mathbb{S} \subseteq \mathbb{X}^+ \tag{4.11}$$

The union of all boxes in $\mathbb{X}^+$ completely contains $\mathbb{S}$. In Example 24, the outer approximation consists of the union of the pink and yellow boxes that cover the entire annulus.

Definition 40. *The **boundary** of the set $\mathbb{S}$ is denoted by $\partial\mathbb{S}$ and is enclosed between $\mathbb{X}^-$ and $\mathbb{X}^+$.*

The complete approximation of a set $\mathbb{S}$ is then given by the couple $(\mathbb{X}^-, \mathbb{X}^+)$, where:

$$\mathbb{X}^- \subseteq \mathbb{S} \subseteq \mathbb{X}^+ \tag{4.12}$$

4.4.3 Limitations of Pavings

Such pavings can be computed for low dimensional sets, but the scalability is computationally limited. Actually, as the algorithm is based on bisections of boxes, the number of boxes grows exponentially with the dimension of the set. This approach is then very limited for sets with dimensions greater than 3 or 4. In these cases, the use of contractors is preferred to only remove parts of boxes that do not satisfy the constraints.

4.5 Contractors

A contractor is a tool used in set methods to contract a box, i.e., to remove parts of the box that do not satisfy a given constraint. It is a function that maps boxes to smaller boxes, ensuring that the resulting box is contained within the original box and that it satisfies the constraint.

Definition 41. *A **contractor** $C_{\mathcal{L}}$ associated to a constraint $\mathcal{L}$ is a function $C_{\mathcal{L}} : \mathbb{R}^n \to \mathbb{R}^n$, such that*

(i) (Contraction) $\forall [\mathbf{x}] \in \mathbb{R}^n, \quad C_{\mathcal{L}}([\mathbf{x}]) \subseteq [\mathbf{x}]$

(ii) (Consistency) $\forall \mathbf{x} \in [\mathbf{x}], \quad \mathcal{L}(x) \implies \mathbf{x} \in C_{\mathcal{L}}([\mathbf{x}])$.

Figure 4.3 shows an example of a contractor $C_{\mathcal{L}}$ associated to a constraint $\mathcal{L}$, applied on a box $[\mathbf{x}]$. The set of points where the constraint $\mathcal{L}$ is satisfied is shown in green, and the contractor removes parts of the box $[\mathbf{x}]$ where this constraint is not satisfied.

Remark. *A contractor $C_{\mathcal{L}}$ only removes parts of $[\mathbf{x}]$ where the constraint $\mathcal{L}$ is not satisfied, but the contracted box may still contain points that do not satisfy the constraint $\mathcal{L}$.*

Contractors can be used to contract boxes respecting a constraint $\mathcal{L}$, or can be used to represent the set consistent with the constraint by computing the outer approximation of the set $\mathbb{S}$ such that $\mathbb{S} \subseteq \mathbb{X}^+$.

Algorithm 4 is the recursive SIVIA algorithm to compute the outer approximation of a set using a contractor C. The algorithm is called on an initial box $[\mathbf{x}_0] \subseteq \mathbb{R}^n$, a contractor C, and a termination criterion $\varepsilon > 0$. It will compute the outer approximation of the set $\mathbb{S}$ such that $\mathbb{S} \subseteq \mathbb{X}^+$.

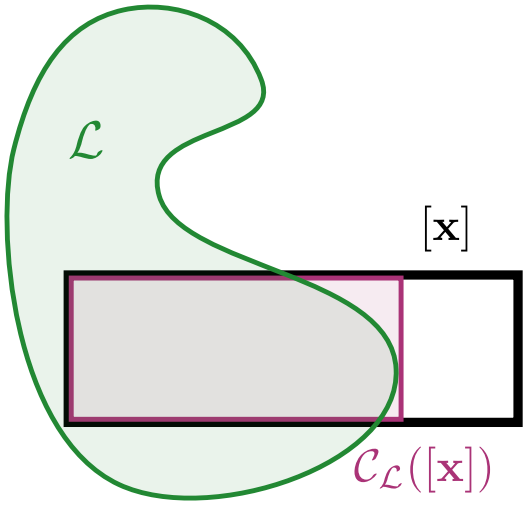


Figure 4.3 Contractor applied on a box $[\mathbf{x}]$

Algorithm 4 Recursive SIVIA for Contractor

Input: $[\mathbf{x}] \in \mathbb{R}^n, C, \varepsilon > 0$
Output: $\mathbb{X}^+$

function SIVIA($[\mathbf{x}], C, \varepsilon$)
  if width($[\mathbf{x}]$) $< \varepsilon$ **then** ▷ Termination condition
    return $\{[\mathbf{x}]\}$
  end if
  $[\mathbf{x}_{out}] \leftarrow C([\mathbf{x}])$ ▷ Contracted box by the Contractor
  if $[\mathbf{x}_{out}] = \emptyset$ **then** ▷ $[\mathbf{x}] \cap \mathbb{S} = \emptyset$
    return $\emptyset$
  else
    $([\mathbf{x}_1], [\mathbf{x}_2]) \leftarrow$ bisect($[\mathbf{x}]$) ▷ Bisect $[\mathbf{x}]$
    $\mathbb{X}_1^+ \leftarrow$ SIVIA($[\mathbf{x}_1], C, \varepsilon$) ▷ Call SIVIA on $[\mathbf{x}_1]$
    $\mathbb{X}_2^+ \leftarrow$ SIVIA($[\mathbf{x}_2], C, \varepsilon$) ▷ Call SIVIA on $[\mathbf{x}_2]$
    return $\mathbb{X}_1^+ \cup \mathbb{X}_2^+$
  end if
end function

Example 25. *Recalling the distance example from Example 24, a contractor can be defined to remove points that are not at a distance $d = [2, 3]$ from the origin. Figure 4.4 shows the paving of the contractor consistent with the distance constraint $d = [2, 3]$ from the origin $(0, 0)$.*

Contractors is an efficient tool to compute the outer approximation of a set. However, if a box is fully inside the set, the paving algorithm will bisect it until it reaches the termination criterion ε. This can lead to an unnecessarily large number of boxes, and computation time can be significantly increased. To avoid this, separators are introduced to remove points that are satisfying the constraint, and to also compute the inner approximation of a set.

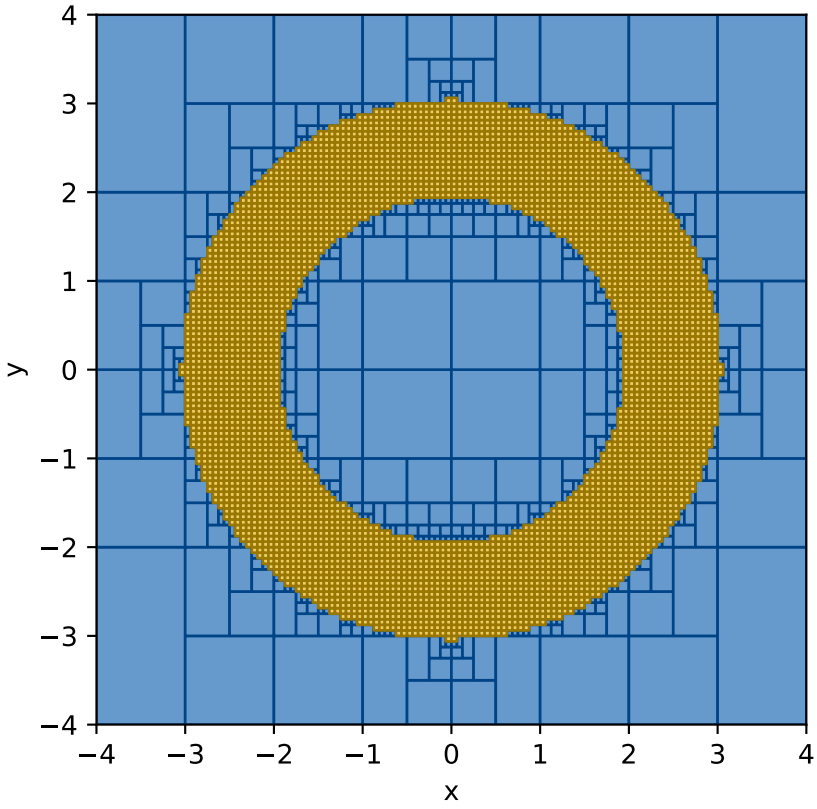


Figure 4.4 Paving of the contractor $C_{\mathcal{L}}$ applied on the annular set $\mathbb{S}$ defined by the distance constraint $d = [2, 3]$ from the origin

4.6 Separators

A separator is defined as a pair of contractors that work jointly to remove points belonging to and not belonging to the set with respect to constraint $\mathcal{L}$.

Definition 42. *A **separator** $\mathcal{S}_{\mathcal{L}} = \{\mathcal{S}^{in}_{\mathcal{L}}, \mathcal{S}^{out}_{\mathcal{L}}\}$ is a set of two contractors $\mathcal{S}^{in}_{\mathcal{L}}$ and $\mathcal{S}^{out}_{\mathcal{L}}$ such that*

$$\begin{aligned} \mathcal{S}_{\mathcal{L}} : &\mathbb{IR}^n \to \mathbb{IR}^n \times \mathbb{IR}^n \\ &[x] \mapsto (\mathcal{S}^{in}_{\mathcal{L}}([x]), \mathcal{S}^{out}_{\mathcal{L}}([x])) \end{aligned}. \tag{4.13}$$

Figure 4.5 shows the action of a separator $\mathcal{S}_{\mathcal{L}}$ on a box $[\mathbf{x}]$. The contractor $\mathcal{S}^{in}_{\mathcal{L}}$ removes points satisfying constraint $\mathcal{L}$, while the contractor $\mathcal{S}^{out}_{\mathcal{L}}$ removes points that do not satisfy $\mathcal{L}$.

Property 2. *A separator $\mathcal{S}$ satisfies the **complementary property**.*

$$\forall [\mathbf{x}] \in \mathbb{R}^n \quad , \mathcal{S}^{in}_{\mathcal{L}}([\mathbf{x}]) \cup \mathcal{S}^{out}_{\mathcal{L}}([\mathbf{x}]) = [\mathbf{x}] \tag{4.14}$$

Separator algebra is defined [39, 36]. This enables set operations with separators such the complementary, union, intersection, and difference of a separator by another.

Definition 43. *Some set operators on separators are defined as follows:*

(i) $\overline{\mathcal{S}}_{\mathcal{L}} = \{\mathcal{S}^{out}_{\mathcal{L}}, \mathcal{S}^{in}_{\mathcal{L}}\}$

(ii) $\mathcal{S}_{\mathcal{L}_1} \cap \mathcal{S}_{\mathcal{L}_2} = \{\mathcal{S}^{in}_{\mathcal{L}_1} \cup \mathcal{S}^{in}_{\mathcal{L}_2}, \mathcal{S}^{out}_{\mathcal{L}_1} \cap \mathcal{S}^{out}_{\mathcal{L}_2}\}$

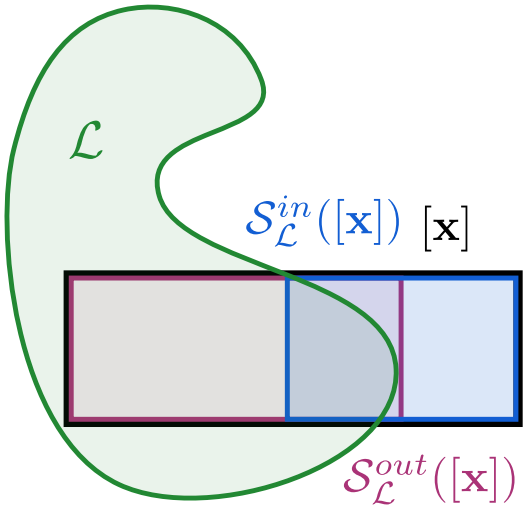


Figure 4.5 Separator applied on a box $[\mathbf{x}]$

(iii) $\mathcal{S}_{\mathcal{L}_1} \cup \mathcal{S}_{\mathcal{L}_2} = \{\mathcal{S}_{\mathcal{L}_1}^{in} \cap \mathcal{S}_{\mathcal{L}_2}^{in}, \mathcal{S}_{\mathcal{L}_1}^{out} \cup \mathcal{S}_{\mathcal{L}_2}^{out}\}$
(iv) $\mathcal{S}_{\mathcal{L}_1} \setminus \mathcal{S}_{\mathcal{L}_2} = \mathcal{S}_{\mathcal{L}_1} \cap \overline{\mathcal{S}}_{\mathcal{L}_2}$

These operators are necessary to combine separators and to compute sets respecting many constraints.

Algorithm 5 is the recursive SIVIA algorithm to compute outer and inner approximations of a separator $\mathcal{S}$. The algorithm is called on an initial box $[\mathbf{x}_0] \in \mathbb{R}^n$, a separator $\mathcal{S}$, and a termination criterion $\varepsilon > 0$. It will compute the inner approximation $\mathbb{X}^-$ and the outer approximation $\mathbb{X}^+$ of the set $\mathbb{S}$ such that $\mathbb{X}^- \subseteq \mathbb{S} \subseteq \mathbb{X}^+$.

4.7 Paver Algorithms

Paver algorithms for classical inclusion tests, contractor and separator paving were introduced in the previous sections. These algorithms are used to compute the outer approximation, and when it makes sense, the inner approximation of a set defined by a constraint $\mathcal{L}$. The general principle of these algorithms is summarized in the following steps:

(i) Starting with an initial search domain (bounding box)
(ii) Recursively subdividing boxes based on inclusion tests
(iii) Classifying each box as inside, or outside
(iv) Terminating when a desired precision is reached and classifying the box as boundary

4.7.1 Contracting SIVIA Algorithm

The classical SIVIA algorithm is effective, but does not take advantage of the contractions provided by contractors and separators of the boxes during its execution. Therefore, the computed paving is regular, leading to a uniform subdivision of the search space.

Algorithm 6 is the recursive SIVIA algorithm to compute outer and inner approximations of a separator $\mathcal{S}$, taking advantage of the contractions at each iteration. The algorithm is called on an initial box $[\mathbf{x}_0] \in \mathbb{R}^n$, a separator $\mathcal{S}$, and a termination criterion $\varepsilon > 0$. It

Algorithm 5 Recursive SIVIA for Separators

Input: $[\mathbf{x}] \in \mathbb{IR}^n$, $\mathcal{S}$, $\varepsilon > 0$
Output: $(\mathbb{X}^-, \mathbb{X}^+)$

function SIVIA($[\mathbf{x}]$, $\mathcal{S}$, ϵ)
  if width($[\mathbf{x}]$) $< \varepsilon$ **then** ▷ Termination condition
    return $(\emptyset, \{[\mathbf{x}]\})$
  end if
  $([\mathbf{x}_{in}], [\mathbf{x}_{out}]) \leftarrow \mathcal{S}([\mathbf{x}])$ ▷ Contracted boxes by the Separator
  if $[\mathbf{x}_{out}] = \emptyset$ **then** ▷ $[\mathbf{x}] \cap \mathbb{S} = \emptyset$
    return $(\emptyset, \emptyset)$
  else if $[\mathbf{x}_{in}] = \emptyset$ **then** ▷ $[\mathbf{x}] \subseteq \mathbb{S}$
    return $(\{[\mathbf{x}]\}, \emptyset)$
  else
    $([\mathbf{x}_1], [\mathbf{x}_2]) \leftarrow$ bisect($[\mathbf{x}]$) ▷ Bisect $[\mathbf{x}]$
    $(\mathbb{X}_1^-, \mathbb{X}_1^+) \leftarrow$ SIVIA($[\mathbf{x}_1]$, $\mathcal{S}$, ε) ▷ Call SIVIA on $[\mathbf{x}_1]$
    $(\mathbb{X}_2^-, \mathbb{X}_2^+) \leftarrow$ SIVIA($[\mathbf{x}_2]$, $\mathcal{S}$, ε) ▷ Call SIVIA on $[\mathbf{x}_2]$
    return $(\mathbb{X}_1^- \cup \mathbb{X}_2^-, \mathbb{X}_1^+ \cup \mathbb{X}_2^+)$
  end if
end function

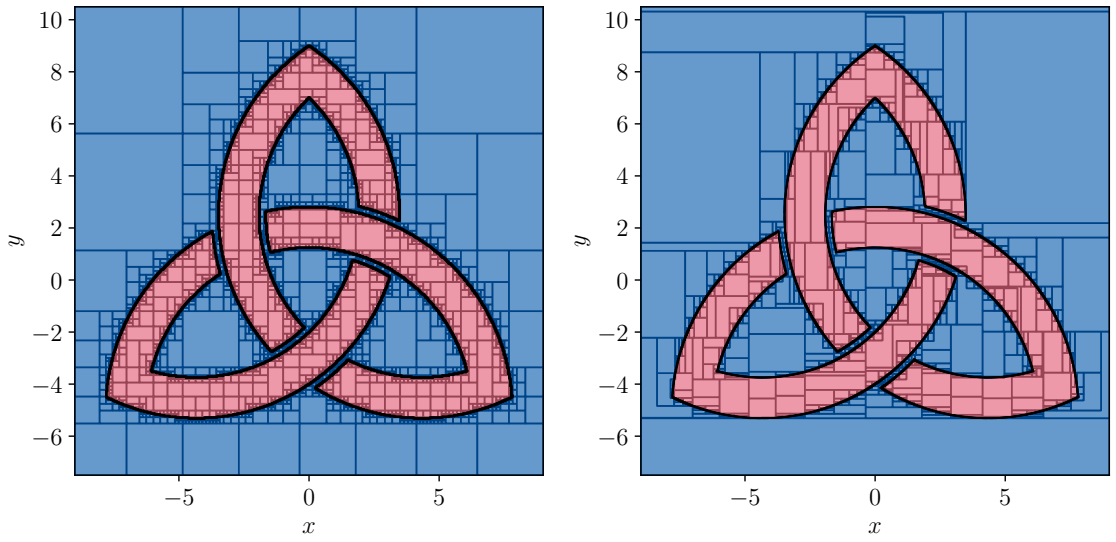
will compute the inner approximation $\mathbb{X}^-$ and the outer approximation $\mathbb{X}^+$ of the set $\mathbb{S}$ such that $\mathbb{X}^- \subseteq \mathbb{S} \subseteq \mathbb{X}^+$. The difference between this algorithm and Algorithm 5 is that at each iteration, the part $[\mathbf{x}] \setminus [\mathbf{x}_{in}]$ is added to $\mathbb{X}^-$, the bisection is done on $[\mathbf{x}_{in}] \cap [\mathbf{x}_{out}]$, and a final contraction is applied at termination.

Algorithm 6 Recursive SIVIA for Separators with contractions

Input: $[\mathbf{x}] \in \mathbb{IR}^n$, $\mathcal{S}$, $\varepsilon > 0$
Output: $(\mathbb{X}^-, \mathbb{X}^+)$

function SIVIA_Contracting($[\mathbf{x}]$, $\mathcal{S}$, ϵ)
  $([\mathbf{x}_{in}], [\mathbf{x}_{out}]) \leftarrow \mathcal{S}([\mathbf{x}])$ ▷ Contracted boxes by the Separator
  if width($[\mathbf{x}]$) $< \varepsilon$ **then** ▷ Termination condition
    return $([\mathbf{x}] \setminus [\mathbf{x}_{in}], \{[\mathbf{x}_{in}]\})$ ▷ Benefit from the last contraction
  else if $[\mathbf{x}_{out}] = \emptyset$ **then** ▷ $[\mathbf{x}] \cap \mathbb{S} = \emptyset$
    return $(\emptyset, \emptyset)$
  else if $[\mathbf{x}_{in}] = \emptyset$ **then** ▷ $[\mathbf{x}] \subseteq \mathbb{S}$
    return $(\{[\mathbf{x}]\}, \emptyset)$
  else
    $([\mathbf{x}_1], [\mathbf{x}_2]) \leftarrow$ bisect($[\mathbf{x}_{in}] \cap [\mathbf{x}_{out}]$) ▷ Bisect $[\mathbf{x}_{in}] \cap [\mathbf{x}_{out}]$
    $(\mathbb{X}_1^-, \mathbb{X}_1^+) \leftarrow$ SIVIA($[\mathbf{x}_1]$, $\mathcal{S}$, ε) ▷ Call SIVIA on $[\mathbf{x}_1]$
    $(\mathbb{X}_2^-, \mathbb{X}_2^+) \leftarrow$ SIVIA($[\mathbf{x}_2]$, $\mathcal{S}$, ε) ▷ Call SIVIA on $[\mathbf{x}_2]$
    return $([\mathbf{x}] \setminus [\mathbf{x}_{in}] \cup \mathbb{X}_1^- \cup \mathbb{X}_2^-, [\mathbf{x}] \setminus [\mathbf{x}_{in}] \cup \mathbb{X}_1^+ \cup \mathbb{X}_2^+)$
  end if
end function

Figure 4.6 shows an example of the paving of a Celtic triangle using two different paving algorithms. Figure 4.6a uses the classical SIVIA paver for separators presented in Algorithm 5, while Figure 4.6b takes advantage of contractions at each iteration and uses the paver presented in Algorithm 6. Therefore, the paving is no longer regular, but there are fewer boxes to handle.



(a) Classical paving **(b)** Paving using contractions

Figure 4.6 Paving types comparison on a Celtic triangle

Remark. *Contracted pavings take advantage of the contractions at each iteration, which allows solving higher-dimensional problems with a reasonable number of boxes. For example, if the number of boxes is fixed, and relatively low, a regular paving will barely represent the considered set, while a contracted paving will use the first iteration to contract the initial box around the considered set, and use the other boxes to refine the representation of this set.*

4.7.2 Paving Resolution

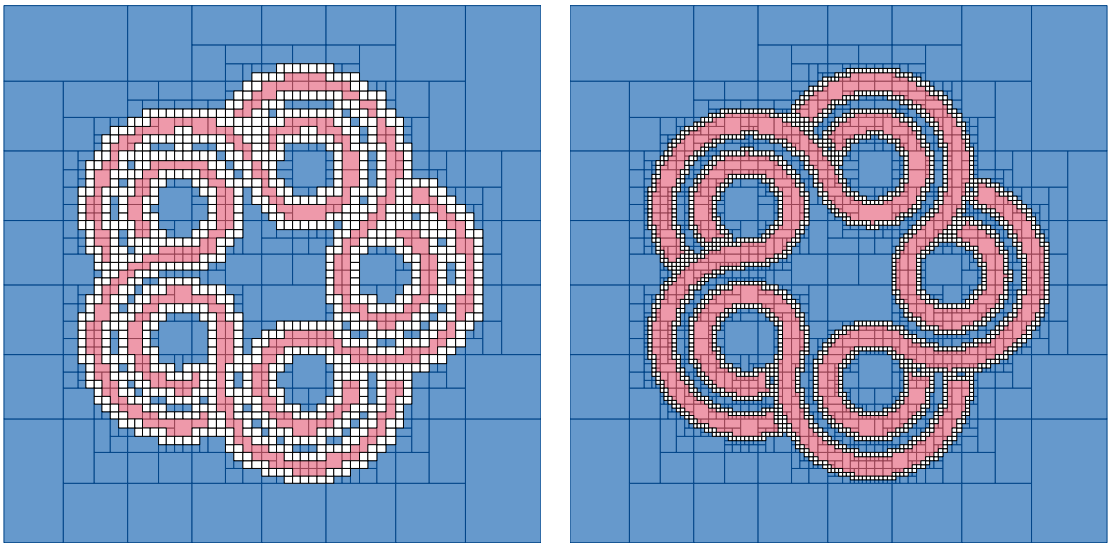
Figure 4.7 shows the effect of the paving resolution on the paving of a 5 balls figure [52]. The smaller the resolution ϵ, the more precise the paving, but the more boxes are generated. There is then a trade-off between the precision of the paving and the number of boxes to handle.

4.8 Conclusion

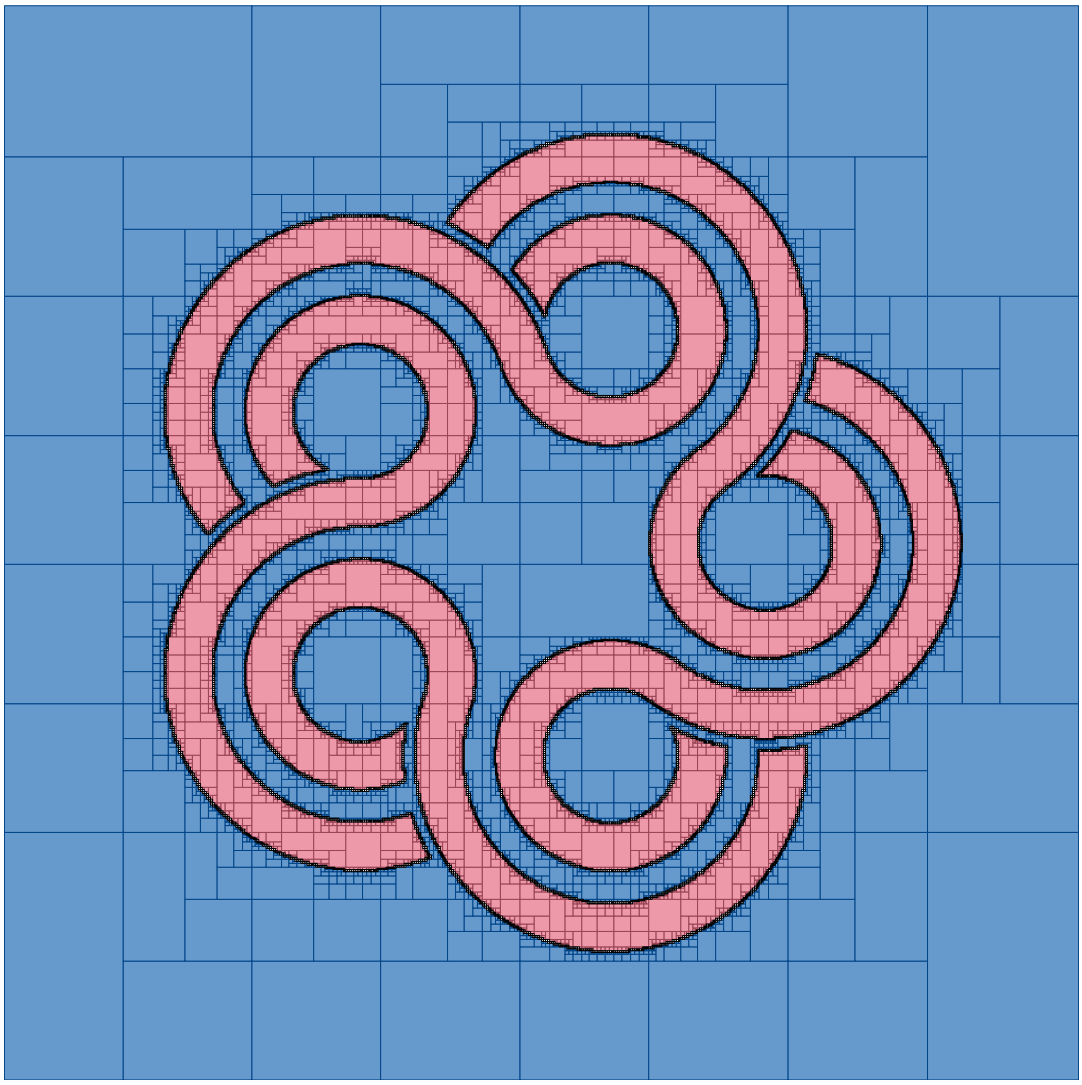
This chapter has laid the theoretical foundation for the use of Set Methods in robotics and state estimation, with a focus on handling constraints and uncertainties in a reliable and computationally tractable way. We began by introducing fundamental set operations and representations, which provide a language to deal with uncertainties. Sets naturally capture bounded uncertainties, and their operations enable the formulation of robust and conservative approximations of possible states or behaviors.

Contractors and separators were also introduced as tools to characterize sets and to propagate constraints. Both are essential for representing sets consistent with a constraint. Pavings can be computed to represent sets by enclosing them with collections of boxes, which can be inner or outer approximations. This paving approach allows for a flexible and efficient representation of sets. They are particularly useful in two-dimensional cases, as the paving can be visualized in a figure. In higher-dimensional cases, the number of boxes can be very large due to bisections, so the computation of a paving is less efficient, and the representation of these pavings is not as intuitive.

Some paving algorithms were presented, including the classical SIVIA algorithm and its variants. These algorithms were primarily introduced to pave a box to compute the outer and eventually the inner approximation of a set defined by a constraint using a



(a) $\epsilon = 0.5$ **(b)** $\epsilon = 0.25$



(c) $\epsilon = 0.05$

Figure 4.7 Paving resolution on the 5 balls figure

regular paving. Modern paving methods take advantage of contractors to reduce the size of boxes at each iteration, leading to more efficient computations, but producing a non-regular paving. The SIVIA algorithm, in its various forms, is a cornerstone of set methods, enabling the computation of guaranteed enclosures of solution sets under uncertainty.

This manuscript will take a stability analysis and an invariance analysis approach, in contrast to other uses of set methods for the study of dynamic systems, which focus on the integration of system evolution equations and the propagation of uncertainties along the time. Actually, there is a community that aims to enclose the trajectories of dynamical systems in sets. There is for instance the guaranteed integration of CAPD [103], or the enclosing of trajectories of robots in tubes developed in the Codac Library [77].

Taken together, all these tools presented in this part constitute a powerful framework for modeling and solving problems in robot navigation and state estimation, especially in environments characterized by limited sensing, complex constraints, and modeling uncertainty. In particular, they form the backbone of the approaches developed in the second part of this manuscript, where they are applied to stability analysis and state estimation in practical navigation tasks. Chapter 2 present the formalism of dynamical systems that will be necessary to model robots, then Chapter 3 introduces finite state machines and timed automaton that plays a central role in the cycle navigation. Finally, Chapter 4 have presented the set methods that will be used to ensure the stability of the navigation cycle, and to estimate the state of the robot in a robust way. The formal guarantees offered by set methods are essential for ensuring robust performance in safety-critical and poorly observable systems.

Part II:

Contributions

5 Cycle Control

5.1 Introduction

Robotic systems conventionally solve the state estimation problem before starting to navigate. This approach, formalized by Smith & Cheeseman [86] and extensively documented in probabilistic robotics literature [91], mandates accurate localization as a prerequisite for navigation tasks. The methodology relies on continuous pose estimation through sensor fusion algorithms, typically incorporating Global Navigation Satellite Systems (GNSS) for terrestrial applications or acoustic positioning systems in underwater environments.

The dependency on external positioning references presents fundamental limitations in GNSS-denied operational domains. Underwater robotics applications suffer from this constraint, where electromagnetic signal attenuation renders satellite-based positioning inoperative. Current underwater positioning methodologies employ acoustic systems, specifically Long Baseline (LBL) and Ultra-Short Baseline (USBL) configurations. LBL systems achieve positioning accuracies between 0.01m and 1m through triangulation from multiple seafloor-mounted transponders [62]. However, these systems require extensive infrastructure deployment, including pre-mission bathymetric surveys and geometric optimization of transponder arrays.

USBL systems offer operational flexibility through surface vessel integration but exhibit range-dependent accuracy degradation [94, 17]. Both acoustic positioning modalities impose significant economic and logistical constraints on autonomous underwater vehicle (AUV) operations, particularly in deep-water or remote deployment scenarios where infrastructure establishment becomes economically prohibitive. It also requires that a surface support vessel is able to navigate the mission area to set up the acoustic positioning system, which is not always possible.

Recent advances in GNSS-denied navigation have focused on computational complexity reduction and sensor fusion optimization [1]. However, these approaches maintain the fundamental requirement for initial state estimation prior to navigation execution, perpetuating operational limitations in infrastructure-constrained environments.

This chapter presents cycle navigation, a new paradigm for autonomous robotic navigation. This approach, unlike traditional methods, starts by controlling the robot to move along a predefined cyclic trajectory without knowing its position. Then, by gathering a few exteroceptive measurements, the cycle is progressively shifted and stabilized in the environment. Once the cycle is stabilized, the state estimation problem is resolved. This method represents a significant departure from conventional navigation paradigms, enabling autonomous operation in infrastructure-denied environments while maintaining a robust and efficient navigation framework.

The cycle navigation method combines a dynamical system and a timed automaton into a discrete system with transition durations as inputs the transition durations of the automata. Adjusting the transition durations will affect the position of the cycle. The theoretical analysis proves the system controllability, and a controller is designed to stabilize the cycle to the desired stable cycle in the environment.

Cycle navigation eliminates dependency on external positioning infrastructure while enabling immediate navigation task execution. The method exhibits robustness in feature-sparse environments where simultaneous localization and mapping (SLAM) algorithms may fail due to insufficient observational data. The mathematical framework provides convergence guarantees for limit cycle stability, ensuring simultaneous resolution of navigation and localization objectives.

The theoretical contributions of this chapter include:

(i) Formalization of the cycle abstraction of a dynamical system and its associated timed automaton,
(ii) Controllability analysis of the cycle discrete system,
(iii) Design of a controller to stabilize the cycle in the environment,
(iv) Simulation and experimental validation of the cycle navigation method

The methodology represents a fundamental shift from conventional navigation paradigms, replacing the sequential localize-then-navigate approach with a concurrent navigate-while-localizing framework. This inversion enables autonomous operation in infrastructure-denied environments while maintaining mathematical rigor through hybrid system theory and providing theoretical guarantees of convergence to stable, localizable cyclic trajectories.

5.2 Formalism

Consider a dynamical system with state $\mathbf{x}$ and of input $\mathbf{u}$ following the dynamical evolution equation

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{5.1}$$

Let $\mathbf{x}_0$ be the initial state of the system. Then there exists a flow function φ which is the solution of Equation (5.1)

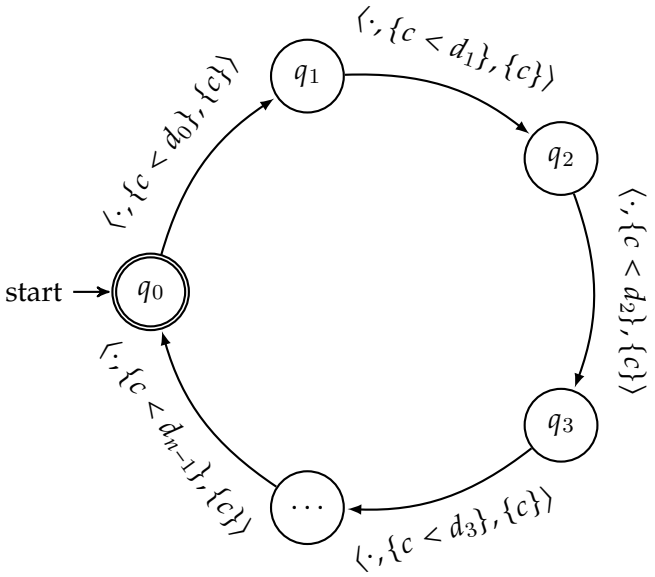
$$\mathbf{x}(t) = \varphi(\mathbf{x}_0, t). \tag{5.2}$$

This flow function φ is the same as presented in Chapter 2, and is the solution of the dynamical system evolution equation Equation (5.1).

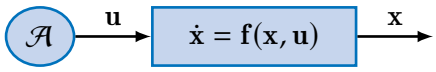
Consider now a cyclic deterministic timed automaton

$$\mathcal{A} = \langle \Sigma, Q = \{q_i, i \in [\![0, n-1]\!]\}, Q_0 = \{q_0\}, C, F, E \rangle, \tag{5.3}$$

with an input $\mathbf{u}_i$ associated to each state q_i. As the state machine is executed, the input associated with the current state is used to control the dynamical system. The block diagram of this concept is shown in Figure 5.1b.



(a) Cyclic timed automaton



(b) Block diagram of the system controlled by a timed automaton

Figure 5.1 Controlling a dynamical system using a cyclic timed automaton

The robot is a continuous dynamical system, and the timed automaton is a discrete one. The coupling of these two systems forms a hybrid system [28], where the timed automaton controls the dynamical system by providing inputs $\mathbf{u}_i$ to the evolution equation Equation (5.1). The details of such systems is not developed in this manuscript as it is not the focus of this work, but the interested reader can refer to [93] for a complete introduction to hybrid systems.

The dynamical system is then governed by an evolution equation with a constant input $\mathbf{u}_i$ during the state q_i. This function is denoted by $\mathbf{f}_i$ for convenience. The flow function $\varphi_i : \mathcal{S} \times \mathcal{T} \to \mathcal{S}$ is also defined for the dynamical system following the evolution equation $\mathbf{f}_i$. This flow function models the evolution of the system under the action of the evolution function $\mathbf{f}_i$ for the duration d_i.

This hybrid system is an event-driven system, where transitions between states are triggered by the clock constraints of the timed automaton. When the clock c exceeds the duration d_i, an event triggers the transition from state q_i to state $q_{(i+1)}$. Event-driven systems are not new [60, 19], and are used in robotics to trigger measurements or changes of control laws based on events.

By denoting by $\mathbf{x}_i$ the state of the system when the timed automaton switches from state q_{i-1} to q_i, the flow function φ_i links these different states during execution of the finite automaton

$$\mathbf{x}_{i+1} = \varphi_i(\mathbf{x}_i, d_i). \tag{5.4}$$

Through transitions, the cyclic timed automaton controls the dynamical system by moving it through successive states $\mathbf{x}_i$ as shown in Figure 5.2. Introducing the operator $\bigcirc_{i=0}^{n}$ to denote the composition of functions over the cyclic time automaton states, ϕ represents the flow functions over a complete iteration of the automaton.

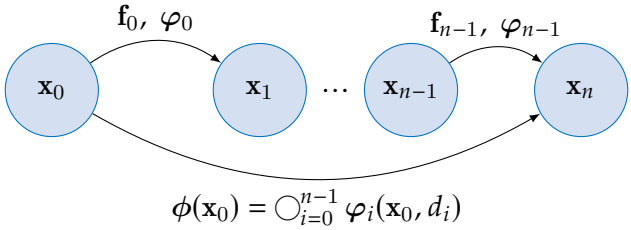


Figure 5.2 Composition of flow functions over a cycle

5.3 Cyclic period

For a cyclic deterministic timed automaton, it is possible to define the cyclic period.

Definition 44. *The **cyclic period** T of a cyclic deterministic timed automaton is the duration of a complete iteration of the timed automaton until it returns to its initial state. Suppose that for each edge $e \in E$ starting from the state q_i, the clock constraint $b \in \mathbb{B}(C)$ is $\{c > d_i\}$, then*

$$T(\mathbf{x}_0) = \sum_{i=0}^{i=n} d_i. \tag{5.5}$$

In the case of the cyclic navigation, the cyclic period $T(\mathbf{x}_0)$ is considered independent of the initial state of the dynamical system $\mathbf{x}_0$.

Definition 45. *The coupled timed automaton and the robot are **fully cyclic**, if the cyclic period T does not depend on the initial condition of the robot $\mathbf{x}_0$. In this case*

$$\forall \mathbf{x}_0 \in \mathbb{R}^n, T(\mathbf{x}_0) = T. \tag{5.6}$$

This condition requires that the dynamical system should behave in the same way regardless of its initial state. This is the case for many dynamical systems, such as the unicycle model [22], where the trajectory of the robot is independent of its initial position. This condition implies that the system is time invariant, and that the disturbances are constant spatially and temporally during the experiments.

5.4 Synchronization condition

Definition 46. *A dynamical system and a timed automaton are **synchronized** if the state of the dynamical system $\mathbf{x}$ returns to the same state after a complete iteration of the cyclic timed automaton. This condition is fulfilled if the flow function ϕ meets the condition*

$$\phi(\mathbf{x}(t)) \triangleq \mathbf{x}(t+T) = \mathbf{x}(t). \tag{5.7}$$

The synchronization of the timed automaton and the dynamical system is essential to have the trajectory of the vehicle describing cycles. If the synchronization condition is not satisfied, either the timed automaton structure is inadequate in which case the trajectory

imposed on the robot is not well-designed to perform cycles, or the transition durations are not correctly adjusted, in which case the dynamical system and the timed automaton can be synchronized by adjusting the transition durations.

Example 26. *Suppose a dynamical system of state* $\mathbf{x} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$*, following the unicycle evolution equation*

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u) = \begin{cases} v \cdot cos(\theta) \\ v \cdot sin(\theta) \\ u \end{cases}, \tag{5.8}$$

where x*, and* y *are the abscissa and the ordinate of the vehicle,* v *is its velocity, and the input* u *controls its turning rate. When* u *is equal to zero, the vehicle is moving straight forward, when it is positive the vehicle turns anti-clockwise, and when it is negative the vehicle turns clockwise.*

A cyclic deterministic timed automaton $\mathcal{A} = \langle \Sigma, Q, Q_0, C, F, E \rangle$ *is defined to control this vehicle, with the following parameters:*

- $\Sigma = \emptyset$
- $Q = \{q_i, \forall i \in [\![0, 7]\!]\}$
- $Q_0 = \{q_0\}$
- $C = \{c\}$
- $F = Q$
- $E = \{e_i, \forall i \in [\![0, 7]\!], e_i = \langle q_i, q_{(i+1)\%7}, \cdot, c \geq d_i, \{c\} \rangle\}$*, where* $d_i(\boldsymbol{\omega})$ *is the duration of the* i^{th} *state, before the transition triggering.*

To each state q_i *is associated an input* $u = i\%2$*. The trajectory of the vehicle is then alternatively describing straight lines or circle arcs. The duration of each state* d_i *is chosen following Equation (5.9) such that with a vehicle velocity* $v = 1$*, straight lines are 1 meters long, and circle arcs are* 90 *degrees left turns.*

$$d_i = \begin{cases} 1 & i\%2 = 0 \\ \frac{\pi}{2} & else \end{cases} \tag{5.9}$$

Figure 5.3a shows the cyclic timed automaton and the associated trajectory described by the vehicle under the timed automaton input. The vehicle and the cyclic timed automaton are synchronized as the robot state returns to its initial state after an iteration of the timed automata. The cyclic period is $T = 4 + 2\pi$*.*

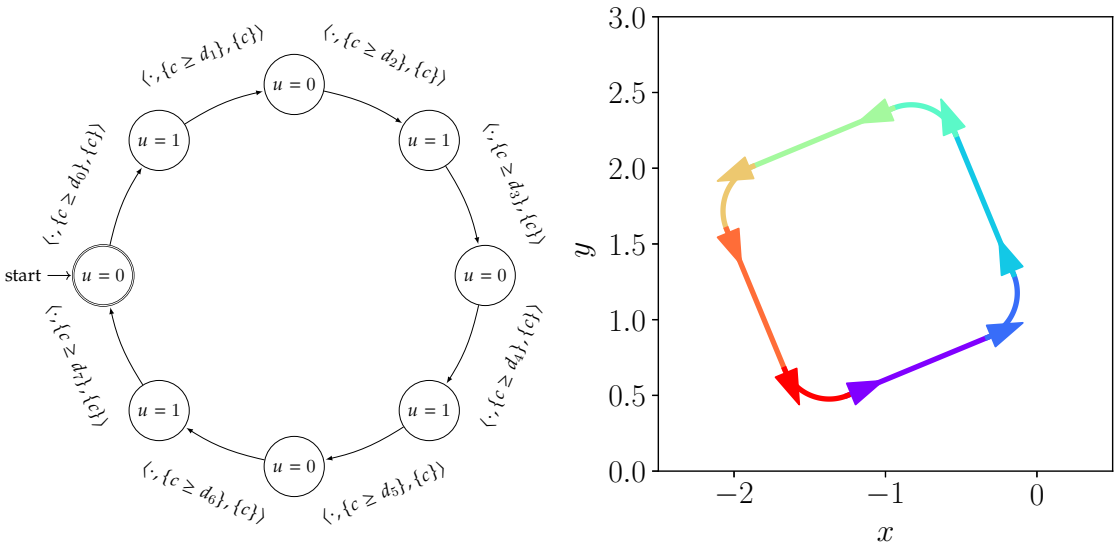
Remark. *At this step, the flow function* $\boldsymbol{\varphi}$ *is important to ensure the synchronization condition is met. It is not always possible to find an analytical expression for the flow function, especially for complex dynamical systems. In this case, the flow function can be approximated using numerical methods if required, even with guarantees when dealing with set methods for numerical integration [42, 80] of CAPD [103] or codac [77].*

5.5 Cycle discretization

Now a discretization of the system is required, to look at the evolution of the cycle through the cyclic timed automaton iterations.

Definition 47. *Let* $\boldsymbol{\eta}_k$ *be the state of the dynamical system at the beginning of the* k^{th} *iteration of the timed automaton.* $\boldsymbol{\eta}_k$ *is called the* ***cyclic state*** *of the system.*

Remark. *In navigation using cycles, the cyclic state* $\boldsymbol{\eta}_k$ *is never measured, and almost never estimated, except in Chapter 8 where the cyclic state is estimated using set methods. This is one of*



(a) Square cyclic timed automaton

(b) Trajectory of the robot

Figure 5.3 Square cycle described by the trajectory of a robot controlled by a cyclic timed automaton

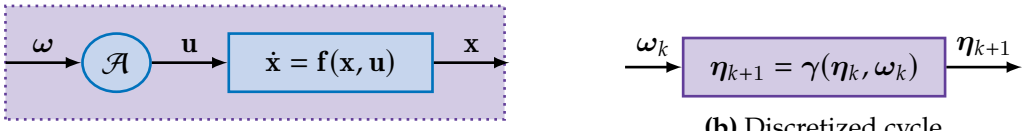
the main differences with traditional navigation methods, where the state of the robot is continuously estimated.

Also introduce the possibility of adjusting the transition durations at the beginning of the k^{th} iteration of the state machine using an input parameter $\boldsymbol{\omega}_k$.

Definition 48. *Let $\boldsymbol{\omega}_k$ be a vector of parameters of the timed automaton at the beginning of the k^{th} iteration. $\boldsymbol{\omega}_k$ is called the **cyclic input** of the system and can adjust the transition durations $d_i(\boldsymbol{\omega}_k)$ of the timed automaton.*

Then, transition durations now depend on this parameter $d_i(\boldsymbol{\omega})$, but remain constant over an iteration.

Expressions of $d_i(\boldsymbol{\omega})$ should ensure that when $\boldsymbol{\omega}_k$ is a null vector, the dynamical system and the timed automaton should be synchronized. In contrast, this synchronization condition is temporarily unmet when $\boldsymbol{\omega}_k$ is not null. Figure 5.4 shows the block diagram of the discretization.



(a) Timed automaton and dynamical system

(b) Discretized cycle

Figure 5.4 Discretization step of the controlled system

Remark. *The input parameter $\boldsymbol{\omega}_k$ of the timed automaton does not have to alter all the durations $d_i(\boldsymbol{\omega}_k)$, but could modify only a subset of the transition durations.*

5.6 Moving the cycle

To move the cycle in the two-dimensional plane, a subset of transition durations d_i has to be chosen in such a way that the cycle itself is isoactuated. Changing some durations will affect the state of the cycle $\boldsymbol{\eta}_k$ through iterations.

Example 27. *For the square cycle example, the cycle has to be controlled around its three degrees of freedom, which are the x-axis, the y-axis, and the rotation around itself. It requires then three transition durations d_i to be controlled, as long as no redundancy occurs in their action on the displacement of the cycle.*

Therefore, $\boldsymbol{\omega}_k = \begin{bmatrix}\omega_{k,0} & \omega_{k,1} & \omega_{k,2}\end{bmatrix}^T$ is a three-dimensional vector, and chosen durations can be

$$\begin{cases} d_4 = 1 + \omega_{k,0} \\ d_6 = 1 + \omega_{k,1} \\ d_7 = \frac{\pi}{2} + \omega_{k,2} \end{cases}. \tag{5.10}$$

By changing the durations d_4, d_6 and d_7, the length of the green line, the orange line, and the red circle arc are respectively affected, and this allows moving the starting position of the next cycle. Figure 5.5 shows the effect of modifying some transition durations d_i of the timed automaton on the trajectory of the vehicle.

It is then possible for this example to find the expression of the evolution function $\boldsymbol{\gamma}(\boldsymbol{\eta}_k, \boldsymbol{\omega}_k)$ of the cycle

$$\boldsymbol{\gamma}(\boldsymbol{\eta}_k, \boldsymbol{\omega}_k) = \boldsymbol{\eta}_k + \begin{bmatrix} -cos(\theta_k)\cdot\omega_{k,0} - sin(\theta_k)\cdot\omega_{k,1} + \frac{-sin(\theta_k)+sin\left(\theta_k+\frac{\pi}{2}\cdot\omega_{k,2}\right)}{\pi} \\ sin(\theta_k)\cdot\omega_{k,0} - cos(\theta_k)\cdot\omega_{k,1} + \frac{cos(\theta_k)-cos\left(\theta_k+\frac{\pi}{2}\cdot\omega_{k,2}\right)}{\pi} \\ \omega_{k,2} \end{bmatrix}. \tag{5.11}$$

Figure 5.6 shows the action of the modification of these durations on the trajectory of the cycle through iterations. The cycle is moved in its own frame along its three degrees of freedom.

Remark. *As the flow function is available for the Dubins car as presented in Chapter 2 and in [67], there is an analytical expression of the trajectory of the robot in the two-dimensional plane. This may not be the case for all dynamical systems. As a reminder, in this case the trajectory of the robot can be estimated using numerical integration [42] of CAPD [103] or codac [77] methods.*

In the case of the square cycle, the cycle is controlled in its own frame, i.e. the cycle is shifted along its x-axis and y-axis. A change of input could lead to a control in the world frame, regardless of the orientation of the cycle.

Remark. *Some transition durations d_i are affecting many degrees of freedom. For instance, altering d_7 will change the orientation of the cycle $\eta_{k,2}$, but also its position $\eta_{k,0}$ and $\eta_{k,1}$. This phenomenon can be seen in Figure 5.6d, in which the position of the cycle is shifted through iterations.*

5.7 Change of input

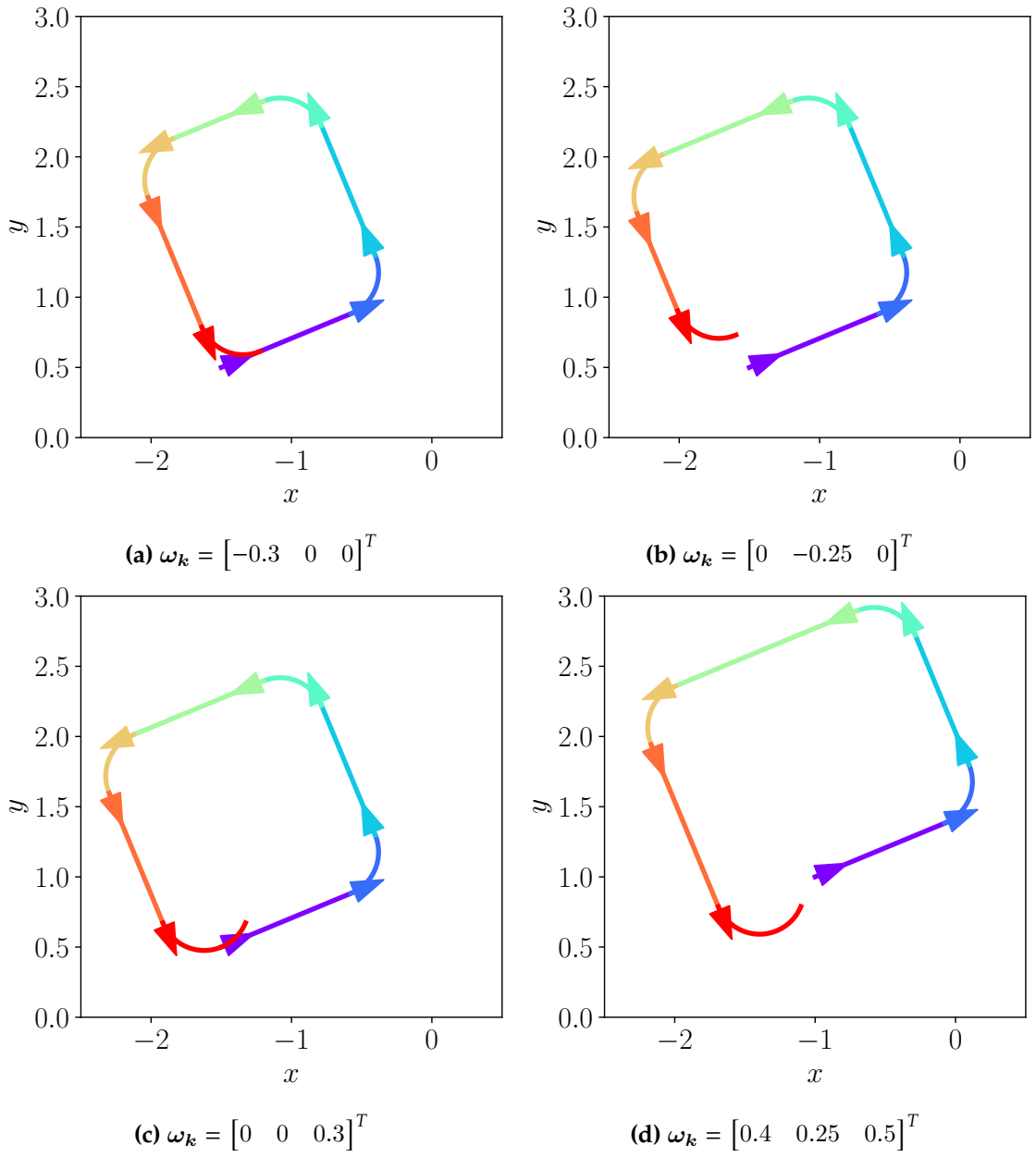
The goal now is to control the state of the cycle in the world frame, regardless its orientation. This first stage of regulation will compute $\boldsymbol{\omega}_k$, the input of the cycle, from the requested move $\boldsymbol{\nu}_k$ expressed in the world frame using

$$\boldsymbol{\omega}_k = \boldsymbol{\zeta}(\theta_k, \boldsymbol{\nu}_k) = \begin{bmatrix} -cos(\theta_k)\cdot\nu_{k,0} + sin(\theta_k)\cdot\nu_{k,1} - \frac{-sin(\theta_k)+sin\left(\theta_k+\frac{\pi}{2}\cdot\nu_{k,2}\right)}{\pi} \\ -sin(\theta_k)\cdot\nu_{k,0} - cos(\theta_k)\cdot\nu_{k,1} - \frac{cos(\theta_k)-cos\left(\theta_k+\frac{\pi}{2}\cdot\nu_{k,2}\right)}{\pi} \\ \nu_{k,2} \end{bmatrix}. \tag{5.12}$$

With this change of input, the current dynamics of the system are then modeled by

$$\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + \boldsymbol{\nu}_k, \tag{5.13}$$

and the evolution equation of the system is now linear. Figure 5.7 shows the block diagram of the regulation.



(a) $\omega_k = \begin{bmatrix} -0.3 & 0 & 0 \end{bmatrix}^T$ **(b)** $\omega_k = \begin{bmatrix} 0 & -0.25 & 0 \end{bmatrix}^T$

(c) $\omega_k = \begin{bmatrix} 0 & 0 & 0.3 \end{bmatrix}^T$ **(d)** $\omega_k = \begin{bmatrix} 0.4 & 0.25 & 0.5 \end{bmatrix}^T$

Figure 5.5 Square cycle controlled using some durations of the timed automaton

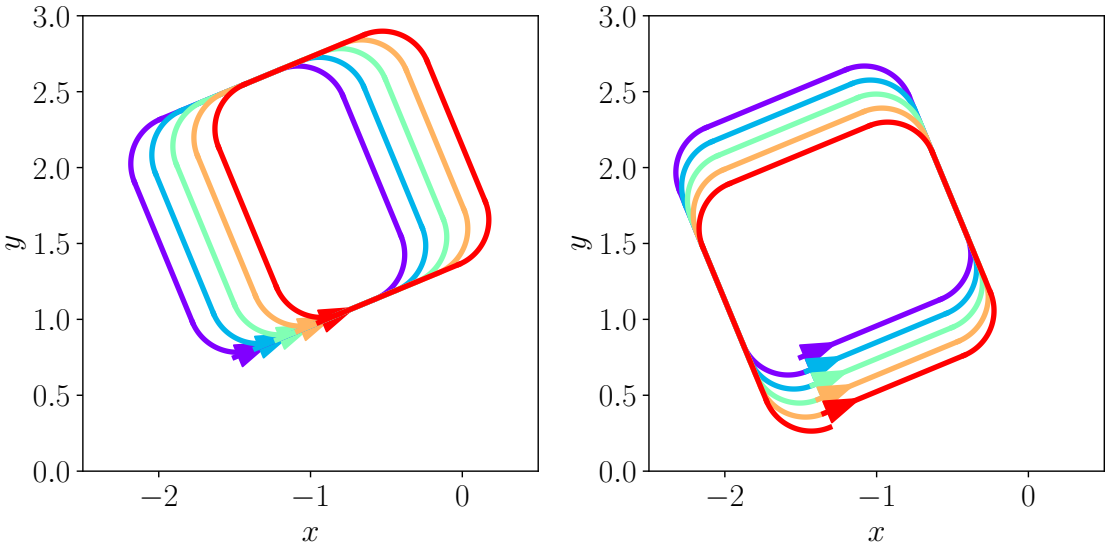
5.8 Degrees of freedom and control saturation

The dynamical system may have constraints that prevent it from moving in some directions, as with the unicycle for instance which is a non-holonomic system. However, using the cycle discretization, the new system may be controlled along all the directions if the controlled transitions are well-chosen without redundancy.

The inputs ω_k can be seen as actuators for the cycle. There is a kind of one-way saturation phenomenon on these actuators, as the transition durations must satisfy the non-negativity constraint:

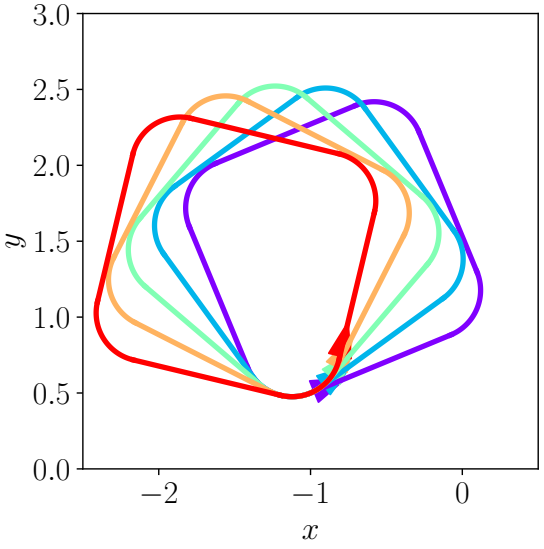
$$\forall i \in [\![0, n]\!], d_i(\omega_k) \geq 0 \tag{5.14}$$

This means that for any desired state η_d near the current state η_k is reachable in one step, but a condition is linking η_d, η_k, and the dimensions of the cycle.

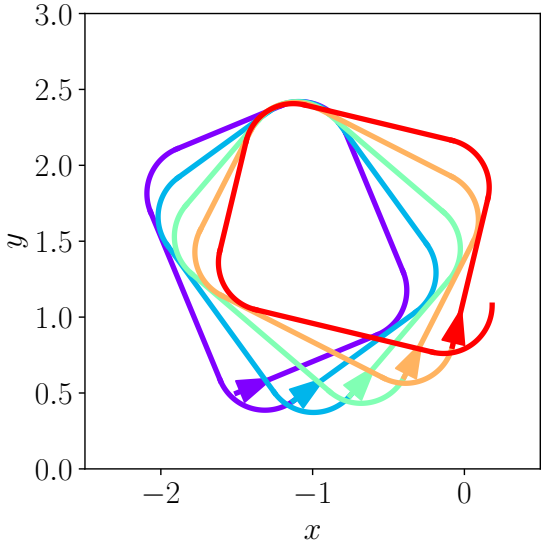


(a) $\omega_k = \begin{bmatrix} -0.15 & 0 & 0 \end{bmatrix}^T$

(b) $\omega_k = \begin{bmatrix} 0 & 0.1 & 0 \end{bmatrix}^T$



(c) $\omega_k = \begin{bmatrix} 0 & 0 & 0.15 \end{bmatrix}^T$



(d) $\omega_k = \begin{bmatrix} -0.25 & 0.2 & 0.15 \end{bmatrix}^T$

Figure 5.6 Moving the cycle using ω_k

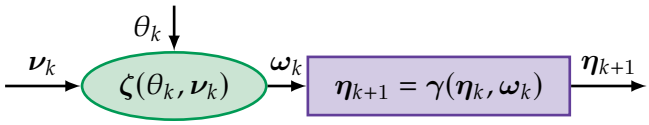


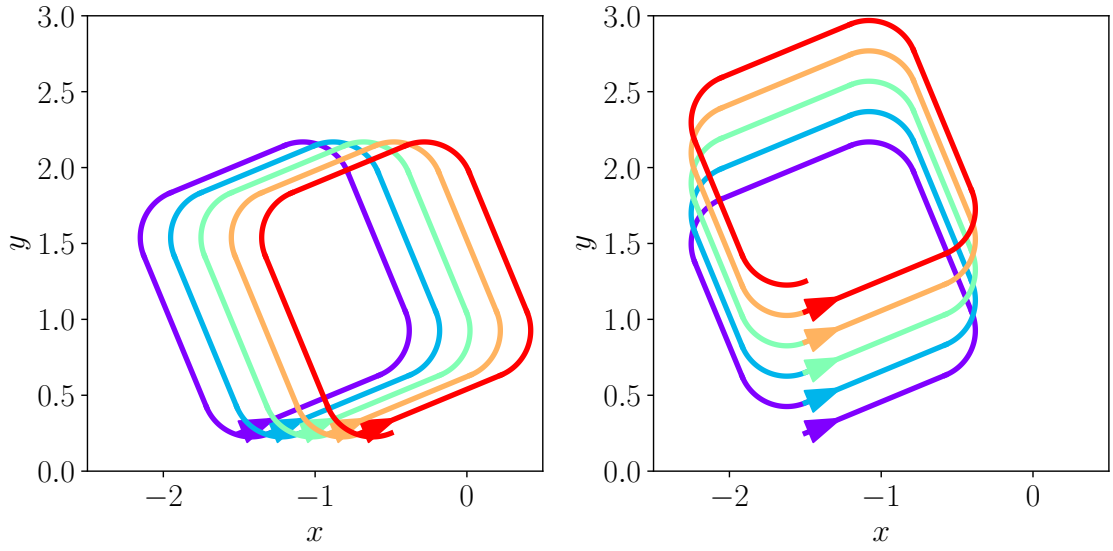
Figure 5.7 Block diagram of the controlled cycle

5.9 Sensor referenced control

The robot is now able to perform measurements μ_k at each iteration k. These measurements can be used to adjust the input of the cycle ω_k to stabilize the cycle in the environment. The whole system is modelled by

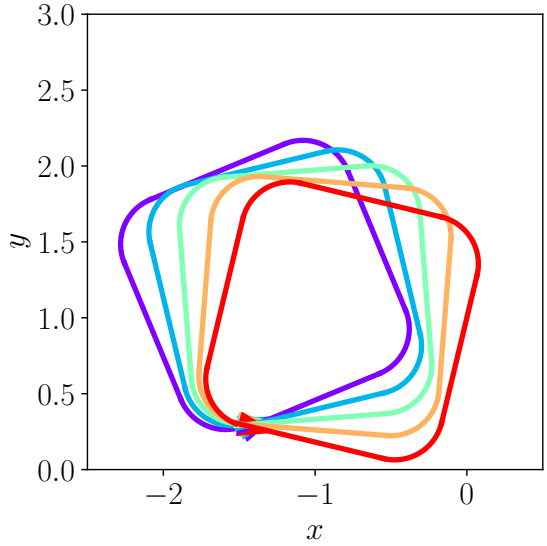
$$\mathcal{S} : \begin{cases} \eta_{k+1} = \gamma(\eta_k, \omega_k) \\ \mu_k = \sigma(\eta_k, \omega_k) \end{cases}. \tag{5.15}$$

The measurements μ_k can be used to control the system by generating inputs ω_k to

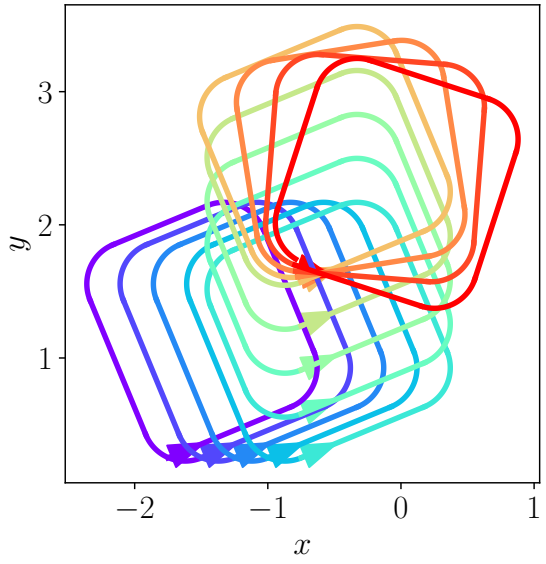


(a) $\boldsymbol{\nu}_k = \begin{bmatrix} 0.2 & 0 & 0 \end{bmatrix}^T$

(b) $\boldsymbol{\nu}_k = \begin{bmatrix} 0 & 0.2 & 0 \end{bmatrix}^T$



(c) $\boldsymbol{\nu}_k = \begin{bmatrix} 0 & 0 & 0.1 \end{bmatrix}^T$



(d) Successive control along the three degrees of freedom

Figure 5.8 Cycle control in the world frame

maintain the desired cycle. The measurement function $\boldsymbol{\sigma}$ maps the state and input to the measurements. It depends on the environment, the physical quantity sensed by the robot, and the position of the measurements relative to the cycle.

Remark. *The measurement equation $\boldsymbol{\sigma}(\boldsymbol{\eta}_k, \boldsymbol{\omega}_k)$ is almost never invertible with minimal sensors (one dimensional depth ranging sonar, temperature sensor, electric sense, etc.), as there exist many cyclic states $\boldsymbol{\eta}_k$ that will give the same measurements $\boldsymbol{\mu}_k$. With classical navigation sensors, such as GNSS, the goal is to have this measurement equation invertible to estimate the position of the system without any ambiguity from a measurement. The cyclic state $\boldsymbol{\eta}_k$ is then not measured, and not estimated at this point. Chapter 8 will show how to estimate the cyclic state using set methods and measurements gathered by minimal sensors.*

In the cycle formalism, it is important to notice that k is the counter of the cycle iteration, but it does not represent a time. Therefore, as k is used for the cyclic state $\boldsymbol{\eta}_k$, and for the

cyclic measurements $\boldsymbol{\mu}_k$, these are not representing the same instant, but less classically, $\boldsymbol{\mu}_k$ is a vector of measurements gathered all along the cycle at different times. Measurements performed in the future are transported at the start of the k^{th} iteration of the cycle, by a Poincaré transport operation [89, 69]. Focusing on measurement times in an event-driven system parallels max-plus algebra and the dater approach [38].

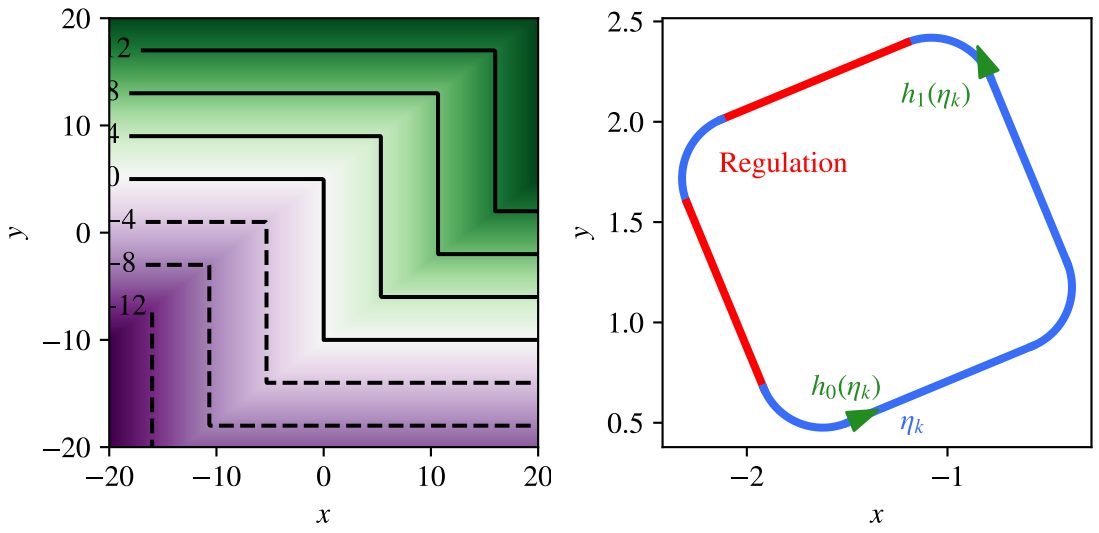
Remark. *It is possible to decouple the position of the measurements and the regulation of the cycle in order to simplify the control. This means that it is preferable to have the measurement equation only depending on the state of the cycle*

$$\boldsymbol{\mu}_k = \boldsymbol{\sigma}(\boldsymbol{\eta}_k) \tag{5.16}$$

to avoid the input $\boldsymbol{\omega}_k$ affecting the measurements $\boldsymbol{\mu}_k$. In practice, it implies that the measurements have to be gathered in states of the timed automata before transitions affected by the automata input $\boldsymbol{\omega}_k$.

Using these measurements, a feedback control law can be designed to adjust the input $\boldsymbol{\omega}_k$ to ensure the cycle remains stable and follows the desired trajectory.

Example 28. *Suppose the robot is equipped with an echosounder and is able to measure the depth below itself. The measurement positions are fixed relative to the state of the cycle, in other words relative to the start of the cycle. Two measurements are performed by the robot on the cycle, and these positions are shown in Figure 5.9b.*



(a) Simulated seafloor **(b)** Measurements positions

Figure 5.9 Environment and measurements positions

The measurements $\boldsymbol{\mu}_k$ are the depth below the robot at the positions of the measurements. The measurement function $\boldsymbol{\sigma}$ is then defined by

$$\begin{cases} g(\mathbf{x}) = 10 + \min\limits_{i=0,1} \left\{ 0.1 \cdot \det(\mathbf{b_i} - \mathbf{a_i}, \begin{bmatrix} x & y \end{bmatrix}^T - \mathbf{a_i}) \right\} \\ a_0 = \begin{bmatrix} 0 & 5 \end{bmatrix}^T, \quad b_0 = \begin{bmatrix} -1 & 0 \end{bmatrix}^T \\ a_1 = \begin{bmatrix} -1 & 0 \end{bmatrix}^T, \quad b_1 = \begin{bmatrix} 5 & -3 \end{bmatrix}^T \end{cases} . \tag{5.17}$$

Figure 5.9a shows this simulated seafloor. It is noticeable that there are two corners in which a square cycle could be stabilized. This will be the goal of the regulation stage which will be designed hereafter.

5.10 Controller design

As the inputs are decoupled in eq. (5.13), the controller can be designed for each input independently. Then, the controller will be able to stabilize the cycle in the environment using measurements $\boldsymbol{\mu}_k$.

It is then possible to consider the discrete one-dimensional problem modelling the state of the cycle through iterations

$$\begin{cases} x_{k+1} = x_k + u_k \\ z_k = a \cdot x_k + b \end{cases}, \tag{5.18}$$

where x_k is the state of the system at the k^{th} iteration, and z_k is the measurement of the depth below the robot. The seafloor is assumed to be linear with coefficients a and b estimated.

The goal is to find a controller to generate inputs u_k to move the state x_k toward x_d only by using measurements z_k. First, we use the measurement equation to define z_d

$$z_d = a \cdot x_d + b. \tag{5.19}$$

Then, without loss of generality, suppose that $x_d = 0$. It comes $b = z_d$. A Lyapunov based-synthesis [14] will be used to design the controller which will stabilize the system around its reference z_d. This controller is designed to ensure the system stability using the Lyapunov stability analysis presented in Chapter 2. By denoting by $e_k = z_d - z_k$ the measured error of position, it is possible to define the Lyapunov function

$$V_k = e_k^2. \tag{5.20}$$

Then, $V_{k+1} - V_k$ is developed as follows

$$\begin{aligned} V_{k+1} - V_k &= e_{k+1}^2 - e_k^2 \\ &= (z_d - z_{k+1})^2 - e_k^2 \\ &= (z_d - (ax_{k+1} + b))^2 - e_k^2 \\ &= (z_d - (a(x_k + u) + b))^2 - e_k^2 \\ &= (z_d - (z_k + au_k))^2 - e_k^2 \\ &= (e_k - au_k)^2 - e_k^2 \\ &= e_k^2 + au_k^2 - 2ae_k u_k - e_k^2 \\ &= au_k(u_k - 2e_k). \end{aligned}$$

This difference $V_{k+1} - V_k$ has to be negative to have a stable behavior of the controlled system. This quantity is negative if and only if

$$au_k(u_k - 2e_k) < 0. \tag{5.21}$$

This criterion will helps to design a suitable controller for the system that ensure the stability of the cycle in the environment. The next subsections will present the design of four controllers for the one-dimensional system of Equation (5.18). The progression toward the well-suited controller is shown, as this is exactly the approach that was used to find the controller in field experiments.

5.10.1 Dead-Beat controller

A dead-beat controller will find the input u_k such that the output of the system z_k reaches the reference z_d in one or a few steps [101]. In the current case, this is possible to reach the set-point in one step. To do this, the input should cancel the current state of the system x_k and feed the desired state for the output.

The dead-beat controller for this system is defined by

$$u_k = \frac{z_k - b}{a} + z_d. \tag{5.22}$$

If the initial state is x_0, then $x_1 = x_0 + u_0 = x_0 + \frac{z_0 - b}{a} + z_d = x_d$, and the reference is reached in one step. Figure 5.10 shows the application of this dead-beat controller on the proposed one-dimensional example. The seabed is shown in purple, the reference z_d is shown in red, and the position of the robot through iterations is shown in blue. In this example, as the dead-beat controller is able to reach the reference in one step, there is only the initial point at $x = 5$ and the final point at $x = -7.5$ that are visible. These colors will be used in the next figures to show the evolution of the system.

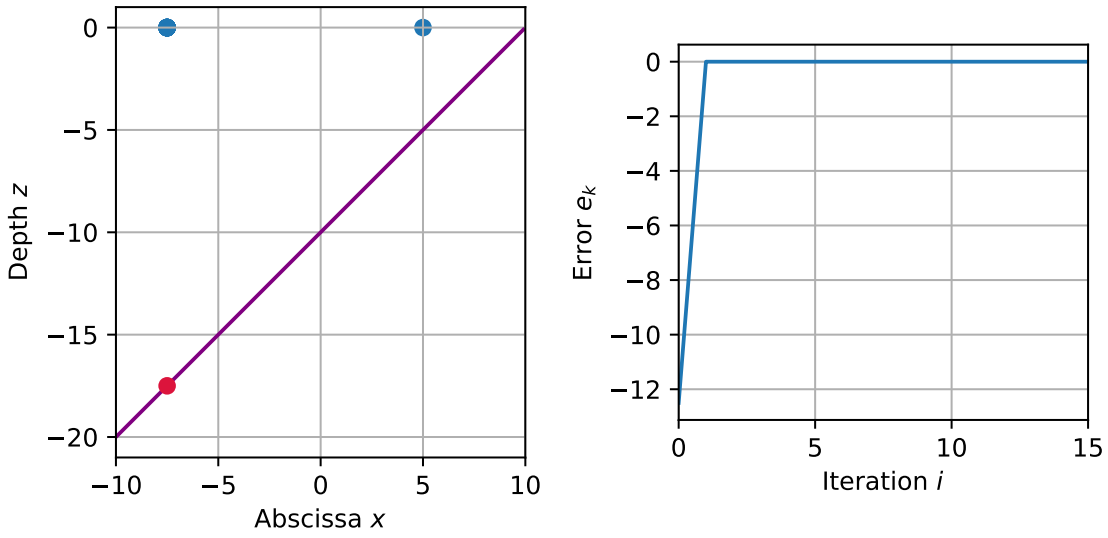


Figure 5.10 Simulation of the dead-beat controller

Remark. *The dead-beat controller works well on this example because the measurement equation $z_k = ax_k + b$ is perfectly known. In the case the measurement equation is uncertain, the system dynamics are not exactly compensated; convergence will occur over multiple iterations, and the behavior of the dead-beat controller is similar to a proportional controller.*

5.10.2 Proportional controller

A proportional controller can be designed to stabilize the system around the reference z_d [45, 85]. The proportional controller is defined by

$$u_k = Ke_k. \tag{5.23}$$

Theorem 4. *The proportional controller is Lyapunov-stable if and only if $0 < K < 2$.*

Proof. The controller should satisfy the condition Equation (5.21) to ensure a stable behavior. The difference $V_{k+1} - V_k$ is then

$$\begin{aligned} V_{k+1} - V_k &= a u_k (u_k - 2e_k) \\ &= aKe_k(Ke_k - 2e_k) \\ &= aKe_k^2(K-2). \end{aligned}$$

And the controller is Lyapunov-stable if and only if $V_{k+1} - V_k < 0$. This is equivalent to

$$\begin{aligned} V_{k+1} - V_k < 0 &\Leftrightarrow aKe_k^2(K-2) < 0 \\ &\Leftrightarrow K - 2 < 0 \\ &\Leftrightarrow 0 < K < 2. \end{aligned}$$

Figure 5.11 shows the set of stability of the proportional controller. The pink area is the set of (e_k, K) that do not satisfy the Lyapunov condition, and the blue area is the set of (e_k, K) that respect the Lyapunov condition. Choosing a gain K means drawing a horizontal line, and by choosing a gain K such that $0 < K < 2$ ensures the stability of the system, for the entire range of the error e_k.

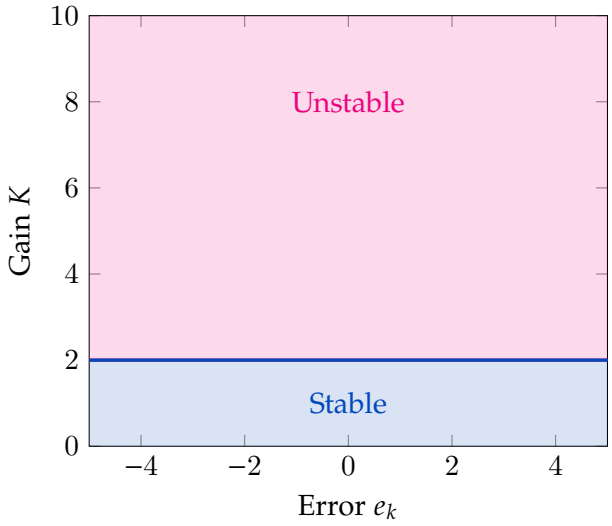


Figure 5.11 Stability condition for the proportional controller

□

Figure 5.12 shows the application of this proportional controller on the one-dimensional example with a gain $K = 1$. We see that the system converges and remains stable around its reference z_d.

Remark. *With a large error e_k, the input $u_k = Ke_k$ will be large too. Then, the cycle will no longer be a cycle, and the robot's trajectory will be greatly distorted. This is a problem of the proportional controller, particularly when the gain K is high. This can also be a problem if the depth under the robot is measured incorrectly. If the signal bounces off a fish passing underneath the sensor, the cycle can go very far off in one direction.*

5.10.3 Sign controller

The sign controller is a controller that will stabilize the system around the reference z_d using the sign of the error e_k. The controller is defined by

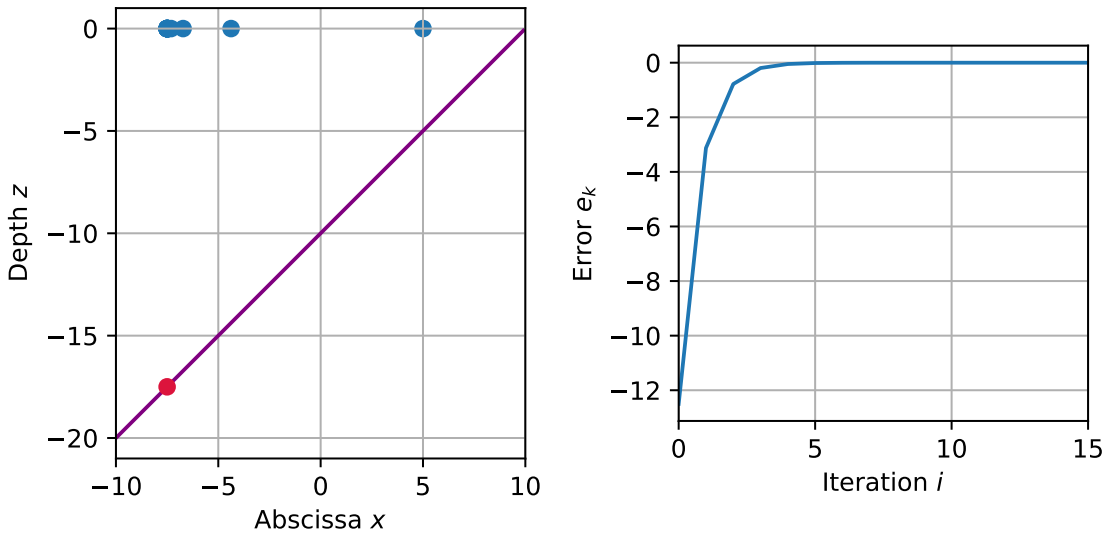


Figure 5.12 Simulation of the proportional controller

$$u_k = Ksign(e_k). \tag{5.24}$$

This sign controller moves the state of the cycle through iterations by a constant shift K. This controller is inspired by a sliding mode controller [45, 85], but it is not in its classical form as it is used in set-point regulation, and the system is discrete.

Theorem 5. *There is no constant gain K for the sign controller over the entire error range e_k such that the controller converges the system to its reference.*

Proof. Let us choose a gain $K \in \mathbb{R}_+^*$. Define the error $e_k = \frac{K}{4} > 0$, then the difference $V_{k+1} - V_k$ is

$$V_{k+1} - V_k = aK\left(K - 2\frac{K}{4}\right) = \frac{K^2}{2} > 0. \tag{5.25}$$

Therefore, this shows that the controller is not Lyapunov-stable for the whole range of the error e_k. Figure 5.13 shows the set of stability of the sign controller. The pink area is the set of (e_k, K) that does not respect the Lyapunov condition, and the blue area is the set of (e_k, K) that respect the Lyapunov condition. Choosing a gain K means drawing a horizontal line, but it is impossible to find one that does not cross the pink area.

□

The system behavior is shown in Figure 5.14. The system is not globally stable in the Lyapunov sense, but it is stable in a bounded region of the error e_k. The controller ensures the error will converge and remains in a ball around the reference z_d of radius $r = \frac{K}{2}$.

5.10.4 Tanh controller

The tanh controller is a controller that presents the advantage of the sign controller, i.e. the saturation that prevent the cycle from being overly distorted, but also adapt the behavior of the cycle around the reference. The tanh controller is defined by

$$u_k = K\tanh(e_k). \tag{5.26}$$

Theorem 6. *The tanh controller is globally stable if the gain K is chosen such that $0 < K < 2$.*

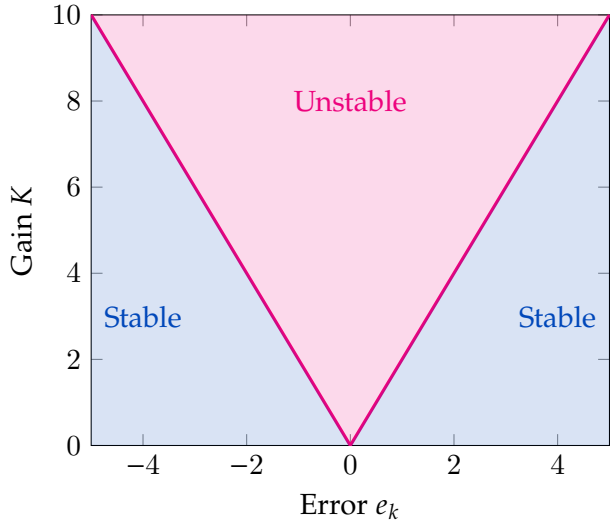


Figure 5.13 Stability condition for the sign controller

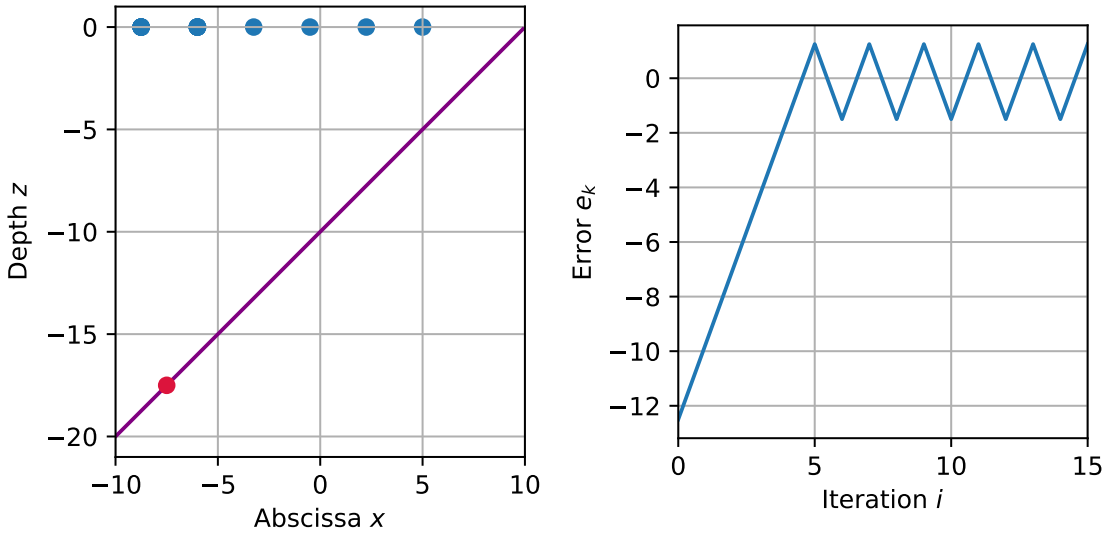


Figure 5.14 Simulation of the sign controller

Proof. The Lyapunov stability of the controller is ensured if the difference $V_{k+1} - V_k$ is negative. The difference is then

$$V_{k+1} - V_k = aK \tanh(e_k) \left(K \tanh(e_k) - 2e_k\right) \tag{5.27}$$
$$= aK \tanh(e_k) \left(K \tanh(e_k) - 2e_k\right). \tag{5.28}$$

□

5.11 Choice of the controller

A controller comparison is shown in Table 5.1, and a choice of the controller is made according to the desired behavior of the system. The dead-beat controller is the most efficient controller, but it requires a perfect knowledge of the system dynamics and the seafloor. The proportional controller is a good compromise between efficiency and robustness, but it deforms cycles a lot for large errors. The sign controller is interesting because

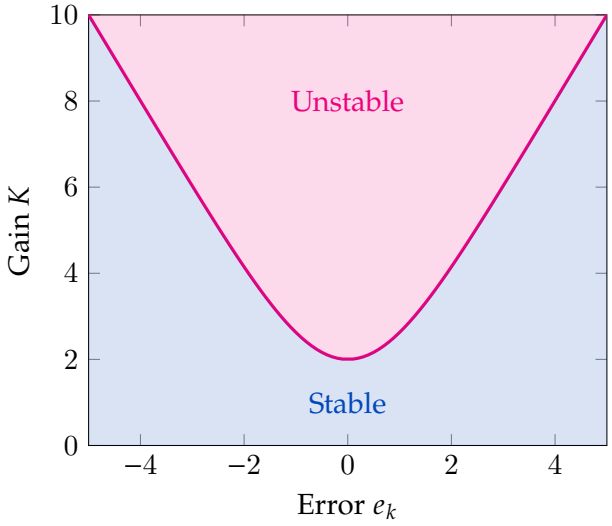


Figure 5.15 Stability condition for the tanh controller

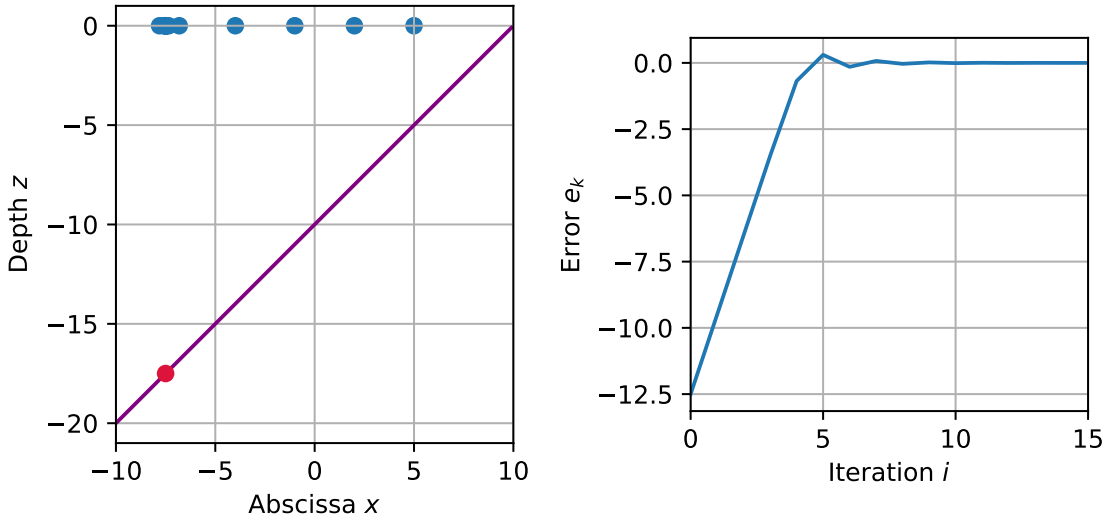


Figure 5.16 Simulation of the tanh controller

of its saturation, and ensures a finite time convergence, but the error only converge in a bounded region around the reference. The tanh controller is a good compromise between the proportional and the sign controller, as it ensures tha convergence of the error toward zero, and the saturation of the input.

Controller	Global stability	Saturation	Convergence
Dead-beat	Yes	No	Finite-Time
Proportional	Yes	No	Asymptotic
Sign	No	Yes	Finite-Time
Tanh	Yes	Yes	Asymptotic

Table 5.1 Comparison of the controllers

5.12 BlueBoat Application

A trial of the square cycle was conducted on the Guerlédan Lake in central Brittany, France. Figure 5.17 shows the Guerlédan departmental nautical base on the Guerlédan Lake where the trials were conducted.

Figure 5.17 Guerlédan departmental nautical base

This area is a perfect place to test the cycle control as the seafloor is well known, as shown in Figure 5.18. This map shows the depth at each point of the lake. Constant depth lines, isobaths, are also shown. This will be used to stabilize the cycle in the environment.

As the lake is a reservoir, its level can vary according to the seasons and the turbines. However, the level varies very slowly compared to the duration of the experiments, and it could be considered constant. The only problem is to reproduce the experiments a few months apart, as the target isobath could no longer be placed at the same location.

The BlueBoat was used to conduct the trials. The BlueBoat is a small autonomous boat developed by Blue Robotics. It is equipped with an inertial unit, a magnetometer, and a GNSS receiver, and has two propellers that are differentially controlled. The boat's hull is designed with two anti-drift surfaces to prevent sway. Its dynamical behavior is then close to the unicycle model. The boat is also equipped with an echosounder to measure the depth below the robot. The boat is shown in Figure 5.19.

For the experiments, GNSS signals were recorded to obtain the ground truth of the trajectory of the robot, but were not used in the control loop. The timed automaton was implemented inside the robot to control itself. The tanh controller was chosen to stabilize the cycle in the environment relative to the $18\ m$ isobath. As the degrees of freedom are decoupled, measurements and control were performed independently along the x-axis and the y-axis. The boat was then able to stabilize itself around the $18\ m$ isobath, and the results are shown in Figure 5.20.

A video of the experiments is available at `https://youtu.be/MDJ6iHYhxyM`. On this video we can see the BlueBoat navigating toward the $18\ m$ isobath, and then stabilizing itself around this isobath. This corresponds to the trial shown in Figure 5.20. The video footage was shot by a drone at fixed altitude, and with a camera pointing down at the scene. The movement of the BlueBoat is visible as the robot moves relative to the shore.

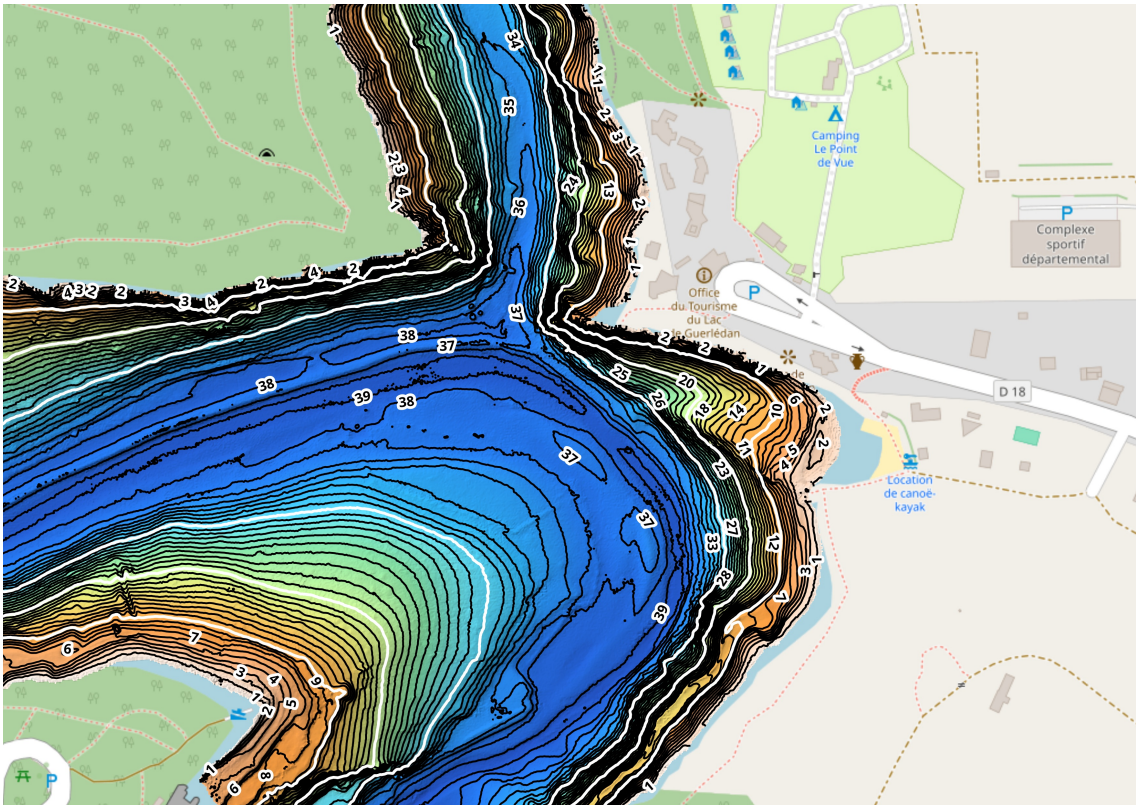


Figure 5.18 Guerlédan Lake seafloor made at ENSTA

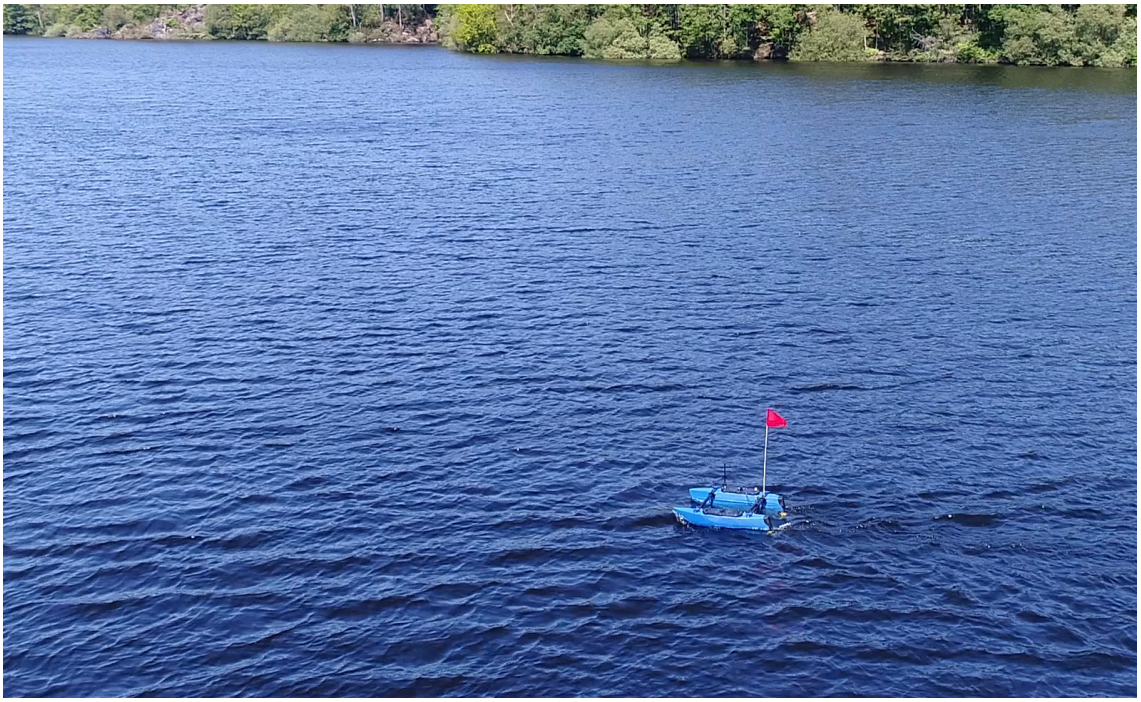


Figure 5.19 BlueBoat on the Guerlédan Lake

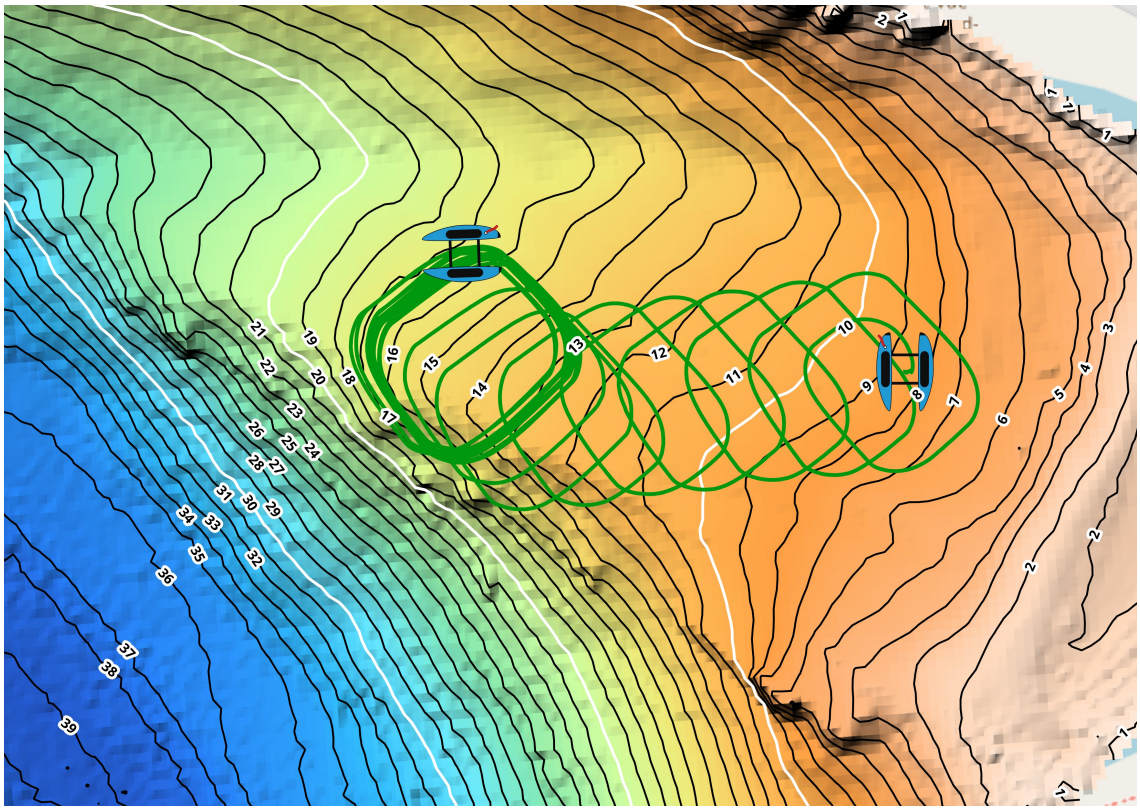


Figure 5.20 BlueBoat navigating using stable cycles

5.13 Conclusion

This chapter has established the mathematical foundation for cycle navigation as a viable alternative to conventional localization-first navigation paradigms. The formal framework developed herein demonstrates that robotic systems associated with a timed automaton can be modeled as discrete dynamical systems, to achieve autonomous navigation in GNSS-denied environments.

The primary theoretical contributions of this work include the formal characterization of the cycle navigation problem within the framework of discrete system theory, the demonstration of system controllability properties, and the design of adaptive control algorithms for cycle stabilization.

The controller design developed in this chapter addresses the fundamental challenge of cycle stabilization through adaptive modification of automaton transition parameters based on sparse environmental measurements. The control strategy operates by iteratively refining cycle parameters to minimize deviation from predefined reference trajectories while maintaining system stability throughout the convergence process. This approach enables the simultaneous resolution of navigation and localization objectives without requiring prior state estimation.

Validation of the theoretical framework has been conducted through comprehensive simulation studies and experimental verification using real robotic platforms. The simulation results demonstrate convergence properties under various initial conditions and environmental configurations, confirming the theoretical predictions of system behavior. Experimental validation conducted on physical robotic systems provides empirical evidence of the practical applicability of cycle navigation in real-world scenarios, validating the transition from theoretical formulation to operational implementation.

The experimental results corroborate the simulation findings, demonstrating successful cycle stabilization and convergence to predefined reference trajectories. These results

validate the fundamental premise that navigation can precede precise localization while maintaining system stability and task completion objectives. Once converged on the stable cycle, the state estimation problem is solved, as the position of the robot is known.

However, the analysis has revealed that system convergence is contingent upon initial condition constraints. The experimental and simulation evidence indicates that certain initial states may lead to divergent behavior, preventing convergence to the desired stable cycle. This observation highlights a critical limitation of the current framework and identifies the boundary conditions that define the domain of convergence for the cycle navigation approach.

The successful implementation of cycle navigation represents a paradigmatic advancement in autonomous robotic systems operating in infrastructure-constrained environments. The methodology eliminates dependency on external positioning references while providing mathematical guarantees of convergence within specific operational domains. The cycle abstraction offers a rigorous theoretical foundation for further development of navigation algorithms that prioritize immediate task execution over precise initial localization.

Future research directions emerging from this work focus on the comprehensive characterization of the basin of attraction for stable cycle convergence. The identification and formal definition of initial conditions that guarantee convergence to stable cyclic trajectories represent critical advancements for practical implementation of cycle navigation systems. This analysis will provide operational guidelines for system initialization and define the operational envelope within which cycle navigation can be reliably deployed.

6 Stability of the cycle navigation

6.1 Introduction

Chapter 5 showed how to control cycles in the world frame using measurements. A candidate control law was found to ensure the stability of the system, and to move the cycle toward a desired position in the world frame described by reference measurements. This chapter will focus on the proof of stability of the cycle on a bathymetric map.

There are mainly two kinds of approaches to prove the stability of a dynamical system. Lyapunov methods consist of finding a positive potential with a negative derivative along the trajectories of the system. By exhibiting such a function, the system is guaranteed to have a neighborhood around the equilibrium point where trajectories converge. However, this method will not prove the global convergence of the system.

Another approach is to find a positive invariant set for the dynamics of the considered system. This set is a set of states where the system will remain once it has entered this set. This proves the existence of a stable set. Then, it is possible to iteratively expand this positive invariant set by adding states that lead to it. This builds the capture basin of the dynamical system, which not only characterizes the region of stability of the system, but also the set of initial states where the system will globally converge to the equilibrium point. The tools developed in Chapter 2, and Chapter 4 will be used to prove the stability of the cycle navigation in this chapter, especially positive invariant sets, and capture basin characterization for dynamical systems.

Figure 6.1 highlights that for the same timed automaton, cycle trajectories could converge towards a stable cycle, or could slide indefinitely on isobaths depending on the initial conditions. This requires the study and characterization of a set of initial conditions which will lead to the convergence of the cycle towards a stable cycle.

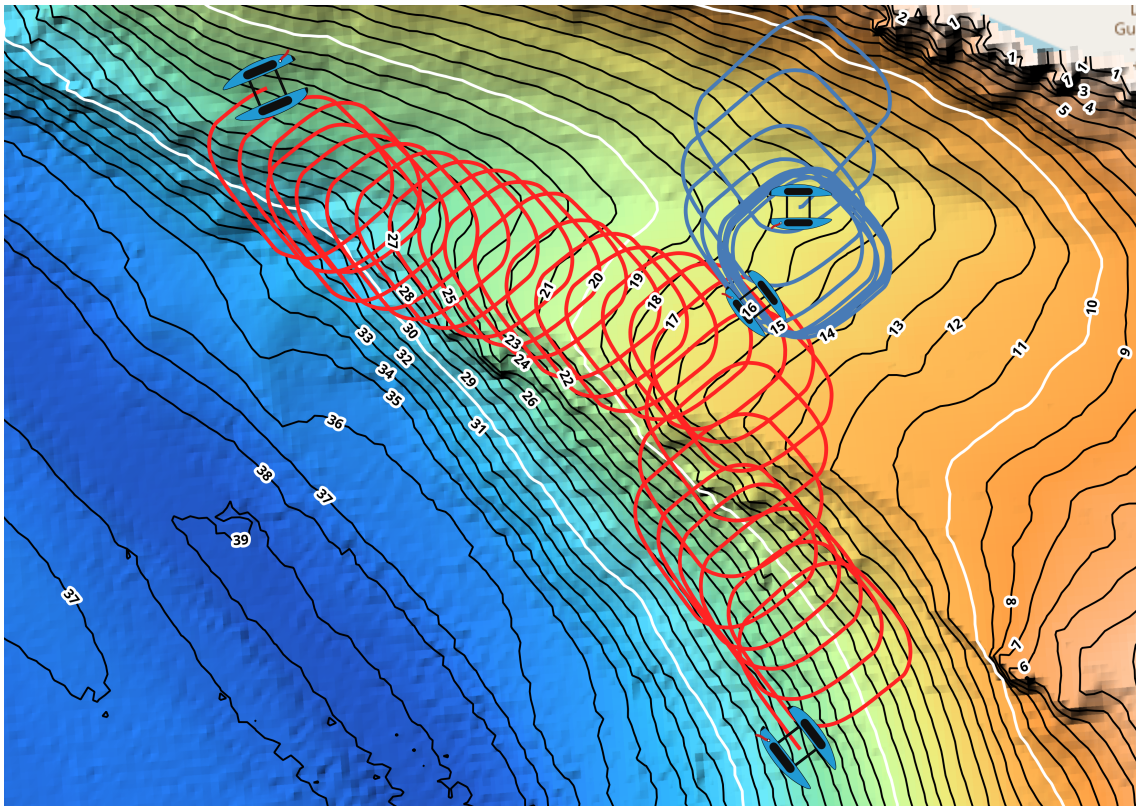
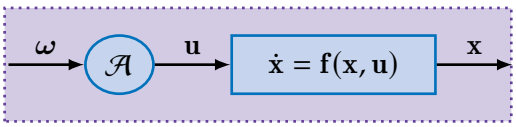
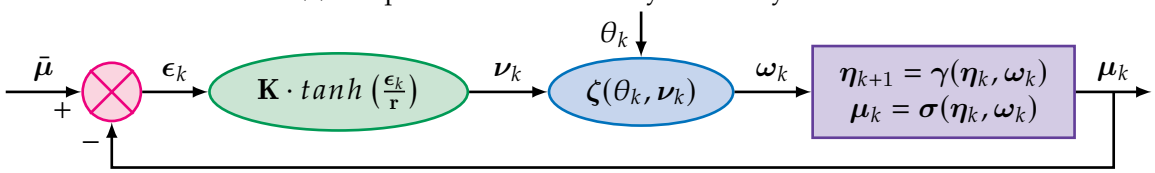


Figure 6.1 Stability of the cycle navigation

We recall here that the considered system is a coupled timed automaton $\mathcal{A}$ and a dynamical system $\mathcal{S}$, with the automaton feeding inputs for the dynamical system, as presented in Chapter 5. Figure 6.2a shows the structure of the coupled automata and robot, while Figure 6.2b shows the control structure of the cycle. These figures show that the control is measurement based, i.e. the reference is $\bar{\mu}$ and the measurements are μ_k. The cyclic state of the system is η_k, which is the state of the cycle at iteration k. This state is not measured nor estimated in this case. Moreover, the goal of this chapter is to prove the stability of the cycle navigation in the cyclic state space, even if the control loop is done in the measurement spare.



(a) Coupled automaton and dynamical system



(b) Controlled cycle

Figure 6.2 Block diagrams of the cycle navigation system

6.2 Cycle iteration stability

First, classical linear theory will be used to study the stability of the cycle. As shown in Chapter 7, the system composed of the timed automaton and the dynamical system follows Equation (5.13). Then, the system is modelled by Equation (6.1), with $\mathbf{A} = \mathbf{I}_n$ and $\mathbf{B} = \mathbf{I}_n$.

$$\boldsymbol{\eta}_{k+1} = \mathbf{A}\boldsymbol{\eta}_k + \mathbf{B}\boldsymbol{\nu}_k \tag{6.1}$$

The stability of the system over an iteration is then studied using the eigenvalues of the matrix $\mathbf{A}$. The system is stable if and only if the eigenvalues of A are in the unit circle. As the matrix $\mathbf{A}$ is the identity matrix, the eigenvalues of $\mathbf{A}$ are all equal to 1. Therefore, the system is marginally stable.

Example 29. *Referring back to the square cycle example, Equation (6.1) shows the evolution of the system. In this equation $\mathbf{A} = \mathbf{I}_3$, and $\mathbf{B} = \mathbf{I}_3$. The eigenvalues of $\mathbf{A}$ are then $\lambda_1 = 1$, $\lambda_2 = 1$, and $\lambda_3 = 1$. To ensure asymptotic stability, eigenvalues should belong to the unit circle. As eigenvalues are exactly on the unit circle, the system is then marginally stable.*

Therefore, under a null input $\boldsymbol{\nu}_k = \mathbf{0}_n$, the system state $\boldsymbol{\eta}_k$ does not evolveg. However, whatever the disturbance applied on the system, it will be destabilized as the state will not naturally remain the same. This highlights the need to implement a suitable control law to have a stable behavior.

It is noticeable here that the discretization of the cycle is not classical. Actually, measurements are performed along the trajectory of the robot, during the k^{th} iteration, but all measurements are transported to the same cyclic state $\boldsymbol{\eta}_k$. Therefore, k is not a time of measurement, but an index of the cycle. Figure 6.3 shows the transport of measurements on the cyclic state.

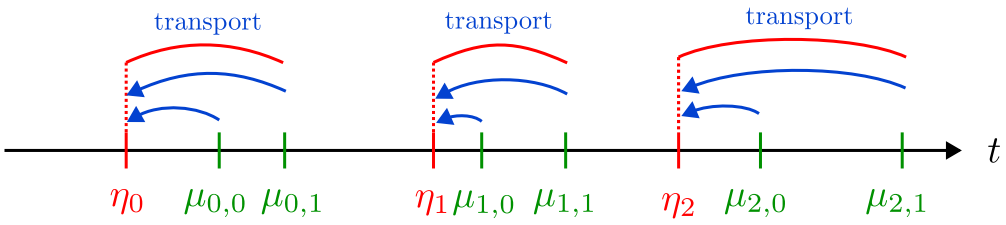


Figure 6.3 Transport of measurements on the cyclic state

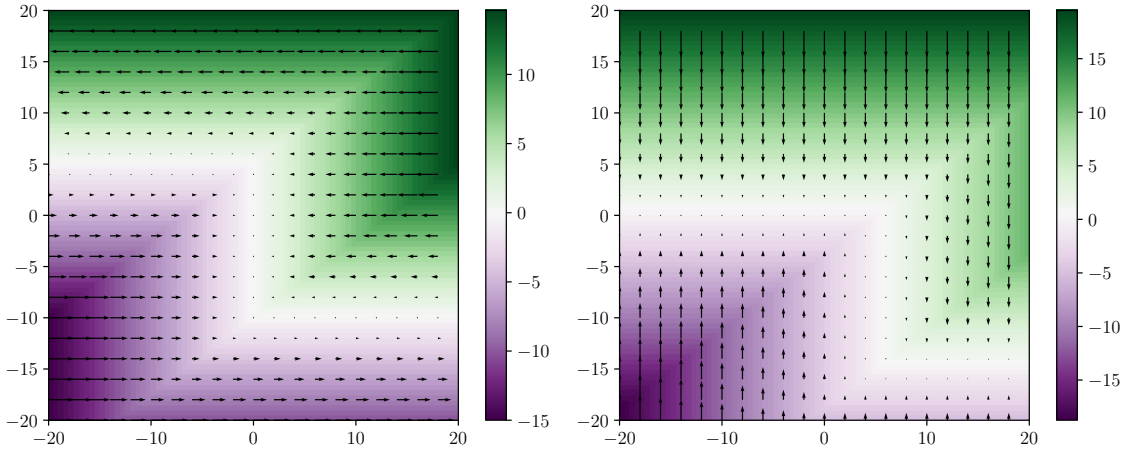
6.3 Stability of the cycle on the bathymetric map

The stability of the cycle on a bathymetric map is now studied. The chosen simulated environment is shown in Figure 5.9a. The cycle is moved in the world frame using the $tanh$ control law described in Chapter 5. The goal is to prove the global stability of the cycle on the bathymetric map.

6.3.1 Vector field of the system

The closed-loop system composed of the dynamical system and controller is now evolving on the bathymetric map. Two measurements $\boldsymbol{\mu}_k = \begin{bmatrix}\mu_0 & \mu_1\end{bmatrix}^T$ are taken along the cycle, μ_0 controls the displacement along the y-axis of the cycle, as shown in Figure 6.4a, and μ_1 controls the displacement along its x-axis, as shown in Figure 6.4b. On these two figures,

the largest is the error, the largest will be the requested displacement of the cycle, with the saturation of the $tanh$ function. The color represents the intensity and the sign of the displacement along the x and y axis.



(a) Influence of μ_0 on the control of the cycle **(b)** Influence of μ_1 on the control of the cycle

Figure 6.4 Vector field along x and y axis

It is noticeable that the vector field depends on measurements to guide the cycle toward a desired state. This phenomenon is more visible in Figure 6.5, which is the sum of the two vector fields related to the two measurements. This vector field seems to present an equilibrium state around $\bar{\eta} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ as all vectors seem to converge toward this state.

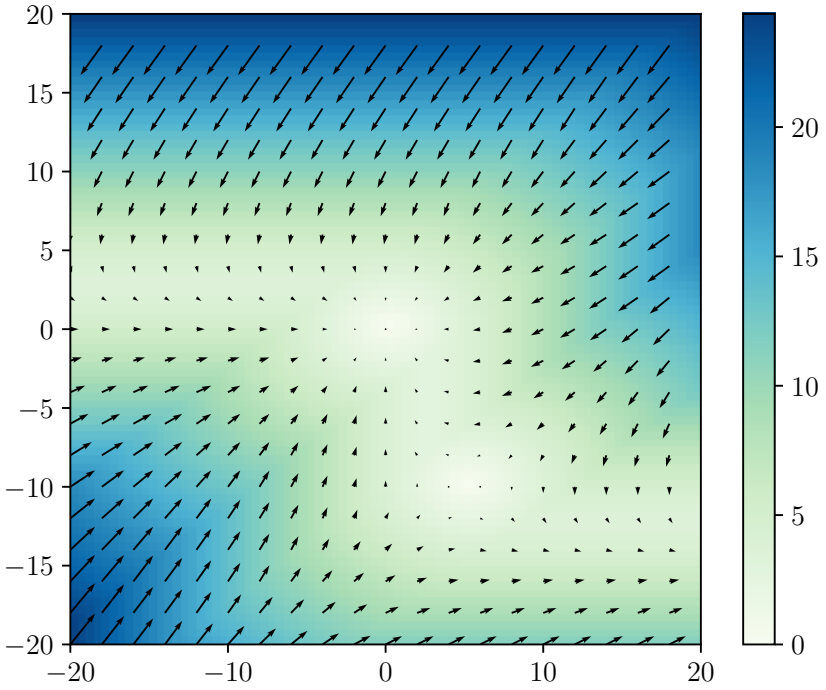


Figure 6.5 Vector field of the controlled cycle

Remark. *Another equilibrium point seems to be at* $\eta = \begin{bmatrix} 5 & -10 \end{bmatrix}$*, but this equilibrium seems to be a hyperbolic fixed point as the vector field induces no displacement only at* $\eta = \begin{bmatrix} 5 & -10 \end{bmatrix}$*, and the vector field tends to move the surrounding points away from each other [73].*

6.3.2 Positive invariant set

The hypothesis formulated above now needs to be verified. If the state around $\bar{\eta} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ is an equilibrium point, it should be possible to exhibit a positive invariant set around this state. We recall here that the regulation is done on measurements μ_k, but that the stability is studied on the cyclic state η_k.

A positive invariant set $\mathbb{P}$ is invariant under the evolution equation of the dynamical system, that is $[\gamma](\mathbb{P}) \subseteq \mathbb{P}$ [5, 82, 9].

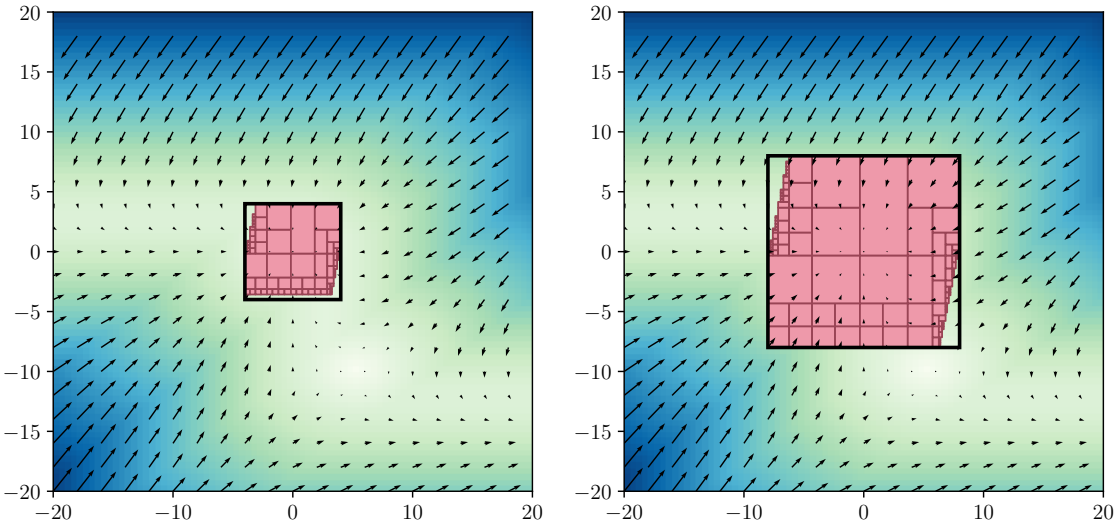
A way to compute an inner approximation of this set is to build a sequence of sets $\mathbb{P}_k$ which will converge towards $\mathbb{P}$. $\mathbb{P}_0$ is initialized by a box of the state space around a supposed stable state, and $\mathbb{P}_{k+1}$ is computed as the intersection of $\mathbb{P}_k$ and $[\gamma](\mathbb{P}_k)$. Therefore, each state which belongs to $\mathbb{P}_k$ and which is moved out of this set by the application of the system dynamics is removed from the solution for $\mathbb{P}_{k+1}$. Thus, the set $\mathbb{P}_k$ is iteratively contracted.

$$\begin{cases} \mathbb{P}_0 = \mathbb{P}_0 \\ \mathbb{P}_{k+1} = \mathbb{P}_k \cap [\gamma](\mathbb{P}_k) \end{cases} \tag{6.2}$$

Then, $\exists n \in \mathbb{N}, \forall k \geq n, [\gamma](\mathbb{P}_k) \subseteq \mathbb{P}_k$. This index n is the minimal index to reach the fixed point, and after this iteration $\mathbb{P}_k$ is positively invariant. $\mathbb{P}_n$ is an inner approximation of a capture basin associated to the starting set $\mathbb{P}_0$.

Figure 6.6 shows the computation of an inner approximation of the largest positive invariant set contained in an initial set $\mathbb{P}_0$. Because of the lattice structure, this largest positive invariant set exists, and could eventually be $\emptyset$. Starting for the set $\mathbb{P}_0$ drawn in black, the inner approximation of the positive invariant set for the cycle navigation is shown in pink.

For the example shown in Figure 6.6a, the starting set $\mathbb{P}_0$ is the box $[-4, 4] \times [-4, 4]$, while it is the box $[-8, 8] \times [-8, 8]$ for the example shown in Figure 6.6b. It is noticeable that this pink set $\mathbb{P}_n$ follows the vector field describing the system dynamics shown in Figure 6.5, and that each point of $\mathbb{P}_n$ enters the pink area by application of the system dynamics. In this example, a paving of the inner approximation of the largest positive invariant set contained inside the initial box $\mathbb{P}_0$ is iteratively computed, as defined in Chapter 4.



(a) $\mathbb{P}_0 = [-4, 4] \times [-4, 4]$ **(b)** $\mathbb{P}_0 = [-8, 8] \times [-8, 8]$

Figure 6.6 Inner approximation of the largest invariant set included in the initial set $\mathbb{P}_0$

The computed set $\mathbb{P}_n$ is the inner approximation of the largest positive invariant of the starting set $\mathbb{P}_0$. Computing the outer approximation of the positive invariant set $\mathbb{P}$ is computationally demanding, as the set of state out of $\mathbb{P}$ should be proved to never enter this set.

Remark. *It is not possible to stop the algorithm before the convergence of the set $\mathbb{P}_k$ on the largest invariant set contained in the initial set $\mathbb{P}_0$, because it could remain solutions which are not meeting the condition $\mathbb{P} \subseteq [\gamma](\mathbb{P})$ while the fix point is not reached.*

6.3.3 Capture basin characterization

The capture basin is a set from which the system cannot escape once it has reached it. This capture basin $\mathbb{B}$ is defined by Equation (6.3) [5, 82, 9].

$$\mathbb{B} = \{x \in \mathbb{B}_0 \mid \exists t_0 \in \mathbb{R}_+, \forall t \geq t_0, x(t) \in \mathbb{B}\} \tag{6.3}$$

As $\mathbb{P}$ is positive invariant, then it is already a capture basin [9, 5]. Then this set $\mathbb{P}_n$ will be the starting set to compute the capture basin $\mathbb{B}$ for the dynamical system.

Using the lattice structure, if $\mathbb{B}$ is a capture basin for a dynamical system governed by an evolution function $\mathbf{f}$, then $\forall k \in \mathbb{N}, \mathbf{f}^{-k}(\mathbb{B})$ is also a capture basin. It comes from the fact that $\forall \mathbf{x} \in \mathcal{S}, \mathbf{f}^k(\mathbf{x}) \in \mathbb{B}$.

It is then possible to expand a capture basin by using the inverse of the evolution function $[\gamma]$ iteratively on a first identified capture basin. By starting from the identified positive invariant set $\mathbb{P}$, it is then possible to compute an inner approximation of a capture basin for the stable cycles by computing the sequence presented in Equation (6.4) [82].

$$\begin{cases} \mathbb{B}_0 = \mathbb{P}_n \\ \mathbb{B}_{k+1} = \mathbb{B}_k \cup [\gamma]^{-1}(\mathbb{B}_k) \end{cases} \tag{6.4}$$

The more iterations are performed to determine the basin of attraction, the larger the basin of attraction becomes. This is because, at each iteration, we add the set of points that enter the basin of attraction by the application of the evolution function $[\gamma]$.

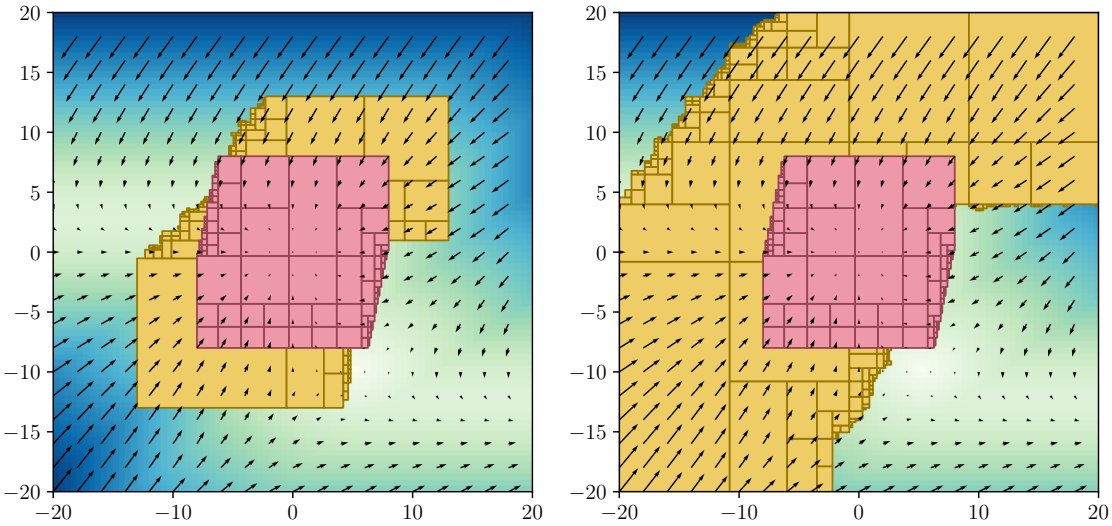
Remark. *Unlike the computation of the positively invariant set $\mathbb{P}$, where the computation could only be stopped when the condition $\mathbb{P}_k \subset [\gamma](\mathbb{P}_k)$ was met, the characterization of the capture basin can be stopped whenever desired. The more iterations are performed, the larger is the characterized area, but stopping computations early does not alter the guarantee of the results.*

Figure 6.7 shows an example of the computed capture basin after $n_1 = 5$ and $n_2 = 20$ iterations from the previously computed positive invariant set $\mathbb{P}$ shown in Figure 6.6b. States added iteratively to the capture basin are shown in yellow, and the starting positive invariant set is shown in pink. With these sets then defined, if the cyclic state enters the yellow area, it will be moved through iterations by the dynamic of the system to the pink area and will be captured forever.

There is obviously points outside the computed capture basin which could be added to it by applying more iterations of the vector field. However, each iteration is time-consuming and while the set $\mathbb{B}_k$ inflates, new states could be added at the next iteration.

6.4 Conclusion

In conclusion, the stability of the cycle on a bathymetric map is proven by the computation of a positive invariant set $\mathbb{P}$ and a capture basin $\mathbb{B}$. The positive invariant set is an intermediate step useful for the stability analysis, and the capture basin highlights the set of initial conditions which will lead to the convergence of the cycle toward a stable cycle. The



(a) 5 iterations of the vector field **(b)** 20 iterations of the vector field

Figure 6.7 Capture basin computation

characterization of the capture basin is helpful in field experiments, as it determines the set of consistent initial conditions for the bathymetric map.

This chapter addressed two fundamental theoretical questions critical to the viability of cycle-based navigation in GNSS-denied environments. Building upon experimental observations that revealed convergent behavior for certain initial conditions and divergent behavior for others, we sought to establish a rigorous mathematical foundation for understanding when and why the proposed navigation method achieves stable operation. Specifically, the chapter aimed to: (1) formally prove the stability of the target cycles by demonstrating the existence of positive invariant sets, and (2) characterize the capture basin that defines the set of initial conditions leading to successful cycle stabilization through successive applications of the system's evolution function.

6.4.1 Stability analysis through positive invariant sets

The first major contribution of this chapter lies in the formal stability proof of the navigation cycles through the construction and analysis of positive invariant sets. By establishing the mathematical framework for these invariant regions, we demonstrated that once a robot trajectory enters the neighborhood of a stable cycle, it remains bounded within a well-defined region around that cycle. This theoretical result provides crucial validation for the experimental observations and establishes that the convergent behavior is not merely empirical but stems from fundamental dynamical properties of the system.

The identification of positive invariant sets also reveals the underlying mechanism that enables robust navigation with minimal exteroceptive sensing. The invariant nature of these regions ensures that measurement uncertainties and system perturbations do not lead to unbounded trajectory drift, thereby guaranteeing that the robot maintains its navigational capability even when operating with sparse sensor feedback. This stability property is essential for practical implementation in challenging environments.

6.4.2 Characterization of the capture basin

The second key contribution involves the characterization of the capture basin. This defines the set of initial conditions that lead to successful cycle stabilization. This characterization is computed using set methods and an initial positive invariant set, by adding iteratively

the set of initial conditions that will reach the union of the initial positive invariant set and the set of states that will enter this positive invariant set. This iterative process allows us to systematically expand the capture basin, providing a clear delineation of the operational limits of the cycle-based navigation system.

Capture basin characterization is particularly valuable for navigation strategies, as it shows the set of initial conditions that will lead to a stable behavior. This is essential to correctly initialize the system, but also to determine during a mission if the robot is fully inside the capture basin of a stable cycle, and then to ensure that the robot will converge toward the stable cycle.

6.4.3 General Conclusions and Future Directions

The theoretical analysis presented in this chapter establishes cycle-based navigation as a mathematically sound approach for autonomous navigation in GNSS-denied environments. The combination of stability guarantees through positive invariant sets and the precise characterization of operational limits through capture basin analysis provides the foundation necessary for confident deployment of this navigation paradigm. These results bridge the gap between empirical observations and theoretical understanding, transforming cycle-based navigation from an experimental concept into a rigorously validated methodology.

The stability properties and capture basin characterization developed in this chapter naturally lead to the development of multi-cycle navigation strategies. With a clear understanding of when individual cycles provide stable navigation and knowledge of their associated operational envelopes, we can now address the challenge of seamlessly transitioning between cycles to achieve long-range navigation objectives. The next chapter will leverage these stability results to develop systematic approaches for cycle-to-cycle transitions, creating a comprehensive navigation framework that combines the local stability of individual cycles with the global coverage achievable through strategic cycle sequences. This progression from single-cycle stability analysis to multi-cycle navigation strategies represents a crucial step toward practical implementation of cycle-based navigation systems in complex, real-world environments.

7 Navigation with cycles

7.1 Introduction

The exploration and navigation of partially known environments represent one of the fundamental challenges in autonomous robotics, particularly in scenarios where Global Navigation Satellite Systems (GNSS) are unavailable or unreliable. Traditional approaches to simultaneous localization and mapping (SLAM) and autonomous exploration typically rely on continuous sensor measurements and allow robots to follow unconstrained trajectories optimized for coverage efficiency, obstacle avoidance, or information gain maximization [90, 11]. These methods, while effective in many scenarios, often require substantial computational resources for real-time localization and mapping, and may struggle in environments with limited or ambiguous sensory information.

In contrast to these conventional paradigms, this chapter introduces a fundamentally different approach to robotic navigation which is the navigation with stable cycles. As the concept of stable cycle is introduced in Chapter 5 and has been proven to be stable in Chapter 6, it is now possible to use this concept for navigation.

The theoretical foundation established in the preceding chapters has demonstrated both the existence and stability of such cyclical navigation patterns. Building upon these theoretical guarantees, this chapter addresses the practical implementation of cycle navigation for systematic environmental exploration. The methodology enables a robot to initially stabilize its position within a first cycle through iterative measurement and trajectory adjustment, subsequently transition to adjacent cycles through carefully planned inter-cycle trajectories, and progressively explore larger regions through a network of

interconnected stable cycles. This approach to environmental exploration differs markedly from traditional frontier-based exploration [107] or information-theoretic planning methods [87]. While these conventional approaches optimize for rapid coverage or maximum information gain, cycle navigation prioritizes localization certainty and systematic coverage through a structured network of stable reference trajectories. The resulting exploration process may require longer traversal times compared to direct path planning methods, as the robot must complete full cycles rather than taking direct routes to unexplored regions. However, this apparent inefficiency is compensated by enhanced robustness to sensor noise, reduced computational complexity for localization, and the ability to operate reliably in environments where traditional SLAM methods might fail due to insufficient or ambiguous sensory information.

The reachability relationship between cycles forms the cornerstone of the exploration strategy, enabling the robot to systematically expand its operational domain while maintaining stable localization references. This chapter presents the mathematical framework for inter-cycle transitions, analyzes the coverage properties of cycle-based exploration, and establishes conditions under which complete area exploration can be guaranteed through the proposed methodology.

Furthermore, the cycle navigation approach offers unique advantages in scenarios requiring periodic revisiting of specific locations, long-term autonomous operation with minimal computational resources, or robust navigation in environments with sparse or unreliable landmarks. The inherent redundancy in cyclical trajectories provides natural fault tolerance, while the structured nature of the exploration process facilitates predictable system behavior and simplified mission planning.

7.2 Leap from cycle to cycle

7.2.1 Leaping to navigate

As soon as the concept of stable cycles is formulated, and the capture basin of these cycles is defined, the idea of using this cycle-based navigation paradigm for navigation in GNSS-denied environments is obvious. Indeed, it seems possible to explore vast areas without getting lost by stabilizing the robot's trajectory on stable cycles along the way. This leads us to define the notion of navigation by leaping from cycle to cycle to explore the environment. This navigation method, of course, can be applied to underwater navigation.

7.2.2 Stabilization condition

The trajectory of the robot is considered stable when the robot is on the stable cycle γ and the measurements match their reference $\bar{\boldsymbol{\mu}}$.

Definition 49. *The trajectory of the robot has **converged** on the stable cycle once measurements of the k^{th} cycle iteration $\boldsymbol{\mu}_k$ are on their reference $\bar{\boldsymbol{\mu}}$. In this case, the measurements satisfy*

$$||\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}_k|| = 0. \tag{7.1}$$

Remark. *Once the convergence is reached for the measurements, the state of the cycle is not evolving as the error $(\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}_k) = \mathbf{0}_n$. For practical purposes, a small value $\epsilon \in \mathbb{R}$ is chosen, and the condition is met when $||\bar{\boldsymbol{\mu}} - \boldsymbol{\mu}_k|| < \epsilon$.*

7.2.3 Dead-reckoning navigation

Definition 50. ***Dead reckoning** is a navigation method which uses proprioceptive sensors to estimate the state of the robot through time. This method uses mainly an accelerometer, a gyroscope,*

a magnetometer, and encoders, and performs numerical integration to estimate the trajectory of the robot [71].

Due to numerical integration errors, dead reckoning is not a perfect method to estimate the trajectory of the robot, and the precision will decrease over time and traveled distance. However, by knowing precisely the initial state of the robot, this method allows navigating for a duration depending on the accuracy of the sensors, bounding the uncertainty of the system state estimate in a bounded set.

This enables the possibility to reach the capture basin of another cycle using dead-reckoning, by knowing the direction and the distance to travel to reach the center of the capture basin of the cycle from a previous stable cycle.

Let $\hat{\mathbb{X}}$ be the current estimated state of the robot, and $\mathbb{B}$ be the capture basin of the cycle γ. To be guaranteed that the robot has reached the capture basin of the cycle, the current estimated state of the robot $\hat{\mathbb{X}}$ should belong to the capture basin of the cycle $\mathbb{B}$, i.e. $\hat{\mathbb{X}} \subseteq \mathbb{B}$. The difficulty is that $\hat{\mathbb{X}}$ is inflating over time, and that it exists some configurations where the robot is not able to ensure that $\hat{\mathbb{X}} \subseteq \mathbb{B}$, as the uncertainty of the estimation is too large.

Remark. *Due to obstacles the dead-reckoning navigation may not follow a straight line, and the robot could have to follow a path that is well suited to its environment.*

7.2.4 Reachability relationship

In the cycle navigation, we consider a directed graph $G = (\Gamma, E)$ where vertices Γ represent distinct navigation cycles and edges E encode direct transitions between cycles. A cycle in this context refers to a closed trajectory that a robot can execute repeatedly, such as patrol routes, inspection circuits, or maintenance loops.

Definition 51 (Cycle Reachability Relationship)**.** *Let $G = (\Gamma, E)$ be a directed graph representing the cycle navigation structure. The reachability relationship $\mathcal{R} \subseteq \Gamma \times \Gamma$ is defined as:*

$$\mathcal{R} = \{(\gamma_0, \gamma_1) \in \Gamma^2 \mid \exists \textit{a directed path} \gamma_0 \textit{to} \gamma_1 \textit{in } G\} \tag{7.2}$$

More formally, $(\gamma_0, \gamma_1) \in \mathcal{R}$ if and only if there exists a sequence of vertices $\gamma_0 = v_0, v_1, \ldots, v_k = \gamma_1$ such that $(v_i, v_{i+1}) \in E$ for all $i \in 0, 1, \ldots, k-1$.

This relationship captures the fundamental notion that a robot operating in cycle γ_0 can potentially transition to operate in cycle γ_1 through a sequence of valid transitions. The non-symmetric nature of $\mathcal{R}$ reflects the directional constraints inherent in robotic navigation systems.

The reachability relationship can be abbreviated by $\gamma_0 \mathcal{R} \gamma_1$, meaning that the cycle γ_1 is reachable from the cycle γ_0, or that $(\gamma_0, \gamma_1) \in \mathcal{R}$.

Property 3. *The reachability relationship $\mathcal{R}$ satisfies:*

(i) ***Reflexivity****: $\forall \gamma \in \Gamma, \gamma \mathcal{R} \gamma$*
(ii) ***Transitivity****: $\forall \gamma_0, \gamma_1, \gamma_2 \in \Gamma^3, \gamma_0 \mathcal{R} \gamma_1 \wedge \gamma_1 \mathcal{R} \gamma_2 \implies \gamma_0 \mathcal{R} \gamma_2$*
(iii) ***Non-symmetry****: $\gamma_0 \mathcal{R} \gamma_1 \nRightarrow \gamma_1 \mathcal{R} \gamma_0$ in general*

Proof. (1) Reflexivity follows from the existence of the trivial path of length zero from any vertex to itself. (2) Transitivity follows from path concatenation: if there exists a path from γ_0 to γ_1 and a path from γ_1 to γ_2, then concatenating these paths yields a path from γ_0 to γ_2. (3) Non-symmetry is demonstrated by considering any directed acyclic subgraph within G. Depending on environment constraints, such as currents or obstacles, it is possible to have a directed path from γ_0 to γ_1 without a corresponding path from γ_1 to γ_0. □

An illustration of this concept is shown in Figure 7.1, where the capture basin of the cycle γ_0 is small and not easily reachable from the cycle γ_1, as it is smaller than the capture basin of the cycle γ_1.

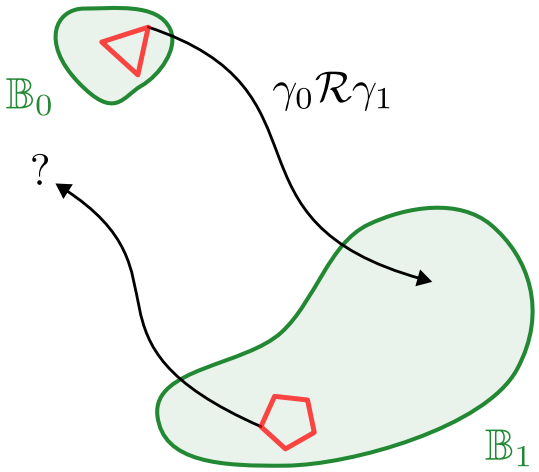


Figure 7.1 Example of a non-symmetrical reachability relationship

Example 30. *Figure 7.2 shows an example of a reachability relationship graph between cycles. The reflexivity property of the reachability relationship is visible as for each cycle γ_i, there is a reachability relation r_j from γ_i to γ_i. There is also reachability relationship between some cycles γ_i and γ_j which are defined by the relationships r_k. The reachability relationship graph is a directed graph, as the reachability relationship is not symmetrical.*

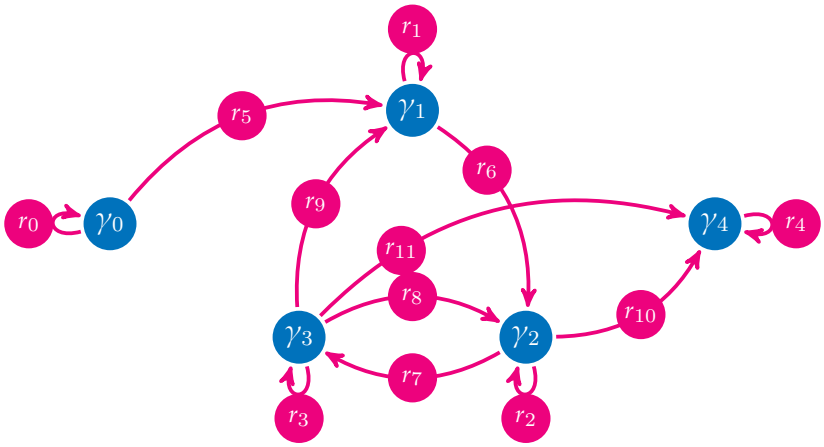


Figure 7.2 Reachability relationship graph

Stable cycles can have properties of starting cycles and recovering cycles. These properties are fundamental to design navigation strategies to explore the environment, as starting cycles are not reachable, so the cycle as to be initialized on this cycle if it is mandatory to reach it, and recovering cycles are terminal cycles, so the robot should reach them only at the end of the mission as it is not possible to leave them once reached. These latter cycles are useful to stabilize the robot at the end of the mission, and to recover it from the surface vehicle.

Definition 52. *A **starting cycle** γ_0 is a cycle without any cycle able to reach this cycle. Therefore, starting cycles satisfy the condition*

$$\nexists \gamma \in \Gamma, \gamma \mathcal{R} \gamma_0. \tag{7.3}$$

Definition 53. *A **recovering cycle** γ_0 is a cycle unable to reach any other cycles. Therefore, recovering cycles satisfy the condition*

$$\nexists \gamma \in \Gamma, \gamma_0 \mathcal{R} \gamma. \tag{7.4}$$

7.3 BlueBoat Application

A switch between two cycles on the Guerlédan Lake was performed using the BlueBoat. An area with two possible cycles was found on the lake. This area presents an isobath with two candidate corners to stabilize a square cycle. The robot is first placed in the capture basin of the first cycle $\mathbb{B}_0$, and follows the timed automaton to converge to the cycle γ_0. Then, once stabilized, the robot reaches the capture basin of the next cycle $\mathbb{B}_1$ in dead-reckoning, by knowing the direction and the distance to travel to reach the center of the capture basin $\mathbb{B}_1$ of the cycle γ_1. Once in this set, the robot follows the second timed automaton designed to converge to the cycle γ_1.

Figure 7.3 shows the GNSS trace of the robot during the experiment. This experiment demonstrates the feasibility of switching between stable cycles is proven, and this is a first step toward the navigation in GNSS denied environment using stable cycles. By extending this concept to more cycles, it could be possible to extensively explore an area without localization. Moreover, this concept is perfectly suitable for underwater navigation.

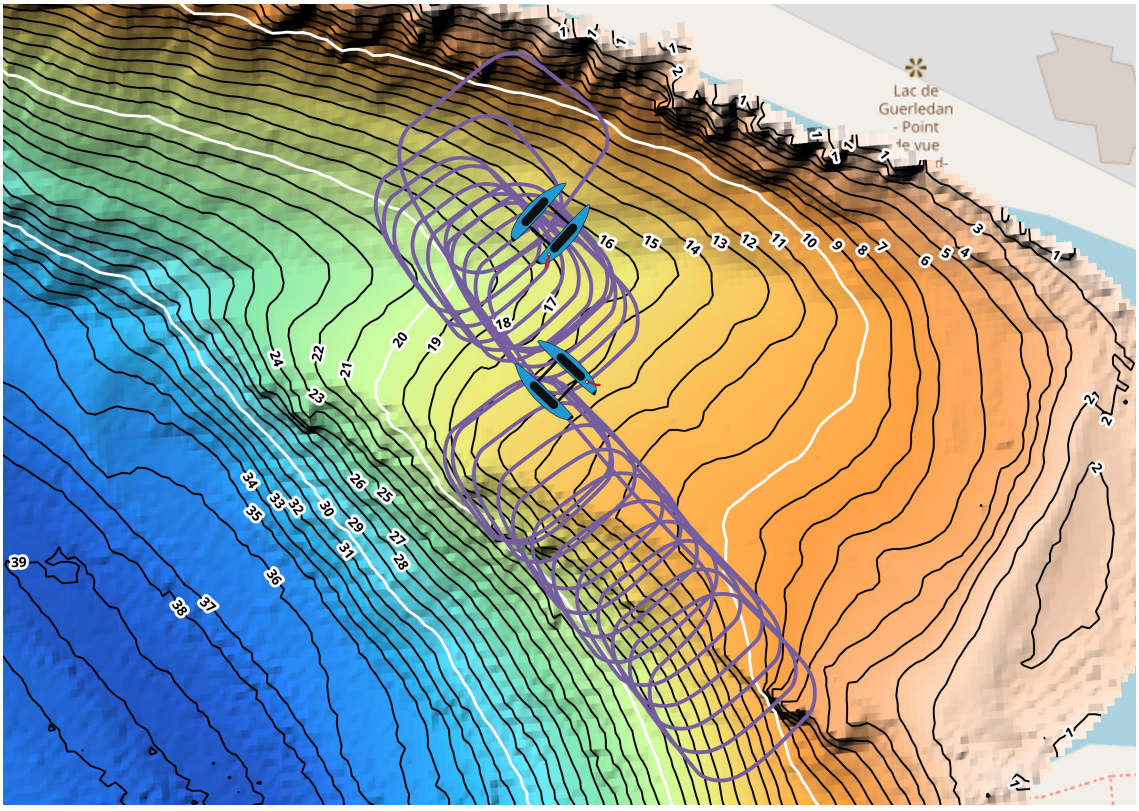


Figure 7.3 BlueBoat switch between cycles

7.4 Cycles and worlds exploration

7.4.1 Graph of relationship

By analyzing the reachability relationship between cycles, it is noticeable that navigation between some cycles is possible, and others are not. This is understandable when the concepts of starting and recovering cycles are introduced.

Figure 7.4 shows an example of strongly connected components of the reachability relationship graph. In this example, the cycles γ_1, γ_2, and γ_3 are strongly connected, as it is possible to reach each other in a loop. Once the robot is in one of these cycles, it is not possible to return to cycle γ_0. If the robot reaches the cycle γ_4, it is not possible to return to cycles γ_1, γ_2, and γ_3.

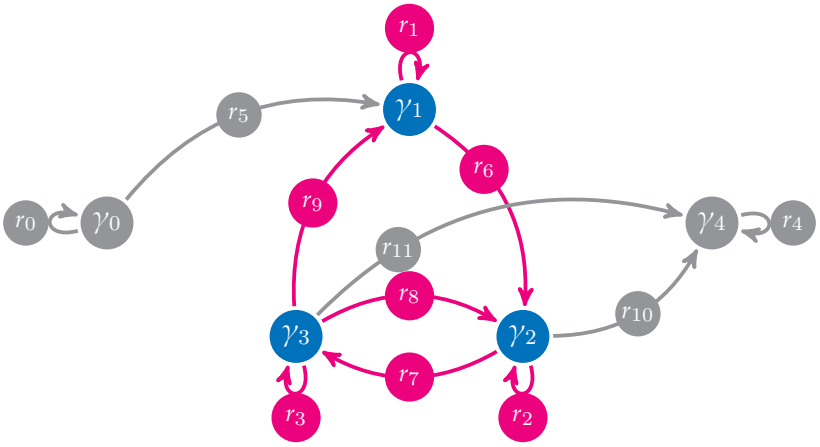


Figure 7.4 Strongly connected subset of the reachability relationship graph

Remark. *There are strategies to guide the robot to explore its environment, as transiting between two cycles will let the robot navigate and perform measurements in different areas. For instance, to reach the cycle γ_3 from the cycle γ_2, the robot should use the relationship r_7, but to reach the cycle γ_2 from the cycle γ_3, the robot can use the relationship r_8, or use r_9 to reach the cycle γ_1 and then r_6 to reach the cycle γ_2. This let the robot explore different areas of the environment through cycle transitions.*

7.4.2 Concept of worlds

Some cycles are then not reachable from other cycles. Moreover, once the robot is operating in an area in which it is able to converge to stable cycles, the robot will not be able to safely reach some other cycles again. A tool to analyze this ability to reach certain cycles is the concept of worlds.

The reachability relationship $\mathcal{R}$ naturally induces an equivalence relation that captures the notion of mutually reachable cycles.

Definition 54. *Two cycles are **mutually reachable** if they can reach each other through the reachability relationship $\mathcal{R}$. Formally, we define the relation $\mathcal{R}$ on the set of cycles Γ as follows:*

$$\gamma_0 \sim \gamma_1 \Leftrightarrow \gamma_0 \mathcal{R} \gamma_1 \wedge \gamma_1 \mathcal{R} \gamma_0 \tag{7.5}$$

Theorem 7. *The relation $\sim$ is an equivalence relation on Γ.*

Proof. We must verify the three properties of equivalence relations:

Reflexivity: For any $\gamma_0 \in \Gamma$, we have $\gamma_0 \mathcal{R} \gamma_0$ by Property 3, therefore $\gamma_0 \mathcal{R} \gamma_0 \wedge \gamma_0 \mathcal{R} \gamma_0$, which implies $\gamma_0 \sim \gamma_0$.

Symmetry: If $\gamma_0 \sim \gamma_1$, then by definition $\gamma_0 \mathcal{R} \gamma_1 \wedge \gamma_1 \mathcal{R} \gamma_0$. This is equivalent to $\gamma_1 \sim \gamma_0 \wedge \gamma_0 \sim \gamma_1$, which implies $\gamma_1 \sim \gamma_0$.

Transitivity: Suppose $\gamma_0 \sim \gamma_1$ and $\gamma_1 \sim \gamma_2$. Then:

$$\gamma_0 \mathcal{R} \gamma_1 \wedge \gamma_1 \mathcal{R} \gamma_0 \quad \text{(from } \gamma_0 \sim \gamma_1\text{)} \qquad \gamma_1 \mathcal{R} \gamma_2 \wedge \gamma_2 \mathcal{R} \gamma_1 \quad \text{(from } \gamma_1 \sim \gamma_2\text{)} \tag{7.6}$$

By transitivity of $\mathcal{R}$: $\gamma_0 \mathcal{R} \gamma_1 \wedge \gamma_1 \mathcal{R} \gamma_2 \implies \gamma_0 \mathcal{R} \gamma_2$

Similarly: $\gamma_2 \mathcal{R} \gamma_1 \wedge \gamma_1 \mathcal{R} \gamma_0 \implies \gamma_2 \mathcal{R} \gamma_0$ Therefore $\gamma_0 \mathcal{R} \gamma_2 \wedge \gamma_2 \mathcal{R} \gamma_0$, which means $\gamma_0 \sim \gamma_2$. □

We want now to use this equivalence relation to link cycles by group of cycles that are mutually reachable. The equivalence classes of this relation will be used to define the concept of Worlds.

Definition 55. *Let $[\gamma] = \{\gamma_0 \in \Gamma \mid \gamma_0 \sim \gamma\}$ denote the equivalence class of γ under $\sim$. The **quotient graph** $G/\sim$ is defined by:*

$$\Gamma/\sim = \{[\gamma] : \gamma \in \Gamma\} \quad \textit{(the set of equivalence classes)} \tag{7.7}$$

$$E/\sim = \{([\gamma_0], [\gamma]) : \gamma_0 \nsim \gamma \wedge \gamma_0 \mathcal{R} \gamma\} \tag{7.8}$$

Definition 56. *A **strongly connected component** of G is a maximal set of vertices $S \subseteq \Gamma$ such that for every pair $\gamma_0, \gamma_1 \in S$, there exists a directed path from γ_0 to γ_1 and a directed path from γ_1 to γ_0.*

Theorem 8. *The strongly connected components of G correspond exactly to the equivalence classes under $\sim$.*

Proof. Let S be a strongly connected component and let $[\gamma]$ be an equivalence class.

($\Rightarrow$) If S is a strongly connected component, then for any $\gamma_0, \gamma_1 \in S$, there exist directed paths from γ_0 to γ_1 and from γ_1 to γ_0. This means $\gamma_0 \mathcal{R} \gamma_1$ and $\gamma_1 \mathcal{R} \gamma_0$, so $\gamma_0 \sim \gamma_1$. By maximality of S, we have $S = [\gamma]$ for any $\gamma \in S$.

($\Leftarrow$) If $[\gamma]$ is an equivalence class, then for any $\gamma_0, \gamma_1 \in [\gamma]$, we have $\gamma_0 \sim \gamma_1$, which means $\gamma_0 \mathcal{R} \gamma_1$ and $\gamma_1 \mathcal{R} \gamma_0$. This implies the existence of directed paths in both directions. By maximality of equivalence classes, $[\gamma]$ forms a strongly connected component. □

Tarjan's algorithm efficiently computes strongly connected components using a single depth-first search traversal with auxiliary data structures to track low-link values [88].

The algorithm maintains the following data structures:

- visited[γ]: Boolean array indicating if vertex γ has been visited
- disc[γ]: Discovery time of vertex γ
- low[γ]: Low-link value of vertex γ
- onStack[γ]: Boolean array indicating if vertex γ is on the stack
- stack: Stack containing vertices of the current SCC being processed

Algorithm 7 Tarjan's SCC Algorithm for Cycle Navigation

```
1: function TarjanSCC(G = (Γ, E))
2:     Initialize all arrays and stack
3:     time ← 0
4:     SCCs ← ∅
5:     for each vertex γ ∈ Γ do
6:         if ¬visited[γ] then
7:             TarjanDFS(γ)
8:         end if
9:     end for
10:    return SCCs
11: end function
12: function TarjanDFS(γ)
13:    visited[γ] ← true
14:    disc[γ] ← low[γ] ← time + +
15:    stack.push(γ)
16:    onStack[γ] ← true
17:    for each edge (γ0, γ1) ∈ E do
18:        if ¬visited[γ1] then
19:            TarjanDFS(γ1)
20:            low[γ0] ← min(low[γ0], low[γ1])
21:        else if onStack[γ1] then
22:            low[γ0] ← min(low[γ0], disc[γ1])
23:        end if
24:    end for
25:    if low[γ0] = disc[γ0] then
26:        SCC ← ∅
27:        repeat
28:            γ1 ← stack.pop()
29:            onStack[γ1] ← false
30:            SCC.add(γ1)
31:        until γ1 = γ0
32:        SCCs.add(SCC)
33:    end if
34: end function
```

Theorem 9. *Tarjan's algorithm runs in $O(|\Gamma| + |E|)$ time and uses $O(|\Gamma|)$ space.*

Proof.
- Each vertex is visited exactly once during the DFS traversal: $O(|\Gamma|)$
- Each edge is examined exactly once: $O(|E|)$
- Stack operations are performed at most once per vertex: $O(|\Gamma|)$
- Space complexity is dominated by the recursion stack and auxiliary arrays: $O(|\Gamma|)$

The linear time complexity makes this approach highly suitable for large-scale robotics applications where cycle networks may contain thousands of nodes. □

This concept of strongly connected components of a graph can be applied to the reachability graph of cycles, where each cycle is a vertex and the reachability relationship is a directed edge. The strongly connected components then represent sets of cycles that can reach each other, forming a coherent navigation area.

Definition 57. *A **World** is a subset of strongly connected components of the reachability graph of cycles.*

By decomposing the reachability graph of cycles into Worlds, it is possible to characterize if transiting between two cycles will be irreversible or not, as it exists no reachability relationship to come back in a part of the graph of cycles. Figure 7.5 shows an example of a world concept with strongly connected components. Each cycle is represented by a blue circle, and reachability relationships with black arrows. The world concept is then represented by a yellow circle, which is a subset of strongly connected components of the reachability graph of cycles. There are many Worlds in this example with one-way relationships that cannot be revisited. Then, strategies must be adopted when trying to explore different worlds to avoid being stuck in one of them.

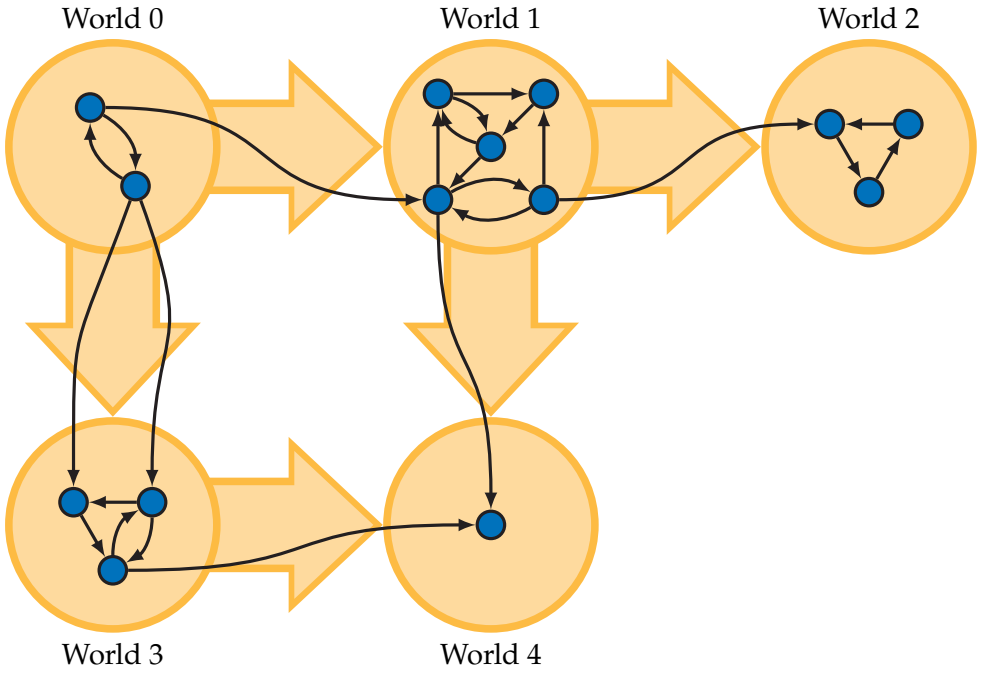


Figure 7.5 World concept with strongly connected subsets

Definition 58. *A **starting world** $\mathcal{W}_s$ in the set of worlds $\mathcal{W}$, is a world such that no other world can reach it, in the sense of the reachability relationship $\mathcal{R}$. Therefore, starting worlds satisfy the condition*

$$\nexists \boldsymbol{\mathcal{W}} \in \mathcal{W}, \boldsymbol{\mathcal{W}}_s \mathcal{R} \boldsymbol{\mathcal{W}}_0. \tag{7.9}$$

Definition 59. *A **recovering world** $\mathcal{W}_r$ in the set of worlds $\mathcal{W}$, is a world such that it cannot reach any other world, in the sense of the reachability relationship $\mathcal{R}$. Therefore, recovering worlds satisfy the condition*

$$\nexists \boldsymbol{\mathcal{W}} \in \mathcal{W}, \boldsymbol{\mathcal{W}}_0 \mathcal{R} \boldsymbol{\mathcal{W}}_r. \tag{7.10}$$

7.5 When dead reckoning is not sufficient

When distance between cycles is large, and environments are complex, the dead-reckoning navigation between stable cycles may not be sufficient to ensure a robust navigation. Instead of full dead-reckoning, other cyclic strategies could be set up, such as isobath bouncing.

Suppose an isobath connecting two areas of interest that the robot should explore. The isobath is considered approximately straight with smooth variations in the two-dimensional plane. To guide it along the isobath, the robot follows a simple automaton shown in Figure 7.6.

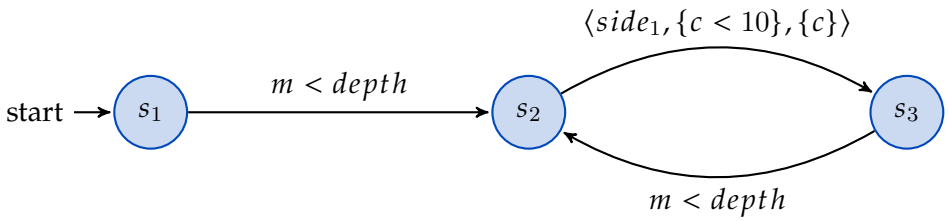
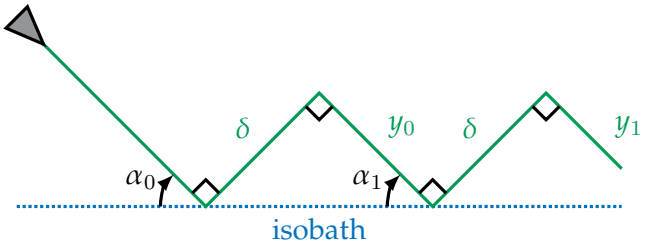


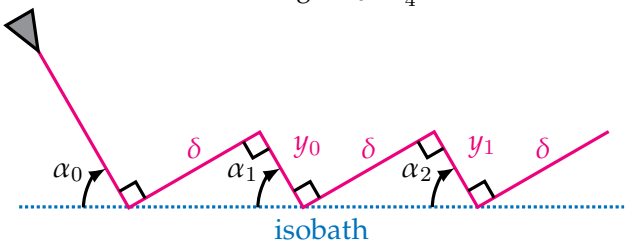
Figure 7.6 Isobath bounce automaton

Figure 7.7 shows the trajectory of a robot following the timed automaton designed for isobath following shown in Figure 7.6. The robot is initially placed at an unknown position and angle α_0 relative to the isobath, but it is assumed that the trajectory of the robot will cross the isobath at some time. The robot performs continuous measurements of the depth below itself m. At the time the robot reaches the isobath, the condition $m < depth$ is satisfied, and the robot is assumed to be initialized. The robot executes a right-angle turn. The timed automaton enters the state s_2, and the robot navigates for a duration δ. Then the state machine switches to the state s_3 and the robot executes a right-angle turn in the other way to reach back the isobath. The navigation duration of this last segment is denoted by y_k and is measured by the robot.

In the case the robot has an initial angle $\alpha_0 = \frac{\pi}{4}$, the navigation duration of the two segments at each iteration will be the same, in other words $y_k = \delta$. Figure 7.7a shows the trajectory of the robot in the case the initial angle is $\alpha_0 = \frac{\pi}{4}$. If the initial angle is different from $\frac{\pi}{4}$, then the navigation duration of the two segments will be different. Figure 7.9 shows the trajectory of the robot in the case the initial angle is $\alpha_0 = \frac{\pi}{3}$.



(a) Initial angle $\alpha_0 = \frac{\pi}{4}$



(b) Initial angle $\alpha_0 = \frac{\pi}{3}$

Figure 7.7 Long-range navigation using isobath bounce

Through iterations and using measurements y_k the robot is able to correct the initial angle for the next iteration. Then from an unknown initial angle it is able to converge to $\alpha_k = \frac{\pi}{4}$.

It is possible to give the expression of the angle α_k depending on the durations δ, y_k, and the geometric parameters of the system:

$$tan(\alpha_k) = \frac{y_k}{\delta}. \tag{7.11}$$

Figure 7.8 shows the plot of the ratio $\frac{y_k}{\delta}$ as a function of α_k. It is noticeable that this function crosses the line $y = x$ at $\alpha_k = \frac{\pi}{4}$, which correspond to the initial angle where $y_k = \delta$.

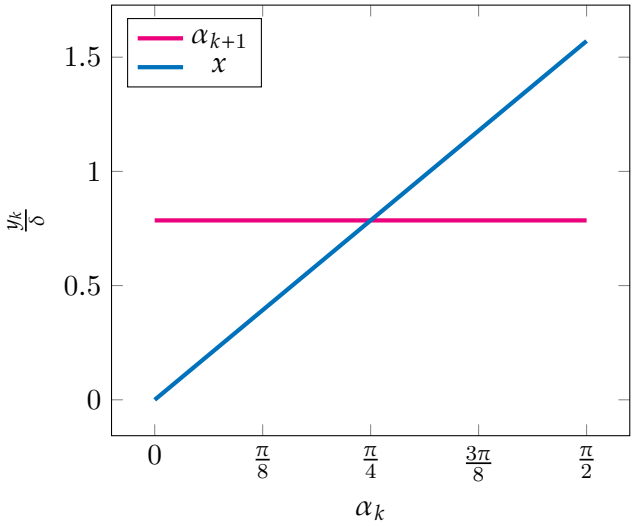


Figure 7.8 Evolution of $\frac{y_k}{\delta}$ as a function of α_k

A controller is designed to ensure that y_k tends to δ, and so that α_k tends to $\frac{\pi}{4}$, to guide the robot along the isobath, even if the isobath is not exactly straight. Instead of turning by an angle of $\frac{\pi}{2}$, the robot will turn by an angle $\frac{\pi}{4} + tan^{-1}\left(\frac{y_k}{\delta}\right)$ to correct the angle α_k at each iteration. This dead-beat controller ensures that the angle α_k will evolve following

$$\alpha_{k+1} = \alpha_k + \left(\frac{\pi}{4} - tan^{-1}\left(\frac{y_k}{\delta}\right)\right). \tag{7.12}$$

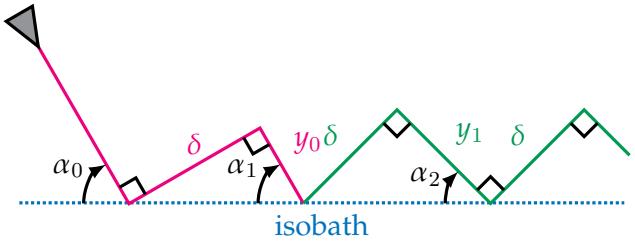


Figure 7.9 Dead-beat controller for isobath bounce

This method is designed for a straight isobath.Actually, isobaths rarely remain straight but vary smoothly. In this case, the controller is robust enough to ensure that the robot trajectory will follow the isobath, even with small variations.

When designing this kind of navigation, a trade-off has to be made between the robustness of the method and travel efficiency. In fact, to enhance forward motion, the angles between the isobath and the robot's trajectory need to be small, so that at each iteration the robot advances a consistent distance along the isobath. If the isobath has varied too abruptly, the robot may lose the isobath and not be able to follow it anymore.

On the other hand, to make this method robust, the angle between the robot's trajectory and the isobath needs to be large, so that the robot moves farther away from the isobath at each iteration. This allows the isobath to change abruptly and the navigation to be robust, but as a trade-off, the progress at each iteration is small.

Theorem 10. *If the selected isobath is l-Lipschitz, and the robot follows the long-range isobath bounce method described above, then the trajectory should meet the condition:*

$$l < tan(\alpha_k). \tag{7.13}$$

Proof. If the robot follows the isobath bounce method, and to ensure that the isobath always remains on the same side of the trajectory to avoid its loss, the ratio of the standoff distance between the isobath and the robot to the distance the robot travels along the isobath must be greater than the variation of the isobath along this distance.

The standoff distance between the isobath and the robot is $y = \delta sin(\alpha_k)$, and the traveled distance along the isobath is $x = \delta cos(\alpha_k)$. Then, by ensuring that the l-lipschitz coefficient meets $l < tan(\alpha_k)$, the robot will never lose the isobath. □

This experiment has been conducted on the Guerlédan Lake using the same Uncrewed Surface Vehicle as in the dead-reckoning navigation. The robot is placed at an unknown position and angle relative to the 30 m isobath of the lake. The robot then follows the timed automaton designed to navigate using isobath bounce.

Figure 7.10 shows the trajectory of the robot following the isobath bounce method. Two trials were performed with different initial conditions. The robot is able to converge to the 30 m isobath of the lake in a first part of the experiment, then it follows it along a large distance without losing it.

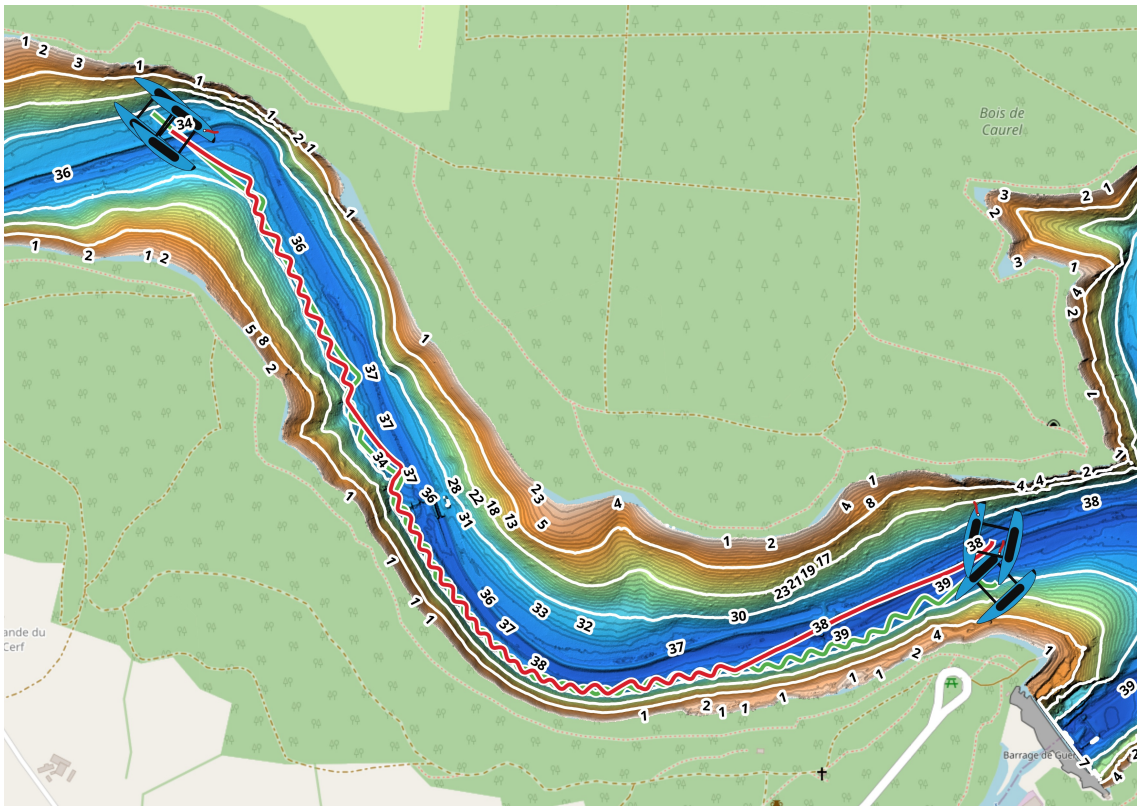


Figure 7.10 Isobath bounce navigation trial on the BlueBoat

Figure 7.11 shows the beginning of the experiment. Independently of the initial position and angle of the robot, the robot is able to converge to the isobath in the two experiments. It is noticeable that the reached position is not the same for the two experiments, but it is not an issue for the rest of the experiment.

Figure 7.12 shows that in the two experiments, the robot is able to follow the isobath along a large distance, even if the isobath is not straight. The two trajectories are similar, but are not exactly the same.

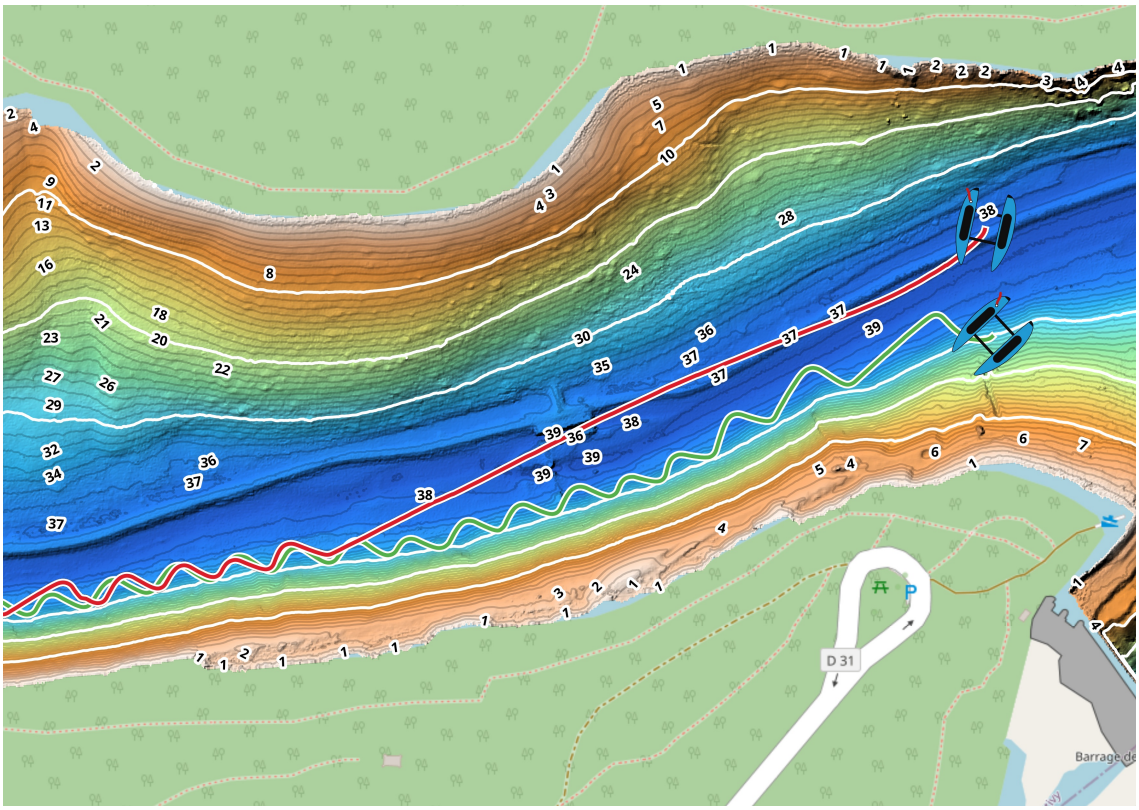


Figure 7.11 Start of the experiment

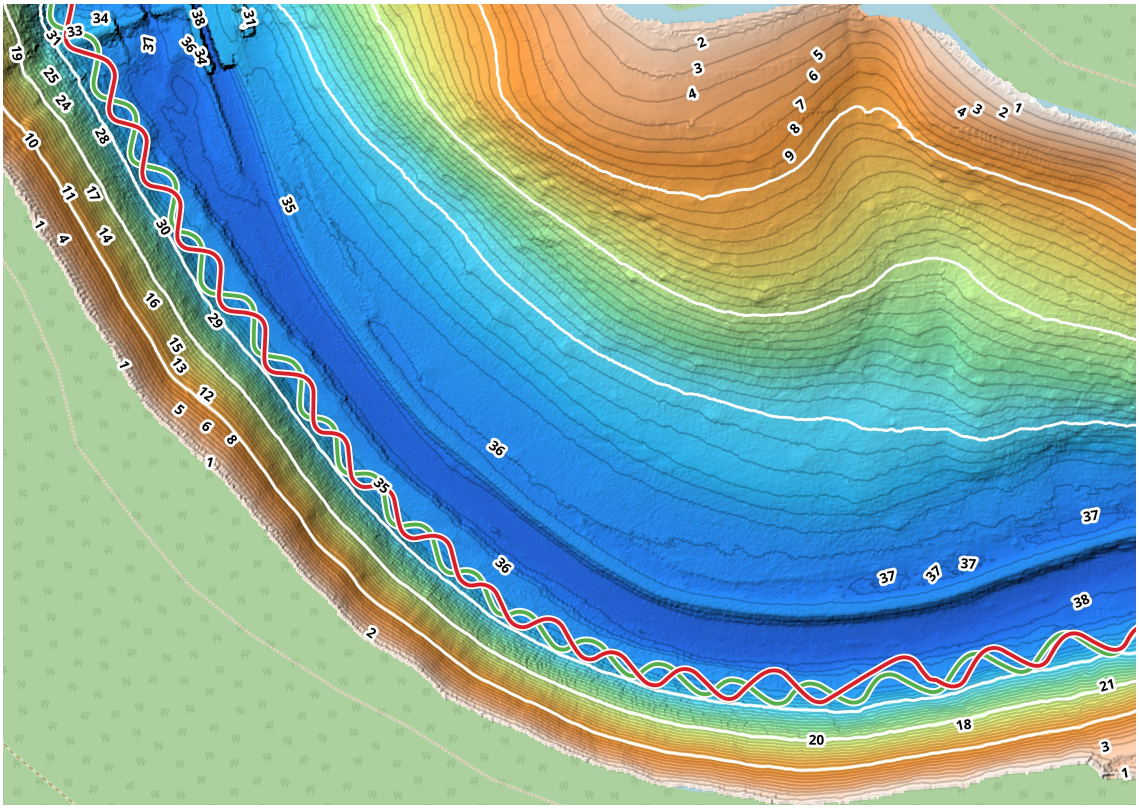


Figure 7.12 A robust navigation along the isobath

Figure 7.13 shows the end of the experiment. First, it is noticeable that the two trajectories lose the isobath due to a sharp variation of the isobath. The robot is then not

able to bounce on the isobath anymore, and the navigation method can no longer be used. However, the two trials are end at approximately the same position on the lake, despite the different initial conditions. This method is then robust enough to reach the capture basin of another cycle which could be in the neighborhood of the end of the experiment.

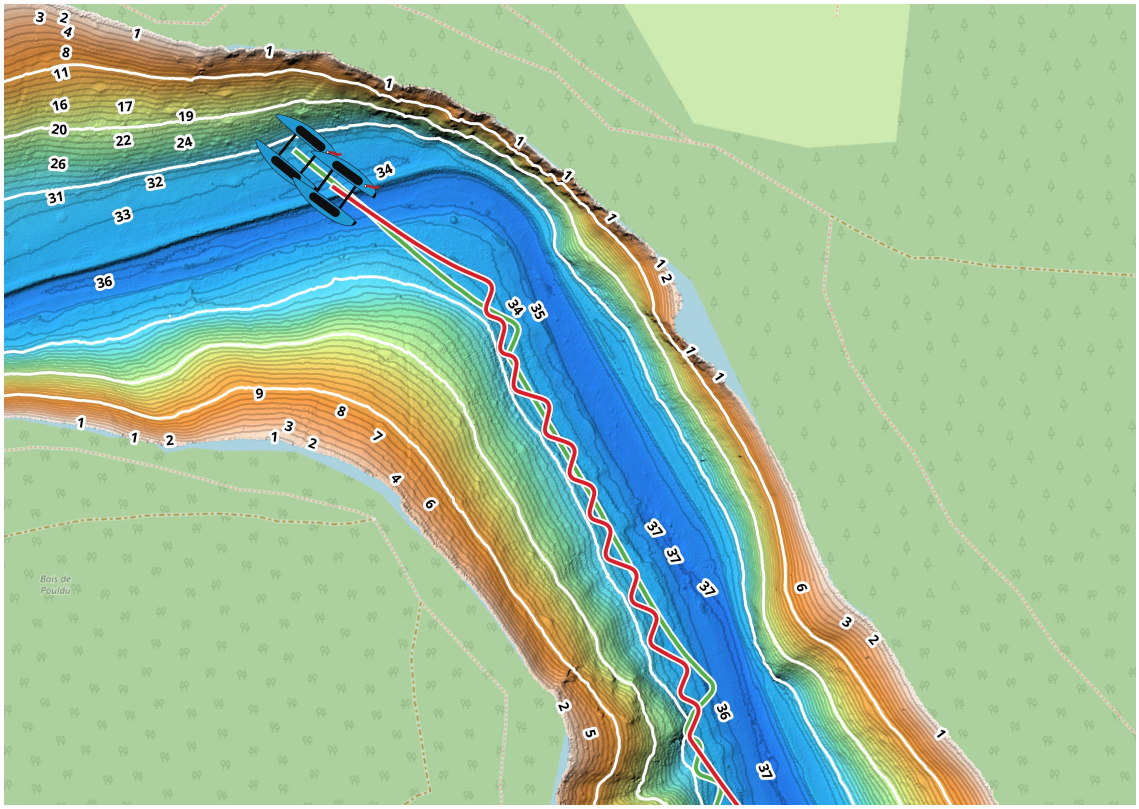


Figure 7.13 The end of the experiment

A condition to detect the end of the experiment could be used to stop this navigation and switch to a cycle stabilization. This condition could be: if the duration in the state s_3 exceeds a certain threshold then the robot is considered to have lost the isobath, and the navigation is stopped.

This method is robust enough to ensure that the robot will be able to follow the isobath, even if it is not straight. It will add a new navigation strategy to the cycle navigation framework, allowing the robot to explore large distances between stable cycles without relying on dead reckoning. This isobath bounce method allows new reachability relationships $\mathcal{R}$ between cycles, and then enriches the cycle navigation framework with new possibilities to explore the environment.

7.6 Conclusion

This chapter has presented a comprehensive navigation framework based on cycle navigation theory, specifically designed for autonomous operation in GNSS-denied environments such as underwater robotics applications. The developed approach addresses the fundamental challenge of maintaining accurate positioning and navigation capabilities when traditional satellite-based systems are unavailable.

The navigation strategy introduced leverages the inherent stability properties of cycle navigation through a two-tier approach. First, stable cycles serve as natural waypoints that provide trajectory stabilization at predefined positions, creating reliable reference points in the robot's operational environment. The mathematical foundation established in previous

chapters demonstrates that these stable cycles act as attractors, ensuring convergence of the robot's trajectory toward desired locations with guaranteed stability margins.

The dead reckoning navigation component enables exploration between stable cycles, allowing the robot to traverse unknown or partially mapped environments while maintaining reasonable position estimates. This approach balances the trade-off between exploration capability and navigational accuracy, providing sufficient precision for most inter-cycle transitions while acknowledging the inherent drift limitations of dead reckoning systems.

Recognizing that dead reckoning alone may be insufficient in certain scenarios, the isobath bounce strategy provides a robust fallback mechanism. This method ensures guaranteed convergence to the capture basin of target cycles, even when accumulated navigation errors would otherwise not guarantee a successful transition. The proposed isobath bounce technique exploits environmental features to navigate between stable cycles along isobaths. It shares the same philosophy of cycle navigation by providing a way to navigate without localization along isobaths, which become a curvilinear abscissa along which the robot is moving.

The validation approach employed demonstrates the practical viability of the proposed methods. Simulation results confirm the theoretical predictions regarding stability and convergence properties, while real-world experiments using the BlueBoat provide compelling evidence of the effectiveness of this navigation method. The experimental setup records GNSS signals as ground truth data, allowing for objective evaluation of the navigation performance without relying on GNSS for control.

The experimental results validate both navigation strategies under realistic conditions, demonstrating that the cycle-based approach can maintain adequate positioning accuracy for extended autonomous operations. The successful implementation on a surface vehicle platform provides confidence for future deployment in fully GNSS-denied environments, such as underwater applications where the ultimate utility of this navigation framework will be realized.

The navigation strategies presented in this chapter represent a significant step toward truly autonomous operation in challenging environments. By combining the theoretical rigor of cycle navigation with practical implementation considerations, the proposed framework offers a viable solution for robots operating beyond the reach of conventional positioning systems. The dual strategy approach, incorporating both dead reckoning and isobath bounce methods, provides the redundancy and robustness necessary for reliable autonomous navigation in complex, unknown environments.

Future work will focus on extending these navigation strategies to three-dimensional environments and investigating adaptive methods for cycle selection and transition planning based on real-time environmental conditions and mission objectives.

8 State estimation

8.1 Introduction

In previous chapters, we have explored various aspects of robotic navigation where the cyclic state of the system was neither directly measured nor explicitly estimated. This was not the focus of the previous chapters. This chapter addresses the problem of estimating the cyclic state of a robotic system, specifically in the context of underwater. The goal is now to estimate the cyclic state of the system to answer these three questions:

- Where is the robot at a given time?
- What is the area that the robot can observe at a given time?
- What is the position of a sensed object along the cycle?

This chapter addresses this gap by developing tools for estimating the state of the robot using set methods. We consider a robot operating within a known environment,

where obstacles are present at known positions. For underwater robotic applications, this environment could be a swimming pool or a structured harbor area where the geometry and obstacle locations are well-characterized. The known nature of the environment provides crucial geometric constraints that can be exploited for state estimation purposes.

The foundation of our approach lies in the use of contractors on geometric constraints. These constraints naturally emerge from the robot's interaction with its environment. We introduce for instance the visibility constraint, which is the set of areas visible from an observation point relative to obstacles in the environment. The visibility constraint establishes a geometric relationship between the robot's pose and the observable portions of the environment [30], providing valuable information for state estimation. Another classical geometric constraint is the no cross constraint [24]. If the robot has a sensor which measures a distance to an obstacle, then the robot have a minimal distance to all obstacles of the environment, as the segment between the robot and the obstacle cannot cross the obstacle.

These geometric constraints, while providing valuable information, introduce a significant computational challenge: the formation of fake boundaries at the union of adjacent constraint sets [10]. This phenomenon occurs when multiple geometric constraints are combined, often resulting in artificial discontinuities that do not reflect the true underlying geometry of the problem. These fake boundaries can severely impact the accuracy and reliability of set-based state estimation algorithms. Two methods will be introduced in this chapter to address this issue: the introduction of a boundary preserving form, and a boundary approach to characterize the solution set. The goal of these methods is to preserve the boundaries of the solution set, ensuring that the resulting set accurately reflects the true geometry of the problem. These two methods are based on the topological analysis of set operators, which will be presented in this chapter.

Another particularly important constraint in underwater robotics is the remoteness constraint [46], which becomes fundamental when dealing with sonar-based range measurements. Sonar sensors, commonly employed in underwater environments, typically have a small directivity, meaning they provide range information within an angular cone of measurement. This directivity constraint creates a remoteness relationship between the robot and detected obstacles, where the sensor can measure its distance to the nearest obstacle that lies within the sensor's directivity cone. We implement a separator specifically designed for the remoteness constraint, which efficiently estimate the compatible states for the robot with the measured range and the known map of the environment.

The developed separator is applied to the problem of state estimation for cycle navigation in a pool environment. In this application, the robot follows a repetitive trajectory pattern while gathering only two measurements along the cycle. Then, the robot is able to estimate its state in the pool using the remoteness constraint from its echosounder measurements. Once the state of the cycle is estimated, it is possible to answer the questions mentioned above: the robot's position at a given time can be known by transporting the cyclic state using the flow function of the cycle, set methods have tools to characterize the explored area by the robot, as thicksets [21], and the position of a sensed object along the cycle can be estiamted.

The pool environment serves as an ideal testbed for validating our approach, as it provides well-defined boundaries, known obstacle positions, and controlled conditions that allow for systematic evaluation of the state estimation performance. The cyclic nature of the navigation pattern also enables assessment of the algorithm's ability to maintain consistent state estimates over repeated trajectory segments. This chapter presents the theoretical foundations, algorithmic implementations, and experimental validation of our set-based state estimation approach, demonstrating its effectiveness for autonomous underwater vehicle navigation in structured environments.

8.2 Union of adjacent contractors

First, we introduce the problem of union of adjacent contractors, which is the main issue addressed in this chapter. This problem arises when two sets are adjacent, and their union is computed. The union of these two sets may lead to the appearance of a fake boundary, which is not supposed to belong to the union of these two sets.

For example, the visibility constraint is defined as the set of points that are visible from a given point in the environment [30]. The visibility constraint is defined by the intersection of the half-spaces defined by the obstacles in the environment. For obstacles defined by union of segments, union of adjacent sets may lead to the appearance of a fake boundary. This issue is illustrated in Figure 8.1. In this figure, set of points not visible from the observation point is shown in pink, set of points visible from the observation point is shown in blue, and the boundary is shown by yellow boxes. Fake boundaries are line of yellow boxes appearing inside the set of not visible points, which are clearly not visible from the observation point shown in red. The goal of this section is to address this issue by introducing methods to avoid the appearance of fake boundaries when computing the union of adjacent sets.

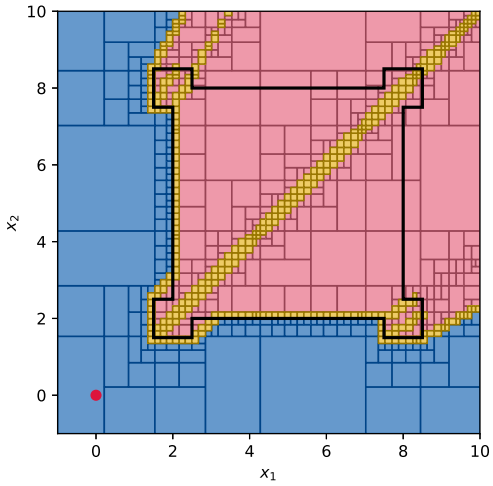


Figure 8.1 Separator on the visibility constraint

8.2.1 Illustrative example

Consider three sets $\mathbb{A}$, $\mathbb{B}$ and $\mathbb{C}$ defined by Equation (8.1).

$$\begin{aligned}
\mathbb{A} &: \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 + 3 \cdot x_2 \in [-\infty, 0]\} \\
\mathbb{B} &: \{(x_1, x_2) \in \mathbb{R}^2 \mid (x_1 + 0.5)^2 + x_2^2 \in [-\infty, 4]\} \\
\mathbb{C} &: \{(x_1, x_2) \in \mathbb{R}^2 \mid (x_1 - 0.5)^2 + x_2 \in [-\infty, 4]\}
\end{aligned} \tag{8.1}$$

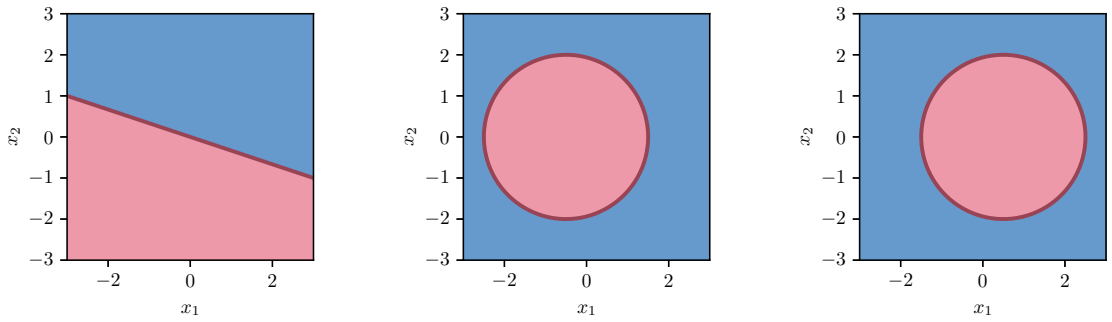
These sets are shown in Figure 8.2. The interior of the set is shown in pink, and the exterior is shown in blue.

Define a set $\mathbb{Z}$, computed using $\mathbb{A}$, $\mathbb{B}$ and $\mathbb{C}$, as shown in Equation (8.2).

$$\mathbb{Z} = (\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap \mathbb{C}) \tag{8.2}$$

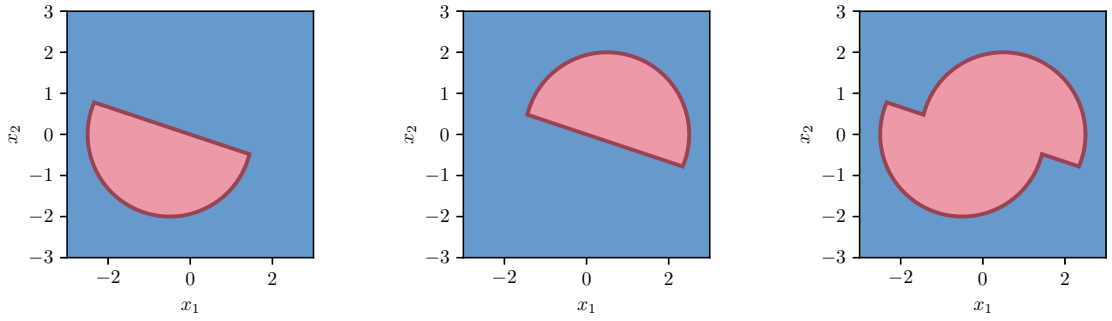
Set $\mathbb{Z}$, shown in Figure 8.3c, is built as the union of sets $\mathbb{A} \cap \mathbb{B}$ and $\overline{\mathbb{A}} \cap \mathbb{C}$ represented in Figure 8.3a, and Figure 8.3b.

Note that the two sets $\mathbb{A} \cap \mathbb{B}$ and $\overline{\mathbb{A}} \cap \mathbb{C}$ share a common and non-overlapping boundary. While paving set $\mathbb{Z}$ using the SIVIA algorithm [40], this common boundary appears as



(a) Set $\mathbb{A}$ **(b)** Set $\mathbb{B}$ **(c)** Set $\mathbb{C}$

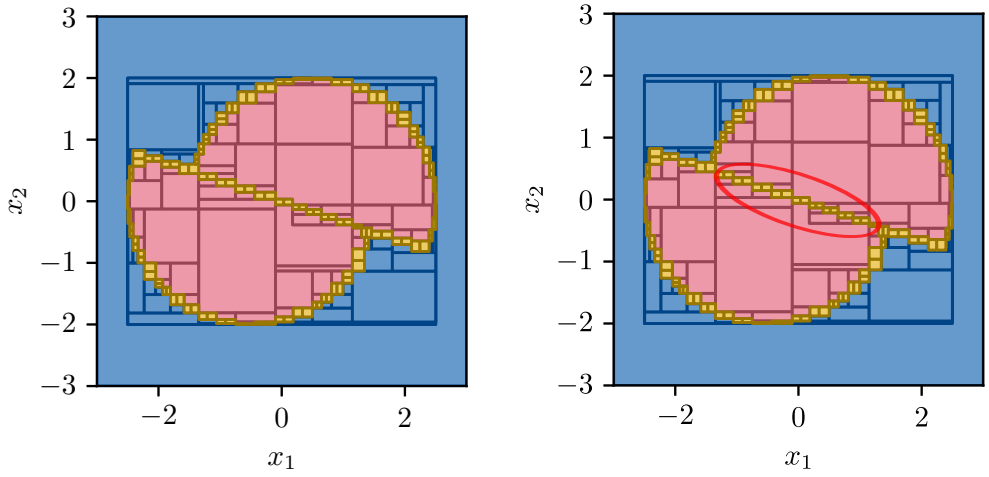
Figure 8.2 Sets $\mathbb{A}$, $\mathbb{B}$ and $\mathbb{C}$



(a) Set $\mathbb{A} \cap \mathbb{B}$ **(b)** Set $\overline{\mathbb{A}} \cap \mathbb{C}$ **(c)** $(\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap \mathbb{C})$

Figure 8.3 Construction of set $\mathbb{Z}$ from $\mathbb{A}$, $\mathbb{B}$ and $\mathbb{C}$

shown in Figure 8.4a. This boundary, which is circled in red in Figure 8.4b is called a fake boundary [100] and these yellow boxes should be represented as pink boxes as they belong to $\mathbb{Z}$. Furthermore, the paving algorithm should be able to classify a box overlapping this fake boundary as fully inside $\mathbb{Z}$ without bisecting it.



(a) Paving of set $\mathbb{Z}$ **(b)** Highlighting the fake boundary

Figure 8.4 Paving of set $\mathbb{Z}$ with the fake boundary

8.2.2 Paving point of view

As shown in Figure 8.5, the paving algorithm is unable to classify an inner box $[\mathbf{b}]$ overlapping the fake boundary as fully inside $\mathbb{Z}$. Using contractors defined for $\mathbb{Z}$, inner parts $[\mathbf{b}] \setminus [\mathbf{b}_1] = [\mathbf{b}] \setminus C_{\mathbb{A}\cap\mathbb{B}}([\mathbf{b}])$, and $[\mathbf{b}] \setminus [\mathbf{b}_2] = [\mathbf{b}] \setminus C_{\overline{\mathbb{A}}\cap\mathbb{C}}([\mathbf{b}])$ are well classified. The remaining part $[\mathbf{b}_3] = [\mathbf{b}] \setminus [\mathbf{b}_1] \setminus [\mathbf{b}_2]$ is classified as unknown and is bisected until the paving algorithm reaches the desired precision.

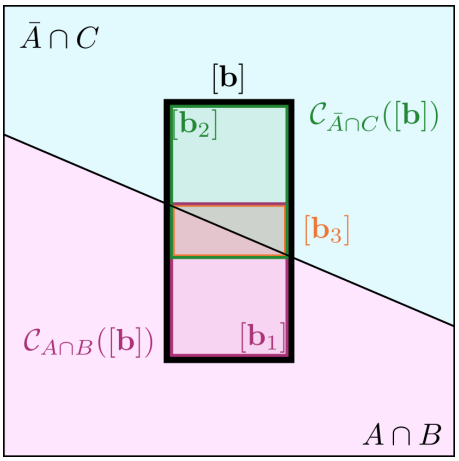
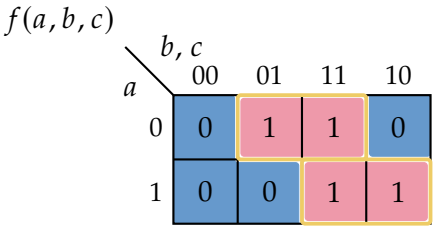


Figure 8.5 Paving of the fake boundary

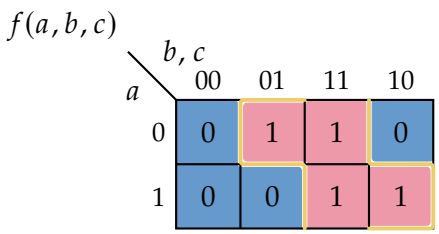
To avoid this issue, the paving algorithm has to take into account the fact that $\mathbb{A} \cup \overline{\mathbb{A}} = \mathbb{R}^n$. With this piece of information, the box $[\mathbf{b}]$ can be classified as fully inside $\mathbb{Z}$ in one step.

8.2.3 Karnaugh map point of view

Karnaugh maps [43] for $(\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap \mathbb{C})$ and $\mathbb{Z}$ shown in Figure 8.6a and Figure 8.6b, respectively. The interior is shown in pink, the exterior is shown in blue, and the boundary is shown in yellow. Although the interior and the exterior of these two sets are equal, the boundaries differ. By denoting by $\delta\mathbb{A}$ the boundary of a set $\mathbb{A}$, the fake boundary appearing on the paving in Figure 8.4a is $\partial\mathbb{A} \cap \mathbb{B} \cap \mathbb{C}$ and is exactly the difference between the boundaries of $(\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap \mathbb{C})$ and $\mathbb{Z}$.



(a) Karnaugh map of $(\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap C)$



(b) Karnaugh map of $\mathbb{Z}$

Figure 8.6 Comparing Karnaugh maps of $(\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap \mathbb{C})$ and $\mathbb{Z}$

8.2.4 Raised issues

Fake boundaries raise two issues. First they add pessimism to the results by classifying boxes around the common boundary as uncertain, whereas these boxes should belong to the union of the two sets. Secondly, fake boundaries slow down the paving algorithm by causing unnecessary box bisections around them.

8.3 Stability of Set Operators

8.3.1 Topological analysis of set operators

To better understand the issue around the union of adjacent sets, we need to define some tools to analyze the origin of these fake boundaries. Actually, although it turns out that fake boundaries may occur regardless of whether a set operator is Hausdorff-stable, solutions to avoid these fake boundaries are not the same in the two cases.

8.3.2 Hausdorff distance

Let $(\mathbb{S}, d)$ be a metric space. Define the ϵ-fattening [65] of a set $\mathbb{X}$ of $\mathbb{S}$ by Equation (8.3).

$$\mathbb{X}_\epsilon = \bigcup_{x \in \mathbb{X}} \{z \in \mathbb{S} \mid d(z, x) \leq \epsilon\} \tag{8.3}$$

The ϵ-fattening of a set $\mathbb{X}$ is the set of all points in $\mathbb{S}$ that are at most ϵ away from a point in $\mathbb{X}$ relative to the distance d of the metric space, as shown in Figure 8.7.

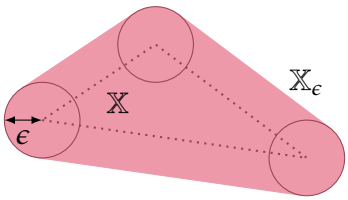


Figure 8.7 ϵ-fattening of a set

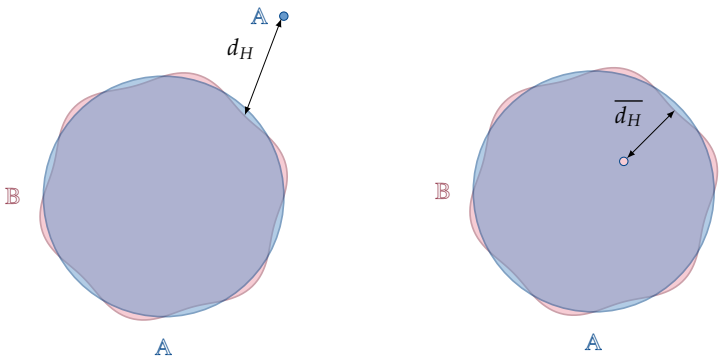
The Hausdorff distance [65] between two subsets $\mathbb{X}$ and $\mathbb{Y}$ of $\mathbb{S}$ is defined by Equation (8.4):

$$d_H(\mathbb{X}, \mathbb{Y}) = inf\{\epsilon \in \mathbb{R}^+ \mid \mathbb{X} \subseteq \mathbb{Y}_\epsilon \quad and \quad \mathbb{Y} \subseteq \mathbb{X}_\epsilon\} \tag{8.4}$$

We also introduce the complementary Hausdorff distance defined in Equation (8.5):

$$\overline{d_H}(\mathbb{X}, \mathbb{Y}) = d_H(\overline{\mathbb{X}}, \overline{\mathbb{Y}}) \tag{8.5}$$

Example 31. *Figure 8.8 illustrates cases where Hausdorff distance and complementary Hausdorff distance are significant. Figure 8.8a shows an example of two sets $\mathbb{A}$ and $\mathbb{B}$ with $d_H(\mathbb{A}, \mathbb{B})$ large because of the small part of $\mathbb{A}$ far from the main part, but $\overline{d_H}(\mathbb{A}, \mathbb{B})$ is tiny, whereas Figure 8.8b shows an example where $d_H(\mathbb{A}, \mathbb{B})$ is tiny and $\overline{d_H}(\mathbb{A}, \mathbb{B})$ is large because of the hole in $\mathbb{A}$.*



(a) Large $d_H(\mathbb{A}, \mathbb{B})$ **(b)** Large $\overline{d_H}(\mathbb{A}, \mathbb{B})$

Figure 8.8 Illustration of large Hausdorff and complementary Hausdorff distances

To take into account the general topology of sets, and to be able to compare it, the generalized Hausdorff distance is introduced and defined in Equation (8.6). It is the maximum between the Hausdorff distance and the complementary Hausdorff distance.

$$H_d(\mathbb{X}, \mathbb{Y}) = max\{d_H(\mathbb{X}, \mathbb{Y}), \overline{d_H}(\mathbb{X}, \mathbb{Y})\} \tag{8.6}$$

8.3.3 Hausdorff stability

Consider two subsets $\mathbb{X}$ and $\mathbb{Y}$ of $\mathbb{S}$. Then a binary operator $\diamond$ acting on set $\mathbb{X}$ and $\mathbb{Y}$ is stable if it meets condition of Equation (8.7).

$$\forall \eta \in \mathbb{R}_+, \quad \exists \epsilon \in \mathbb{R}_+^*, \quad \begin{cases} H_d(\mathbb{X}, \widetilde{\mathbb{X}}) & \leq \epsilon \\ H_d(\mathbb{Y}, \widetilde{\mathbb{Y}}) & \leq \epsilon \end{cases} \implies H_d(\mathbb{X} \diamond \mathbb{Y}, \widetilde{\mathbb{X}} \diamond \widetilde{\mathbb{Y}}) \leq \eta \tag{8.7}$$

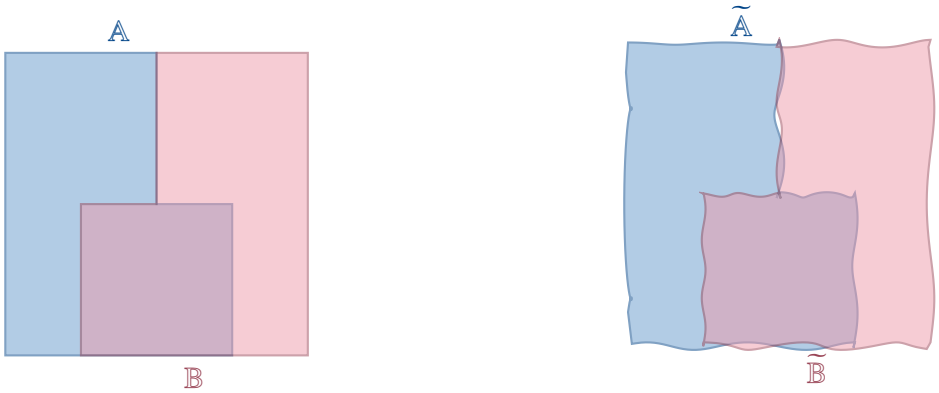
Example 32. *Consider two subsets $\mathbb{A}$ and $\mathbb{B}$ shown in Figure 8.9a and two other sets $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$ shown in Figure 8.9b.*

For the union operator, $d_H(\mathbb{A} \cup \mathbb{B}, \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}})$ is small, but $\overline{d_H}(\mathbb{A} \cup \mathbb{B}, \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}})$ is large as the union of $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$ generates holes at the common boundary of $\mathbb{A}$ and $\mathbb{B}$. Then $H_d(\mathbb{A} \cup \mathbb{B}, \widetilde{\mathbb{A}} \cup \widetilde{\mathbb{B}})$ is large, and the union operator is not Hausdorff-stable for these sets, as it does not meet the condition of Equation (8.7).

Example 33. *Consider two sets $\mathbb{A}$ and $\mathbb{B}$ shown in Figure 8.9a and two other sets $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$ shown in Figure 8.9b.*

For the intersection operator, $\overline{d_H}(\mathbb{A} \cap \mathbb{B}, \widetilde{\mathbb{A}} \cap \widetilde{\mathbb{B}})$ is small, but $d_H(\mathbb{A} \cap \mathbb{B}, \widetilde{\mathbb{A}} \cap \widetilde{\mathbb{B}})$ is large as the intersection of $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$ generates residual sets at the common boundary of $\mathbb{A}$ and $\mathbb{B}$. Then $H_d(\mathbb{A} \cap \mathbb{B}, \widetilde{\mathbb{A}} \cap \widetilde{\mathbb{B}})$ is large, and the intersection operator is not Hausdorff-stable for these sets, as it does not meet the condition of Equation (8.7).

Example 34. *Consider the illustrative example presented in Section 8.2. The union operator between $\mathbb{A} \cap \mathbb{B}$ and $\overline{\mathbb{A}} \cap \mathbb{C}$ is Hausdorff stable as the generalized Hausdorff distance is small. This come from the fact that the same set $\mathbb{A}$ is involved in the computation of $H_d(\mathbb{A} \cap \mathbb{B}, \widetilde{\mathbb{A}} \cap \widetilde{\mathbb{B}})$ and $H_d(\overline{\mathbb{A}} \cap \mathbb{C}, \widetilde{\overline{\mathbb{A}}} \cap \widetilde{\mathbb{C}})$.*



(a) Sets $\mathbb{A}$ and $\mathbb{B}$ **(b)** Sets $\widetilde{\mathbb{A}}$ and $\widetilde{\mathbb{B}}$

Figure 8.9 $\mathbb{A}$ and $\mathbb{B}$ are not Hausdorff-stable for union and intersection operators

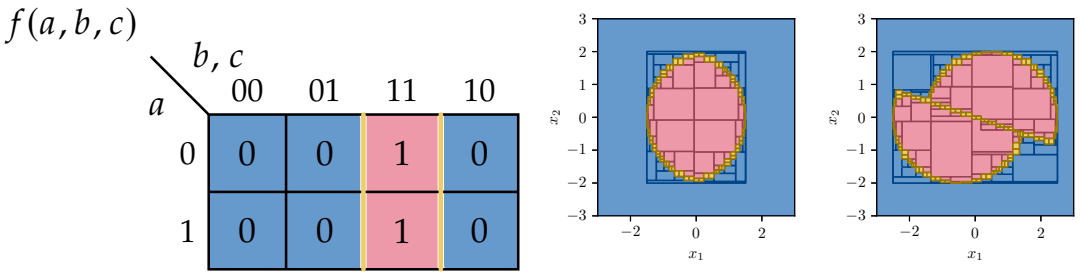
This Hausdorff stability condition characterizes the fact that a small perturbation on sets will change the topology of the result by opening boundaries or creating additional ones. It allows identifying topologically different problems. Adapted solutions for Hausdorff-stable and non Hausdorff-stable problems will be proposed in the following sections.

8.4 Stable Case Solution: Boundary-Preserving Form

In the Hausdorff-stable case, it is possible to change the expression of the computed set $\mathbb{Z}$ by adding a set overlapping the fake boundary. This set helps classify boxes around the fake boundary in the paving algorithm. It must be chosen such that the interior and the exterior of $\mathbb{Z}$ are preserved, but also its boundary. In this example, the set $\mathbb{D} = \mathbb{B} \cap \mathbb{C}$ is added to the expression of $\mathbb{Z}$ which becomes $\mathbb{Z}'$ Equation (8.8).

$$\mathbb{Z}' = (\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap \mathbb{C}) \cup (\mathbb{B} \cap \mathbb{C}) \tag{8.8}$$

The Karnaugh map of the set $\mathbb{D}$ is shown in Figure 8.10a, and the paving of $\mathbb{D}$ is shown in Figure 8.10b. This set ensures that the Karnaugh map of $\mathbb{Z}'$ is the same as the Karnaugh map of $\mathbb{Z}$ shown in Figure 8.6b. The resulting paving of $\mathbb{Z}'$ is shown in Figure 8.10c. There is no more fake boundaries.



(a) Karnaugh map of $\mathbb{D}$ **(b)** Paving of Set $\mathbb{D}$ **(c)** Paving of Set $\mathbb{Z}'$

Figure 8.10 Boundary preserving form

Using the boundary preserving form leads to a correct paving without any fake boundaries. Therefore, to use this solution, the set boundaries have to be analyzed to find the fake boundaries and add a set overlapping these fake boundaries in the expression of the set to pave. This approach is working but is problem-specific and needs to be adapted on a case-by-case basis. This method works well in the Hausdorff-stable case, as there is the possibility to add a set that overlap the fake boundary. For non Hausdorff-stable operators, the boundary preserving form is not possible as the Karnaugh map is not highlighting any sets that can be added to the expression of the paved set to avoid fake boundaries, and another approach is needed.

8.5 Non-Stable Case: Boundary Approach

8.5.1 Topology of the boundary

Let $T = (\mathbb{S}, \tau)$ be a topological space. $\forall \mathbb{X} \in \mathbb{S}$, denote by $\overline{\mathbb{X}}$ the complementary of $\mathbb{X}$ in $\mathbb{S}$, by $cl_{\mathbb{S}}(\mathbb{X})$ the closure of $\mathbb{X}$ in $\mathbb{S}$, by $int_{\mathbb{S}}(\mathbb{X})$ the interior of $\mathbb{X}$ in $\mathbb{S}$, and by $\partial\mathbb{X}$ the boundary of $\mathbb{X}$ in $\mathbb{S}$.

Theorem 11. *Let $T = (\mathbb{S}, \tau)$ be a topological space. Then*

$$\forall (\mathbb{A}, \mathbb{B}) \in \mathbb{S}^2 \quad \partial(\mathbb{A} \cup \mathbb{B}) \subseteq \partial\mathbb{A} \cup \partial\mathbb{B}$$

Proof. By definition of the boundary

$$\forall \mathbb{A} \in \mathbb{S}, \quad \partial\mathbb{A} = cl_{\mathbb{S}}(\overline{\mathbb{A}}) \cap cl_{\mathbb{S}}(\mathbb{A})$$

By property, intersection is a subset of each set

$$\forall (\mathbb{A}, \mathbb{B}) \in \mathbb{S}^2, \quad \begin{cases} \mathbb{A} \cap \mathbb{B} \subseteq \mathbb{A} \\ \mathbb{A} \cap \mathbb{B} \subseteq \mathbb{B} \end{cases}$$

Then

$$\begin{aligned} \partial(\mathbb{A} \cup \mathbb{B}) &= cl_{\mathbb{S}}(\overline{\mathbb{A} \cup \mathbb{B}}) \cap cl_{\mathbb{S}}(\mathbb{A} \cup \mathbb{B}) \\ &= cl_{\mathbb{S}}(\overline{\mathbb{A}} \cap \overline{\mathbb{B}}) \cap cl_{\mathbb{S}}(\mathbb{A} \cup \mathbb{B}) \\ &= cl_{\mathbb{S}}(\overline{\mathbb{A}} \cap \overline{\mathbb{B}}) \cap (cl_{\mathbb{S}}(\mathbb{A}) \cup cl_{\mathbb{S}}(\mathbb{B})) \\ &= (cl_{\mathbb{S}}(\overline{\mathbb{A}} \cap \overline{\mathbb{B}}) \cap cl_{\mathbb{S}}(\mathbb{A})) \cup cl_{\mathbb{S}}(\overline{\mathbb{A}} \cap \overline{\mathbb{B}}) \cap cl_{S}(\mathbb{B}) \\ &\subseteq (cl_{\mathbb{S}}(\overline{\mathbb{A}}) \cap cl_{\mathbb{S}}(\mathbb{A})) \cup (cl_{\mathbb{S}}(\overline{\mathbb{B}}) \cap cl_{\mathbb{S}}(\mathbb{B})) \\ &= \partial\mathbb{A} \cup \partial\mathbb{B} \end{aligned}$$

□

Theorem 11 demonstrates that the boundary is not preserved over union of sets as $\partial(\mathbb{A} \cup \mathbb{B}) \subseteq \partial\mathbb{A} \cup \partial\mathbb{B}$. This is why the paving of the union of contractors leads to fake boundaries.

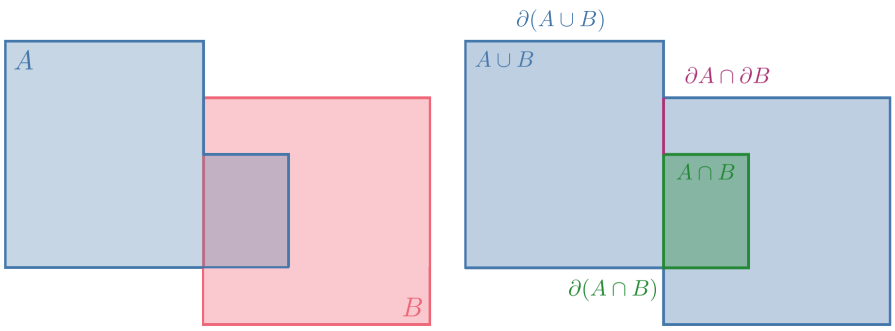
Theorem 12 present the general formula for the union of the boundary of two sets.

Theorem 12. *Let (S, τ) be a topological space. Then*

$$\forall (\mathbb{A}, \mathbb{B}) \in \mathbb{S}^2 \qquad \partial\mathbb{A} \cup \partial\mathbb{B} = \partial(\mathbb{A} \cup \mathbb{B}) \cup \partial(\mathbb{A} \cap \mathbb{B}) \cup (\partial\mathbb{A} \cap \partial\mathbb{B})$$

Proof. Theorem 12 is proven in [51]. □

From Theorem 12, it is noticeable that the union of boundaries is not the boundary of union. This is the reason why $\mathbb{Z}$ and $(\mathbb{A} \cap \mathbb{B}) \cup (\overline{\mathbb{A}} \cap \mathbb{C})$ do not have the same boundaries when paving these sets. An illustration of Theorem 12 is shown in Figure 8.11.



(a) Set $\mathbb{A}$ and $\mathbb{B}$ **(b)** Decomposition of $\partial\mathbb{A} \cup \partial\mathbb{B}$

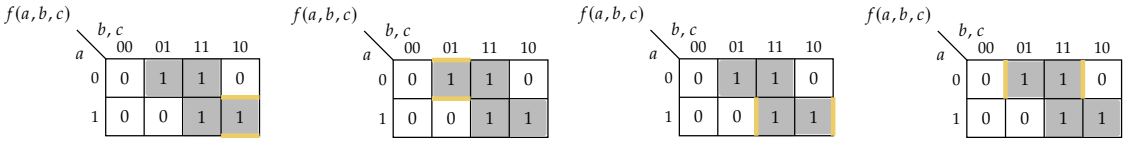
Figure 8.11 Illustration of Theorem 12

Remark. *When $\mathbb{A} \cap \mathbb{B} = \emptyset$ and $\partial\mathbb{A} \cap \partial\mathbb{B} = \emptyset$ in Theorem 12, the union of boundaries is the boundary of union. This is the case where the sets are non-overlapping with no common boundary.*

8.5.2 Boundary approach

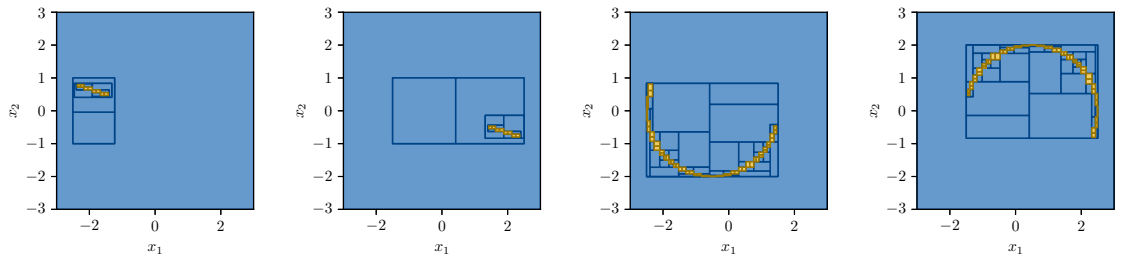
A boundary approach can be used to get rid of this fake boundary. This will help to solve this purely computational problem, as the mathematical expression of the set $\mathbb{Z}$ do not have any fake boundaries. This approach consists in computing the boundary of the set $\mathbb{Z}$. This boundary will separate an inner and an outer subpaving. The classification of the resulting subpavings as inside or outside is done using a predicate. The boundary approach method was first introduced in [37] to speed up the solving of set inversion problems.

First, $\partial\mathbb{Z}$ has to be expressed from set $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ without the fake boundary. Figure 8.12 and Figure 8.13 respectively show Karnaugh maps and paving of intermediate sets involved in the building of $\partial\mathbb{Z}$. Then, $\partial\mathbb{Z}$ is computed as the union of these boundaries, and it matches the Karnaugh map of $\mathbb{Z}$ shown in Figure 8.6b.



(a) $\partial\mathbb{A} \cap \mathbb{B} \cap \overline{\mathbb{C}}$ **(b)** $\partial\mathbb{A} \cap \overline{\mathbb{B}} \cap \mathbb{C}$ **(c)** $\mathbb{A} \cap \partial\mathbb{B}$ **(d)** $\overline{\mathbb{A}} \cap \partial\mathbb{C}$

Figure 8.12 Karnaugh map of the boundaries



(a) $\partial\mathbb{A} \cap \mathbb{B} \cap \overline{\mathbb{C}}$ **(b)** $\partial\mathbb{A} \cap \overline{\mathbb{B}} \cap \mathbb{C}$ **(c)** $\mathbb{A} \cap \partial\mathbb{B}$ **(d)** $\overline{\mathbb{A}} \cap \partial\mathbb{C}$

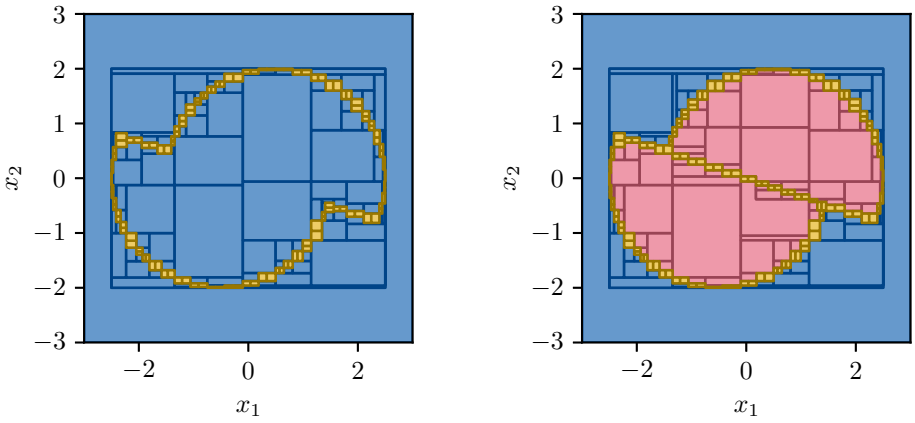
Figure 8.13 Building the boundary of $\mathbb{Z}$

Then using a predicate, the connected subsets separated by $\partial\mathbb{Z}$ are classified as inside or outside. This predicate is based on the expression of $\mathbb{Z}$ of Equation (8.2), and is tested on box corners until an in and an out points are found. Then, boxes containing each point are classified as in and out boxes, and this information is propagated to boxes belonging to the same connected subsets of the paving. Finally, each box is classified as in, out, or uncertain.

Figure 8.14a shows $\partial\mathbb{Z}$ built from boundaries shown in Figure 8.13 using the proposed method. The resulting paving of Z is shown in Figure 8.14b, which is classified using the subpaving coloration method.

This boundary approach is efficient to get rid of fake boundaries. Set $\mathbb{Z}$ is computed from the union of two separators, $\mathcal{S}_{\mathbb{A}\cap\mathbb{B}}$, and $\mathcal{S}_{\overline{\mathbb{A}}\cap\mathbb{C}}$, and this union is reinforced by a contractor on the boundary $\mathcal{C}_{\partial\mathbb{Z}}$.

Remark. *This method is also working for the Hausdorff-stable case, but it is more efficient to use the boundary preserving form presented in Section 8.4, as the contractor on the boundary is not easy to define, and the subpaving coloration method is not needed.*



(a) Paving of ∂Z **(b)** Paving of Z

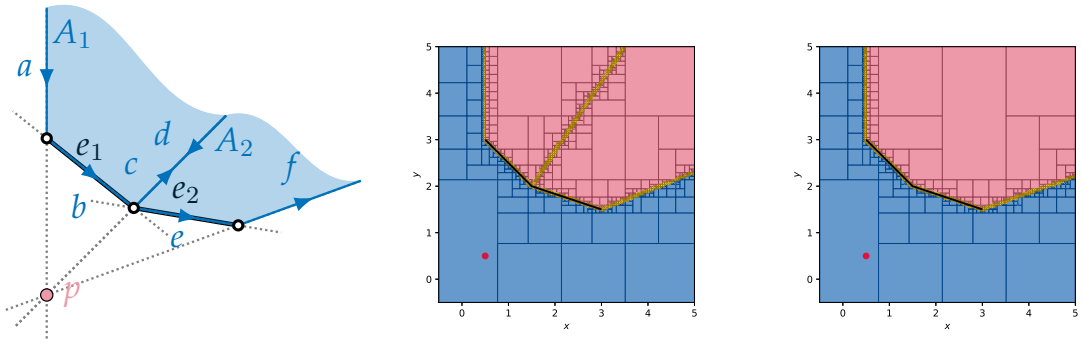
Figure 8.14 Boundary approach

8.6 Applications

8.6.1 Boundary approach application to the separator on the visibility constraint

Separator over the visibility constraint, as implemented in [30], suffers from this fake boundaries when it deals with polygon obstacles. In fact, the contractor on the visibility constraint is defined for an obstacle segment. The extension to polygons involves the union of non-visible areas relative to each segment, and this union leads to fake boundaries.

Figure 8.15a shows an illustration of the separator on the visibility constraint as implemented in [30]. For each obstacle segment, three segments are defined that specify visible and non-visible parts of the space. For segment e_1 is defined relative to the observation point p, the oriented half space on the left of segment a, the one on the left of segment b, and the same for segment c. It is the same for the set of visible points for segment e_2 defined by half planes on the left of segments d, e, and f. The set of masked points from p by e_1 and e_2 i then the union of these two sets A_1 and A_2. Paving this separator shows a fake boundary as shown in Figure 8.15b.



(a) Separator construction **(b)** Classical approach **(c)** Boundary approach

Figure 8.15 Separator on the visibility constraint using the boundary approach

To avoid this problem, the boundary approach can be applied. The set of masked points from observation point p relative to segments e_1 and e_2 should be defined by half planes on the left of segments a, b, e, and f. The simplification of $c = -d$ has to be taken into account while contracting to avoid this fake boundary. This simplification is based on algebraic topology [92] in which boundary simplifications are defined and used.

Figure 8.15c shows the paving of this separator using the boundary approach. There is no longer fake boundaries appearing.

Remark. *For now, fake boundaries have to be identified and removed by hand, as it not the main topic of this paper. Neither [30] nor this work propose an automatic boundary simplification to avoid fake boundaries in union of adjacent sets. Therefore, it is necessary to find solutions that are problem-specific in order to avoid fake boundaries, as developed in the next subsection.*

8.6.2 Toward a generic implementation of the separator on the visibility constraint

In the case of the visibility constraint another approach to solve this problem can be proposed. The set of visible points from an observation point placed at $(0, 0)$ relative to a shape $\mathbb{Y}$ can be defined by :

$$\mathbb{S} = \{x \in \mathbb{R}^2, \exists \alpha \in \mathbb{R} \mid \alpha \cdot \mathbf{x} \in \mathbb{Y}\} \tag{8.9}$$

Denoting by f the homothety defined in Equation (8.10).

$$\begin{aligned} f : \mathbb{R}^3 &\mapsto \mathbb{R}^2 \\ (\mathbf{x}, \alpha) &\rightarrow \alpha \cdot \mathbf{x} \end{aligned} \tag{8.10}$$

Remark. *If the observation point is not placed at (0, 0), a simple translation of the problem leads to the presented solution.*

The set $\mathbb{S}$ can then be defined as the projection of $f(\mathbb{Y})$ for $\alpha \in [0, 1]$. Listing 1 show the implementation of this separator using the Codac Library [78]. Figure 8.16b shows the paving of this implementation of the visibility constraint. The comparison with Figure 8.16a, where the classical implementation of this constraint from [30] on the same obstacle polygon is shown, validates that the problem of fake boundaries is avoided with this method. Figure 8.16b requires 391 bisections whereas Figure 8.16a requires 321 bisections. The complexity of these two approaches is quite similar, although all the tests carried out lead to a slightly higher number of bisections for the proposed method, with the benefit of a set without fake boundaries. This is mainly due to the projection algorithm which induces bisections in the dimension of the homothety factor α.

Listing 1 Separator on the visibility constraint using Codac Library

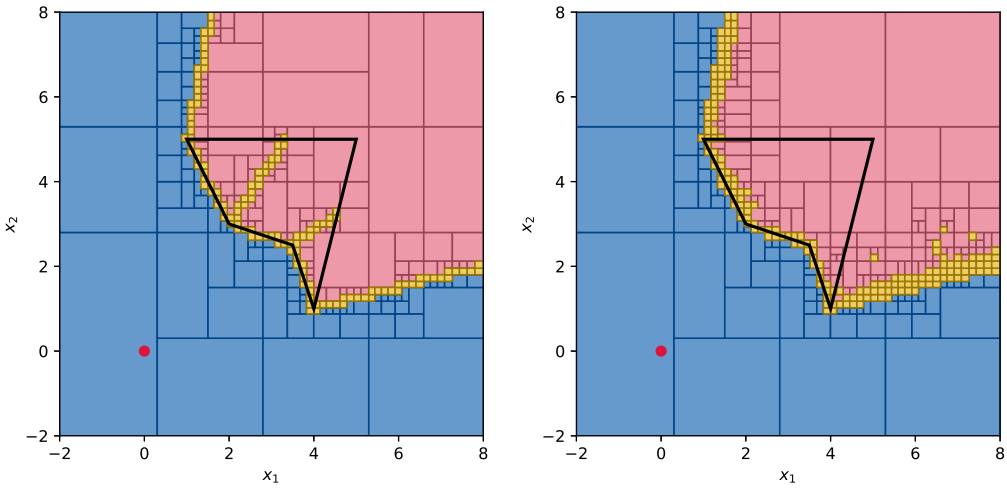
```
import codac as cd

# Set Y definition
polygon = [[2, 3], [3.5, 2.5], [4, -1], [5, 5], [1, 5], [2, 3]]
Sy = cd.SepPolygon(polygon)

# Set Z definition
f = cd.Function("x", "y", "a", "(a*x,a*y)")
Sz = cd.SepInverse(Sy, f)

# Projection of for a in [0, 1]
epsilon = 0.1
Sx = cd.SepProj(Sz, cd.Interval(0, 1), epsilon)
```

Figure 8.17 shows the paving of the visibility separator on the same obstacle presented in Figure 8.1, but the proposed method avoid fake boundaries.



(a) Classical implementation **(b)** Proposed implementation

Figure 8.16 Generic SepVisible implementation

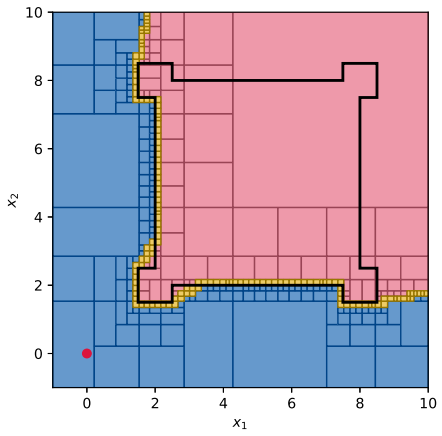


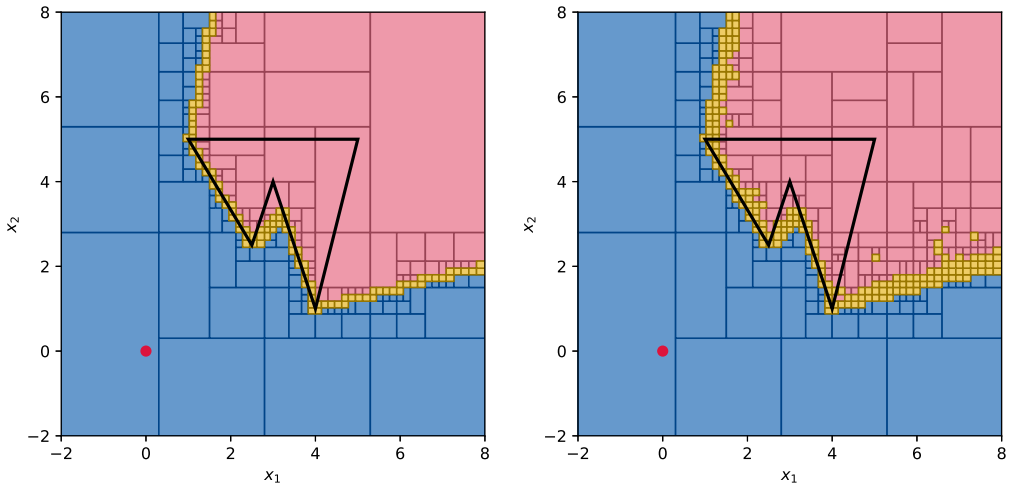
Figure 8.17 Separator on the visibility constraint on a room

Remark. *The separator representing the obstacle could be any separator. However, the separator must be in a closed form with an interior. This method is not applicable to segments or open polygons for instance. But the advantage of this approach is that it can be applied to an ellipsis obstacle.*

Remark. *There are polygons for which the separator on the visibility constraint does not generate fake boundaries. In these cases the classical implementation proposed in [30] is more efficient than the proposed approach, as the algorithm used to project a separator is based on contractors over quantifiers which requires bisections [13, 66]. Figure 8.18 shows the comparison between the classical and the projection approaches on the paving of a visibility separator without fake boundaries. The proposed implementation shown in Figure 8.18b requires* 419 *bisections whereas the classical implementation shown in Figure 8.18a requires only 278 bisections.*

8.7 Separator on the Remoteness constraint

The remoteness concept establishes a fundamental geometric constraint that relates the spatial configuration of a range measurement sensors to the measured distance of obstacles within its sensing cone. This constraint provides a mathematical framework for incorporating sensor measurements into state estimation algorithms for robotic systems operating in structured environments. This constraint is particularly relevant for robots equipped



(a) Classical implementation **(b)** Proposed implementation

Figure 8.18 Generic implementation of the separator on the visibility constraint

with acoustic range sensors, as ultrasonic sensors used with Uncrewed Aerial Vehicles, or Uncrewed Ground Vehicles, and as sonars used with Uncrewed Underwater Vehicles. The remoteness constraint is defined in the context of a two-dimensional environment, and was introduced in [46]. In this paper, the remoteness constraint was introduced with the expression of its inclusion function, and was used in an inclusion test. The goal of this section is to present the remoteness constraint, and to provide a separator on this constraint that can be used in a paving algorithm to solve state estimation problems.

Let $\mathbf{m} \in \mathbb{R}^2$ denote the position of the acoustic range sensor in the global coordinate frame. The measurement cone of the sensor is characterized by two unit vectors $\mathbf{u}_1$ and $\mathbf{u}_2 \in S^1$, where S^1 represents the unit circle. These vectors define the angular boundaries of the field of view of the sensor, establishing the geometric limits within which distance measurements are valid.

Consider an obstacle represented as a line segment $[\mathbf{a}, \mathbf{b}]$ in the environment, where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$ are the endpoints of the segment. The acoustic sensor measures the distance $d \in \mathbb{R}_+$ to the closest point on this obstacle segment that lies within the measurement cone $C(\mathbf{m}, \mathbf{u}_1, \mathbf{u}_2)$ defined by $\mathbf{m}$, $\mathbf{u}_1$, and $\mathbf{u}_2$.

Figure 8.19 shows an illustration of the remoteness constraint. The sensor is located at point $\mathbf{m}$, and the measurement cone is defined by two unit vectors $\mathbf{u_1}$ and $\mathbf{u_2}$. The obstacle segment is defined by its endpoints $\mathbf{a}$ and $\mathbf{b}$. The measured distance d is the distance to the closest point on the obstacle segment within the measurement cone. In this example, the distance measured by the sensor is the is $d = ||\mathbf{mb}||$.

By denoting by $\mathbf{h}$ the foot of the perpendicular from $\mathbf{m}$ to the line segment $[\mathbf{a}, \mathbf{b}]$, by $\mathbf{h}_1$ the intersection of the line defined by $\mathbf{u}_1$ and the line segment $[\mathbf{a}, \mathbf{b}]$, and by $\mathbf{h}_2$ the intersection of the line defined by $\mathbf{u}_2$ and the line segment $[\mathbf{a}, \mathbf{b}]$, the measured distance $d \in \{||\mathbf{ma}||, ||\mathbf{mb}||, ||\mathbf{mh}||, ||\mathbf{mh}_1||, ||\mathbf{mh}_2||, +\infty\}$, depending on the position of $\mathbf{m}$ relative to the segment $[\mathbf{a}, \mathbf{b}]$ and the measurement cone $C(\mathbf{m}, \mathbf{u}_1, \mathbf{u}_2)$.

These distances are computed using the following expressions :

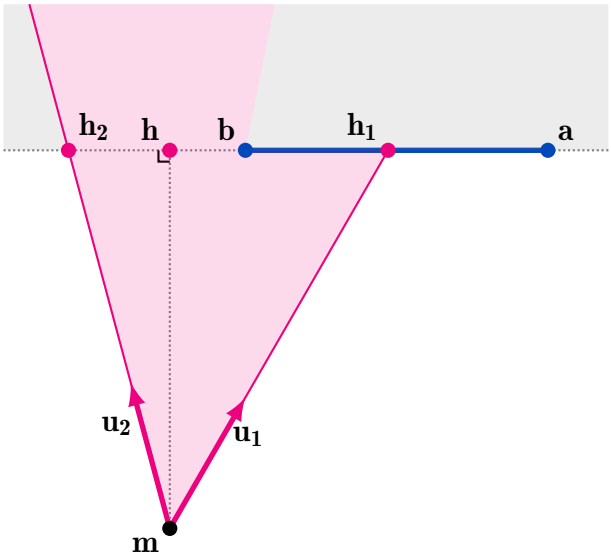


Figure 8.19 Remoteness of a segment [**a**, **b**] relative to a point **m** and two unit vectors $\mathbf{u_1}$ and $\mathbf{u_2}$

$$||\mathbf{ma}|| = \sqrt{(m_x - a_x)^2 + (m_y - a_y)^2} \tag{8.11}$$

$$||\mathbf{mb}|| = \sqrt{(m_x - b_x)^2 + (m_y - b_y)^2} \tag{8.12}$$

$$||\mathbf{mh}|| = \frac{det(\mathbf{ab}, \mathbf{am})}{||\mathbf{ab}||} \tag{8.13}$$

$$||\mathbf{mh_1}|| = \frac{det(\mathbf{ab}, \mathbf{am})}{det(\mathbf{u}_1, \mathbf{ab})} \tag{8.14}$$

$$||\mathbf{mh_2}|| = \frac{det(\mathbf{ab}, \mathbf{am})}{det(\mathbf{u}_2, \mathbf{ab})} \tag{8.15}$$

Separators respectively consistent with these distance constraints are defined and denoted by $\mathcal{S}_{d,a}$, $\mathcal{S}_{d,b}$, $\mathcal{S}_{d,h}$, $\mathcal{S}_{d,h_1}$, $\mathcal{S}_{d,h_2}$.

Figure 8.20 shows the case when the condition c_a is true, i.e. $\langle \mathbf{u}_1, \mathbf{ab} \rangle \geq 0 \wedge \langle \mathbf{u}_2, \mathbf{ab} \rangle \geq 0$. In this case, there are three areas:

(i) If $det(\mathbf{u}_1, \mathbf{bm}) \leq 0 \wedge \det(\mathbf{ab}, \mathbf{am}) \geq 0 \wedge det(\mathbf{u}_1, \mathbf{am}) \geq 0$, then $\mathbf{m} \in \mathbb{S}_{h_1}$, and the distance is $||\mathbf{mh_1}||$,
(ii) If $\det(\mathbf{u}_1, \mathbf{am}) \leq 0 \wedge det(\mathbf{u}_2, \mathbf{am}) \geq 0$, then $\mathbf{m} \in \mathbb{S}_a$, and the distance is $||\mathbf{ma}||$,
(iii) Else the distance is $+\infty$.

Figure 8.21 shows the case when the condition c_h is true, i.e. $\langle \mathbf{u}_1, \mathbf{ab} \rangle < 0 \wedge \langle \mathbf{u}_2, \mathbf{ab} \rangle \geq 0$. In this case, there are four areas:

(i) If $\det(\mathbf{u}_1, \mathbf{bm}) \leq 0 \wedge \langle \mathbf{ba}, \mathbf{bm} \rangle \leq 0$, then $\mathbf{m} \in \mathbb{S}_b$, and the distance is $||\mathbf{mb}||$,
(ii) If $\langle \mathbf{ba}, \mathbf{bm} \rangle \geq 0 \wedge \det(\mathbf{ab}, \mathbf{am}) \geq 0 \wedge \langle \mathbf{ab}, \mathbf{am} \rangle \geq 0$, then $\mathbf{m} \in \mathbb{S}_h$, and the distance is $||\mathbf{mh}||$,
(iii) If $\langle \mathbf{ab}, \mathbf{am} \rangle \leq 0 \wedge det(\mathbf{u}_2, \mathbf{am}) \geq 0$, then $\mathbf{m} \in \mathbb{S}_a$, and the distance is $||\mathbf{ma}||$,
(iv) Else the distance is $+\infty$.

Figure 8.22 shows the case when the condition c_b is true, i.e. $\langle \mathbf{u}_1, \mathbf{ab} \rangle < 0 \wedge \langle \mathbf{u}_2, \mathbf{ab} \rangle < 0$. In this case, there are three areas:

(i) If $\det(\mathbf{u}_1, \mathbf{bm}) \leq 0 \wedge det(\mathbf{u}_2, \mathbf{bm}) \geq 0$, then $\mathbf{m} \in \mathbb{S}_b$, and the distance is $||\mathbf{mb}||$,
(ii) If $det(\mathbf{u}_2, \mathbf{bm}) \leq 0 \wedge \det(\mathbf{ab}, \mathbf{am}) \geq 0 \wedge det(\mathbf{u}_2, \mathbf{am}) \geq 0$, then $\mathbf{m} \in \mathbb{S}_{h_2}$, and the distance is $||\mathbf{mh_2}||$,

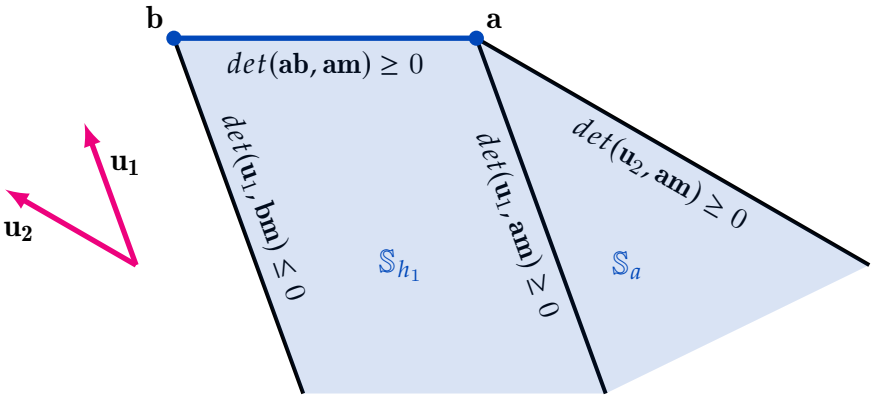


Figure 8.20 Remoteness when c_a is true

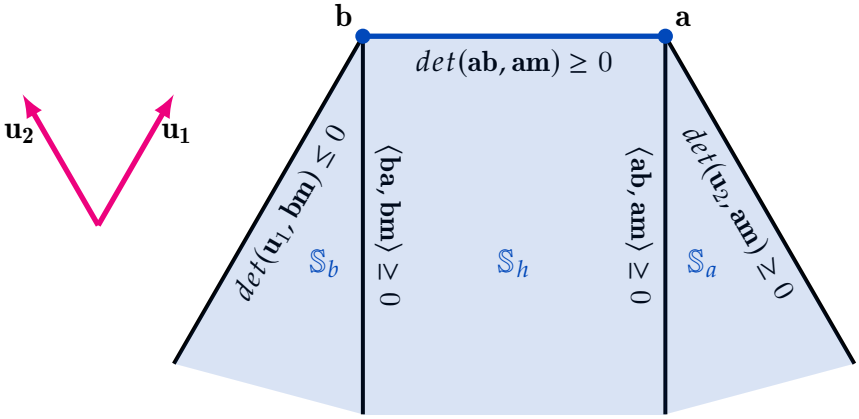


Figure 8.21 Remoteness when c_h is true

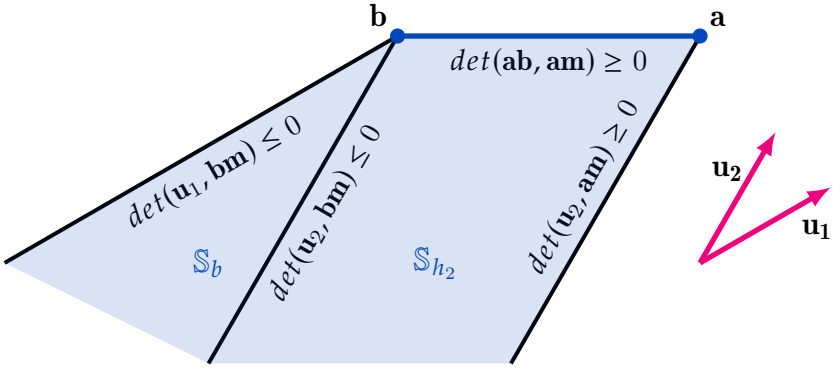


Figure 8.22 Remoteness when c_b is true

(iii) Else the distance is $+\infty$.

Whatever the condition, the remoteness distance is $+\infty$ when $\mathbf{m}$ satisfies the condition $det(\mathbf{u_1}, \mathbf{bm}) > 0 \wedge det(\mathbf{ab}, \mathbf{am}) < 0 \wedge det(\mathbf{u2}, \mathbf{am}) < 0$. In this case $\mathbf{m} \in \mathbb{S}_{out}$.

Separators consistent with sets $\mathbb{S}_a$, $\mathbb{S}_b$, $\mathbb{S}_h$, $\mathbb{S}_{h_1}$, $\mathbb{S}_{h_2}$, and $\mathbb{S}_{out}$ are defined and are respectively denoted by $\mathcal{S}_a$, $\mathcal{S}_b$, $\mathcal{S}_h$, $\mathcal{S}_{h_1}$, $\mathcal{S}_{h_2}$, and $\mathcal{S}_{out}$.

The definition of the separator on the remoteness constraint is then expressed using the separator on the χ-constraint [44], separators on the distances, and separators on the areas depending on the different cases expressed above. The χ-constraint express an if-then-else condition, which is used to select the appropriate separator based on the conditions c_a, c_h, and c_b.

If $c_a \wedge +\infty \in [d]$, then the separator on the remoteness is:

$$S_{remoteness} = \mathcal{S}_{out} \cup (\mathcal{S}_a \cap \mathcal{S}_{d,a}) \cup (\mathcal{S}_{h_1} \cap \mathcal{S}_{d,h_h}) \tag{8.16}$$

If $c_h \wedge +\infty \notin [d]$, then the separator on the remoteness is:

$$S_{remoteness} = (\mathcal{S}_a \cap \mathcal{S}_{d,a}) \cup (\mathcal{S}_{h_1} \cap \mathcal{S}_{d,h_h}) \tag{8.17}$$

If $c_h \wedge +\infty \in [d]$ is true, then the separator on the remoteness is:

$$S_{remoteness} = \mathcal{S}_{out} \cup (\mathcal{S}_a \cap \mathcal{S}_{d,a}) \cup (\mathcal{S}_h \cap \mathcal{S}_{d,h}) \cup (\mathcal{S}_b \cap \mathcal{S}_{d,b}) \tag{8.18}$$

If $c_h \wedge +\infty \notin [d]$ is true, then the separator on the remoteness is:

$$S_{remoteness} = (\mathcal{S}_a \cap \mathcal{S}_{d,a}) \cup (\mathcal{S}_h \cap \mathcal{S}_{d,h}) \cup (\mathcal{S}_b \cap \mathcal{S}_{d,b}) \tag{8.19}$$

If $c_b \wedge +\infty \in [d]$, then the separator on the remoteness is:

$$S_{remoteness} = \mathcal{S}_{out} \cup (\mathcal{S}_b \cap \mathcal{S}_{d,b}) \cup (\mathcal{S}_{h_2} \cap \mathcal{S}_{d,h_2}) \tag{8.20}$$

Finally, if $c_b \wedge +\infty \notin [d]$, then the separator on the remoteness is:

$$S_{remoteness} = (\mathcal{S}_b \cap \mathcal{S}_{d,b}) \cup (\mathcal{S}_{h_2} \cap \mathcal{S}_{d,h_2}) \tag{8.21}$$

Figure 8.23 shows that computing the remoteness in this way can lead to the creation of fake boundaries, which are visible as a line of uncertain yellow boxes in the inner approximation of the set.

To avoid this issue, a boundary preserving form can be used, as the fake boundaries are appearing at the interface between $\mathcal{S}_a$ and $\mathcal{S}_h$, at the interface between $\mathcal{S}_a$ and $\mathcal{S}_{h_1}$, at the interface between $\mathcal{S}_h$ and $\mathcal{S}_b$, and at the interface between $\mathcal{S}_{h_2}$ and $\mathcal{S}_b$. Four new sets are then defined:

$$\mathbb{S}_{ah} = \mathbb{S}_a \cup \mathbb{S}_h \tag{8.22}$$
$$\mathbb{S}_{ah_1} = \mathbb{S}_a \cup \mathbb{S}_{h_1} \tag{8.23}$$
$$\mathbb{S}_{hb} = \mathbb{S}_h \cup \mathbb{S}_b \tag{8.24}$$
$$\mathbb{S}_{h_2b} = \mathbb{S}_{h_2} \cup \mathbb{S}_b \tag{8.25}$$

These sets are defined by their boundaries and not directly by performing union on the consistent contractors. This means that:

(i) $\mathbb{S}_{ah}$ is the set of positions $\mathbf{m}$ such that $\langle \mathbf{ba}, \mathbf{bm} \rangle \geq 0 \wedge det(\mathbf{ab}, \mathbf{am}) \geq 0 \wedge det(\mathbf{u}_2, \mathbf{am}) \geq 0$,

(ii) $\mathbb{S}_{ah_1}$ is the set of positions $\mathbf{m}$ such that $det(\mathbf{u}_1, \mathbf{bm}) \leq 0 \wedge det(\mathbf{ab}, \mathbf{am}) \geq 0 \wedge det(\mathbf{u}_2, \mathbf{am}) \geq 0$,

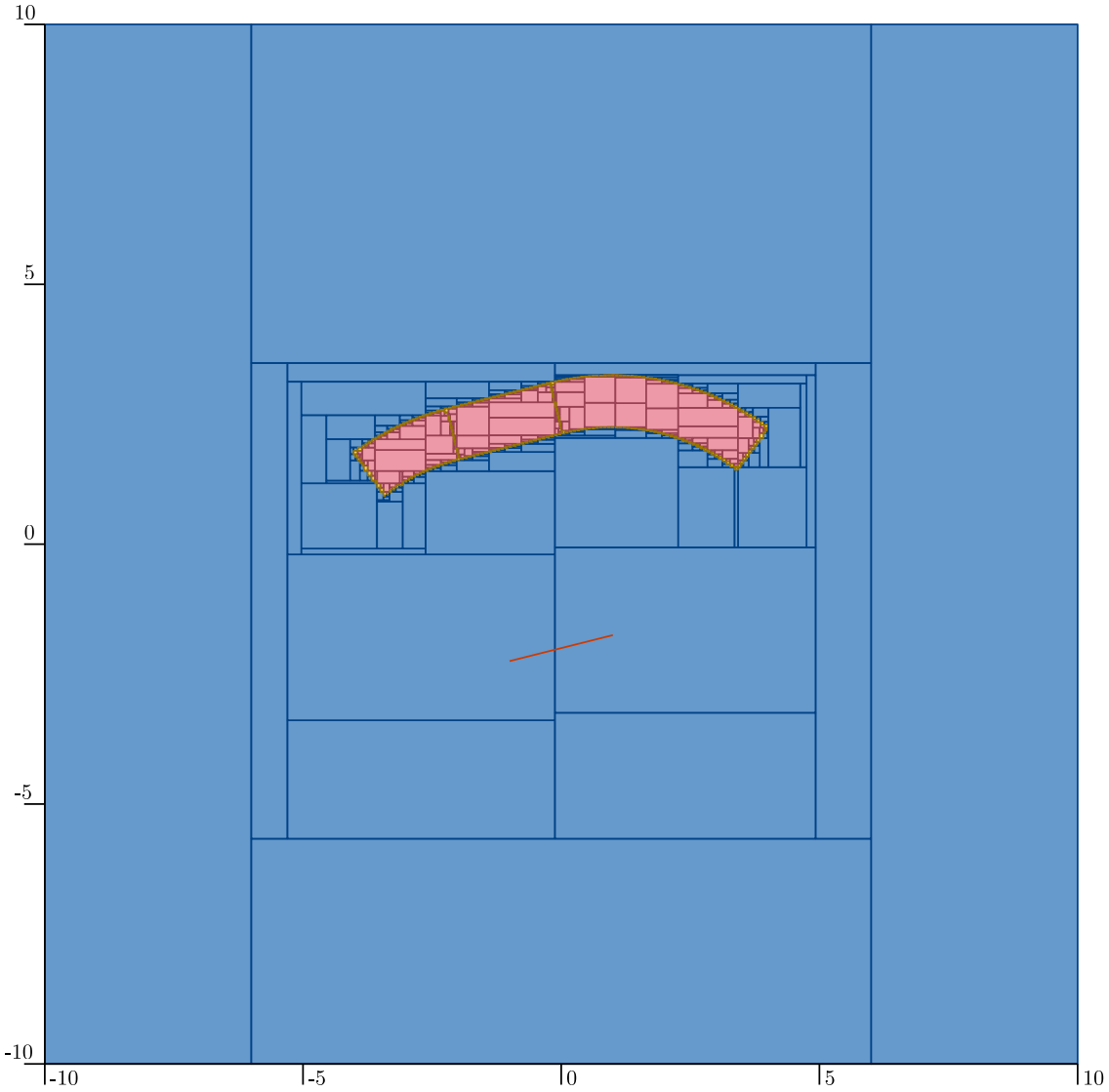


Figure 8.23 Separator on the remoteness constraint with fake boundaries

(iii) $\mathbb{S}_{hb}$ is the set of positions $\mathbf{m}$ such that $det(\mathbf{u}_1, \mathbf{bm}) \leq 0 \wedge det(\mathbf{ab}, \mathbf{am}) \geq 0 \wedge \langle \mathbf{ab}, \mathbf{am} \rangle \geq 0$,

(iv) $\mathbb{S}_{h_2b}$ is the set of positions $\mathbf{m}$ such that $\det(\mathbf{u}_1, \mathbf{bm}) \leq 0 \wedge det(\mathbf{ab}, \mathbf{am}) \geq 0 \wedge det(\mathbf{u}_2, \mathbf{am}) \geq 0$.

Then, using these new sets, boundary overlapping sets can be added in the expression of the separator to avoid fake boundaries. If c_a is true, then the separator on the remoteness is:

$$S_{remoteness} = (\mathcal{S}_a \cap \mathcal{S}_{d,a}) \cup (\mathcal{S}_{h_1} \cap \mathcal{S}_{d,h_1}) \cup (\mathcal{S}_{ah_1} \cap \mathcal{S}_{d,a} \cap \mathcal{S}_{d,h_1}) \tag{8.26}$$

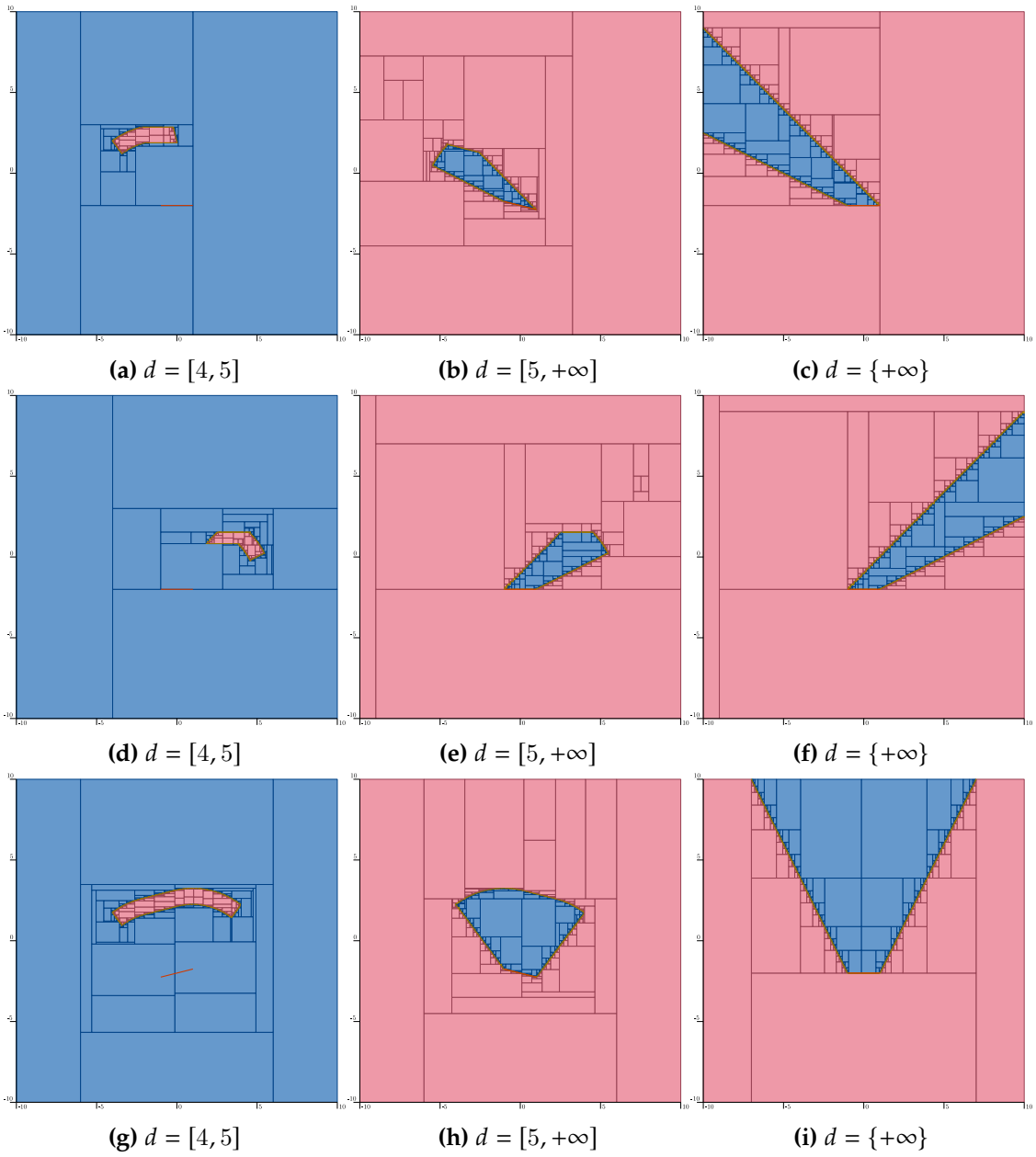
If c_h is true, then the separator on the remoteness is:

$$S_{remoteness} = (\mathcal{S}_a \cap \mathcal{S}_{d,a}) \cup (\mathcal{S}_h \cap \mathcal{S}_{d,h}) \cup (\mathcal{S}_b \cap \mathcal{S}_{d,b}) \cup (\mathcal{S}_{ah} \cap \mathcal{S}_{d,a} \cap \mathcal{S}_{d,h}) \cup (\mathcal{S}_{hb} \cap \mathcal{S}_{d,h} \cap \mathcal{S}_{d,b}) \tag{8.27}$$

And if c_b is true, then the separator on the remoteness is:

$$S_{remoteness} = (\mathcal{S}_{h_2} \cap \mathcal{S}_{d,h_2}) \cup (\mathcal{S}_b \cap \mathcal{S}_{d,b}) \cup (\mathcal{S}_{bh_2} \cap \mathcal{S}_{d,h_2} \cap \mathcal{S}_{d,b}) \tag{8.28}$$

Figure 8.24 shows pavings of the remoteness constraint on an obstacle segment $[\mathbf{a}, \mathbf{b}]$ represented in red, with two unit vectors. The separator is characterizing the set of possible positions $\mathbf{m}$ for the sensor compatible with a measured distance $d = [4, 5]$. Each line represents a different case of $\langle \mathbf{u}_1, \mathbf{ab} \rangle$ and $\langle \mathbf{u}_2, \mathbf{ab} \rangle$ as presented above. Each row represents a different value of measured distance $[d] \in \{[4, 5], [5, +\infty], \{+\infty\}\}$. The pink area represents the set of positions $\mathbf{m}$ compatible with the measured distance d and the unit vectors $\mathbf{u}_1$ and $\mathbf{u}_2$, while the blue area represents the set of positions $\mathbf{m}$ not compatible with this measurement, and the yellow area represents the uncertain set.



(a) $d = [4, 5]$ **(b)** $d = [5, +\infty]$ **(c)** $d = \{+\infty\}$

(d) $d = [4, 5]$ **(e)** $d = [5, +\infty]$ **(f)** $d = \{+\infty\}$

(g) $d = [4, 5]$ **(h)** $d = [5, +\infty]$ **(i)** $d = \{+\infty\}$

Figure 8.24 Paving of the separator on the remoteness constraint

It is noticeable on these figures that once $+\infty \in [d]$, the area surrounding the remoteness domain is characterized as inside by the separator. This area became the only area compatible with the measurement d as soon as $d = \{+\infty\}$.

8.8 State estimation in cycles using the remoteness

The separator on the remoteness constraint can be applied to solve state estimation problem in cycles. The objective is to estimate the initial position of the cycle by using measurements performed along the cycle.

Consider an underwater robot navigating in a pool without obstacles of dimension $10 \times 20\text{m}$. The robot is equipped with an acoustic range sensor that measures the distance to its right wall and returns the closest distance. The robot is controlled in the two-dimensional plane at a constant depth and speed. The robot performs square cycles in the pool by following the timed automata shown in Figure 8.25.

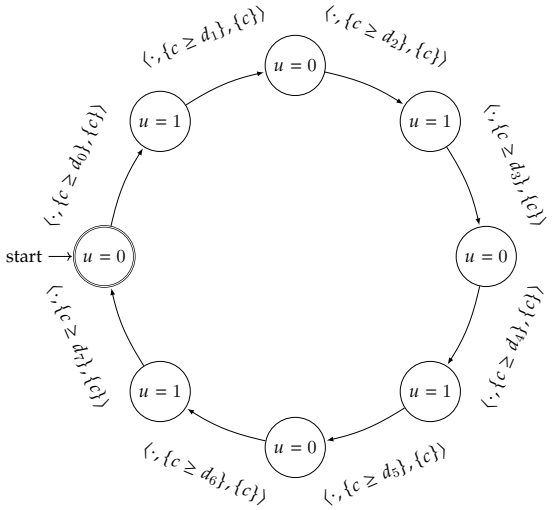


Figure 8.25 Timed automaton of the square cycle in the pool example

By following this automaton, the robot performs a square cycle in the pool. A range measurement y_0 is performed at the beginning of the cycle, and a second range measurement y_1 is performed at the beginning of the second side of the square. Using the remoteness constraint it is then possible to estimate the position of the robot at the beginning of the cycle iteration, denoted by $\mathbf{p}$.

Figure 8.26 shows the trajectory of the robot represented in white in the pool represented in black. The position of the robot when starting the i^{th} iteration of the cycle is represented in yellow, which is also the position of the measurement y_0, and the position of the second measurement y_1 is represented in gray.

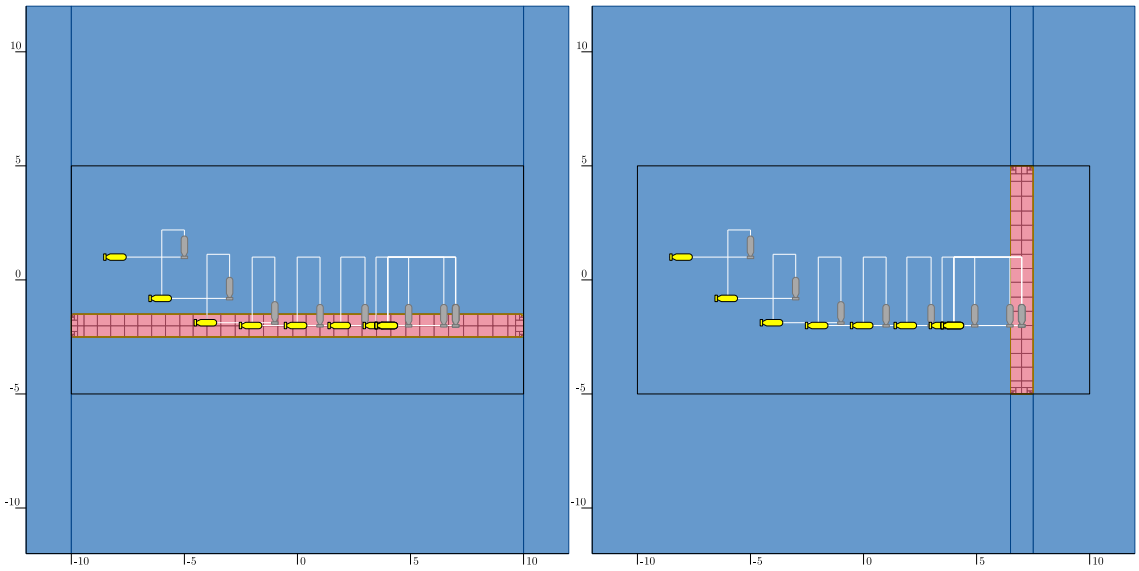
Figure 8.26a represents in pink the set of compatible positions $\mathbf{m}$ for the robot compatible with the measurement y_0 performed at the beginning of the last cycle. This set is estimated using the separator on the remoteness constraint on all the segments of the pool.

Figure 8.26b represents in pink the set of compatible positions $\mathbf{m}$ for the robot compatible with the measurement y_1 performed at the second segment of the last cycle. This set is estimated using the separator on the remoteness constraint on all the segments of the pool.

To solve the state estimation problem, the two sets of compatible positions $\mathbf{m}$ have to be intersected. The set of positions $\mathbf{m}$ compatible with the measurement y_0 is already at the start of the cycle, but for the second measurement, as it is not taken at the start of the cycle, it has to be translated to it. This is done by applying the transformation of the separator [36] using the composition of the two flow functions of the robot between the start of the cycle and the position of the second measurement.

$$f_{fwd}(\mathbf{m}) = \mathbf{m} + \phi_1(d_1, \phi_0(d_0, \mathbf{0})) \tag{8.29}$$

$$f_{bwd}(\mathbf{m}) = \mathbf{m} - \phi_1(d_1, \phi_0(d_0, \mathbf{0})) \tag{8.30}$$



(a) Range measurement y_0 **(b)** Range measurement y_1

Figure 8.26 Trajectory of the robot in the pool and set of possible positions for the sensor compatible with the measurements y_0 and y_1

Remark. *As the transformation is independent of the initial position, the displacement is computed using the flow function applied to an initial condition* $\mathbf{x}_0 = \mathbf{0}$*. This is then used to compute the transformation of the separator whatever the initial position of the cycle.*

Figure 8.27 shows the intersection of the two sets of compatible positions for the cyclic state of the last iteration of the cycle performed by the robot in the pool. The set in pink well encloses the real position of the robot at the beginning of its last cycle iteration.

8.9 Conclusion

This chapter has addressed fundamental challenges in set-based state estimation, developing novel theoretical frameworks and practical solutions that significantly advance the field's capability to handle complex estimation problems in constrained environments. The work presented establishes a comprehensive foundation for robust state estimation in scenarios where traditional methods face inherent limitations, particularly in underwater robotics applications where GNSS is unavailable and environmental constraints are paramount.

A central contribution of this work has been the identification and a method to deal with fake boundaries in set-methods. These fake boundaries emerge at the union of adjacent contractors, and particularly occurs with geometric constraints. These fake boundaries not only compromise the accuracy of the estimated sets but also introduce computational inefficiencies that increase the complexity of the estimation problem.

A significant theoretical contribution of this work has been the comprehensive topological and Hausdorff stability analysis of set operators, particularly union and intersection operations fundamental to contractor-based methods. This analysis has provided the mathematical foundation for understanding when and why fake boundaries emerge, leading to a rigorous classification of estimation problems based on their stability properties.

The distinction between Hausdorff-stable and non-stable problems has proven to be fundamental in determining the appropriate solution strategy. Hausdorff-stable problems exhibit predictable behavior under small perturbations, allowing for the development of boundary-preserving techniques that maintain topological fidelity. Conversely, non-

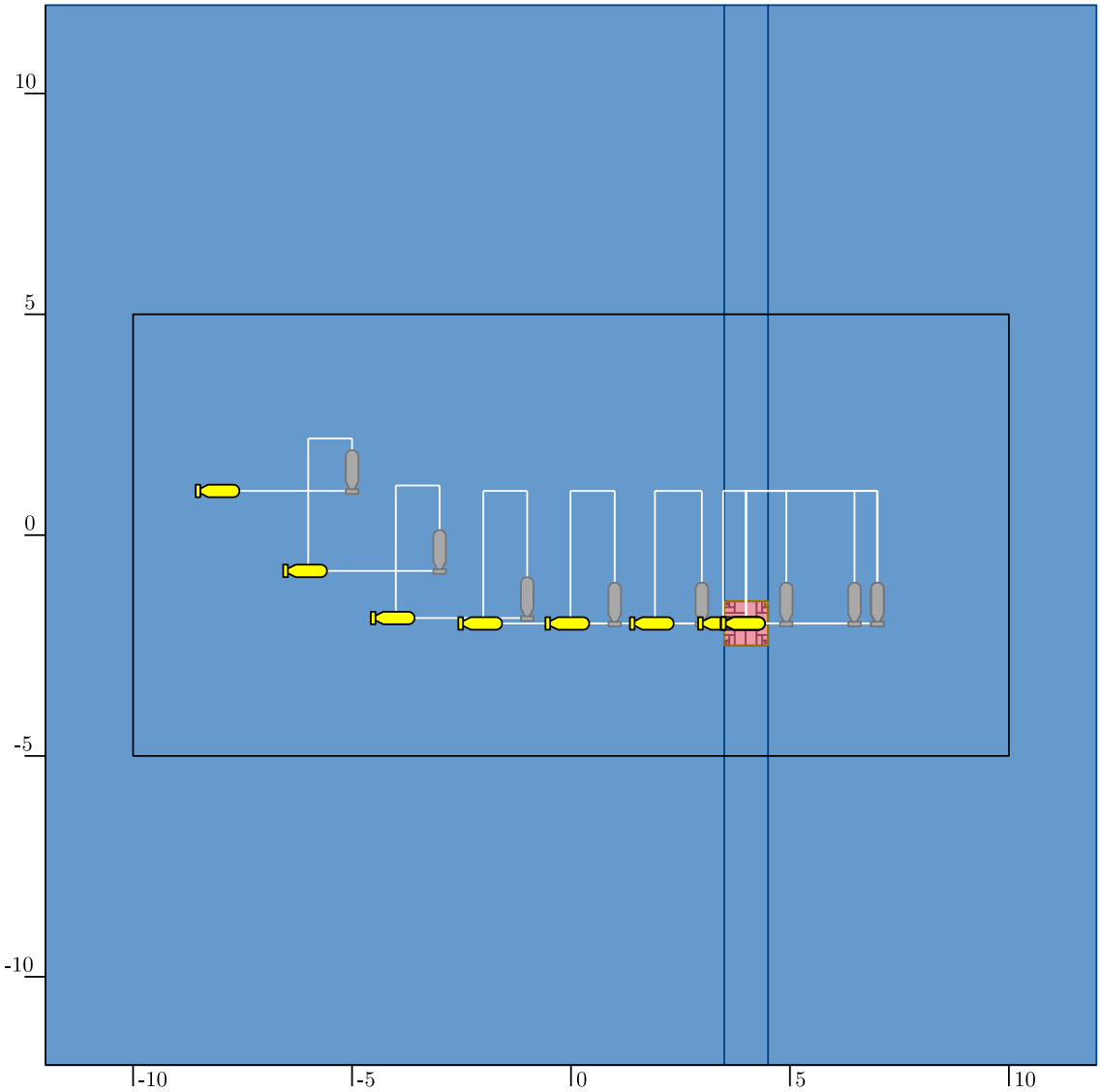


Figure 8.27 Estimation of the cyclic state of the last iteration using the remoteness constraint

stable problems require more sophisticated handling through specialized approaches that account for their inherent sensitivity to discretization effects.

Building upon the stability analysis, two complementary solution strategies have been developed to avoid fake boundaries while preserving computational efficiency. For Hausdorff-stable cases, the boundary preserving form maintains the topological integrity of the solution set by ensuring the boundary of the paved set is preserved.

For general and non-stable cases, the boundary approach focus on building a paving of the boundary of the set of interest, and then to characterize the inner approximation and the outer one by using a predicate test on the sub pavings separated by the boundary. As the boundary has been build by hand, no fake boundaries are introduced in the result.

The practical significance of our theoretical developments has been demonstrated through their application to visibility constraint problems. The classic implementation of visibility constraints has been shown to suffer from fake boundaries, particularly when dealing with the visibility relative to a polygon obstacle.

A novel contribution of this work has also been the development and comprehensive formulation of the separator for remoteness constraints, addressing a previously underexplored aspect of set-based estimation. The remoteness constraint is representing the set of possible position for a sensor respecting a measured distance constraint from segment

obstacles using a sensor with a thick directivity.

This allows to solve state estimation problem for stable cycles navigating a known environment such as a pool. Measurements taken along the cycle could estimate the set of states for the robot compatible with the measurement, which could characterize the set of possible starting points for the cycle in the pool.

9 Conclusion

This thesis introduces a paradigm shift in robotic navigation for GNSS-denied environments, moving from the traditional *localize-then-navigate* approach to an integrated *navigate-while-localizing* methodology. Through the development of cycle-based navigation, we have demonstrated that robots can achieve robust autonomous navigation in challenging environments such as underwater domains, where conventional localization infrastructure is unavailable or unreliable.

9.1 Main Contributions of this Manuscript

The work presented in this manuscript establishes a comprehensive theoretical and practical framework for cycle-based navigation through four interconnected contributions that collectively address the fundamental challenges of autonomous navigation in infrastructure-poor environments.

9.1.1 Cycle Control Framework

The foundation of our approach lies in the introduction of coupled control systems that leverage timed automata synchronized with a dynamical system. By formalizing cycle abstraction and establishing control over input durations, we have created a novel means to control robots along desired trajectories without relying on precise localization. The comparative analysis of multiple controllers—including dead-beat, proportional, sign, and tanh implementations—demonstrates the versatility of this approach in stabilizing cycles using only sparse environmental measurements. Successfully tested both in simulation and in real-world BlueBoat trials navigating along isobaths without GNSS, this method provides compelling evidence of cycle control's viability.

9.1.2 Theoretical Foundations of Cycle Stability

Our theoretical analysis extends beyond empirical validation to establish rigorous mathematical foundations for cycle navigation. Stability is proved at two levels. On one hand, a method to characterize the inner approximation of the positive invariant set of the cycle is presented, which provides a formal guarantee of cycle stability around an equilibrium point. On the other hand, the capture basin of the stable cycle leading to the positive invariant set is computed, which characterizes the set of initial conditions that will lead to a stable cycle. This theoretical framework not only validates the robustness of the approach but also reveals important limitations and sensitivities to initial conditions, providing clear boundaries for the method's applicability and informing practitioners about optimal deployment strategies.

9.1.3 Navigating using Stable Cycles

The introduction of the capture basin associated to a cycle enables cycle-based navigation. By constructing reachability graphs that connect multiple cycles, it is possible to navigate by leaping between locally stable cycles, effectively creating a multi-cycle navigation strategy. This approach is based on the reachability of the capture basin of one cycle from another in dead-reckoning. If dead reckoning is insufficient, a long-range navigation strategy could be implemented by using the cycle formalism to follow an isobath linking a stable cycle to the capture basin of another cycle. This method allows coverage and exploration of worlds Worlds, which are strongly connected subsets of cycles. This multi-cycle framework transforms isolated cycle behaviors into a comprehensive navigation solution capable of handling complex, large-scale missions while maintaining the robustness guarantees of individual cycles.

9.1.4 State Estimation using Interval Methods

The final contribution addresses the challenge of state estimation of the robot during cycle execution. Two contributions are proposed in the manuscript. First, the introduction of a boundary approach to deal with the union of adjacent contractors is proposed to decrease the uncertainty introduced by fake boundaries in the interval-based state estimation. This problem occurs generally with geometric contractors, such as the visibility or cross constraint. Second, the development of separators for remoteness constraints enables the robot to extract meaningful position information from measurements gathered during cycle execution. This is particularly important in underwater environments where traditional localization methods are ineffective, and where the robot must rely on sonar distance measurements to estimate its position. The remoteness concept is not new, but the proposed implementation of the separator for the remoteness constraint is a contribution.

9.2 Paradigm Shift and Theoretical Significance

The cycle-based navigation paradigm represents a fundamental departure from conventional robotic navigation architectures. Traditional approaches require accurate localization as a prerequisite for navigation, creating a dependency on external infrastructure or computationally intensive algorithms. Our approach inverts this relationship, using navigation behaviors as the primary mechanism for both locomotion and localization.

This paradigm shift offers several theoretical advantages. First, it provides natural robustness against localization failures, as the system does not rely on maintaining a global position estimate. Second, it leverages the structure of the environment as an integral part of the navigation solution, rather than treating it as an obstacle to be mapped and avoided. Third, it enables operation in environments where traditional methods fail or are difficult to use, such as underwater domains with limited and expensive localization solutions.

The theoretical guarantees provided by our stability analysis ensure that this paradigm shift is not merely pragmatic but mathematically sound. The formal characterization of convergence conditions and stability regions provides confidence that the system will perform predictably even in the face of environmental uncertainty and measurement noise.

9.3 Experimental Validation and Practical Impact

The real-world validation of our approach on the BlueBoat platform demonstrates the transition from theoretical concept to practical implementation. The successful navigation along underwater isobaths without GNSS represents a significant achievement in

autonomous marine robotics, where traditional navigation methods face severe limitations due to signal attenuation and infrastructure absence.

The experimental results validate not only the technical feasibility of cycle-based navigation but also its practical advantages in terms of computational efficiency and hardware requirements. By eliminating the need for complex mapping algorithms and high-precision sensors, our approach enables autonomous navigation on resource-constrained platforms, potentially democratizing access to autonomous underwater vehicle technology.

The robustness demonstrated in real-world conditions, including environmental disturbances and measurement uncertainties, confirms that the theoretical guarantees translate effectively to practical applications. This validation is particularly significant given the challenging nature of underwater environments, where traditional validation approaches often rely on controlled conditions.

9.4 Limitations and Scope

While our contributions represent significant advances in autonomous navigation, it is important to acknowledge the limitations and scope of the current work. The cycle-based paradigm is particularly well-suited to structured environments where meaningful geometric features can be exploited for navigation. In completely unstructured or highly dynamic environments, the benefits of our approach may be diminished.

The sensitivity to initial conditions, identified through our theoretical analysis, requires careful consideration during deployment. While the capture basins provide formal guarantees for convergence, successful initialization remains a critical factor for system performance. Additionally, the current framework assumes relatively static environmental conditions during cycle execution, which may limit applicability in highly dynamic scenarios.

The computational requirements for interval-based state estimation, while more modest than full SLAM solutions, still represent a consideration for severely resource-constrained platforms. The trade-offs between estimation accuracy and computational load require careful tuning for specific applications.

9.5 Future Research Directions

The work presented in this thesis opens several promising avenues for future research that could significantly extend the impact and applicability of cycle-based navigation.

9.5.1 Global Convergence Characterization

While our theoretical analysis provides local stability guarantees, characterizing global convergence properties remains an open challenge. Understanding the global behavior of multi-cycle systems could enable more sophisticated mission planning and provide stronger theoretical foundations for large-scale deployments.

9.5.2 Automated Cycle Design

The current approach requires manual specification of cycle parameters and environmental features. Developing automated methods for cycle design, potentially through machine learning or optimization techniques, could significantly improve the accessibility and adaptability of the framework. This could include learning optimal cycle parameters from environmental data or adapting cycle behaviors based on performance feedback.

9.5.3 Integration with Learning-Based Methods

The combination of cycle-based navigation with modern machine learning approaches presents exciting opportunities. Reinforcement learning could be employed to optimize cycle parameters or to learn adaptive behaviors for different environmental conditions. Deep learning methods could enhance feature detection and environmental understanding, while maintaining the robust foundation provided by cycle-based control.

9.5.4 Multi-Agent Coordination

Extending the framework to multi-agent systems could enable cooperative navigation strategies where multiple robots coordinate their cycles to achieve collective objectives. This could be particularly valuable for applications such as underwater surveying or search and rescue operations. Moreover, the cycle paradigm could solve the rendezvous problem in multi-agent systems [56, 32], where agents need to meet at a specific location without prior knowledge of their relative positions.

9.5.5 Environmental Adaptation

Developing methods for real-time adaptation to changing environmental conditions could significantly broaden the applicability of cycle-based navigation. This could include online learning of environmental features or adaptive modification of cycle parameters based on performance metrics.

9.5.6 Sensor Fusion and Multi-Modal Perception

While our current work focuses primarily on geometric features, integrating additional sensing modalities such as acoustic, magnetic, or chemical sensors could enrich the environmental understanding and enable navigation in environments where geometric features are insufficient.

9.6 Concluding Remarks

This manuscript demonstrates that robust autonomous navigation in GNSS-denied environments is achievable through a fundamental reconceptualization of the navigation problem. By embracing the cycle-based paradigm and the *navigate-while-localizing* philosophy, we have created a framework that is both theoretically sound and practically viable.

The contributions presented here extend beyond the specific domain of underwater robotics to offer insights applicable to any autonomous system operating in infrastructure-poor environments. The theoretical foundations, validated through real-world experimentation, provide a solid basis for future developments in autonomous navigation.

As autonomous systems increasingly operate in challenging environments where traditional infrastructure is unavailable, the principles and methods developed in this work offer a pathway toward more robust, adaptable, and capable autonomous agents. The cycle-based navigation paradigm represents not just a technical solution, but a conceptual framework that could influence the design of autonomous systems across multiple domains. The journey from theoretical concept to practical implementation demonstrates the value of combining rigorous mathematical analysis with empirical validation. As we continue to push the boundaries of autonomous capabilities, this integrated approach will remain essential for developing solutions that are both scientifically sound and practically impactful.

Through this work, we have taken a significant step toward enabling truly autonomous operation in the world's most challenging environments, opening new possibilities for exploration, monitoring, and intervention in domains previously inaccessible to autonomous systems.

Bibliography

[1] Gnss-denied unmanned aerial vehicle navigation: analyzing computational complexity, sensor fusion, and localization methodologies. *Satellite Navigation*, 6(1), 2025. Article number 4.

[2] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[3] A. Anier and J. Vain. Timed automata based provably correct robot control. In *2010 12th Biennial Baltic Electronics Conference*, pages 201–204, 2010.

[4] Jean-Pierre Aubin. A survey of viability theory. *SIAM Journal on Control and Optimization*, 28(4):749–788, 1990.

[5] Jean-Pierre Aubin, Alexandre M Bayen, and Patrick Saint-Pierre. *Viability theory: new directions*. Springer Science & Business Media, 2011.

[6] Fabrice L.E. BARS, Alain BERTHOLOM, Jan SLIWKA, and Luc JAULIN. Interval slam for underwater robots; a new experiment. *IFAC Proceedings Volumes*, 43(14):42–47, 2010. 8th IFAC Symposium on Nonlinear Control Systems.

[7] Béatrice Bérard and Catherine Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1-2):1–7, 2000.

[8] Lucia Bergantin, Charles Coquet, Amaury Negre, Thibaut Raharijaona, Nicolas Marchand, and Franck Ruffier. Minimalistic in-flight odometry based on two optic flow sensors along a bouncing trajectory. In *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*, pages 1321–1326, 2022.

[9] Franco Blanchini, Stefano Miani, et al. *Set-theoretic methods in control*, volume 78. Springer, 2008.

[10] Quentin Brateau, Fabrice Le Bars, and Luc Jaulin. Considering adjacent sets for computing the visibility region. working paper or preprint, November 2024.

[11] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.

[12] Enrique Carrera and Christian Soria. Positioning of autonomous underwater vehicles using machine learning techniques. 10 2023.

[13] Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173:1079–1100, 07 2009.

[14] Ying Chen and Néstor O. Pérez-Arancibia. Lyapunov-based controller synthesis and stability analysis for the execution of high-speed multi-flip quadrotor maneuvers. In *2017 American Control Conference (ACC)*, pages 3599–3606, 2017.

[15] Felix L Chernousko. Ellipsoidal state estimation for dynamical systems. *Nonlinear analysis: Theory, methods & applications*, 23(11):1421–1426, 1994.

[16] William W Cochran, Henrik Mouritsen, and Martin Wikelski. Migrating songbirds recalibrate their magnetic compass daily from twilight cues. *Science*, 304(5669):405–408, 2004.

[17] J Collin. *Navigation and Surveying with GPS*. Institute of Navigation, Alexandria, VA, 2000.

[18] Iain D Couzin, Jens Krause, et al. Self-organization and collective behavior in vertebrates. *Advances in the Study of Behavior*, 32(1):10–1016, 2003.

[19] José de Jesus Castillo-Zamora, Amaury Negre, Jean-Marc Ingargiola, Abdoullah Ndoye, Florian Pouthier, Jonathan Dumon, Sylvain Durand, Nicolas Marchand, and Franck Ruffier. Synchronization of a new light-flashing shield with an external-triggered camera. *IEEE Sensors Letters*, 7(8):1–4, 2023.

[20] Jacqueline Degen, Andreas Kirbach, Lutz Reiter, Konstantin Lehmann, Philipp Norton, Mona Storms, Miriam Koblofsky, Sarah Winter, Petya B Georgieva, Hai Nguyen, et al. Exploratory behaviour of honeybees during orientation flights. *Animal Behaviour*, 102:45–57, 2015.

[21] Benoît Desrochers and Luc Jaulin. Thick set inversion. *Artificial Intelligence*, 249:1–18, 2017.

[22] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[23] Carsten Egevang, Iain J Stenhouse, Richard A Phillips, Aevar Petersen, James W Fox, and Janet RD Silk. Tracking of arctic terns sterna paradisaea reveals longest animal migration. *Proceedings of the National Academy of Sciences*, 107(5):2078–2081, 2010.

[24] A. Ehambram, B. Wagner, and L. Jaulin. *Localization in Urban Environments. A Hybrid Interval-probabilistic Method*. Gottfried Wilhelm Leibniz Universität, 2023.

[25] Karl von Frisch. *The dance language and orientation of bees*. Harvard University Press, 1993.

[26] Gabriel Gattaux, Roxane Vimbert, Antoine Wystrach, Julien R. Serres, and Franck Ruffier. Antcar: Simple Route Following Task with Ants-Inspired Vision and Neural Model. working paper or preprint, February 2023.

[27] Antoine Girard, Colas Le Guernic, and Oded Maler. Zonotope/hyperplane intersection for hybrid systems reachability analysis. *Hybrid Systems: Computation and Control*, pages 215–228, 2008.

[28] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. Hybrid dynamical systems. *IEEE Control Systems Magazine*, 29(2):28–93, 2009.

[29] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Lecture Notes in Computer Science. Springer, 2002.

[30] Rémy Guyonneau. *Méthodes Ensemblistes Pour La Localisation En Robotique Mobile*. PhD Thesis, Angers, January 2013.

[31] Graeme C. Hays, Luciana C. Ferreira, Ana M.M. Sequeira, Mark G. Meekan, Carlos M. Duarte, Helen Bailey, Fred Bailleul, W. Don Bowen, M. Julian Caley, Daniel P. Costa, Victor M. Eguíluz, Sabrina Fossette, Ari S. Friedlaender, Nick Gales, Adrian C. Gleiss, John Gunn, Rob Harcourt, Elliott L. Hazen, Michael R. Heithaus, Michelle Heupel, Kim Holland, Markus Horning, Ian Jonsen, Gerald L. Kooyman, Christopher G. Lowe, Peter T. Madsen, Helene Marsh, Richard A. Phillips, David Righton, Yan Ropert-Coudert, Katsufumi Sato, Scott A. Shaffer, Colin A. Simpfendorfer, David W. Sims, Gregory Skomal, Akinori Takahashi, Philip N. Trathan, Martin Wikelski, Jamie N. Womble, and Michele Thums. Key questions in marine megafauna movement ecology. *Trends in Ecology & Evolution*, 31(6):463–475, 2016.

[32] Ali Jadbabaie, Jie Lin, and A Stephen Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[33] Luc Jaulin. *Solution globale et garantie de problemes ensemblistes : applications a l'estimation non lineaire et a la commande robuste*. PhD thesis, 1994. Thèse de doctorat dirigée par Walter, Éric Sciences appliquées Paris 11 1994.

[34] Luc Jaulin. Path planning using intervals and graphs. *Reliable Computing Journal*, 7(1):1–15, 2001.

[35] Luc Jaulin. A nonlinear set membership approach for the localization and map building of underwater robots. *IEEE Transactions on Robotics*, 25(1):88–98, 2009.

[36] Luc Jaulin. Separator algebra for state estimation. In *SMART 2015*, 2015.

[37] Luc Jaulin. A boundary approach for set inversion. *Engineering Applications of Artificial Intelligence*, 100:104184, 2021.

[38] Luc Jaulin, Jean-Louis Boimond, and Laurent Hardouin. Estimation of Discrete-Event Systems using Interval Computation. *Reliable Computing Journal*, 5(2):165–173, 1999.

[39] Luc Jaulin and Benoît Desrochers. Introduction to the algebra of separators with application to path planning. *Engineering Applications of Artificial Intelligence*, 33:141–147, 2014.

[40] Luc Jaulin, Michel Kieffer, Olivier Didrit, Eric Walter, Luc Jaulin, Michel Kieffer, Olivier Didrit, and Éric Walter. *Interval analysis*. Springer, 2001.

[41] R.E. Kalman. On the general theory of control systems. *IFAC Proceedings Volumes*, 1(1):491–502, 1960. 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960.

[42] Tomasz Kapela and Piotr Zgliczyński. A lohner-type algorithm for control systems and ordinary differential inclusions, 2007.

[43] Maurice Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, 1953.

[44] Ralph Kearfott. A fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear equations and global optimization. *ACM Transactions on Mathematical Software*, 21, 06 1995.

[45] Hassan K Khalil. Nonlinear systems.

[46] Michel Kieffer, Luc Jaulin, Éric Walter, and Dominique Meizel. Robust autonomous robot localization using interval analysis. *Reliable computing*, 6(3):337–362, 2000.

[47] James C Kinsey, Ryan M Eustice, and Louis L Whitcomb. A survey of underwater vehicle navigation: Recent advances and new challenges. *IFAC Proceedings Volumes*, 39(16):1–12, 2006.

[48] Thomas Koshy. *Discrete mathematics with applications*. Elsevier, 2004.

[49] Wolfgang Kühn. Rigorously computed orbits of dynamical systems without the wrapping effect. *Computing*, 61(1):47–67, 1998.

[50] Marit Lahme, Andreas Rauh, and Guillaume Defresne. Interval observer design for an uncertain time-varying quasi-linear system model of lithium-ion batteries. In *2024 European Control Conference (ECC)*, pages 3696–3702, 2024.

[51] Miros Law Wysocki and Agata Darmochwa. Subsets of topological spaces. *Journal of Formalized Mathematics*, 1, 1989.

[52] Michel Le Gallo. *Motifs bretons et celtiques*. Méthode de construction. Coop Breizh, 4e éd edition, 2001.

[53] John J Leonard and Alexander Bahr. Autonomous underwater vehicle navigation. *Springer handbook of ocean engineering*, pages 341–358, 2016.

[54] Andrew D Lewis. A Mathematical Approach to Classical Control.

[55] Hong Li, Mingyong Liu, and Kun Liu. Bio-inspired geomagnetic navigation method for autonomous underwater vehicle. *Journal of Systems Engineering and Electronics*, 28(6):1203–1209, 2017.

[56] Zhiyun Lin, Mireille Broucke, and Bruce Francis. Asymptotic stability of an n-dimensional alpha-flocking model. *IEEE Transactions on Automatic Control*, 49(7):1083–1096, 2004.

[57] Kenneth J Lohmann and Catherine MF Lohmann. There and back again: natal homing by magnetic navigation in sea turtles and salmon. *Journal of Experimental Biology*, 222(Suppl_1):jeb184077, 2019.

[58] KJ Lohmann, Paolo Luschi, and GC Hays. Goal navigation and island-finding in sea turtles. *Journal of Experimental Marine Biology and Ecology*, 356(1-2):83–95, 2008.

[59] David G. Luenberger. Observing the state of a linear system. *IEEE Transactions on Military Electronics*, 8(2):74–80, 1964.

[60] Nicolas Marchand, Sylvain Durand, and Jose Fermi Guerrero Castellanos. A general formula for event-based stabilization of nonlinear systems. *IEEE Transactions on automatic control*, 58(5):1332–1337, 2012.

[61] Nicolas Marchand, Sylvain Durand, and Jose Fermi Guerrero Castellanos. A general formula for event-based stabilization of nonlinear systems. *IEEE Transactions on Automatic Control*, 58(5):1332–1337, 2013.

[62] Francesco Maurelli, Szymon Krupiński, Xianbo Xiang, and Yvan Petillot. Auv localisation: a review of passive and active techniques. *International Journal of Intelligent Robotics and Applications*, 6(2):246–269, 2022.

[63] Larry Mayer, Martin Jakobsson, Graham Allen, Boris Dorschel, Robin Falconer, Vicki Ferrini, Geoffroy Lamarche, Helen Snaith, and Pauline Weatherall. The nippon foundation—gebco seabed 2030 project: The quest to see the world's oceans completely mapped by 2030. *Geosciences*, 8(2), 2018.

[64] Ramon E Moore. *Interval analysis*, volume 4. prentice-Hall Englewood Cliffs, 1966.

[65] J.R. Munkres. *Topology*. Featured Titles for Topology. Prentice Hall, Incorporated, 2000.

[66] Jordan Ninin. Global optimization based on contractor programming: An overview of the ibex library. In Ilias S. Kotsireas, Siegfried M. Rump, and Chee K. Yap, editors, *Mathematical Aspects of Computer and Information Sciences*, pages 555–559, Cham, 2016. Springer International Publishing.

[67] R Orosco-Guerrero, M Velasco-Villa, and E Aranda-Bricaire. Discrete-time controller for a wheeled mobile robot. In *Proc. XI Latin-American Congress of Automatic Control, La Habana, Cuba*, 2004.

[68] Fernando Molano Ortiz, Matteo Sammarco, Luís Henrique M. K. Costa, and Marcin Detyniecki. Applications and services using vehicular exteroceptive sensors: A survey. *IEEE Transactions on Intelligent Vehicles*, 8(1):949–969, 2023.

[69] P. Palaniyandi. On computing poincaré map by hénon method. *Chaos, Solitons & Fractals*, 39(4):1877–1882, 2009.

[70] Yannis P Papastamatiou, Yuuki Y Watanabe, Urška Demšar, Vianey Leos-Barajas, Darcy Bradley, Roland Langrock, Kevin Weng, Christopher G Lowe, Alan M Friedlander, and Jennifer E Caselle. Activity seascapes highlight central place foraging strategies in marine predators that never stop swimming. *Movement Ecology*, 6:1–15, 2018.

[71] KyuCheol Park, Hakyoung Chung, and Jang Gyu Lee. Dead reckoning navigation for autonomous mobile robots. *IFAC Proceedings Volumes*, 31(3):219–224, 1998. 3rd IFAC Symposium on Intelligent Autonomous Vehicles 1998 (IAV'98), Madrid, Spain, 25-27 March.

[72] Liam Paull, Sajad Saeedi, Mae Seto, and Howard Li. Auv navigation and localization: A review. *IEEE Journal of oceanic engineering*, 39(1):131–149, 2013.

[73] Lawrence Perko. *Differential Equations and Dynamical Systems*, volume 7 of *Texts in Applied Mathematics*. Springer, 2013. A classic reference on phase portraits, trajectories, and stability theory.

[74] Henri Poincaré. Sur le problème des trois corps et les équations de la dynamique. *Acta Mathematica*, 13:1–270, 1890. Seminal work introducing qualitative analysis of orbits in dynamical systems.

[75] Florian Pouthier, Sylvain Durand, Nicolas Marchand, Jonathan Dumon, Abdoullah Ndoye, Amaury Negre, Pierre Susbielle, Jose Castillo-Zamora, J. Fermi Guerrero Castellanos, and Franck Ruffier. Guaranteed Self-Triggered Control of Disturbed Systems: A Set Invariance Approach. *International Journal of Robust and Nonlinear Control*, 2025.

[76] A. Rauh, L. Senkel, E. Auer, and H. Aschemann. Interval methods for real-time capable robust control of solid oxide fuel cell systems. *Mathematics in Computer Science*, 8:525–542, 2014.

[77] Simon Rohou, Benoit Desrochers, and Fabrice Le Bars. The Codac library. *Acta Cybernetica*, 26(4):871–887, Mar. 2024.

[78] Simon Rohou, Benoît Desrochers, and Fabrice Le Bars. The codac library. *Acta Cybernetica*, Mar. 2024.

[79] Simon Rohou, Peter Franek, Clément Aubry, and Luc Jaulin. Proving the existence of loops in robot trajectories. *The International Journal of Robotics Research*, 37(12):1500–1516, 2018.

[80] Simon Rohou, Luc Jaulin, Lyudmila Mihaylova, Fabrice Le Bars, and Sandor M. Veres. Guaranteed computation of robot trajectories. *Robotics and Autonomous Systems*, 93:76–84, 2017.

[81] Daniel L Rudnick. The 2023-2024 el niño in the california current system as observed by the california underwater glider network. In *2024 Ocean Sciences Meeting*. AGU, 2024.

[82] Patrick Saint-Pierre. Approximation of the viability kernel. *Appl. Math. Optim.*, 29(2):187–209, March 1994.

[83] Christian Schoppmeyer, Martin Hüfner, Subanatarajan Subbiah, and Sebastian Engell. Timed automata based scheduling for a miniature pipeless plant with mobile robots. In *2012 IEEE International Conference on Control Applications*, pages 240–245, 2012.

[84] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT Press, 2nd edition, 2011.

[85] Jean-Jacques E Slotine and Weiping Li. Nonlinear applied control. *Li, W., Ed*, 1991.

[86] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1986.

[87] Cyrill Stachniss, Giorgio Grisetti, and Wolfram Burgard. Information gain-based exploration using rao-blackwellized particle filters. In *Robotics: Science and systems*, volume 2, pages 65–72, 2005.

[88] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[89] Gerald Teschl. *Ordinary differential equations and dynamical systems*, volume 140. American Mathematical Soc., 2012.

[90] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[91] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, Cambridge, MA, 2005.

[92] Tammo Tom Dieck. *Algebraic topology*, volume 8. European Mathematical Society, 2008.

[93] Arjan J Van Der Schaft and Hans Schumacher. *An introduction to hybrid dynamical systems*, volume 251. springer, 2007.

[94] K Vickery. Acoustic positioning systems: A practical overview of current systems. In *Proceedings of Autonomous Underwater Vehicle Conference*, pages 5–17. IEEE, 1998.

[95] Rui Wang, Ping Luo, Yong Guan, Hongxing Wei, Xiaojuan Li, Jie Zhang, and Xiaoyu Song. Timed automata based motion planning for a self-assembly robot system. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5624–5629, 2014.

[96] Xiaotian Wang, Xinnan Fan, Pengfei Shi, Jianjun Ni, and Zhongkai Zhou. An overview of key slam technologies for underwater scenes. *Remote Sensing*, 15(10), 2023.

[97] PW Webb and Raymond S Keyes. Swimming kinematics of sharks. *Fish. Bull*, 80(4):803–812, 1982.

[98] Rüdiger Wehner, Mandyam V Srinivasan, et al. Path integration in insects. *The neurobiology of spatial behaviour*, pages 9–30, 2003.

[99] Daniel Weihs. Stability versus maneuverability in aquatic locomotion. *Integrative and Comparative Biology*, 42(1):127–134, 2002.

[100] Anthony Welte, Luc Jaulin, Martine Ceberio, and Vladik Kreinovich. Avoiding Fake Boundaries in Set Interval Computing. *Journal of Uncertain Systems*, 11(2):137 – 148, 2017.

[101] Louis C. Westphal. *A special control law: deadbeat control*, pages 461–471. Springer US, Boston, MA, 2001.

[102] Stephen Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*, volume 2 of *Texts in Applied Mathematics*. Springer, 2003. Covers invariant manifolds, orbits, and bifurcations in detail.

[103] Daniel Wilczak and Piotr Zgliczyński. C^{r}-lohner algorithm. *Schedae Informaticae*, 2011.

[104] TC Williams and JM Williams. The orientation of transoceanic migrants. In *Bird migration: Physiology and ecophysiology*, pages 7–21. Springer, 1990.

[105] Jonatan Scharff Willners, Yaniel Carreno, Shida Xu, Tomasz Łuczyński, Sean Katagiri, Joshua Roe, Èric Pairet, Yvan Petillot, and Sen Wang. Robust underwater slam using autonomous relocalisation. *IFAC-PapersOnLine*, 54(16):273–280, 2021. 13th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2021.

[106] Mahdi Yahyaei, Georg Seifert, Thomas Hempen, and Werner Huber. Review of exteroceptive sensors for autonomous driving. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 4005–4010, 2022.

[107] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE, 1997.

[108] Junyi Yang, Yutong Yao, and Donghe Yang. Particle filter based on harris hawks optimization algorithm for underwater visual tracking. *Journal of Marine Science and Engineering*, 11(7), 2023.

[109] V. Zaitsev, O. Gvirsman, U. Ben Hanan, A. Weiss, A. Ayali, and G. Kosa. Locust-inspired miniature jumping robot. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 553–558, 2015.

[110] Fumin Zhang and Naomi Ehrich Leonard. Cooperative filters and control for cooperative exploration. *IEEE Transactions on Automatic Control*, 55(3):650–663, 2010.

[111] Günter M Ziegler. *Lectures on polytopes*. Springer, 1995.