Observation missions with UAVs : defining and learning models for active perception and proposition of an architecture enabling repeatable distributed simulations

Christophe Reymann

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut National des Sciences Appliquées de Toulouse

---

Présentée et soutenue par

Christophe REYMANN

Le 8 juillet 2019

Missions d'observations pour des drones : definition et apprentissage de modèles pour la perception active, et proposition d'une architecture permettant des simulations distribuées répétables.

Ecole doctorale : **SYSTEMES**

Spécialité : **Informatique et Robotique**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Simon LACROIX

Jury

M. Jean-Baptiste MOURET, Rapporteur
M. Olivier SIMONIN, Rapporteur
M. David FILLIAT, Examinateur
Mme Janette CARDOSO, Examinatrice
M. Rachid ALAMI, Examinateur
M. Simon LACROIX, Directeur de thèse

Résumé

Cette thèse se focalise sur des tâches de perceptions pour des drones à voilures fixes (unmanned aircraft vehicles ou UAV en anglais). Lorsque la perception est la finalité, un bon modèle d'environnement couplé à la capacité de prédire l'impact de futures observations sur celui-ci est crucial. La *perception active* traite de l'intégration forte entre modèles de perception et processus de raisonnement, permettant au robot d'acquérir des informations pertinentes à propos du statut de la mission et de replanifier sa trajectoire de mesure en réaction à des évènements et résultats imprévisibles.

Ce manuscrit décrit deux approches pour des tâches de *perception active*, dans deux scénarios radicalement différents.

Le premier est celui de la cartographie des phénomènes météorologiques de petite échelle et fortement dynamiques, en particulier de nuages de type cumulus. Les objets à cartographier sont ici de grand volume et fortement dynamiques, alors que les capteurs récoltent des informations ponctuelles le long de la trajectoire de l'UAV. Ceci rend construction d'un modèle précis de ces processus particulièrement difficile, le raisonnement devant être capable de traiter des données très éparses. L'approche présentée utilise la *régression par processus Gaussien* pour construire un modèle d'environnement, les hyper-paramètres étant appris en ligne. Des métriques de gain d'information sont introduites pour évaluer la qualité de futures trajectoires d'observation. Un algorithme de planification stochastique est utilisé pour optimiser une fonction d'utilité équilibrant maximisation du gain d'information avec des buts de minimisation du coût énergétique. La dynamique du robot, ainsi que les contraintes environnementales comme le vent, sont prise en compte et utilisées pour garantir des trajectoires réalisables.

Dans le second scénario, un UAV cartographie des champs de grandes cultures pour les besoins de l'agriculture de précision. Le phénomène à observer est maintenant statique et l'environnement est peu ou prou une surface bidimensionnelle. Utilisant le résultat d'un algorithme de *localisation et cartographie simultanée* (SLAM), une approche nouvelle pour la construction d'un modèle d'erreurs relatives est proposée. Ce modèle est appris à partir d'attributs provenant des structures de données du SLAM, ainsi que de la topologie sous-jacente du graphe de covisibilité formé par les observations.

Dans les deux scénarios, des résultats de simulations réalistes sont présentés et commentés.

Si la référence absolue en robotique est le test sur le terrain en conditions réalistes, un tel déploiement est souvent impossible, que ce soit pour des raisons de temps ou budgétaires, ou encore parce que les robots capables physiquement de mener à bien la mission sont encore en développement. Ainsi dans cette thèse, tous les développement ont été testés en simulation. Pour garantir que les résultats produits en simulation soient aussi précis que possible, des modèles d'environnements

ainsi que modèles dynamiques aussi réalistes que possible doivent être utilisées. La plupart des simulations effectuées en robotique sont faites en temps réel, en essayant de faire correspondre l'avancement du temps simulé avec le temps d'horloge réel. Cependant pour des simulations complexes, des résultats précis ne sont pas possible à produire en temps réel, et la synchronisation des simulateurs devient une tâche ardue.

Une analyse des problématiques de simulations en robotique est proposée. Se focalisant sur la problématique de gestion de l'avancement tu temps et de la synchronisation de simulateurs hétérogènes dans une architecture distribuée, une solution originale basée sur une architecture décentralisée est proposée.

Abstract

This thesis focuses on perception tasks for an unmanned aerial vehicle (UAV). When sensing is the finality, having a good environment model as well as being capable of predicting the impacts of future observations is crucial. *Active perception* deals with integrating tightly perception models in the reasoning process, enabling the robot to gain knowledge about the status of its mission and to replan its sensing trajectory to react to unforeseen events and results.

This manuscript describes two approaches for *active perception* tasks, in two radically different settings.

The first one deals with mapping highly dynamic and small scale meteorological phenomena such as cumulus clouds. Here the object to be mapped is three dimensional and highly dynamic, whereas the sensors gather punctual observations along the trajectory of the UAV. Building an accurate model of the process is extremely challenging, and reasoning must be performed on incomplete data. The presented approach uses *Gaussian Process Regression* to build environment models, learning its hyperparameters online. Normalized marginal information metrics are introduced to compute the quality of future observation trajectories. A stochastic planning algorithm is used to optimize an utility measure balancing maximization of theses metrics with energetic minimization goals. The dynamics of the robot as well as environmental constraints such as wind are taken into account and used to compute feasible trajectories.

The second setting revolves around mapping crop fields for precision agriculture purposes. The phenomena to observe is now static, and the environment roughly a two dimensional surface. Using the output of a monocular graph Simultaneous Localization and Mapping (SLAM) algorithm, a novel approach to building a relative error model is proposed. This model is learned both from features extracted from the SLAM algorithm's data structures, as well as the underlying topology of the covisibility graph of the observations.

In both cases, results on realistic simulations are presented and discussed.

If the golden standard of robotics is testing in the field with realistic conditions, it is often not feasible either due to time or financial constraints, or because robots capable of sensing and following the produced trajectories are still in development. In this thesis and for all of these reasons, all developments have been tested in simulation. Most robotics simulation are done in real time, trying to match simulator speed with wall clock advancement. However when one deals with complex simulations, accurate results are not possible in real time, and synchronization of the simulators becomes a daunting task.

An analysis of the simulation issue in robotics is proposed. Focusing on the problem of managing time advancement of multiple interconnected simulators, a novel solution based on a decentralized scheme is presented.

Acknowledgments

A warm thank you to all colleagues, friends and family who helped me through my doctorate and during the writing of this thesis.

To the PhD students of the RIS group for all the wacky conversations and the hearty laughs,

to Mohammed for his precious collaboration,

to Simon for his exceptional humane qualities,

to all friends who helped me during hard times,

to my parents for their unconditional love and support,

to Samuel, always teeming with joy and mischief.

Contents

Introduction

Motivations: active perception

Autonomous robotics has been structured since its inception by the *sense - plan - act* loop paradigm. The robot gathers data to build a representation of its environment, then plans its actions in this environment model according to its own dynamics model and goals, and finally these actions are fed to a controller responsible of carrying them out in the real world using the robots actuators. Sensory inputs then allow to monitor the plan execution, to update the environment representation, to compute updated plans, and so on.

However such a simple view with clearly separated concerns is but an illusion: it is not possible to develop the solutions to one of these tasks without knowledge of the others. Indeed the sensing shall build an environment model on which the planning algorithm can make relevant requests. Conversely the planning algorithm needs to know what kind of environment model it will be provided with, along with the expected precision of this model given the robots sensors and the complexity of the representation. It also needs to incorporate the acting capabilities of the robot to account for their specificity and imprecision, and in also consider a specific type of action: perceiving.

Indeed what characterizes a robot is its *embodiment* in our world, which is too complex and unpredictable to model perfectly and therefore forces the roboticist to develop simplified models and account for the associated uncertainties. The quality of models is therefore central to the success of a robot's mission: it may achieve its tasks with bad algorithms but good models, a good algorithm is useless without a good model and sensory data to populate it.

Hence sensors and perception models are central in the robotic architecture. Good models not only allow to integrate sensory data to build a faithful map of the environment and relate the robots action in this map, but are also predictive. Allowing to infer the future state of the world model given predicted perception actions can yield the efficient achievement of a series of robot tasks, that can be set as "which information on the world are required to properly achieve the task?". This overal approach is coined as *active perception*, which encourages even tighter coupling between sensing, planning and acting by having the building and maintenance of the environment model as one of the primary goals of the robot. It is a driving force in numerous real world scenarios, especially in interaction scenarios, any time the information available on the environment, including on interacting actors, are incomplete or uncertain.

Recently, the success of *deep learning end-to-end* techniques, taking sensory data as input of a neural network and producing actions as controller inputs has shown the power of tight integration of all three concerns. This new techniques effectively remove an intermediate *planning* stage, deriving directly actions from

sensory inputs. Despite these successes, deep learning techniques alone have shown limited success in constructing revisable abstractions during the course of the mission. Memory cells and recurrent networks have been introduced, but these networks remain untrainable and error prone except for the simplest usages. However, these results question the classical separation of concerns of the *sense - plan -act* paradigm.

In this thesis, active perception is a direct consequence of the mapping missions that were the motivations behind the *Skyscanner* and *Precidrone* projects, which objectives are respectively mapping clouds with a fleet of UAVs and mapping crop field with a UAV. A tighter coupling between sensing and planning through the development of rich models along with ways to query them is the driving force behind the developments presented in the first two research contributions of the thesis developped within the context of these projects.

Repeatable Simulations Field experimentation is the gold standard of validation in robotics. However as systems and algorithms grow in complexity, it is not enough anymore. Due to costs and time constraints, deployment of robots - or fleets of robots - in varied environments and extensive statistical validation through repeated experiments becomes hardly possible. Moreover comparisons between field experiments of different teams is not an easy task, as we try more and more to deploy our robots in uncontrolled environments exhibiting high variability. This often prevents to draw meaningful conclusions, whether testing reproducibility or comparing approaches.

As an intermediate step, validation of algorithms in simulation, especially at an early stage, is the only answer to this dilemma. Quite a number of dedicated simulators dedicated to robotics and autonomous vehicles have been developed. However integrating simulators with distributed, component based architectures, deploying simulations with multiple robots as well as multiple - possibly heterogeneous - simulators is not trivial. For the most part, no thought has been given in the robotics community on how to ensure the consistency and repeatability of such simulations.

Arising from the issues faced during the development of the simulations needed for the first two contributions of this thesis, the third contribution argues for the need for consistent and repeatable simulations in robotics and proposes a distributed architecture for time management as a first step towards this goal. It has been inspired from the works of the *Parallel and Discrete Event Simulation* community and the *High Level Architecture (HLA)* simulation standard, but its novel, completely decentralized, architecture as well as its simplicity may make it easier to integrate in robotics architectures.

Structure of the manuscript

This manuscript consists of a collection of articles, either already published or in the course of publication. Chapter 2 and 3 contain additional material that was not included in the submitted articles.

Chapter 1 deals with the problem of mapping atmospheric phenomena using a fleet of drones, studied in the context of the *Skyscanner* project. It presents a solution involving the usage of Gaussian Process, for which the hyperparameters are learned online, in combination with a stochastic sampling exploration technique. Extensive statistical results from realistic simulations involving large-eddy atmospheric simulations are presented, including an in-depth analysis of the evolution and learning of the Gaussian Process hyperparameters.

The material of this chapter has been published as is in the journal *Autonomous Robots* [Reymann 2018] (a preliminary version had been published in [Renzaglia 2016]) and some of which is the product of collaboration:

- The usage of the stochastic optimization scheme (section 1.3.2.2) has been developed in collaboration with Alessandro Renzaglia.
- The cloud simulations have been produced by Fayçal Lamraoui (section 1.4.1.1).
- The UAV control model (appendix 1.A) has been developed by Murat Bronz.

The rest is my own contribution, including all software developments.

Chapter 2 argues for the need of repeatable simulations in robotics, proposing a novel approach for time management in distributed simulation. It also argues for more vertically modular software architectures, instead on relying on jack-of-all-trades frameworks. This work has been partly motivated by a participation in the development of a simulation for the *Skyscanner* project [Bailon-Ruiz 2017].

The core material of this chapter has been submitted for publication to the Software Quality, Reliability and Security conference [Reymann 2019a].

The formalization effort (section 2.4) has been contributed by Mohammed Foughali, the rest of the material including the original idea, developments, proofs and implementation are my own.

Chapter 3 proposes an architecture mixing neural networks with graph topology to learn an error model for the monocular *Simultaneous Localization and Mapping* problem. This model has the particularity to provide with *relative* positional errors between all positions, instead of absolute ones. Illustrative and statistical results from simulations are discussed. This work has been motivated by the *Precidrone* project, which partly financed this thesis and consisted in studying innovative means to map crop fields with UAVs. Work related to replanning mapping trajectories in a similar context, to which I partially contributed, can be found in [Pěnička 2017].

The core material of this chapter has been submitted to the Robotics: Science and Systems conference [Reymann 2019b].

All work of this chapter is my own.

Bibliography

[Bailon-Ruiz 2017] R. Bailon-Ruiz, C. Reymann, S. Lacroix, G. Hattenberger, H. Garcia de Marina and F. Lamraoui. *System simulation of a fleet of drones to probe cumulus clouds.* In International Conference on Unmanned Aircraft Systems, Miami, United States, June 2017.

[Pěnička 2017] R. Pěnička, M. Saska, C. Reymann and S. Lacroix. *Reactive Dubins Traveling Salesman Problem for Replanning of Information Gathering by UAVs.* In The European Conference on Mobile Robotics, Palaiseau, France, September 2017.

[Renzaglia 2016] A. Renzaglia, C. Reymann and S. Lacroix. *Monitoring the Evolution of Clouds with UAVs.* In IEEE International Conference on Robotics and Automation, Stockholm, Sweden, May 2016.

[Reymann 2018] C. Reymann, A. Renzaglia, F. Lamraoui, M. Bronz and S. Lacroix. *Adaptive sampling of cumulus clouds with a fleet of UAVs.* Autonomous robots, vol. 42, no. 2, pages 1–22, 2018.

[Reymann 2019a] C. Reymann, M. Foughali and S. Lacroix. *On the need of distributed simulations.* In submitted to the 19th IEEE International Conference on Software Quality, Reliability and Security conference, Sofia (Bulgaria), July 2019.

[Reymann 2019b] C. Reymann and S. Lacroix. *Learning error models for graph SLAM.* In submitted to the Robotics: Science and Systems conference, Freiburg (Germany), June 2019.

Chapter 1

Adaptive sampling of cumulus clouds with UAVs

Contents

Adaptive sampling of cumulus clouds with UAVs

Alessandro RENZAGLIA[1], Fayçal LAMRAOUI[2], Murat BRONZ[3], and Simon LACROIX[1]

[1]LAAS-CNRS, INSA, Université de Toulouse, CNRS, Toulouse, France
[2]Météo-France/CNRS, CNRM/GAME, Toulouse, France
[3]ENAC, 7 avenue Edouard-Belin, F-31055 Toulouse, France

Published in *Autonomous Robots*, 2018, 42 (2), pp.491-512

Abstract

This paper presents an approach to guide a fleet of Unmanned Aerial Vehicles to actively gather data in low-altitude cumulus clouds with the aim of mapping atmospheric variables. Building on-line maps based on very sparse local measurements is the first challenge to overcome, for which an approach based on Gaussian Processes is proposed. A particular attention is given to the on-line hyperparameters optimization, since atmospheric phenomena are strongly dynamical processes. The obtained local map is then exploited by a trajectory planner based on a stochastic optimization algorithm. The goal is to generate feasible trajectories which exploit air flows to perform energy-efficient flights, while maximizing the information collected along the mission. The system is then tested in simulations carried out using realistic models of cumulus clouds and of the UAVs flight dynamics. Results on mapping achieved by multiple UAVs and an extensive analysis on the evolution of Gaussian Processes hyperparameters is proposed.

1.1 Introduction

Context. Atmospheric models still suffer from a gap between ground-based and satellite measurements. As a consequence, the impact of clouds remain one of the largest uncertainties in the climate General Circulation Model (GCM): for instance the diurnal cycle of continental convection in climate models predicts a maximum of precipitation at noon local time, which is hours earlier compared to observations – this discrepancy being related to insufficient entrainment in the cumulus parameterizations [Del Genio 2010]. Despite the continual efforts of cloud microphysics modelers to increase the complexity of cloud parameterization, uncertainties continue to persist in GCMs and numerical weather prediction [Stevens 2013]. To alleviate these uncertainties, adequate measurements of cloud dynamics and key micro-physical parameters are required. The precision of the instruments matters for this purpose, but it is the way in which samples are collected that has the most important impact. Fully characterizing the evolution over time of the various parameters (namely pressure, temperature, radiance, 3D wind, liquid water content and aerosols) within a cloud volume requires dense spatial sampling for durations of the order of one hour: a fleet of autonomous lightweight Unmanned Aerial Vehicles (UAVs) that coordinate themselves in real time could fulfill this purpose.

The objective of the SkyScanner project[1], which gathers atmosphere and drone scientists, is to conceive and develop a fleet of micro UAVs to better assess the formation and evolution of low-altitude continental cumulus clouds. The fleet should collect data *within and in the close vicinity* of the cloud, with a spatial and temporal resolution of respectively about 10 m and 1 Hz over the cloud lifespan. In particular, by reasoning in real time on the data gathered so far, an *adaptive data collection scheme* that detects areas where additional

[1] https://www.laas.fr/projects/skyscanner/

measures are required can be much more efficient than a predefined acquisition pattern: this article focuses on the definition of such adaptive acquisition strategies.

Challenges The overall control of the fleet to map the cloud must address the two following problems:

- It is a poorly informed problem. On the one hand the UAVs perceive the variables of interest only at the positions they reach (contrary to exteroceptive sensors used in robotics, all the atmosphere sensors perform pointwise measures at their position), and on the other hand these parameters evolve dynamically. The mapping problem in such conditions consists in estimating a 4D structure with a series data acquired along 1D manifolds. Furthermore, even though the coarse schema of air currents within cumulus clouds is known (Fig. 1.1), the definition of laws that relate the cloud dimensions, the inner wind speeds, and the spatial distribution of the various thermodynamic variables is still a matter of research – for which UAVs can bring significant insights.

- It is a highly constrained problem. The mission duration must be of the order of a cumulus lifespan, that is about 1 hour, and the winds considerably affect both the possible trajectories of the UAVs and their energy consumption – all the more since we are considering small sized motor gliders aircrafts (maximum take off weight of 2.0 kg). Since winds are the most important variables that influence the definition of the trajectories and are mapped as the fleet evolves, mapping the cloud is a specific instance of an "explore vs. exploit" problem.

Exploring cloud with a fleet of UAVs is therefore a particularly complex problem. The challenge to overcome is to develop *non-myopic adaptive strategies* using myopic sensors, that define UAV motions that maximize both the amount of gathered information and the mission duration.

Related work. Atmospheric scientists have been early users of UAVs[2], thanks to which significant scientific results have rapidly been obtained in various contexts: volcanic emissions analysis [Diaz 2010], polar research [Holland 2001, Inoue 2008] and naturally climatic and meteorological sciences [Ramanathan 2007, Corrigan 2008, Roberts 2008]. UAVs indeed bring forth several advantages over manned flight to probe atmospheric phenomena: low cost, ease of deployment, possibility to evolve in high turbulence [Elston 2011], etc. An in-depth overview of the various fixed-wing airframes, sensor suites and state estimation approaches that have been used so far in atmospheric science is provided in [Elston 2015].

[2]Cf the activities of the International Society for Atmospheric Research using Remotely piloted Aircraft – ISARRA, http://www.isarra.org

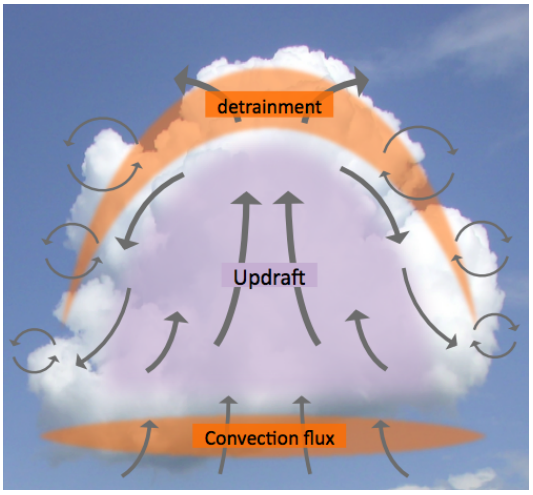


Figure 1.1: Schematic representation of a cumulus cloud. The arrows represent wind velocities, the orange blobs denote areas where mixing is occurring between the cloud and the surrounding atmosphere. This representation is very coarse: for instance the updrafts in the center of the cloud are known to behave as "bubbles" when the cloud is young. The cloud dimensions can vary from one to several hundreds of meters.

In these contexts, UAVs follow pre-planned trajectories to sample the atmosphere. In the robotics literature, some recent works have tackled the problem of *autonomously* exploring or exploiting atmospheric phenomena. The possibility of using dynamic soaring to extend the mission duration for sampling in supercell thunderstorms has been presented in [Elston 2014]. In this case, only the energetic consumption is optimized, and the gathered information does not drive the planning. In [Lawrance 2011a, Lawrance 2011b], Lawrance and Sukkarieh present an approach where a glider explores a wind field trying to exploit air flows to augment flight duration. The wind field is mapped using a Gaussian Process Regression framework (GPR), and the wind currents are simulated using combinations of sines and cosines and a toroidal model for updrafts, and a constant lateral drift is added to introduce dynamicity. The authors propose a hierarchic approach for the planning, where a target point is firstly selected and then a trajectory to reach it is generated for every planning cycle. In a similar scenario, a reinforcement learning algorithm to find a trade-off between energy harvesting and exploration is proposed in [Chung 2015]. Autonomous soaring has also been studied, as in [Nguyen 2013], where a glider has to search for a target on the ground. The goal here is to maximize the probability of detecting the target traveling between thermals with known location. The problem of tracking and mapping atmospheric phenomena with a UAV is also studied in [Ravela 2013]. The authors use GPR to map the updraft created by a smoke plume. Even though the mapped currents are not taken into account for the navigation, it is worth to remark that contrary to the previous contributions, here experiments with a real platform are presented. This shows the possibility of online mapping

of atmospheric phenomena by a fixed-wing UAV using GPR. An other significant contribution on wind-field mapping is presented in [Langelaan 2012]: aiming at autonomous dynamic soaring with a small UAV, the authors present and approach in which the wind field is modelled by polynomials, which parameters are estimated with a Kalman Filter. Experiments in which the mapped wind-field is compared to an "air-truth" obtained by tracking lighter than air balloons small are presented. Finally, autonomous exploration of current fields is not exclusively related to aerial applications: the use of Autonomous Underwater Vehicles for oceanographic studies has been recently investigated [Das 2013, Michini 2014].

Besides in [Michini 2014], in all the aforementioned work only the use of a single vehicle to achieve the mission is considered, and no multi-UAV systems are proposed.

Contributions and outline. The work presented in this article tackles the following problem: a fleet of a handful of UAVs is tasked to autonomously gather information in a specified area of the cloud. The UAVs trajectories are optimized using an on-line updated dense model of the variables of interest. The dense model is built on the basis of the gathered data with a Gaussian Processes Regression, and is exploited to generate trajectories that minimize the uncertainty on the required information, while steering the vehicles within the air flows to save energy. The results presented here significantly extend the preliminary work depicted in [Renzaglia 2016]: they use a realistic dynamic aircraft model, and extensive simulation in dynamic clouds models are analyzed. The main contributions of this work with respect to the state of the art are:

- The mapped phenomenon varies a lot in time and space, and the ability to build proper wind maps is essential, as it conditions the ability to derive optimal adaptive sampling schemes. The hyperparameters of the GP are hence learned online (section 1.2), and an analysis of their evolution is proposed (section 1.4).
- An original exploration technique based on a stochastic optimization scheme is proposed to plan feasible trajectories in the mapped current field (section 1.3). The approach aims at optimizing possibly contradictory goals: augmenting the information gathered so far, while minimizing energy consumption.
- Realistic simulations based on a cumulus cloud model produced by realistic large-eddy simulations and a realistic motor glider flight dynamics model are presented (section 1.4). Some exploration tasks are depicted, varying the criteria to optimize, and we show the ability of our approach to perform the specified mission in a realistic setting.

A discussion concludes the paper and proposes further research and development directions to explore, so as to effectively deploy adaptive fleets of drones within atmospheric phenomena.

1.2 Mapping Clouds

Maintaining a reliable map of the environment is of course of utmost importance for exploration tasks, as it is necessary to assess both the feasibility of trajectories and the relevant sampling locations. In the case of atmospheric phenomena, there are numerous variables of interest to the meteorologist. Of particular interest is the 3D wind vector: it is both one of the most dynamic atmospheric variables and an essential information for planning feasible and energy efficient paths. We therefore focused our work on the mapping of dynamic 3D wind currents.

Due to the sparsity of the sampling process in a dynamic 3D environment, the GPR probabilistic framework is particularly adapted for this mapping problem, as shown by related work [Lawrance 2011a, Lawrance 2011b, Ravela 2013, Das 2013]. Being statistical in nature, GPR allows to estimate the quality of predictions. This is naturally exploited in active perception tasks, such as in [Souza 2014], where the authors derives exploration strategies for outdoor vehicles, and in [Kim 2015], where a GP framework is used to drive surface reconstruction and occupancy mapping.

The mapping framework used in this paper is similar to the one presented in [Lawrance 2011b], where three independent GP models are used to map the components of the 3D wind vector, but we propose an online hyperparameter optimization, which is activated between each planning iteration. We have also focused efforts on deriving interesting and fast to compute information metrics from the model to drive the exploration strategies.

1.2.1 Gaussian process regression model

We introduce here briefly the usage of Gaussian processes for regression tasks. We refer the reader to the work of Rasmussen & Williams [Rasmussen 2006] for an in-depth view of the subject.
Gaussian Process Regression is a very general statistical framework, where an underlying process $y = f(x) : \mathbb{R}^n \to \mathbb{R}$ is modeled as "a collection of random variables, any finite number of which have a joint Gaussian distribution" [Rasmussen 2006]. One can view this as a way to set a Gaussian prior over the set of all admissible functions: given a location $x \in \mathbb{R}^n$, the values y taken by all admissible functions are distributed in a Gaussian manner. Under the gaussianity assumption, the process is fully defined by its mean and covariance:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})]\,, \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x})(f(\mathbf{x}') - m(\mathbf{x}'))]\,. \end{aligned} \tag{1.1}$$

In this model, the mean m and covariance k are not learned directly from the data, but given as parameters. In most cases, the process is assumed to have zero mean, so that the only parameter is the covariance function or *kernel*. We currently do not use any particular prior information and adopt a zero mean process: this is a matter of further work, that will explicitly define the relations between the higher level coarse cloud model and the dense GP-based model, as well as the relations

between the various variables of interest.

The kernel encodes the similarity of the target process f at a pair of given inputs and so describes the spatial correlations of the process. Given a set of n samples (X, Y) and assuming zero mean, the GP prior is fully defined by the $n \times n$ Gram matrix $\Sigma_{X,X} = [k(X_i, X_j)]$ of the covariances between all pairs of sample locations. Inference of the processes value $y_\star$ at a new location $x_\star$ is then done by conditioning the joint Gaussian prior distribution on the new samples:

$$\begin{aligned} \bar{y}_\star &= \Sigma_{\mathbf{x}_\star,\mathbf{X}} \Sigma_{X,X}^{-1} \mathbf{Y}, \\ \mathbb{V}[y_\star] &= k(\mathbf{x}_\star, \mathbf{x}_\star) - \Sigma_{\mathbf{x}_\star,\mathbf{X}} \Sigma_{X,X}^{-1} \Sigma_{\mathbf{x}_\star,\mathbf{X}}^\top \end{aligned} \tag{1.2}$$

The posterior Gaussian distribution at location $x_\star$ of the values of all admissible functions in the GP model therefore has mean $\bar{y}_\star$ and variance $\mathbb{V}[y_\star]$, which can be used both to predict the value of the function and to quantify the uncertainty of the model at this location.

Thanks to the gaussianity assumption, inference has a closed form solution involving only linear algebraic equations. Computing the model is done in $\mathcal{O}(n^3)$, due to the cost of inversion of the Σ matrix and subsequent computation of the posterior are done in $\mathcal{O}(n^2)$. This can be done online using optimized linear algebra software for models of up to a few hundreds of samples.

1.2.2 Learning hyperparameters

The choice of the expression of the kernel function k is central: it sets a prior on the properties of f such as its isotropy, stationarity or smoothness. The only requirement for the kernel function is that it has to be *positive semidefinite*, which means the covariance matrix Σ of any set of inputs must be positive semidefinite and therefore invertible. In practice, one selects a family of kernels, whose so called *hyperparameters* are learned to fit the data. We selected the most widely used squared exponential kernel with additive Gaussian noise:

$$k_{SE}(\mathbf{x_i}, \mathbf{x_j}) = \sigma_f^2 e^{-\frac{1}{2}|\mathbf{x_i}-\mathbf{x_j}|M|\mathbf{x_i}-\mathbf{x_j}|} + \delta_{ij}\sigma_n^2 \tag{1.3}$$

where δ_{ij} is the Kronecker delta function, $M = \mathbf{l}^{-2}I$ is a diagonal matrix that defines the characteristic anisotropic length scales $\mathbf{l}$ of the process, and σ_f^2 and σ_n^2 are respectively the process variance and the Gaussian noise variance over the measures. The squared exponential kernel is stationary, anisotropic and infinitely smooth.

The kernels hyperparameters $\theta = (\sigma_f, \mathbf{l}, \sigma_n)$ are chosen by maximizing the Bayesian log marginal likelihood criterion:

$$\log p(\mathbf{Y}|\mathbf{X}, \theta) = -\frac{1}{2}\mathbf{Y}^\top \Sigma^{-1}\mathbf{Y} - \frac{1}{2}\log|\Sigma| - \frac{n}{2}\log 2\pi \tag{1.4}$$

This is a non-convex optimization problem: the optimization function may be subject to local maxima, and therefore may not always converge in finite time.

Optimizing the kernel hyperparameters is computationally demanding: although the partial derivative of eq. (1.4) with respect to θ can be computed in $\mathcal{O}(n^3)$, convergence to a local optimum may involve a great number of steps. Usually the optimization of hyperparameters is therefore done offline.

In our context, the input space is the four dimensional space-time location of the UAVs, and the estimated variables are the three components of the 3D wind vector. Therefore we train three GP models separately, making the simplifying assumption that there is no correlation between the three components. The optimization of hyperparameters is done online: indeed the underlying atmospheric process's length scales may vary from one cloud to the other, and the stationarity assumption may not hold during the course of the mission. To alleviate computational issues we keep only the most relevant samples. This is done by setting a tolerance value h_{tol}, so that when comparing the newest sample x_n to an older one x_o, and setting all spatial coordinates to zero, the older one is discarded if $k(x_n, x_o) < h_{tol}$. Using this criteria on the covariance instead of the age of the samples lets the amount of retained samples adapt to the the temporal length scale of the process. In order to avoid dropping all data if the optimization produces very short length scale on the time dimension, we also specify a minimum amount of time it has to stay in the model, which was set to one minute in our experiments.

Figure 1.2 illustrates the mapping of the wind vertical component, by virtually gathering wind data in a realistically simulated wind field (see section 1.4.1.1).

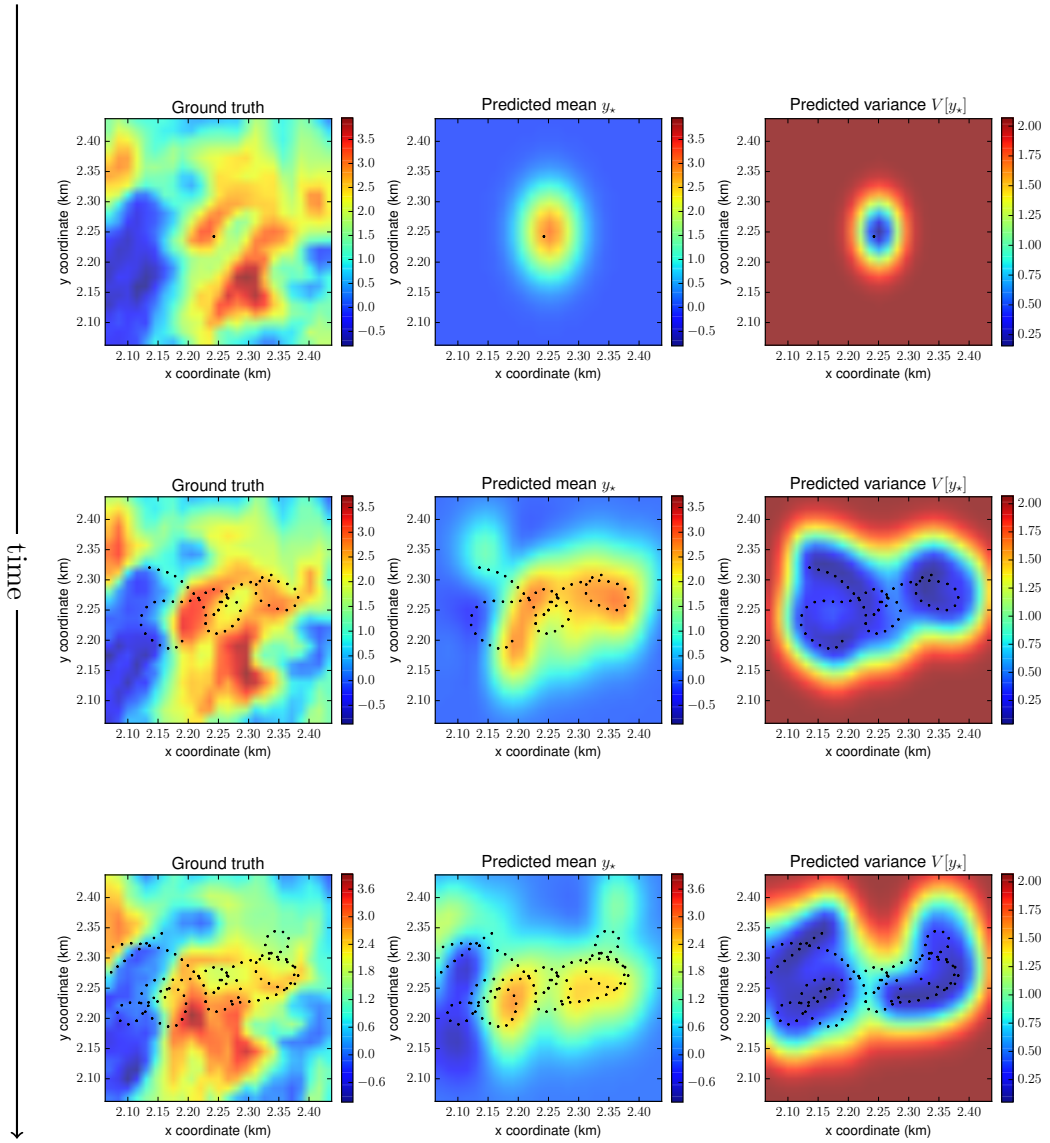


Figure 1.2: Illustration of the GP-based mapping process applied in a realistic wind field. The left pictures represent the ground truth of the vertical wind velocity, pictures in the middle show the computed maps on the basis of the measures taken at the positions denoted by a black dot (the measure standard deviation σ_n is equal to 0.25 m), and the right images show the predicted variances of the map. Unit of all depicted values is $\mathrm{m.s}^{-1}$. The sequences of measures are defined by the planning approach presented in section 1.3. From top to bottom, a time lapse of 20 s separates each line of pictures, the altitude shown is the one of the first UAV, respectively 800 m, 825 m and 850 m.

1.2.3 Computing information metrics on trajectories

As the task at hand is an exploration one, it is crucial to be able to properly evaluate the quantity of information a set of new samples add to the current map, so as to enable a planning process to select informative trajectories. Since the GP framework encodes probability distributions, the resulting maps are particularly well adapted for this purpose: the variance of the GP as defined by eq. (1.2), which represents the uncertainty of the current model at each point in space, is a natural candidate to evaluate the utility of new measurements.

The problem of selecting the best possible future measurements to estimate a statistical model has been extensively studied. The idea is to minimize the variance of the estimator using a statistical criterion. In [Chung 2015], Chung *et al.* integrate the variance of the GP over the region to be mapped and derive a measure of the quality of the model. Unfortunately, a closed-from expression of this integral does not exist in general. The integration criterion defined in [Chung 2015] is an instance of I-optimality. Other classical criteria directly seek to minimize the covariance matrix:

- D-optimality aims to maximize the differential Shannon entropy of the statistical model, which comes to maximizing the determinant of the information matrix (the inverse of the covariance matrix).
- T-optimality aims to maximize the trace of the information matrix.

We have found very few differences between these two criteria in previous work [Renzaglia 2016], so we used the T-optimality criterion in the experiments, which it is slightly faster to compute.

To efficiently evaluate the information gain of a new set of measurement points, we define the conditional covariance $\Sigma_{\mathbf{X}_{new}|\mathbf{X}}$ of the set of m new points $\mathbf{X}_{new}$ against $\mathbf{X}$, the points already included in the regression model:

$$\Sigma_{\mathbf{X}_{new}|\mathbf{X}} = \Sigma_{\mathbf{X}_{new},\mathbf{X}_{new}} - \Sigma_{\mathbf{X}_{new},\mathbf{X}}\Sigma^{-1}_{\mathbf{X},\mathbf{X}}\Sigma^{\top}_{\mathbf{X}_{new},\mathbf{X}} \tag{1.5}$$

The $\Sigma_{\mathbf{X}_{new}|\mathbf{X}}$ matrix is of fixed size $m \times m$, independent of the size of the model, which yield swift computations. The matrix itself is computed in $\mathcal{O}(nm^2 + mn^2)$, subsequent inversion or computation of the determinant are performed in $\mathcal{O}(m^3)$. The value v_T of the T-optimality criterion is thus defined as:

$$v_T(\Sigma_{\mathbf{X}_{new}|\mathbf{X}}) = tr([\Sigma_{\mathbf{X}_{new}|\mathbf{X}} + \sigma_n^2 I]^{-1}) \tag{1.6}$$

This criterion does not yield absolute values, not even positive ones. The scale will depend on the kernel function and on the number of samples in the model. Therefore it is only useful for comparing trajectories generated using the same model (and with the same amount of new samples). To be able to integrate this

information measure in a multi-criteria optimization framework, it is necessary to normalize it. We introduce here an empirical way of normalizing the information measure.

Assuming a fixed sampling rate, the *feasible* trajectory maximizing the information measure in a completely empty model is a straight line in the spatial direction where the covariances length scale is the shortest. Computing the utility measure v_{Tb} for such a trajectory gives an upper bound for a *best* set of samples. Driving the UAV along a straight line in the direction of the longest length scale would still provide a *passable* utility v_{Tp} (as the model is still empty). Setting absolute utilities for v_{Tb} and v_{Tp} then enables to define a normalization for the information measure. This normalized value is not relative anymore, it takes into account the current state of the model: if the model is very dense, with low variances everywhere, then the normalized utility will be very low (depending on the normalization function) because we compare it to an ideal empty model. Note that these ideal trajectories must be feasible for the UAV, at least in a windless environment. As we use fixed wing UAVs, it is not realistic to drive them along a strictly vertical trajectory. Therefore when the shortest length scale is the vertical one, we assume a trajectory at maximum climb rate, with the horizontal component along the second shortest length scale.

1.3 Energy-efficient Data Gathering

The regression model presented in the previous section is the basis on which the energy-efficient data gathering strategy is developed. The local map built with the GPs is indeed the source of two fundamental information to plan the trajectories: generate feasible trajectories, and predicting the information gain their execution will bring. The optimization problem to solve can be then formulated as follows: generating safe and feasible trajectories which minimize the total energy consumption according to the mapped wind field, while maximizing the information collected along the paths.

Planning in currents fields is a challenging problem even in standard *start-to-goal* problems, where a robot moves in a static two-dimensional flow and assuming a perfect knowledge of the map [Petres 2007], [Soulignac 2011]. Our scenario is deeply different, since the field is changing over time, is initially unknown and sensed during the mission and it is not possible to identify an optimal final goal to reach in order to reduce the problem complexity. Furthermore, even though we do not consider large swarm of UAVs, the deployement of a small number of aircrafts (typically around 3-4) is crucial for the success of the mission, leading to larger planning spaces. All these complex issues, combined with strong computational constraints imposed by the requirements of on-line planning, make this multi-criteria optimization problem particularly challenging. As a result, obtaining a global optimal solution is not feasible and we limit our convergence requirements to local optimal solutions.

For the trajectory generation and evaluation, we consider short-time horizons

(typically in the order of ~ 20 seconds). This choice is motivated by two main reasons: firstly, the reliability of our local models significantly decreases in time, making unrealistic any long-term prediction; secondly, the computational constraints would be harder to respect with larger optimization spaces. Each planning horizon ΔT is then divided in m sub-intervals of duration dt in which the optimization variables (controls) are constant. As a result, the trajectory for the UAV j during ΔT is described by the sequence $u_i^{(j)}$, with $i \in \{1, ..., m\}$, and a given initial condition $u_0^{(j)}$.

1.3.1 Trajectory evaluation

The first criterion to evaluate the trajectories is the energy consumption. Flying within currents leads indeed to energy costs which strongly depend on the planning: flying against strong wind can hugely increase the amount of required energy, while optimally exploiting these currents (and especially ascending air flows) can on the other hand allow the UAVs to significantly extend their flight duration. To take into account this phenomenon in the trajectory evaluation, we explicitly consider the total energy consumed by the fleet over a planning horizon ΔT. This value is simply represented by the sum over time of the input power $P_{in}(t)$, which is one of the controls on which the trajectories are optimized. Introducing normalization terms, for the aircraft j, we have:

$$U_E^{(j)}(t_0, \Delta T) = 1 - \frac{1}{P_{in}^{max} \Delta T} \sum_{t=t_0}^{t_0+\Delta T} P_{in}^{(j)}(t) dt \,. \tag{1.7}$$

The total fleet energetic utility U_E is given by average this value over the UAVs. Note that this criterion is strictly local and does not take into account the total amount of energy stored in the batteries: this is rather a concern for the higher-level decision process, that must for instance make sure every UAV can come back the ground station. It is also independent of the trajectory of the UAVs. Indeed exploitation of the wind field is an indirect result of the combination of minimization of the energetic expense with other goals. For example when trying to reach a higher altitude, trajectories that traverse updrafts will need less energy for the same climb rate and therefore will be preferred over trajectories outside the updraft.

The second criterion for the trajectory evaluation is the information gain U_I. To predict the information utility acquired by a given set of trajectories we use the T-information metric: after sampling the planned trajectories at at fixed sampling rate, we compute the relative utility of the X_{new} samples $v_T(\Sigma_{\mathbf{X}_{new}|\mathbf{X}})$ using (1.6). A linear normalization is obtained using $U_I(v_{Tb}) = 1$ and $U_I(v_{Tp}) = 0.5$, subsequently clipping values above one and below zero:

$$U_I(v) = max\left(0,\ min\left(1,\ \frac{v + v_{Tb} - 2v_{Tp}}{2(v_{Tb} - v_{Tp})}\right)\right) . \tag{1.8}$$

As discussed in section 1.2.3, v_{Tb} and v_{Tp} are respectively the best and worst expected information gains for an ideal rectilinear trajectory that is not influenced by winds, on an empty model with current hyperparameters. This way we are able to compute an absolute measure for the information gain, that takes into account both the current samples in the model and the hyperparameters. As the model fills up with samples, the information gain lowers in already visited areas. The choice of $U_I(v_{Tp})$ influences what is considered a good sampling: setting it to a low value degrades the utility of sampling along dimensions with a longer length scale, whereas setting it to a value close to one would not favor any particular sampling direction. The values $U_I(v_{Tb}) = 1$ and $U_I(v_{Tp}) = 0.5$ have been empirically chosen.

The third considered criterion, U_G, is strictly dependent on the specific goal of each mission. The acquisition of information within a given area is one of the essential task issued by the higher planning level. Formally, defining a rectangular box b, the utility of a given trajectory for this task is defined as:

$$U_G^j(t_0, \Delta T) = \frac{d_b(X^j_{t_0+\Delta T}) - d_b(X^j_{t_0})}{V_{zmax}\Delta T}\,, \tag{1.9}$$

where X_t^j is the position of the j-th UAV at time t and $d_b(X)$ is the distance between the UAV and the closest point of the box boundary ($d_b(X) = 0$ if the UAV is inside the box). The total utility for the fleet is the mean value over all UAVs.

To tackle this centralized multi-criteria optimization problem we consider a linear combination of the three criteria:

$$U_{tot} = w_E U_E + w_I U_I + w_G U_G\,. \tag{1.10}$$

In Section 1.4.2 we analyze in details the effects on the mission of different choices of the weights w_x. In future work it is our intention to explore also different methods to tackle this multi-criteria optimization problems, e.g using Multi Criteria Decision Making approaches [Basilico 2011].

1.3.2 Trajectory optimization

For every ΔT, we can now formulate the trajectory optimization problem, which consists in maximizing a global utility function $U_{tot}(\boldsymbol{u})$ as a function of the control variables $\boldsymbol{u}$, subject to some constraints:

$$|u_i^{(j)} - u_{i-1}^{(j)}| \leq \Delta u_{max} \quad \forall\, i, j\,. \tag{1.11}$$

As defined in details in Appendix 1.A, our controls inputs are the turn radius R and the motor power input P_{in}. To tackle this optimization problem, we propose a centralized two-step approach[3]: a first phase based on a blind random search in order to have a good trajectories initialization, followed by a gradient ascent algorithm to optimize them.

[3]Section 1.5 discusses this centralization issue

1.3.2.1 Trajectory initialization

The first phase of the optimization process, based on a blind random search, is achieved creating a set of feasible trajectories obtained by a constrained random sampling of controls u_t, and exploiting the approximated field generated by the GP regression. The trajectories are then evaluated using the utility function U_{tot} and the best set of N_r trajectories is the initial configuration for the gradient ascent phase. The presence of the first sampling step is due to the strong dependence of the gradient-based solution on the initial configuration. In this way, even though we only have local convergence guarantees, the probability of getting stuck in local maxima far from the global optimal trajectories is reduced.

1.3.2.2 Stochastic gradient approximation

To perform the gradient ascent we adopt a constrained version of the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm [Spall 2005], [Sadegh 1997]. This algorithm is based on successive evaluations of the utility function to obtain a numerical approximation of the gradient. At every algorithm iteration k, the optimization variables $\boldsymbol{u}$ are hence updated as follows:

$$\boldsymbol{u}_{k+1} = \Pi(\boldsymbol{u}_k + a_k \hat{g}(\boldsymbol{u}_k)) \,, \tag{1.12}$$

where Π is a projection operator to force $\boldsymbol{u}$ to stay in the feasible space, and $\hat{g}$ is the gradient approximation, for which we used the two-sided version:

$$\hat{g}_k(\boldsymbol{u}_k) = \begin{bmatrix} \frac{U(\boldsymbol{u}_k + c_k\Delta_k) - U(\boldsymbol{u}_k - c_k\Delta_k)}{2c_k\Delta_{k1}} \\ \vdots \\ \frac{U(\boldsymbol{u}_k + c_k\Delta_k) - U(\boldsymbol{u}_k - c_k\Delta_k)}{2c_k\Delta_{kN}} \end{bmatrix} , \tag{1.13}$$

where Δ is a random vector. Note that, due to the simultaneous perturbation of all the optimization variables, every iteration requires only two evaluations of U, regardless of the optimization space dimension. This is in contrast with other popular stochastic gradient approximation algorithms, such as the Finite Difference Stochastic Approximation (FDSA), which require $2p$ evaluations, where p is the dimension of the vector $\boldsymbol{u}$. At the same time, under reasonable general conditions, these algorithms achieve the same level of statistical accuracy for a given number of iterations [Spall 2005]. This point may be crucial for real-time applications and when the optimization function evaluation is time consuming, as in our case. To ensure the convergence of the algorithm, a simple and popular distribution for the random perturbation vector Δ_k is the symmetric Bernoulli ± 1 distribution, and the

conditions on the gain sequences a_k, c_k are:

$$a_k > 0,\ c_k > 0,\ a_k \to 0,\ c_k \to 0,$$
$$\sum_{k=0}^{\infty} a_k = \infty, \quad \sum_{k=0}^{\infty} \frac{a_k^2}{c_k^2} < \infty\,. \tag{1.14}$$

A standard choice which satisfies the previous conditions is:

$$a_k = \frac{a}{(k+1)^{\alpha}} \quad c_k = \frac{c}{(k+1)^{\gamma}}\,. \tag{1.15}$$

Practically effective and theoretically valid values for the decay ratings α and γ are 0.602 and 0.101 [Spall 1998]. The coefficients a and c are more dependent on the particular problem and their choice significantly affects the optimization result. The parameter c represents the initial magnitude of the perturbations on the optimization variables and, for every variable, we fixed it at $\sim 5\%$ of its maximum range. Lower values would increase the number of required iterations to converge, while higher values would result in instability of the algorithm since the perturbation would not be local anymore. The coefficient a is instead chosen as a function of the desired variation in these variables after the update at early iterations, which can be set at same order of the perturbations. To do this reliably, we use few initial iterations to have a good estimation of the gradient $\hat{g}_k(\boldsymbol{u}_0)$ and we then exploit eq. (1.12) to fix a.

1.3.3 Illustrative examples

Figure 1.3 shows the different phases of the trajectory generation for one UAV in one planning horizon within a realistic wind field. An xy projection of the trajectories is shown, including the vertical wind prediction at the starting altitude and time. The random sampled trajectories are shown in black, with the best one in blue. The SPSA algorithm then locally optimizes the best trajectory, resulting here in a loop more tightly closed around the center of the updraft (in green). The open-loop executed trajectory is shown in red, following closely the planned trajectory. Only a portion of the optimal trajectory is actually executed before the next planning phase. The utility function in this case is given by U_{tot}, with no information term. Its maximization as a function of SPSA iterations is shown in Fig. 1.4.

Figure 1.5 shows some trajectories obtained after a few iterations of the trajectory planning process, in an illustrative two-dimensional case where fictitious current fields and utility maps have been defined so as to ease results visualization and understanding. Here the utility is defined as a scalar map and the optimization function is given by the sum of the utility collected along the trajectory. The figures show the results for both a single and a multi-UAV case. It is clear how the algorithm forces the UAVs to spread to avoid visiting the same locations in the three-UAVs case. When possible, currents are also exploited in order to visit more locations, and so collect more utility, in the same amount of time.

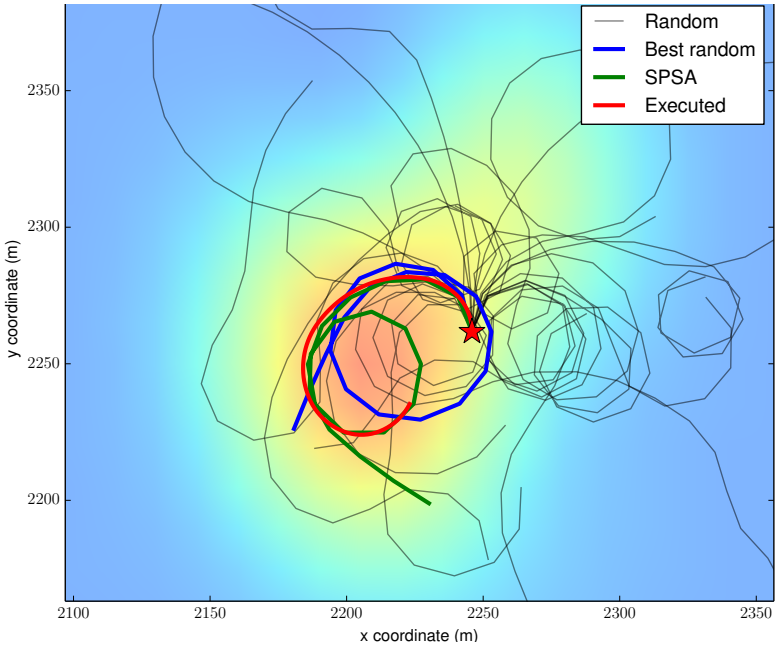


Figure 1.3: Illustration of the trajectory generation process in a realistic wind field, for a planning horizon $\Delta T = 20$ seconds. Projections on the xy plane for the random sampling initialization, optimized trajectory and final trajectory executed by the UAV are shown. The red star represents the initial position and the map colors shows the vertical component of the wind – with no particular units, the redder being the highest.

1.4 Integrated Simulations

The final objective of the SkyScanner project to experiment the flight of a fleet of drones within actual cumulus clouds is yet to be achieved. Beforehand, intensive simulations are required to assess the validity of the proposed solutions. This section depicts a first integrated simulation architecture, which aims at validating the mapping and planning algorithms. Various results are depicted, focusing in particular on the mapping performance and the learning of the GPR hyperparameters.

1.4.1 Simulation setup

1.4.1.1 Cloud model

To validate the mapping and planning algorithms, realistic cloud simulations are required: this is provided by atmospheric models, that can simulate the microphysical, dynamical, optical and radiative properties of clouds.

The atmospheric model used for the current study is Meso-NH [Lafore 1998]. This model is the result of the joint collaboration between the national center of meteorological research (CNRM, Météo-France) and Laboratoire d'Aéorologie (LA,

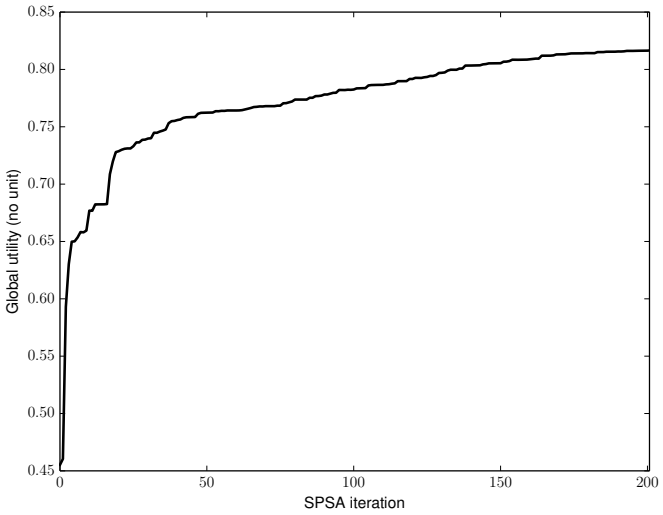


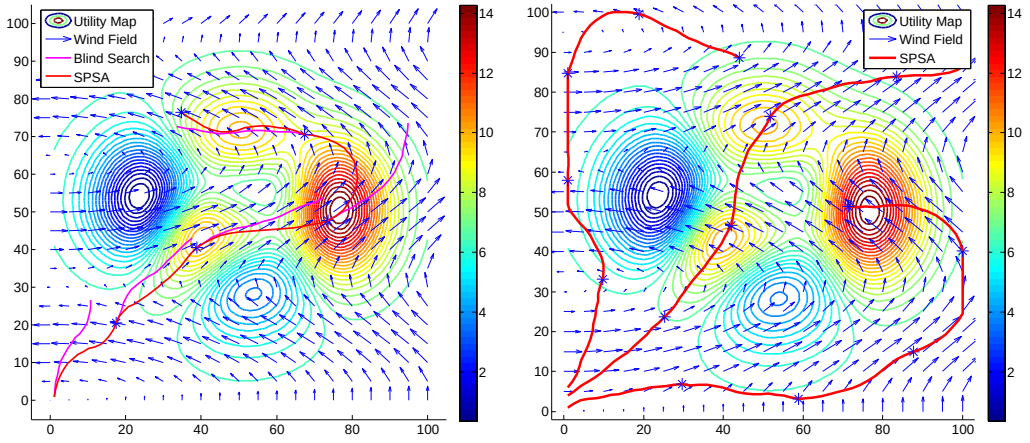
Figure 1.4: Behavior of U_{tot} over a planning horizon ΔT as a function of SPSA iterations.

UPS/CNRS). Meso-NH is a Non-Hydrostatic Model with the flexibility to simulate atmospheric phenomena at a wide range of resolutions that extends from one meter up to tens of kilometers. For this work, non-precipitating shallow cumulus clouds over land are simulated with the LES (Large-eddy simulations) version of Meso-NH, with resolutions down to ten meters. The simulation was driven by realistic initial conditions obtained on June 21, 1997 from meteorological measurements at the Southern Great Plains site in Oklahoma, U.S.A [Brown 2002]. This site is the first field measurement site established by the Atmospheric Radiation Measurement (ARM) Program.

To capture more details about clouds and their surroundings, it is preferable to set the atmospheric model at its highest resolution. The considered simulation domain is a cube of 400x400x161 grid points representing a volume of 4 km × 4 km × 4 km with horizontal resolutions of $dx = dy = 10$ m, vertical resolutions from $dz = 10$ m to 100 m and a time-step of 0.2 s. This setup is a compromise between the desired high resolutions and a reasonable simulation computation time[4].

The 161 vertical levels have a high resolution of 10 m in both convective cloud and surface layers; in the upper cloud-free troposphere, the domain has stretched resolutions from 10 m up to 100 m. The upper five layers of the simulation domain act as a sponge layer to prevent wave reflection. In addition, the horizontal boundary conditions are cyclic with a periodicity equal to the horizontal width of the simulation domain. The simulation estimates the following atmospheric variables: cloud liquid water content, water vapor, pressure, temperature, and the three components of wind. Figure 1.6 illustrates the 3D cloud water content of convective cumulus clouds at a given time. The overall simulation covers a time period of

[4]Days of computing on a large cluster are required to produce such simulations.



(a) (b)

Figure 1.5: a) one UAV is moving in a 2D environment where a scalar utility map and a wind field are defined. The trajectories initialized by a blind search at every planning-horizon ΔT are shown in magenta, and the final trajectories provided by the SPSA algorithm are in red. b) 3 UAVs are steered in the same environment to maximize the total utility (only the final trajectories are shown).

15 hours, but variables of interest have been saved every second only during one hour that corresponds to the maximum of surface fluxes.

1.4.1.2 UAV control

For the trajectory planning we choose a simplified aircraft model which, whilst being computationally light, captures the essential characteristics of the flight dynamics necessary to simulate realistic trajectories. The considered UAV is a Mako aircraft, 1.3 m wingpsan tail-less fixed wing airframe (figure 1.7), which model is depicted in Appendix 1.A. The realism of the model relies on two hypotheses. The first one is that the UAVs evolve in wind fields of moderate strength and turbulence, such as in the fair weather conditions in which cumulus clouds form. The second hypothesis is that the trajectories of the UAVs are not overly dynamic, which is consistent with the general design of the UAVs sent by meteorologist for such missions and the selected UAV for the project: dynamic maneuvers indeed degrade the measurements quality, and are not energy efficient.

We assume a constant total airspeed V: the fixed pitch propeller that is used on the aircraft yields a small flight envelope where it works efficiently, and the Paparazzi controls are designed for a fixed airspeed. Trajectories consists of a series of command pairs (R, P_{in}), which correspond to time slices during which the commands are constant. Their computation is depicted in Appendix 1.B: R defines the turn radius and direction of the UAV, and P_{in} the power drawn by the motor from the battery. As V is kept constant, all other parameters are bound. To further

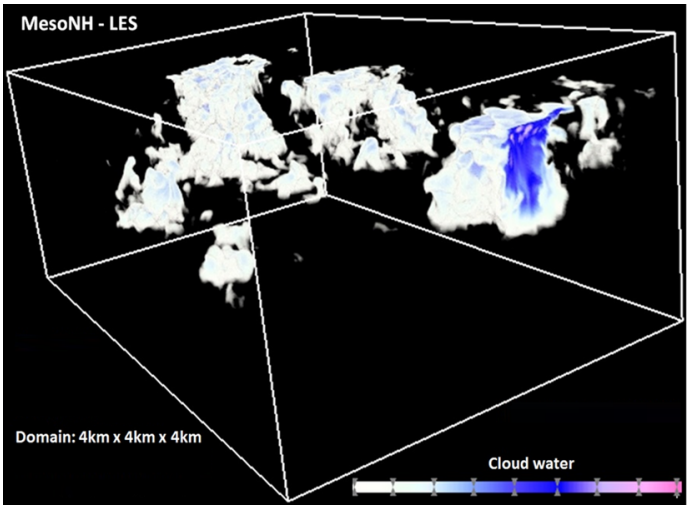


Figure 1.6: Meso-NH LES simulation: liquid cloud water content of the cumulus formed at 1h30 PM (ARM Southern Great Plains, June 21, 1997 conditions)

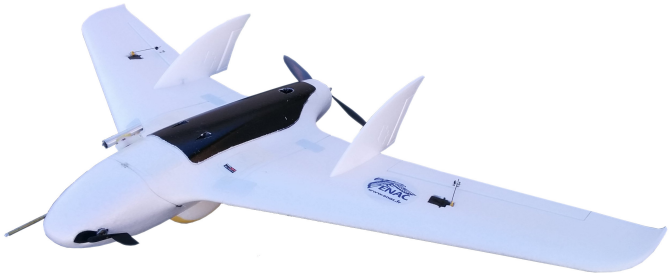


Figure 1.7: Mako aircraft used as a model for the simulations.

simplify the dynamics, changes in turn radius and direction between planned steps are assumed to happen instantaneously. Changes in climb rate as a result of a change of propulsive power P_{in} are linear. The key parameters and coefficients are estimated from the analysis of the Mako aircraft selected for field experiments in the SkyScanner project.

1.4.1.3 Simulation architecture

We tested our planning and mapping framework using a fairly simple simulation architecture, depicted Figure 1.8. The planning algorithm optimizes the joint trajectory of all UAVs using the predictions of the GP mapping framework. The resulting control sequences are then sent to the UAVs, which execute them with the dynamic model used to plan the trajectories in open-loop: no trajectory tracking is applied, but the wind ground truth influences the UAV actual motions, which then

differ slightly from the planned motions[5].

A wind sampling process is simulated by adding zero mean and fixed variance Gaussian noise on the wind ground truth. This constant noise model is a simplification of the actual errors made by processes that estimate the wind on board micro UAVs (in [Langelaan 2011, Condomines 2015] the errors are indeed Gaussian, but depend of the airspeed – yet in our simulations the aispeed is kept constant).

These wind samples are then fed to the three GP models, each modeling one component of the 3D wind vector. The GP hyperparameter optimization step is performed before each planning iteration.

The whole simulation loop is not real-time, as all steps happen in sequence and no particular attention has been paid to optimize the computing time. In fact, whilst the planning algorithm runs faster than real-time, the hyperparameter optimization step can last up to a few minutes. This costly step has not been finely tuned, and its implementation has not been optimized – all computations were performed using only one core on a i7 3.60GHz CPU. The simulation framework is implemented in Python, with the exception of the UAV model and the GP mapping framework which are implemented in Cython and C++ for speed purposes. The GP hyperparameter optimization is performed using the basin-hopping algorithm implementation of the SciPy package, with the 'L-BFGS-B' algorithm for local bound constrained optimization. Ten local optimization steps are achieved each time. Bounding the hyperparameters allows avoiding completely incoherent solutions, particularly in the beginning of the simulations, and quickens the convergence.

1.4.1.4 Scenarios

We conducted three sets of simulations corresponding to three different optimization scenarios:

$$\begin{aligned} \text{All: } U_{tot} &= \frac{1}{3}U_E + \frac{1}{3}U_I + \frac{1}{3}U_G \\ \text{No information: } U_{tot} &= \frac{1}{2}U_E + \frac{1}{2}U_G \\ \text{No energy: } U_{tot} &= \frac{1}{2}U_I + \frac{1}{2}U_G \end{aligned}$$

These scenarios have been chosen to show the possibilities and versatility of the proposed framework, even with simple linear combinations of criteria. For each scenario, a set of 80 simulations were run, with all other parameters remaining the same. The task to achieve is the exploration of an area defined as a box, that spans 400 m in the x and y axes, and which is 20 m thick along the z axis and centered at an altitude $z = 1.0$ km. Three identical UAVs start the mission from a unique position located under the center of the box, at an altitude of 800 m.

[5]The implementation of a trajectory tracker in the Paparazzi autopilot is under way within the SkyScanner project

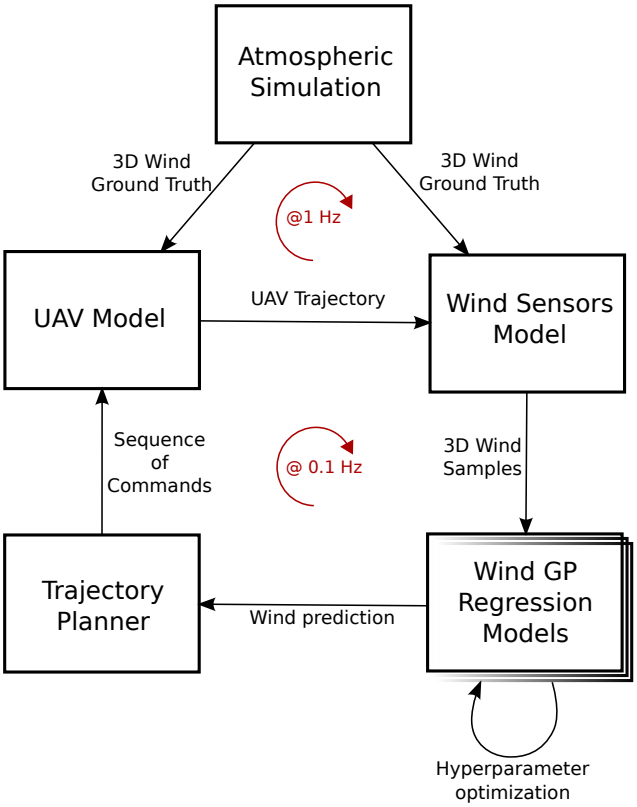


Figure 1.8: Simulation architecture.

The total simulation duration is five minutes. Although all simulations share the same starting point for the UAVs, the initial direction is chosen at random. This, coupled to the fact that the initial map starts empty, results in completely different trajectories and maps after a few dozen of seconds of simulation. Also, we picked a single cloud in the weather simulation to carry our experiments, but the whole weather simulation shares the same properties, and clouds of similar size are very similar. To generate cloud with different properties, one would have to run the weather simulations again with other initial conditions and models, which was not within our reach.

We choose a planning horizon of $\Delta T = 20\,\mathrm{s}$ with a $dt = 1\,\mathrm{s}$ resolution, but a re-planning is done each 10 s, so that only the first half of each planned trajectory is executed. The UAVs airspeed V is set to $15\,\mathrm{m.s}^{-1}$. Planning is done by sampling 200 random perturbations, then performing 400 SPSA algorithm steps. The mapping algorithm tolerance on the time dimension is set to 0.1 or 60 s, whichever the longest (see section 1.2.2). Spatio-temporal length scale hyperparameters $\mathbf{l} = (l_x, l_y, l_z, l_t)$ are bounded between 1 and $e^5 \approx 150\,\mathrm{m}$ (respectively seconds) and the σ_f and σ_n parameters are bounded between e^{-10} and $\mathrm{e}^{10}\,\mathrm{m}$ (values are expressed as exponentials because GP library optimizes the logarithm of the hyperparameters). In the

absence of a precise prior on the σ_f and σ_n parameters, we chose an interval wide enough to let the hyperparameters optimization step scale them as needed.

1.4.2 Results

Figure 1.9 shows typical trajectories for the three UAVs up to 150 s of simulation, in the 'No Information' scenario that favors the reduction of energetic expense. A first interesting result is that the planning algorithm allows the UAVs to benefit from updrafts by circling in high vertical wind regions, thus climbing with a lesser energy expense up to the target altitude. The UAVs then manage to stay around the desired altitude: the trajectories are less characteristic here, as the information gathering utility is disabled. The planned trajectories seem quite natural, avoiding unnecessary changes in heading.

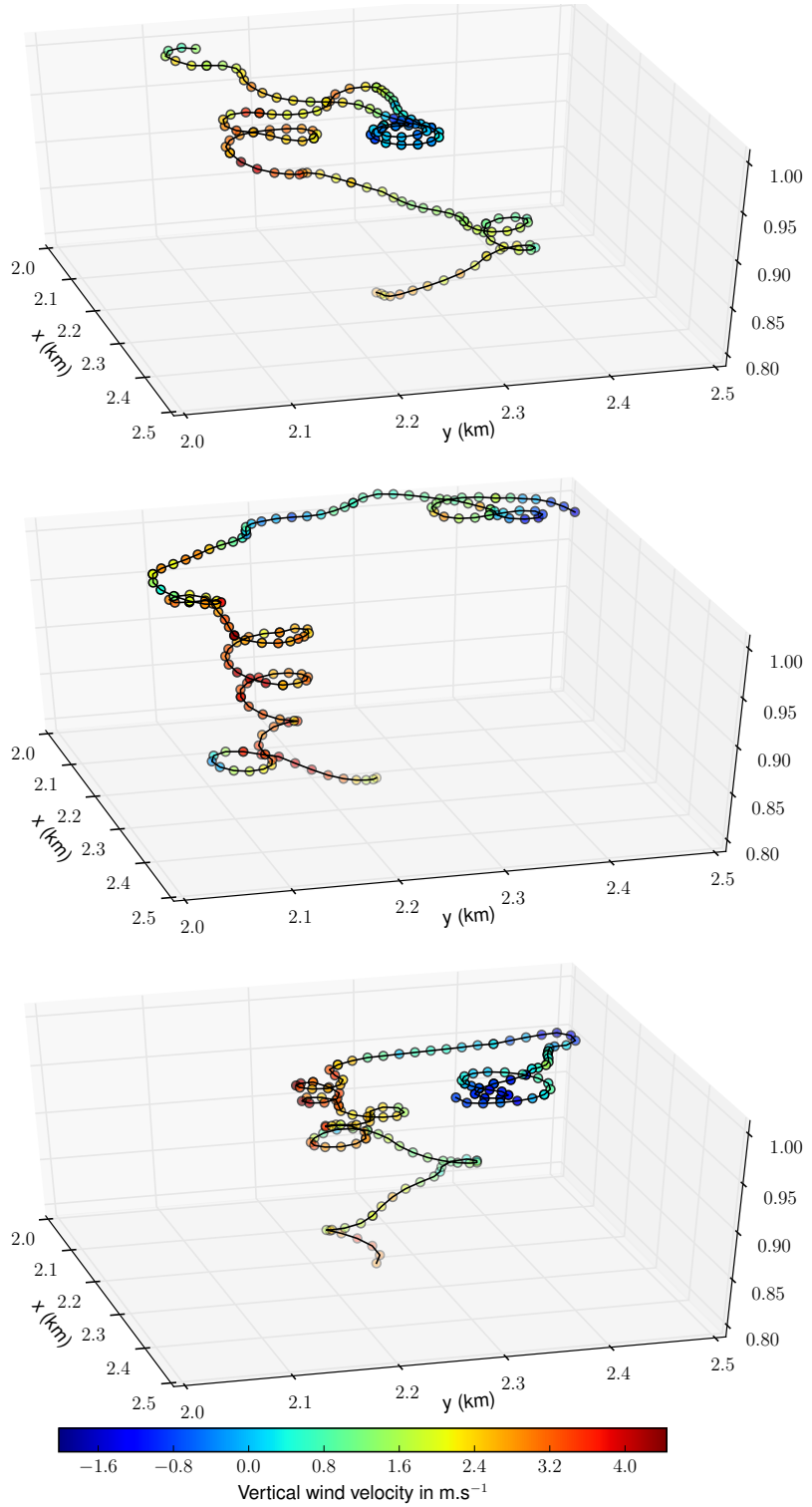


Figure 1.9: Examples of 3D trajectories of the three UAVs from $t = 0\,\mathrm{s}$ to $t = 150\,\mathrm{s}$ in the 'No information' scenario. The vertical wind component ground truth is shown in colors. All UAVs start on the bottom at altitude 0.8 km at the beginning of the simulation.

1.4.2.1 Performance of the wind prediction for trajectory planning

We analyze results for the vertical component of the wind, as it is both the most relevant information for the planning algorithm and the one with the largest amplitude in the Meso-NH simulation. Over all the simulated flights, the sampled values span from $-2.8\,\mathrm{m.s^{-1}}$ to $5.0\,\mathrm{m.s^{-1}}$. Figure 1.10 shows the RMSE of the wind prediction during the planning iterations as a function of time, computed along all the executed trajectories. After an initial stabilization period of about five iterations (50 seconds), the mean error converges towards $0.5\,\mathrm{m.s^{-1}}$ (6% relative error), which is quite a good estimate considering the sensors additive Gaussian noise of standard deviation $0.25\,\mathrm{m.s^{-1}}$ (3% relative error). It is also interesting to note that the scenario does not impact significantly the mean error of the GP regression, when considering the mean value over the trajectory. The prediction of the error is also quite good, the standard deviation estimated by the GPP follows a similar pattern as the RMSE (Fig. 1.11).

We can further investigate the quality of the prediction by looking at the standard normal deviate, obtained by dividing the (signed) prediction error by the predicted standard deviation (Fig. 1.12). Here again there is not much difference between the three scenarios: the standard deviation remains approximately constant at $\approx 1.4\,\sigma$, meaning the model has a slight tendency to under-estimate the variance (e.g. the error). In contrast, the mean decreases constantly: at the beginning of the simulations the GP model has a tendency to under-estimate the wind and at the end to slightly over-estimate it. If the first behavior is expected whilst the hyperparameters are being learned with only a few samples, the over-estimation of the wind at the end of the simulations is not expected. This bias may be due to the two following factors: first the time and altitude covariance estimation is relatively poor (see Section 1.4.2.2), and second the vertical wind values below and at 1 km are very different, as can be seen figure 1.9. Hence samples gathered at altitudes lower than the final plateau may still influence the prediction of the GP, resulting in a slight positive bias.

Although very little difference can be observed when looking at the average error over the whole planned trajectory for each iteration (Fig. 1.10), we can see more differences when zooming in on single planning iterations. Fig. 1.13 shows how the prediction error evolves in average along a single planned trajectory as the points are predicted both farther from the current location and, more importantly, farther in the future. Here we can clearly see that the 'No energy' scenario comes ahead of the other, especially in the middle of the simulation. However as we are replanning every 10 seconds, and as the highest difference is on the points farther in the future, the effect is minimal when computing the mean over the first half of the planned trajectories (such as in Figs. 1.10 and 1.14).

Another indirect way to assess the quality of the mapping is to look at the position error induced by the wind prediction error at the end of each planning iteration. Indeed, as we execute the planned trajectories computed over the predicted wind

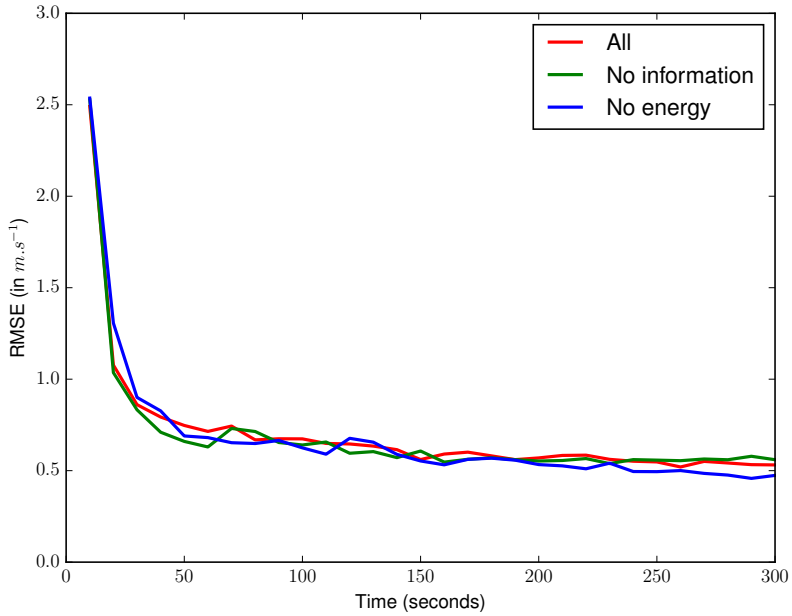


Figure 1.10: RMSE of the up wind prediction $y_\star$ along the executed trajectory at each iteration, computed over all the iterations of the three scenarios.

in open loop, the wind prediction is the sole source of error in the final position of the aircraft. The up wind prediction error induces error in predicting the altitude of the UAV, while the horizontal wind prediction contributes to the horizontal xy position error (Fig. 1.14). The xy and z position errors are of the same order of magnitude, with an average around 2.5 m in both cases, with a standard deviation of about 2.5 m, over a constant trajectory length of 150 m. The key importance of a good prediction of the up wind is clear: at the start of the simulations, the mean error in altitude is about ten times more than at the end of the simulation, whereas the xy error is only about 1.5 m. This is due to the nature of our Meso-NH simulations, which are initialized without advection (that is, no horizontal wind). In real-life cases, one should also estimate independently the mean horizontal wind, even if it varies only very slowly across space and time.

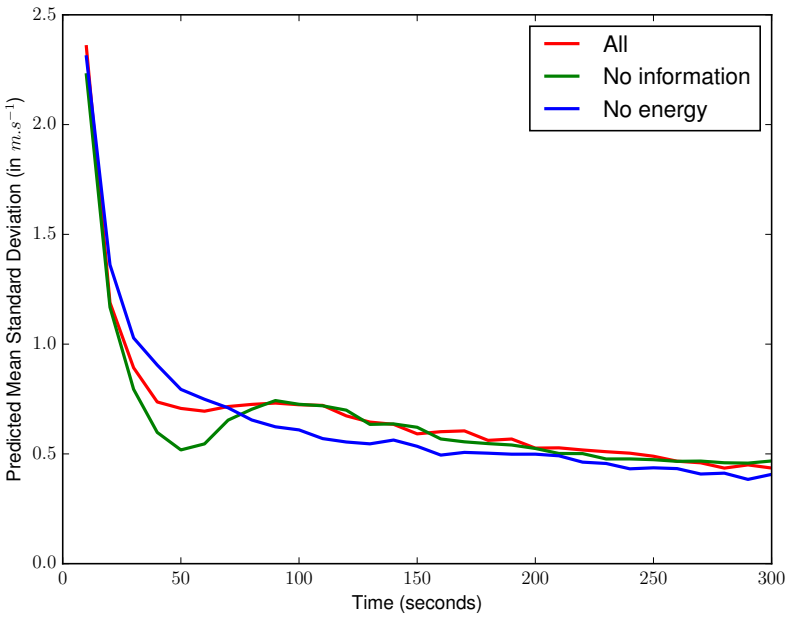


Figure 1.11: Mean of the standard deviation of the up wind prediction $\sqrt{\mathbf{V}[y_\star]}$ along the executed trajectory at each iteration, computed over all the iterations of the three scenarios.

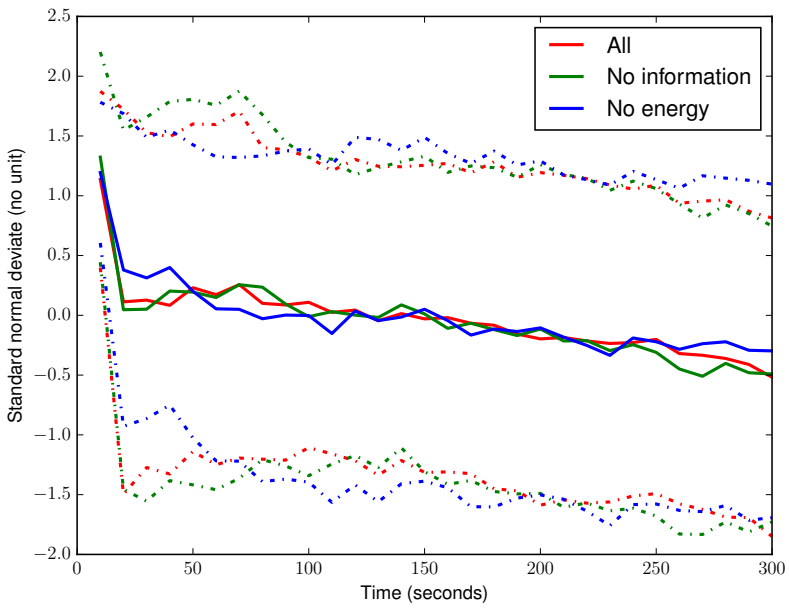


Figure 1.12: Mean and variance of the standard normal deviate $\frac{y-y_\star}{\sqrt{\mathbb{V}[y_\star]}}$ of the up wind prediction along the executed trajectory at each iteration, computed over all the iterations of the three scenarios.

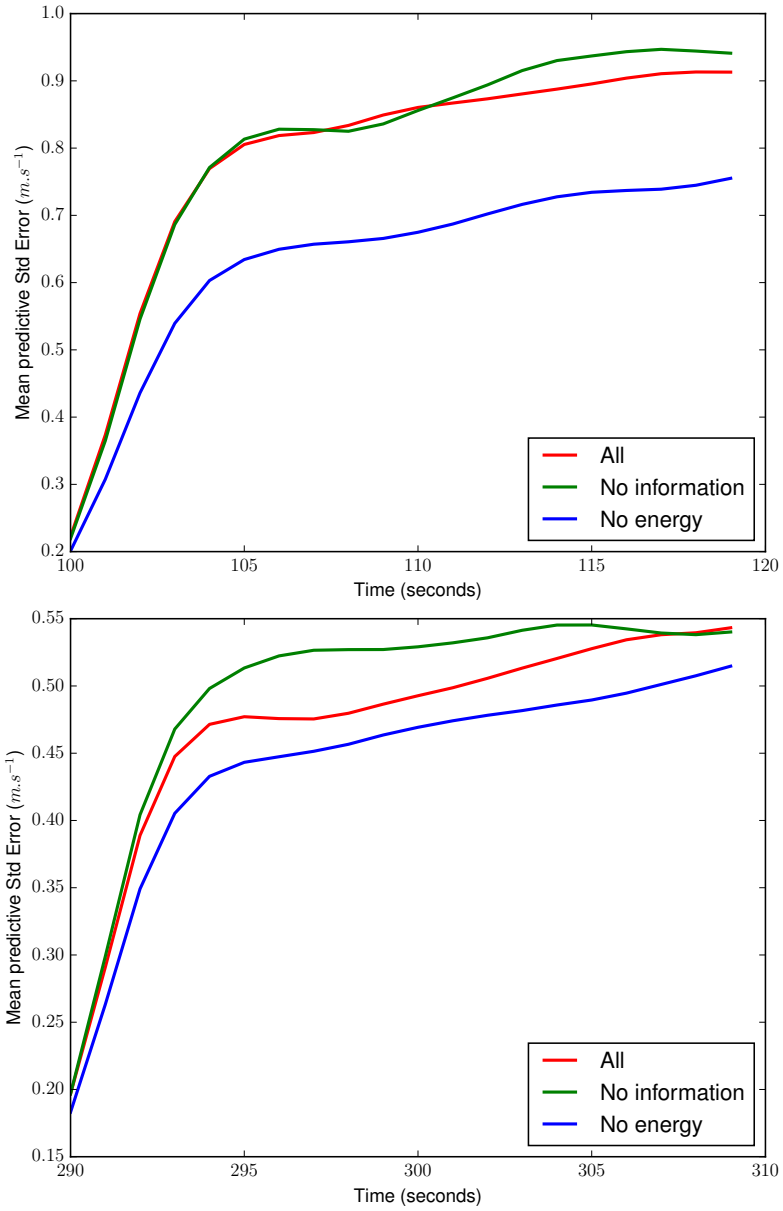


Figure 1.13: Mean of the Standard Error of the up wind prediction along the planned trajectory from the current planning time up to 20s in the future, computed over all the instances of the three scenarios. Top: current planning time is t=100s. Bottom: current planning time is t=290s.

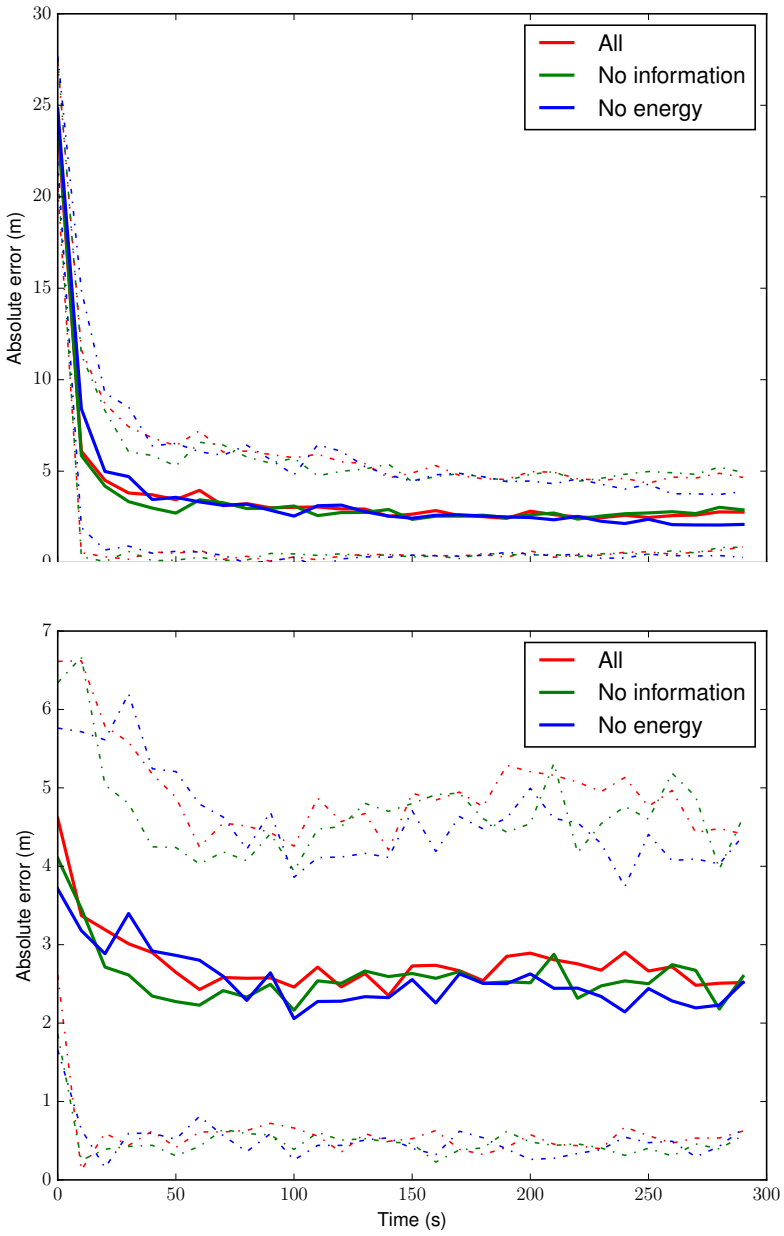


Figure 1.14: Position error after ten seconds of open-loop following of the planned trajectory with respect to the planned end position. Top: error on the z axis. Bottom: error on the xy axis.

1.4.2.2 Hyperparameters optimization

The evolution of the hyperparameter values of the GP model are also an indication of the quality and stability of the model. Figure 1.15 shows 2D histograms of the hyperparameter values, for all simulation scenarios combined, during the course of the simulation. The type of scenario does not impact much the hyperparameter optimization, but all parameters are not estimated equally well. The estimation of the parameter σ_n is particularly good and stable, with a median slightly above its actual value (set to $0.25\,\mathrm{m.s^{-1}}$), and a narrow distribution. It is not surprising as the additive noise added to the samples is a pure Gaussian one, and modeled as such. The process variance σ_f starts quite high at about 4 to end around 0.5, with the distribution narrowing around the median rapidly as simulation time passes. Considering the GP posterior variance in eq. (1.2), we see that the maximal posterior variance is $k(\mathbf{x}_\star, \mathbf{x}_\star) = \sigma_f^2$. Therefore the process variance indicates the maximum possible predictive uncertainty of the model. As the amplitude of the measurements does not decrease so drastically through time, it seems to indicate that it is not directly the process variance that is optimized. Indeed, as in the second half of the flights the UAVs gather samples in the same area (the plateau at 1 km altitude), the type II maximum likelihood (eq. (1.4)) used to optimize the hyperparameters overfits the data, which leads to the model being overconfident.

The evolution of the spatio-temporal length scales also exhibits some discrepancies. The length scales in the x and y directions are quite precisely estimated, and seem stable. The l_x hyperparameter converges in most cases to a value between 30 and 40 meters, the l_y parameter distribution being a bit wider, with the median around 40 to 50. Both hyperparameters show distributions peeking around the median and densely packed, indicating convergence towards a common value. On the other hand the vertical length scale l_z and the temporal length scale l_t exhibit a completely different behavior. The distributions remain very spread out in both cases. For a significant proportion of the simulations, the values are set to the upper bound given to the optimizer, indicating that the GP model is unable to estimate them properly. Even after five minutes, about 2% of the simulations still have a maximum value for the l_t parameter, and more than 20% of the simulations exhibit this behavior for the l_z parameter. This failure to estimate properly the time and altitude correlations is most probably caused by a problem of observability. As a matter of fact, the UAVs rapidly explore the x and y axis, and change altitude much slower. In addition, the time correlations are more difficult to estimate because time is the only dimension upon which the UAVs have no control: good estimation of this parameter is dependent on the frequency at which UAVs go back to a similar location in space. It is also possible that temporal correlations cannot be well fitted with our choice of model.

It should be noted that, as we have seen, the failure to extract some of the underlying processes parameter does not induce bad prediction performances: on the contrary, the model adapts the parameters to keep a good predictive accuracy.

This problem could be mitigated by keeping more samples in the model, or more interestingly by defining utility measures that lead to a more informative sampling for the hyperparameter optimization (as opposed to gathering information for map exploration purpose only). Finally, one could modify the GP model to account for the particular nature of the temporal dimension, either by developing a more appropriate kernel or by encoding the temporal dimension within a meta-model.

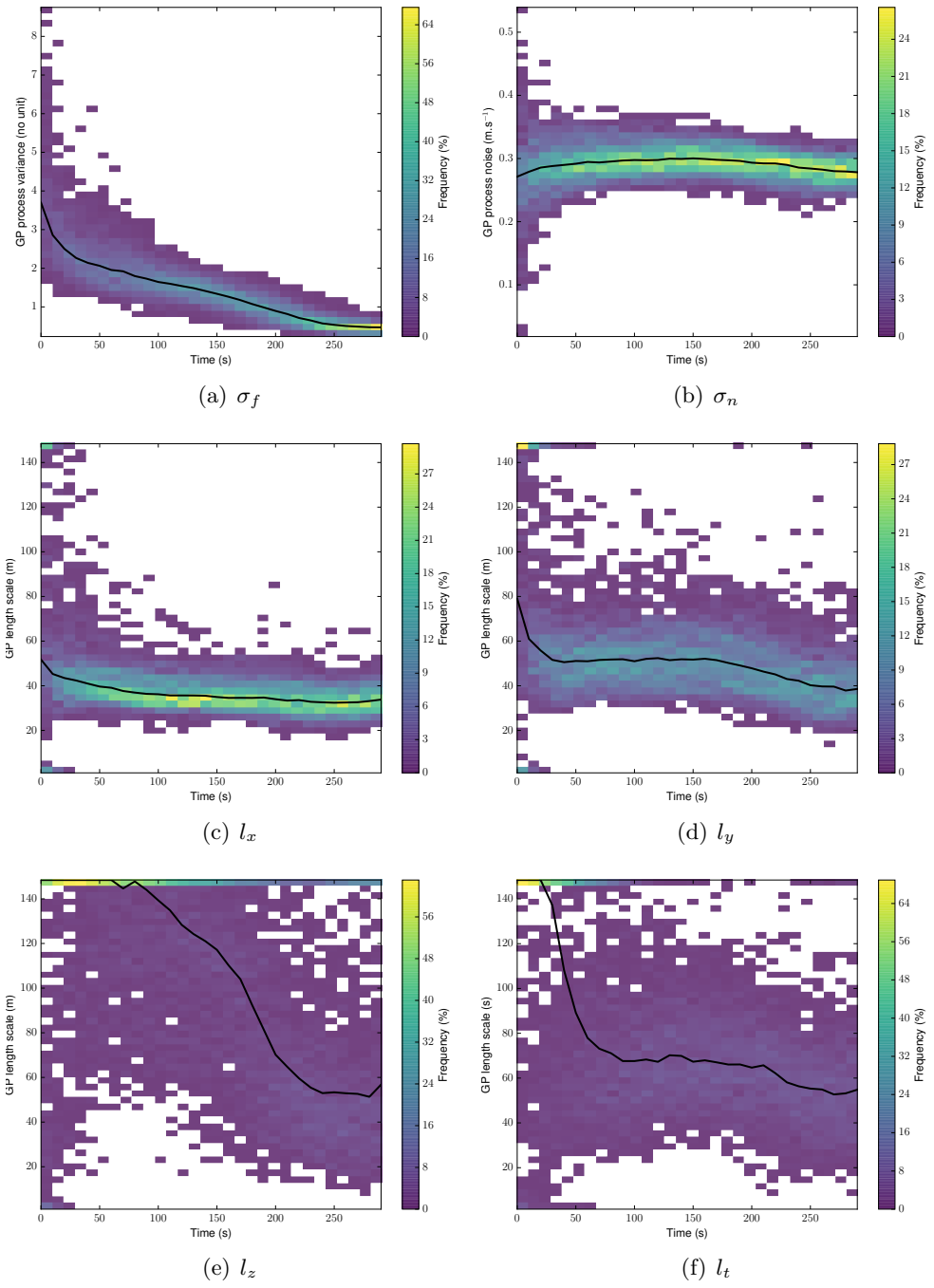


Figure 1.15: 2D histograms of the values of all GP parameters through time. The median is plotted as a black line.

1.4.2.3 Impact of weights in the utility function

While the definition of the scenario does not affect much the GP prediction, it strongly impacts the way the mission is achieved.

Starting with the "explore-box" objective, the UAVs have no trouble staying in the xy bounds of the box, but the choice of weights on the combination affects how they reach and stay in the z bounds (Fig. 1.16). When the optimization of energy is disabled ('No energy' scenario), the UAVs go faster to a mean altitude of 1 km (the center of the box), staying until the end of the simulation a bit above. Standard deviation lies equally on both sides of the 1 km mark. On the contrary when the information gain utility is disabled ('No information' scenario), the 1 km altitude is reached later and the mean altitude then starts to slowly decrease until the end of the simulation: the UAVs have difficulties maintaining their altitude because of the associated energy expense. We see the same effect when all utilities are equally weighted, although the mean altitude is a bit higher, allowing the UAVs to explore a larger portion of the box. Interestingly, when the energy utility is turned on, the mean altitude's variance starts decreasing after the first half of the simulation. It can be explained by the fact that, once the box is sufficiently explored, the absolute value of the information gain decreases up to a point where it does not counterbalance the energetic expense anymore, forcing the UAVs to lose altitude.

The impact of the energy optimization on the battery level is shown in figure 1.17. Optimizing the energy saves on average 4% percent of the battery level at the end of the simulation in the 'All' scenario and 5% in the 'No information' scenario as opposed to the 'No energy' one. Considering that the 'No information' scenario consumes just under 20% of the battery on average, the relative gain is respectively 20% and 25%. When looking at the progression of the battery level throughout the simulation, it is clear that even though the difference in battery savings is higher during the ascension phase ($0-1$ min), optimizing energy consumption is helpful throughout the whole mission to save battery.

The information gain is more difficult to assess: we have already seen that optimizing this utility impacts only a little the predictive power of the GP for the planning purpose and not at all the quality of the hyperparameter optimization. Also, as old samples are discarded in the model and the observed process is dynamic, it is difficult to compute a total amount of gathered information. Nevertheless one can look at the RMSE of the prediction computed on the whole "mission-box" at the end of the simulation (Fig. 1.18). Unsurprisingly there is little difference between the three scenarios: at the end of the mission, they all perform well enough. The limited temporal validity of the samples (due to the weather dynamics) and the instability of some hyper-parameters weigh heavily on the results. However the "No energy" is still indubitably statistically better than the other two scenarios, even if the absolute difference remains small (under $0.1\, m.s^{-1}$ between medians).

Another way to approach this is to look directly at the covariance matrices. Figure 1.19 illustrates the differences in density of the covariance matrices of the GP models by plotting the cumulative distribution of values in the matrices in

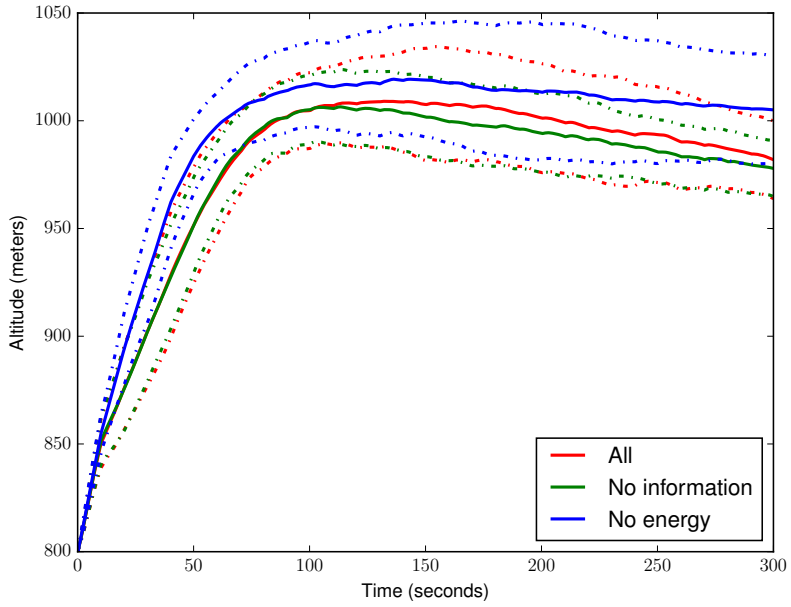


Figure 1.16: Mean altitude of the UAVs through time for all three scenarios. The standard deviation is plotted in dashed lines of the same color.

each scenario. The smaller the values, the more information the matrices contain (the more the samples are considered independent). It is clear that optimizing the information gain utility has a direct effect on the density of the matrix: when we plot the distance of the other distributions to the one of the "No information" scenario, there is up to 10% more values smaller than 10^{-2} in the "All" scenario, and 20% for the "No energy" one. In other words, the "No energy" scenario has half of the values in the covariance matrix under 10^{-2} whereas the "No information" one has only thirty percent, which is about a 65% relative increase. This result is also interesting to speed up the GP regression. Indeed if one were to use compact support kernels, the resulting matrices would become sparser with the information gathering utility, thus potentially speeding up the linear algebra computations.

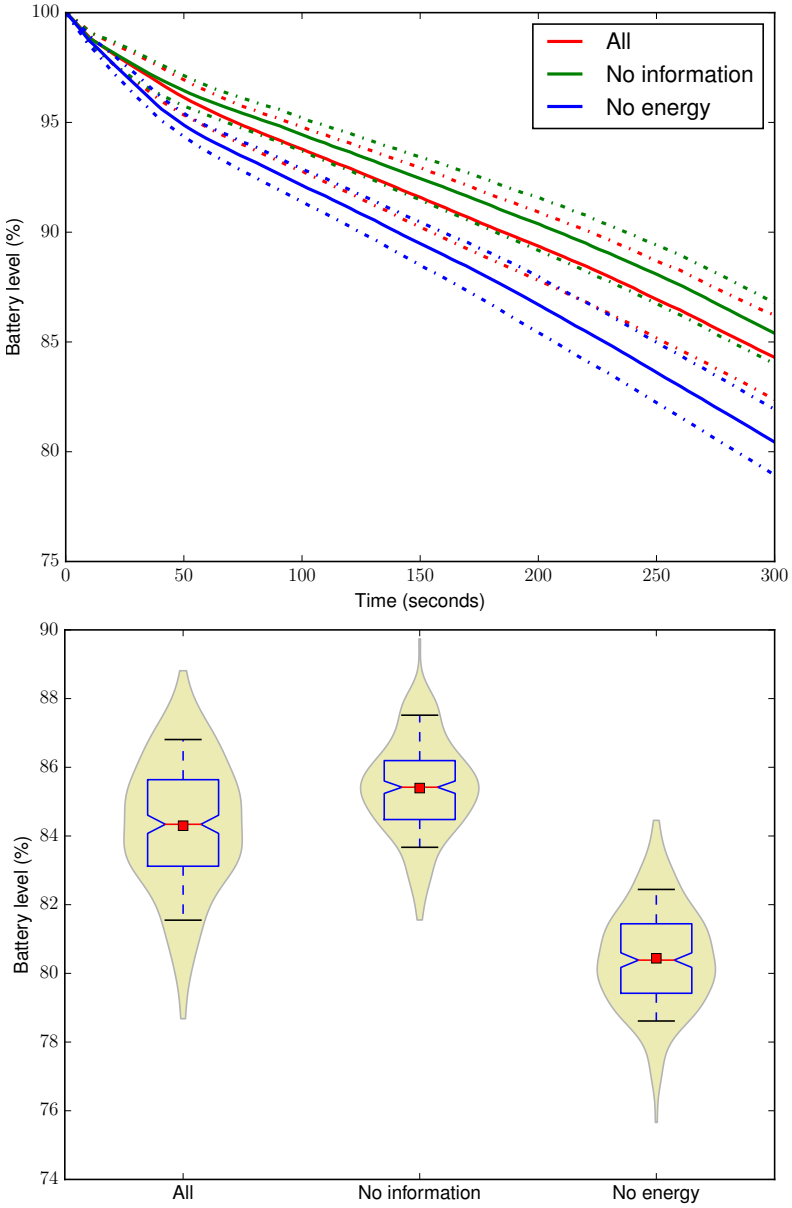


Figure 1.17: Evolution of the battery level. Top: Mean of the battery level through the simulation, over all instances of the three scenarios. The standard deviation is plotted in dashed lines of the same color. Bottom: Box and violin of the battery level at the end of the simulation, for all three scenarios. Whiskers at 9th and 91th percentiles, the red square mark indicates the mean.

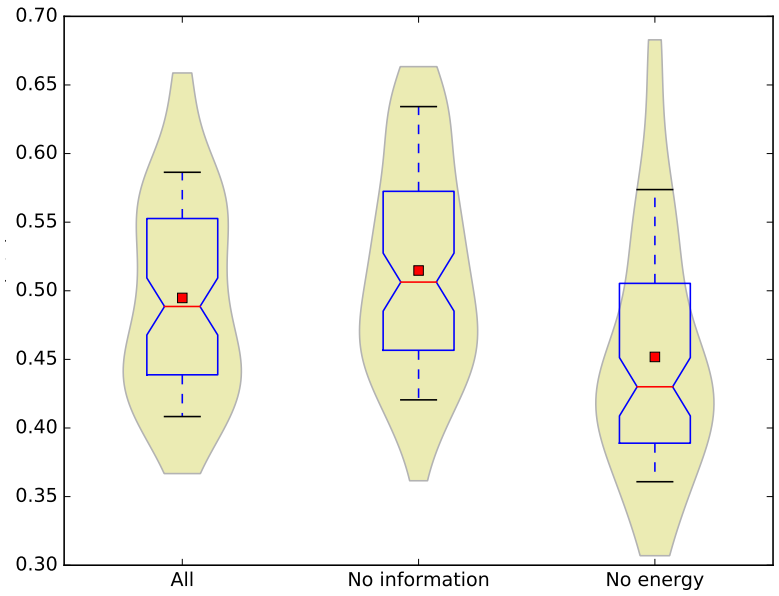


Figure 1.18: Box and violin plot of the RMSE of the prediction over the box defining the mission, at the end of the simulation (t=300s), over all instances of the three scenarios. The box is sampled at a spatial resolution of 10m, and the GP predictions on the samples at t=300s are compared with the ground truth.

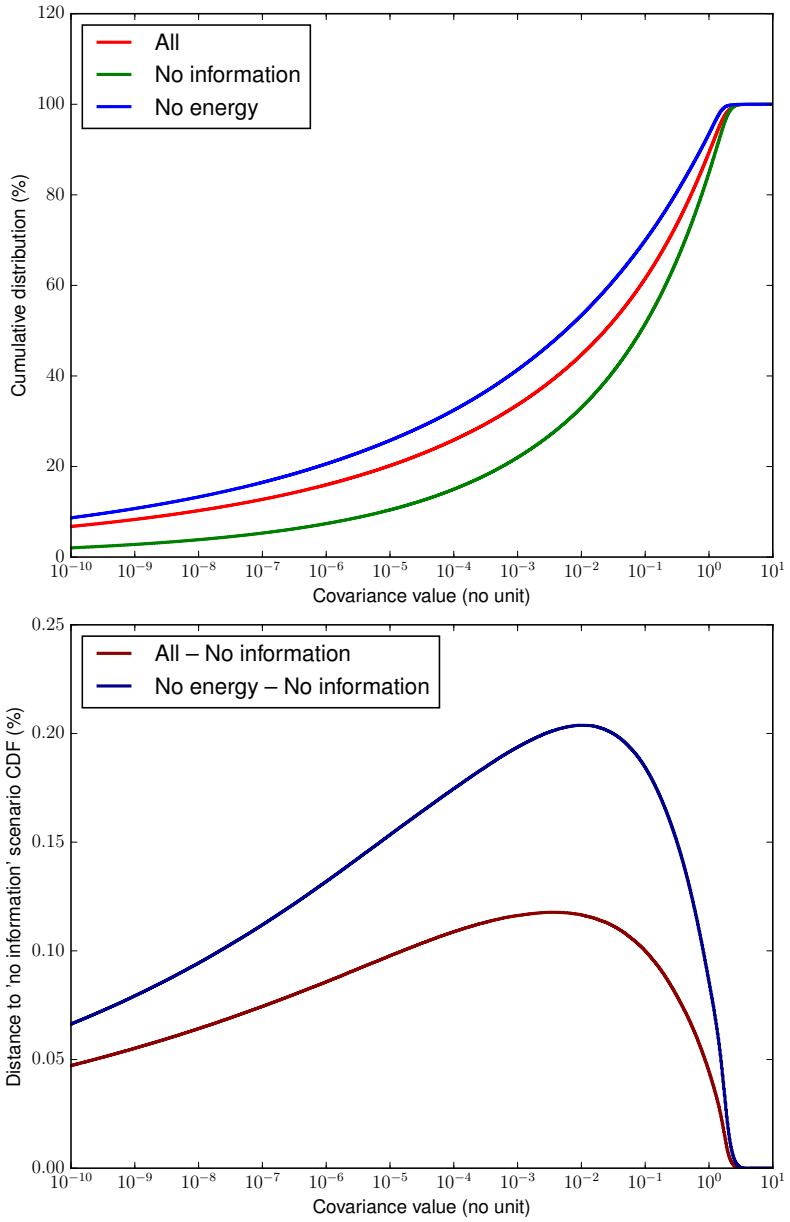


Figure 1.19: Top: Cumulative distributions of values in the covariance matrices of all three scenarios between 50 s and 100 s, in percents. Bottom: Distances between the cumulative distributions of values in the covariance matrices of the 'no information' and the other scenarios. Values smaller than 10^{-2} represent respectively 10 and 20 % more space in the covariance matrices of 'All' and 'No energy' scenarios than the in the 'No information' scenario (note the abscissas have a logarithmic scale).

1.5 Discussion

1.5.1 Summary

We have presented an approach to drive a fleet of information gathering UAVs to optimize the acquisition of information in a given area, while minimizing the energy expenses. The approach generates flight patterns that properly exploit the updrafts and generate accurate wind maps.

In particular, we have shown how a GPR framework can be used online to faithfully map wind fields corresponding to realistic cloud simulations. The resulting predictions are accurate enough to drive the team of UAVs during the course of the mission. A careful inspection of the resulting GP model, in particular of its hyperparameters, has shown its strengths and weaknesses. The predictive error of the GP model is quite low during the mission, resulting in negligible positional errors after the execution of the planned trajectories, and regardless of failures to estimate properly the underlying hyperparameters of the process. The proposed simulations exploit realistic wind and aircraft models, address the main task that atmosphere scientists expect from a fleet of UAVs, and exhibit the properties of the approach.

1.5.2 Future work

Numerous improvements remain to be addressed in order to define an efficient operational system.

The mapping framework would certainly benefit from better priors. A prior on the mean could in particular be provided by a macroscopic model of the cumulus cloud, which relates high level variables such as mean updraft as a function of the cloud base diameter for instance. The macroscopic model could also exhibit various regions in the cloud, within which priors on the hyperparameters could help to learn them more rapidly and more precisely. Furthermore, one should be able to explicit and exploit the correlations between the various atmosphere variables that the UAVs can measure, correlations that are pretty well known by atmosphere scientists (*e.g.* between the liquid water content and the vertical wind). One would need to resort to multi-task GPR for this purpose [Chai 2010]. Finally the temporal dimension should be more carefully accounted for, either by using a more appropriate kernel or as a parameter of a separate meta-model.

We have seen that the hyperparameter estimation can suffer from lack of observability. Deriving a utility measure aiming at maximizing information gain for their estimation would certainly improve the accuracy of the model. Another way to stabilize the hyperparameters would have the UAVs perform predefined synchronized measurements, for instance at different altitude levels, when the system detects a too important variability of the parameters.

As for trajectory planning, using more complex utility measure definition than a simple linear combination would allow the UAVs to achieve more complex tasks and to take into account specific preferences defined by the user. Additional criteria

and constraints (such as anti-collision) should also be considered. Finally, there remain various free parameters in the system (planning horizon, planning resolution, sampling rate...), that have been manually set: they could be learned online during the mission.

Finally, an overall integrated system architecture is yet to be defined. We foresee an approach that casts the problem in a hierarchy of two modeling and decision stages. A macroscopic parametrized model of the cloud, updated from the gathered data, would provide to the ground operator a coarse description akin to figure 1.1. He would then tasks the fleet with information gathering goals such as "map the top of the cloud", "assess updraft currents over the whole cloud height", "quantify the convection flux at the cloud base", "monitor the liquid water content within a given area and time lapse", etc. Given the required tasks and the current fleet situation (the UAVs positions, their on-board energy level, and their sensing capacities), a high-level decision process allocates UAVs to each task. This two tier approach would break the overall complexity of the problem, and allows the operator to task the fleet with high level goals, which are then achieved autonomously and independently by sub-teams of UAVs, using the mapping and trajectory planning approaches proposed in this paper.

We believe that deploying such a fleet within a distributed architecture remains far-fetched, and that a centralized architecture would provide more benefits than drawbacks. GP regression and hyperparameter learning are intensive computing tasks, that can for now hardly be embedded on-board lightweight UAVs. An approach in which all the gathered data are sent down to a powerful ground station that executes both the mapping and fleet control processes is a realistic option: only a very low bandwidth data link would be required for both the data reception and command transmission. Furthermore, the overall system would benefit from the use of a ground atmosphere radar, that would provide real-time estimations of some global cloud parameters such as its geometry, which directly conditions the amplitude of updrafts for instance. Such a system would also allow to update the position of the cloud: indeed most cumulus clouds develop in conditions with significant advective (lateral) wind[6]. The cloud simulations we exploited have been generated in the absence of such advective wind, and the cloud map is built in a geo-referenced frame, whereas in real flights one should map the cloud in a cloud relative reference frame, which evolution would be provided by the radar.

1.A Aircraft Model

In this Appendix, we provide the details of the flight dynamics model adopted for this work. We consider a simplified aircraft model to enable fast trajectory computations, but still able to capture the essential characteristics of the flight mechanics for a realistic trajectory optimization simulation. The key parameters

[6]except in tropical areas where cumulus can hover over the same position during their whole lifespan.

and coefficients used for the analytical calculations are estimated from a modified vortex-lattice analysis [Bronz 2013] of the aircraft.

In particular, we consider the Mako aircraft shown in Fig. 1.7, which will be employed for future experiments within the SkyScanner project. Table 1.1 shows its general aerodynamic and geometrical specifications.

Table 1.1: General aerodynamic and geometrical specifications of the Mako aircraft.

Description	Symbol	Value	Unit
Wing Span	B_{ref}	1.288	$[m]$
Wing Surface Area	S_{ref}	0.27	$[m^2]$
Wing Aspect Ratio	AR	6.14	$[-]$
Cruise Flight Speed	V_{cruise}	15.0	$[m/s]$
Minimum Flight Speed	V_{min}	12.0	$[m/s]$
Maximum Flight Speed	V_{max}	25.0	$[m/s]$
Propulsion Efficiency	η_p	≈ 0.45	$[-]$
Max Lift Coefficient	$C_{L_{max}}$	0.4	$[-]$
Profile Drag Coefficient	C_{D_0}	0.0233	$[-]$
Oswald Efficiency Number	e	0.93	$[-]$
Total Mass	m	0.9	$[kg]$

The trajectory computation is based on two control inputs: the power input P_{in} and the turn radius R. The airspeed V is considered constant, therefore the angle of attack is kept fixed for simplification. The trajectory optimization mainly requires the drag force evaluation, which has to be compensated by the input power. Note the air density is kept constant during the simulations, again for the sake of simplicity.

In order to calculate the performance within the flight envelope, the flight phases are isolated as steady banked turn and constant climb. The climb calculations derived from energy equations, the pull-up and down transition phases are approximated according to the maximum lift capability limit.

1.A.1 Steady Banked Turn Phase

During the steady banked turn phase, the vertical component of the lift force L_v is equal to the weight of the aircraft and its lateral component L_h compensates the centrifugal force as shown in Figure 1.20.

$$\sum F_z = 0 \Rightarrow W = L_v = L\cos\phi \tag{1.16}$$

For a given bank angle ϕ, the load factor n, given by

$$n = \frac{L_v}{W} = \frac{L}{L\cos\phi} = \frac{1}{\cos\phi}\,, \tag{1.17}$$

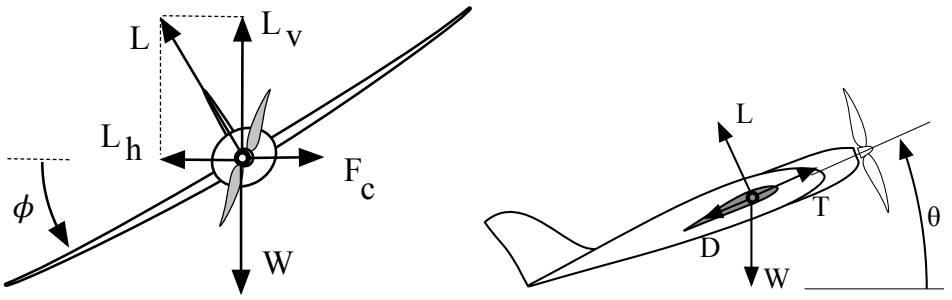


Figure 1.20: Forces applied during the steady banked turn phase (left) during the steady climb phase (right).

has to be accounted for in the lift force, which can then be expressed as:

$$L = \frac{1}{2}\rho V^2 S_{ref}(nC_L)\ [C_{L_{max}} > nC_L]\,, \tag{1.18}$$

with drag coefficient and resultant drag force being

$$C_D = C_{D_0} + k(nC_L)^2 \qquad \text{where} \qquad k = \frac{1}{\pi ARe} \tag{1.19}$$

$$D = \frac{1}{2}V^2 S_{ref} C_D \tag{1.20}$$

in oder to take into account the additional drag contribution coming from the induced lift force.

1.A.2 Rate of Climb (ROC) and power consumption

To maintain level flight, the required aerodynamic power is

$$P_{aero} = DV \tag{1.21}$$

Incorporating the thrust, we can calculate the climb rate of the aircraft as:

$$ROC = V_{climb} = V\frac{(T-D)}{W} = \frac{P_{prop} - P_{aero}}{W} \tag{1.22}$$

The maximum propulsive power is limited according to the specifications of the propulsion system used, whose efficiency η_p results in a higher power drawn from battery: $P_{prop} = \eta_p P_{in}$, where P_{in} is the input power drawn from the battery. The resulting V_{climb} is then:

$$V_{climb} = \frac{\eta_p \times P_{in} - P_{aero}}{W}\,. \tag{1.23}$$

The total propulsion system efficiency varies, as it is related to the flight speed and generated thrust force: a fine modeling of these variations would be required for a precise propulsion model. However, comparing electrical power input P_{in} versus

aerodynamic power output P_{aero} shows that a linear relation is a fairly accurate model for the considered flight speeds range, as shown in Fig. 1.21. This fact is exploited to define the total propulsion efficiency η_p in our simulation model.

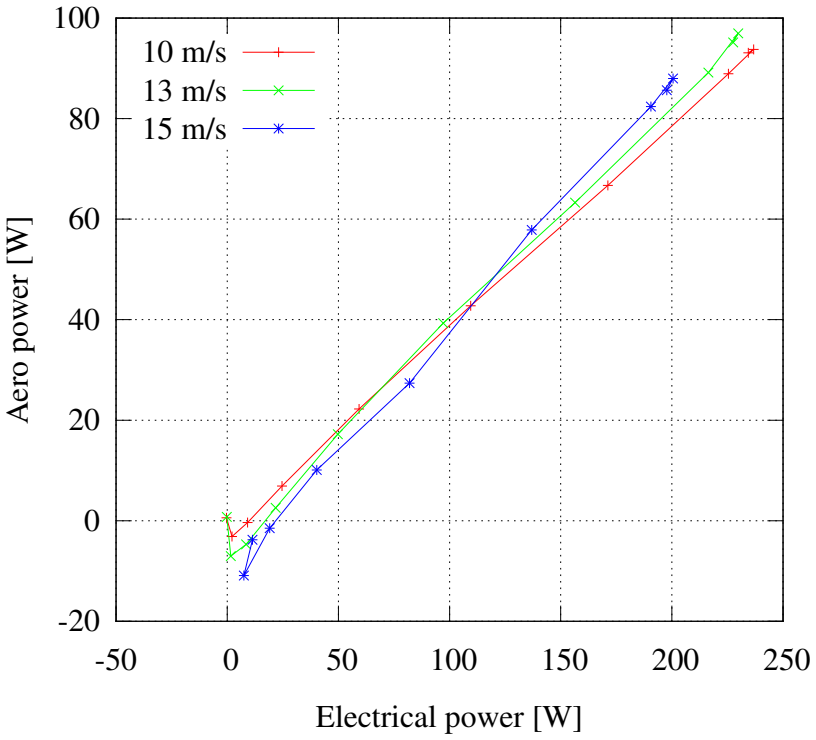


Figure 1.21: Resultant propeller aerodynamic power from battery input power for three different flight speeds.

1.A.3 Pull-up and Pull-down

The transition from level flight to steady climb is achieved by a short pull-up flight maneuver. Likewise, a pull-down flight maneuver is used to transition from level flight to steady descend phase. In our simplified aircraft model, as the flight angle of attack is assumed constant at all times, the distinction between climb and descent transitions is defined by the given power input P_{in}: if $P_{in} \times \eta_p$ is higher than the required level flight aerodynamic power P_{aero}, then the aircraft climbs.

The pitch turn radius can be calculated as:

$$R_{up} = \frac{V^2}{g(n-1)} \quad , \quad R_{down} = \frac{V^2}{g(n+1)} \tag{1.24}$$

where the contribution of the total aircraft weight is in the $(n-1)$ and $(n+1)$

terms. As the angle of attack is constant, the pitch rate $\dot{\gamma}$ is given by:

$$\dot{\gamma} = \pm \frac{V}{R_{up/down}} \tag{1.25}$$

1.B Trajectory Computation

Assuming a fixed airspeed V, a steady turn radius R and input power P_{in}, the trajectory can be computed by separating it in a pull-up (or pull-down) phase and a steady phase. Pull-up and pull-down phases are executed assuming a maximal allowed bank angle, thus assuring that the maximal allowable load factor will not be exceeded. First we compute the maximum allowed load factor n_{max}:

$$C_L = \frac{2 * W}{\rho V^2 S_{ref}} \tag{1.26}$$

$$n_{max} = \frac{C_{Lmax}}{C_L} \tag{1.27}$$

Using eq. (1.23), V_{climb} can be obtained from P_{aero} and P_{in}. This value represents the target value for vertical velocity for given P_{in} and R. The climb rate $\gamma(t)$ can then be computed as:

$$\Delta_\gamma = \arcsin(V_{climb}/V) - \gamma(0) \tag{1.28}$$

$$u = \text{sign}(\Delta_\gamma) \tag{1.29}$$

$$R_{up/down} = \frac{V^2}{g * (n_{max} - u)} \tag{1.30}$$

$$\dot{\gamma} = \frac{uV}{R_{up/down}} \tag{1.31}$$

$$\Delta_{tpull}(t) = min(t, \frac{\Delta_\gamma}{\dot{\gamma}}) \tag{1.32}$$

$$\gamma(t) = \gamma(0) + \dot{\gamma}\Delta_{tpull}(t) \tag{1.33}$$

where $\Delta_{tpull}(t)$ represents the duration of the pull up or pull down phase. Finally we can compute the projection on the z axis of the path on the $R_{up/down}$ circle during the pull phase, and assume a constant vertical velocity during the remaining time:

$$\Delta_{zpull}(t) = uR_{up/down}(\cos\gamma(0) - \cos\gamma(\Delta_{tpull}(t))) \tag{1.34}$$

$$z(t) = \Delta_{zpull}(t) + V_{climb}(t - \Delta_{tpull}(t)) \tag{1.35}$$

Knowing the vertical velocity, and assuming a constant total velocity V and turn radius R, we finally compute $x(t)$ and $y(t)$ using the change in the heading ψ to project the position on a circle of radius R tangent to the horizontal velocity vector. We first compute the heading $\psi(t)$:

$$\dot{\psi}(t) = \frac{V_H(t)}{R} \tag{1.36}$$

$$= \frac{V}{R}\cos\gamma(t) \tag{1.37}$$

$$\Delta_\psi(t) = \int_0^t \dot{\psi}(x)dx \tag{1.38}$$

$$= \frac{V}{R}(\sin\gamma(\Delta_{tpull}) - \sin\gamma(0)) + \gamma(\Delta_t)(t - \Delta_{tpull}) \tag{1.39}$$

$$\psi(t) = \psi_0 + \Delta_\psi(t) \tag{1.40}$$

Using $\psi(t)$ we deduce the xy position of the aircraft:

$$\begin{pmatrix} x \\ y \end{pmatrix}(t) = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + R \begin{pmatrix} -\sin(\psi_0) + \sin(\psi(t)) \\ +\cos(\psi_0) - \cos(\psi(t)) \end{pmatrix} \tag{1.41}$$

Finally, the remaining capacity of the battery $J(t)$ (in joules) is:

$$J(t) = J(0) - \int_0^t P_{in}(x)dx = J(0) - P_{in}t\,. \tag{1.42}$$

Bibliography

[Basilico 2011] Nicola Basilico and Francesco Amigoni. *Exploration strategies based on multi-criteria decision making for searching environments in rescue operations.* Autonomous Robots, vol. 31, no. 4, page 401, 2011.

[Bronz 2013] Murat Bronz, Gautier Hattenberger and Jean-Marc Moschetta. *Development of a Long Endurance Mini-UAV: ETERNITY.* International Journal of Micro Air Vehicles, vol. 5, no. 4, pages 261–272, 2013.

[Brown 2002] A. R. Brown, R. T. Cederwall, A. Chlond, P. G. Duynkerke, J.-C. Golaz, M. Khairoutdinov, D. C. Lewellen, A. P. Lock, M. K. MacVean, C.-H. Moeng, R. A. J. Neggers, A. P. Siebesma and B. Stevens. *Large-eddy simulation of the diurnal cycle of shallow cumulus convection over land.* Quarterly Journal of the Royal Meteorological Society, vol. 128, no. 582, pages 1075–1093, 2002.

[Chai 2010] Kian Ming Chai. *Multi-task learning with gaussian processes.* PhD thesis, The University of Edinburgh, 2010.

[Chung 2015] Jen Jen Chung, Nicholas RJ Lawrance and Salah Sukkarieh. *Learning to soar: Resource-constrained exploration in reinforcement learning.* The International Journal of Robotics Research, vol. 34, no. 2, pages 158–172, 2015.

[Condomines 2015] Jean-Philippe Condomines, Murat Bronz, Gautier Hattenberger and Jean-François Erdelyi. *Experimental Wind Field Estimation and Aircraft Identification.* In IMAV 2015: International Micro Air Vehicles Conference and Flight Competition, Aachen, September 2015.

[Corrigan 2008] C. E. Corrigan, G. C. Roberts, M. V. Ramana, D. Kim and V. Ramanathan. *Capturing vertical profiles of aerosols and black carbon over the Indian Ocean using autonomous unmanned aerial vehicles.* Atmospheric Chemistry and Physics, vol. 8, no. 3, pages 737–747, 2008.

[Das 2013] J. Das, J. Harvey, F. Py, H. Vathsangam, R. Graham, K. Rajan and G.S. Sukhatme. *Hierarchical probabilistic regression for AUV-based adaptive sampling of marine phenomena.* In IEEE International Conference on Robotics and Automation (ICRA), pages 5571–5578, 2013.

[Del Genio 2010] Anthony D. Del Genio and Jingbo. Wu. *The Role of Entrainment in the Diurnal Cycle of Continental Convection.* Journal of Climate, vol. 23, no. 10, pages 2722–2738, 2010.

[Diaz 2010] J.A. Diaz, D. Pieri, C. R. Arkin, E. Gore, T.P. Griffin, M. Fladeland, G. Bland, C. Soto, Y. Madrigal, D. Castillo, E. Rojas and S. Achí. *Utilization of in situ airborne MS-based instrumentation for the study of gaseous emissions at active volcanoes.* International Journal of Mass Spectrometry, vol. 295, no. 3, pages 105–112, 2010.

[Elston 2011] Jack S Elston, Jason Roadman, Maciej Stachura, Brian Argrow, Adam Houston and Eric Frew. *The tempest unmanned aircraft system for in situ observations of tornadic supercells: design and VORTEX2 flight results.* Journal of Field Robotics, vol. 28, no. 4, pages 461–483, 2011.

[Elston 2014] J. Elston and B. Argrow. *Energy efficient UAS flight planning for characterizing features of supercell thunderstorms.* In IEEE International Conference on Robotics and Automation (ICRA), pages 6555–6560, 2014.

[Elston 2015] Jack Elston, Brian Argrow, Maciej Stachura, Doug Weibel, Dale Lawrence and David Pope. *Overview of Small Fixed-Wing Unmanned Aircraft for Meteorological Sampling.* Journal of Atmospheric and Oceanic Technology, vol. 32, pages 97–115, 2015.

[Holland 2001] G. J. Holland, P. J. Webster, J. A. Curry, G. Tyrell, D. Gauntlett, G. Brett, J. Becker, R. Hoag and W. Vaglienti. *The Aerosonde robotic aircraft: A new paradigm for environmental observations.* Bulletin of the American Meteorological Society, vol. 82, no. 5, pages 889–901, 2001.

[Inoue 2008] J. Inoue, J. A. Curry and J. A. Maslanik. *Application of Aerosondes to Melt-Pond Observations over Arctic Sea Ice.* Journal of Atmospheric and Oceanic Technology, vol. 25, no. 2, pages 327–334, 2008.

[Kim 2015] Soohwan Kim and Jonghyuk Kim. *GPmap: A Unified Framework for Robotic Mapping Based on Sparse Gaussian Processes.* In Field and Service Robotics, pages 319–332, 2015.

[Lafore 1998] J. P. Lafore, J. Stein, N. Asencio, P. Bougeault, V. Ducrocq, J. Duron, C. Fischer, P. Héreil, P. Mascart, V. Masson, J. P. Pinty, J. L. Redelsperger, E. Richard and J. Vilà-Guerau de Arellano. *The Meso-NH Atmospheric Simulation System. Part I: adiabatic formulation and control simulations.* Annales Geophysicae, vol. 16, no. 1, pages 90–109, 1998.

[Langelaan 2011] Jack W Langelaan, Nicholas Alley and James Neidhoefer. *Wind field estimation for small unmanned aerial vehicles.* Journal of Guidance, Control, and Dynamics, vol. 34, page 1016–1030, 2011.

[Langelaan 2012] Jack W Langelaan, John Spletzer, Corey Montella and Joachim Grenestedt. *Wind field estimation for autonomous dynamic soaring.* In Robotics and Automation (ICRA), 2012 IEEE International Conference on, 2012.

[Lawrance 2011a] Nicholas RJ Lawrance and Salah Sukkarieh. *Autonomous exploration of a wind field with a gliding aircraft.* Journal of Guidance, Control, and Dynamics, vol. 34, no. 3, pages 719–733, 2011.

[Lawrance 2011b] N.R.J. Lawrance and S. Sukkarieh. *Path planning for autonomous soaring flight in dynamic wind fields.* In 2011 IEEE International Conference on Robotics and Automation (ICRA), pages 2499–2505, May 2011.

[Michini 2014] Matthew Michini, M Ani Hsieh, Eric Forgoston and Ira B Schwartz. *Robotic tracking of coherent structures in flows.* Robotics, IEEE Transactions on, vol. 30, no. 3, pages 593–603, 2014.

[Nguyen 2013] J. Nguyen, N. Lawrance, R. Fitch and S. Sukkarieh. *Energy-constrained motion planning for information gathering with autonomous aerial soaring.* In IEEE International Conference on Robotics and Automation (ICRA), pages 3825–3831, 2013.

[Petres 2007] Clement Petres, Yan Pailhas, Pedro Patron, Yvan Petillot, Jonathan Evans and David Lane. *Path planning for autonomous underwater vehicles.* Robotics, IEEE Transactions on, vol. 23, no. 2, pages 331–341, 2007.

[Ramanathan 2007] Veerabhadran Ramanathan, Muvva V Ramana, Gregory Roberts, Dohyeong Kim, Craig Corrigan, Chul Chung and David Winker. *Warming trends in Asia amplified by brown cloud solar absorption.* Nature, vol. 448, no. 7153, pages 575–578, 2007.

[Rasmussen 2006] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning.* the MIT Press, 2006.

[Ravela 2013] S. Ravela, T. Vigil and I. Sleder. *Tracking and Mapping Coherent Structures.* In International Conference on Computational Science (ICCS), 2013.

[Renzaglia 2016] A. Renzaglia, C. Reymann and S. Lacroix. *Monitoring the Evolution of Clouds with UAVs.* In IEEE International Conference on Robotics and Automation, Stockholm, Sweden, May 2016.

[Roberts 2008] C. G. Roberts, M.V. Ramana, C. Corrigan, D. Kim and V. Ramanathan. *Simultaneous observations of aerosol–cloud–albedo interactions with three stacked unmanned aerial vehicles.* Proceedings of the National Academy of Sciences of the United States of America, vol. 105, no. 21, pages 7370–7375, 2008.

[Sadegh 1997] Payman Sadegh. *Constrained optimization via stochastic approximation with a simultaneous perturbation gradient approximation.* Automatica, vol. 33, no. 5, pages 889–892, 1997.

[Soulignac 2011] M. Soulignac. *Feasible and Optimal Path Planning in Strong Current Fields.* Robotics, IEEE Transactions on, vol. 27, no. 1, pages 89–98, Feb 2011.

[Souza 2014] J.R. Souza, R. Marchant, L. Ott, D.F. Wolf and F. Ramos. *Bayesian optimisation for active perception and smooth navigation.* In 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 4081–4087, 2014.

[Spall 1998] James C Spall. *Implementation of the simultaneous perturbation algorithm for stochastic optimization.* Aerospace and Electronic Systems, IEEE Transactions on, vol. 34, no. 3, pages 817–823, 1998.

[Spall 2005] James C Spall. Introduction to stochastic search and optimization: estimation, simulation, and control, volume 65. John Wiley & Sons, 2005.

[Stevens 2013] Bjorn Stevens and Sandrine Bony. *What Are Climate Models Missing?* Science, vol. 340, no. 6136, pages 1053–1054, 2013.

Chapter 2

Repeatable decentralised simulations for cyber-physical systems

Abstract

Simulation is very helpful for the development of cyber-physical systems, as it enables the testing functionalities and their integration without full hardware deployment. For complex systems, such as fleets of heterogeneous robots, multiple simulators dedicated to particular physical processes must be interconnected through a middleware, so as to build a wholesome simulation and test the overall system. By using dedicated time management capabilities of a simulation middleware, repeatability can be achieved. However, simulation middlewares are not widespread in the robotics and cyber-physical systems community, and most simulations used in these domains are therefore not repeatable. We propose a new lightweight distributed architecture for time management, allowing to easily deploy complex simulations while strictly ensuring repeatability. A formal model of the architecture is provided, along with a proof of absence of livelocks. An open source implementation, with a binding to the robotics ROS framework is made available[1].

2.1 Motivations

2.1.1 On the need of distributed simulations

Complex cyberphysical systems integrate a large number of software components, some being close to the hardware such as drivers and controllers, others being more abstract components performing signal processing, reasoning and supervision tasks. For the development of such systems, simulation is key, as it allows a wide range of tests, from individual software components to the whole integrated software system. However, the interactions of complex cyberphysical systems with the world span numerous domains of physics. Let's consider for instance a mobile robot endowed with a variety of sensors such as cameras, lidars, radars, GPS and of actuators such as wheels, tracks, or even manipulation arms: its evolution is governed by physical process that range from electromagnetic wave propagations and interactions with a rich environment, to dynamics, including wheels/soil interactions.

[1] `https://github.com/crey0/dsaam`

To simulate such complex systems, one could resort to a great simulator (referred to as multiphysics simulator), capable of handling any number of composite entities, each composed of small simulated blocks. It would require simulating all the physics at the requested level for every sensor and actuator whilst managing the consistency of the whole simulation. However, such a simulator would be very complex, and each developer would have to integrate it in its development process. On the other hand, numerous simulators dedicated to a given domain of physics already exist, some being generic such as dynamic engines or graphics rendering to simulate vision, others being mode specialised, such as flight simulators or terramechanic simulators for wheels/soil interactions.

A wholesome multiphysics simulator would also hardly scale up to accommodate for cooperating heterogeneous cyberphysical systems, from fleets to swarms, of possibly heterogeneous robots, developed by different teams. The need for simulation in the integration of multiple actors has long been recognized and investigated, mostly in the context of battlefield simulations scenarios. This has lead to the development of comprehensive international standards such as HLA [HLA 2010] and DIS [dis 2012], which have been designed to integrate a distributed series of simulators.

2.1.2 On repeatability

In the context of simulation, repeatability can be defined as: *the same initial conditions should always produce the same simulation results.* One should not mistake repeatability (or replicability) for reproducibility, which entails the introduction of variability, such as running on different simulators, independent reimplementation of algorithms, switching data and initial conditions.

Repeatability cannot be used to prove scientific results and it's usefulness to detect scientific fraud is debatable [Drummond 2009]. Nevertheless, repeatability is a cornerstone of scientific simulation: it is the basic requirement to obtain verifiable results. If the simulation result depends on system or network latency, it may produce non-realistic results in unexpected ways, without the user noticing it.

Furthermore, repeatability eases a lot the development process: running the same simulation twice should produce the exact same results, allowing to easily reproduce bugs, and perform regression testing. It also enables launching batches of simulation, without worrying about system load: the simulation should produce the same results even in resource starvation scenarios where all the simulations cannot run concurrently in real time.

However such repeatability comes at a price. Indeed cyberphysical systems are complex, and there may be variability coming from the on-board processor load, depending on the scheduling protocol. Therefore to have an accurate and fully repeatable simulation, one would need a precise model of the hardware and run the software on the simulated hardware, as part of the global simulation in a *complete system simulation.*

On non-real time operating systems, the variability is the main issue preventing repeatability of simulations. It arises from many factors such as networking delays,

dropped messages (full buffers), hardware load (interrupts, CPU load, IO delays, ...), operating system load (time spent in kernel space is usually dependent on load from other processes), scheduling policies (other processes preempting on the simulation)...

An other issue preventing the repeatability of simulations are the non-deterministic interactions between the simulation environment and the tested software components, which can embed non-deterministic algorithms. Out of order message processing can occur, non-deterministic results can arise from complex multi-threaded interactions, and the usage of hardware clocks when the component depends on elapsed time can affect the components behavior If a time-management library can easily enforce message processing order, all the other issues should be considered by the developer of the components, so as to ensure deterministic results.

Because of all these issues, none of the two main approaches to simulation guarantees repeatability. The first approach is to accept having less accuracy in the simulation, and not model the hardware – it is referred to as Software In The Loop (SITL) simulation. It does not allow to detect software loops that take too long to execute, or to prove the consistency of the software in the presence of high system load or network latency. The second approach, referred to as Hardware In The Loop (HITL) simulation, is to run the software directly on the target hardware during the simulation. Yet it forces running the simulation in real time and still does not guaranty repeatability of the simulation due to system variability.

The time management library is responsible for managing the advancement of simulation time in each simulator. It should ensure that messages are processed and sent in timestamp order, and prevent message losses.

2.2 Distributed simulations: state of the art

2.2.1 Distributed simulation standards

We focus on software or standards intended to interconnect heterogeneous simulators to perform large-scale simulations in a distributed fashion. We refer to such tools as *simulation frameworks*. Two widely used international standards exist: High Level Architecture (HLA) [HLA 2010] and Distributed Interactive Simulation (DIS) [dis 2012].

Both frameworks have originally been developed with the support of public agencies for battlefield simulations. Monolithic in nature, these standards define everything from communication to time and simulation management, as well as domain specific models for objects (planes, ships, etc...) and algorithms (e.g. dead reckoning). A key difference lies in how messages are dispatched in the system.

HLA is *centralized* by nature, all messages from individual simulators are exchanged through a central process called the RTI (Run Time Infrastructure). On the other hand, DIS nodes that encapsulate simulators exchange message in a decentralized, peer-to-peer fashion. However this decentralized mode is only usable for real-time simulations. When switching to non real-time simulations, DIS needs

a central process, which is used to perform time management. In both cases, time management for non real time simulation is performed in a centralized way.

2.2.1.1 Common distributed simulation architecture

These standards exhibit a structure which is common to every simulation framework and can be used to compare them. Figure 2.1 shows a very simple abstraction of a simulation middleware in layers.

- At the core is the capability to exchange messages through a *middleware*, which can be centralized (HLA) or not (DIS).
- The *time management* layer is used to synchronize the simulators and enforce global coherency of the simulation. In both HLA and DIS, this is done using a central supervisor node, but other solutions exist (see subsection 2.2.2).
- *Simulation management* refers to setup, tear-down and monitoring of the simulation, managing the whole life process of the simulation.
- *Simulation components* are then built on top of this software stack. Here we can distinguish between *discrete event* and *continuous* simulation components (see section 2.2.1.3).

Not shown here is the *common object models* (i.e. data structures with appropriate semantics), necessary for the intercommunication between components and usually described using an *interface description language (IDL)*.

2.2.1.2 Time management

There exists multiple time management modes, which can best be described by the relation between *simulated time* and *physical time*.

- *Real time*: the simulated time flows exactly at the same rate as physical time.
- *Linear time*: the simulated time flows linearly with respect to real time, using a *speedup* coefficient.
- *Non-linear time*: the simulated time flows non-linearly, it can pause for arbitrary periods of time before resuming, and in some situations even go back in time (see 2.2.2).

Most simulators only support real and linear time, while HLA and DIS both support non linear (monotonic) time. While real-time simulation is best for validation of the integration of the software within the tested system (in particular using HITL simulation), non-linear time can be used (via the time management layer) to perform repeatable simulations, which is better to validate the result of algorithms.

2.2.1.3 Simulation models

There exists two main simulation models: discrete event simulation (DES) and continuous simulation. The former models the system as a discrete sequence of events, each event marking a change in the system. By contrast, in continuous simulation, the system is described by a set of partial differential equations which are integrated to compute the state of the system at any point in time.

Physical processes are obviously continuous in nature. Sensors and actuators, however, deal with discretized data. Therefore, in a typical simulation, the physical components of the system are modeled using continuous equations, but all input and output variables are updated and broadcasted periodically using a fixed time step for integration. Hence the simulator's interface could also be described by a DES model. Only a limited set of interactions between objects are non-periodic discrete events, such as interactions with third-party systems in the environment.

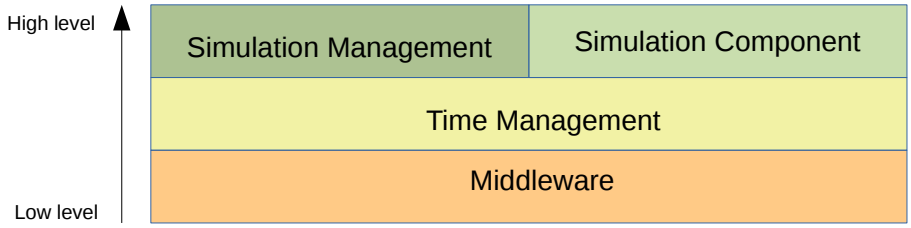


Figure 2.1: Software layers in a distributed simulation

2.2.2 Time management in parallel and distributed simulations

Besides the HLA and DIS standards, time management is a much researched problem in *parallel discrete event simulation (PDES)*, which addresses the problem of the execution of simulations on high performance computing platforms. The output of the simulations must be the same as the sequential version, therefore the output of the time management scheme should always produce deterministic results. A number of papers deal with reviewing time management in these fields, such as [Fujimoto 2015, Jafer 2013]. We only briefly introduce the different methods that have emerged from these fields, which can be split in two main categories: conservative and optimistic methods.

2.2.2.1 Conservative methods

Conservative time management enforces that all messages between simulators are processed in the order of their timestamps, in a deterministic fashion. Moving time forward (i.e. simulating the next step, or the next event) can only happen if all preceding events have been processed. This is enforced by blocking simulators until

all messages have been received, but can lead to deadlocks if the topology contains circuits.

A number of methods have been developed to address this problem, such as synchronous operation (all simulator share a global clock) or deadlock detection and resolution. Other methods, closer to the use case of distributed simulation, assume that the simulator exchange data through directional links and make use of *null messages*, messages containing only a synchronization data but no payload, to entirely avoid deadlocks. The Chandy-Misra-Bryant (CMB) algorithm [Bryant 1977, Chandy 1979] is based on the declaration of a *lookahead* value L for each node, which acts as a promise regarding the timestamp of the next produced message. If the node's last message was at time T, then it promises through the sending of null messages not to send any messages to any other node before $T + L$. However this method can suffer from livelock in some topologies, and from small lookahead values leading to large amount of null messages exchanged.

More recent work focus on methods to minimize the number of synchronization messages, as well as ensuring that no deadlock may happen. In [Chandy 1989], the authors make use of *"conditional events"*: each node sends with each message the probable timestamp of its next message, which is valid only provided no more incoming events with smaller timestamp will arrive in the future on the node's inputs. This helps reducing the number of null messages, but still requires the nodes to periodically broadcast some synchronization messages to all other nodes.

2.2.2.2 Optimistic methods

Pioneered by the "Time Warp" algorithm [Jefferson 1982], optimistic methods assume that all messages arrive and are processed in order. If an event is processed out of order, the entire simulation needs to rollback to a previous state. Thus optimistic methods require substantially more memory resource to maintain snapshots of previous states of the simulation. This method has lead to the development of dedicated operating systems such as *Time Warp OS* [Jefferson 1987], with the rollback process integrated in the core of the OS.

2.2.2.3 Time management in HLA

We briefly present time management in HLA, as it has a large record of usage, including in robotics [Gervais 2012, Chaudron 2011, Degroote 2015], and it is very similar to the *modus operandi* of DIS when deterministic processing order is enforced. Both use conservative methods to achieve it, using a central node and null messages to move time forward. The following is a quick summary of chapter 8 of the HLA standard documentation [HLA 2010].

In HLA a central node called the Run Time Infrastructure (RTI) is responsible for time management, and acts as a relay for message exchanges. All nodes, called *federates* in HLA, communicate only through the RTI during the simulation. The

standard supports enabling or disabling time management for individual federates as well as individual messages. Messages can be sent and delivered using two modalities:

- *Receive Order (RO)*: messages are delivered in received order by the RTI using a first-in, first-out policy.
- *Time Stamp Order (TSO)*: messages are delivered in strictly increasing timestamp order by the RTI.

Depending on its status, the federate can send and receive messages from either type:

- *Time Regulating*: can send and receive RO and send TSO messages
- *Time Constrained*: can send and receive RO and receive TSO messages

Non-regulating, non-constrained federates can only send and receive RO messages. Additionally, Time Regulating federates must declare a non-negative *Lookahead* value L. This lookahead indicates a lower bound on the next outgoing message. "Specifically, a time-regulating joined federate shall not send a TSO message that contains a timestamp less than its current logical time plus its current lookahead" *(excerpt from [HLA 2010], ss. 8.1.4, p. 152)*. A node can be both time-regulating and time-constrained.

Federates can only advance their logical time (i.e. the simulation time) by sending request to the RTI, and getting in response a *Time Advance Grant (TAG)*. There are two, slightly different, methods to achieve this:

- *Time Advance Request TAR(t)* : indicates that the federate is ready to advance to time t. RTI sends all TSO messages up to t before sending TAG(t).
- *Next Message Request NMR(t)* : indicates that the federate is ready to advance to time t, provided no messages are incoming before t. RTI responds either by sending a message with smaller timestamp t_m, or by directly granting the time advance. Upon completion, a TAG(t_m) or TAG(t) is delivered by the RTI.

Messages updating the world state at time t are called *Reflect Attribute Value* (RAV) messages, and are always timestamped when using TSO. Figure 2.2 illustrates a typical message exchange between federate and RTI allowing the federate to advance its logical time using a TAR.

TAR and NMR are almost equivalent, but NMR would be the preferred mode for discrete event simulators, while TAR would usually be preferred for time stepped simulators.

The lookahead L is not strictly needed for time management (indeed zero lookaheads are possible), but allows for quicker simulation. Indeed the RTI can advance the logical time of other federates further before the time regulating federates send a TAR or NMR request.

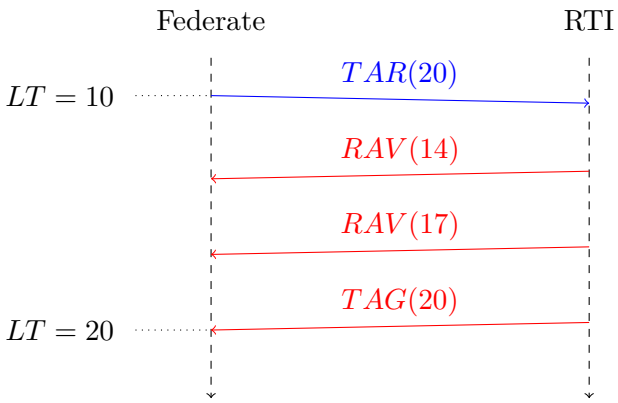


Figure 2.2: Typical HLA message exchange between a federate and the RTI in Time Stamp Order strategy. Excerpt from [Degroote 2015]

2.2.3 The case of robotics

2.2.3.1 On the lack of repeatability in robotic simulators

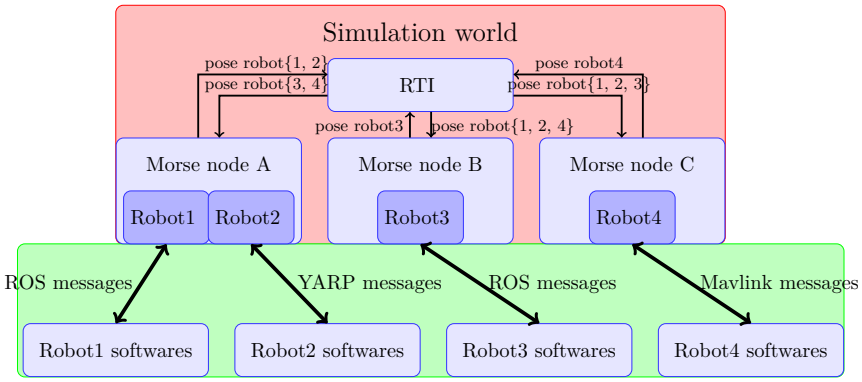


Figure 2.3: A Morse-HLA simulation with three Morse nodes and four robots. The HLA middleware is used to exchange message in the red zone (through the RTI noted RTIG in this figure), other heterogeneous middleware in the green zone. Excerpt from [Degroote 2015].

.

Simulators for robotic applications have mainly been developed and integrated in two ways:

- Monolithic, general purpose simulators such as Morse or Gazebo (or even FlightGear [Perry 2004]) built with at their heart the need to simulate physical interactions and graphically render the environment, very similarly to a game engine.

- Specialized simulators that have been integrated in the software stack of robots control architectures, which is very typical for drone autopilots (such

as Paparazzi [Brisset 2006]).

By their tight integration in the software stack, or by their monolithic nature, they lack in flexibility, and have not been constructed to run in a distributed fashion. As a notable exception, the Morse simulator supports the HLA middleware [Degroote 2015]. Together with the fixed time step mode, it is possible to run properly time-synchronized distributed simulations. To our knowledge, no further usage has been made of this capability of Morse outside the proof of concept described in this paper. Furthermore as depicted in figure 2.3, a typical usage would have multiple simulators communicating with each other through Morse and other middleware used to link the functional components to the simulation. Hence, except by moving all others components to the HLA middleware, those will not be aware of the time management, and the simulation still may not be repeatable. Even though there are very few, other case studies involving HLA to deploy robotics simulation exist. It is used in [Gervais 2012] to simulate embedded systems such as UAVs, but the paper includes very few details. A more detailed example can be found in [Brito 2015], here we find the same architecture as the previous Morse example (fig. 2.3), with HLA used as middleware for the simulation side and the robotics framework ROS for the functional components side, with a bridge linking the two. All example found outside Morse have been built ad-hoc, not as part of a reusable framework available to the general public.

Besides of the Morse simulator's proof of concept [Degroote 2015], to our knowledge no readily available robotic framework or simulator can properly run time-synchronized distributed simulations. Indeed, these simulators can only be used in real time mode and the *simulation time* advancement can only be slowed down or accelerated up to a constant factor with respect to the *physical time*. This corresponds to the way roboticists are used to test their software, not in a systematic way but as part of the development process to avoid testing it directly on board of the robot, or to avoid deploying it.

Repeatability of the simulations is not a widespread concern, and validation is done mostly by sampling a number of simulations until the developers are "confident enough" in their implementation. Still, we argue that with the growing complexity of robotic systems, and considering the number of published papers where all the work is done in simulation, there is a growing need for distributed, repeatable simulations, and the first requirement for it is to be able to manage time advancement between the components.

2.2.3.2 From robotic to simulation frameworks

Robotic frameworks Given the complexity of the software stack that needs to run on a robot and the wide range of abstraction, from low-level drivers for sensors and actuators to high-level reasoning algorithms, roboticists have long understood the need for *compositionality*. Usually, the software is broken in *components*, each responsible for a specific functionality. These components are decoupled from each other, typically living in different processes, and are interacting through

a middleware. Numerous frameworks have been developed along this modularity principle, among others YARP [Metta 2006], Orocos [Bruyninckx 2001], MOOS [Benjamin 2010], Genom3 [Mallet 2010] and ROS [Quigley 2009]. Most of these provide their own middleware, their ecosystem being tightly coupled with it. Others, such as Genom3, are more decoupled. They do not provide a middleware, allowing the user to choose between several options by the means of automatic generation of the interface with the selected middleware.

A middleware is a collection of libraries and executables responsible for exchanging data and triggering remote procedure calls between entities such as processes, living on the same or another computer. The data structures of the messages are described using an *interface description language*. The middleware provides an implementation of the data structures in the target programming languages and is responsible for the exchange of these between components. A large number of middleware exists, but can be broken down into two categories according to their architecture:

- Centralized communication: all messages are collected by a central process, which is then responsible for dispatching them (e.g. MOOS).
- Decentralized communication: the messages are exchanged in a peer-to-peer fashion (e.g. ROS, Orocos, YARP)

Building on middleware, a large number of *robotic frameworks* have emerged. These provide additional tools and functionality to ease the development process, such as:

- Standardized messages for common data types (images, poses...)
- Reusable components for common use cases (such as drivers for sensors and actuators) through common software repositories
- New semantics of communication (*e.g. ROS Actions*)
- Management of the component's behavior (*e.g.* Genom3 description language exposing Tasks, Activities, Codels and ports)
- Introspection, display and debugging tools
- A common build and deployment process

Simulation frameworks Much on the same way, simulation frameworks, such as HLA and DIS, are building upon middleware, adding specific capabilities needed for the simulation process:

- Simulation management: setup, tear-down, joining and leaving of entities, transfer of ownership of simulated objects between simulators.

- Time management: running the simulation in real time or non-real time (slower, faster, or adaptive time flow), making the simulators wait and synchronize with each other.

- Geographic zoning: grouping objects by geographic zones of interaction for large scale simulations.

- Defining and integrating domain specific standard object types (for example planes, ships, soldier units for battlefield simulation)

- Integrating domain specific capabilities (such as dead reckoning for fast moving objects in DIS)

Roboticists have been developing simulators for their needs, and have been integrating them in their software architecture. This is usually done in two complementary ways. Common simulators are integrated in an *ad-hoc* way, either as part of a component (outside the communication framework), or replacing one or many components with simulated counterparts. The simulated component is then directly connected through the robotic framework as if it were the real one. But it does not permit to run distributed simulations, where the simulators are aware of each other and interconnected. For this purpose, the world is split in two categories: real components and simulators. Real components live in the robotic framework, while a simulation framework is used for the simulators. However, the functional components do not know they are part of a simulation. This has the advantage of simplicity, but also a few drawbacks:

- The roboticist needs to master two different frameworks, which adds complexity, slowing down the development and leading to mistakes.

- Since components are not aware that they are part of a simulation, it is more adapted to simulations were time flows in real time, or at least with a constant time dilation factor. Being unaware of it, one component can not wait for the completion of a simulation in another one. Therefore, to have proven consistent simulations, one would need to know the Worst Case Execution Time (WCET) of all the simulators, and verify that the requested period can be held. However, the most frequent solution is to perform a trial-and-error process, where one adjusts the frequency and time factor of the simulation until it is "accurate enough" in most cases, on a specific machine. The required level of accuracy is not well defined and deploying the simulation on another computer may need hand tuning.

- Simulating slower or faster than real time can have unforeseen consequences, because the developer or integrator of each component is not aware of this possibility. Some processes may assume that data arrive at a certain real time frequency, taking the hardware clock as a reference point. Complex multi-threaded interaction may happen that render the results of the component's

computations useless, such as new incoming messages canceling computations still running on previous data.

- It is not possible to perform simulations where simulated time flows in an adaptive way, depending and synchronized with the advancement of the simulators.

2.2.3.3 Towards a thin simulation layer

Therefore, instead of relying on external simulation frameworks, another solution would be to add a thin simulation layer on top of the robotic one. This would allow to integrate simulations in a more seamless way with existing code. Reuse of the same underlying middleware and framework would also ease the cognitive load of the developer. Furthermore, it would draw its attention to potential problems coming from the time flow of the simulation, by allowing all components to be aware of their taking part of a simulation.

One could summarize the requirements of such a satisfying simulation framework for robotics as follows:

- Compose distributed, modular simulations, interconnect simulators
- Switch between simulated and real components, mix them within the same simulation
- Synchronize simulations, control time advancement (faster or slower than real time simulations)
- Run real time simulations
- Run simulations in batches
- Integrate well in existing robotic software
- Produce *repeatable* simulation results

We present in the rest of this paper an approach to *time management* in distributed simulations (as exposed at the end of 2.1.2) that is the keystone towards repeatable simulations, and we will not consider *simulation management*.

2.3 DSAAM: a decentralized time management architecture

Overview We propose a *time management* framework that can be easily implemented on top of existing robotic frameworks. It manages time in a completely decentralized manner, without the need for a central node or direct communication between all simulated components. Most robotic framework share this feature: communication is performed in a peer to peer fashion, with a supervisor node only

responsible for bookkeeping. Some, such as *ROS2*[2], are altogether migrating toward a completely decentralized approach (for the discovery of services as well): the introduction of a central node would thus break the philosophy of the underlying middleware, and also add a significant overhead.

The proposed solution is close to existing *conservative* algorithms found in the PDES literature. In particular, it could be seen as an extension of the "conditional event" scheme proposed in [Chandy 1989], but additional constraints allow us to simplify the algorithms and lower the communication between nodes by getting rid of the broadcasting step. Note though that the concerns of the PDES community are quite different from distributed simulations in robotics: in PDES, a single simulation is usually distributed on a computer or a cluster and parallelism is intended to reduce execution time, whereas our primary objective is the interconnection of heterogeneous simulators and modularity, while guaranteeing repeatability.

Our framework, named DSAAM for "Decentralized Synchronization Architecture for Asynchronous Middleware", satisfies the two following essential properties:

- *Decentralization*: it relies on a peer-to-peer, one to many, communication model.
- *Time consistency*: messages are emitted and processed in a deterministic fashion, and components must wait for each others messages in order to advance. This guarantees repeatability of the simulation if each simulator itself is deterministic (see 2.1.2).

Characteristics of a DSAAM System To enforce time consistency, the following constraints are imposed from the underlying simulated components:

- *Time-stamped messages*: all exchanged messages must be time-stamped in a meaningful way. Implicitly, the semantics of the timestamp is that each message represents a piece of the world state at the simulation time indicated by its timestamp.
- *Periodicity*: messages are sent with a timestamp period that is known in advance, so as to know at which simulation time to expect the next message. This period can vary during the course of the simulation.

The last constraint is a particularly strong one. For example, it forbids request-response mechanisms that may be triggered at arbitrary points in simulation time. We will see in detail why this constraint is necessary and discuss how it could be loosened.

A DSAAM system is made of a collection of nodes that encapsulate the simulators, and that exchange messages through *flows*. *Flows* have always one source (publisher), and any number of sinks (subscribers), as depicted in fig. 2.4. We also assume that every exchanged message includes a *timestamp* and a validity *period*.

[2] https://index.ros.org/doc/ros2/

Timestamps and periods can be any kind of variable that all belong to a totally ordered set endowed with addition. We use the natural integers $\mathbb{N}$.

The introduction of the period represents a contract between sources and sinks: if a sink receives a message with timestamp t and period δ, then the *next* message on this flow will have timestamp $t + \delta$. Semantically, the message represents the state of the variables that it contains between simulation time in $[t, t + \delta[$, $t + \delta$ being the time at which the next message will carry the updated state of the variables. It is very similar in nature to the notion of lookahead, but instead of providing a lower bound it provides the *exact* timestamp of the next message. The period δ may be fixed, but it is not a requirement in our architecture, it may change between each message.

Properties of a DSAAM node To satisfy these constraints, the following four rules on consumption and emission of messages inside a node must be enforced:

1. Messages are consumed by increasing timestamp order, no matter the source they are coming from.
2. The emission of a message with timestamp T forbids future consumption of messages with timestamp $T' < T$.
3. The consumption of a message with timestamp T forbids future emission of messages with timestamp $T' \leq T$
4. No incoming message can be lost or discarded before it is consumed by the simulation.

These rules are necessary to ensure the time consistency of the variables a node uses as input for the simulation, which is the first pre-requisite to provide a repeatable simulation.

To ensure time management, DSAAM defines for each simulator it encapsulates the following three states:

- Wait (**Wa**): the simulator is computing its next step based on previous inputs and state.
- Emit (**Em**): the node is sending the updated state on one of its outgoing flows (may be blocking if the buffer of one of the sink nodes is full).
- Consume (**Co**): the node is consuming the next (smallest timestamp) incoming message on one of its inputs (may be blocking if the corresponding message has not yet arrived).

After each emission or consumption of message, the node returns in the *wait* state. Figure 2.5 is an informal graphical description of the transitions between these states according to the aforementioned rules.

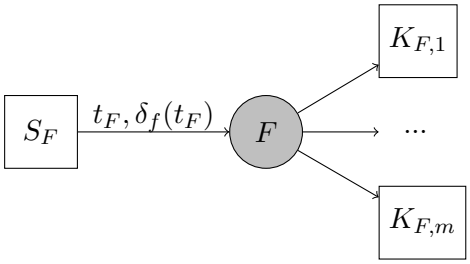


Figure 2.4: A flow F with source node S and sinks $K_{F,1}...K_{F,m}$, with next emitted message timestamp t_F and period $\delta_f(t_F)$. The state of sinks is not represented.

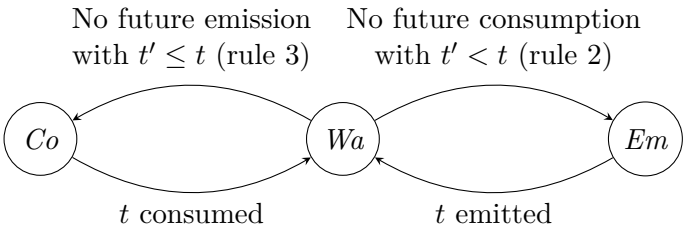


Figure 2.5: Informal description of the behavior of a node with transitions between *Wa*it, *Co*nsume and *Em*it states according to the rules 2 and 3.

Illustration We further illustrate the behavior of DSAAM using a small example with two nodes and two flows forming a circuit shown figure 2.6. Figure 2.7 depicts a valid message exchange sequence in this system. In particular, node b must wait processing of message $t = 1$ to send its own message at $t = 2$. However both have to send their message $t = 2$ before processing (consuming) the corresponding message of the other node. Indeed if both were to wait to receive the message before sending their own, it would result in a deadlock.

An invalid message exchange sequence is shown figure 2.8. In this case node b fails to wait for the processing of the message emitted by node a at $t = 1$ before sending its own message at $t = 2$, which is prohibited by rule 2, thus potentially breaking the time consistency of its internal simulation. Such an error is prevented by the time management layer, which would detect the fault and block the sending of this message by node b as long as the message of node a emitted at $t = 1$ is not processed.

In the following section, we propose a rigorous formal model for DSAAM, on which we rely to prove crucial properties that hold for any DSAAM system.

2.4 Formal Model and Proof

In this section, we fully formalize DSAAM. We start by presenting transition systems (Sect. 2.4.1.1), the formalism on which operational semantics of DSAAM are based. We give syntactical definitions then operational semantics of a generic

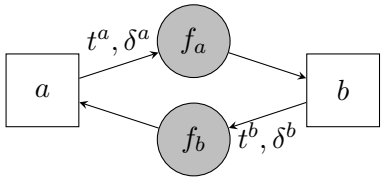


Figure 2.6: An example with two nodes a and b and two flows with messages timestamped t^a and t^b, forming a loop. Each node has only one outgoing flow with the same name as the node, and each flow has only one sink.

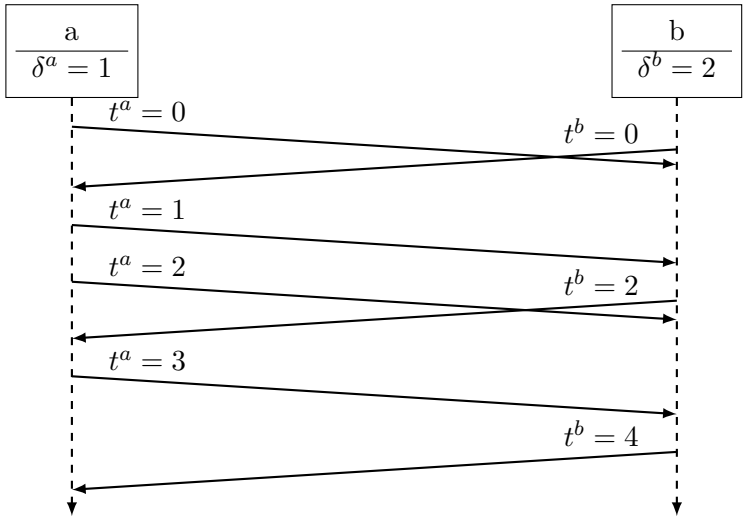


Figure 2.7: Valid message exchange sequence between nodes a and b from example depicted in fig. 2.6. Period is constant with $\delta_a = 1$ and $\delta_b = 2$. Initial timestamp is $t_0^a = t_0^b = 0$. Physical time elapses along the dashed lines, while message timestamps are displayed above the arrows. After reception, messages are assumed to be processed instantaneously.

DSAAM system. Operational semantics are derived from syntactical elements while unambiguously specifying the behavior described informally in Sect. 2.3.

2.4.1 Preliminaries

2.4.1.1 Transition Systems

Syntax A Transition System TS is a tuple $\langle U, Q, q_0, \longrightarrow \rangle$ where:

- U is a finite set of variables. Each variable is implicitly typed. We use $\mathrm{dom}(u)$ to denote the domain of variable u in U,
- Q is a set of states. Each state in Q is an interpretation of variables in U, that is a mapping from variables $u \in U$ to values in $\mathrm{dom}(u)$,
- $q_0 \in Q$ is the initial state that maps each variable to its *initial* value,

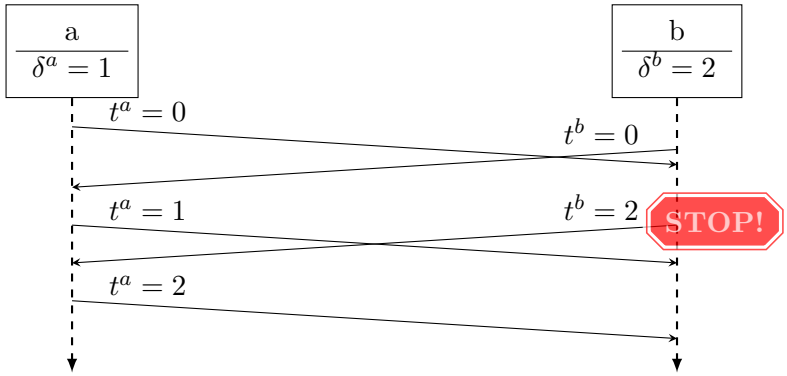


Figure 2.8: Invalid message exchange between nodes a and b from example depicted in fig. 2.6. Period is constant with $\delta_a = 1$ and $\delta_b = 2$. Initial timestamp is $t_0^a = t_0^b = 0$. Node b sends a message timestamped $t^b = 2$ but did not yet receive and process the input $t^a = 1$, violating constraint 2.

- $\longrightarrow$ is a set of transitions. Each transition t in $\longrightarrow$ is a binary relation that defines for every state q in Q a (possibly empty) set of successors $t(q)$ subset of Q. We write $q \xrightarrow{t} q'$ iff $q' \in t(q)$.

A TS is *deterministic* iff there exist not more than one successor of a given state q over the same transition t. Formally, a deterministic TS must satisfy the following condition:
$\forall q \in Q, \forall t \in \longrightarrow : |t(q)|1$ where $|s|$ denote the cardinality of the set s.

Semantics The semantics of a TS is retrieved through the following notions:

- Current state: the current state of the TS is the state q in Q that agrees with the current valuation of each variable u in U. Initially, the current state is q_0,
- Enabled transition: a transition t in $\longrightarrow$ is enabled iff q is the current state of the TS and $t(q) \neq \emptyset$.

The evolution of TS is thus subject to *taking* enabled transitions. After taking an enabled transition t in $\longrightarrow$, the current state of the TS is a state q' in $t(q)$ (uniquely defined in the TS is deterministic). The possible evolutions in a TS define the set of *reachable* states Q_R subset of Q. We say that state q is reachable, that is $q \in Q_R$ iff there exists a (possibly empty) sequence of transitions σ such that $q_0 \xrightarrow{\sigma} q$.

2.4.1.2 Transition Diagrams

Syntax We define a graphical notation for a TS (called a Transition Diagram, or TD for short) and a composition operation between TDs. We use a TD to describe a *component* of the system. Therefore, the composition of TDs is a way to build complex systems through synchronization and shared variables. In sum,

the composition of multiple TDs (viewed as components) results in a TS (viewed as the *system*).

A TD C (component) is a finite directed graph where V is the set of vertices and E the set of edges. C operates on a finite set of variables, X. The vertex v_0 in V is the unique initial vertex of C. Each *edge* e in E is associated with a *guard* g_e and a set of operations op_e. If r connects vertex v_a to vertex v_b, then we may write $v_a \xrightarrow{e(g_e, op_e)} v_b$.

- g_e is a boolean expression over X whose truth value denotes the enabledness of the edge e. That is, e is enabled iff v_a is the current vertex of C and g_e evaluates to *true*. In the remainder of this paper, guards that are always true are not represented,

- op_e is an atomic sequence of operations over variables in its *scope* X. Thus, op_e maps each variable x in X to a value in $dom(x)$, resulting from applying such operations. If op_e maps a variable x in X to its current value (that is, $op_e(x) = x$), then we say that op_e is *side-effect free* on variable x. In the remainder of this paper, side-effect free operations are not represented.

We show in Fig. 2.9 a simple TD example with two vertices, v_0 and v_1, and two edge e and e'. The initial vertex, in this case v_0, is denoted with an incoming edge without source vertex. Guards are in green and instructions of operations in red.

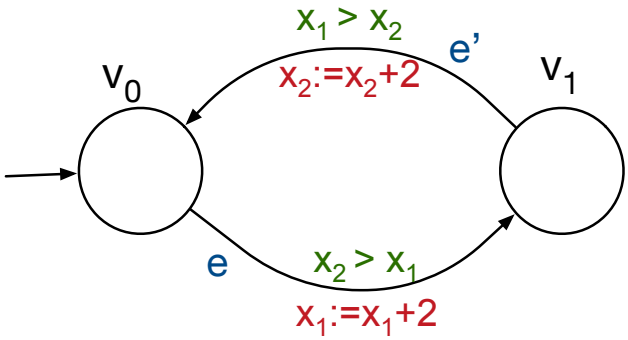


Figure 2.9: A TD example

This TD operates over a set of variables $X = \{x_1, x_2\}$, where $dom(x_1) = dom(x_2) = \mathbb{N}$ ($\mathbb{N}$ is the set of natural numbers). The guard g_e (respect. g'_e) evaluates to true iff $x_2 > x_1$ (respect. $x_1 > x_2$). The set of operations op_e (respect. op'_e) maps the variable x_1 (respect. x_2) to the result of adding 2 to its current value, and the variable x_2 (respect. x_1) to its current value (which explains why the latter operation, side-effect free, is not represented).

Semantics Given a TD C, we associate its meaning, $[\![C]\!]$, that is a TS $\langle U, Q, q_0, \longrightarrow \rangle$ that corresponds to C and give thus its semantics (Sect. 2.4.1.1). We use the following notations:

- $q(g)$ denotes the truth value of a guard g at the state q,
- $q'_{|Y} = op(q_{|Y})$ denotes that the valuation of each variable y in Y at state q' agrees with the result of op over y from state q (in particular, if op is side-effect free on some variable y' in Y then $q'(y') = q(y')$). Similarly, we write $q'_{|Y} = q_{|Y}$ iff the valuations of each y in Y at q and q' are identical ($q(y) = q'(y)$ for each y in Y).

Now, we can define each tuple element of $[\![C]\!]$ as follows:

- $U = X \cup \pi$ is the set of variables X and the variable π denoting the current vertex of C. Therefore, $\text{dom}(\pi) = V$ and the initial value of π is v_0,
- Q is the set of states, each state is an interpretation of π and each variable in X,
- q_0 is the initial state, that is the mapping associating π to v_0 (initially the current vertex is v_0) and each variable in X to its initial value,
- $\longrightarrow$ is the set of transitions resulting from mapping each edge e in E to a transition t_e in $\longrightarrow$ as follows. If e connects vertice v_a to v_b, that is $v_a \xrightarrow{e(g_e, op_e)} v_b$, then:

$$q' \in t_e(q) \; iff \begin{cases} (1)\ (q(\pi) = v \wedge q'(\pi) = v') \wedge \\ (2)\ q(g_e) \wedge \\ (3)\ (q'_{|X} = op_e(q_{|X})) \end{cases}$$

Properties Let us get back to the TD in Fig. 2.9. According to the initial values of x_1 and x_2, we can reason on properties such as *progress*. We say that the progress property is satisfied iff for any reachable state in the corresponding TS there exist an enabled transition:

$$\forall q \in Q_r \; \exists t \in \longrightarrow : t(q) \neq \emptyset$$

Now, if $q_0(x_2) - q_0(x_1) = 1$, this property is satisfied. Indeed, op_e (respect. $op_{e'}$) results always in satisfying $g_{e'}$ (respect. g_e). Thus, t_e (respect. $t_{e'}$) is enabled at any reachable state q satisfying $q(\pi) = v_0$ (respect. $q(\pi) = v_1$), regardless of what $q(x_1)$ and $q(x_2)$ evaluate to.
Dually, any initial configuration of x_1 and x_2 such that $q_0(x_2) - q_0(x_1) \neq 1$ results in violating the progress property and thus the TD is said *deadlockable*. For instance, if $q_0(x_2) - q_0(x_1) > 1$, then $g_{e'}$ is not satisfied when reaching a state q such that $q(\pi) = v_1$ and the TD deadlocks at this very state.

This example is useful when we introduce the semantics of DSAAM in 2.4.2.2. Indeed, progress of TDs is an important property in DSAAM systems, which we will discuss and prove in Sect. 2.4.3.

2.4.1.3 Composition of Transition Diagrams

Through shared variables *Syntax:*

The parallel composition of a finite number of TDs, $C_1, \ldots, C_n$, over a set of shared variables, U_s, results in a TS denoted:

$$\{Init\}\left[\underset{i\in 1..n}{||} C_i\right]$$

Where *Init* is the function that defines the initial values of each u in U_s. Thus, *Init* gives for each variable u in U_s its initial value in $dom(u)$.

By means of compositionality, edges of different components are always distinct: if e is an edge in C_i then it cannot be an edge in C_m with $i \neq m$.

Each TD C_i operates a set of local variables, denoted U_i, besides the variables in U_s ($U_i \cap U_s = \emptyset$ and $U_i \cap U_m = \emptyset$ for all indexes $i, m \in 1..n$ with $i \neq m$). Besides, each component C_i has a variable π_i to store its current vertex (Sect. 2.4.1.2). Therefore, the set of variables declared in the TS is:

$$U = U_s \cup \left(\bigcup_{i\in 1..n} U_i\right) \cup \left(\bigcup_{i\in 1..n} \{\pi_i\}\right)$$

Semantics:

Given the parallel composition $\{Init\}\left[\underset{i\in 1..n}{||} C_i\right]$, we can define a TS with the set of variables U that will give the semantics of the system. The meaning of $\{Init\}\left[\underset{i\in 1..n}{||} C_i\right]$ is the TS $\langle U, Q, q_0, \longrightarrow\rangle$ such that:

- $U = U_s \cup (\bigcup_{i\in 1..n} U_i) \cup (\bigcup_{i\in 1..n} \{\pi_i\})$ is the set of variables (see above),
- Q is the set of states, each state is an interpretation of each variable in U,
- q_0 is the initial state, that is the mapping associating (i) π_i to v_0^i (the initial vertex of C_i) and each u in U_i to its initial value, for each C_i and (ii) each u in U_s to its initial value $Init(u)$,
- $\longrightarrow$ is the set of transitions resulting from mapping each edge e in E^i for each component C_i to a transition t_e in $\longrightarrow$ as follows. If e connects vertices v_a^i and v_b^i, that is $v_a^i \xrightarrow{e(g_e, op_e)} v_b^i$, then:

$$q' \in t_e(q) \; iff \begin{cases} (1)\ (q(\pi_i) = v_a^i \wedge q'(\pi_i) = v_b^i) \wedge \\ (2)\ q(g_e) \wedge \\ (3)\ (q'_{|U_i \cup U_s} = op_e(q_{|U_i \cup U_s}) \wedge \\ \quad q'_{|U \setminus (U_i \cup U_s \cup \{\pi_i\})} = q_{|U \setminus (U_i \cup U_s \cup \{\pi_i\})}) \end{cases}$$

Note that the precise definition of the scope of operations *op* in Sect. 2.4.1.2 allows a compact definition of TDs and their compositions as TS. Indeed, op_e cannot operate on any π variable or any variable in U_m with $m \neq i$. Therefore, when a transition $q \xrightarrow{t_e} q'$ is taken, q' must agree with q on (i) all current locations π_m in

all components C_m with $m \neq i$ and (ii) all variables in U_m for all components C_m with $m \neq i$. These variables U_m and π_m are exactly the set $U \backslash (U_i \cup U_s \cup \{\pi_i\}$ on which q and q' must agree (rule (3) in the transitions definition above).

Adding synchronizations The parallel composition seen until now allows only *asynchronous* communication between the TDs participating in it (Sect. 2.4.1.3). We enrich the notion of composition of TDs with *synchronous* communication through edges. Let us consider the same system seen in Sect. 2.4.1.3, resulting from the parallel composition of n TDs:

$$\{Init\}\left[\mathop{||}_{i\in 1..n} C_i\right]$$

Let $\mathcal{E}$ be the set of all edges in the system, that is $\mathcal{E} = \bigcup_{i\in 1..n} E^i$. We define a set of *send* edges E_S and a set of *receive* edges E_R, both disjoint, possibly empty, subsets of $\mathcal{E}$, and therefore we may write $E^S \cup E^R \subseteq \mathcal{E}$ and $E^S \cap E^R = \emptyset$. We denote by the set E^i_X $(X \in \{S, R\})$ the edges of E_X that belong to component C_i, that is $E_X \cap E^i$. For instance, E^1_S is the set of send edges in component C_1. We define now the *matching* function M that maps each send edge e to a set of corresponding receive edges that do not belong to the same component as e. That is $M : E_S \mapsto \mathcal{P}(E_R)$, with $\mathcal{P}(s)$ denoting the powerset of set s, such as the following property is always satisfied for all i ranging from 1 to n:

$$\forall e \in E^i_S \forall e' \in M(e) : e' \notin E^i_R$$

Using these definitions and notations, we enrich the composition of TDs with a *strong pairwise send/receive* synchronization paradigm. First, an enabled edge cannot be taken "alone" if it is a send or a receive edge (in $E_S \cup E_R$), that is the synchronization is done in a *rendezvous* (*strong*) fashion. Second, only a send edge e and a matching receive edge e' can be taken "together", in which case $op_{e,e'}$, denoting executing the operations op_e **then** op'_e, is performed, that is the synchronization is *send/receive*. Finally, the number of synchronized edges taken "together" is always two, that is the synchronization is *pairwise* (if a send edge e is enabled and some of its matching receive edges $M(e)$ are also enabled, only one of them is taken together with e).

Semantics:

The meaning of $\{Init\}\left[\mathop{||}_{i\in 1..n} C_i\right]$, with $E^S \cup E^R \neq \emptyset$, is the TS $\langle U, Q, q_0, \longrightarrow\rangle$ such that:

- U same as in Sect. 2.4.1.3 (semantics),
- Q same as in Sect. 2.4.1.3 (semantics),
- q_0 same as in Sect. 2.4.1.3 (semantics),

- $\longrightarrow$ is the set of transitions $\longrightarrow_e \cup \longrightarrow_s$ such that:
  (i) $\longrightarrow_e$ results from mapping each e in $\mathcal{E}\backslash(E_S \cup E_R)$ to the transition t_e in $\longrightarrow_e$, according to the same rules given for $\longrightarrow$ in Sect. 2.4.1.3.
  (ii) $\longrightarrow_s$ maps each pair of edges $\{e, e'\}$ s.t. $e \in E_S$ and $e' \in M(e)$ to the transition $t_{e,e'}$ in $\longrightarrow_s$ as follows. If e connects vertice v_a^i to v_b^i and e' connects vertice v_k^j to v_l^j, that is $v_a^i \xrightarrow{e(g_e, op_e)} v_b^i$ and $v_k^j \xrightarrow{e'(g_{e'}, op_{e'})} v_l^j$, then:

$$q' \in t_{e,e'}(q) \; iff \begin{cases} (1)\ (q(\pi_i) = v_a^i \wedge q'(\pi_i) = v_b^i \wedge \\ \quad q(\pi_j) = v_k^j \wedge q'(\pi_j) = v_l^j) \wedge \\ (2)\ (q(g_e) \wedge q(g'_e)) \wedge \\ (3)\ (q'_{|U_i \cup U_j \cup U_s} = op_{e,e'}(q_{|U_i \cup U_j \cup U_s}) \wedge \\ \quad q'_{|(\bigcup\limits_{\substack{m \in 1..n \\ m \notin \{i,j\}}} U_m) \cup (\bigcup\limits_{\substack{m \in 1..n \\ m \notin \{i,j\}}} \{\pi_m\})} = q_{|(\bigcup\limits_{\substack{m \in 1..n \\ m \notin \{i,j\}}} U_m) \cup (\bigcup\limits_{\substack{m \in 1..n \\ m \notin \{i,j\}}} \{\pi_m\})}) \end{cases}$$

2.4.2 Formalizing DSAAM

2.4.2.1 Syntax

Inputs An input I is a triple $I = \langle B, T, IF \rangle$ where B is a *buffer* and T a *timestamp variable* (see below). IF is an input interface.
Variables B and T: B is a queue of size n of *message* elements, where each message m_i has a *content* c_i, a *timestamp* $t_i \in \mathbb{N}^*$ and a period $\delta_i \in \mathbb{N}^*$ (where $\mathbb{N}^*$ is the set of non-null naturals). The index i (a natural) is an "identifier" of the message that denotes precedence between them (m_{i+1} is the message following m_i in time, and thus $t_{i+1} = t_i + \delta_i$). We use the following functions:

- empty(B) returns a boolean that evaluates to true iff B is empty, that is the first element of B contains no message,
- full(B) returns a boolean that evaluates to true iff B is full, that is the n^{th} element of B contains a message,
- enqueue(B,m) (with $B \models \neg full(B)$) returns the buffer B with message m inserted in a FIFO fashion,
- first(B) (with $B \models \neg empty(B)$) returns the first element of B,
- dequeue(B) (with $B \models \neg empty(B)$) returns B deprived from its first element.

T stores the timestamp of the message yet to arrive in B (if $empty(B)$) or contained in $first(B)$ (otherwise). That is, it stores the timestamp of the next message to consume (possibly not yet received) by input I. Therefore, T is initialized as the timestamp of the first message m_0 expected to be received in B (retrieved from t_0) and updated, each time a message m_i (returned by $first(B)$) is consumed, to the value $t_i + \delta_i$. Fig. 2.10 shows an example on B (containing one message at some point in time) and how to update T when dequeueing B. This example is convenient in showing how T is updated even when the buffer is empty after dequeueing.

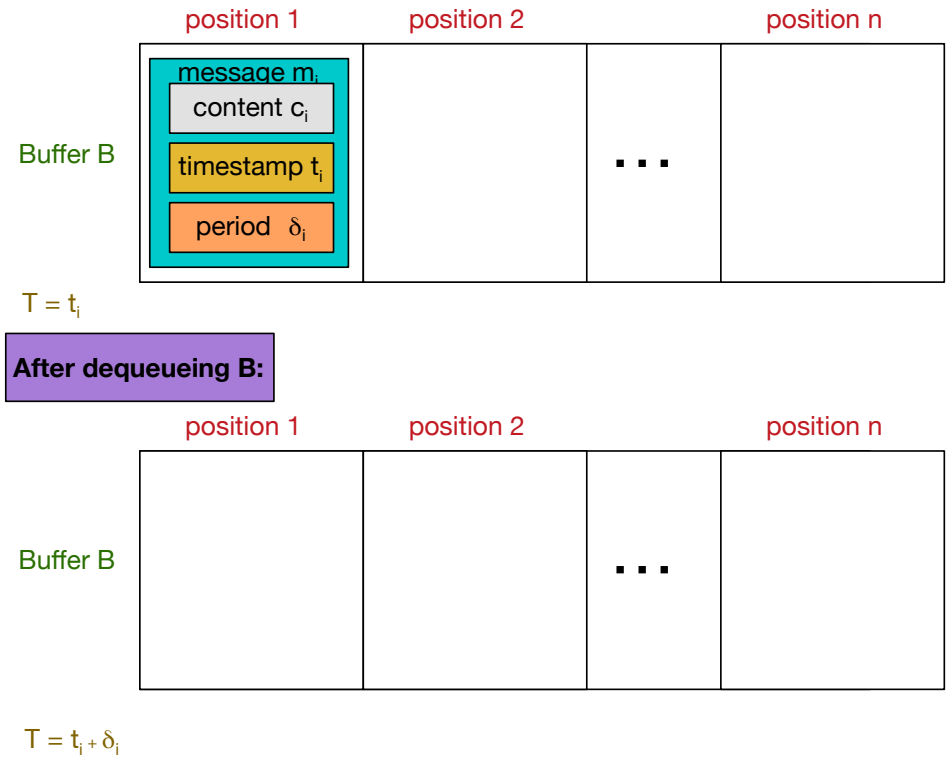


Figure 2.10: Variables B and T (example)

Since the content c relates to the message type and nature, it is implementation dependent and does not intervene in the semantics. Furthermore, t_i (timestamp of the current message m_i) is redundant as long as the initial value of T (t_0) is known. We propose thus a simplified version of B in Fig. 2.11 where each message m_i contains simply the timestamp of the next message to consume, retrieved from the value $t_i + \delta_i$. Fig. 2.11 shows the same example given in Fig. 2.10 with elements of B simplified as explained above. In this example, t_i (the value of T before dequeueing B) is retrieved when the message m_{i-1} was consumed (if $i - 1$ is defined, that is the index $i - 1$ is a natural, *i.e.* $i \in \mathbb{N}^*$) or the initially known t_0 (otherwise).

Outputs An output O is a double $O = \langle S, OF \rangle$ where S is the timestamp of the next message to emit on O and OF is the interface of O.

Nodes A node N is a triple $N = \langle \mathcal{I}, \mathcal{O}, \mathcal{UP} \rangle$ where:

- $\mathcal{I} = \{I_1, ..., I_k\}$ is a set of inputs (Sect. 2.4.2.1),
- $\mathcal{O} = \{O_1, ..., O_l\}$ is a set of outputs (Sect. 2.4.2.1),
- UP is a software blackbox that performs some update operations.

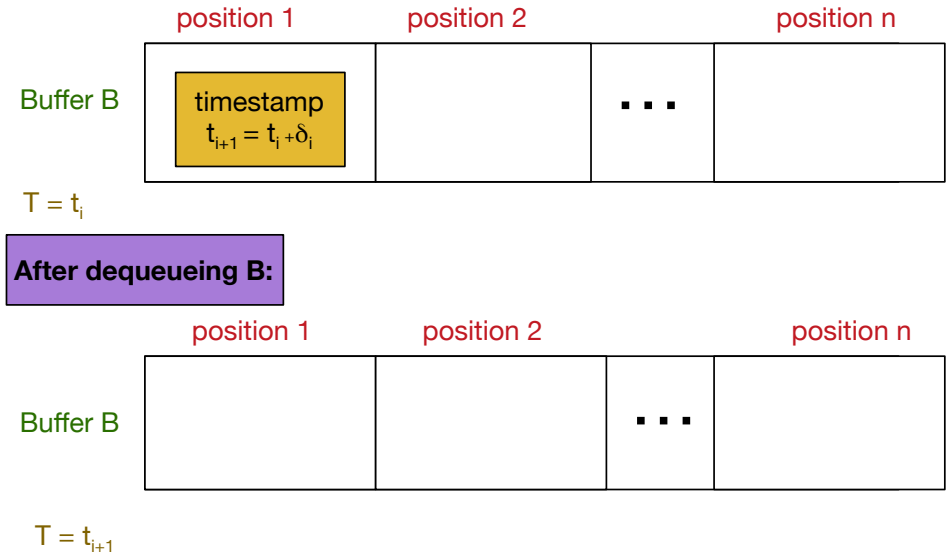


Figure 2.11: Example in Fig.2.10 (simplified syntax)

The operations performed by *UP* pertain to, for instance, processing the message content and performing internal computations accordingly. Most of these operations are thus implementation dependent (relate to the content of the message) and do not take part in the semantics. However, *UP* is in charge of other operations that are relevant to the semantics, mainly the operation that updates the timestamp variables on outputs (Sect. 2.4.2.2).

DSAAM System A DSAAM system is a network of x nodes N_1 ... N_x that communicate together by exchanging messages.
Notations: Before we give the syntactical definition formally, we precise some notations to enhance the readability of the remainder of this paper. We use the superscript $(^i)$ to denote that an input (or an output) belongs to node N_i. Furthermore, to alleviate notations, the elements of an input (resp. output) are uniquely defined through the propagation of the subscripts and superscripts of the input (resp. output) they belong to. That is, $\langle S^i_j, OF^i_j \rangle$ are the elements (timestamp variable and interface) of the output O^i_j (the j^{th} output of node N_i). Similarly, $\langle B^i_j, T^i_j, IF^i_j \rangle$ are the elements (buffer, timestamp variable and interface) of the input I^i_j (the j^{th} input of node N_i). We omit the subscript/superscript when it is unimportant. For instance, when we are at the level of one node, the superscript is irrelevant. Similarly, the subscript does not matter if only the identity of the node the input/output belongs to is important (*e.g.* O^i for any output of node N_i). These notations and simplifications are adopted in the remainder of this paper.
Definition: A DSAAM System $\mathcal{S}$ of x nodes is thus a double $\mathcal{S} = \langle \mathcal{N}, \mathcal{F} \rangle$ where:

- $\mathcal{N} = \{N_1, ..., N_x\}$ is a set of nodes (Sect. 2.4.2.1) and

- $\mathcal{F} : \mathcal{OF} \mapsto \mathcal{P}(\mathcal{IF})$ is the flow function where:
  - $\mathcal{OF} = \bigcup_{i \in 1..x} \left(\bigcup_{j \in 1..|\mathcal{O}^i|} OF_j^i \right)$,
  - $\mathcal{IF} = \bigcup_{i \in 1..x} \left(\bigcup_{j \in 1..|\mathcal{I}^i|} IF_j^i \right)$.

Therefore, we give the syntax of a DSAAM system as a network of nodes where the flow function $\mathcal{F}$ defines for each output in each node a (possibly empty) set of inputs on which it may emit messages, through reconfigurable interfacing. We preserve thus the compositionality and reusability of nodes that may be implemented in different systems (by simply redefining the flow function).

Syntactical restrictions In the rest of this paper, we consider only *well-formed* systems. The DSAAM system defined in Sect. 2.4.2.1 is said well formed if and only if:
(1) All inputs are connected, each to one and only one output. We formalize this requirement as follows:
(i) $\mathcal{IF} \subseteq \bigcup_{i \in 1..x} \left(\bigcup_{j \in 1..|\mathcal{O}^i|} \mathcal{F}(OF_j^i) \right)$
(ii) $\forall \{OF, OF'\} \in \mathcal{P}(\mathcal{OF}) : \mathcal{F}(OF) \cap \mathcal{F}(OF') = \emptyset$
(2) A node does not send messages to itself. Formally:
$\forall OF^i \in \mathcal{OF} : IF^j \in \mathcal{F}(OF^i) \Rightarrow i \neq j$

2.4.2.2 Operational Semantics

Nodes The operational semantics of a node N is given over a *TD* (Sect. 2.4.1.2).
Vertices: $V = \{Wa, Co, Em\}$ is the set of vertices, with Wa (for waiting) being the initial vertex. The vertex Co (resp. Em) denotes that the node is currently *consuming* (resp. *emitting*) a message.
Variables: The *TD* of N accesses the variables given by the syntax of N (Sect. 2.4.2.1), that is the set of inputs $\mathcal{I}$ and the set of outputs $\mathcal{O}$. Consequently, it accesses the variables in these inputs and outputs, that is for each input I_i in $\mathcal{I}$ (resp. each output O_i in $\mathcal{O}$) the variables B_i and T_i, Sect. 2.4.2.1 (resp. the variable S_i, Sect. 2.4.2.1). Additionally, a local variable m is introduced to restore the index of the input/output on which the node will receive/send messages (see below).
Edges: $E = Sw \cup Re$ where Sw is the set of *switch* edges and Re the set of *remain* edges. That it, taking any edges in Sw changes the current vertex of the *TD* where taking any edge in Re does not (self-loop).
(1) $Sw = \{be, ee, bc, ec\}$ such that:

- $Wa \xrightarrow{be(g_{be}, op_{be})} Em$ (begin emitting),
- $Em \xrightarrow{ee(g_{ee}, op_{ee})} Wa$ (end emitting),

- $Wa \xrightarrow{bc(g_{bc},op_{bc})} Co$ (begin consuming),
- $Co \xrightarrow{ec(g_{ec},op_{ec})} Wa$ (end consuming).

(2) $Re = snd \cup recv$ is retrieved from the syntax of N (Sect. 2.4.2.1) as follows:

- $snd = \left(\bigcup_{i \in 1..|\mathcal{O}|} snd_i \right)$ such that $Em \xrightarrow{snd_i(g_{snd_i},op_{snd_i})} Em$ (emit message on output O_i) for each i in $1..|\mathcal{O}|$. This permits emitting messages from the emitting vertex Em,
- $recv = \left(\bigcup_{i \in 1..|\mathcal{I}|} recv_i \right)$ such that $v \xrightarrow{recv_i(g_{recv_i},op_{recv_i})} v$ for each v in V (receive message on input I_i) for each i in $1..|\mathcal{I}|$. This allows receiving a message no matter what the current vertex is.

To define the guard g_e and operations op_e on each edge e, we use the function $rand(s)$ (returns randomly one element of s) and $min(s)$ (returns the smallest element of s), with s being a non empty set. We recall that we do not show guards (resp. operations) that are always true (resp. side-effect free). The guards and operations are then defined as follows:
Edge be:

- g_{be}: $\exists i \in 1..|\mathcal{O}| \mid S_i \leq min \left(\bigcup_{j \in 1..|\mathcal{I}|} T_j \right)$. This is to ensure no emission begins unless the smallest timestamp (Sect. 2.4.2.1) is of an output.
- op_{be}: $m := rand(\{i \in 1..|\mathcal{O}| \mid S_i = min \left(\bigcup_{j \in 1..|\mathcal{O}|} S_j \right)\})$;. This permits to store in m the subscript of an output that may emit a message (having the smallest timestamp).
- op_{ee}: $S_m := up(S_m)$. This updates the timestamp of the next message to emit through the function up, performed by the blackbox UP from the syntactical definition (Sect. 2.4.2.1).

Edge bc:

- g_{bc}: $\exists i \in 1..|\mathcal{I}| \mid T_i < min \left(\bigcup_{j \in 1..|\mathcal{O}|} S_j \right)$. This is to ensure consumption does not begin unless the smallest timestamp (Sect. 2.4.2.1) is of an input. The strict comparison $<$ is to favor outputs in case of equality,
- op_{bc}: $m := rand(\{i \in 1..|\mathcal{I}| \mid T_i = min \left(\bigcup_{j \in 1..|\mathcal{I}|} T_j \right)\})$. This permits to store in m the subscript of an input that may consume a message (having the smallest timestamp).

Edge *ec*:

- g_{ec}: $\neg empty(B_m)$. This is to ensure no message is consumed unless buffer B_m (Sect. 2.4.2.1) is not empty,
- op_{ec}: $T_m := first(B_m)$; $B_m := dequeue(B_m)$. This updates T_m and B_m (Sect. 2.4.2.1) when consuming the message.

Edges *snd*:

g_{snd_i}: $m = i$. This is to ensure emitting on the right output (computed by op_{be}).

Edges *recv*:

g_{recv_i}: $\neg full(B_i)$. This is to ensure reception happens only when buffer B_i (Sect. 2.4.2.1), corresponding to input I_i, is not full.

Example: We give in Fig. 2.12 the TD that gives the operational semantics of a node

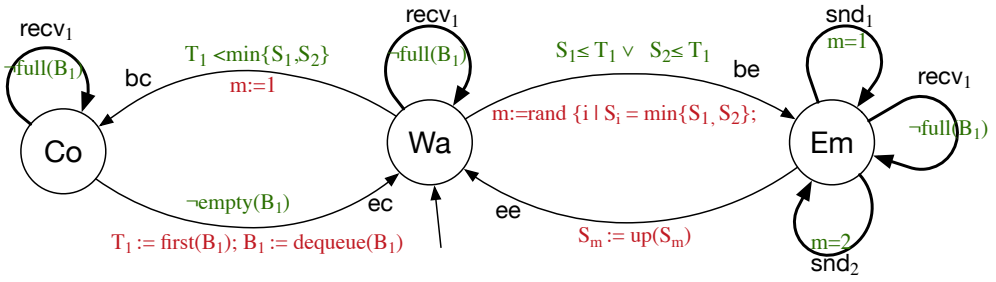


Figure 2.12: A node TD example (one input and two outputs)

with one input and two outputs (resulting from applying the rules in Sect. 2.4.2.2). Guards are in green and operations in red (both simplified when possible, *e.g.* since there is only one input, we know that the only possible value to assign to m when taking *bc* is 1).

Notice how taking any send or receive edge does not actually correspond to sending/receiving a message to/from another node. Indeed, with no operations on any of these edges, taking them is effect-less. This is normal because we are, so far, only reasoning at a "component" level, where the flow is not defined. In the following, we develop compositionally the operational semantics of a DSAAM system, made of multiple nodes, where we constrain the composition of nodes TDs with synchronizations and communication through shared variables, derived exclusively from the syntactical definition of the flow, to correctly reflect messages exchange between nodes.

System A DSAAM system is the parallel composition $\{Init\}\left[\underset{i\in 1..x}{||} N_i\right]$ of x nodes N_i over shared variables (Sect. 2.4.1.3), constrained with synchronizations (Sect. 2.4.1.3). We define in the following the set of shared variables and how the guards and operations are enriched in nodes accordingly, then the synchronizations

and how we derive them from the syntax.

Synchronizations: We derive the synchronization constraints from the syntax of the system (Sect. 2.4.2.1) as follows. The set of send edges E_S (Sect. 2.4.1.3) is the set of all *snd* edges in all nodes N_i, that is $E_S = \bigcup_{i \in 1..x} E_S^i$ with $E_S^i = \bigcup_{j \in 1..|\mathcal{O}^i|} snd_j^i$. Similarly, the set of receive edges E_R is the set of all *recv* edges in all nodes N_i, that is $E_R = \bigcup_{i \in 1..x} E_R^i$ with $E_R^i = \bigcup_{j \in 1..|\mathcal{O}^i|} snd_j^i$. Now, the matching function M (Sect. 2.4.1.3) is simply derived from the flow function $\mathcal{F}$ as defined at the syntactical level (Sect. 2.4.2.1): an edge $recv_k^l$ belongs to the set of the matching edges of an edge snd_j^i iff $IF_k^l \in \mathcal{F}(OF_j^i)$. Therefore, the matching function is fully retrieved from the syntactical definition of the system, and outputs/inputs are prevented from evolving "alone", that is out of the send/receive context defined by the system.
Shared variables: We define the set of shared variables U_s (Sect. 2.4.1.3) as follows: $U_s = \{Msg\} \cup \{\alpha^1, ..., \alpha^i, ..., \alpha^x\}$. *Msg* is the message passing variable where each α^i is used to store and update the inputs on which N_i is currently emitting. Accordingly, we enrich some edges in the nodes (Sect. 2.4.2.2). Only the guards and operations involving shared variables, and the synchronizations imply knowledge of the system (the glue between nodes), which guarantees preserving compositionality when mapping syntactical entities to their operational meanings.
- On each edge be^i (edge *be* in each N_i), the operation $\alpha^i := \mathcal{F}(OF_m^i)$ is added to op_{be^i}. This is to store the inputs on which the selected output, OF_m^i (OF_m of node N_i) will emit messages,
- On each edge snd^i (each edge in *snd* in each N_i), the guard is conjuncted with the expression $\alpha^i \neq \emptyset$ to disable this edge when all inputs have been served. Each snd^i is enriched with the operation $Msg := up(S_m)$ to emit the timestamp of the next message to be sent (current time stamp plus its period), through the shared variable *Msg*,
- On each edge $recv_l^k$ (each edge in *recv* in each N_k), the guard is conjuncted with the expression $IF_l^k \in \alpha^i$ where i is defined statically through the node identity of the only output that serves I_l^k (the only output O_j^i such that $IF_l^k \in \mathcal{F}(OF_j^i)$, Sect. 2.4.2.1 and Sect. 2.4.2.1). This, with some of the operations defined afterwards, prevents the input from being re-served with the same message. Now, $recv_l^k$ edges are enriched with the operations $B_l^k := enqueue(B_l^k, Msg)$ (insert the received message in the buffer) and $\alpha^i := \alpha^i \backslash \{IF_l^k\}$ (I_l^k served, prevent it from being served the same message again),
- Edge ee^i (edge *ee* in each N_i) is guarded with the expression $\alpha^i = \emptyset$ to allow the end of emission only when all inputs corresponding to the chosen output are served.

Timestamps It is worth mentioning that, to start the system in a time-consistent state, input timestamp variables are equal to the output timestamp variables according to the input-output relation defined by the flow function $\mathcal{F}$. That is,

if q_0 is the initial state of the TTS $\{Init\}\left[\underset{i\in 1..x}{||} N_i\right]$, then $q_0(T_j^i) = q_0(S_l^k)$ iff $IF_j^i \in \mathcal{F}(OF_l^k)$.

Example: Let us illustrate with an example. We consider the DSAAM system $\mathcal{S} = \langle \mathcal{N}, \mathcal{F} \rangle$ such that
- $\mathcal{N} = \{N_1, N_2, N_3\}$ with:

- Outputs: $|\mathcal{O}^1| = 2$ and $|\mathcal{O}^2| = |\mathcal{O}^3| = 1$,
- Inputs: $|\mathcal{I}^3| = 3$ and $|\mathcal{I}^2| = |\mathcal{I}^1| = 1$.

- The flow function $\mathcal{F}$:

- $\mathcal{F}(OF_1^1) = \{IF_1^2, IF_1^3\}$,
- $\mathcal{F}(OF_2^1) = \{IF_3^3\}$,
- $\mathcal{F}(OF_1^2) = \{IF_2^3\}$,
- $\mathcal{F}(OF_1^3) = \{IF_1^1\}$.

Therefore, O_1^1 (the first output O_1 of node N_1) emits on I_1^2 (the first and unique input I_1 of node N_2) and I_1^3 (the first input I_1 of node N_3). O_1^2 (the second output O_2 of node N_1) emits on I_3^3 (the third input I_3 of node N_3). O_1^2 (the first and unique output O_1 of node N_2) emits on I_2^3 (the second input I_2 of node N_3). And finally, O_1^3 (the first and unique output O_1 of node N_3) emits on I_1^1 (the first and unique input I_1 of node N_1). We represent these connections in Fig. 2.13.

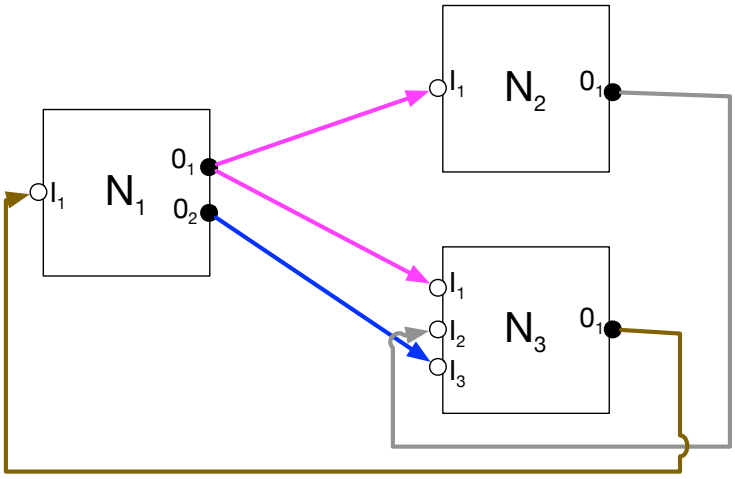


Figure 2.13: DSAAM system example (flow illustration)

Now, using the rules given in Sect. 2.4.2.2 and Sect. 2.4.2.2, we derive the TS representing the operational semantics of $\mathcal{S}$. We show in Fig. 2.14 this TS (in terms of its nodes) with some simplifications:

- Some guards and operations are simplified on an example-dependent basis (*e.g.* the operation of *be* in N_2),
- Superscripts are removed for local variables, edges and locations. For instance, buffer B_1 in operations *op1* is actually the buffer B_1^1, that is the buffer B_1 in node N_1. The superscript here is superfluous since, by means of compositionality, we know that the scope of *op1* is U_s (shared variables) and U_1 (local variables to N_1), and thus *op1* cannot modify buffers not in N_1. Similarly, the edge snd_1 in component N_2 in the figure is actually snd_1^2 and thus different from snd_1 in N_3 (which is snd_1^3),
- When a shared variable is not used in guards/operations in a node, then it is removed from its parameters (*e.g.* N_2 does not need to share α^3 since it does not receive message from N_3).

Matching edges are represented using the same colors (we keep the same color code used in Fig. 2.13).
Let us now explain how this works through an emission scenario, by taking the transition $t_{be_1^1}$ (see mapping edges to transitions in Sect. 2.4.1.3), and assuming the chosen output is O_1^1 ($m = 1$ after taking $t_{be_1^1}$). In this case, the message should be sent to both inputs I_1^2 and I_1^3 ($\alpha^1 = \{IF_1^2, IF_1^3\}$). Now, the enabled transitions in the system involving N_1 depend on the status of buffers B_1^2 and B_1^3. If none of them is full, one of the transition $t_{snd_1^1,recv_1^2}$ or $t_{snd_1^1,recv_1^3}$ is taken. In the former (resp. latter) case, *Msg* is emitted on input I_1^2 (resp. I_1^3) and IF_1^2 (resp. IF_1^3) is removed from α^1. Subsequently, $t_{snd_1^1,recv_1^2}$ (resp. $t_{snd_1^1,recv_1^3}$) is no longer enabled and the only possible transition involving N_1 is $t_{snd_1^1,recv_1^3}$ (resp. $t_{snd_1^1,recv_1^2}$) because all other transitions involving N_1 are disabled (transitions involving snd_1^2 are disabled because $m \neq 2$ and $t_{ee_1^1}$ is disabled because $\alpha^1 \neq \emptyset$). Consequently, the remaining input to serve is delivered *Msg* by taking $t_{snd_1^1,recv_1^3}$ (resp. $t_{snd_1^1,recv_1^2}$) and α^1 becomes empty, which enables ending the emission by taking $t_{ee_1^1}$.

Notice how, since operations in a synchronization transition are executed sequentially and atomically (Sect. 2.4.1.3), *Msg* is always guaranteed to deliver the correct message. Indeed, it is assigned a message and enqueued to the buffer within the same atomic (uninterruptible) sequence of operations (see *e.g.* $t_{snd_1^1,recv_1^3}$), so its value may not be modified by another *snd* edge in between.

2.4.3 Proof of progress

In section 2.3, we laid out a set of properties to ensure the time consistency of the system by forcing nodes to emit and consume messages sequentially with monotonically increasing timestamps and preventing the loss of undelivered messages. These properties hold by construction of the system: guards and operations on edges *bc* and *be* enforce the monotonicity of timestamps, whereas guard on *snd* edge ensures that no message is lost by preventing the emission of messages on an input the buffer of which is full.

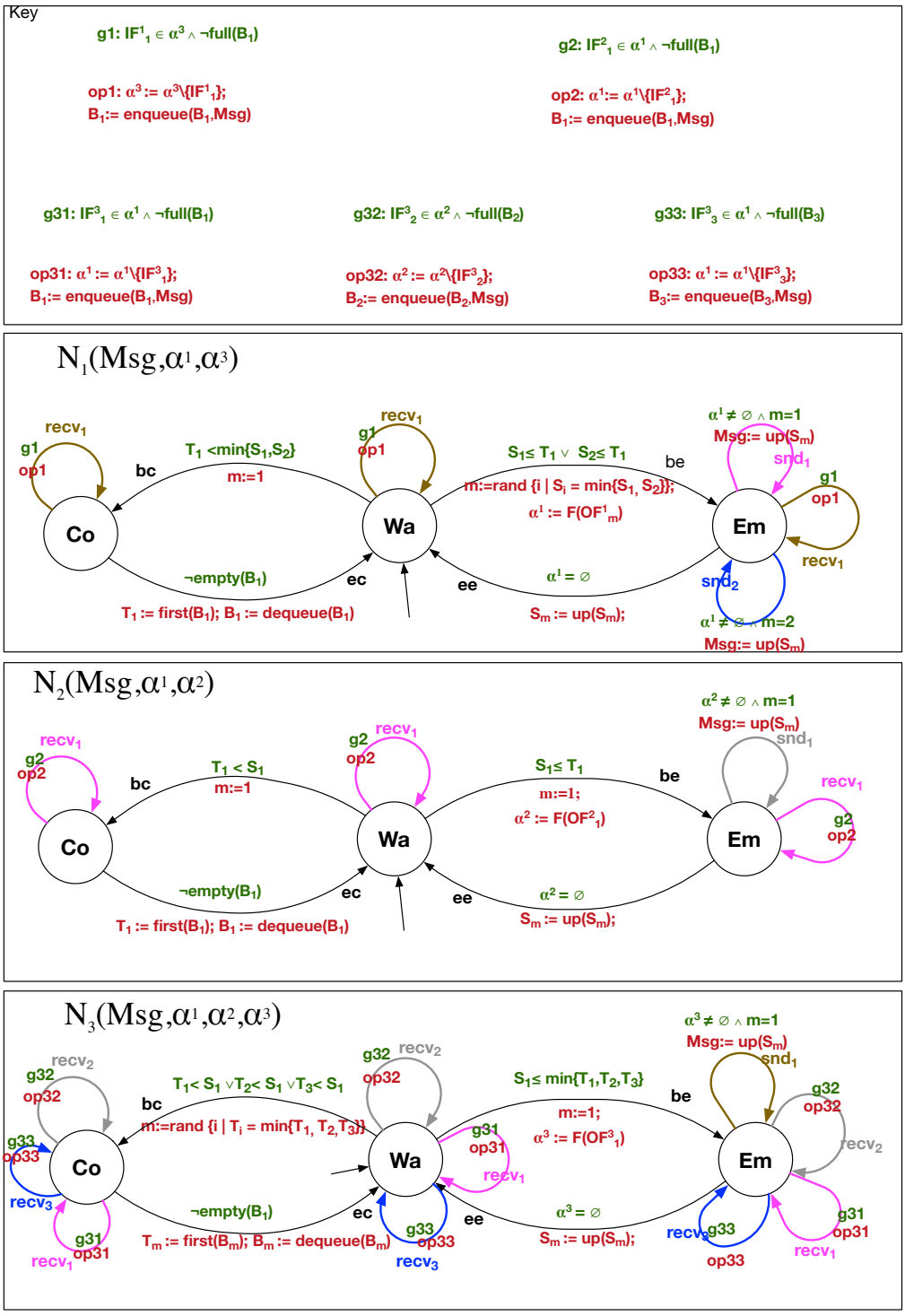


Figure 2.14: DSAAM system example (operational semantics)

Still in such a distributed system with many synchronizations that can form dependency loops, progress (Sect. 2.4.1.2) is a very important property to verify.

We will prove that the progress property holds for all nodes in the system, regardless of the actual topology, the initial valuation of variables and the implementation of the *UP* blackbox.

Before we go further, we need to emphasize that we prove progress for all nodes in the system, which is a stricter property than progress of the system. Indeed, a system may be deadlock free while one of its components is deadlockable (it is sufficient for the system to progress if one of its components does). We start by proving the progress of the system, then derive the progress of each of its nodes accordingly.

2.4.3.1 Progress (system)

In a TS, there is no deadlock if it is always true that the system can take a transition and change state, in other words *there is at least one enabled transition in* $\longrightarrow$ *for each reachable state* $q \in Q_R$.

Theorem 1. *Progress property in a DSAAM system*

$$\forall q \in Q_R, \exists t \in \longrightarrow \wedge\, t(q) \neq \emptyset$$

Proof. We will prove the progress property by contradiction. Let us assume that theorem 1 is false, therefore:

$$\exists q \in Q_R, \forall t \in \longrightarrow, t(q) = \emptyset \tag{2.1}$$

From the definition of $\longrightarrow$ in the DSAAM system (semantics in section 2.4.1.3) we see that only (1) and (2) are relevant for the enabling of the transition, as a suitable q' can always be constructed from the relevant *op* to satisfy (3). Therefore in state q, for all non-synchronizing transitions $t_e \in \longrightarrow_e$ mapping edge $v_a^i \xrightarrow{e(g_e, op_e)} v_b^i$, either:

$\neg(1)$ The corresponding node N_i is in vertex $\pi_i \neq v_b^i$ or

$\neg(2)$ The guard g_e is false: $\neg q(g_e)$.

For the *send* and *receive* edges, either $\neg(1)$ or $\neg(2)$ must hold for the pair of edges that are synchronized.

Let us exhibit a specific node N_i which has the output OF_j^i with minimum S_j^i. Let $S = \bigcup_{i \in 1..x} \left(\bigcup_{j \in 1..|\mathcal{O}^i|} S_j^i \right)$:

$$S_j^i = min(S) \tag{2.2}$$

We will prove the following:

(a) N_i is not in vertex Wa ($\pi_i \neq Wa$)

(b) N_i is not in vertex Co ($\pi_i \neq Co$)

(c) N_i is not in vertex Em ($\pi_i \neq Em$)

which results in a contradiction with $dom(\pi) = \{Wa, Co, Em\}$ and therefore hypothesis 2.1 is false and theorem 1 holds.

(a) $\pi_i \neq Wa$ Let us assume $\pi_i = Wa$, from hypothesis 2.1 we have:

$$\neg q(g_{be^i}) \wedge \neg q(g_{bc^i}) \tag{2.3}$$

However from the well-ordering principle, the following is a tautology:

$$\begin{aligned} q(g_{be^i}) \vee q(g_{bc^i}) := \exists k \in 1..|\mathcal{O}^i| \mid S^i_k \leq min \left(\bigcup_{l \in 1..|\mathcal{I}^i|} T^i_l \right) \vee \\ \exists k \in 1..|\mathcal{I}^i| \mid T^i_k < min \left(\bigcup_{l \in 1..|\mathcal{O}^i|} S^i_l \right) \end{aligned} \tag{2.4}$$

Combining 2.3 and 2.4 we get:

$$\neg q(g_{be^i}) \wedge \neg q(g_{bc^i}) \wedge (q(g_{be^i}) \vee q(g_{bc^i})) \implies \bot \tag{2.5}$$

Therefore $\pi_i \neq Wa$. $\square$

(b) $\pi_i \neq Co$ Let us assume $\pi_i = Co$, from hypothesis 2.1 we have:

$$\neg q(g_{ec^i}) := \neg\neg empty(B^i_m) \iff empty(B^i_m) \tag{2.6}$$

Let us define the timestamp $S^{j'}_{i'}$ of the next message that will be emitted on the flow which links output $OF^{j'}_{i'}$ to the input IF^i_m corresponding to the buffer B^i_m:

$$IF^i_m \in \mathcal{F}(OF^{i'}_{j'}) \tag{2.7}$$

Because the buffer B^i_m is empty, variable T^i_m holding the timestamp of the next message that will be consumed on this input is equal to the timestamp of the next message that will be emitted on the corresponding output. More precisely, we have:

- $T^i_m = S^{i'}_{j'}$ if node $N^{i'}$ is not in vertex Em or if it is in vertex Em but input IF^i_j was not served yet ($IF^i_j \in \alpha^{i'}$).
- $T^i_m = up(S^{i'}_{j'})$ if $N^{i'}$ is in vertex Em but the input has already been served ($IF^i_j \notin \alpha^{i'}$)

Indeed $S^{i'}_{j'}$ is updated only in the operation of edge ee, but the message enqueued has timestamp $Msg := up(S^{i'}_{j'})$ from the definition of $op_{snd^{i'}}$. And using the definition

of $S^i_j = min(S)$ from equation 2.2 we get:

$$S^i_j \leq S^{i'}_{j'} \wedge (T^i_m = S^{i'}_{j'} \vee T^i_m = up(S^{i'}_{j'})) \implies S^i_j \leq T^i_m \tag{2.8}$$

In addition, N_i being in vertex Co, it must have taken the edge bc^i with guard:

$$g_{bc^i} := \exists k \in 1..|\mathcal{I}^i|, T^i_k < min\left(\bigcup_{l \in 1..|\mathcal{O}^i|} S^i_l\right) \tag{2.9}$$

and operation op_{bc^i}:

$$m := rand\left(\left\{k \in 1..|\mathcal{I}^i| \mid T^i_k = min\left(\bigcup_{l \in 1..|\mathcal{I}^i|} T^i_l\right)\right\}\right) \tag{2.10}$$

Equations 2.9 and 2.10 still hold because edge $recv^i_m$ has not been taken since bc^i, otherwise B^i_m would not be empty, as $dequeue(B^i_m)$ is present only on edge ec^i. Combining equations 2.9 and 2.10 we get:

$$S^i_j > T^i_m = min\left(\bigcup_{l \in 1..|\mathcal{I}^i|} T^i_l\right) \tag{2.11}$$

And finally combining 2.8 and 2.11:

$$S^i_j \leq T^i_m \wedge S^i_j > T^i_m \implies \bot \tag{2.12}$$

By contradiction, $\pi_i \neq Co$ □

(c) $\pi_i \neq Em$ Let us assume $\pi_i = Em$. From hypothesis 2.1:

$$\neg q(g_{ee^i}) \implies \alpha^i \neq \emptyset \tag{2.13}$$

That is there remains at least an input $IF^{i'}_{j'} \in \alpha^i$ which has not been served with the message currently being emitted by node N_i. However hypothesis 2.1 holds for the synchronized snd/rcv pair which have to be taken together. Therefore the guard for the corresponding edge $recv^{i'}_{j'}$ in node $N_{i'}$ must be inactive:

$$\neg q(g_{recv^{i'}_{j'}}) \implies full(B^{i'}_{j'}) \tag{2.14}$$

Receiving is not enabled on this input because the buffer is full. Therefore there it at least one message in $B^{i'}_{j'}$. By construction of the up function ensuring that the timestamps of consecutive messages are strictly monotonic increasing, we can state the following about the timestamp variable $T^{i'}_{j'}$ associated to this buffer:

$$T^{i'}_{j'} < S^i_m = S^i_j \tag{2.15}$$

Remember that by definition $S^i_j = min(S)$ and the op of edge ee^i assures that $S^i_m = \min_{k \in 1..|\mathcal{O}^i|}(S^i_k)$ and therefore $op_{ee^i} \implies S^i_m = S^i_j$.

Let us now consider the vertex of $N_{i'}$:

(a') $\pi_{i'} \neq Wa$ as (a) holds for any node.

(b') $\pi_{i'} \neq Em$ as assuming $\pi_{i'} = Em$ implies $S^{i'}_{m'} \leq T^{i'}_{j'} < S^i_j$ from the guard of $be^{i'}$ and by definition of S^i_j as the minimum of S (eq. 2.2). This is in contradiction with 2.15 and therefore $N_{i'}$ is not in vertex Em.

(c') From (a') and (b'), $\pi_{i'} = Co$. However from hypothesis 2.1:

$$\neg q(g_{ec^{i'}}) \implies empty(B^{i'}_{m'}) \tag{2.16}$$

From equation 2.14 we deduce $m \neq j'$, that is node $N_{i'}$ is blocked consuming a message on another input m. Let us define the timestamp S^k_l of the next message that will be emitted on the flow which links output OF^k_l to the input $IF^{i'}_m$ corresponding to the buffer B^i_m:

$$S^k_l \in S, IF^{i'}_{m'} \in \mathcal{F}(OF^k_l) \tag{2.17}$$

Because the buffer $B^{i'}_{m'}$ is empty we have, following the same logic as in equation 2.8:

$$T^{i'}_{m'} = S^k_l \geq S^i_j \tag{2.18}$$

From the operation on the transition $bc^{i'}$ we have:

$$op_{bc^{i'}} \implies T^{i'}_{m'} \leq T^{i'}_{j'} \tag{2.19}$$

which still holds, as the T variables change value only from $op_{ec^{i'}}$ when leaving vertex Co. Combining 2.15 and 2.19 and considering with equation 2.18 we obtain:

$$S^k_l = T^{i'}_{m'} \leq T^{i'}_{j'} < S^i_m \wedge S^k_l \geq S^i_j \implies \bot \tag{2.20}$$

Therefore $\pi_{i'} \neq Co$

From (a'), (b'), (c') and by definition of the domain of π_i we can deduce:

$$\pi_{i'} \notin \{Co, Wa, Em\} \wedge \pi_{i'} \in \{Co, Wa, Em\} \implies \bot \tag{2.21}$$

And by contradiction $\pi_i \neq Em$. □

$(a) \wedge (b) \wedge (c) \implies \bot$ From (a), (b) and (c) and the definition of the domain of π_i we deduce:

$$\pi_i \notin \{Co, Wa, Em\} \wedge \pi_i \in \{Co, Wa, Em\} \implies \bot \tag{2.22}$$

By contradiction, hypothesis 2.1 is false and therefore theorem 1 holds. □

2.4.3.2 Progress (node)

We have proven the progress of the DSAAM System as a whole, therefore there it always at least a transition, or a synchronized pair of transitions, that have their guard active in the system. We will now prove succinctly that the progress of the system in implies the absence of livelock, in the sense that each individual node will eventually progress. For this we will assume that there is at least a node that is unable to progress (all guards are and remain inactive forever), and prove that it will result in the deadlock of the entire system. We present here a schematic version of the proof, in order to avoid describing explicitly the temporal evolution of the system in details.

Let $\{\mathcal{N}, \mathcal{F}\}$ be a DSAAM system and:

- $dead(\mathcal{N}) \subset \mathcal{N}$ be the subset of node that are dead, i.e. will never again have transitions with active guards in their vertex.
- $alive(\mathcal{N}) \subset \mathcal{N} \setminus dead(\mathcal{N})$ be the set of nodes that are alive and never will be dead.

Theorem 2. *Progress of all nodes in a DSAAM system:*

$$alive(\mathcal{N}) = \mathcal{N}$$

Proof. Let us make the hypothesis:

$$\exists N_i \in dead(\mathcal{N}) \tag{2.23}$$

A DSAAM system, $\{\mathcal{N}, \mathcal{F}\}$ forms a graph that is always connected, that is there is a path through flows between all pair of nodes. Hence without loss of generality we may pick a dead node $N_i \in$ such that it is connected through an input or outpout flow to another node which is alive:

$$\exists N_j \in alive(\mathcal{N}), (\exists\, IF_k^i \in \mathcal{F}(OF_l^j)) \vee (\exists\, IF_l^j \in \mathcal{F}(OF_k^i)) \tag{2.24}$$

Because self loops are prohibited, $N_i \neq N_j$, and from the proof of progress of the system we know that $alive(\mathcal{N}) \neq \emptyset$. Let us inspect both cases:

$\boldsymbol{\exists\, IF_k^i \in \mathcal{F}(OF_l^j)}$ Because N_i may not progress, the $recv^i$ transitions never will be enabled. Eventually node N_j will be in vertex *Em* with $IF_k^i \in \alpha^j$. Indeed in each node the input or output with smallest timestamp is selected when leaving vertex *Wa*. Upon reentering this vertex again the timestamp of the corresponding input or output will have been strictly incremented. This means that eventually each input and output will be selected in turn. Because snd^j must be synchronized with $recv^i$ which will never be enabled, it will never be enabled and N_j is dead at this point and is therefore not alive.

$\exists\, \boldsymbol{IF_l^j} \in \boldsymbol{\mathcal{F}(OF_k^i)}$ Because N_i is blocked, the snd^i transition never will be enabled or the node is not in vertex Em. With the same argument as for the output case, eventually node N_j will be in vertex Co waiting on the input $IF_l^j = IF_m^j$ with an empty buffer $empty(B_l^j)$. Therefore guard g_{ec}^j is not enabled. Because Transition $recv_l^j$ cannot be taken, N_j is dead at this point and is therefore not alive.

By contradiction, hypothesis 2.23 is false and therefore $alive(\mathcal{N}) = \mathcal{N}$.

□

2.5 Implementation

A proof of concept of the DSAAM library has been implemented [3] and tested on a toy example (see Appendix 2.A) and a simple UAV simulation scenario. The core implementation is middleware agnostic. An implementation using bare threads communicating in the same process using shared memory allows for testing the core principles of the library without bothering with the complexity of a middleware, as well as benchmarking without the overhead of it. An implementation for the ROS middleware is also provided. In both cases, both the Python and C++ languages have been targeted (see listings in Appendix 2.B for examples of C++ and Python usage).

At the heart of the core library, messages are pushed and popped on thread safe queues, utilizing locks for synchronization. In the bare thread implementation, it allows to directly block on full queues. However for the ROS implementation, message sending is done asynchronously and therefore additional messages are needed in order to deduce the remaining size of the queue. Each time a message is consumed on an input flow, an acknowledgment is sent back to the source node.

2.5.1 The *Precidrone* use case

The *Precidrone* project aims at investigating active perception algorithms to enhance UAV autonomy when mapping a mostly planar environment such as crop fields. The simulation architecture is illustrated in figure 2.15. Two simulators are integrated in a simulation loop, one responsible of simulating the dynamics of the UAV with respect to the environment and control inputs, the other simulating a camera taking pictures of the ground. All nodes interconnect through ROS with DSAAM responsible for time management. The software to be tested is fully integrated in the simulation.

Simulation management is performed using an ad-hoc script, the ROS parameter server and the *roslaunch* utility.

As in this case the processes to be simulated as well as the mapping and planning software perform heavy computations, the overhead of the time management is negligible, and much better parallelism speedup is achieved than in the toy example.

[3] https://redmine.laas.fr/projects/dsaam

Integration of this simulation on a single machine without dedicated time management would have been more difficult but feasible by using the ROS "/time" topic to tune the time advancement to the slowest node, but running multiple simulations concurrently on the same machine would have been impossible. With DSAAM we can run up to three simulations concurrently on an eight core machine to maximize usage of resources when performing batches of simulations.

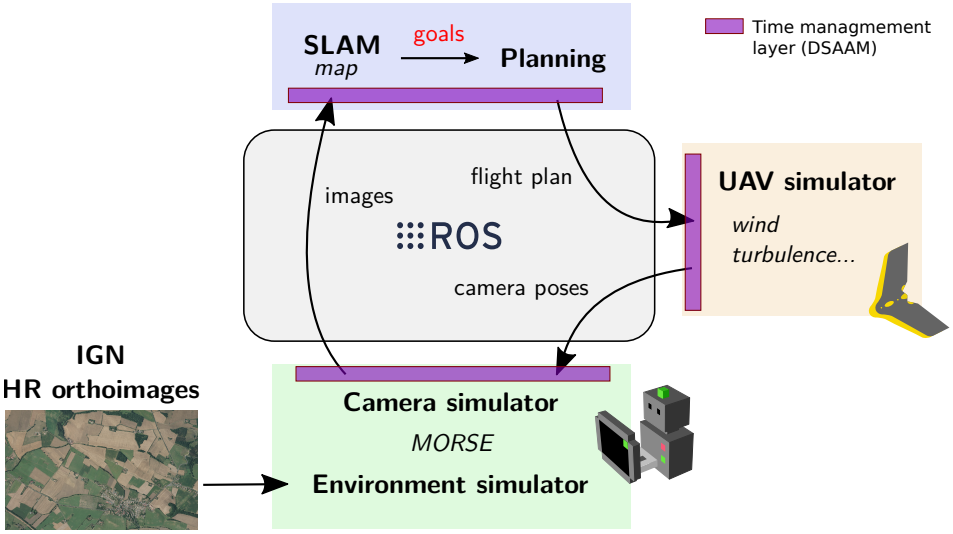


Figure 2.15: Software architecture of an UAV simulation in the context of the *Precidrone* project, using the DSAAM proof of concept for time management. The MORSE and UAV simulator are written in Python, whereas the SLAM and Planning software are written in C++.

2.6 Discussion

2.6.1 Contributions

We proposed a novel decentralized approach for a time management scheme to perform distributed simulations. Based on a set of rules and constraints, it is easy to implement on top of any middleware with a very limited computational overhead, as shown in the proof of concept. Relying on the fact that most simulators are step-based in nature, it only requires each simulator to know precisely, when emitting a message, the timestamp of the next message it will emit. This constraint allows to minimize the number of synchronization messages, while guaranteeing that the system will not encounter any deadlock. However it prevents the integration of event-based simulators.

We emphasized the need of repeatable simulations, even for complex, distributed simulation infrastructures. Repeatability is beneficial from both an engineering point of view to validate algorithms, perform regression testing and speed up the

development process, and a scientific point of view, to ensure repeatability and validity of simulation results irrespective of the computing platform used.

A *formal model* is proposed, which allows to prove the absence of livelocks and deadlocks in the defined distributed system and associated scheme of synchronization between components. Such a model also defines a clear specification of the behavior of the system, and can thus be used to check the conformity of a specific implementation, even across programming languages.

The proposed implementation is easy to use and generic with respect to the communication middleware used, keeping it very lightweight. It is built upon the ROS middleware, but it can very easily be adapted to any other middleware. It respects a clear separation of concerns, which allows to integrate directly simulation layers in an existing ecosystem and lowers the developer's efforts when switching from simulation to deployment.

2.6.2 Future work

The proposed solution is a basis upon which a full-fledged integrated simulation architecture could be built. The following enhancements are in particular relevant:

- Addition of an "observation" flow type in the formal model. For the sake of simplicity, the presented formal model misses the capability of the proof of concept implementation to specify an "observation" flow type. Such a flow type changes the behavior of the system: an output of this type must wait for messages *up to and including timestamp* T being consumed before emitting a message with timestamp T. Indeed DSAAM flows describe variables that are computed from the past state of the world. This maps well with the simulation of *actuators*: from the past state and control input, a new state is predicted. However most *sensors* csn be modeled as instantaneous: e.g. a camera takes a snapshot of the *current* state of the world. Using "observation" flows, *sensors* could be modeled in a more natural way in DSAAM. Relaxing this constraint comes to a cost: the absence of deadlock can not be assessed for any topology, yet it can be proven that as long as there exists no directed circuit of "observation" type flows, progress is guaranteed. Future works include an extended version of the DSAAM formal model with multiple flow types and additional proofs.

- Event-based support. The principal disadvantage of the approach over other time-management solutions is the lack of support for event based simulators, for which there is no guaranty of the time at which the next event will be generated. Support for event-based simulation could however be added with small efforts:

  - Add a new *event* flow type , which will have $\delta = 0$
  - Add a request for such flows from sink to source to deliver the next message or guaranty that no message will arrive before a given timestamp,

as in the *NMR* request in HLA.

– Forbid directed circuits of *event* flow types.

Such flow types resemble to *observation* flow types, and the same no-circuit constraint would be needed in the absence of a central node to prevent deadlocks whilst keeping the synchronization protocol simple. Requirements and proof of progress would be very similar to the one for *observation* flow types (no circuits).

- Real time support. An important feature would be a real time mode support. Having the time management layer switch in a soft constraint mode, the timestamps of the message would be linearly related to the wall clock time, and messages arriving too late discarded – issuing a warning, but without blocking the node blocking on missing messages. Besides, a non trivial issue for real time support is the initialisation of a simulation scenario, which would require all nodes to wait for the readiness of each other and start synchronously. The decentralized implementation of a global barrier of this sort is not trivial.

Finally, our work only pertains to the time management, whereas a wholesome simulation infrastructure would call for *simulation management* services, which we argued should be part of an independent module. Managing simulations life cycle is not an easy architectural task. It should support configuration, deployment and monitoring of the simulation, and even *dynamic reconfigurations*, e.g. to allow joining and leaving of nodes in the midst of a simulation run, and *zoning*, i.e. the automatic subscription to flows of specific types that are close enough according to some metric (geographic, on the same robot, etc...), and unsubscribing when the node leaves the zone.

2.A Implementation benchmarking results

The toy example involves simulation of 2D colored balls or "planets" in a square, closed universe with wrapped coordinates (exiting on one side of the square, an object appears on the other side). Each color of ball has a mass and attracts a restricted set of other balls using a set of rules:

- red attracts red, green and yellow
- green attracts green, blue and yellow
- blue attracts blue and yellow
- yellow attracts red

Position and speed are sent in separate flows. Each node performs the computations for one planet, having all its effectors position and speeds as input (even if only the position is needed for the computations). In the python example, a drawer node is

also available to display the position of the planets, as can be seen in fig. 2.16. An idea of the added complexity of dealing with DSAAM over ROS in the code can be seen in listings 2.1 for Python and 2.2 for C++.

We run the simulation with one planet of each color, four nodes and twelve flows in total (each flow having one flow for position and one for speed). Red updates every four ticks, green every three, blue every two and yellow every tick. Simulation is tested with the bare threads implementation as well as the ROS framework. The bare threads implementation allows to compute the overhead that is due solely to the time management code, and to compare it to the overhead of the ROS middleware, where serialization is needed and communication is performed through sockets. The ROS version simulates only one tenth of the bare threads simulation. There is only one thread processing and emitting messages as well as performing the simulation computations for each node (not counting ROS threads).

Each test is run five times on a computer with an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz CPU, and timing results are compiled in table 2.1. From these results, the overhead due to time management can be computed as shown in table 2.2. In particular in this scenario, the overhead due to time management is very low. Adding user space overhead to the time spent in the kernel, it is approximately one second per million message exchanged or 0.7 second per million queue entries. In our test case, the time spent in the kernel – essentially time spent in the synchronization code taking and releasing locks – is higher than the user space overhead. However the time spent in locks is constant (per queue entry), but computation of the timestamp of the next message to be consumed (or sent) grows logarithmically with the number of in- (or out-) flows.

Using the ROS middleware results, userspace overhead is 95 times higher and kernel overhead is 60 times higher, therefore the overhead due to time management is negligible in this case.

This test case has been devised to maximize resource contention: time spent in the simulation loop is very low, and therefore gains due to parallelism are negligible with respect to a sequential execution, because time management and synchronization code overhead are on par with simulation time, and a lot of time is spent waiting for messages to arrive w.r.t. processing time.

	wall (s)	user (s)	sys(s)	sim (s)	$\#_m$	$\#_m/s$	$\#_q$	$\#_q/s$
Threads								
mean	33.0	50.4	27.8	33.4	41.7M	1.26M	63.3M	1.92M
std	.105	.350	.475	.128	-	4.04k	-	6.15k
ROS								
mean	83.9	165.	169.	3.34	4.17M	49.6k	6.33M	75.5k
std	.758	.722	.388	1.29e-2	-	.450k	-	.683k

Table 2.1: Timing of the planet test with one planet of each color. 'wall' indicates wall clock time, 'user', 'sys' and 'sim' respectively CPU time spent in user space, in the kernel and in the simulation code. $\#_m$ the number of unique exchanged messages, and $\#_q$ the number of queued messages (i.e. counting each message twice if there is two sinks on a flow).

	o.%user	o./M$\#_m$	o./M$\#_q$	sys/M$\#_m$	sys/M$\#_q$
Threads					
mean	33.8%	.409s	.269s	.668s	.469s
std	.525%	6.96e-3s	5.16e-3s	1e14e-2s	7.51e-3s
ROS					
mean	98.0%	39.0s	25.6s	40.6s	26.7s
std	1.17e-2%	.170s	.161s	9.31e-2s	6.13e-2s
ROS/Thr.		95.2×		60.8×	

Table 2.2: Further statistics computed from results shown on table 2.1. 'o.' indicates user space overhead of the time management code, o. = user-sim.

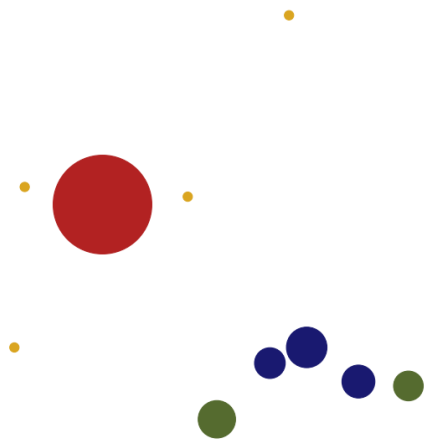


Figure 2.16: A screenshot of the display of the drawer node during a run of the planet toy example using the ROS middleware and a mix of python and C++ nodes. Counting the drawer node, the simulation is composed of eleven nodes and twenty flows.

2.B Listings

```
from geometry_msgs.msg import PointStamped
from dsaam.ros import RosNode, Time

def process_pos(name, message, next_message_time):
    #process message
    pass


node = RosNode('B', start_time=Time(0),
               default_qsize=3)

my_period=Time(4)
node.setup_publisher('/B/position', PointStamped,
                     dt=my_period,
                     subscribers=['C'])

node.setup_subscriber('/A/position', PointStamped,
                      callback=process_pos,
                      dt=Time(2))
(...)

node.init_ros()
while true:
    nextAt = node.next()
    while nextAt >= node.t + my_period:
        m = simulation_step()
        node.send('/C/position', m, node.t + my_period)
        node.step(node.t + my_period)
.
```

Listing 2.1: Example showing the DSAAM API using the ROS middleware in Python. In addition to usual ROS parameters such as topic name and message type, it requires the period of each flow as well as the list of subscribers for the publisher. This last parameter is used to make the node wait for all subscribers to subscribe before starting the simulation (in the *node.init_ros()* call).

```
#include<dsaam/ros/ros_node.hpp>
#include<geometry_msgs/PointStamped.h>

using shared_cptr_t = dsaam::ros::shared_cptr_t;
using PointStamped = ros::geometry_msgs::PointStamped
void process(const shared_cptr_t<PointStamped> &m,
             const ros::Time &next)
{
    // (...)
}
using point_callback_t = \
    dsaam::ros::function_type<decltype(&process)>;

using RosNode = dsaam::Node<dsaam::ros::RosTransport>;

node = RosNode("B", ros::Time(0), default_qsize=3);

my_period = ros::Time(4)
node.setup_publisher<PointStamped>(
    "/B/position",
    ros::Time(0), //timestamp of the first message
    my_period,
    {'A'}); //subscribers

node.setup_subscriber<PointStamped>(
    "A", "/A/position", "B", //A --/A/position--> B
    ros::Time(0), ros::Time(2), //start time and period
    point_callback_t(&process)); //callback

auto send_B_pos = node.send_callback("/B/position");

// (...)

node.init_ros()
while(true)
{
   node.next();
   while(node.nextAt() <= node.time() + my_period)
   {
       m = simulation_step();
       send_B_pos(m);
       node.stepTime(node.time() + my_period)
   }
}
```

Listing 2.2: Example showing the DSAAM API using the ROS middleware in C++. Except from the declaration of types and a few C++ quirks, it is very similar to the Python API.

Bibliography

[Benjamin 2010] Michael R Benjamin, Henrik Schmidt, Paul M Newman and John J Leonard. *Nested autonomy for unmanned marine vehicles with MOOS-IvP*. Journal of Field Robotics, vol. 27, no. 6, pages 834–875, 2010.

[Brisset 2006] Pascal Brisset, Antoine Drouin, Michel Gorraz, Pierre-Selim Huard and Jeremy Tyler. *The paparazzi solution*. In MAV 2006, 2nd US-European competition and workshop on micro air vehicles, pages pp–xxxx, 2006.

[Brito 2015] Alisson V. Brito, Harald Bucher, Helder Oliveira, Luis Feliphe S. Costa, Oliver Sander, Elmar U.K. Melcher and Juergen Becker. *A Distributed Simulation Platform Using HLA for Complex Embedded Systems Design*. In 2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pages 195–202, Chengdu, October 2015. IEEE.

[Bruyninckx 2001] Herman Bruyninckx. *Open robot control software: the OROCOS project*. In Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, volume 3, pages 2523–2528. IEEE, 2001.

[Bryant 1977] Randal Everitt Bryant. *Simulation of Packet Communication Architecture Computer Systems*. Technical Report, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1977.

[Chandy 1979] K. Mani Chandy and Jayadev Misra. *Distributed simulation: A case study in design and verification of distributed programs*. IEEE Transactions on software engineering, no. 5, pages 440–452, 1979.

[Chandy 1989] K Chandy and R Sherman. *The Conditional-Event Approach to Distributed Simulation*. July 1989.

[Chaudron 2011] Jean-Baptiste Chaudron, Martin Adelantado, Eric Noulard and Pierre Siron. *HLA High Performance and Real-Time Simulation Studies with CERTI*. 2011.

[Degroote 2015] Arnaud Degroote, Pierrick Koch and Simon Lacroix. *Integrating Realistic Simulation Engines within the MORSE Framework*. In Workshop on Rapid and Repeatable Robot Simulation (R4 SIM), at Robotics: Science and Systems, 2015.

[dis 2012] *IEEE Standard for Distributed Interactive Simulation–Application Protocols*. IEEE Std 1278.1-2012 (Revision of IEEE Std 1278.1-1995), pages 1–747, Dec 2012.

[Drummond 2009] Dr Chris Drummond. *Replicability Is Not Reproducibility: Nor Is It Good Science*. In Proc. of the Evaluation Methods for Machine Learning Workshop, 26th ICML, Montreal (Canada), June 2009.

[Fujimoto 2015] R. Fujimoto. *Parallel and Distributed Simulation.* In 2015 Winter Simulation Conference (WSC), pages 45–59, December 2015.

[Gervais 2012] C. Gervais, J. B. Chaudron, P. Siron, R. Leconte and D. Saussié. *Real-Time Distributed Aircraft Simulation through HLA.* In 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pages 251–254, October 2012.

[HLA 2010] *IEEE Std 1516.1^{TM}-2010, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification.* page 378, 2010.

[Jafer 2013] Shafagh Jafer, Qi Liu and Gabriel Wainer. *Synchronization Methods in Parallel and Distributed Discrete-Event Simulation.* Simulation Modelling Practice and Theory, vol. 30, pages 54–73, January 2013.

[Jefferson 1982] David Jefferson and Henry Sowizral. *Fast Concurrent Simulation Using the Time Warp Mechanism. Part I. Local Control.* Technical Report, RAND CORP SANTA MONICA CA, 1982.

[Jefferson 1987] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M. Diloreto, D. Jefferson, B. Beckman, F. Wieland, L. Blume and M. Diloreto. *Time Warp Operating System.* In ACM SIGOPS Operating Systems Review, volume 21, pages 77–93. ACM, January 1987.

[Mallet 2010] Anthony Mallet, Cédric Pasteur, Matthieu Herrb, Séverin Lemaignan and Félix Ingrand. *GenoM3: Building middleware-independent robotic components.* In Robotics and Automation (ICRA), 2010 IEEE International Conference on, pages 4627–4632. IEEE, 2010.

[Metta 2006] Giorgio Metta, Paul Fitzpatrick and Lorenzo Natale. *YARP: yet another robot platform.* International Journal of Advanced Robotic Systems, vol. 3, no. 1, page 8, 2006.

[Perry 2004] Alexander R Perry. *The flightgear flight simulator.* In Proceedings of the USENIX Annual Technical Conference, 2004.

[Quigley 2009] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler and Andrew Y Ng. *ROS: an open-source Robot Operating System.* In ICRA workshop on open source software, volume 3, page 5. Kobe, Japan, 2009.

Chapter 3

Learning error models for graph SLAM

Contents

3.1 Introduction

3.1.1 Context

The use of UAVs for observation purposes has pervaded a large number of application contexts, from core mapping use cases to archaeology or atmosphere sciences. Most of the observation applications for UAVs share the common need to map as well as possible a predefined area or volume with on-board sensors, and can be broken down into three stages:

- A sensor coverage pattern or flight plan is defined on the basis of the area to be mapped,
- The UAV automatically follows the defined flight plan, using GPS as the primary localization mean,
- Upon flight completion, the collected data is post-processed to produce the map.

In such a scheme, the data acquisition process is passive, the UAV adaptation consisting only in adapting the flight control to track the planned trajectories. As a consequence, there is no insurance of exhaustivity nor quality of the collected data. Indeed, during the flight, wind gusts may significantly perturb the trajectory execution, cloud casting shadows, fog or motion blur may degrade the quality of sensor data: all these events may yield maps of low quality, or even with un-mapped areas, which can only be clearly assessed after the data post-processing stage. To prevent such failures, the usual approach is to define very conservative coverage patterns, *e.g.* to ensure that small deviations of the trajectory will have a minimal impact on the map quality, or that observed areas are perceived several times. This comes at the cost of gathering redundant data and prolonging mission duration. In addition, this restricts the UAVs to fly only under optimal weather conditions, so as to make sure trajectory control will perform well.

A natural evolution is to deploy *an active perception scheme*, that is using the sensory outputs of the UAV as inputs to drive the flight so as to optimize mission time while ensuring adequate coverage and mapping of the area – in other words, turning the automatic execution of pre-defined mapping missions into an autonomous instance of the sense / plan / act paradigm, in which planning is driven by perception objectives. This implies requirements on two different concerns:

- *perception*: not only the map building must run on-line, but it should allow a precise assessment of the map quality,
- *planning*: one should be able to replan online the coverage of the area, which involves a predictive model of the mapping process that explicits the information content brought by new observations.

Of course numerous instances of such active schemes have been proposed in the robotics community. For instance, they are required by definition for *exploration tasks* where the structure of the environment is not a priori known. For tasks which consists in mapping or observing continuous surfaces or volumes, the main contributions relate to *adaptive sampling* [Rahimi 2004, Hollinger 2014], which consists in optimizing the number of measurements to assess a spatially continuous processes (*e.g.* marine processes [Das 2015] or atmosphere phenomena [Reymann 2018, Lawrance 2011]).

However, there are few contributions on the development of adaptive schemes in the context of precision agriculture, where using UAVs to monitor large crop fields

has become a viable commercial application[1]. Here the application of a passive scheme forces the operator to define redundant coverage patterns with large longitudinal and lateral ground overlaps of the camera footprint, which yields longer missions and larger datasets to post-process. No adaptive sampling scheme can be applied in this context, as the missions require a full coverage of the surfaces of interest.

3.1.2 Problem statement and contribution

We are aiming at developing an active mapping scheme in the context of large crop monitoring missions, or more generally for surface coverage missions with UAVs. Planning observation trajectories that balance map quality with mission time requires both the ability to compute a world model online and to estimate an associated error model from which the information content of future trajectories can be assessed.

The solution of choice for mapping crops is to feed a bundle adjustment (BA) technique with images acquired by an on-board multi-spectral camera: this generates very high precision maps, but requires heavy post-processing. Progresses in visual SLAM, and in particular in monocular graph SLAM approaches [Younes 2017, Mur-Artal 2015], let seriously consider the possibility to achieve online mapping with a precision comparable to off-line BA techniques. Relying on such a mapping technique, the problem at hand becomes an *active SLAM* problem, for which both an estimation and predictive error models are keys. Yet, the definition of such models remains a difficult problem, especially for graph SLAM approaches, where the extraction of a precise information matrix from the result of the optimization process is not straightforward.

This chapter introduces an approach to learn a full SLAM error model. Building on the seminal work of Kasra Khossousi [Khosoussi 2017], exploiting the graphical nature of SLAM and spectral decomposition, we propose an architecture to learn relative error metrics between any pair keyframes – hence the adjective "full" of the error model. The input of the learning architecture is not directly the data itself, as is usual with deep learning techniques, but instead signatures of the structure of the covisibility graph maintained by the SLAM algorithm, as well as features computed from statistics on each edge of the current graph. This error model also yields a prediction ability: new observations features are inferred by a regression technique, from which the new covariance matrices can be predicted.

3.1.3 Outline

After a brief state of the art, section 3.2 introduces the work of K. Khossoussi. The proposed approach is introduced in section 3.3, and its implementation is depicted in section 3.4. Results are analyzed in section 3.5, and a discussion concludes the chapter.

[1] A noticeable exception is the work of Lliam Paul [Paull 2014]

3.2 Related work

There are a great number of formulations of the SLAM problem, depending on the sensors and measurements available (single camera, stereoscopic cameras, LIDAR, odometry, IMU), on the family of techniques used to solve it (filtering, MAP estimation, deep learning), and on the data association technique used to compare measurements (landmark based, dense methods).

The original focus of SLAM techniques is to provide the robot with an online reconstruction of the robots trajectory along with a map of the environment, which structure is in most cases devoted to localization. While SLAM solutions based on Bayesian filtering explicitly manage the pose and map errors, the more recent (and more robust) approaches that exploit optimization techniques have dimmed the need for an accurate uncertainty model, aiming at solutions that are real time, avoid catastrophic failure as much as possible, and with as low error as possible on the trajectory reconstruction. A recent focus is the production of denser maps for navigation and planning [Whelan 2016, Gao 2018]. In these schemes the quality of the reconstruction is not available, and the only queries one can make on the produced maps are binary (is this location mapped or not?).

In *active SLAM*, being able to quantify robot and map uncertainties and predict the utility of new observation sequences is central. In the recent and comprehensive survey of SLAM [Cadena 2016], section VIII. introduces and discusses briefly active SLAM, which serves as the basis of our analysis. Active SLAM is a decision making problem where one tries to balance *exploration* of unmapped areas with *exploitation*, *i.e.* revisiting locations to improve map quality and robot pose precision. A number of classical techniques have been applied to this problem, the most common being Theory of Optimal Experimental Design (TOED) [Carrillo 2012] and information theoretic approaches [Carrillo 2015].

Note that most contributions to active SLAM stem from the ground robotics community, where the robot usually evolves in a structured environment, often indoors. Closer to our concern of coverage path planning, [Kim 2015] develop an active SLAM algorithm for coverage path planning applied to an underwater autonomous vehicle. Navigation uncertainty is computed using the existing map, the planning algorithm proposing new links. Gaussian Process regression is used to infer the visual saliency of unexplored locations which in turn is used to estimate the probability of making successful landmark observations. This enables to estimate the pose uncertainty along the path using the information matrix of the poses, odometry constraints and expected camera measurements. The objective function balances pose uncertainty along the path with a term dependent of area coverage. Thus the robot is able to cover the area while keeping the pose uncertainty under a predefined threshold, however reducing the map uncertainty is not an direct objective of the algorithm.

All active SLAM approaches have in common the need to compute the utility of an action, which has to rely on an uncertainty model of the robot pose and of the environment. Fortunately, both Kalman filter and graph optimization

based approaches share the property of being probabilistic frameworks and therefore maintain a representation of uncertainty, in the form of a covariance or information matrix. Computing the utility of new observations is done by using the current model as prior, and predicting the effect on the model by computing the posterior distribution after the integration of the new observations. In general, this is an intractable problem and therefore several approximations have to be made, such as assuming isotropic Gaussian noise and using a simple fixed-variance model for unknown locations.

Even if computing the posterior would be an easy problem, there would remain numerous issues that hinder the precise computation of uncertainties. Indeed the uncertainties produced by the SLAM algorithms in real world scenarios are often optimistic (over confident): this is due to imprecision and simplifications used in the observation models, such as:

- Correlations between observations of the same landmark. Consecutive observations are almost always assumed independent, it is however a classical result that this is not the case (*e.g.* due to sensor calibration biases), the posterior mean of repeated observations converging to a fixed bias.

- Wrong matches (outliers) in landmark-based SLAM. If recent developments seem to indicate that graph optimization based techniques are quite robust to a small amount of outliers for estimating the posterior mean, outliers necessarily introduce overconfidence in the variance.

Some of these issues have been addressed in the literature, such as in [Zhu 2017] for EKF, where adaptively "inflating" the prior uncertainty matrices (which requires hand-tuning of some parameters) as well as replacing EKF updates with a covariance intersection technique that can take as input a correlation factor between measurements leads to more conservative estimates. Accounting for the presence of outliers remains however unsolved, and most authors assume that an adequate detection and removal of outliers process ensures that very few remain in the model.

Lastly it is worth to mention recent developments in deep learning, such as [Kendall 2016] where the authors do deep regression of camera pose for relocalization using Bayesian convolutional networks. Uncertainty is estimated using Monte Carlo sampling. In [Wang 2018], probabilistic visual odometry is performed using deep neural networks, the network producing both the mean and the variance of the estimate. In both works uncertainty estimate seem to be less overconfident than traditional Bayesian inference techniques. If deep learning has been successfully applied to relocalization and visual odometry and is able to produce conservative uncertainty estimates, such works are not straightforwardly transferable to SLAM where one has to be able to close loops, i.e. reuse old data and revise estimates, propagating the information across the whole observation graph.

There is a more subtle issue, arising from the nature of SLAM. In absence of fusion with global positioning, localization is done in a local frame. Essentially the position of each element is optimized relatively to that of the other ones.

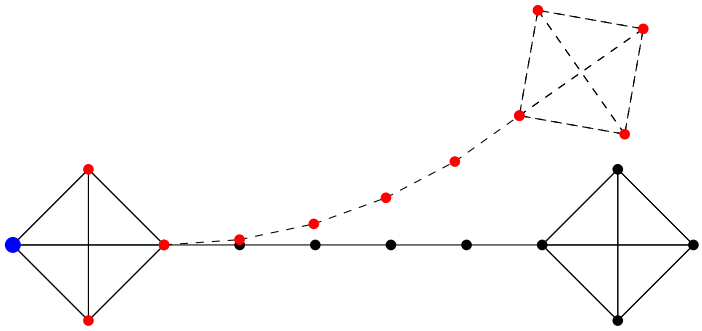


Figure 3.1: A pose graph of the robot's path, edges representing relative motion measurements between poses. ● marks the reference frame, ● SLAM estimated poses and ● the pose ground truth. In the absence of large loop closing with the reference frame, the SLAM estimate accumulates drift. The square's shape at the end of the path remains locally consistent.

And because error accumulates, it can lead to scenarios such as described in figure 3.1, where the robot has described a path resembling a barbell. Two areas are very well connected (i.e. the graph contains many local cycles), but there is only one path between these two areas. In the absence of loop closing, their position relative to each other exhibits a large drift. In this instance, adding a single observation is not enough to lower the pose uncertainty significantly, large loops have to be closed to minimize drift. Being far apart, a long path has to be planned, with very little gain until the loop finally closes. Because the planning algorithm needs to sample in the observation space, and because the complexity of the planning problem grows exponentially with the number of future observations planned, heuristics are needed to guide the planning so as to cope with such situations. It is evident that by closing a loop between high and a low covariance locations will lower the drift and greatly improve the SLAM estimate. This could be used as such an heuristic. It has to be noted that there is no reason to give preference to one local frame or the other, this choice is purely arbitrary.

Another scenario, described in figure 3.2, clearly exhibits the limits of an absolute uncertainty measure for SLAM. Here the robot's path has described a Y shape, with the reference frame being at the bottom of the Y. The covariance is highest at the top of the branches. Therefore an active SLAM algorithm seeking to reduce this covariance and guided by the aforementioned heuristic would propose paths closing the loop between the top and the bottom of the Y. However as previously stated, the choice of reference is arbitrary: choosing as anchor the middle of the Y exhibits another equivalent potential loop closure, by linking the two branches. Therefore the choice of anchor is very important and will greatly impact the covariance of each robot pose, as well as hide information in the covariance matrix. One possibility to avoid this is to estimate relative covariance between each pair of nodes, thus exposing the full information to the planning

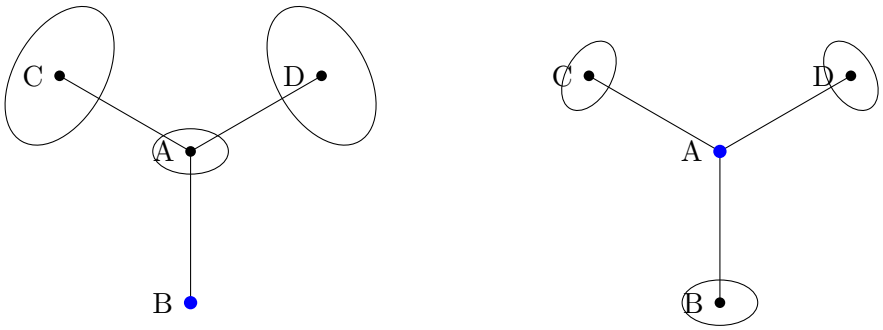


Figure 3.2: The Y example, illustrating the effect of the choice of reference frame, marked ●, on the covariance estimate. Closing loops between B-C or C-D does not yield the same information gain in the left case, whereas it does in the right case

algorithm. To our knowledge, this is a costly operation and would need $\mathcal{O}(n)$ matrix inversions, for a total cost in $\mathcal{O}(n^4)$ using the straightforward solution of updating the information matrix and performing the inversion for each anchoring.

These figures illustrate the strong influence of the topology of the observation graph on the quality of the SLAM estimate. The usage of purely topological information to perform active SLAM has also been explored: in [Kim 2013] the authors use topological information on frontier-based exploration algorithms to efficiently guide a group of robots. Probabilistic reasoning on topological maps has been introduced in [Ranganathan 2004] and shown to be helpful to maintain the global consistency of the graph and close loops (see also [Ranganathan 2011]). Active SLAM planning on topological maps has been developed [Mu 2016], reasoning on the entropy of the produced topological graph. More recently, [Kitanov 2018] proposes to plan on reasoning on the topological properties of the factor graph produced by the SLAM algorithm. It exploits the recent findings of [Khosoussi 2014, Khosoussi 2015], that state that the topological properties of the factor graph is determinant of the accuracy of the estimation. It therefore completely avoids any Bayesian reasoning, the developed criteria depending only of the degree of the nodes is very easy to compute.

3.2.1 Relationship between pose graph topology and uncertainty

The seminal work of [Khosoussi 2014, Khosoussi 2015] explores the relationship between pose graph topology and the uncertainty estimate recovered from the information matrix of the maximum likelihood estimate in the 2D pose graph SLAM problem. The first paper [Khosoussi 2014] proposes three indicators relating these two concerns, which we briefly outline here. The author reasons on the pose graph, with weight on edges taken as the measurement translational and rotational covariance.

Ratio of Costs ⇔ Average Node Degree A function of the average node degree of the pose graph can be used to estimate the ratio between the value of the cost function at the SLAM estimate and the value of the same function at the ground truth.

Diameter of Confidence Ellipsoid ⇔ Algebraic Connectivity The algebraic connectivity, which is the second-largest eigenvalue of the weighted Laplacian matrix of the graph is a lower bound of the diameter of the largest confidence ellipsoid.

Volume of Confidence Ellipsoid ⇔ Number of Spanning Trees The weighted number of spanning trees is linked to the volume of confidence ellipsoids through the determinant of the Fisher information matrix. In particular, it is shown in [Khosoussi 2015] that the determinant converges to a pure function of the weighted number of spanning trees when a parameter δ converges to zero. This parameter depends only on the degree of the nodes, the sensing range and the precision of translational measurements. These results are proven in the case of planar SLAM, and empirical evidence on publicly available datasets seem to indicate a linear relationship between the log determinant of the fisher information matrix and the tree connectivity for 3D pose graph SLAM.

Chapter 5 of [Khosoussi 2017] exploits the "tree connectivity" criterion as defined as the normalized weighted number of spanning trees to solve the *edge selection problem*: adding (or removing) edges so as to maximize tree connectivity of the resulting graph. The optimal selection of the 1-edge selection problem, is shown to be selecting the edge between nodes exhibiting the highest effective resistance between them.

The results of this work are highly effective when dealing with pose graph SLAM with an accurate error model on the relative error measurements. However in the case of feature based SLAM with indirect measurements, such as in monocular SLAM, this information is not directly available. One could work with the full graph incorporating both robot poses and landmarks, but this would result in graphs several orders of magnitude larger. Furthermore, uncertainty estimates are often heavily biased as discussed previously.

3.3 Learning the error model from SLAM topology

In our surface mapping context, we exploit the monocular camera SLAM problem defined as a landmark-based graphical model formulation solved by MAP estimation. The field is now mature enough that there exists open sources implementations that perform fairly well in a variety of situations – we rely on the work of [Mur-Artal 2017].

In this SLAM formulation, landmarks are key points features that are extracted and tracked in the images, and a keyframe selection process select the images which

position constitute nodes in the factor graph. The estimation of the locations of landmarks and keyframes is posed as a non-linear least square problem, solved using Gauss-Newton or similar methods. Note that the Hessian of the constraint graph at the optimum is also (an approximation) of the information matrix of the problem.

3.3.1 The covisibility pose graph

The graph-based optimization formulation of SLAM builds a factor graph that encodes the constraints imposed by observations on hidden variables. Each node represents a hidden variable, edges are called factors, and encode an error function between a measurement and the expected observation given the estimated values of the hidden variables. The formulation assumes a Gaussian error model, each measurement (and therefore factor) has an associated covariance, which, along with the error function, allows to compute the joint likelihood of the hidden variables and the observations. The optimization process then maximizes the product of all joint likelihoods.

In the monocular SLAM problem, and in the absence of odometry measurements, there is no direct observation of the relative pose of two robot locations. The landmark based graphical model formulation of monocular SLAM builds a bipartite factor graph. Nodes either represent keyframe poses or landmark positions (figure 3.3), and factors encode the reprojection error on the image of landmarks, given the detected position of the landmark in the image, the estimated pose of the camera and the estimated position of the landmark.

We call *covisibility graph* the undirected graph G_α derived from the factor graph, keeping only the camera pose nodes and adding edges between two nodes if they share covisible landmarks (figure 3.3). Each edge (i, j) has a weight α_{ij}, which represents the tightness of the constraints linking i and j camera poses, imposed by the common landmark observations.

Such a covisibility graph is for example introduced in [Mur-Artal 2015], with weights simply being the number of covisible landmarks. It is used in order to reason on the topology of the model and prune redundant observations, while keeping the global topology intact. The authors also use it to select edges involved in local optimizations when adding a new keyframe, and to define the "essential graph" on which optimization is performed when closing loops.

3.3.2 The resistance distance

Relying on and expanding Khossoussi's work, we aim at instantiating the covisibility graph G_α so as to derive uncertainty estimates between camera poses. In order to exploit this graph to plan further observations, the uncertainty estimates must be as close as possible to the actual error[2], and the graph must yield the possibility to assess the impact of future observations.

[2] As discussed above, the covariance matrix recovered from monocular SLAM may not be a good estimate of the ground truth error.

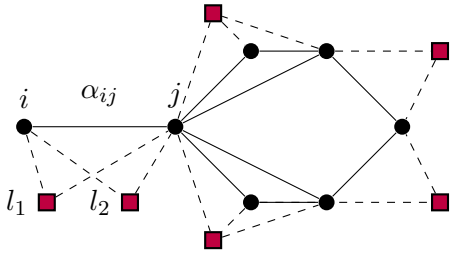


Figure 3.3: A factor graph and the associated covisibility graph. ● correspond to camera poses, and ■ to landmark positions. All edges define the factor graph, whereas only solid edges define the covisibility graph (landmarks are not part of the covisibility graph). The relative pose constraints between i and j are defined by the factors linking covisible landmarks l_1 and l_2 with i and j: they are abstracted by the weight α_{ij} encoding the tightness of the constraints on the ij edge.

The challenge is to generate meaningful α weights. Instead on relying on an error model of the sensors to derive the α_{ij} weights, we propose an architecture that enables the learning of these weights from errors measured with respect to the ground truth positions.

The work of Khossoussi shows that the structure of the covisibility graph is correlated to the volume of the uncertainty ellipsoids through the (weighted) number of spanning trees in the graph. Furthermore the optimal solution to the *1-edge selection problem* is the edge between the nodes with maximum resistance distance between them. Although the resistance distance measure is quite new in the field of active perception, it is a well researched notion and arises in a number of fields outside such as Markov chains and networking problems. For a primer on resistance distance and its optimization depending on graph topology see [Ghosh 2008, Ellens 2011].

We postulate that the resistance distance between two nodes in the covisibility graph G_α is correlated to the relative error between the estimate of the corresponding keyframes.

The resistance distance of two nodes in a graph G is equal to the resistance between the same two nodes in an electrical resistor network of the same topology as G, in which resistance values between vertices correspond to the edge weights of G.

Unsurprisingly, the resistance distance may be expressed as a function of the set of spanning trees in a graph. Let $G = (V, E)$ be a graph with unit edge weights and T the set of spanning trees of G, then the resistance distance R_{ij} between vertices i and j can be computed with the formula:

$$R_{ij} = \begin{cases} \frac{|\{t \mid t \in T,\, (i,j) \in T\}|}{|T|} & \text{if } (i,j) \in E \\ \frac{|T' - T|}{|T|} & \text{if } (i,j) \notin E \end{cases} \tag{3.1}$$

where T' is the set of spanning trees of $G' = (V, E + (i,j))$. This formula can be

extended for a graph with non unit edge weights using the *weighted* number of spanning trees.

The resistance can also be computed directly from the Laplacian matrix. Let L_α be the weighted Laplacian, or conductance matrix, of the graph G_α:

$$L_\alpha = A \operatorname{diag}(\alpha) A^T \tag{3.2}$$

where $A \in \mathbb{R}^{n \times m}$ is the incidence matrix of G and $\operatorname{diag}(\alpha) \in \mathbb{R}^{m \times m}$ is the diagonal matrix composed from the edges weights, also called conductances.

Let $\tilde{L}_\alpha^k$ be the matrix derived from L_α by removing the k-th row and column. Let Γ_α be the Moore-Penrose pseudoinverse of L_α. The effective resistance distance between nodes i and j can be computed using:

$$R_{kl} = \begin{cases} (\tilde{L}_\alpha^k)_{ii}^{-1} + (\tilde{L}_\alpha^k)_{jj}^{-1} - 2(\tilde{L}_\alpha^k)_{ij}^{-1} & i, j \neq k \\ (\tilde{L}_\alpha^k)_{ii}^{-1} & j = k \end{cases} \tag{3.3}$$

$$= (\Gamma_\alpha)_{ii} + (\Gamma_\alpha)_{jj} - 2(\Gamma_\alpha)_{ij} \tag{3.4}$$

As its name denotes, the resistance distance is a distance function:

- $R_{ij} \geq 0$ (positive)
- $R_{ij} = R_{ji}$ (symmetry)
- $R_{ij} \leq R_{ik} + R_{kj}$ (satisfies the triangle inequality)

Therefore R is a metric on G_α. Intuitively, R_{ij} is small when there are many paths between nodes i and j with high conductance, and high when there are few paths with low conductance. Adding a new edge, i.e. a new path always lowers the resistance distance between nodes. Thus it behaves as is expected of a relative error measure in SLAM: adding new measurements linking robot poses always lowers their relative localization error. Increasing the uncertainty of relative measures, thus decreasing the amount of information it encodes, increases the error.

3.3.3 Learning the relative error through the resistance distance.

We have seen that the resistance distance seems an interesting proxy for the relative error between nodes. To exploit this distance to precisely estimate errors, two problems must be solved:

- How to compute the weights α?
- How to derive the relative error from the resistance distance?

We propose an architecture that combines neural networks with the resistance distance in a semi-deep fashion to estimate the relative errors from the full graphical

representation of SLAM and the current solution. The overall process consists of the following four steps:

1. From two keyframes i, j that share covisible landmarks (*i.e.* an edge of the covisibility graph G_α), we extract a feature vector X_{ij} that encodes how well the two keyframes would be colocalized from the matched landmarks.

2. We learn a function f mapping features to the weight edges of G_α.

3. From G_α we can compute the resistance distance R_{kl} between any pair of nodes $(k, l) \in V$ (be they connected by edges in G_α or not).

4. Then a second learned function g maps the resistance distance R_{kl} to the relative error metric $\hat{e}_{kl}$.

If needed w independent metrics can be learned in parallel by using w outputs for f and inputs for g, thus computing the resistance distance on independent $G_{\alpha_0} \cdots G_{\alpha_w}$ graphs. Figure 3.4 illustrates the proposed architecture with two learned metrics.

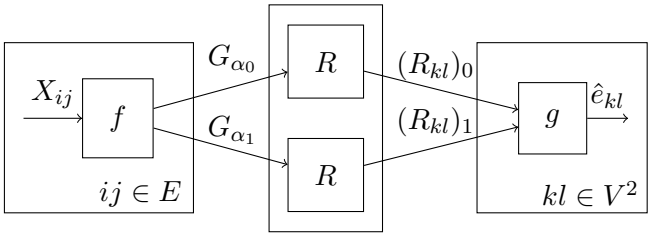


Figure 3.4: Architecture of the function computing the SLAM error estimate $\hat{e}_{kl}$ for $k, l \in V^2$. $G = (V, E)$ is the covisibility graph without weight on edges, G_α is G augmented with weights α_{ij}, with $(i, j) \in E$. $X_{ij} \in \mathbb{R}^k$ is a feature vector. The function f computes weights of the covisibility graph from features, R_{kl} computes a scalar value for any two nodes in G_α, and g transforms the output of R_{kl} in a relative error metric between nodes k and l. In this example two independent metrics are learned through f computing two weighted covisibility graphs G_{α_0} and G_{α_1}.

Because the resistance distance is differentiable, we can learn f and g together using neural networks. We simply need to define a cost function relating the estimated relative error $\hat{e}_{kl}$ to the ground truth error $\bar{e}_{kl}$, and use backpropagation to compute the gradients on the neural network parameters of f and g. The following section depicts the details of the implementation of this learning architecture.

3.4 Implementation of the learning architecture

3.4.1 Selecting informative features

Although the choice of features to feed to the neural network is crucial to its ability to learn the graph weights, choosing the best features is not the focus of this work. Hence we handpicked features capturing as best as we thought the nature of the constraints between keyframes, while remaining easy to compute. The following features are defined on the basis of the covisible landmarks between the two keyframes:

- the number of covisible landmarks
- the total number of additional observations per landmark (*i.e.* the number of other keyframes in which they are visible)
- the parallax defined by the camera poses associated to the keyframes
- the average distance between the landmark and the keyframes positions
- the global saliency of the landmarks (defined in the dictionary used for the bag of words place recognition)

Histograms are computed with the parallax, distance, number of observations and global saliency features associated to all covisible landmarks of an image pair. These histograms, the number of covisible landmarks and the ground area of the overlap between the two considered images are aggregated in the feature vector X_{ij}.

3.4.2 Loss function

The definition of an appropriate loss function for the optimization is conditioned by the definition of the error model for $\bar{e}_{kl}$ that we are trying to learn. Ideally, one would want to produce a full six dimensional error model for the poses. However, our learning architecture rather explicits synthetic topological information than metric information and relationships between the variables estimated by the SLAM. It is therefore impossible with this model to predict the full error model defined by the covariances between the 6 pose parameters. Following the results and observations of [Khosoussi 2017], we can however hope to learn synthetic information about the pose uncertainty ellipsoids.

Because we can not differentiate the dimensions, we aim at learning the norm of the position error, not modeling the rotational error. Let $\bar{p}_k$ be the ground truth position for node k, and $\hat{p}_k$ the SLAM position estimate for the same node. We define the relative positional error norm as:

$$\bar{e}_{kl} = \| \, (\hat{p}_k - \hat{p}_l) - (\bar{p}_k - \bar{p}_l) \, \| \tag{3.5}$$

The probabilities of the relative errors are supposed independent of each other given the model, therefore for n graph samples with m nodes, we can write the joint probability of the concatenated error vector $\bar{\mathbf{e}}$ given the model:

$$p(\bar{\mathbf{e}} \mid \theta, X) = \prod_{i=1}^{n} \prod_{k=1}^{m_i} \prod_{l=i+1}^{m_i} p(\bar{e}_{ikl} \mid \theta, X_i) \tag{3.6}$$

with θ the model parameters and X the feature vector corresponding to $\bar{\mathbf{e}}$.

The objective of the optimization is to find the parameters θ maximizing the joint probability of the data given the model:

$$\arg\max_{\theta} \; p(\bar{\mathbf{e}} \mid \theta, X) \tag{3.7}$$

Maximizing the log probabilities, this simplifies to:

$$\arg\max_{\theta} \; p(\bar{\mathbf{e}} \mid \theta, X) = \arg\max_{\theta} \; \log \prod_{i=1}^{n} \prod_{k=1}^{m_i - 1} \prod_{l=i+1}^{m_i} p(\bar{e}_{ikl} \mid \theta, X_i) \tag{3.8}$$

$$= \arg\max_{\theta} \sum_{i=1}^{n} \sum_{k=1}^{m_i - 1} \sum_{l=i+1}^{m_i} \log \; p(\bar{e}_{ikl} \mid \theta, X_i) \tag{3.9}$$

Assuming a Gaussian error model on the SLAM poses, we can compute the probability of observing an error as $\bar{e}_{ikl}$ given the learned error model $\sigma_{ikl} = \hat{e}_{kl}(\theta, X_i)$:

$$p(\bar{e}_{kl} \mid \theta, X) = \mathcal{N}(0, \sigma_{ikl} = \hat{e}_{ikl}(\theta, X_i)) \tag{3.10}$$

$$= \frac{1}{\sqrt{2\pi\sigma_{ikl}^2}} e^{-\frac{1}{2}\left(\frac{\bar{e}_{ikl}}{\sigma_{ikl}}\right)^2} \tag{3.11}$$

After simplification of all terms not affecting the maximum, and switching to minimizing the negative log probability:

$$\arg\min_{\theta} \; -\log p(\bar{\mathbf{e}} \mid \theta, X) = \arg\min_{\theta} \sum_{i=1}^{n} \sum_{k=1}^{m_i - 1} \sum_{l=i+1}^{m_i} \log(\sigma_{ikl}^2) + \left(\frac{\bar{e}_{ikl}}{\sigma_{ikl}}\right)^2 \tag{3.12}$$

with predicted standard deviations $\sigma_{ikl} = \hat{e}_{kl}(\theta, X_i)$.

We use equation 3.12 to compute the loss function for the neural network architecture.

3.5 Results

We present here results from applying the learning architecture on a simulated dataset relating to coverage path planning of crop fields. The simulation setup

allows to generate at will datasets with precise ground truth position of the observations, while being faithful enough to evaluate the SLAM and error prediction architecture. An alternate to runs on simulated environments would be to exploit real datasets, resorting to the application of full Bundle Adjustment to define the keyframe pose ground truth.

3.5.1 Simulation setup

The architecture of the simulation setup is showed in figure 3.5, it instantiates a plan / execute / perceive loop in the context of UAV coverage.

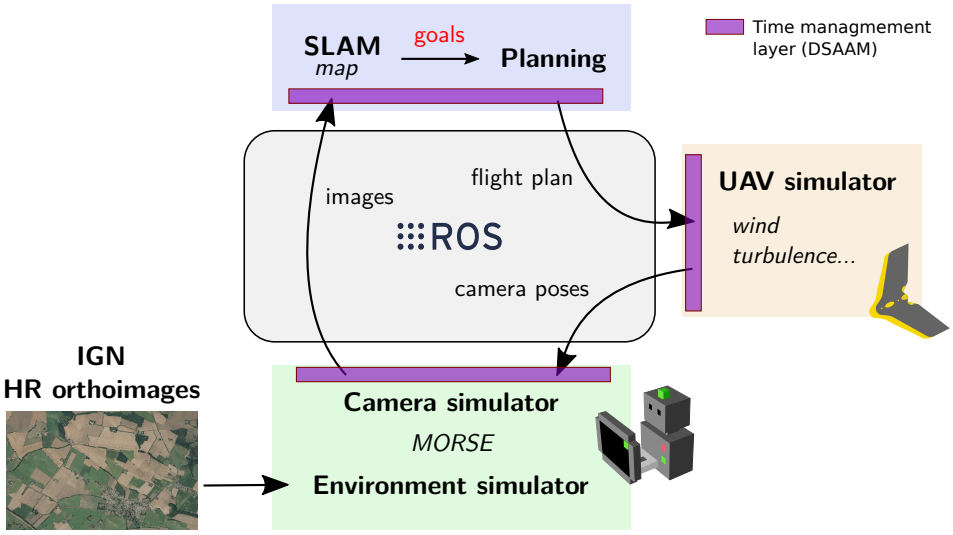


Figure 3.5: Simulation software architecture

The environment is simulated with high resolution 25 *cm* orthorectified image data from the french *National Geographic Institut (IGN)* [3], projected on a ground plane. The scenes are rural plains, with numerous crop parcels, roads, sparse trees and bushes... Three 12×12 kilometer tiles define three different environments.

A planning algorithm generates a Dubins trajectory covering the area to be mapped [Holvoet 2018]. It decomposes the area in cells, which are then covered in a boustrophedon manner. A value for the overlap of images on the ground is set: it defines the distance between parallel trajectory legs over a cell. The algorithm allows for skipping legs to optimize the trajectory length and to cope for radius of curvature constraints, so that there may be no side overlap between images at first, until the UAV circles back to complete the observation legs that were skipped during the first pass over the cell.

The plan is fed to a UAV flight simulator following the Dubins trajectory perfectly, but with added noise from perturbations, using the von Kármán wind gust model. The camera is simulated as if stabilized in roll and pitch, barring small

[3] http://ign.fr

perturbations of a few degrees. If this model is not perfectly realistic, it is good enough to evaluate the SLAM algorithm performances in the presence of moderate perturbations to the trajectory. The UAV flies at a constant altitude of $150\,m$, and a constant $15\,m.s^{-1}$ airspeed.

During flight, the position of the UAV is transmitted to the *Morse*[4] simulator, where a camera observes the overflown scene. The camera has an horizontal field of view of 61 degrees and generate 800×600 pixels images at 32 hertz. These images are then processed by a monocular SLAM implementation, forked from ORB_SLAM2 [Mur-Artal 2017][5].

All processes communicate with each other using the ROS framework and the DSAAM library is used for time management, ensuring the global consistency of the simulation.

The dataset comprises of fives areas to be mapped, with very distinct shapes (see section 3.5.3). Every hundred new generated keyframes, the covisibility graph with computed features as well as the SLAM and ground truth poses are dumped. The overflown areas range from about a half to a few km in length and width. Eighty runs were performed with a ground image overlap varying from 30 to 90%, and a turn radius between 35 and 50 meters. A handful simulations were rejected due to unrecoverable failures of the SLAM algorithm leading to inconsistent data.

3.5.2 Learning setup

Simulation results were aggregated and the dataset used to test the learning architecture, adding up to about four thousand samples. Even if the same five missions are used for all the dataset, varying the overlap and turn radius produces very different trajectories and SLAM results. In addition, taking samples every hundred new keyframes assures diversity in the sizes of the graphs, even more so that ORB_SLAM2 has a keyframe culling routine and therefore old parts of the graph may change when closing loops, especially with back and forth boustrophedon trajectories.

Fully connected neural network layers with rectified linear units were used for the f and g functions. A manual trial and error process was used to set the hyperparameters of the network, number of layers and units per layers. Function f has four hidden layers with 700, 100, 100 and 10 units respectively, and the g function has two hidden layers of 10 units each. Three different weighted covisibility graphs $G_{\alpha_{0..2}}$ are learned in parallel and the three resistance distance computed for each pair of nodes kl are combined by g to produce the σ_{kl} output. In order to ensure that no arcs disappear in the covisibility graph, a minimum threshold is used on the weights. This threshold is learned along with the other

[4] https://morse-simulator.github.io/

[5] This fork deviates mainly in bug fixes pertaining to multi-threading and in allowing a closer external usage from the data structure in order to produce the features necessary for our learning architecture

parameters of the network. Finally another fixed minimum threshold on the output σ_{kl} is set to 1e-5 to avoid division by zero in the loss function (see equation 3.12).

Although the gradients were computed on batches of 50 samples, we observed a quite noisy loss function during the learning process. One caveat is that the computation of the resistance distance relies on matrix inversion, and therefore is prone to fail in case of ill conditioned matrices. A few simulations had to be removed from the dataset, some samples producing Laplacian matrices with very high condition numbers, resulting in unusable outputs. All such instances were from quite large samples, with especially sparse covisibility graphs. Even without those instances, the architecture is quite susceptible to bad initialization leading to ill-conditioned matrices and very high losses, and the optimization algorithm being unable to recover. To circumvent this, we restarted the learning process with new random weights until the loss of the first batch crossed a certain maximum threshold.

Results shown are produced using models learned on the whole dataset excluding trajectories from the same mission as the example.

3.5.3 Qualitative analysis

Example A First we analyze the output of the learning on a simple boustrophedon trajectory. Figure 3.6 shows the covisibility graph. In this case there are many loop closures between the boustrophedon tracks at the start of the trajectory, with a larger gap towards the end.

Errors relative to the first keyframe are shown in figure 3.7, alongside with the 1σ predicted uncertainty. One has to keep in mind that the observed error is only one realization of the predicted probability distribution and thus it is difficult do evaluate the quality of the prediction on specific examples. One can however already see interesting topological results. As expected, the error grows with the distance to the reference keyframe. However loop closures (around keyframe 75 and again 150) bring the predicted error down. We can also observe the growth of the predicted uncertainty in the last (index > 175), weakly connected to the other part, as well as an observed error peak in the same region. In this instance the true error also follows globally the same pattern as the prediction – note however that the predictions being probabilistic in nature, there may be deviations with the actual error, as we will see in following examples.

Figure 3.8 shows the whole relative error matrices, measured and predicted (figure 3.7 actually plots the first line of these 2 matrices). We find the same global structure in the uncertainty estimate as in the ground truth error, with the prediction seemingly being more conservative in the error estimate. We find again the error peak in the band around index 215, which correspond to the last turn of the trajectory ($x = -0.7$, $y = -0.3$) that is topologically the farthest away from very well connected region in the beginning. Local maximums and minimums seem

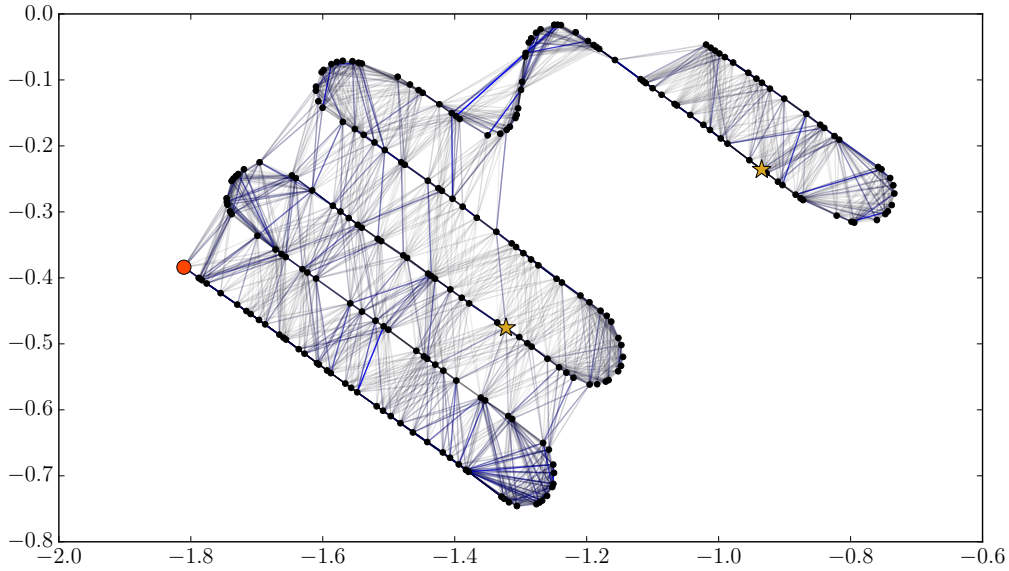


Figure 3.6: Covisibility graph for example A (coordinates in km). Every keyframe is marked with •, while ● marks the reference keyframe and ★ marks every hundredth keyframe. Covisibility edges showing the learned weights are drawn in shades ranging from solid blue (highest weights) to pale grey (lowest weights).

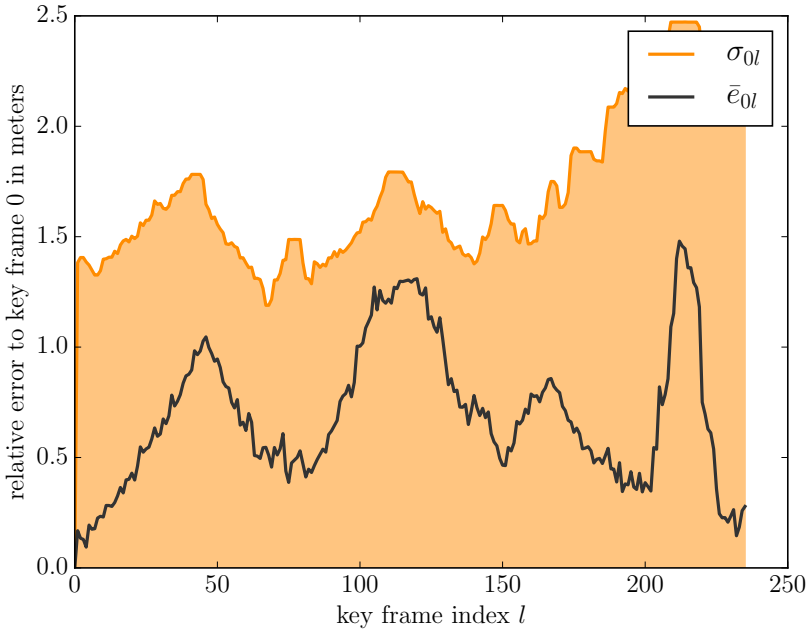
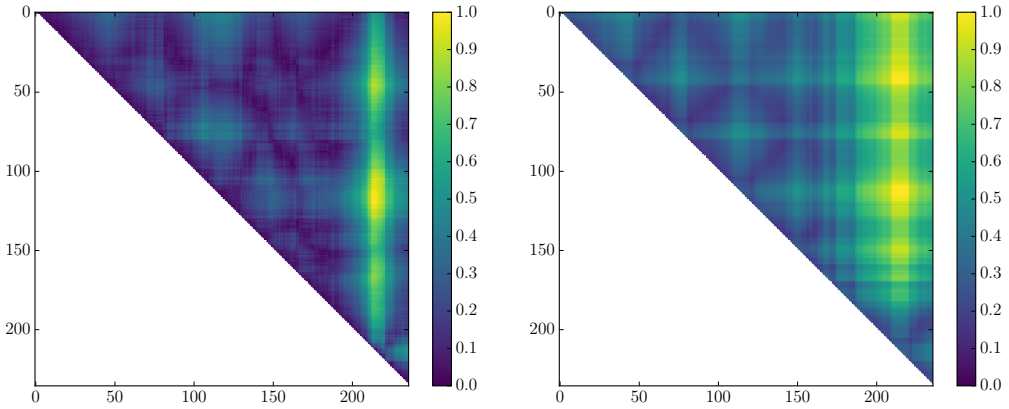


Figure 3.7: Relative errors with respect to keyframe 0 for example A. Ground truth $\bar{e}_{0l}$ is drawn in black, while the shaded orange area covers the predicted one sigma standard deviation σ_{0l}.



(a) Normalized $\bar{e}_{kl}$ (b) Normalized σ_{kl}

Figure 3.8: Relative error matrices for example A. Relative error between keyframe k and l is indicated by the shade of the cell (k, l). (a) Shows the normalized ground truth relative error $\bar{e}_{kl}$ and (b) the normalized predicted standard deviation σ_{kl}.

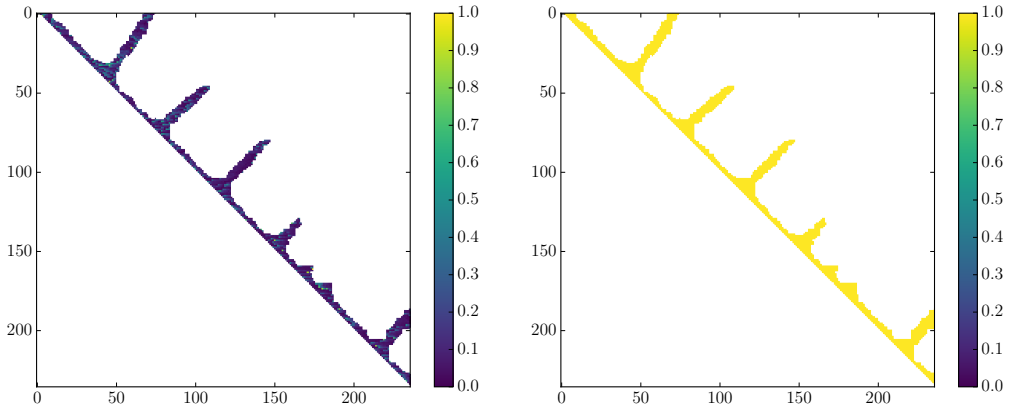
to correlate well with the back and forth motions closing numerous local loops.

Finally we can observe the intermediate weights G_{α_0} and G_{α_1}: figure 3.9 shows the learned weighted adjacency matrices of the covisibility graph for this example. Here we can make a few observations, that apply to all other samples we have explored. In (a), we can clearly see that a variety of weights have been learned, however (b) shows for the second learned weight matrix a purely topological adjacency (all weights have the same value), and in practice this corresponds to the lowest possible value given the threshold. Adding more intermediate G_α seem to always produce only one informative weight matrix, the other being (or very close to) a simple adjacency matrix multiplied by the value of the threshold. We conjecture that it seems to indicate that the structure of the problem as posed need only one distance metric to perform the prediction without adding redundant information, at least given the input features that were used in our implementation. Indeed, using only one intermediate G_α produced very similar results in the values of the loss function.

Example B This sample exhibits a single, long back and forth trajectory that links to a well connected rectangle, resembling the shape a hammer, as can be seen in figure 3.10. Here we expect the largest relative errors between the hammer head and the base of the shaft.

In this instance, the errors relative to the starting keyframe barely fit in the predicted 1-σ envelope, as shown in figure 3.11. The predicted uncertainty is as expected at its maximum at the base of the shaft, around index 100. However, due to the random walk nature of the error, it exhibits two peaks corresponding to a maximum error in the middle of the shaft.

This structure can be found again in the relative error and uncertainty prediction



(a) Normalized G_{α_0} (b) Normalized G_{α_1}

Figure 3.9: Weighted adjacency matrices $G_{\alpha_{0..1}}$ for example A, with normalized weight of the (k, l) indicated by the shade of the corresponding cell.

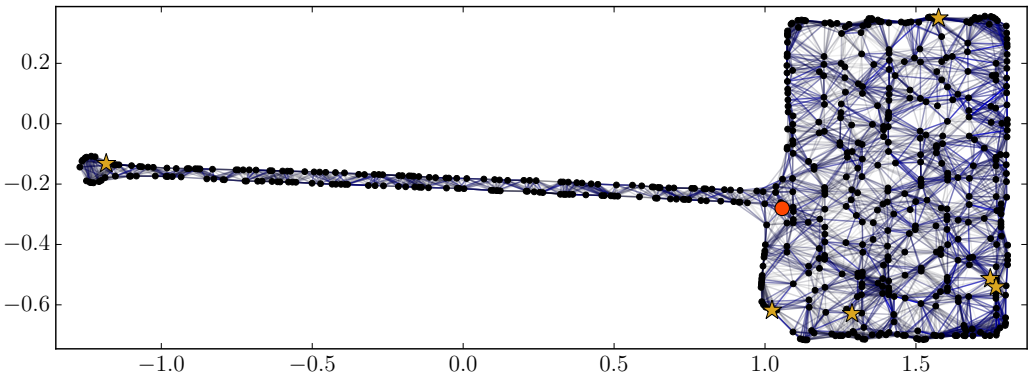


Figure 3.10: Covisibility graph for example B (coordinates in km). The trajectory starts at the meeting point of the shaft and the head of the hammer, descends into the shaft, then back up again and finally the head is mapped with back and forth motions.

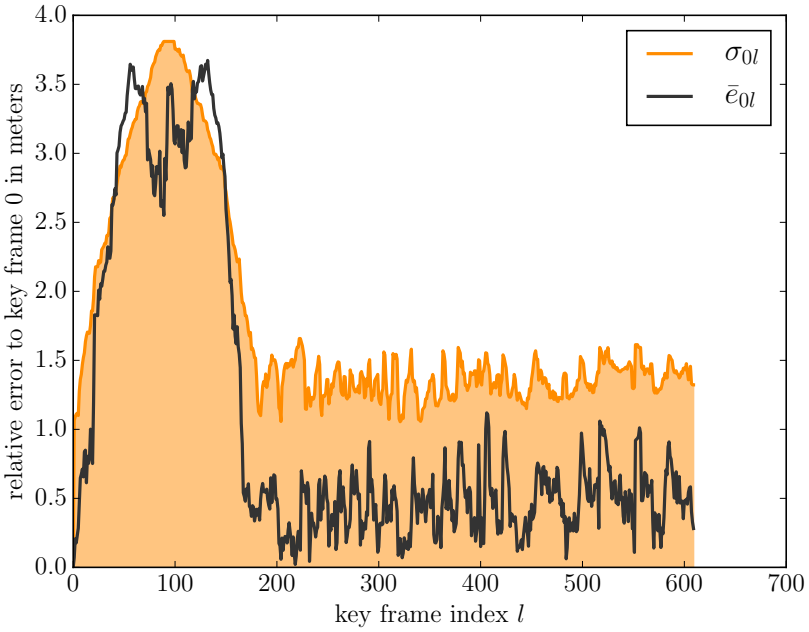


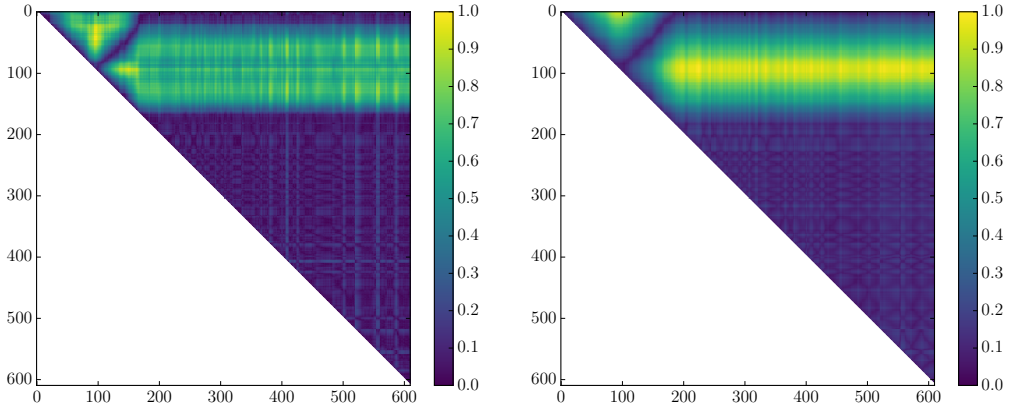
Figure 3.11: Relative errors with respect to keyframe 0 for example B. We see the expected peak of the uncertainty prediction at the base of the shaft around index 100. However due to the random walk nature of the error, it exhibits two peaks corresponding to the maximum error in the middle of the shaft.

matrices (figure 3.12). We clearly see the bands corresponding to the shaft, as well as the dual peaks in the ground truth error.

Example C Here we are interested in observing the results of a very large loop closure in the graph's topology. The trajectory follows an "O" shape four kilometers across, with enough width so that back and forth boustrophedon motions locally maintain a good local connectivity of the graph so as to avoid too large drifts. The covisibility graphs before and after loop closure are shown in figure 3.13.

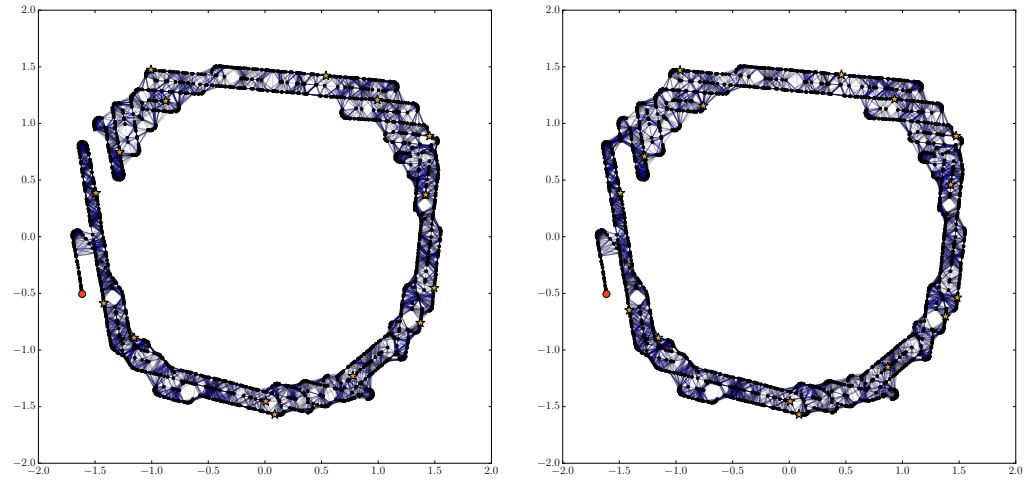
Relative errors to the starting keyframe (figure 3.14) exhibit two interesting results. First the loop closure is well taken into account by the model, the uncertainty prediction dropping radically from a maximum σ_{0l} around 5m dropping to 3m. Another interesting result is the peak of the error before loop closure around index 400. If the model predicts a small peak at this location, as its maximum the error is above the 2.5σ mark, which is quite unlikely. Furthermore, looking at other samples these high errors where the prediction does not keep up are far more common than the Gaussian model would assume, as will be shown in the statistical results (section 3.5.4).

Again in the relative error matrices (figure 3.15), we can observe that if small structural details are smoothed out, the dramatic effect of the loop closing is evident in the prediction as well as in the ground truth error. Smaller loop closures can



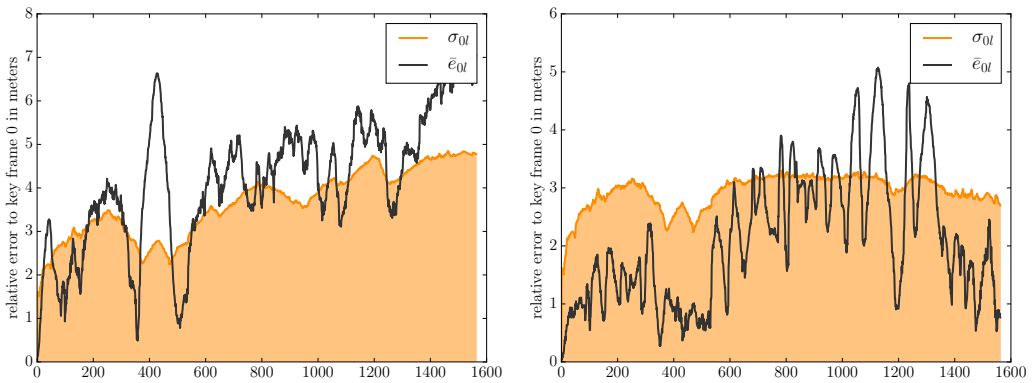
(a) Normalized $\bar{e}_{kl}$ (b) Normalized σ_{kl}

Figure 3.12: Relative error matrices for example B.



(a) Before loop closure (b) After loop closure

Figure 3.13: Covisibility graph for example C, before and after loop closure. Coordinates in kilometers.



(a) Before loop closure (b) After loop closure

Figure 3.14: Relative error between keyframe 0 and keyframe l for example C, before and after loop closure. The loop closure in (b) makes both the predicted uncertainty as well as the realized error drop consequently.

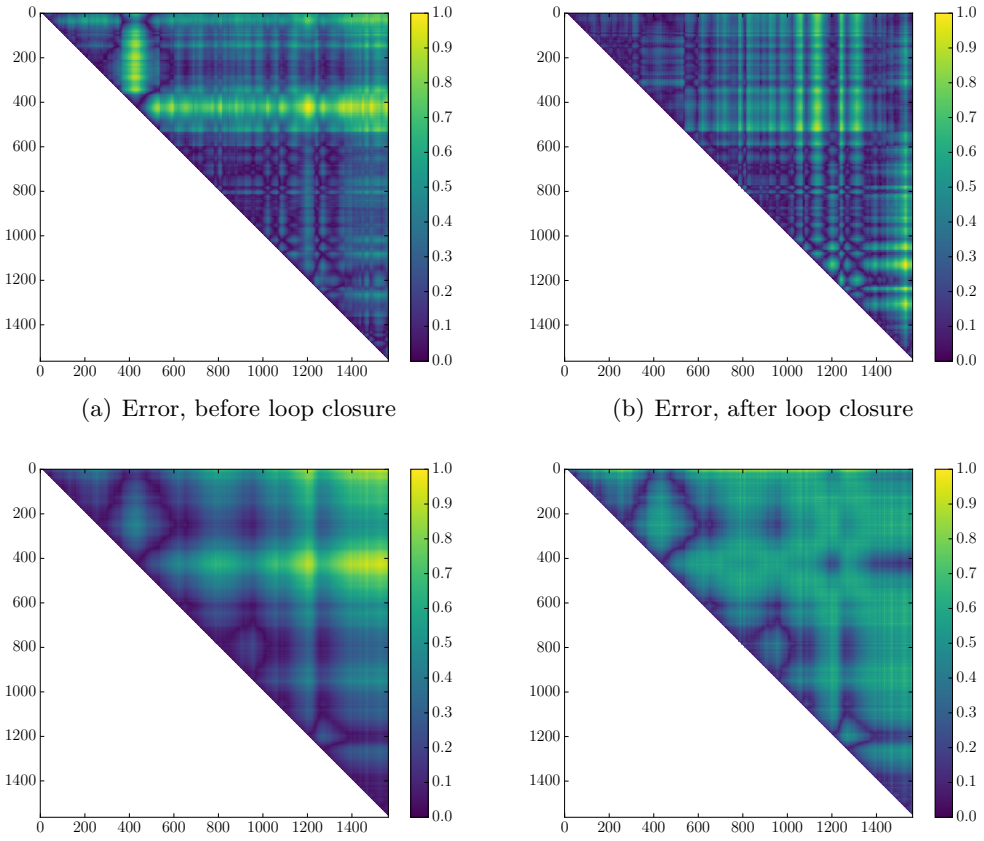
also be observed quite well in the predicted σ matrices, less so in the ground truth error, particularly after loop closure. For such large graphs, the error prediction is seemingly more often overconfident, failing to identify potential large deviations from accumulated errors and particularly the lever effect of rotational errors. It is not clear if the finer structure of the error is always due random walk noise, or if a more complete error model could capture it.

Example D This example shows a case with very high errors, accumulated quite quickly resulting in the prediction failing to produce probable results. The executed trajectory can be guessed in figure 3.16, which shows the covisibility graph. In this case the relative errors to the first keyframe attain higher than average values, with peaks around 10m, furthermore these errors vary very rapidly (figure 3.17). In contrast, the prediction never rises much above 2m, which results in errors deviating by more than 5σ from the model. Such events are not rare, more so in less extreme cases. The model seems to be unable to cope with very high error levels, especially accumulating very rapidly. The source is unclear, but the problem persists even when optimizing the model on the whole dataset: the model fails to fit these instances.

3.5.4 Quantitative results

We now examine statistical results computed on 400 trajectory samples, or about 10% of the dataset, for a total of about 59 millions data points. Three histograms of the ground truth error values $\bar{e}$, of the predicted standard deviation σ, and of the deviations of the error from the predicted standard deviation $\bar{e}/\sigma$ are respectively shown in figures 3.18, 3.19 and 3.20.

As can be seen in figure 3.18 the maximum error lies at 22 meters, while the



(a) Error, before loop closure

(b) Error, after loop closure

(c) Prediction, before loop closure

(d) Prediction, after loop closure

Figure 3.15: Normalized relative error matrices for example C, before and after loop closure. Some of the structure of the error matrix can be found in the uncertainty prediction, but most of the high frequency variations are smoothed out by the prediction, as if they corresponded to noise.

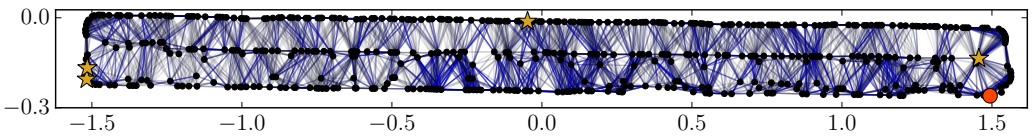


Figure 3.16: Covisibility graph for example D (coordinates in km).

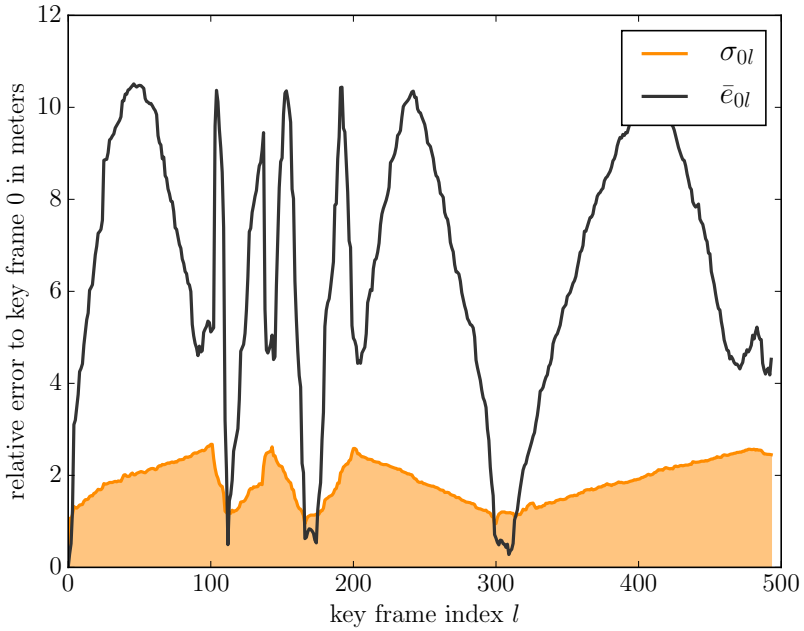


Figure 3.17: Relative error between keyframe 0 and keyframe l for example D. Deviations of the maximum ground truth error from uncertainty prediction attains up to 5σ.

maximum predicted standard deviation is under 8 meters (3.19). The peak of standard deviation prediction is around 1.5 meters and almost no predicted standard deviations lie under $1\,m$. Looking at the deviation of the error from the standard deviation 3.20, we begin to clearly see a slight bias: the peak is shifted between 0.25 and 0.5σ, whereas in the absence of bias the peak should be at zero in a normal distribution. Figure 3.21 shows the histogram of the same deviations as a function of distance. Even though it does not account for topology, nodes farther away in the euclidean space are also in average topologically farther away than closer nodes. The same bias can be found again, also it is increasing with the relative distance and much less drastic for short distances, under 500 meters.
The maximum event is over five but under six sigma. Events over 5σ are expected (approximately 1 every 1.7 millions), but events over 6σ would not be (approximately 1 every 506 millions). Fractions of the population lying inside the n-σ range, as well as the expected fraction value in a normal distribution are compiled in table 3.22. These indicate a slight overconfidence tendency of the uncertainty model, as well as a distribution with heavier tails. Possible explanations include the aforementioned bias, as well as the failure to predict very large and rapid deviations, such as shown in example D of the previous subsection 3.5.3.

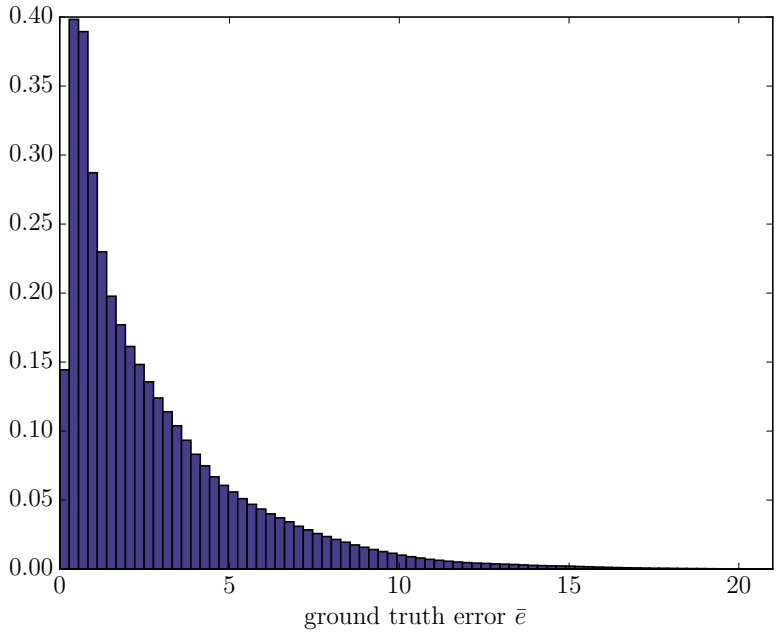


Figure 3.18: Histogram of the ground truth error $\bar{e}$ (in meters) across 400 random samples, or about 10% of the dataset.

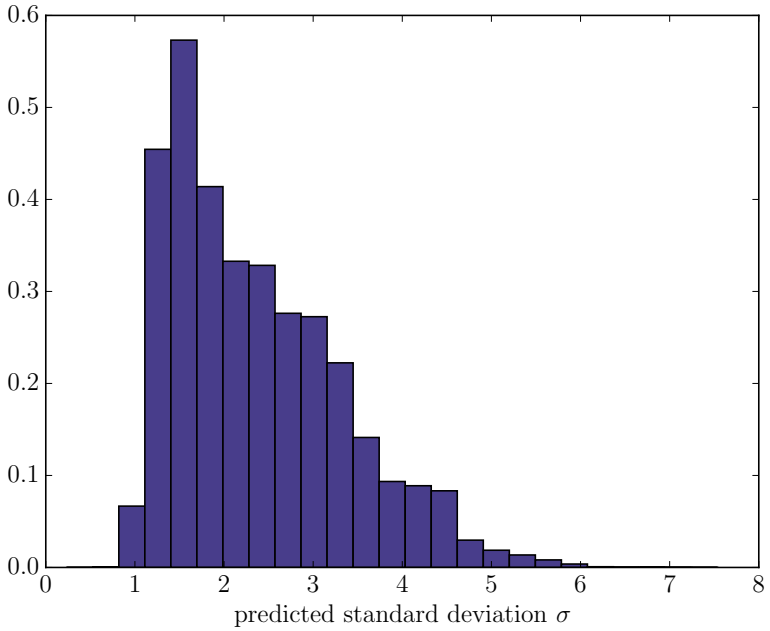


Figure 3.19: Histogram of the predicted standard deviation of the error σ (in meters) across 400 random samples, or about 10% of the dataset.

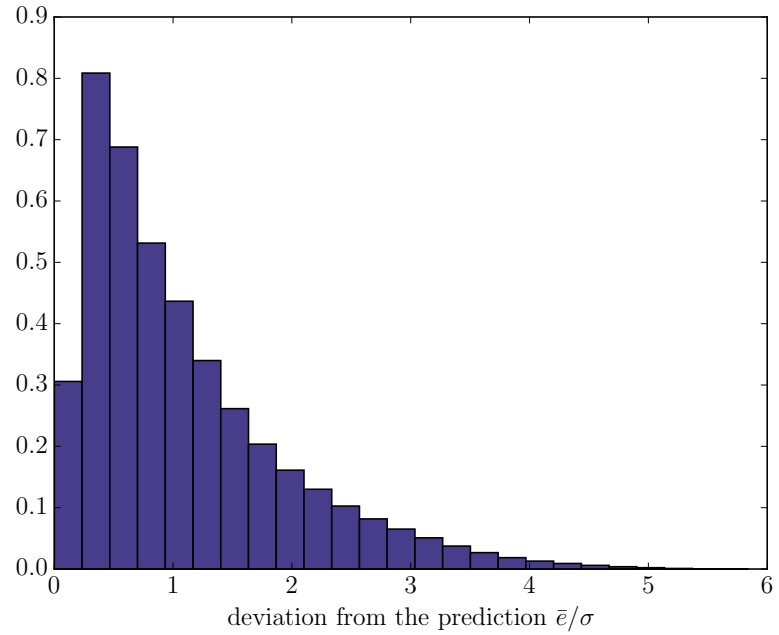


Figure 3.20: Histogram of the deviation of the ground truth error from the predicted standard deviation $\bar{e}/\sigma$ across 400 random samples, or about 10% of the dataset.

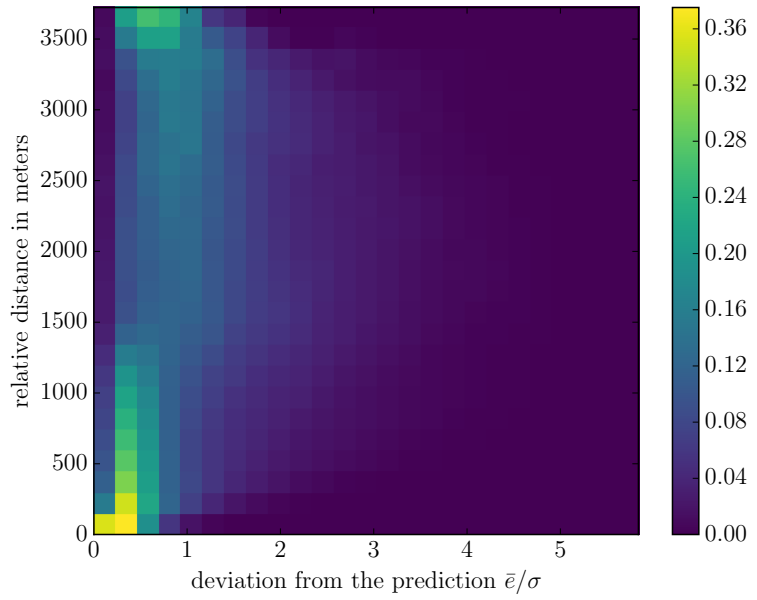


Figure 3.21: 2D histogram of the deviation of the ground truth error from the predicted standard deviation and relative keyframe distance in meters, across 400 random samples, or about 10% of the dataset.

Range	Realized fraction	Expected fraction
1σ	0.57	0.68
2σ	0.86	0.95
3σ	0.96	0.997
4σ	0.992	0.99993
5σ	0.9995	0.9999994
6σ	1	0.999999998

Figure 3.22: Fraction of the errors lying in the n-σ range of the predicted distribution and expected fraction according to the normal distribution. Statistics computed across 400 samples, or about 10% of the dataset. Total number of data points is just above 59 millions, so the maximum expected sigma event is under 6σ (approximate frequency for 6σ event is 1 in 506 millions).

3.6 Discussion and future works

We have presented a novel approach to learn a relative error model for a monocular SLAM algorithm, using mostly topological information from the weighted covisibility graph through the resistance distance. Weights are learned using features extracted from the data structures and the current state of the SLAM process. The learning architecture was tested in simulation using varied trajectories and shows promising results. Although there is a statistical slight tendency of overconfidence, the learned model captures well the error variations caused by the inherent graph topology, especially large scale topological features. Finer variations seem less precisely modeled, but it remains unclear what part of these is random walk noise.

We have argued how learning a relative error model benefits the planning process in the context of active SLAM, by exposing useful information that would otherwise remains hidden in the covariance matrix and partially erased by the anchoring process of the optimization. Furthermore a learned model allows for predictions of future observations, provided expected features can be computed. But a stronger argument for learning the error model from ground truth values is that it is less susceptible to outliers, as well as being less impacted by model inaccuracies such as missing biases and correlations. Therefore it has a far smaller tendency to overconfidence as opposed to the uncertainty recovered from the hessian matrix of the optimization process.

Nevertheless, this work is only the first step in the direction of mixing topological and metric information to predict SLAM uncertainties. Even on our simplified (no harsh rotations, flat ground) simulated environment, predictions are not perfect. We have seen that the model fails to capture the bigger uncertainties, especially with rapidly occurring variations, as well as in some instances smoothing too much the structure of the error in the graph. As proven in [Khosoussi 2017] in the case of 2D SLAM, topological information and sensor precision alone are not

sufficient to completely explain the volume of the uncertainty ellipsoids, metric information (i.e. the distance between keyframes) is also necessary. In our case, even though some metric information is included in the features, it is probably too rudimentary, and the reliance on the SLAM solution for the computations of these features introduces bias.

Although the lack of feature engineering is certainly a factor, there are certainly other factors. In particular the observed bias of the ground truth error seem to indicate a deviation from the zero-mean Gaussian distributed error that we use as hypothesis for the prediction. This deviation seems in particular to grow with the relative distance: it may be related to the lever effect of angular errors in the pose on positional error – which our model does not account for.

Certainly an interesting direction for future works would be produce richer models, with more complex architectures, integrating more metric information to predict a full error model (on each coordinate), as well as predicting rotational errors. One of the difficulties is to engineer a loss function that enables to mix positional and rotational errors with widely different scales. One direction might be to focus on minimizing the projection of the error on the landmarks. However the errors in rotation have a strong non linear effect, integrating the error projection in the loss function is therefore far from trivial.

Bibliography

[Cadena 2016] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid and John J. Leonard. *Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age.* IEEE Transactions on Robotics, vol. 32, no. 6, pages 1309–1332, December 2016.

[Carrillo 2012] Henry Carrillo, Ian Reid and Jose A. Castellanos. *On the Comparison of Uncertainty Criteria for Active SLAM.* In 2012 IEEE International Conference on Robotics and Automation, pages 2080–2087, St Paul, MN, USA, May 2012. IEEE.

[Carrillo 2015] Henry Carrillo, Philip Dames, Vijay Kumar and Jose A. Castellanos. *Autonomous Robotic Exploration Using Occupancy Grid Maps and Graph SLAM Based on Shannon and Rényi Entropy.* In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 487–494, Seattle, WA, USA, May 2015. IEEE.

[Das 2015] Jnaneshwar Das, Frédéric Py, Julio B.J. Harvey, John P. Ryan, Alyssa Gellene, Rishi Graham, David A. Caron, Kanna Rajan and Gaurav S. Sukhatme. *Data-driven robotic sampling for marine ecosystem monitor-*

ing. The International Journal of Robotics Research, vol. 34, no. 12, pages 1435–1452, 2015.

[Ellens 2011] W. Ellens, F.M. Spieksma, P. Van Mieghem, A. Jamakovic and R.E. Kooij. *Effective Graph Resistance*. Linear Algebra and its Applications, vol. 435, no. 10, pages 2491–2506, November 2011.

[Gao 2018] X. Gao, R. Wang, N. Demmel and D. Cremers. *LDSO: Direct Sparse Odometry with Loop Closure*. In International Conference on Intelligent Robots and Systems (IROS), October 2018.

[Ghosh 2008] Arpita Ghosh, Stephen Boyd and Amin Saberi. *Minimizing Effective Resistance of a Graph*. SIAM Review, vol. 50, no. 1, pages 37–66, January 2008.

[Hollinger 2014] Geoffrey A. Hollinger and Gaurav S. Sukhatme. *Sampling-based robotic information gathering algorithms*. The International Journal of Robotics Research, vol. 33, no. 9, pages 1271–1287, 2014.

[Holvoet 2018] Nicolas Holvoet. Planning for active mapping of crop fields using UAVs. Master's thesis, ENAC, September 2018.

[Kendall 2016] Alex Kendall and Roberto Cipolla. *Modelling Uncertainty in Deep Learning for Camera Relocalization*. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 4762–4769, Stockholm, Sweden, May 2016. IEEE.

[Khosoussi 2014] Kasra Khosoussi, Shoudong Huang and Gamini Dissanayake. *Novel Insights into the Impact of Graph Structure on SLAM*. In Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference On, pages 2707–2714. IEEE, 2014.

[Khosoussi 2015] Kasra Khosoussi, Shoudong Huang and Gamini Dissanayake. *Good, Bad and Ugly Graphs for SLAM*. In RSS Workshop on the Problem of Mobile Sensors, 2015.

[Khosoussi 2017] Kasra Khosoussi. *Exploiting the Intrinsic Structures of Simultaneous Localization and Mapping*. PhD thesis, UTS, 2017.

[Kim 2013] Soonkyum Kim, Subhrajit Bhattacharya, Robert Ghrist and Vijay Kumar. *Topological Exploration of Unknown and Partially Known Environments*. In 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3851–3858, Tokyo, November 2013. IEEE.

[Kim 2015] Ayoung Kim and Ryan M. Eustice. *Active Visual SLAM for Robotic Area Coverage: Theory and Experiment*. The International Journal of Robotics Research, vol. 34, no. 4-5, pages 457–475, 2015.

[Kitanov 2018] Andrej Kitanov and Vadim Indelman. *Topological Multi-Robot Belief Space Planning in Unknown Environments.* In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 1–7, Brisbane, QLD, May 2018. IEEE.

[Lawrance 2011] Nicholas RJ Lawrance and Salah Sukkarieh. *Autonomous exploration of a wind field with a gliding aircraft.* Journal of Guidance, Control, and Dynamics, vol. 34, no. 3, pages 719–733, 2011.

[Mu 2016] Beipeng Mu, Matthew Giamou, Liam Paull, Ali-akbar Aghamohammadi, John Leonard and Jonathan How. *Information-Based Active SLAM via Topological Feature Graphs.* In Decision and Control (CDC), 2016 IEEE 55th Conference On, pages 5583–5590. IEEE, 2016.

[Mur-Artal 2015] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós. *ORB-SLAM: A Versatile and Accurate Monocular SLAM System.* IEEE Transactions on Robotics, vol. 31, no. 5, pages 1147–1163, October 2015.

[Mur-Artal 2017] Raul Mur-Artal and Juan D. Tardos. *ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras.* IEEE Transactions on Robotics, vol. 33, no. 5, pages 1255–1262, 2017.

[Paull 2014] L. Paull, C. Thibault, A. Nagaty, M. Seto and H. Li. *Sensor-Driven Area Coverage for an Autonomous Fixed-Wing Unmanned Aerial Vehicle.* IEEE Transactions on Cybernetics, vol. 44, no. 9, pages 1605–1618, September 2014.

[Rahimi 2004] M. Rahimi, R. Pon, W. J. Kaiser, G. S. Sukhatme, D. Estrin and M. Srivastava. *Adaptive sampling for environmental robotics.* In IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004, volume 4, pages 3537–3544 Vol.4, April 2004.

[Ranganathan 2004] A. Ranganathan and F. Dellaert. *Inference in the Space of Topological Maps: An MCMC-Based Approach.* In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), volume 2, pages 1518–1523, Sendai, Japan, 2004. IEEE.

[Ranganathan 2011] Ananth Ranganathan and Frank Dellaert. *Online Probabilistic Topological Mapping.* The International Journal of Robotics Research, vol. 30, no. 6, pages 755–771, May 2011.

[Reymann 2018] C. Reymann, A. Renzaglia, F. Lamraoui, M. Bronz and S. Lacroix. *Adaptive sampling of cumulus clouds with a fleet of UAVs.* Autonomous robots, vol. 42, no. 2, pages 1–22, 2018.

[Wang 2018] Sen Wang, Ronald Clark, Hongkai Wen and Niki Trigoni. *End-to-End, Sequence-to-Sequence Probabilistic Visual Odometry through Deep Neural*

Networks. The International Journal of Robotics Research, vol. 37, no. 4-5, pages 513–542, 2018.

[Whelan 2016] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison and Stefan Leutenegger. *ElasticFusion: Real-Time Dense SLAM and Light Source Estimation*. The International Journal of Robotics Research, vol. 35, no. 14, pages 1697–1716, December 2016.

[Younes 2017] Georges Younes, Daniel Asmar, Elie Shammas and John Zelek. *Keyframe-based monocular SLAM: design, survey, and future directions*. Robotics and Autonomous Systems, vol. 98, 2017.

[Zhu 2017] Zhen Zhu and Clark Taylor. *Conservative Uncertainty Estimation in Map-Based Vision-Aided Navigation*. IEEE Transactions on Aerospace and Electronic Systems, vol. 53, no. 2, pages 941–949, April 2017.

Discussion

Summary Chapter 1 presents an approach to drive a fleet of information gathering UAVs to optimize the acquisition of information in a given area, while minimizing the energy expenses. The approach generates flight trajectories that properly exploit updrafts and generate accurate wind maps. Strengths and weaknesses of the Gaussian Process model are discussed. The lack of informative prior about the structure of the process, combined with the sparse sampling resolution is exposed as the principal hurdle to the predictive power of the model.

Chapter 3 presents a novel approach to learn a relative error model for the monocular SLAM problem. It exploits the relationship between the topology of the underlying graph representation of the problem with the precision of the solution. This relationship is not new, but it has been for the first time integrated in a learning architecture. Preliminary results obtained in simulation are promising, further widening the way towards interesting usage of topology and learning in SLAM.

Chapter 2 presents a decentralized architecture for simulation time management. It allows to synchronize multiple simulators to perform non-realtime, but repeatable, simulations. This work, although it may seem off topic compared to the other two contributions, has been inspired by the difficulties encountered while building the simulations necessary for the first contribution (Chapter 1). Subsequently it was used for the simulations performed in Chapter 3. Hence, along with the formal work, a major effort of this contribution is trying to convince for the pertinence of testing robotics software using repeatable simulations.

Concluding remarks Besides the technical contributions, more general concepts emerge from this work. Even though they might not be novel, we feel that they are important to state and conclude on.

First, incorporating the *structure of the problem* as a prior in the learning algorithm through careful *architecture of the solution* is paramount. It is the key to breaking the curse of dimensionality and enabling the network to learn interesting models, even when learning on small datasets. Indeed a large portion of the success of deep learning can be attributed to the introduction of convolutional layers, incorporating the specificity of the structure of images into neural networks. The work of Chapter 1 suffered from a lack of structure, while Chapter 3 shows promising result in trying to include graph topology into the learning architecture to enable learning of properties on graphs.

Second, the idea of building repeatable simulation diffusing through the robotics community more widely would certainly benefit it greatly. With recent computer graphics advancements such as real time ray tracing, fast photorealistic simulations will become a solid alternative to real world datasets as input of learning algorithms. Because it is artificially created, labeling of the dataset as well as exact ground

truth values come for free with it. Thanks to solid fundations on which to ensure repeatability, simulations become a valid solution to alleviate the ever increasing thirst for data of learning algorithms.

Finally, just as functional modularity - through the breaking of software into reusable components - has been a driving force of the expansion of the robotics community, software layer modularity inside components can be as structuring and greatly improve reusability. Instead of relying on swiss-knife frameworks and software that quickly become complex and hard to maintain, the software glue that we use should be broken down into thin libraries that handle specific concerns and can be combined together to build more complex software. This idea may be starting to germinate in the community: the Genom3 framework already allowed to target multiple middleware and recently the second iteration of the ROS framework, ROS2, relies on DDS as its middleware. We argue that the same principle should be used to incorporate simulation abilities within robotics frameworks.