Universidad de Alcalá

Departamento de Automática
Programa de Doctorado en Investigación Espacial

Characterizing and Evaluating Autonomous Controllers

Dissertation written by
Pablo Muñoz Martínez

Under the supervision of
Dra. María Dolores Rodríguez Moreno

International advisors
Dr. Amedeo Cesta and Dr. Andrea Orlandini

Dissertation submitted to the School of Computing of the Universidad de Alcalá, in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Characterizing and Evaluating Autonomous Controllers

2016

"The Viking Lander is a superbly instrumented and designed machine. It extends human capabilities to other and alien landscapes. By some standards, it's about as smart as a grasshopper, by others, only as intelligent as a bacterium. There's nothing demeaning in these comparisons; it took nature hundreds of millions of years to evolve a bacterium, and billions of years to make a grasshopper. With only a little experience in this sort of business, we're getting pretty good at it."

Carl Sagan
Cosmos – episode 5, 1980.

Acknowledgements

Como en cualquier tesis, el primer agradecimiento no puede ser más que para el director, o directora en este caso. En gran medida esta tesis es fruto de María Dolores. El esfuerzo y dedicación que he puesto yo en su confección posiblemente se vea superado por el suyo. No puedo más que expresar mi más profunda admiración por sus conocimientos, aptitudes y el entusiasmo que pone en todo lo que hace. Por ello, muchas gracias por permitirme hacer este viaje junto a ti.

También quiero agradecer a todos mis compañeros del laboratorio E31 en el que he pasado más tiempo que en mi propia casa durante los últimos años, con mención especial a Yolanda y Javi. El laboratorio sin vosotros se hizo un lugar aburrido. Por suerte he seguido contando con más gente de la que poder aprender: Alex y Rubén primero y ahora Dani, Diego y Fernando. También agradecer a los miembros del *Intelligent Systems Group* por su apoyo, sobre todo al Dr. David F. Barrero que siempre tendrá un hueco en el laboratorio. Y por supuesto al Dr. Bonifacio Castaño; espero que disfrutes de tu jubilación en lo más alto del mundo.

¿Y qué sería de una tesis sin la familia y amigos? Tengo que dedicar esta tesis a mis padres, Julián y Florinda y a mi hermana Flor. Siempre habéis estado ahí dándome apoyo en los buenos y malos momentos. Por ello esta tesis también es vuestra. Aunque para sobrellevar los peores momentos nada mejor que el baloncesto, jugando en Rucker junto a Xuma, Juampa, Alberto y el resto de la plantilla que tanto ha cambiado estos años.

During this thesis I spent some time in Rome with the PST group at ISTC-CNR. I want to thank all the PST staff, but specially to the senior members: Amedeo, Andrea, Riccardo and Angelo. You form an excellent group and I hope I can continue working and learning from you; this thesis is only a first step.

Finally, I want to thank to the ESA's technical officer Mr. Michel Van Winnendael for his continuous support. I really appreciate his comments during every stage of this thesis. Also, I appreciate the help from the people of the Automation and Robotics Section in ESA-ESTEC, with emphasis to Martín Azkarate and Carlos Crespo and their valuable knowledge about robotics.

This thesis was supported by the European Space Agency under the Networking and Partnering Initiative titled *Cooperative Systems for Autonomous Exploration Missions*, contract 4000106544/12/NL/PA in collaboration with ISTC-CNR (Italy).

Abstract

Autonomy in robotics by means of Artificial Intelligence (AI) Planning & Scheduling (P&S) is a widely research area with great interest in applications such as exploration robots in hazardous or human unreachable areas. However, autonomous controllers for robotics are usually not well assessed. For instance, it is not easy to compare newer assets with previous works in the field. In this thesis we propose a framework, called On-Ground Autonomy Test Environment (OGATE), to support testing and assessment of autonomous controllers. It is supported on a methodology and a set of generally applicable and domain independent metrics to generate objective evaluations, and a software tool to enable automatic benchmarking processes. To demonstrate the effectiveness of the framework, we exploit two autonomous controllers based on different P&S paradigms. First, the Goal Oriented Autonomous Controller (GOAC) developed under an European Space Agency (ESA) contract. Second, the Model-Based Architecture (MoBAr) developed during this PhD that exploits a different paradigm for autonomy. Particularly, MoBAr is designed with the objective of testing different Planning Domain Definition Language (PDDL) based planners to achieve on-board autonomy. In this regard, we also introduce a new planner, Unified Path Planning and Task Planning Architecture (up2ta), that integrates a state of the art PDDL planner with path planning algorithms. The objective of up2ta is to produce efficient plans for robotics exploration missions. Regarding to the path planning algorithm, in this thesis we introduce two algorithms focused on mobile robots: S-Theta* that effectively reduces the heading changes of the path, and the 3D Accurate Navigation Algorithm (3Dana) that deals with Digital Terrain Models (DTMs) and traversability cost maps to produce safer and reachable paths in realistic environments. Given both controllers, OGATE has been successfully exploited to evaluate them, allowing to characterize relevant aspects about the integration between Planning & Execution (P&E) that are hardly to be assessed in other ways. Moreover, the results are reproducible and objective, enabling comparison among controllers with different P&S technologies and paradigms.

Extended abstract

The expectations for robotic systems have evolved from the classical teleoperation paradigm of the previous century to the current aim of fully autonomous robots interacting with each other, being able to take their own decisions. However, although progress over the past few decades has been significant, such full autonomy still remains an open issue, partially due to the extensive variability and complexity of the scenarios to operate within. Reaching high autonomy levels can be justified in scenarios such as planetary and underwater exploration or natural disasters, as scenarios in which the human presence is rather than complicated, while also teleoperation may be infeasible.

In recent time the European Space Agency (ESA) has started to explore possibilities for autonomy on board in different internal activities and studies. In particular, they have produced the Goal Oriented Autonomous Controller (GOAC) architecture, which is a prototype for robotics autonomy in space missions. GOAC can be modified in order to test on-board Artificial Intelligence (AI) Planning & Scheduling (P&S) techniques to provide goal oriented autonomy. Moreover, GOAC can be customized exploiting different autonomy levels and P&S systems, allowing other research groups not to start from scratch but to take advantage of the work done. For this reason this PhD starts from ESA assets in robotics to investigate open problems. In particular, we propose research effort to better explore the evaluation aspects of different autonomy architectures, to study efficient reaction policies when executing the goals and allow operators to focus on what they want the mission to do. In this direction, a main objective is to generate a framework that deals with autonomous controllers evaluation, assessing aspects related to the P&S system and its integration in robotics.

With the objective of providing a wider support to verify the generated framework, this PhD provides the description of a new autonomous controller called Model-Based Architecture (MoBAr), that has similar functionalities as GOAC, but employing different technologies and paradigms for P&S. From this robotic controller, this thesis also addresses the decisional capabilities of such controllers. Particularly, the deliberative layer has to produce feasible and safe plans to guide the execution to achieve the mission goals. In this extent, robotics applications in planetary surfaces require navigation planners that take into consideration terrain constraints to safely achieve the target locations. Then, path planning capabilities seems to be a required component in an autonomous mobile robot when facing long term missions.

For the generation of safer routes two aspects have been covered. First, as some robotics have limited turning capabilities, a new path planning algorithm, called S-Theta*, has been produced. This algorithm effectively reduces the heading changes of the robot respect to its former one, Theta*. From this point, classical path planning algorithms usually exploit a loosely realistic 2D terrain discretisation, being in the best case a traversability cost map. However, this seems not to be enough to provide safer paths in uneven terrains. Then, a second assess produced in this thesis is the 3D Accurate Navigation Algorithm (3Dana), a path planning algorithm that provides support to deal with traversability cost maps as well as Digital Terrain Models (DTMs). DTMs provide realistic 3D surfaces, so using them it is possible to extract paths considering the terrain relief. 3Dana evolves from S-Theta*, and provides different parametrizations that allows generation of smooth paths, while also avoids excessive terrain slopes that cannot be reached by the robotic platform.

However, just providing a safe route between two points does not guarantee optimality in large missions with several targets. For this reason, we have merged a state of the art Planning Domain Definition Language (PDDL) planner with the proposed path planning algorithms. This new deliberative, called Unified Path Planning and Task Planning Architecture (UP2TA), interleaves task planning and path planning for optimal sequencing of activities taking into consideration the robot paths. UP2TA has been properly deployed as the planner for the MoBAr architecture, obtaining good results either in simulated and real scenarios.

Taking as input the GOAC and MoBAr controllers, it is required a framework to assess their performance. The motivation for such effort is twofold: (i) the difficulty to generate intensive test campaign for a given plan-based controller for robotics; and (ii) the lack of a general approach to assess and compare different Planning & Execution (P&E) approaches for the same robotic platform. This open issue has been clearly demonstrated during the development and testing of GOAC and MoBAr. Performing experiments with these controllers requires considerable hand made work, while the data generated is specific for the controller under study and not reproducible by other researchers.

Then, we propose a framework, called On-Ground Autonomy Test Environment (OGATE), that supports the integration, testing and operationalisation of autonomous robotic controllers. Particularly the OGATE framework is the union of three elements: (i) a methodology that defines a set of steps to generate a testbench for autonomous robotics; (ii) a set of general applicable metrics to characterize its performance; and (iii) a software environment that automatically carries on with the testbench. More in detail, the evaluation starts from the definition of the controllers under study and the operative scenarios. Then, through run series of plan execution experiments, relevant parameters measures are gathered to enable assessment. Such data is exploited to compute the proposed metrics values, which are domain and application independent. This is automatically done by means of the OGATE software that allows benchmarking in either, real and simulated robotic platforms. At the end of the tests execution objective and reproducible results of the controller are produced. OGATE has been successfully exploited to evaluate the GOAC and MoBAr plan-based controllers, allowing to characterize relevant aspects about the integration between P&E with different technologies and paradigms.

Resumen extendido

Las expectativas en robótica han evolucionado desde el clásico paradigma de teleoperación del pasado siglo al actual objetivo de completa autonomía, en la que los robots son capaces de cooperar e interaccionar tomando sus propias decisiones. En las últimas décadas se han realizado significativos progresos, pero la completa autonomía es todavía una meta inalcanzada. Esto se debe, en parte, a la gran variedad y complejidad de escenarios en los cuales los robots pueden operar. Entre ellos la exploración planetaria o submarina y escenarios de catástrofes naturales son potenciales candidatos a beneficiarse de robots autónomos, debido a que tanto la presencia humana como la teleoperación son complicadas, y, en algunos casos, imposibles.

Recientemente, la Agencia Espacial Europea (ESA) ha empezado a explorar la posibilidad de usar autonomía a bordo en sistemas robóticos. Entre los estudios realizados se encuentra el Goal Oriented Autonomous Controller (GOAC), un prototipo de arquitectura de control autónomo para robótica espacial. GOAC permite explotar técnicas de Inteligencia Artificial mediante Planning & Scheduling (P&S) para dotar al sistema de autonomía. En este sentido, diferentes niveles de autonomía y sistemas de planificación pueden ser utilizados, permitiendo desde la teleoperación hasta la autonomía basada en objetivos con planificación a bordo. Esta adaptabilidad permite que otros investigadores puedan realizar trabajos partiendo de una base consistente. Por ello, esta tesis se basa en de los trabajos realizados por la ESA en robótica autónoma para investigar problemas abiertos. En particular, proponemos una línea de investigación para explorar la evaluación de arquitecturas de control autónomo, analizando la ejecución de la misión para que los operadores se concentren en las metas a conseguir y no en cómo llevarlas a cabo. En esta dirección, un objetivo principal es la creación de un entorno de trabajo que simplifique la operación de sistemas autónomos, a la par que evalúa diferentes aspectos acerca de los componentes de P&S y su integración en plataformas robóticas.

Con el objetivo de proporcionar una comparativa relevante y verificar el entorno de trabajo desarrollado, en esta tesis presentamos un nuevo controlador autónomo llamado Model-Based Architecture (MoBAr), que integra capacidades similares a GOAC, pero empleando diferentes tecnologías y paradigmas de P&S. Uno de los aspectos más relevantes de la arquitectura es la capa deliberativa, la cual genera los planes que guían la ejecución en pro de los objetivos marcados. Centrándonos en las aplicaciones para exploración planetaria, el sistema autónomo debe integrar planificación de rutas para alcanzar los objetivos considerando el terreno en que se halla, tratando de minimizar la distancia recorrida para completar la misión.

Para enfrentarnos a la generación de rutas, hemos analizado dos aspectos. Primero, dado que ciertos robots tienen limitada capacidad de rotación, hemos desarrollado el algoritmo S-Theta*. Este reduce los cambios de dirección en comparación con el algoritmo original, Theta*. No obstante, ambos algoritmos utilizan una representación del terreno poco realista y limitada a entornos 2D, siendo en el mejor de los casos un mapa de costes transversales. En todo caso, es insuficiente para representar terrenos abruptos. Por ello, como segundo resultado, presentamos el 3D Accurate Navigation Algorithm (3Dana) que permite utilizar tanto un mapa de costes transversales como un Modelo Digital del Terreno (MDT). Este último permite representar fielmente el entorno, permitiendo extraer rutas que consideren parámetros como la elevación o la pendiente. 3Dana evoluciona de S-Theta* y, mediante el uso de MDTs, permite generar rutas optimizadas en función de la elevación, los cambios de dirección y limitadas por la pendiente máxima alcanzable por el robot.

Además de la ruta entre dos puntos, abordamos la optimización de los planes cuando hay que alcanzar varios objetivos alejados entre sí. Para ello, hemos integrado un planificador basado en el Planning Domain Definition Language (PDDL) con los algoritmos de planificación de rutas desarrollados. Como resultado, hemos implementado el planificador llamado Unified Path Planning and Task Planning Architecture (UP2TA) que entrelaza planificación de tareas y planificación de rutas para optimizar el recorrido de robots en misiones multiobjetivo. Este planificador ha demostrado su utilidad como capa deliberativa de la arquitectura MoBAr.

Analizando el desarrollo y las pruebas realizadas con las arquitecturas GOAC y MoBAr, queda patente que se requiere un entorno de trabajo para analizar el rendimiento de éstas. Esto se basa en dos frentes abiertos: (i) la dificultad de generar pruebas intensivas para controladores basados en P&S y, (ii) la falta de una solución general para analizar y comparar diferentes integraciones entre Planning & Execution (P&E). En el caso de las arquitecturas presentadas, la experimentación requiere un laborioso trabajo manual específico para cada una, mientras que los resultados generados no son (en gran parte) comparables ni reproducibles por otros investigadores.

En esta tesis proponemos un entorno de trabajo llamado On-Ground Autonomy Test Environment (OGATE) para llevar a cabo la ejecución, testeo y operacionalización de arquitecturas de control autónomo para robótica. OGATE es la conjunción de tres componentes: (i) una metodología que define los pasos para generar un conjunto de experimentos; (ii) un grupo de métricas globalmente aplicables para caracterizar el rendimiento; y (iii) una herramienta que automáticamente lleva a cabo las pruebas experimentales. Más en detalle, la evaluación empieza definiendo los controladores a estudiar y los escenarios sobre los que operan. Después, mediante múltiples ejecuciones de los experimentos definidos, se obtiene información relevante sobre el desempeño de las diferentes partes del controlador. Dicha información es usada para, mediante las métricas, obtener medidas de rendimiento que son independientes del dominio y de la aplicación. Esto es llevado a cabo automáticamente por la herramienta software tanto en plataformas simuladas como reales. Al final de la fase experimental, OGATE genera resultados objetivos y reproducibles de los controladores bajo estudio. En este sentido, OGATE se ha empleado para analizar GOAC y MoBAr, permitiendo evaluar y caracterizar diferentes aspectos de la integración de P&E usados por estos controladores.

Contents

List of Figures

List of Tables

List of Acronyms

3Dana	3D Accurate Navigation Algorithm
3T	Three Tiers/Layers
A*PS	A* Post Smoothed
ACT-R	Adaptive Control of Thought-Rational
AI	Artificial Intelligence
ALFA	A Language For Action
ALFUS	Autonomy Levels For Unmanned Systems
APSI	Advanced Planning & Scheduling Initiative
ARMADiCo	Autonomous Robot Multi-agent Architecture with Distributed Coordination
ATLANTIS	A Three-Layered Architecture for Navigating Through Intricate Situations
AuRA	Autonomous Robot Architecture
BIP	Behaviour Interaction Priority
CDT	Controller Dispatching Time
CLARAty	Coupled Layer Architecture for Robotic Autonomy
CMT	Controller Monitoring Time
CMU	Controller Memory Usage
CPU	Controller Processor Usage
CRT	Controller Reaction Time
CSP	Constraint Satisfaction Problem
CST	Controller Sensing Time
CSV	Comma Separated Values
CTD^{lb}	Command Time Discrepancy Lower-Bound
CTD^{ub}	Command Time Discrepancy Upper-Bound
DDL	Domain Definition Language
DTM	Digital Terrain Model
ECSS	European Cooperation for Space Standardization
ESA	European Space Agency
ESL	Execution Support Language
EUROPA	Extensible Universal Remote Operations Architecture
GAPPS	Goals As Parallel Program Specifications
GCS	Ground Control Station
G$^{\mathbf{en}}$oM	Generator Of Modules
GOAC	Goal Oriented Autonomous Controller

GS	Global Score
GUI	Graphical User Interface
HiRISE	High Resolution Imaging Science Experiment
HMI	Human Machine Interface
HSTS	Heuristic Scheduling Testbed System
HTN	Hierarchical Task Network
IDEA	Intelligent Distributed Execution Architecture
IPC	International Planning Competition
ISS	International Space Station
JPL	Jet Propulsion Laboratory
KE	Knowledge Engineering
LAAS	Laboratory of Analysis and Architecture of Systems
MER	Mars Exploration Rovers
MoBAr	Model-Based Architecture
MRO	Mars Reconnaissance Observer
NASA	National Aeronautics and Space Administration
OBDD	Ordered Binary Decision Diagrams
OCRD	Ordered Constrained Rules Diagram
OGATE	On-Ground Autonomy Test Environment
OpenPRS	Open Procedural Reasoning System
P&E	Planning & Execution
P&S	Planning & Scheduling
PDDL	Planning Domain Definition Language
PDE	Planner Deliberation Efficiency
PDL	Problem Definition Language
PDM	Planner Deliberation Memory
PDT	Planner Deliberation Time
PE	PLEXIL Executive
PerMFUS	Performance Measures For Unmanned Systems
PET	Plan Effective Time
PLEXIL	Plan Execution Interchange Language
PMA	Planner Model Analogy
POCL	Partial Order Casual Link
PRM	Probabilistic Road Maps
PRS	Procedural Reasoning System
PSF	Planner Synchronization Frequency
PSR	Planner Synchronization Ratio
PTA^{lb}	Plan Time Accuracy Lower-Bound
PTA^{ub}	Plan Time Accuracy Upper-Bound
PTU	Pan-Tilt Unit
R^2**C**	Request and Reports Checker
RA	Remote Agent
RAP	Reactive Action Package
ROS	Robot Operating System
RPC	Remote Procedure Call
RRT	Rapidly-exploring Random Trees

S-Theta*	Smooth Theta*
SARA	Science Assessment and Response Agent
SLAM	Sample Localization And Mapping
SRI	Stanford Research Institute
SSS	Servo, Subsumption, Symbolic
STN	Simple Temporal Network
STRIPS	Stanford Research Institute Problem Solver
T-REX	Teleo-Reactive Executive
TCA	Task Control Architecture
TDL	Task Description Language
TRF	Timelines Representation Framework
TVCR	Time-line Validation and Control and Repair
UAV	Unmanned Aerial Vehicle
UP2TA	Unified Path Planning and Task Planning Architecture
XML	eXtensible Markup Language

Chapter 1

Introduction

In this chapter we present a frame of reference to the work done in this PhD. First, we describe the motivation. Then, we define the objectives and the structure of this dissertation. Finally, we list the publications generated in this thesis.

1.1 Motivation

Robotics have evolved from teleoperated systems to autonomous platforms that are able to safely operate in industrial applications or exploration missions. While the first case can be typically supervised by humans operators, planetary or underwater exploration scenarios requires autonomous capabilities such as on-board planning to survive and to achieve the mission goals with low human interaction. Then, several approaches have been followed to make autonomous and reliable control architectures for robotics. These systems are typically conformed by various software components hierarchically structured in levels or layers to provide autonomous capabilities. The most representative schema is the Three Tiers/Layers (3T) architecture [63]. In such schema, the higher layer corresponds to the Artificial Intelligence (AI) Planning & Scheduling (P&S) system, while the lower one provides the platform functionality, i.e., access to the robot sensors/actuators. Then, between both layers, there is an executive that enables Planning & Execution (P&E) integration.

In recent time, the European Space Agency (ESA) has started to explore possibilities for on-board autonomy in different internal activities and studies. For instance, a recent outcome on the robotics and automation field is the plan-based controller called Goal Oriented Autonomous Controller (GOAC) [28], which is a prototype for robotics autonomy in space missions. However, such system is technically complex to operate and to deploy, requiring a deep understanding of the different layers to safely operate it. Then, it is interesting to explore possibilities to create a more general autonomous controller that enables exploiting different P&S systems, but also abstracted models for the execution modelling. Exploiting high level abstractions not only for the deliberative layer, but also for lower layers could lead to more adaptable robotics controllers, while the models can be easier reusable for different applications/platforms.

In this direction, providing a controller that can use different P&S system can also be useful as a platform to test different technologies, so we can evaluate various approaches to choose the best one for our particular application. Notwithstanding, assessing the integration of different P&S system in an autonomous controller is currently an open issue in the autonomous robotics field. While evaluating P&S systems in standalone tests is a well established practice for Planning Domain Definition Language (PDDL) based planners, dealing with other paradigms (e.g., timelines based planners) and/or its integration in robotics requires to analyse the P&E integration. Currently, there is not a common framework for such purpose, making hard such assessments. In fact, experimental evaluations of autonomous controllers reported in the literature rely on rather specific evaluation criteria and experimental configurations that are hardly reproducible and/or exportable to different systems [56]. Also, it is hard to extract conclusions about how the integration between P&E behaves, as it is not easy to analyse the performance of each layer of the autonomous controller during execution. For instance, there is no data about the performance of the P&S system used for deliberation, or how the model employed fits the robot behaviours. Then, if we want not only to evaluate different P&S systems over an autonomous controller but also to compare different controllers (e.g., Model-Based Architecture (MoBAr) and GOAC), we need to define a framework generally applicable, regardless the technologies used.

From the state of the art we can find different approaches that theoretically cover this problematic, but they do not provide any suitable method to perform tests in a general system. Other researchers are able to identify particular problems and correctly analyse them in a standalone manner. However, they are not enough to provide feasible assessments for the whole controller. These advances are in the good direction, but we need to deal with a complete autonomous controller, not only with separate layers, while also providing sufficient details for assessing the P&S system and its integration in a robotic platform. How the different layers are connected and how they interact have a great impact in the overall system performance. As well, the implicit uncertainty of the real world has to be properly captured in the P&S model, so the controller can safely operate autonomously.

Thus, it is required a research effort to better investigate the key points that affect the performance of plan-based controllers. In particular, investigating this issue could lead to an improvement in the autonomous robotics field by providing a general approach for performance assessment based on reproducible and quantitative testbenchs. Looking at other fields, e.g., automated planners [103], it is quite clear that a well defined testing methodology supported on metrics is highly desirable.

1.2 Objectives

This PhD has a twofold objective. First, generate an adaptable autonomous controller based on high level descriptions of the robot behaviours, and to deploy it in a surface exploration scenario to demonstrate its effectiveness. Second, as the new controller cannot be compared with other state of the art controllers, we want to create a common framework that enables evaluation and comparison of autonomous controllers, regardless the technologies employed or the application domain.

The MoBAr controller presented in this dissertation is the evolution of a previous work [130]. For this thesis, the objective is to include new AI technologies to improve the autonomous capabilities, focusing on surface exploration missions. In this regard, we isolate the following specific objectives:

1. Improving the path planning capabilities by considering the heading changes during the path search. In particular, the objective is to reduce the heading changes that the robot has to make to reach its destination.

2. Creating a path planning algorithm that allows the robot to plan its route having in mind the terrain features (rocks, hazardous areas, etc.) in order to extract safer paths, but also, to avoid those areas that overcome its operational constraints (e.g., high slopes).

3. Proposing a method to interleave path planning and task planning that can be generally applicable in plan-based deliberative layers. The key point is to create a deliberative that effectively optimize the path between multiple tasks to minimize the solution cost.

4. Deploying new models for MoBAr to enable the operationalisation of the different P&S systems, in particular, the proposed above.

The work presented in this dissertation is also motivated by the lack of a general framework to deal with autonomous controllers performance in order to characterize and evaluate them. To do this, we need to focus the research in the key points that affects the integration of P&S systems in robotics. In this direction, this dissertation aims at contributing according to the following points:

5. Defining a methodology for evaluating autonomous controllers that can be generally applicable, independently of the application domain or the technology of the assessed controller.

6. Analysing the key aspects that affect P&S and its integration in robotics, creating a set of metrics that allows us to characterize and evaluate autonomous controllers in a general way. Such metrics shall be formally defined, so, following the methodology and exploiting the metrics we can produce quantitative and reproducible testbenchs.

7. Creating a software tool that, implementing the proposed methodology and metrics, enables the assessment of autonomous controllers in an automated manner. This software has to evaluate actual robotic platforms with simulators or real platforms under controlled and reproducible experimental conditions.

8. Performing large experimental campaigns to compare GOAC and MoBAr in order to test the correctness of the proposed framework.

1.3 Structure

This section provides an outline of the chapters that compose this dissertation.

- **Chapter 1:** describes the motivation, objectives, contents and publications produced in this thesis.
- **Chapter 2:** presents the state of the art for the different fields that have been covered in this thesis. First, this chapter provides an introduction to autonomous controllers and a survey of the most representative ones in the literature. The chapter continues presenting the path planning problem over flat surfaces and the most relevant heuristics search algorithms applied to it. Following, some approaches that perform path planning over 3D terrains are introduced. Then, we summarize different techniques that integrate task planning and path planning for autonomous controllers. Finally, the description of several works that attempt to provide performance evaluation and characterization of autonomous controllers is provided.
- **Chapter 3:** faces the problem of considering the heading changes of the robot in heuristic search path planning algorithms over flat surfaces. The heading changes has been considered as part of the heuristic function (generating greedy algorithms) and for the cost function (obtaining the S-Theta* algorithm). The algorithms produced have been evaluated using randomly generated maps.
- **Chapter 4:** defines a DTM that allows being exploited by path planning algorithms in order to extract paths in realistic surfaces. Using such representation, this chapter presents a new path planning algorithm, 3Dana, that aims to generate safer and smoother paths considering the terrain relief. This algorithm has been tested in either randomly generated and real Mars maps.
- **Chapter 5:** proposes a deliberative schema that interleaves task planning and path planning for efficient mission planning in autonomous exploration applications. Such schema is implemented in the UP2TA deliberative employing a PDDL planner and the path planning algorithms introduced in the previous chapters. This new planner has been tested under a planetary exploration domain.
- **Chapter 6:** describes the MOBAR autonomous controller, its layers and the models used for an exploration domain. This chapter also provides the experimental results of the controller in either real and simulated scenarios, taking advantage of the UP2TA planner presented in the previous chapter.
- **Chapter 7:** defines the OGATE framework for evaluating and characterizing autonomous controllers. First, a methodology for generating objective and reproducible experimental campaigns is provided. Then, a set of generally applicable metrics for P&S assessment in robotics are formally presented. Such methodology and metrics are operationalised in a software tool that automatically carries on with the testbench. Using it, an extensive experimental campaign has been performed to test different configurations of GOAC over

a planetary exploration domain with different problems hardness, allowing a characterization of the controller over that domain. Finally, a comparison between MoBAr and GOAC using the proposed framework has been made.

- **Chapter 8:** presents the conclusions of this thesis and some future research directions.

- ***Appendix A:*** defines the algorithms and parameters used for the random map generation in the different path planning experiments performed in this dissertation.

- ***Appendix B:*** presents three extra Mars maps and the solutions provided by the 3Dana path planning algorithm presented in chapter 4.

1.4 Publications

The results of this monograph have produced several publications in the field of AI in path planning and autonomous controllers. Following, the list of publications is presented, clustered by field.

Path planning:

- Muñoz, P. and R-Moreno, M. D. Improving efficiency in any-angle path planning algorithms. In Procs. of the *6th IEEE International Conference on Intelligent Systems* (Sofia, Bulgaria, September 2012).

- Muñoz, P., R-Moreno, M. D., Martínez, A. and Castaño, B. Fast path planning algorithms for future Mars exploration. In Procs. of the *International Symposium on Artificial Intelligence, Robotics and Automation in Space* (Turin, Italy, September 2012).

- Muñoz, P. and R-Moreno, M. D. S-Theta*: low steering path planning algorithm. In Procs. of the *32nd SGAI International Conference on Artificial Intelligence* (Cambridge, UK, December 2012).

- Muñoz, P., Barrero, D. F. and R-Moreno, M. D. Run-time analysis of classical path planning algorithms. In Procs. of the *32nd SGAI International Conference on Artificial Intelligence* (Cambridge, UK, December 2012).

- Muñoz, P., Barrero, D. F. and R-Moreno, M. D. Statistic methods for path planning algorithms comparison. *Künstliche Intelligenz*, 27 (3), 201–211 (2013).

- Muñoz, P., Barrero, D. F. and R-Moreno, M. D. A statistically rigorous analysis of 2D path planning algorithms. *The Computer Journal*, 58 (11), 2876–2891 (2014).

- Muñoz, P. and R-Moreno, M. D. *On heading change measurement: improvements for any-angle path planning.* Novel Applications of Intelligent Systems, ch. 6 (2015).

- Muñoz, P. and R-Moreno, M. D. 3Dana: Path Planning on 3D surfaces. In Procs. of the *36th SGAI International Conference on Artificial Intelligence* (Cambridge, UK, December 2016).

Autonomous controllers:

- Muñoz, P., R-Moreno, M. D. and Castaño, B. Integrating a PDDL-based planner and a PLEXIL-executor into the PTinto robot. In Procs. of the *23rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: Next-Generation Applied Intelligence* (Córdoba, Spain, June 2010).
- R-Moreno, M. D., Castaño, B., Carbajo, M., Moreno, A., Barrero, D. F. and Muñoz, P. Multi-agent intelligent planning architecture for people location and orientation using RFID. *Cybernetics and Systems*, 42, 16–32, (2011).
- Muñoz, P., R-Moreno, M. D. and Martínez, A. A first approach for the autonomy of the ExoMars rover using a 3-Tier architecture. In Procs. of the *the 11th ESA Workshop on Advanced Space Technologies for Robotics and Automation* (Noordwijk, The Netherlands, April 2011).
- Muñoz, P. and R-Moreno, M. D. Cooperative systems in mission planning. In Procs. of the *12th ESA Workshop on Advanced Space Technologies for Robotics and Automation* (Noordwijk, The Netherlands, May 2013).
- Muñoz, P. and R-Moreno, M. D. Deliberative systems for autonomous robotics: a brief comparison between action-oriented and timelines-based approaches. In Procs. of the *1st Workshop on Planning and Robotics* (Rome, Italy, June 2013).
- Muñoz, P. and R-Moreno, M. D. Model-Based Architecture on the ESA 3DROV simulator. In Procs. of the *23rd International Conference on Automated Planning and Scheduling Application Showcase* (Rome, Italy, June 2013).
- Muñoz, P., Cesta, A., Orlandini, A. and R-Moreno, M. D. Toward a test environment for autonomous controllers. In Procs. of the *5th Italian Workshop on Planning and Scheduling* (Turin, Italy, December 2013).
- Muñoz, P., Cesta, A., Orlandini, A. and R-Moreno, M. D. First steps on an on-ground autonomy test environment. In Procs. of the *5th International IEEE Conference on Space Mission Challenges for Information Technology* (Maryland, USA, September 2014).
- Muñoz, P., Castaño, B. and R-Moreno, M. D. Simulation of the hexapod robot PTinto walking on irregular surfaces. *International Journal of Simulation Modelling*, 14, 1–12 (2015).
- Muñoz, P., Cesta, A., Orlandini, A. and R-Moreno, M. D. The On-Ground Autonomy Test Environment: OGATE. In Procs. of the *13th ESA Workshop on Advanced Space Technologies for Robotics and Automation* (Noordwijk, The Netherlands, May 2015).

- Muñoz, P., Cesta, A., Orlandini, A. and R-Moreno, M. D. A framework for performance assessment of autonomous robotic controllers. In Procs. of the *the 3rd Workshop on Planning and Robotics* (Jerusalem, Israel, June 2015).
- Muñoz, P., Cesta, A., Orlandini, A. and R-Moreno, M. D. Evaluating autonomous controllers: an initial assessment. In *Procs. of the 6th Italian Workshop on Planning and Scheduling* (Ferrara, Italy, September 2015).
- Muñoz, P., R-Moreno, M. D. and Barrero, D. F. Unified framework for path planning and task planning for autonomous robots. *Robotics and Autonomous Systems*, 82, 1–14 (2016).

Chapter 2

State of the art

In this chapter, first, we introduce the state of the art in autonomous or plan-based controllers. Next section introduces the terminology associated to such systems and a review of the most representative ones. Following, a review of different approaches that aim to evaluate autonomous controllers is provided. Then, in sec. 2.2, we focus on the path planning problem, which is a required capability for autonomous controllers when dealing with mobile robots. Particularly, we first analyse heuristic search algorithms for path planning over flat surfaces. Following, we introduce those path planning algorithms that work over more realistic terrains, i.e., considering the terrain relief and other characteristics. Next, we present different techniques to integrate task planning and path planning, which aim to improve the solutions provided by the P&S system of an autonomous controller. Finally, we introduce approaches for autonomous controllers evaluation.

2.1 Autonomous controllers

The advances during the last decades in AI and robotics systems have resulted in an exponential growth of the number of robotics products involved in our lives. Not only in the industrial environment, but also in medical, home and recreational tasks. The list of applications is increasing each day and robots are conquering places unreachable for humans.

Deep sea, space or planetary surfaces are often too expensive (or even impossible with actual technology) to be explored in-situ by humans. However, robots have proven good skills in this topic. Within the growing complexity of the components to control, the tasks to perform and the environmental conditions, a robotic controller must deal with an important set of constraints and requires a high level of autonomy and decision making capabilities to safely operate. Combining hardware designs with AI controllers enables to deploy autonomous robots in those extreme environments.

One of the best known robots in the AI field is Shakey [138], that was built in the AI Center at the Stanford Research Institute (SRI) in 1966–1972. Its objective was to provide a platform to test new ideas, in order to design a real-time controller for a robot able to interact with a complex environment [136]. During their work, the researchers identified three important abilities that are required for autonomous controllers:

- **Problem solving:** this is usually an off-line task (in the sense that this task is performed prior to act) until recent studios in which appears the concept of reactive planning. The problem solving is commonly called planning, and represents the ability to generate a sequence of actions (plan) to reach a specific goal(s). In order to generate a plan, the robot must know how to apply the actions and what are their effects.

- **Modelling:** a model of the world defines both the environment in which the robot is and how it interacts with the world by means of actions. In this regard, the robot can change the environment as a consequence of its actions, but also, others agents can modify the environment as well. Then, the model has to be dynamic to reflect the environment status and must be properly updated to be accessed by the planning system. Moreover, if the control software includes learning processes (e.g., from past actions), it is possible to include new information in the model that is acquired during the execution.

- **Perception:** to retrieve information from the environment, a robot requires a set of sensors. The capabilities of the sensors depend on the environment and functionality of the robot. Typically, the sensors required are those that enable to capture the execution results of the actions performed, so it is possible to track the plan execution.

Camarinha-Matos [25] includes a fourth ability: **communication**, defined as a high level interaction using natural language or graphics between the robot and other agents in the environment, such as humans or other robots. This is an important functionality for those systems in which the Human Machine Interface (HMI) is relevant: autonomous controllers have enough power to keep the integrity of the robot in most of the unexpected situations. However, there are still some challenges for these systems that actually require human intervention, e.g., the opportunistic science detection or recovery from non-nominal states.

These capabilities entail different requirements to achieve full autonomy. Nevertheless, autonomy can be met in different levels: from safe execution of pre-planned commands to autonomous on-board planning. In this sense, the European Cooperation for Space Standardization (ECSS) [47] defines four autonomy levels as stated in table 2.1. In this dissertation we are focused on the goal-oriented autonomy, i.e., level E4.

On E4 controllers, commonly called plan-based or autonomous controllers, there is an entity on top of the control hierarchy called deliberative or planner. For deliberating it uses two inputs: the domain and problem. The domain provides a high level abstraction model of the environment, the robot capabilities and their interactions. The robot capabilities are enclosed into actions that have to be temporary bounded, so it will be possible to estimate the temporal behaviour during execution. The problem includes initial information about the environment status and the goals, which are set by the user.

Given a domain and a problem, the deliberative can generate a plan by means of AI P&S techniques. The plan provides a sequence of actions that, from the current state and maintaining the constraints defined in the domain, guides the platform to

Table 2.1: Mission execution autonomy levels (extracted from [47]).

Level	Description	Functions
E1	Mission execution under ground control; limited on-board capability for safety issues.	Real-time control from ground for nominal operations. Execution of time-tagged commands for safety issues.
E2	Execution of pre-planned, ground-defined, mission operations on-board.	Capability to store time-based commands in an on-board scheduler.
E3	Execution of adaptive mission operations on-board.	Event-based autonomous operations. Execution on-board operations control procedures.
E4	Execution of goal-oriented mission operations on-board.	Goal-oriented mission re-planning.

achieve the goals defined in the problem. In this regard, the plan represents how the platform must behave to achieve the goals. This entails the actions to be carried out by the platform and the temporal scope. Thus, the deliberative provides a planning horizon, i.e., an estimation of the time in which the plan will finish.

The execution cycle usually follows the sense-plan-act cycle (see fig. 2.1). Before generating the plan, the system must capture the current status of the environment/platform. Then, the plan is generated and executed. To do this, the deliberative is integrated with the lower layers by means of an executive layer. This level decomposes the actions of the plan into executable commands for the functional level. Notwithstanding, other approaches interleaves the two upper layers (deliberative and executive) into a decisional layer, in which planning and execution are highly coupled, sharing a knowledge base. Independently of the P&E schema, it is required to synchronize information between layers. For instance, each command planned shall be dispatched to the functional layer. This may imply to translate the data between high level abstractions (deliberative) and raw data (platform). In any case, how the different layers are connected has a great impact in the architecture design.

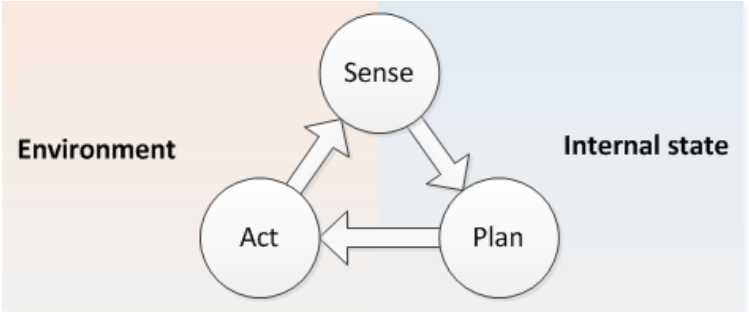


Figure 2.1: The sense-plan-act cycle.

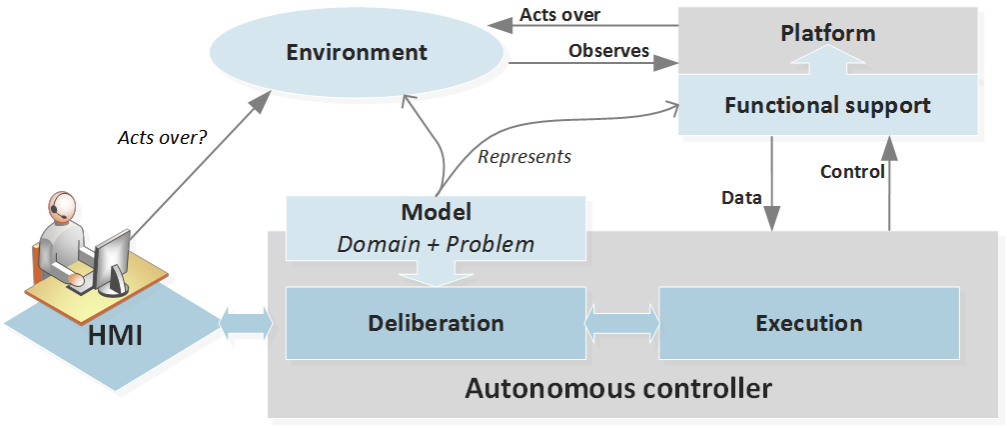


Figure 2.2: General vision of an autonomous controller and its interactions.

Figure 2.2 summarizes the previous discussed topics. Notice that, in some environments (such as industrial ones), the human can modify the environment (or even other robots). In this way, the autonomous controller provides planning via a deliberative or planning system using a model, the domain, that represents the environment and platform, and a problem, that includes the robot goals. By its side, perception is given by the robotic platform that is controlled by the execution system. Then, the human can remotely operate the system or include goals by means of HMI and different autonomy levels. In order to evaluate the performance of the architecture, it is required to take into account the layers hierarchy and how they are connected, but also the model and the P&S techniques employed.

Focused on the deliberative capabilities, we can follow two approaches as discussed by Agre and Chapman [2]:

- **Plan as program:** plans are built from a set of parametrized primitives, using a set of composition operators. Typically this paradigm is followed by classical planning; there is a large number of planners that, given a domain and problem, provide a set of actions that modify the environment in order to reach one or more desired objectives. This implies that another component of the architecture must be able to read the actions and the parameters in order to complete the task imposed by the planner.

- **Plan as communication:** in this paradigm the planner does not provide a solution to the problem, it gives a set of guides about which activities could be required to reach the goals. So, this requires another entity with interpretation and improvisation capabilities that uses these guidelines in the correct manner. The only requirement for a planner that uses this paradigm is that the output must contain some information that is useful for the other components.

Also, it is possible to differentiate two common designs to model the environment for the planning as program approach (since this is the most used approach, we simply refer to it as planning) as R-Moreno et al. [152] differentiate:

- **Action-oriented:** it uses predicates logic and the world is modelled as an entity that can be in different states. The domain specifies actions that can be performed to change the state of the world and when they are applicable. The objective is to find a sequence of actions that, from an initial world state, through applying successive actions, the system achieves a desired goal state. This approach is followed by a large number of planners such as Stanford Research Institute Problem Solver (STRIPS) [51] or those who use the PDDL [57, 107], among others.

- **Timelines-based:** it is based on the first order logic. It represents the world in terms of functions that describes the behaviour of the system from a time perspective: a timeline is a logical structure used to represent and reason about the evolution of an attribute over a period of time. Rules must be defined to specify how the timelines can change, in order to obtain a sequence of decisions from the planner that bring the set of temporal functions to a final state in which a set of constraints are satisfied. There is a growing number of planners based on this approach, and we can mention the Domain Definition Language (DDL) and Problem Definition Language (PDL) [30] as a modelling formalism for this kind of planners.

However, the deliberative layer itself is not enough to control a robotic platform. Integrating the P&S system with other layers to deal with the sensors and actuators requires to define the behaviour of the autonomous controller from a global perspective. In this regard, the global behaviour can be inspired by the nature to find solutions. A first way is to emulate the behaviour of *simple* organisms such as insects. They are behaviour-based and demonstrate intelligence into a reactive schema in which the current state of the environment guides the next action to execute, following a sense-act cycle (i.e., there is no P&S system). Controllers for robotic systems using these ideas are modelled as finite state machines since the behaviour is deterministic. There is a program based on rules that reacts to the current state of the world and defines the behaviour to follow, discarding models of the environment or predictions about future states. These systems are called *reactive controllers* and will be discussed in sec. 2.1.1.

More recently, systems have appeared that try to emulate the swarm intelligence that some kinds of insects exhibit: combining a group of agents with simple reactive behaviours, the whole system could demonstrate intelligent conducts and achieve more complex goals that the attainable by only one agent. A successful example of this system is the Swarm-bots [43]. In this dissertation we are focused on single robotics platforms, and thus, this approach is out of our scope.

Another approach to define the behaviour of autonomous controllers is to take the human as the main source of inspiration for, or even try to model human cognition. These systems have become known as *cognitive systems*. As described by Vernon [170], a cognitive system is that who *"exhibits effective behaviour through perception, action, deliberation, communication, and through either individual or social interaction with the environment"*. This implies, in general, that a cognitive system must be able to operate under unknown circumstances or events for which the system was not implicitly designed. There are two main cognitive paradigms:

(i) cognitivism based on the representation and processing of symbolic information, which is capable of handling complex representations of the environment, and (ii) emergent systems, focused primarily on principles of self-organization.

Langley [96] identifies what are the capabilities that a cognitive architecture should support. These are basically (i) a long-term memory to store relevant information about beliefs, knowledge and goals of the system; (ii) well-organized mental structures that are representation of the contents of the memory (often related to the HMI); and (iii) functional and learning processes which operate on the mental structures. However, depending on the system purpose, a cognitive system must support abilities such as recognizing and classifying elements in its environment, predicting its actions effects or reasoning about how to solve unexpected situations. Different cognitive approaches will be presented in sec. 2.1.2.

However, there is an important gap between these two approaches: reactive systems live in the present and act fast without taking into consideration what could happen in the future. Instead, the cognitive systems manage and deliberate over complex models of the world, so it is possible to consider past actions to learn and to make predictions about future states, which allows these systems to perform more complex tasks. Nevertheless, deliberation is a time consuming task: in a dynamic environment using much time to generate a plan can invalidate it due to changes during the process, which is related to the problem of keeping the world model updated.

An intermediate proposal between reactive and cognitive systems that try to keep the goodness of both approaches are the *hybrid or layered architectures*. The idea behind is to define a set of layers, each one with a different level of abstraction and scope. They are interconnected and combine different levels of deliberation and reactive capabilities, aiming to obtain an intelligent behaviour following the sense-plan-act cycle. The most common schema is the 3T architecture [137] in which the higher layer corresponds to the deliberative system, the middle layer is an execution system and the low layer corresponds to the hardware abstraction of the platform to control. Each layer uses a different model so there is a lack of integration between components and a clear separation between deliberation and reaction. Various architectures based on this approach will be described in sec. 2.1.3.

Some layered architectures have evolved into *multi-agent systems* based on the engineering approach *divide and conquer*. The idea is to split the problem into sub-problems that are interconnected. The architecture is described at a high level, defining the agents that will be part of the system, their roles, the interactions between them and the resources they need under the temporal restrictions imposed by the domain. Each agent could be reactive, deliberative or both, appearing the concept of reactive planning: deliberation systems that are closely coupled with reactive behaviours. Usually those planners work over a small part of the domain, using replanning schemes to deliberate with small latency and with different temporal horizons in function of the controlled system requirements. Examples of these architectures will be presented in sec. 2.1.4.

In the following subsections we present a survey on these kinds of controllers for autonomous robotics. We are going to focus on the intelligent behaviour that these systems exhibit, remarking the deliberative process that they implement to reach the goal for which they are designed.

2.1.1 Reactive controllers

Reactive controllers are those whose intentions are predefined actions to perform when some conditions occur in the environment, without maintaining a model of the world, i.e., they are reflexive systems. The most cited approaches, discussed next, are Subsumption and RAP.

Subsumption [21, 22] is designed using a pure reactive schema based on different levels of confidence, being the base level the most reflexive (when the actuators are close to the sensors), and including each one more complex behaviours. The idea behind that comes from two sides. First is the scalability of the system. It is possible to implement the base level with a basic behaviour and add consecutively more levels with complex behaviours, without changing the lower ones. Second, and related to the last issue, higher levels can take the control of the system, reading and modifying the data flow of the underlying level. This implies that the behaviour of the system is subsumed to the orders of the higher levels.

Internally, a level is implemented as a finite state machine with the ability to perform simple functions, direct access to the sensors data and interaction capabilities (in an asynchronous way) with other levels in order to modify their internal state and outputs. Also, each layer has its own goals (for example, avoid obstacles or grasp an element), and the combination of the behaviours of all levels guide the system to reach the objectives specified for every level.

The concept of Reactive Action Package (RAP) [52] is proposed to act in an efficient manner in dynamic environments. This system uses a reactive execution schema in which actions selection depends only on the actual context (that is, the actual state of the world, the state of the controlled system and its goals) at execution time, which implies that the system does not have any prediction about future states. Thus, the system contains a world model that requires to be permanently updated in order to allow the reactive executor to choose properly the actions to execute.

A RAP is a former autonomous unit of the system. The behaviour of each one is described by three rules: (i) it decides the next action to execute based only on the current state of the world and does not predict following states, (ii) prior to confirm that it has achieved its goal, it must check all related sensors required to confirm the objective, and (iii) if the package is unable to achieve its goal, it implies that it tries every possibility without success, and thus, it does not know how to reach the goal in the current context. Each package has only one goal, so if multiple goals are required, different packages must be defined to accomplish them, and the packages pursue their goals competitively which each other. To select the package to execute, the system performs a search to find packages that can accomplish the specified goals within the current world state. When a package is executing, it could send commands to the hardware interface and updates the model with the data acquired from the sensors.

In order to achieve complex goals, the system requires cooperation between packages. The solution comes from using partially ordered networks of subtasks called task nets. This implies the sequential execution of different RAPs and every package can contain one or more predefined task nets to achieve its goal. To accomplish the

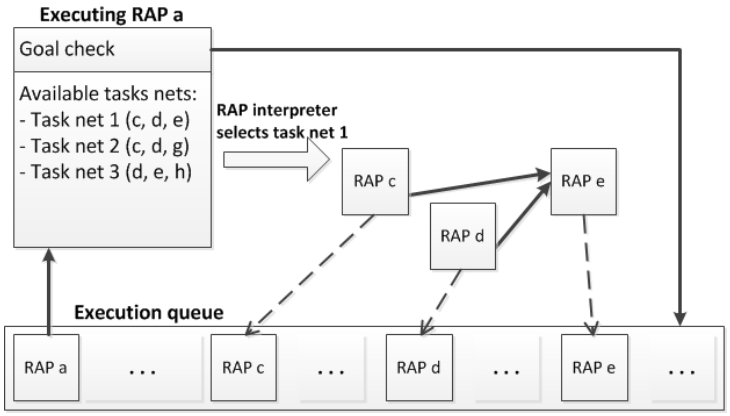


Figure 2.3: Execution of a RAP.

objectives there is a coordinated interleaving schema: when a package is executing, the packages of the task net are added to the queue (see fig. 2.3). Notwithstanding, the problems come with the interactions between task nets when they are executing in parallel and they modify the same state. The RAP execution deals with interactions checking the goals of a task net in the next way: if all the states attached to it still unchanged, then there is no interaction with other task nets. Otherwise, an interaction is detected and the task net is removed from the execution queue.

Due to the manner how a RAP is selected to be executed, it is possible to define opportunistic behaviours. This is done by defining packages that never meet their objective condition, so they are permanently in the queue, executing, and thus, checking the world state. When an opportunity is detected, that package will include a task net to achieve a new goal.

These systems have reactivity and performance. Rejecting complex models of the world and deliberation capabilities, allows them to select the next action to perform in a fast sense-act cycle. Also, it is easier for these systems to deal with opportunistic science; there is no plan to modify when new opportunities appears so the system simply tries to take advantage of them. However, it has important lacks: without a model of the world there is no possibility for deliberative and learning processes. As a consequence, it is not possible to reason about the future; goals are achieved by action rather than deliberation, being proactive systems, which have difficulties to solve complex tasks that usually require prediction to be safely and efficiently performed.

Moreover, the coordination in reactive controllers is problematic. For instance, in the Subsumption architecture sometimes it is not clear when a behaviour must be in a higher level than other. Also, there are side-effects on the level interconnections, every level increments the complexity and is dependent on the lower ones: when lower levels are modified, a complete redesign of all the architecture could be required. For RAP, the problem comes from the coordination between competitive packages in execution. When they modify the same state, only one package can continue its execution, without considering which one is closest to achieve her goal. This turns into some inefficiencies during the execution.

2.1.2 Cognitive systems

Cognitive systems are those who try to imitate the mental human processes that defines the intelligent behaviour. This implies problem solving and learning skills. In these systems the model of the world could be in continuous change, having special importance past events and future predictions to try to anticipate the world state. One of the ways to learn new knowledge about the world is to analyse the decision taken in the past and their results. But there are other ways to learn such as incorporate new information from external agents (such as humans) or by inference from the information contained in the domain. In this section we present four of the most relevant cognitive systems: Soar, ACT-R, ROGUE and ICARUS.

The Soar architecture [111] is an attempt to couple a minimal group of functional modules with a purely symbolic reasoning system. During years this architecture has received some improvements, being the latest version introduced in 2008 [93]. The basic of Soar is the symbolic planning over a long term knowledge represented as production rules, with a short term memory using a graph structure to represent properties and relations between objects. At this level, Soar provides a context dependent representation, updated via monitoring the environment or from the knowledge of the long term memory. So, the actions are produced by rule matching based on operators preference and utility in the current context, and thus, the action selection is dynamic and the rules definition can change over time, not only as function of the context, also for the learning processes in the long term memory.

The long term memory is responsible of the problem solving, which implies a translation between long term memory and short term memory. Soar decomposes plan sub-goals obtained by procedural reasoning into rules to fire the actions which achieve the desired objectives, and, also, it could represent reactive behaviours. The learning is made through a chunking mechanism, learning new rules when exists impasses. An impasse is arisen when there is not enough knowledge to select an action in the current context. In that case, the long term memory produces a new state in which the goal is to resolve the impasse.

Soar extensions rely on the long term memory and its associated processes for learning. One of these enhancements is the reinforcement learning that adds the possibility of including numeric preferences for the rules produced based on the context. Thus, in order to select an action, an epsilon-greedy search algorithm could be employed to select the next one to apply in order to maximize a reward function. This reward is intrinsic to other modules based on emotions, as a computational implementation of a specific appraisal theory [155]. Emotions and feelings are combined to modify the reward value, which affects directly to the reinforcement learning, increasing the learning speed. Also there are included two new memory modules: semantic and episodic. First one provides support to store and learn declarative facts about the environment that tries to improve the chunking learning mechanism. Second one takes snapshots of previous contexts in order to reason and learn from previous experiences.

Some of the enhancements of Soar are inspired in elements or functionalities provided by the Adaptive Control of Thought-Rational (ACT-R) architecture [5].

As Soar, this architecture has also evolved from its first version [99]. ACT-R is inspired in the human brain and has two main modules (see fig. 2.4) in charge of the cognitive and deliberative process (such as the human brain and its two hemispheres). These two modules are connected with a core production system that coordinates the system with other modules into the lower layer, in charge of actuators control and perception. The modules are only connected with the coordination core and they use buffers with only the required amount of information that the core can manage. This information could be modified by the core to make requests or include new knowledge, being the changes triggered by patterns recognition into the buffers. The cycle of execution in ACT-R is managed by these changes in the buffers: the buffers hold representations of the external world (low level modules) or knowledge and intentions in the high level (deliberation and learning modules). When the modules detect changes in the buffers, they perform the required actions or deliberative processes.

While visual and manual modules (those in charge of perception and action respectively) are coupled with the external world, the intentional and declarative modules are responsible of the internal state and procedural problem solving and the long term declarative memory. The declarative memory defines elements called chunks (a kind of associative memory) and the procedural memory define production rules and a schema for learning new rules. The cognition of the system is determined by the way of how these memories are connected. The central module or core is the one that, taking elements from the buffers of both memories, matching a set of rules by pattern recognition, discarding from these sets those who are not free of conflicts and, then, executing the actions making the necessary changes in the low level modules buffers.

Previous systems relies on specific algorithms and representations. However, the ROGUE architecture [68] (see fig. 2.5) is focused on interleaving planning and execution, with a replanning schema when actions fail or to generate new plans to attend user requests or opportunistic changes in the environment. As the architecture

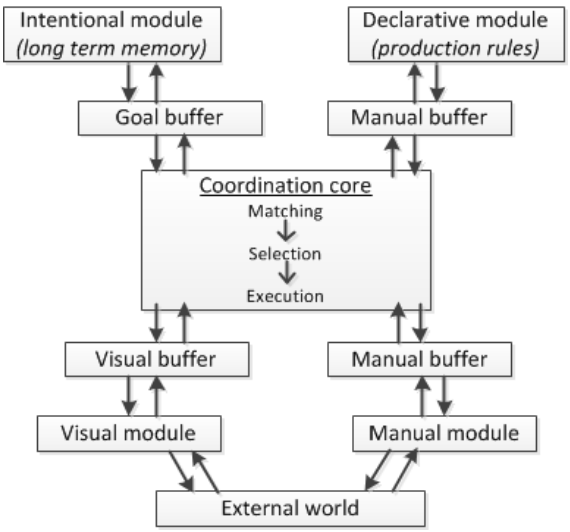


Figure 2.4: Organization of the ACT-R architecture.

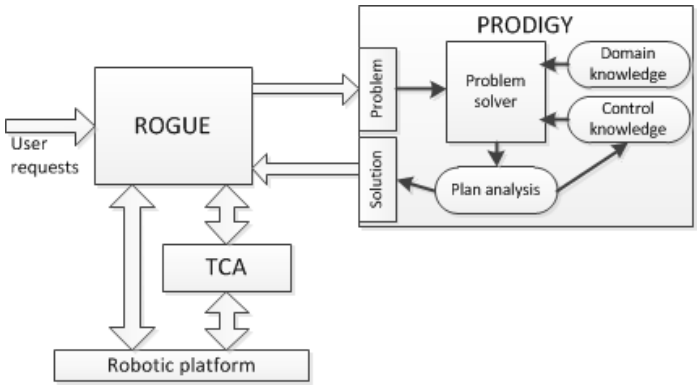


Figure 2.5: Conceptual view of the ROGUE architecture.

is intended to deal with asynchronous requests, the planner must manage priorities in order to be efficient. This implies that a current plan could be suspended in order to attend new priority tasks using a modifiable metric for the plan search. To do this, ROGUE uses the PRODIGY planner [27]. It is a general problem solver focused on learning process and based on a extended version of the STRIPS [51] language. Using a set of operators, the search engine of PRODIGY is able to obtain (if it exists) a sequence of operators to reach a desired set of goals from an initial state, by growing the plan incrementally, trying not to modify the previous plans if possible. To decide which operators to use, PRODIGY exploits a set of rules to choose the most preferred candidates. Using these rules the system improves both, the search performance and the goodness of the solution, while also it is possible to learn new control rules. This is possible in two ways: (i) learning planning domain knowledge to improve search efficiency and, (ii) learning control knowledge to improve plan quality. PRODIGY was expanded during years and the latest version [167] employs a non-linear planner which enables the system to fully interleave plans, exploiting common sub-goals and addressing issues of resource contention.

For action execution ROGUE follows the action-oriented approach. When the plan is generated, it takes every action of the solution and decomposes it in low level commands. During the execution, the system monitors the preconditions, the current state of execution and postconditions in order to keep updated the planner knowledge, and thus, allowing learning from the action monitoring. It is also valuable for replanning in the case of failures: the planning cycle is interleaved with the execution of actions, so, the time spent in replanning is reduced, increasing the efficiency of the system. To control the action execution and monitoring, ROGUE has two possibilities: controlling directly the robot, or using Task Control Architecture (TCA) [159] as the functional layer to support more complex processes.

These architectures are mainly focused on complex symbolic representation and management of knowledge, rather than the perception and execution. Trying to solve that, the ICARUS architecture [95,97] was developed with more emphasis on perception and reaction to the environment changes, while keeping learning and knowledge management to include new skills learned from the experience in an incremental manner as a human does. Following common human cognition theories, ICARUS

employs two memories: long term memory and short term memory. First one, as previous approaches, changes as a function of the learning processes, while second one is modified by the system beliefs and current goals to achieve.

The long term memory provides a general situation of the environment based on hierarchical concept decomposition, from complex concepts at high level, to simpler concepts in the lower levels. A concept could involve objects or relations between objects. Also, there exists knowledge about skills, being those similar to actions in first logic and presenting a structure that defines the conditions to apply the skill and the expected effects. These elements are expressed as concepts, and, like those, the skills can follow a hierarchical organization: a primitive skill could define an executable action or a decomposition into more skills. These elements are instantiated into the short term memory, in which we can found the perception buffer coupled with the status of the sensors and actuators, and memory associated to the current skills in execution, which are generated by inference from the long term memory and represents the goals, intended actions to perform and expected future states.

The execution in ICARUS follows a typical cycle that starts reading the state of the sensors, which produce perceptual elements that are associated by matching with long term concepts, that is, a bottom-up schema. Then, the architecture selects by searching the desired goals and decides which skills to execute based on the current context. During this selection process, ICARUS performs a balance between reactivity and persistence. When different skills are eligible, the system tries to execute high level skills that are currently executing and also, those which have a similar context in terms of required conditions to be executed and their effects. ICARUS performs means-end problem solving to select the skills to execute in function of the current goals, but, it does not give a complete plan. This could lead to suboptimal plans, however, it requires less effort to obtain a plan, and thus, the deliberative process requires less time. This relies positively in the reactive behaviour and a better solution than replan everything when an impasse is detected.

When these partial plans lead the system to undesirable states, ICARUS includes new knowledge learned from the impasse in order to avoid selecting the same skill in a similar context in the future. Also, it is possible to infer skills from the long term memory, constructing new skills to achieve goals that could be decomposed into sub-goals solvable with known low level skills.

Cognitive architectures put the focus on complex symbolic processes to generate actions which guide the system to achieve a desired set of goals. Within these symbolic processes, there is an important effort in learning through different schema to improve the plan quality or to deal with situations not predicted in the design of the system. Despite this represents a desirable quality to deal in uncertainty, the time required to work with these symbolism may imply that these systems became static in term that the environment could change faster than the speed of producing a new plan adapted to the new context. Although some work is performed to make these architectures more reactive, unexpected situations trigger learning and replaning processes, so, it is not usual to see these systems controlling robots which must deal with environments where the reaction time is fundamental to keep the integrity of the robot.

2.1.3 Hybrid architectures

The most common decomposition of hybrid architectures corresponds to a 3T schema in which each layer has a designed functionality. Coupled to the hardware is the functional layer, providing access to the sensors and actuators. At the top of the architecture there is a planner, capable to manage complex abstractions of the world. Between both, there is an execution system that takes the plan generated by the deliberator and executes the actions that it prescribes [137]. The idea is to take the goodness of deliberative capabilities (such as cognitive systems) to model the world, learn and generate high levels plans, while the lower level contains reactive behaviours and sensing capabilities to provide primitive functions and data to the upper layers.

Although it seems clear what are the capabilities that the functional layer must provide (safe access to the hardware actuators and sensors), the division between the deliberative and the executive functionality is not clear at all. The frontier depends on various design challenges, such as how to solve the different granularity an increasing intelligence of the layers (from reflexive, to procedural, to deliberative) or the inconsistencies that could appear on the separate models that the system manages (some information is duplicated with different levels of abstraction). Trying to solve those issues, some systems evolved into a two layered architectures in which deliberator and executor are coupled. In this section we present eight of the most cited in the literature hybrid architectures: TCA, AuRA, ATLANTIS, SSS, the Bonasso 3T architecture, the LAAS architecture, RA and CLARAty.

One of the first efforts to make a general framework for robotics control using a layered schema is TCA [159, 161]. It is designed to provide autonomous capabilities using deliberative components which act during the normal execution, while reactive components are in charge of responding to unexpected situations using a distributed architecture. To do this, the TCA provides a framework that does not implement particular behaviours, it gives to the designers a set of mechanisms to support particular implementations of the system.

TCA is based on a central module in charge of the coordination of the other concurrent processes, called modules, which are specific for every application. The communication between modules occurs via message passing (routing through the central module) allowing synchronous and asynchronous communication. There are different kinds of messages to provide the support to allow data transfer, monitoring, commands and goal interchange between modules. Using the last one, it is possible to define and send sub-goals for different modules in order to achieve task decomposition (in a similar manner to the RAP system). Also, the messages system could be used for resource management: a resource has a defined capacity in terms of the number of messages that the resource can manage simultaneously.

To deal with the subtask hierarchy, the central module of TCA constructs trees to manage the executable commands and monitor messages, as shown in fig. 2.6. Also, it is possible to define temporal relationships (such as sequential activities) or constraints (expressed as temporal intervals) to determine when to dispatch the messages. Instead, the TCA does not provide any deliberative or reactive system, the modules and their interconnections guide the execution of the architecture. In

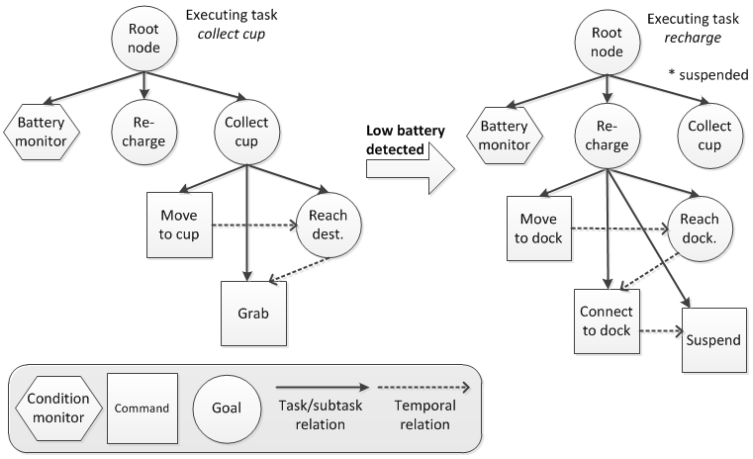


Figure 2.6: Example of TCA subtasking and state monitoring.

order to provide deliberative capabilities, an external planner must be included in a module, and other modules must be implemented to provide reactive capabilities. The TCA provides common control constructs, including distributed communications, hierarchical task decomposition, temporal constraints to coordinate subtasks, resource management, monitoring, and exception handling.

There are some implementations of TCA for multiple robots using different technologies. For instance, the Ambler six-legged planetary exploration robot [162] is implemented using six modules with the Gait planner [173] as the deliberator. It uses the terrain and kinematic constraints to plan the movements for each leg and the body in order to decide safe movements. The other modules are in charge of the perception and map generation, and to provide specific information about slopes and obstacles to the other modules. Other remarkable work is the continuous miner [158], which uses a path planner and a task planner. First one implements the A* algorithm [69] to decide the route and the second takes advantage of the subtask decomposition of TCA to determine the tasks required to advance to the coal face.

TCA highlights one important question about the layered architectures. Each layer can be implemented using different technologies and abstraction models. The interactions between layers and their coordination are an important and not trivial aspect. The importance of the design resides in different questions: how to exchange data between layers, synchronize their operations or select which layer of the architecture is in charge of determining the tasks. TCA chooses a distributed schema with centralized control and standard messages. This enables the architecture to include new modules without modifying other parts, but the central control is a potential bottleneck. Complex systems could require a high bandwidth, especially in order to deal with optical sensors such as video cameras.

On the early 90s appeared the Autonomous Robot Architecture (AuRA) [7]. Its objective was to provide autonomous mobility with reactive capabilities and a modular design. This architecture has evolved [8] following a 3T schema with two components to provide the deliberative capabilities (the mission planner and the spatial reasoner), a schema controller for the hardware abstraction and a plan sequencer

between both layers. The deliberative is divided into two elements being the mission planner focused on providing the HMI in order to establish the high level goals of the system, while the spatial reasoner is in charge of the navigation, employing the A* algorithm over a meadow map [6] to generate the routes to follow, or the Router planner in newer versions [66]. In order to translate these paths into executable commands by the functional layer, the plan sequencer translate it using a finite state machine with temporal behaviours triggered by events [9]. In that point, the deliberation process ceases and the behaviours specified by the plan sequencer are sent to the robot to be executed. The functional layer is responsible of executing and monitoring the commands, which can operate asynchronously, and triggers reactive behaviours which are rule based and included machine learning capabilities for the adaptation of the motor behaviours [33].

The progress monitoring of the commands in execution are essential to guarantee the success of the current plan; if the system detects that there is no progression in the execution, it follows a bottom-up schema to try to solve the situation. First, the spatial reasoner is invoked to try to re-route the robot, using the last data acquired of the environment. If no solution is found, it tries to obtain a new route for the whole mission. When the problem is not solved, the mission planner is informed and it must provide a new plan or abandon the mission.

The 90s continues with the A Three-Layered Architecture for Navigating Through Intricate Situations (ATLANTIS) [61] (fig. 2.7 center). It is focused on providing an asynchronous and heterogeneous system guided by a deliberative based on the plan as communication paradigm. For constructing the architecture, the functional layer provides a network of interconnected functional modules. These modules are described using A Language For Action (ALFA) [60] and are formed by two elements: (i) channels used to communicate data between modules, with high-level systems, to acquire sensor data and to send output commands to the actuators, and (ii) the associated transfer function which computes a set of outputs from a set of inputs, using a data flow function or a state machine.

The executive is a modified version of the RAP system that includes support to resource protection using semaphores. This allows ensuring that two packages which interfere with each other are not enabled simultaneously. The control of the architecture resides in the RAP system, which is able to interrupt activities in a controlled manner that is especially useful to stop the deliberator. This one is in charge of maintaining a database accessible to the executor with guidelines about how to reach the goals of the system. The deliberative computations are requested by the executor, and consists of a set of Lisp programs that implements traditional AI algorithms. This set is not fixed, and can be modified to include different planners, for example, the work of Miller [109] that supports the use of resources, deadlines and travel time for path planning using a Lisp planner designed to search over totally-ordered plans in order to detect ordering dependent interactions. Also, as the control of the system resides in the RAP system, it is possible to remove the deliberative layer.

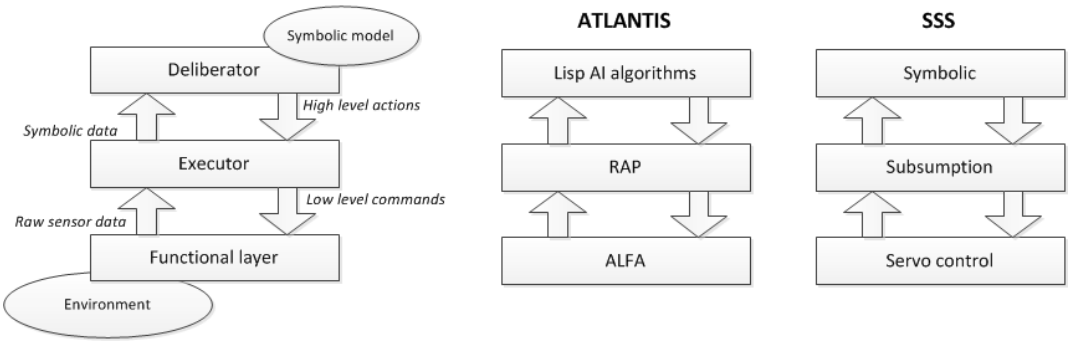


Figure 2.7: Three layer general schema (left), ATLANTIS (center) and SSS (right).

Close in time appeared the Servo, Subsumption, Symbolic (SSS) architecture [34] (fig. 2.7 right), which is very similar to ATLANTIS. It follows a 3T schema with an important difference respect to ATLANTIS: in the SSS the deliberator is on the control loop. The architecture works guided by the symbolic representation of the deliberator, which generates sequence of actions connected through basic temporal relationships. These actions switch on or off different behaviours in the execution layer, which is implemented using the Subsumption system. To correctly decouple the deliberator and executor (first works over discrete time and space, and second uses discrete space, but continuous time), the architecture employs a *contingency table* to keep the efficiency of the system. The entries of this table reflect possible future situations and a one-step plan to maintain the coherence with the current status. To do this, the system permanently monitors the defined situations, and when one occurs, the event is propagated to the Subsumption system modifying the current execution, and also, the deliberator generates a new updated table. Finally, the functional layer is responsible of controlling servo-like robots.

Also, the 3T architecture of Bonasso is presented in that period [17, 18]. This work aims to obtain a synchronous architecture based on competence levels that are similar to the Subsumption levels. However, Subsumption evolves in an asynchronous manner in function of the environmental changes, while the Bonasso architecture works in continuous cycles that guarantees that there is synchronization between the input from sensors and output in the actuators one time per cycle. The organization is structured over three layers, being the lower one attached to the hardware and with a fast cycle time to read sensors and to perform the actions required by the upper layers. The middle one has more latency execution, and it is in charge of more complex behaviours and the translation between the symbolic representation of the environment of the upper layer and the lower one. Finally, the upper layer or deliberator is the one that gives the sequence of actions to reach the goals of the system.

To implement the architecture, the Goals As Parallel Program Specifications (GAPPS) [88] language is employed. This provides a framework to define reaction plans at different levels of competence which are task-driven with sub-goaling decomposition using goal regression to generate plans. In fact, GAPPS generates layered circuits which are similar to the Subsumption schema, but providing more complex

representation to deal with deliberation and to take advantage of prediction about the action effects. The plan selects the competences to execute as actions in the classical meaning: when a current context matches with the specified preconditions (by continuously monitoring the environment), the competence could be instantiated into an action to execute over a particular set of objects. To define the three layers of the architecture, the code is arranged into groups of circuits with different levels of competence, which could define reaction plans (the competence selected is a function of the current context and goals to achieve) or different behaviours with priorities that are valuable for reactive components. Last case is similar to Subsumption: high priority competences could inhibit other with lower priority, using constraint passing between layers and combination algorithms.

At the end of the 90s appeared the first version of the autonomous controller developed at the Laboratory of Analysis and Architecture of Systems (LAAS) [4, 80]. They use a classical three layers approach to allow planning and reactive behaviours for rovers-like robots, using the sense-plan-act with different levels of granularity and symbolic representations (see fig. 2.8). But, in this architecture, the layer competences are quite different from the classical decomposition: the executive is a purely reactive system, while the deliberator (also called decisional level in this architecture) is composed on a procedural executive coupled with a temporal reasoner. As other architectures, this one has some evolution, especially focused on the executive and deliberative layers.

The functional level is composed of a set of modules that contains the hardware functionality (for sensing and acting) and the number and organization depends on the underlying hardware, being a network of modules with different responsibilities. The modules are implemented using Generator Of Modules (G^{en}oM) [104] a framework to generate reliable and distributed control modules. This framework appears in the first deploys of the architecture and remains into newer ones, being updated with newer functionality [105]. In the architecture, several modules are produced to provide different navigation modes (with specialized algorithms) and access to the scientific load, such as cameras.

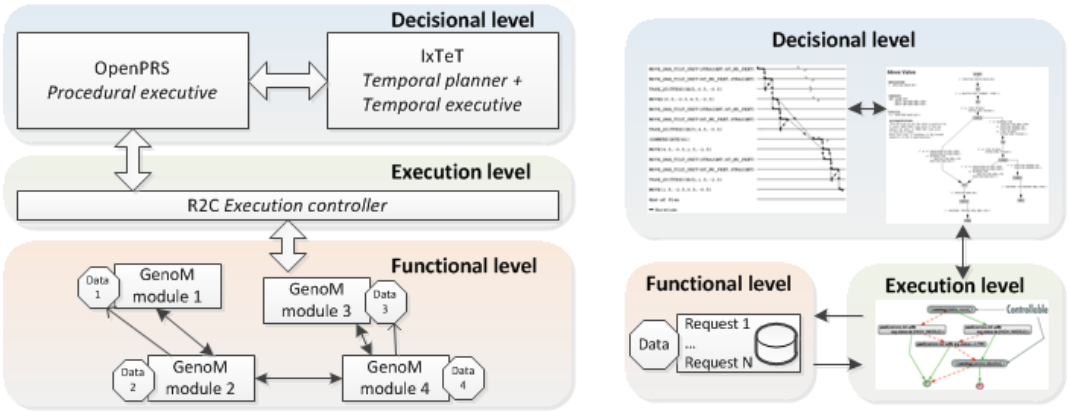


Figure 2.8: Conceptual vision of the LAAS architecture (left) and more detailed model (right).

The executive level in this architecture is purely reactive and fills the gap between the symbolic representation and computationally costly requirements of the deliberator, and the lower level that manages raw data. Initially, the executive employed was Kheops [4], which using a formal language produces automatically an automaton. It receives sequences of actions to execute from the deliberative and, depending on the current state and a set of rules, it is able to decide when to send appropriate commands to the functional level. To avoid conflicts between different modules the executor uses a priority schema to delay command execution or to interrupt a module, while conflicts in the same module are managed locally. In newer versions [80] the execution control level employs Request and Reports Checker (R^2C) which is intended to provide a safe control to avoid entering into unsafe states. In a general case, the upper layers will not produce actions that leaves the system to dangerous states, at the expense of increasing the complexity of the system (especially when dealing with parallelism), and requiring a mechanism to ensure the task refinement and controllability of the system. To provide these functions R^2C captures all events between the deliberator and the functional layers to update its system representation and to check if there exist any inconsistency, and, if any, the R^2C will act, rejecting requests or suspending processes, for instance. Otherwise it allows the system to continue in the normal way. To define the constraints of the system (i.e., safe and unsafe states) there is a language formalism called ExOGen [147]. Internally, the system works with Ordered Constrained Rules Diagram (OCRD) [81] which is similar to Ordered Binary Decision Diagrams (OBDD) [23] (to express first order logic formulas) adding the constraints. The R^2C creates structures from these formulas which define what states of the system are controllable, and, also, they can be validated using formal techniques.

The decisional level is in charge of generating the plan and supervise its execution, and, as difference of others architectures, the planning is concurrent with reactive execution, that is, this level is remaining to incoming events from lower layers. This is a complex task in terms of synchronization and balance between the two different behaviours. To do this, the decisional layer is composed of two elements: a procedural reasoner and a temporal planner. These elements are the Procedural Reasoning System (PRS) [78,79] (or its open source version, Open Procedural Reasoning System (OpenPRS)), in charge of the procedural reasoning, and the IxTeT planner [92], which produces a plan to achieve a set of goals defined by the user.

The IxTeT planner uses a temporal representation based on state variables and a Partial Order Casual Link (POCL) planning process to generate flexible plans, based on Constraint Satisfaction Problem (CSP) (to deal with numeric and atemporal constraint variables) and the time relies on a Simple Temporal Network (STN) representation [37] (to handle the time points and their constraints). The representation of the world describes a set of attributes which are multi-valued functions with time and resources. The evolution of the attributes is guided by a set of propositions which allows the changes of the values. In this representation, similar to the timelines approach, an action is a set of events that describes the evolution induced by the action in the different attributes. Using a POCL framework, the search is guided generating a tree of partial plans which could have flaws (inconsistencies, pending sub-goals or resource conflicts). The planning process consists of detecting

the flaws and resolving them, selecting one, then inserting a resolution for the flaw into the partial plan, and repeating the process until there are no flaws in the plan. The temporal executive of IxTeT [100] introduced in the latest version of the architecture performs a control on how the plan produced will be executed, acting as a coordinator between the planner and the procedural reasoner. Also, it includes a HMI to allow the user to introduce new goals. So, in general, the temporal executive provides a reactive component that can analyse events and, if necessary, try to perform plan adaptation (taking advantage of the temporal flexibility of the plan) and, when it is not possible, abort the execution and replans from scratch.

The procedural executive is attached to the temporal executive and to the functional layer. PRS is designed to deal with parallel activities and to react in a bounded time to events. The role of the procedural executive is to decompose the actions from the high level plan into lower level commands which, in that case, are accepted by the G$^{\text{en}}$oM modules, and, in some cases, to perform recovery routines, in an asynchronously way. PRS maintains the current state of the robot and some facts about the world by performing a permanent monitoring, and, also, a library of procedures which describe a particular sequence of sub-goals, actions and test required to complete a goal. In fact, the execution is goal driven in function of the objectives imposed by the upper layer. The decomposition of the goal into a set of commands is specified into procedures that are selected dynamically in function of the current context.

Nearly in time, with the objective of reducing costs and generating a reliable control system for space operations, the National Aeronautics and Space Administration (NASA) created the Remote Agent (RA) [113]. This system is formed by a set of components that provide reliable deliberative and executive capabilities in a closed loop. The higher component, the P&S system, generates plans based on resource constraints and hard deadlines to ensure achieve the science targets. These plans are executed by an intermediate layer that provides robust and parallel plan execution through a real time control system that enables the operation of the hardware. The executive system is coupled with a model-based system that integrates reconfiguration schemes to overcome unexpected situations and to keep safety configurations of the platform.

The planner used in the RA is the Heuristic Scheduling Testbed System (HSTS) [112], which is also used for the ground operations of the Hubble Space Telescope. It is a timelines-based planner and employs DDL for the domain representation. For deliberation, HSTS uses a heuristic guided backtrack search, producing temporary flexible and concurrent plans. Such flexibility allows the executive to rearrange the plan execution inside a temporal bound. In fact, the planning process is periodic, generating plans for a given plan horizon. Moreover, the plans are not completely specified, i.e., the plan specifies *behaviour envelopes*, which are a set of possible behaviours. Then, the executive selects the best behaviour accordingly to the current status at the execution time.

The executive uses the Execution Support Language (ESL) [62] and exploit the plan flexibility by means of a hybrid procedural and deductive execution. In this regard, the execution is context-based, allowing the selection of actions to carry out analysing the current state of the environment/platform. The executive is coupled

with a reconfiguration component provided by the Livingstone system [174]. It uses a discrete model-based that defines the operational configurations of the platform. In this regard, multiple configurations could enable the consecution of a particular goal. As well, it provides reactive capabilities and reconfiguration scheme to overcome hardware failures.

The RA was exploited to control the Deep Space One in 1999 [16], being the first AI controller to assume the control of a spacecraft. The RA demonstrated that is able to operate the spacecraft not only in nominal operations, but also providing failure tolerant behaviours (using both reconfiguration scheme and on-board replanning) and event-driven events.

Following a similar concept of the LAAS architecture, Coupled Layer Architecture for Robotic Autonomy (CLARAty) [171, 172] goes further and presents an architecture based on two layers: the functional layer and the decisional layer, which couples the planner and the executive. The proposal runs along two axes as fig. 2.9 shows: as typical architectures from bottom to up increasing the intelligence (from reflexive, to procedural, to deliberative), while the other dimension represents the granularity of the system at each level. This tries to avoid the gap between layers, giving an architecture that contains different competences levels in the same layer. So, these two layers interact at different granularity levels: sometimes the functional layer provides only a basic service, and, other times, the decisional layer relies the responsibility on high level commands provided by the upper abstraction levels of the functional layer.

The functional layer is implemented as an object oriented hierarchy that encapsulates the system functionality, acting as a black box for the decision layer. The modular design gives different accessible behaviours, from primitive commands to local planners which perform operations such as trajectory planning or arm placement. These elements are instantiated in function of the system to control, and, the decision layer has access to all capabilities using a generic interface.

The decision layer is in charge of decomposing high level goals into smaller objectives and to decide how to execute them. The goals are represented as constraints states over time (using the timelines approach). The temporal planning specifies the task tree to achieve the goals, while the executive part of the decisional layer decomposes each task into a sequence of commands which interact with the functional layer, using the Task Description Language (TDL) [160] to guide the conditional

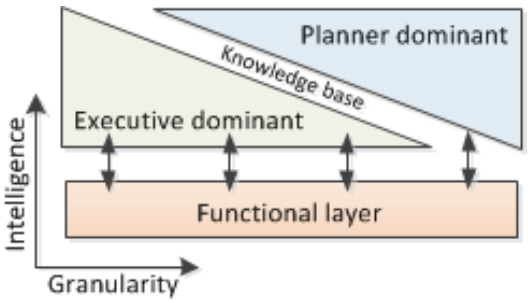


Figure 2.9: CLARAty architecture conceptual vision.

execution of the commands. The planning is done over two temporal horizons, using the resource usage prediction of the functional layer: one is for the immediate future in which the executive part is dominant, and following, the planner dominant for long term planning.

The design philosophy makes the difference about the hybrid architecture capabilities. Depending on the division and specialization of the different layers, the system could be reactive dominant or deliberative dominant, that is, the execution is mainly guided by the context or by the planner, respectively. Also, the scope of different layers and the interfaces which connect them is a complex topic: different abstraction models and granularity requires effort for translating the information that typically flows from the functional layer to the planner (sensor data) and in the opposite direction, from the deliberator to the robot (actions). This is function of the technologies employed and some approaches take advantages of different systems designed with interfaces and functionality to support these issues. However, there is also the synchronization of the system; planners are usually slow in reaction time, while functional layers could implement fast reaction routines, even reflexive in some cases. In general, the executor must deal with all of these questions, including translation support, reactive routines, and, sometimes, plan adaptation capabilities. So, these architectures require an important engineering effort, while keeping the focus on the deliberative capabilities to improve both, the performance of the system and the reaction time.

2.1.4 Multi-agent architectures

The multi-agent systems designed for robotic control have evolved from hybrid approaches. The architectural decomposition into different agents that could involve reactive, deliberative or both capabilities, follows a *divide and conquer* approach. Combining the capabilities of each agent it is possible to solve complex tasks by means of agents cooperation. In this section we present four systems: IDEA, ARMADiCo, the work of Woods et al. and GOAC. We provide a deeper description of GOAC as it will be used in the evaluation framework presented in ch. 7.

Probably, the most representative multi-agent architecture is the Intelligent Distributed Execution Architecture (IDEA) [10, 40]. The basis of IDEA is the unification of the planning and execution functionalities into the same framework, sharing a common knowledge base. In this regard, a controller based in IDEA is a collection of agents, each one with a particular purpose to achieve the mission goals. Then, the deliberation is distributed among agents. Using a single model for the environment/platform each agent deliberates over a subset of the model. Then, merging the solutions provided by the different agents, the full system achieves a solution. To do this, the agents are synchronized using a common language, exploiting the timelines-based approach. Each agent is responsible of a set of timelines over which it can deliberate. For the deliberation process, IDEA allows using different planners when some technical requirements are met. Notwithstanding, the most referred is the Extensible Universal Remote Operations Architecture (EUROPA) constraint propagation package [87], which is descended from HSTS.

The plans generated by the agents are stored into a plan database. This includes the current execution status, the generated plan for a specified temporal horizon and an immediate past state. There are a special agent that reads such database in order to dispatch the plan actions to the functional support. As well, it updates the current status of the database to reflect the execution progress. Then, during execution, the different agents can supervise the plan execution, so it is possible to exploit reactive planning scheme when the current status does not met the planned one. This enables fast response to unexpected events and very specialized agents in charge of potentially dangerous situations.

Focused on mobile robotics is the Autonomous Robot Multi-agent Architecture with Distributed Coordination (ARMADiCo) [82,83]. It is based on the principle of decentralized coordination, i.e., there is not a central coordination module. Then, the different agents have to coordinate their objectives among the others, while also pursuing the resources usage. For this reason, a key point in ARMADiCo is the division between individual and social intelligence. First one entails the required capability of the agent to achieve its own goals. Second one is required to enable proactive interaction with other agents, or even humans. Combining both intelligences, an agent is able to autonomously decide how to act and when, which includes decisional capabilities on other agents requests.

From a high level perspective, ARMADiCo behaves similarly to a 3T architecture. Notwithstanding, the deliberative, executive and functional levels are formed by one or more agents each. In this sense, the lower level corresponds to the perception and actuators agents, in charge of the access to the sensors and actuators. For instance, each sensor can be encapsulated into an agent. The executive level is formed by a set of behavioural agents, which defines the basic behaviours and reactive capabilities of the system. In this way, it is possible to include agents such as *GoTo* or *Avoid* agents to define mobility behaviours. The deliberation level is defined by cognitive agents. They provide the different AI methods to generate the solutions for the given goals. In this regard, task planning and path planning agents can be deployed. For instance, agents for task planning based on a procedural language (similar to PRS) or path planning agents employing A* and Dijkstra algorithms for path planning have been deployed. Finally, there are back agents to provide support for the overall multi-agent system.

A relevant aspect in ARMADiCo is how the decentralized coordination is carried out. First, depending on the situation, an agent could use a resource or could be a resource. Second, to share resources without a central agent that decides on the resource usage, each agent defines an utility function over each required resource. Then, conflicts in the resource usage are solved by means of peer-to-peer coordination between the agents that pursue such resource. The one with a higher utility function obtains the control of the resource. As well, the agent that is currently using a shared resource periodically informs others agents about its utility function, so it is possible that another agent takes control of the resource if the current owner utility decays.

One of the benefits of the multi-agent architectures is the possibility of implementing and deploying highly specialized agents in charge of particular tasks. In this direction is focused the work of Woods et al. [176]. They propose an autonomous

system that could be able to detect new science opportunities during the execution of the normal plan execution (called open-loop) in a rover-like mission. The open-loop execution implies high delays between the ground station and the rover to send the science schedule and to retrieve the previous acquire data. Accordingly, is possible that some interesting target that could be found during the traverse was missing. In this way this is not only required to identify the target, but to autonomous prioritize targets (both, detected and selected by ground team) and replanning taking into consideration time and power available, and to autonomously place the required instrument to analyse new targets.

To deal with the required deliberation capabilities, the autonomous system they proposed is based on the Time-line Validation and Control and Repair (TVCR) [177], a planning and scheduling system based on time-lines approach developed to support goal prioritization and plan adaptation for opportunistic science. Also there are other two agents: one in charge of the detection of opportunistic science, Science Assessment and Response Agent (SARA), and another to perform the instrument placement (mounted on a robotic arm). The three agents are coordinated by and executive system that is a purpose-built implementation.

The TVCR works over a time-line representation, but in a different manner than other approaches: while typical time-line planning represents the evolution of a state over time, here is only one time-line that is composed by fragments. A fragment is a partial ordered plan constructed using action-oriented approach with PDDL [57]. They specify the state goals and a set of actions based on their preconditions and effects to traverse, approach targets or to perform different science operations. The fragments to accomplish the different goals could be constructed onboard (for opportunistic science) or sent by the ground station. The fragments arrive to the TVCR and it inserts them into the time-line considering constraints such as energy consumption, data storage, time dependence and, also, an associated priority value to each fragment, which is an estimated relative science value in comparison with the other fragments. To link them, TVCR uses a similar to a single layer Hierarchical Task Network (HTN) [45] in which the fragments are included in the plan together with constraints on their relative positions. To avoid flaws in the plan (specially for replanning process when a new science target is detected), TVCR has different strategies: it is possible that low priority tasks are removed from the time-line, delaying actions that compete for unitary resources or adding support action to ensure the coherence of the time-line. When these resolutions procedures fail to generate a valid time-line, TVCR can use a fail-safe strategy or set the time-line to the last valid one. This ensures that always exists a coherent plan to execute.

The opportunistic science agent, SARA, is composed of a context based system that defines the attributes to classify geological features. Using 2D pictures and different image analysis, the agent provides a score for each potential science objective (computed giving values for each designed group of attributes and performing an weighted function to obtain a final score). The attributes selected and scores are fixed and learned from the lessons of geologist and results obtained from previous missions such as the Mars Exploration Rovers (MER). Finally, there is another agent that implements some specific vision and motion planning algorithms in charge of the robotic arm control.

Finally, one of the latest developments in the multi-agent architectures is GOAC [28] (see fig. 2.10) an effort from ESA to create a reference robotic software platform for different space missions. It has two differentiated layers: a functional layer and a decisional layer composed of agents (called reactors). These reactors are implemented using the Teleo-Reactive Executive (T-REX) [148] for planning and execution dispatching using a timeline-based representation and an interleaving schema (which is similar to IDEA). More in detail, GOAC is the integration of several components: (i) a timeline-based deliberative layer which integrates a planner, called OMPS [58], built on top of Advanced Planning & Scheduling Initiative (APSI) – Timelines Representation Framework (TRF) [29] to synthesize flexible temporal action plans and revise them according to execution needs; (ii) a T-REX to synchronize the different components under the same timeline representation; and (iii) a functional layer which combines G$^{\text{en}}$oM with a component based framework for implementing embedded real-time systems, Behaviour Interaction Priority (BIP) [13].

GOAC is not an autonomous controller: is a template to create specific controllers for different missions/platforms. In that sense, a GOAC instance is a functional configuration to successfully accomplish an objective. The aspect that determines the capabilities of the architecture is the number and hierarchy of the T-REX reactors. A reactor is an entity that operates over one or more timelines by (i) deliberating over their required status to achieve the mission goals and/or (ii) modifying the status of the timelines as a result of an operation or for an environment/platform change. A timeline represents the temporal behaviour of a particular element of the environment/platform that can be observed and/or modified to accomplish the mission goals while maintaining the operative constraints defined in the domain.

The decisional layer of GOAC can contain different kind of reactors. First are the deliberative reactors, that is, the couple of a T-REX reactor and an APSI planner. Each one follows a sense-plan-act paradigm for goal oriented autonomy over a particular part of the model and its own look-ahead window over which to deliberate. The world is modelled using the DDL language [30, 58], which enables to specify the allowed state transitions as well as the causal and temporal relationships between state variables. So, the model represents a set of relevant features whose temporal evolution need to be controlled to obtain a desired behaviour. Therefore, the result of a deliberative process is a sequence of state transitions for each timeline that achieves a specified condition as a planning goal.

The coordination between reactors is done sending goals and receiving observations. Due to each deliberative reactor reasons about a specific part of the whole domain, appears a *hierarchy* in the set of reactors, being one the high level reactor, which has the more abstract states, and whenever we go down in the hierarchy, the reactors are more specific. In this way, a high level goal state could be decomposed into lower level goals states that must be reached by their respective timelines before the completion of the goal state in the higher level. To do this, the reactors have two types of timelines: internal and external. The first ones are the timelines that the reactor can directly control: the reactor can change its value, but usually the value is dependent on one or more external timelines. These are not controllable from the reactor, but it can produce goal states in those timelines required to reach the goal state of its internal timelines.

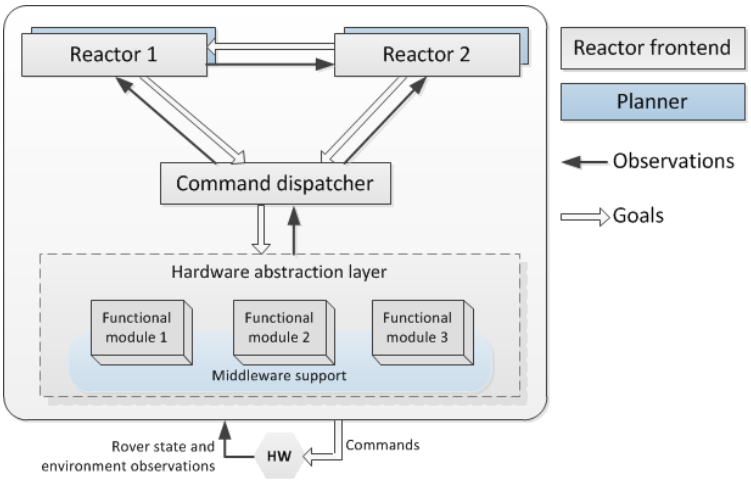


Figure 2.10: A particular instance of the GOAC architecture.

In order to ensure the correct execution of the required commands to achieve the goals, it is possible to have one or more reactors in charge of the command dispatching between the deliberative reactors and the functional layer. These reactors could implement procedural executives that not only give correctness in the command execution, but also provide a better level of abstraction and an interface between the functional layer and the higher level reactors. In fact, these reactors are responsible of gathering observations from the functional layer and to provide it in a convenient way to the deliberative reactors.

In GOAC planning and execution are interleaved: while the functional layer is executing a command, the executive is permanently observing the environment and the robot internal state, so, it is capable of detecting changes and responding in a short time by exploiting reactive planning schemes, instead of performing a replanning process, often more expensive. In any case, at the beginning, the deliberative component generates a plan to achieve the mission goals. The plan corresponds to an assignment of the required transitions for each timeline to accomplish the objectives. The possible transitions of a timeline are defined in the DDL files and each transition changes the timeline status, being the status defined by its value, attributes (if required) and temporal interval, i.e., each action duration is temporary ranged between a minimum and a maximum value. In GOAC the time has a major importance in the system and it is important to mention that the time dedicated for planning is fixed: if the deliberative is not able to obtain a plan in the given slot, the system will halt and wait for human supervision.

In multi-agent architectures the coordination of agents is an important and no trivial question, that requires an important effort to define the interactions. However, this provides the goodness of a distributed and parallel system. The problem decomposition depends on different factors, such as the modelling and agent capabilities. Notwithstanding, good designs rely on the possibility of deploying specialized agents with different reactivity and competence levels, which could be analysed and tested one by one.

2.2 Heuristic search 2D path planning algorithms

Some of the autonomous controllers presented above have been applied to mobile robotics. In this regard, it is common that the deliberative employs path planning algorithms to decide the paths to follow to achieve the mission goals. Then, path planning is a widely research problem in mobile robotics. It is focused on finding an obstacle free path between an initial position and a goal, trying as far as possible that the path is optimal. However, in robotics is common to differentiate two approaches in path planning: local path planning (usually identified as navigation) and global path planning. First type is more related to the functional layer, in which, using the sensors information (or external systems [50, 149, 151]), the robot moves to achieve a (near) target by means of analytic methods. Instead, the global path planning is associated with the P&S systems, exploiting algorithms to generate paths for long traverses. Notwithstanding, both techniques can be complementary to provide better mobility capabilities. In this regard, Guou et al. [67] argue that *"path planning methods such as A* and D* are efficient in a global scale at a coarse resolution to save computational expenses, and analytic methods are good at a regional scale for real time control"*. In this dissertation we are centred on P&S, so we will focus on global scale path planning. Also, we assume that the terrain is *fully-observable* (i.e., we have all information about the terrain) and *static* (i.e., the terrain does not change during the path extraction).

Usually, the path planning problem is represented as a search problem over a discrete environment with blocked or unblocked cells (i.e., a binary occupancy grid). In such case we can perform the path extraction via uninformed search, using algorithms such as Dijkstra [41], which only considers the cost of reaching a position for the search evaluation function. So, a wide area of the search space must be explored to obtain a path, which is an expensive process. As a consequence, this approach presents a bottleneck in the processor and memory usage, as the algorithm complexity grows exponentially with the problem complexity [154]. So, commonly, informed search algorithms are used. Taking advantage of a domain dependent heuristic, heuristic search algorithms provides good solutions with less computational effort.

The most used heuristic search algorithm to solve the path planning problem is A* [69]. It quickly finds routes over a grid at the expense of an artificial restriction of heading changes of $\pi/4$. A* is simple to implement, is very efficient and has lots of scope for optimization [110]. In this direction, there have been many improvements such as A* Post Smoothed (A*PS) [19] that smooths the path obtained by A* at a later stage; Field D* [48] a modification that works on partially known and non-uniform costs maps; or an approach that uses framed cells approach (a subdivision of the grid) for continuous-field path planning with path-dependent state variables [11].

More recently has appeared the Theta* algorithm [134], which removes the restriction on heading changes of A*. Theta* generates shorter paths with less heading changes than A* [36]. The main difference between A* and Theta* is that the former only allows connecting adjacent locations, while in the last, it is possible to connect no adjacent locations if the straight line between them does not cross blocked cells. However, this improvement implies a higher computational cost due to additional operations to be performed to check the line of sight between no adjacent nodes.

The previously introduced algorithms are deterministic. However, there are also non-deterministic search algorithms for path planning. For instance, the Rapidly-exploring Random Trees (RRT) [98] or the Probabilistic Road Maps (PRM) [3] algorithms are become largely popular, specially for the motion planning of robotics arms. In this dissertation we are focused on deterministic algorithms, so we will introduce A*, A*PS and Theta*, that are actually the most well known and used deterministic path planning algorithms for mobile robotics. Next subsections present the nomenclature employed during the rest of this dissertation and the selected algorithms in more detail.

2.2.1 Grid definition and notation

The notation widely used in path planning for the 2D terrain discretisation is a uniform regular grid with blocked and unblocked square cells [178]. Generally, we can find two variants of this kind of grids: (i) the *center-node* (fig. 2.11 left) in which the mobile element or node is in the center of the square; and (ii) the *corner-node* (fig. 2.11 right), where nodes are the vertex of the square. In the *center node* representation a cell is the surrounding area of the node. Instead, in the *corner-node* a cell is the area contained between four adjacent nodes. For both cases, a valid path is that, starting from the initial node, reaches the goal node without crossing a blocked cell. In this dissertation we follow the *corner-node* representation, but typically all algorithms are also able to work with the *center-node* one.

From now on, consider a node p as an arbitrary node, and s and g as the initial and goal nodes respectively. Each node p is defined by its coordinate pair (x_p, y_p). A solution path has the form $(s, p_1, p_2, ..., g)$. To support heuristic search algorithms, we require the following information for each node p:

- parent(p): the predecessor node of p. It is mandatory that the straight line between p and its parent does not cross blocked cells.
- successors(p): a list of nodes that are reachable from p. The adjacent of each node t in that list is p. Therefore, if parent(t) $= p \Rightarrow t \in$ successors(p).
- G(p): the cumulative length to reach the node t from the initial node, that is, the length of the shortest path from the start node to the p node.

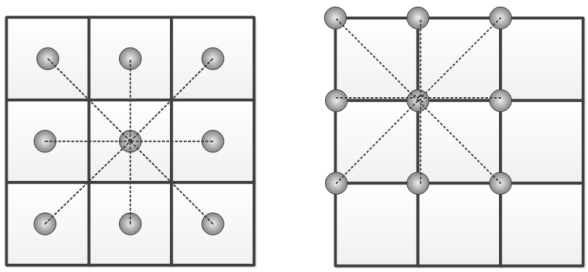


Figure 2.11: Possible node representations in grids: center-node (left); corner-node (right).

- H(p): the heuristic value of p, i.e., an estimation of the distance from p to g.
- F(p): the node evaluation function as in eq. 2.1.

$$\mathrm{F}(p) = \mathrm{G}(p) + \mathrm{H}(p) \tag{2.1}$$

While uniform grids only contain free or blocked cells, non-uniform maps (typically called traversability cost maps) define a numerical value for each cell. This value (called cell cost) defines the *effort* of traversing such cell. The cost can be computed accordingly to the application to express a terrain property (or a lineal combination of different properties). Using traversability cost maps, the cost associated to a path (i.e., G(p)) is obtained by multiplying the cost of the cell by the distance travelled in that cell. Using this representation, path planning algorithms can try to avoid certain areas that can be considered dangerous or unsafe.

To enable comparison of path planning algorithms, it is common to contrast four parameters: length of the solution (path length), search time (runtime), number of expanded nodes and number of heading changes. In the literature, it is common to assess the goodness of an algorithm focusing on the first two parameters.

2.2.2 A* algorithm for path planning

Some path planning algorithms are a variation of the A* search algorithm [69]. It is simple to implement, is very efficient and optimal when it is applied to visibility graphs. But it has an important limitation: is typical to use 8 adjacent nodes, so this restricts the path headings to multiples of $\pi/4$, causing that A* generates a sub-optimal paths with artificial zig-zag patterns. It is possible to use more adjacent nodes to relax this constraint, but the complexity grows exponentially with each increment.

Using an eight-connected grid, A* obtains the best performance employing the *Octile* heuristic, which represents the minimal cost to reach a node with eight movements. It is calculated as in eq. 2.2.

$$\begin{aligned} Octile(p,t) &= \sqrt{2} \cdot dsm + (dlg - dsm) \\ \text{with } dsm &= min(|x_t - x_p|, |y_t - y_p|) \\ \text{and } dlg &= max(|x_t - x_p|, |y_t - y_p|) \end{aligned} \tag{2.2}$$

A* has two lists to manage the search: the *open* list, a priority queue of nodes ordered by their F values; and the *closed* list, which contains the nodes that have been already expanded. The search process, shown in alg. 1, will expand the most promising nodes in the order established in the *open* list, i.e. it first expands the nodes with lower values for F. In the case that the expanded node is the goal, the algorithm will return the path by traversing the parents pointers backwards from the goal to the start node. Instead, if the *open* list is empty, it means that it is impossible to reach the goal node from the initial node and the algorithm will return a failure.

When we are dealing with the successor list, successors(p) being p the current node, we need to update the data of non-expanded nodes. This is done by the *UpdateNode* function at line 18 in alg. 1. When the search reaches a node for the first time, or the cost to reach that node from the current position is less than the previously obtained, the G and H values are updated properly and the parent of that node is set to the current node. Then we update the *open* list and continues with the search process. The pseudo-code of this function is presented in the alg. 2, taking only into consideration the lines 12–19.

Algorithm 1 A* search algorithm

```
1  G(s) ← 0
2  parent(s) ← s
3  open ← ∅
4  open.insert(s, G(s), H(s))
5  closed ← ∅
6  while open ≠ ∅ do
7      p ← open.pop()
8      if p = g then
9          return path
10     end if
11     closed.insert(p)
12     for t ∈ successors(p) do
13         if t ∉ closed then
14             if t ∉ open then
15                 G(t) ← ∞
16                 parent(t) ← null
17             end if
18             UpdateNode(p, t)
19         end if
20     end for
21 end while
22 return fail
```

2.2.3 A* Post-processing: improving A* paths

There are some variations of A* to convert it into an *any-angle* algorithm [89, 166]. Any-angle algorithms are those in which the heading of the robot is not limited to certain values (as happens with A*). In this dissertation we use A* Post Smoothed (abbreviated A*PS) algorithm [19]. It runs A* and then smooths the resulting path in a post-processing step. Therefore, the resultant path may be shorter than the original, but it increases the runtime. If A* finds a path $(p_1, p_2, ..., p_n)$, the smooth process checks the line of sight between the first node and the successor node of its successors. The meaning of line of sight between two nodes is that, if there are no obstacles in the straight line which connects these two nodes, then, there is line of sight, otherwise, there is not. For instance, taking the initial node $s = p_1$ as the current node, A*PS checks if there is line of sight between p_1 and p_3. If it is true,

the parent of p_3 is now p_1 and thus p_2 is eliminated from the path. The algorithm then takes the next node in the path and checks the line of sight with the current node. If there is not visibility between these two nodes, the last node becomes the current node and the line of sight check continues. The process is repeated until it reaches the goal. At the end of the procedure, the resultant path has the same or less nodes than the original one, that is, $n \geq j$, being n the number of nodes in the original path and j the nodes in the post smoothed path. Removing intermediate paths effectively reduces the path length and the heading changes.

2.2.4 Theta* algorithm: any-angle path planning

Theta* [36, 134] is a variation of A* for any-angle path planning on grids. There are two variants for Theta*: Angle-Propagation Theta* [134] and Basic Theta* [36]. We assume that talking about Theta* refers to the last one, as it provides better results. Theta* is identical to A* except the *UpdateNode* function, so the pseudo-code for A* shown in alg. 1 also applies to Theta*.

Theta* works like A*PS: both try to connect no adjacent nodes that have line of sight, erasing intermediate nodes and thus, possible zig-zag patterns generated by A*. But there is an important difference between Theta* and A*PS: first one does not need a post-processing step, it does the line of sight check during the nodes expansion. When Theta* expands a node, p, it checks the line of sight between the parent of the node and its successors. If there is line of sight between a successor of p and its parent, then the parent of the successor is parent(p), not p like in A*. When there is an obstacle blocking the line of sight, then Theta* works like A*. For this reason, the parent of a node can be any node, and the path obtained is no restricted to $\pi/4$ headings. The *UpdateNode* function pseudo-code for Theta* is shown in alg. 2. The algorithm used to check the line of sight is a variant of a drawing lines algorithm [20] that only uses integer operations, which is implemented in the *LineOfSight* function (see alg. 2 line 2). In order to obtain better results, the heuristic employed by Theta* is the *Euclidean* distance (computed as in eq. 2.3) that represents the minimum cost to reach a node in a free obstacle area and without restrictions on the heading angles. As a consequence of the expansion process, Theta* only has heading changes at the corners of the blocked cells.

$$Euclidean(p,t) = \sqrt{(x_t - x_p)^2 + (y_t - y_p)^2} \tag{2.3}$$

Daniel et al. [36] perform an extensive comparison between Theta*, A*, Field D* and A*PS. As a result we can see that Theta* is the one that usually gets the shortest routes. The disadvantage of Theta* is that the line of sight check is frequently performed, which degrades significantly its performance.

Moreover, some effort has been performed in order to improve the Theta* performance, such as Incremental Phi* [135] that takes into consideration the free-space assumption and angle ranges computation to provide a speed-up of approximately one order of magnitude with respect to Theta*. As well, Choi et al. [31] include pruning rules in the line of sight computation that can reduce the runtime of Theta* by up to a factor close to 2 without a significant increase in the path length.

Algorithm 2 Update node function for Basic Theta*

```
1  UpdateNode(p, t)
2  if LineOfSight(parent(p), t) then
3      if G(parent(p)) + Euclidean(parent(p), t) < G(t) then
4          G(t) ← G(parent(p)) + Euclidean(parent(p), t)
5          parent(t) ← parent(p)
6          if t ∈ open then
7              open.remove(t)
8          end if
9          open.insert(t, G(t), H(t))
10     end if
11 else
12     if G(p) + Euclidean(p, t) < G(t) then
13         G(t) ← G(p) + Euclidean(p, t)
14         parent(t) ← p
15         if t ∈ open then
16             open.remove(t)
17         end if
18         open.insert(t, G(t), H(t))
19     end if
20 end if
```

2.3 Path planning considering terrain properties

Path planning algorithms previously presented try to minimize the total distance that the robot should travel. Although this criteria has been widely used to compare algorithms in the path planning community, it is not enough in the case of exploration robots: the shortest path can cross rocky or high hills areas that the robot will not be (safely) able to reach.

To include terrain characteristics some algorithms exploit traversability cost maps that can include different terrain characteristics (e.g., quicksands, rocks, etc.) expressed as a numerical value. This value indicates the estimated effort required to cross an area of the map. Algorithms that exploit cost maps may produce longer paths, but safer, as they include some terrain information during the search. However, merging different terrain characteristics into a single value simplifies the algorithm, but at the expense of losing specific information that can be useful during the path search. In particular, analysing slopes, rocks or other terrain characteristic separately can be desirable. Enabling the path planning algorithm to decide the paths having in mind different terrain characteristics may rely on safer paths, rather than only minimizing the path cost as reported in literature.

In that direction, algorithms such as the D* family (the most representative one is Field D* [48]) deal with traversability cost maps. These algorithms use interpolation to produce better value functions for discrete samples over a continuous state space. The innovation in the Field D* algorithm is a method for computing the cheapest path of each node t to the goal, given the path costs of its successors nodes. This

value is traditionally computed as the minimum cost of traversing the edge between t and any of its successors nodes plus the path cost of the chosen successor to the goal. This computation only allows straight lines from t to its successors. However, Field D* allows a straight line trajectory from a node t to any point on the boundary of the successors cell. Since computing all the boundary points for a nodes is infeasible, Field D* provides and approximation for each boundary point by using linear interpolation. Then, the cost of an edge that resides on the boundary of two cells is defined as the minimum of the traversal costs of each of the two cells. Although in some cases the linear interpolation returns a bad approximation, in general results show the benefits of the algorithm. In particular, the NASA employs a variation of Field D* to help in the path planning operations for the Martian rovers.

Other approaches exploit a cost map to represent the terrain elevation. Jaillet et al. [86] propose the Transition-based RRT algorithm that performs path planning on uneven terrains. This algorithm combines RRT with stochastic optimization methods to explore the search space. The solutions generated can follow valleys in a self-adaptive way. Initially, only very slow slopes are allowed. Then, if the algorithm successfully climb small slopes, the slope limitation is increased. Otherwise, when it is not possible to advance for the current slope limit, the limitation is relaxed.

Guti et al. [59] proposed an integrated 3D path planning architecture composed of two stages: acquisition and map generation. In the acquisition phase the rover obtains the elevation, slope, orientation and roughness of the terrain using a laser scanner. Then, a fuzzy engine produces a 2D traversability cost map associating to each cell a cost represented by a continuous value from 0 to 1. Particularly, they use a lineal combination of different aspects (estimated distance to travel, navigation cost, heading changes and safety areas) to compute the cell costs. Then, a path planning module generates the path using a modified version of A*. However, this solution heritages the unrealistic path generation since it exploits A*.

Another way to consider the terrain information during the search is based on the energy consumption crossing a 2.5D elevation map representation. By 2.5D the literature refers to a 2D grid with connected nodes to navigate areas which changes in their height axes. For the terrain discretisation, a first approach is to use a triangular mesh generated from a real Digital Terrain Model (DTM) [165], while others employ multiple Gaussian functions with randomly selected mean, variance, and height for terrain modelisation [32].

Also, Singh et al. [163] exploit 2.5D elevation maps representations to compute stable trajectories on uneven terrains. Capturing the full dynamics of a 6 degree of freedom wheeled robot, and a function that represents the surface, their approach allows evaluating candidate trajectories. To do this, the velocity, acceleration and wheel-ground contact constraints are used to analyse rover stability during the path. Initially, an arbitrary path is generated (by means of a parametric function [67]) and evaluated. While the stability constraints are not met, a path replanning is performed until a stable path is generated.

A different direction is to use a DTM, so the path planning algorithm will consider the terrain relief. This enables to assess the path slope and to consider the mechanical model of the robot. Following this direction, Ishigami et al. [85] present a technique to evaluate the motion profiles of a rover when it has to follow the shortest path

generated using the Dijkstra algorithm. Based on the DTM, they are able to evaluate the path considering the rover dynamics model and the wheel-soil contact model. Then, if the result of the path evaluation is *not safe*, they can generate a new path modifying weighting factors. This approach can solve the problem, but it is time consuming even for small areas of 50x50 meters. Moreover, it is limited to a particular robot geometry.

Other approach based on the DTM usage is proposed by Page et al. [143]. They are inspired by objects recognition algorithms that are based on a 3D triangle mesh. The input to the path planning algorithm is a DTM model transformed to form a triangle mesh and smoothed using a standard Gaussian algorithm. The generated 3D mesh can be non-uniform and non-regular. The output is a path following either the valleys or ridges of the terrain depending on the user criteria. Also, they introduce the *Geodesian* distance to operate over the discrete terrain instead of the *Euclidean* distance. Although the idea is quite novel, a rigorous analysis and algorithms comparison in a benchmark problem is missing.

2.4 Task planning and path planning integration

The path planning algorithms introduced above typically provide a path between two points. However, mobile robotics usually require to reach different targets in separate locations. Thus, the order in which the objectives are met has a great impact in the solution optimality. In this regard, it is required to perform some integration between a task planner and the path planning algorithm to generate better solutions in the deliberative layer.

Traditionally, task planning and path planning problems were covered by separating both problems, i.e., high level task modelling ignores low level constraints. This simplification results in inefficient or even infeasible solutions. Then, works that integrate task planning and motion/path planning to solve problems such as object manipulation or mobile robotics domains have been reported in the literature, providing better solutions. We can mention work done by Zacharias et al. [179], in which a two-arm humanoid robot must manipulate objects on a table. The particular problem addressed is how to automatically rearrange and manipulate objects without collisions. This is done by using a motion planner to compute trajectories, and a task planner to provide the grasping operations that achieve a obstacle free configuration.

In this direction, Cambon et al. implemented the aSyMov planner [26], which integrates Metric-FF [71] for task planning and PRM for motion planning. For the integration, both planners share a geometric representation of the position of the robots, obstacles, and objects in the environment. The plan extraction is guided by the task planner, which initially provides a solution for a problem in which all motion paths are valid (i.e., no obstacles are considered). Then, a motion planner checks if it is possible to achieve the goals. Selection of actions is guided by a cost function that adds (i) the number of times that an action fails due to infeasible paths, and (ii) the heuristic computed by the task planner. When there are no valid paths to achieve a goal, the motion planner updates the geometrical information of the task

planner, which must generate a new plan. Finally, when a valid plan is generated, an improvement process is executed with the aim of optimizing the planned paths while attempting to perform concurrent actions in multi-robots domains.

The I-TMP algorithm [70] uses a motion planning algorithm (a PRM) to determine trajectory constraints for a logical description of the tasks to perform. Such tasks are encoded in a formal language similar to PDDL and solved with and integrated planner that interleaves task and motion planning. This planner shows a good performance for planning the navigation of legged robots.

The SAHTN algorithm [175] is designed to deal with pick-and-place domains for robots equipped with arms. The algorithm implements a hierarchy from HTN for task planning to RRT for low level actions (i.e., arm movement). In this way, the hierarchy specifies a set of high level actions that can be refined until low level movements. For each movement, a reachable state space is produced that can be reused for different objects. By doing this, the associated search cost is reduced and, therefore, avoids infeasible solutions for the higher level.

Some of these approaches combine task planning and motion planning in the search process in a very specific way, without considering a general approach for interfacing both planners. In that direction, Erdem et al. [44] present a framework that provides bilateral interaction between the task planner and the motion planner. This relationship allows both planners to guide the search: the task planner aims to obtain an optimal task sequence and the motion planner performs geometrical reasoning. Given a plan generated by the task planner, the motion planner must ensure that the plan is kinematically solvable. If it is not, the motion planner will include new constraints in the task planner. At the same time, during the plan search, the motion planner includes specific domain information to guide the search. This framework also allows exploiting different motion planning algorithms such as PRM or RRT.

In a similar direction, Srivastava et al. [164] propose an approach in which they integrate a PDDL planner (they use FF [72] in their experiments) with a motion planner without modifying them. To perform such integration, they design an interface layer that allows them to share information. The objective of the interface layer is to invoke the motion planner to generate paths for the tasks produced by the PDDL planner. If there are no valid dynamics for a given task, the geometrics constraints discovered are abstracted and included in the PDDL model to fix the plan. The way that they perform such abstraction is one of the main contributions of the work.

In general, the above mentioned approaches focus on integrating task planning and motion planning, which are more related to the manipulation of objects employing robotics arms. Meanwhile, this dissertation is focused on mobile robotics. Notwithstanding, both problems are very similar as the objective is the integration of domain specific information in the task planner, interleaving task planning with path planning.

2.5 Evaluating autonomous controllers

Considering the autonomous control architectures presented in sec. 2.1, we can state that an autonomous controller is the result of a multidisciplinary team that shall take into consideration a large number of aspects to safety deploy a robot. Starting from the objectives (or goals) to achieve and the environment in which the robot will operate, the design and implementation shall cover: (i) the specification of the robot or platform and its instrumentation; (ii) the desired autonomy level and; (iii) as function of the autonomy level, the controller shall implement certain capabilities such as teleoperation, reactivity and, in the higher autonomous level, on-board planning.

However, Fontana et. al. [56] identify that in autonomous robotics there is a lack of a scientific testing methodology: the presence of uncertainty, errors and the decision capabilities of the robot makes the results presented in robotics papers a *"proof of concept"* due to the impossibility of reproducing the results, the heterogeneous experimental conditions and the use of subjective and/or insufficiently general performance metrics. Also, Flückiger and Utz represent the common feeling about performance and comparison of autonomous controllers in a phrase: *"its [the autonomous controller] performance is measured based on how well it supports robot software development, not how well the robot control system performs."* [54]. In fact, in the works summarized in the previous section, it is usual to see demonstration of effectiveness in particular circumstances, but not extensive evaluations providing details such as the performance of P&S systems. In this regard, the importance of properly evaluating autonomous controllers with enough details and extensive benchmarks can be summarized as:

- **Validation and verification.** For real applications it is important to validate and verify the correctness of the whole system before deploying it. For AI systems this implies a difficult point due to the complexity of the algorithms and its ability to take decisions based on different factors, for instance, when dealing with environment models with a high number of states or with uncertainty and stochastic models.

- **Improvement and robustness.** In autonomous systems time plays a fundamental role. The time spent to take a decision could lead into inconsistencies if the environment changes faster than the system deliberates or a passive attitude in certain circumstances can lead the system to a critical failure. Thus, analysing the behaviour of the system allows us to improve it and to avoid unsafe situations.

- **Learning.** As appear in Rockel et. al. [153], *"one way to improve the robustness and flexibility of robot performance is to let the robot learn from its experiences"*. In order to learn (or to improve) behaviours, the robot must be able to measure the performance of the different actions. This can be done in two ways: measuring the goodness of the solution only as function of the result or, using metrics that not only takes into consideration the final result, but also the performance of different elements involved. This implies that, if we

have good metrics, we can take into consideration those actions that improve not only the final result, but the performance of the involved components.

- **Comparison.** Some controllers can be exploited to solve the same scenario. However, to determine which one is the best option depends on several factors. If we have a framework that enables the comparison of different approaches, we can objectively choose the autonomous controller that fits better our application.

In robotics research it is hard to compare and to analyse the result of different approaches, in part, due to lack of technical details in the publications and also, because reported results are tested by solving a limited set of specific problems, in particular circumstances that usually are not reported, which include software, hardware and problem representation. As described by Pobil [38], a benchmark is required to avoid that situation, leading to an improvement of the quality of the research results with rapid adoption of the new applications. He pointed out that *"a benchmark is successful if and only if it is widely accepted by the community at which it is targeted."* One question that should be defined is the applicability of the benchmark: it shall be useful, which implies, for a broad field as robotics, that it must be focused on particular sub-domains. For this, Pobil proposes to follow the next activities in the definition of a benchmark: first, make accessible to the community the state of the art in the topic to address; then get involved specialists in the process of benchmarking definition with workshops, discussions, etc.; finally, describe the benchmark in detail, as well as their associated metrics, and an independent measurement procedure.

So, it is clear that creating and providing a general benchmark for autonomous controllers shall bring a big profit for the community involved in this area. However, performing autonomous controllers assessment is not a trivial issue. It is required to take into consideration all elements involved, characterize them, and test the whole autonomous controller under different circumstances. How different layers are interconnected is also relevant, so we need to test the system as a unit, meanwhile we also analyse it from the perspective of the different components.

In the following subsections we will try to address this issue summarizing previous works that try to analyse and characterize specific parts of an autonomous controller or the system as a whole. First, we introduce some efforts that try, taking the environment or the couple platform/functional support as a constant, to analyse the performance of autonomous controllers. Then, we summarize the efforts done to characterize and analyse autonomous systems from a theoretical perspective. Also, we present approaches that try to cover relevant topics of an autonomous controller from the perspective of Knowledge Engineering (KE) and ontologies.

2.5.1 Comparing architectures

Only one work tries to compare different control architectures using the same robotic platform. Orebäck and Christensen [142] implement three autonomous controllers (Saphira [91], TeamBots [12] and BERRA [102]) over a Nomadic Super Scout robot with the objective of accomplishing office navigation tasks. Although this work seems

to be very interesting as a starting point for comparing autonomous controllers, the authors focused on non-objective and weak evaluation criteria to compare the architectures. They evaluate some aspects such as the complexity of installing the system, operating systems supported, inclusion of graphical interfaces, types of HMIs and documentation, among others. The only objective values that they included were sensor actuator latency and memory required, which is very limited information to provide a relevant performance assessment. The main contribution of this work thus, is not the comparison of different architectures. As a result of their effort, they provide a set of guidelines to implement control architectures, with the objective of allowing easy interfacing to a variety of platforms and porting these systems across laboratories, to avoid the difficulties and lacks that they found in their experiments.

Another point of view is the robot competitions, e.g., euRathlon [156], RoCKIn [101], etc. There, the environment and the objectives are the same, but the platforms sometimes are different. Pobil [38] argues that robots competitions provide a good and fast way to improve the research results in autonomous robotics. Behnke [14] goes further and argues that robots competitions are fine to provide benchmarking in the manner that the competitions provide a good channel to evolve and exchange ideas between different teams by solving a common task using different approaches. Such robot competitions provide a standardized way of defining the environment, constraints and application of the participant robots, thus, it is possible to directly compare the different hardware and software solutions.

However, a difference in the performance between two robots could be due to different performance of a subsystem, such as actuators or decision makers, while, also, when a robot does not perform well, maybe this is due to a software/hardware failure or a bad solution for the problem. Thus, it is desirable to include specific subsystem tests in the competitions. Finally the lack of detailed technical documentation of the winning team is an important issue. After the competition a detailed report shall be required in order to advance the entire field.

2.5.2 Defining models

Hudson and Reeker [77] claim that if the autonomy is standardized, researchers and companies interested in robotics and AI could test their innovations with the same criteria than previous works and other teams, allowing a good measuring of the improvement produced. They identify five areas required to measure the autonomy of a system: replication, human dependency, adaptation, communication and learning. The replication gives a score taking into consideration if the robot is able or not to replicate itself (both, software and hardware), the more capacity it has to replicate, the higher the score. For human dependence it is quite similar, higher scores are given when the system has less dependence. The adaptation capability is highly related to the dependence; it measures the ability of dealing with dynamic and changing environments without human intervention. The communication can be expressed as a percentage of understanding between the autonomous system and humans and/or others systems. Finally, the learning component relies on a (not fixed) scale that gives more score to those systems that not only can learn by itself, thus also can transfer knowledge to other ones.

Once all of these five aspects of autonomy have been measured, they can be added together to obtain a global value. This global value can be used in the Sheridan's model [144] to obtain a specific category of autonomy for the system. The Sehridan's model is presented in table 2.2.

Table 2.2: Sheridan's model (extracted from [144]).

		Levels of automation of decision and actions selection
HIGH	**10**	The computer decides everything, acts autonomously, ignoring the human
	9	Informs the human only if it, the computer, decides to
	8	Informs the human only if asked
	7	Executes automatically, then necessarily informs the human
	6	Allows the human a restricted time to veto before automatic execution
	5	Executes that suggestion if the human approves
	4	Suggest one alternative
	3	Narrows the selection down to a few
	2	The computer offers a complete set of decision/action alternatives
LOW	**1**	The computer offers no assistance: human must take all decision and actions

Another attempt to characterize the autonomy of a robotic system is the Autonomy Levels For Unmanned Systems (ALFUS) [76]. It aims to compare the capabilities of unmanned[1] systems from a common perspective, covering a wide range of autonomous systems, such as military, service and exploration robots. Their interest come from two sides: to allow a common communication to express autonomy requirements and to provide a tool that could be used for testing/verification of autonomy systems. To do this, four components are depicted:

- A set of standard terms and definitions, published by the unmanned working group [1].
- A detailed model for Autonomy Levels. To define the levels of autonomy, the ALFUS framework is divided into three categories: (i) mission complexity, (ii) environment difficulty, and, (iii) human independence. These categories are composed of a set of detailed metrics which are (in part) specific for each system and application. Figure 2.12 shows a possible representation of two different autonomous controllers after obtaining the scores using this three-axis model.
- A Summary Model for Autonomy Levels, which is generated from the detailed model. From the previous metrics, a summarized model is a final score that is

[1]In some areas an autonomous system is commonly designed as unmanned system. However, we prefer autonomous system to avoid confusions with terms such as Unmanned Aerial Vehicle (UAV) that can be referred to an autonomous system or to a drone, which is typically teleoperated.

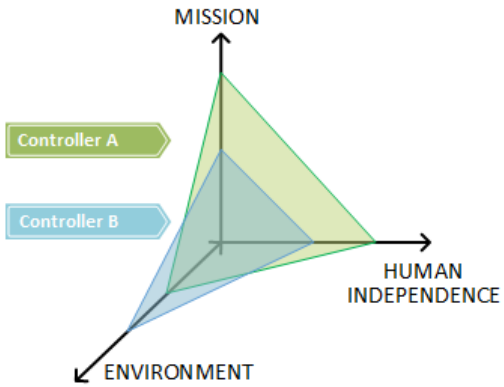


Figure 2.12: Evaluation of two controllers using the ALFUS summary model.

the addition of the three autonomy levels, each of them with values between 0 and 10, being the higher values the most complex mission, environment and human independent systems. So, the final score for the summary model is a value between 0 (no autonomy) and 30 (ideal autonomous robot).

- A set of guidelines, processes, and use cases to explain how to apply the ALFUS framework for autonomous systems evaluation. A set of different unmanned systems are considered following the ALFUS approach and their scores are computed [108]. However, these examples do not present the full set of metrics employed and they are only based on the information published of the different tested systems, but the authors claim that *"the assessments presented are subjective, although all bias has been consciously avoided"*.

For this framework a spreadsheet-based software was planned to be developed. In this tool, is proposed that the three autonomy levels scores are automatically computed based on weights and the metrics scores defined by the user. For every level a set of detailed metrics can be obtained by decomposing each task in a set of subtasks that can be, at the same time, decomposed into more subtasks. With this, a task tree is created, and each subtask have a weighted score. The final score for a task is the average score for its subtask tree. However, some technical difficulties arise in this point, for instance it is not clear how to decompose tasks and what are the correct weights to obtain a valuable metric. Also correlations and interdependency among metrics and measurably are complex issues: sometimes an average value does not give good information and a bounded interval is preferable or, the metric requires a no numeric scale, which is usually difficult to measure and generate with an automated tool. Thus, the ALFUS framework only provides a small set of guidelines as a start point to deal with these questions, and does not give a general framework that can be used to deal with different autonomous systems in a correct and objective way.

2.5.3 Defining methodologies

The Idaho National Laboratory proposes a methodology for testing autonomy of unmanned systems from lessons learned through different experiments [65]. A primary goal of their work is to enhance the understanding of how the robot, the control, interface, context and the human contribute to the mission success. In that point, an important focus is to choose the appropriate level of autonomy and how to measure the performance combining the task allocation between humans and robots. Thus, some metrics shall be focused on the operator (such as cognitive workload, error or frustration), while other rely on the autonomous system (distance travelled, sensor performance, robot initiative, etc.). Also, between both it is possible to distinguish metrics to analyse the communication interface (bandwidth, logging, etc.).

They proposed a methodology for testing autonomy composed of the following sequential steps:

1. Subject selection.
2. Comprehensive planning and study design.
3. Data logging from tests execution.
4. Subjective and objective measures to characterize human-robot performance.

In every step the users play a fundamental role. First, the user selection shall take into consideration the issue that a mission shall involve different kinds of users with different experience levels. Planning a mission requires a multidisciplinary effort and the appropriate level of realism and difficulty. Then, execute the missions and gather data. Although this seems easy, acquiring data is not a trivial task when searching for significant results. Attempting to figure out a priori which data is relevant could lead to miss significant data. Also, logging all data can produce a huge amount of information that is hard to interpret and manipulate. A good approach to retrieve data will help us to address unforeseen questions in order to obtain more objective metrics. Finally, from the data logged, metrics can be defined to determine the performance and autonomy level of the system. A conclusion that the authors present from their study is that *"neither objective nor subjective data alone are sufficient to provide an in-depth understanding of performance and performance issues."*

2.5.4 Defining metrics

Continuing the work started with ALFUS, Huang et. al. present the Performance Measures For Unmanned Systems (PerMFUS) [74,75]. Following the ALFUS schema, the performance is measured as attributes of the mission that the system shall perform (the complexity of the mission's objectives), the environment in which the robot works and the complexity of the system itself. With this, they want to facilitate the understanding of the intelligent system effectiveness to ensure that it meet the operational requirements and, also, devising technological improvements and inspiring innovation in the field.

In this way, a general framework is proposed to, starting from the requirements (captured via formal documents and from the knowledge of domain experts and observation), establish sets of metrics, to describe an approach and to provide a set of guidelines to facilitate the performance measurement of intelligent systems. The key to measure this performance is analysing the behaviours when the system interacts with the environment and humans, while aiming to achieve its mission goals. To do this, PerMFUS decomposes the performance into a set of areas according to the functionality of the system: mobility and navigation, energy, sensing and perception, communication, human-robot interaction, manipulation, coordination and collaboration and payload. For these areas, some general parameters characterization are described along a three-axis model (such as in ALFUS): system, environment and mission. First one explores how the physical or logical properties of the autonomous system affect its performance. Next one describes the parameters from the world that affect the system. Finally, the mission can provide details on how the whole system performs in term of mission goals. In PerMFUS, metrics are the parameters identified for measuring the performance, which can be selected from a general set or obtained through the requirements of the system. In order to obtain the metrics, an evaluation shall be performed starting from subsystem tests, and finishing with scenario based tests for mission levels, while detailing the setup, procedures, equipment, personnel involved and so on.

PerMFUS puts the focus on the concept of contextual metrics, emphasizing that the performance evaluation is based on context. It presents a set of six core generic metrics (completeness, accuracy, efficiency, reliability, safety and autonomy) that can be applied to different types of missions, by identifying the required measured parameters for each particular application and context. In this sense, these metrics are meaningful only when the application and context are properly defined.

By its side, the International Electrotechnical Commission published a technical report [84] in which they describe what and how to measure a set of performance metrics for service robots in a household environment. Although some measures can be easily adopted for other type of robots (such as position accuracy), some measures are highly coupled to the field of application or even are not interesting/practicable for others robotics systems (capability of returning to charging stand, noise test, etc.). Also, these measures are only valid to define performance of the whole system, they do not give any information of how different components perform. For instance, an error in the pose estimation can be due to a problem with a high-level path planner or to an incorrect implementation of the functional layer.

Another important point to discuss when dealing with autonomous systems is the interface. Currently, there is no perfect autonomous agent (which can superbly perform without human intervention), so, how it interacts with a human operator or supervisor (depending on the degree of autonomy) is a question to address. Crandall and Goodrich have made a study [35] to deal with performance metrics combining the autonomy degree of the robot and the interface between it and the human. To do this, they proposed a set of four metrics:

- **Neglect tolerance**: how much the robot can do autonomously. It represents a measure of the effectiveness of the robot autonomy. In general, when the human neglects the system, the performance decreases. For instance, teleoperation requires continuous human operation, while the point-to-point interaction scheme requires only interaction in particular times.

- **Interface efficiency**: how much the robot supports human interaction. It measures the effectiveness of the interface: when the human is attending the robot, it is expected that the performance of the robot increases. A good interface is one which gives enough information and well presented to allow the human to quickly interact with the system. A poor one, instead, can present a huge amount of information, unstructured or in a non-human legible way, causing that the human spends more time searching/interpreting the data rather than interacting with the robot.

- **World complexity**: how difficult is to operate in the environment. This affects both the robot performance and the interface. Usually, the performance of the robot decreases as function of the environment complexity and, also, it can be due to the interface. As the authors say, *"any metric which claims to estimate robot performance must take into account world complexity".* How to measure the world complexity is an open issue, however, as a guideline, the authors claim that a good measure of world complexity is that which gives higher values for environments which make a task difficult for a robot to perform.

- **Instantaneous performance**: how the robot performs in the current time. It is defined as the work performed by the robot with respect to its capacity to perform the work, computed as the ratio $\frac{work}{capacity}$. It is usually easy to compute the performance for a whole task, however, the instantaneous performance (when the task is in progress) can be very difficult to measure. In this work, it is assumed that the performance can be measured or estimated continuously.

Following with Crandall work [35], the performance of an autonomous system decreases as the human is dedicated to other tasks and/or the world complexity increases. Combining the neglect tolerance and interface efficiency it is possible to determine the human interactions frequency and their duration required to maintain a certain performance level of the system. For this, the Robot Attention Demand is defined as $\frac{d_{on}}{d_{on}+d_{off}}$ where d_{on} is the average time spent by the human interacting with the robot and d_{off} is the neglect time. Considering such function, we can identify the most useful schemes: those who offer low workload to the human and obtain high performance, when $d_{off} \gg d_{on}$, that is, the system exhibits more autonomy.

So, the performance of the system (including the interface) is defined as $V(\pi; t, c, t_N)$ where π represents an interaction scheme; t the time index, c the world complexity and t_N the neglect time. It indicates the average frequency and duration of interactions that should take place between human and a robot for a minimum performance level. Considering $t = t_{on}$ (elapsed time since the robot started to interact with the human) we obtain $V_S(\pi; t_{on}, c, t_N)$ a measure of the interface efficiency. Instead, if $t = t_{off}$ (the robot is being neglected), the tuple $V_N(\pi; t_{off}, c)$ provides a measure of the neglect tolerance.

To test this approach an evaluation has been performed. It consists of a simulated system in which a robot shall move inside a map. For the robot three interaction schemes are depicted: teleoperation, P2P and scripted. For the first one the operator uses a joystick to control the robot; in the second the operator tells the robot what to do in the next intersection (which way shall follow, like turn right) and, for the last one, the operator can drop a set of waypoints in a map to lead the robot to its goal. The environment consists of a set of maps with different complexities. The complexity value is computed taking into consideration the obstacles and the density of intersections. Then, with a group of operators a set of 120 tests were performed. In order to neglect the control of the robot, each operator controls two robots, one by one, while also, as a secondary task, s/he shall answer arithmetic problems until it is time to control other robots. As part of the conclusions, authors claim that, although the described metrics are powerful, user studies are very time consuming and sometimes impractical, thus more efficient methods for measuring these metrics are required.

More related to the functional support, Lampe and Chatila present a way to measure the performance of mobile robot autonomy [94]. In that case, the study is focused only on the mobility of a robot using autonomous obstacle avoidance methods, and the environment complexity, taking into consideration that the robot could have partial environment knowledge. One of the intentions of the work is to express the performance as a function of the world complexity, which can allow to predict the performance for a particular mission. To do this, a set of missions shall be executed to obtain the functions that describe the performance. Then, it is possible to extract the operational domains, the acceptable bounds for relevant parameters (such as reactivity, robustness, time or energy required) for a particular mission. Comparing the user constraints to the operational domains gives a percentage of mission constraints that we could expect to be reached by the robot.

The metrics involved in a navigation mission can be the velocity, travelled distance, duration and mission success rate. Also, for some metrics, it is possible to define both, global and local metrics. First one is measured at the end of the mission, while local metrics shall be acquired during mission progress. As well, the world complexity shall be defined. In that case, the complexity is also measured for global and local areas. For the global complexity the map is represented as a grid and the complexity is a measure of the entropy of the map. Locally, it represents a measure of the zone around the robot, taking into consideration the area covered by an obstacle sensor. The local complexity is defined as the largest free obstacle angle detected by the sensor. Finally, there is also a measure that quantifies the amount of information that is shared between the robot map and the real one.

A simulation has been performed consisting of a simulated robot with local avoidance algorithm an a proximity sensor that can detect obstacles at a distance provided as a parameter. The goal of the robot is to move in a corridor in which obstacles are randomly placed. The global analysis of simulations with different sensor ranges shows how it is possible to interpolate a function to link the global world complexity with the total mission duration. Also the local world complexity is related to the instantaneous velocity.

2.6 Summary

In this chapter we presented a review of autonomous controllers for robotics. During the presentation of the different systems, it is clear the diversity of solutions and technologies deployed. In this regard, we are focused in the higher layers, which entail deliberation by means of P&S techniques. As well, most of the presented system are focused on mobile robots, being the most common application the surface exploration domain. In this direction the high level layer has to be able to plan the paths to achieve the mission goals. For this reason, we also presented a survey in heuristic search path planning algorithms for flat surfaces. However, robotics shall deal with uneven terrains, so a brief analysis of path planning algorithms that consider terrain properties are presented as well. Then, we introduced different approaches that interleaves task planning and motion planning for domains such as pick and place, that can be applicable to the exploration domain. In this sense, these researches seem to be relevant to be implemented in deliberative layers, in order to generate better plans for autonomous robotics applications.

However, how to assess these new P&S systems is an open issue. We presented some works that aim to contribute providing evaluation methodologies and measures for autonomous systems. Notwithstanding, none of these approaches enables a general comparison method that is enough to generate objective and reproducible experimental campaigns. Then, the experimental evaluations of autonomous controllers can be seen as a demonstration of effectiveness in particular circumstances, as the results are, currently, neither comparable nor reproducible.

Chapter 3

Heading changes in 2D path planning algorithms

A reliable autonomous controller must be able not only to move between two locations, but also to reason about the path to reach the different goals. In this point, it is required to differentiate between short-term navigation and long-term path planning. First one is coupled to the functional level of the robot; it uses sensors information to navigate between waypoints, sometimes using AI vision algorithms. For the second one, we refer to path planning as a long-term route between two or more locations, obtained using AI methods over a digital representation of the terrain. Then, to reason about the routes to follow, heuristic search path planning algorithms, such as the presented in sec. 2.2, are typically exploited.

In this chapter we improve path planning capabilities for autonomous robots by creating and evaluating new path planning algorithms based on heuristic search, applying constraints based on the heading changes made during the path search [128]. In mobile robotics, the cost of making a turn could be higher than moving forward. As well, performing stationary turns in soft terrains can be dangerous. For this reason, it is interesting to assess the heading changes during the path search. To do this, first, the heading changes are formally defined. Then, we propose to focus on the heading changes to reduce the computational search requirements, i.e., memory and time. Finally, we describe an algorithm suitable for mobile robots that improves the heading changes of the path.

3.1 Measurement and formulation of heading changes

To measure the amplitude of a heading change during the search, we define Alpha (p, t, g) (or simply α) as the deviation required to reach a node t starting from the node p and pointing to the node g. Figure 3.1 shows its graphical representation and eq. 3.1 gives the way to compute the value of $\alpha(p, t, g)$ in the range $[0°, 180°]$, being $0°$ when three nodes are in the same line and ordered $p \to t \to g$ or $p \to g \to t$; and $180°$ when they are in the same line and the p node is in the middle, that is, $t \to p \to g$ or vice versa. The $\text{dist}(p, t)$ function is equivalent to the *Euclidean* distance, defined in eq. 2.3.

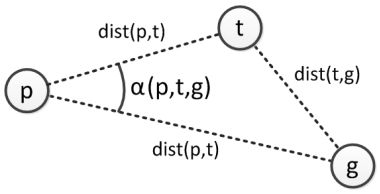


Figure 3.1: Graphical representation of $\alpha(p, t, g)$.

$$\text{Alpha}(p, t, g) = \arccos \frac{\text{dist(p, t)}^2 + \text{dist(p, g)}^2 - \text{dist(t, g)}^2}{2 \cdot \text{dist(p, t)} \cdot \text{dist(p, g)}} \tag{3.1}$$

In path planning, the length of the resulting path is usually employed as a the main measure for the solution optimality when comparing algorithms. Besides, there are other parameters such as the expanded nodes or the execution time. However, in the literature, the total amount of heading changes (i.e., the cumulation of degrees turned) usually is not reported. The number of heading changes cannot give us enough information about how *smooth* is the path: there could be low number of heading changes, but big in amplitude. In order to select a path planning algorithm for that case we can take into consideration how this parameter affects the quality of the path. To do that, we consider also a fifth parameter relevant to the study of path planning algorithms: the *total turn* parameter.

We define the total turn, Beta or β, parameter as the sum value of all heading changes (considering that the mobile element is oriented towards the first node of the resultant path) between the start and goal nodes. This is formally expressed in eq. 3.2.

$$\text{Beta} = \sum_{i=1}^{n-2} \beta_i \tag{3.2}$$

Each heading change, β_i, is the angle variation produced when we go from the node p_i to the node p_{i+2} through node p_{i+1}. In other words, β_i is the resultant angle of the intersection of a line that crosses the nodes p_i and p_{i+1}, and the line that crosses the nodes p_{i+1} and p_{i+2}. Also, the involved nodes must have the parent relationship: $p_i = \text{parent}(p_{i+1})$ and $p_{i+1} = \text{parent}(p_{i+2})$. We assume that the mobile element can rotate both to the left and to the right, so, in case of the resultant angle β_i is greater than 180°, it must be reduced to obtain an angle in the interval $[0°, 180°]$. β_i is computed as in eq. 3.3 and the angle calculation is obtained as in eq. 3.4. This last represents the angle formed by the abscissa axis and the point (x_p, y_p). A visual example of how to compute β_i and the total turn value is shown in fig. 3.2.

$$\begin{aligned} \beta_i &= |\text{angle}(p_{i+2}, p_{i+1}) - \text{angle}(p_{i+1}, p_i)| \\ \beta_i &= 360 - \beta_i \text{ when } \beta_i > 180 \end{aligned} \tag{3.3}$$

$$\text{angle}(t, p) = \arccos \frac{(x_t - x_p)^2 + \text{dist(p, t)}^2 - (y_t - y_p)^2}{2 \cdot (x_t - x_p) \cdot \text{dist(p, t)}} \tag{3.4}$$

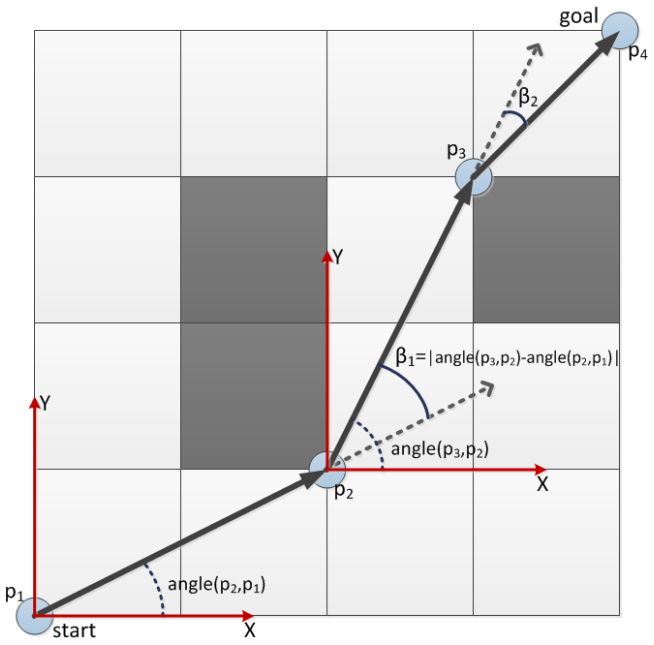


Figure 3.2: Example of β_i calculation for a small path. β_i expresses the heading between three nodes in the current path. The total turn value is $\beta_1 + \beta_2$, considering that the mobile are pointing to p_2 at the beginning.

Beta measures the total turn required to reach a path, so we do not get a measure of the amplitude of each turn. We can obtain the mean value for the turn amplitude simply dividing the total turns by the number of heading changes. But it is not really necessary due to the implicit proportionally relationship between these two values: less number of heading changes with higher values of total turns implies high amplitude turns.

3.2 Heading changes as a heuristic: efficiency improvement

In this section we take into consideration the measurement of heading changes during the search process to guide the search toward the objective. To do this, we consider that the shortest path between two points is the straight line if there are no obstacles. Thereby, points far from this line are not desirable and thus we do not want to expand them: they probably lead us further away from the goal. Using α as part of the heuristic value allow us to minimize the number of expanded nodes, and thus, the processing time and the memory required during the search, but having a negative impact in the path length and heading changes [123].

Considering the definition of α in eq. 3.1, we redefine the heuristic function of a node as the original heuristic employed by the path planning algorithm plus the value of $\alpha(s, p, g)$, obtaining a new evaluation function, F(p), as in eq. 3.5.

$$\mathrm{F}(p) = \mathrm{G}(p) + [\mathrm{H}(p) + \alpha(s, p, g)] \tag{3.5}$$

To compute the α value we require three nodes. One is the current node, p, and the other two are the initial and goal nodes, s and g respectively. Calculating α with these three nodes aims to expand only the nodes that are near (or are contained) in the straight line that connects the start and the goal nodes.

This line is the smallest distance between these two nodes if there are no obstacles blocking the path. For this reason, α takes values in the range $[0°, 180°]$, being $0°$ when the node belongs to the line and the search algorithm goes towards the goal node. It takes the middle value, $90°$, when the deviation of the node is perpendicular to the line. Values greater than $90°$ implies that reaching the successor node increments the distance to the objective, being the maximum value, $180°$, when the node is in opposite direction to the goal.

Figure 3.3 shows an example with the relevant data for two nodes. The α values for nodes p and p' are $11.31°$ and $18.43°$ respectively. If we suppose that the central cell (corresponding to the square formed by nodes $(2, 2)$, $(2, 3)$, $(3, 2)$ and $(3, 3)$) is blocked, A* expands first the nodes located at the top left of the map, expanding the node p', before the node p. But we can see that the predicted any-angle path length (dotted line) has higher β_i value for node p' than for node p. The difference in the heading change is 17% higher for node p' than node p in this example. So expanding the node p' is less likely with our heuristic. Using α with the algorithm forces the search to expand first the node p and relegates nodes far to the optimal unblocked path to the back of the *open* list.

We can deduce a quick conclusion about applying α directly to A*: the search algorithm evolves into a greedy search algorithm which tries to expand only the nodes that belong to the straight line between the start and goal nodes. In fig. 3.4

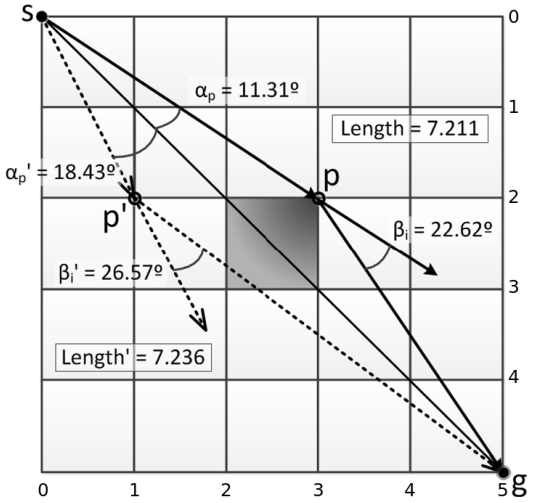


Figure 3.3: Example of α values for two nodes. When A* reaches the obstacle in the center, it expands p' before p. First one is not desirable due to the longer unblocked path length required and the bigger α value. Also, β_i is less for p than p'.

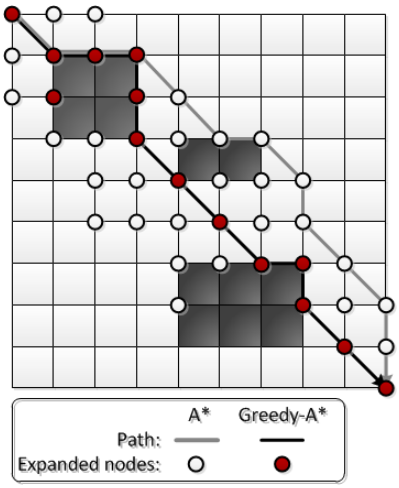


Figure 3.4: Path obtained using original A* (grey) and A* with the evaluation function presented in eq. 3.5 (black). Nodes expanded by A* are represented as a white circle whereas the red filled are expanded by both. With the modified heuristic, A* expands less nodes and therefore, the runtime is lower.

two paths, starting from the top left node and finishing at the bottom right node, obtained with A* are presented. The grey line correspond to the path found by the original A* algorithm (using the *Octile* heuristic), meanwhile the black line is the path generated with A* and the modified heuristic that includes the α value. Also, the expanded nodes for each search have been marked. Non-modified A* expands 41 nodes (empty and filled circles) and A* with α in its heuristic function expands 14 nodes (filled circles only).

A* with the modified heuristic expands near 66% less nodes than the original one in that example. We can also note the tendency of the algorithm to border the obstacles in order to recover the line with $\alpha = 0$. Therefore, although is predictable that it reduces the number of vertex expanded (and thus, the runtime), the use of α with A* does not imply benefits in terms of path length, and, if there is some obstacles blocking the line between the start and the goal nodes, the number of heading changes could increase. On the other hand, this pseudo-capability to detect obstacles could be useful for any-angle algorithms, so for these reasons, we will employ this heuristic over those kind of path planning algorithms.

We must take into consideration that α takes values in the interval $[0°, 180°]$. Considering a map with 100x100 nodes, the cost to transverse from one corner to its opposite corner is $100\sqrt{2} \approx 141$, and we can consider that α is well sized. However, for smaller or bigger maps this shall not be valid. For example, for 50x50 maps, the relative weight of α is double than for a 100x100 map and, for 500x500 nodes maps, is the fifth part. In the last case, α has less effect in the search process, so the algorithm tends to behave like the original one, that is, both expand a similar number of nodes; whereas for small maps the heuristic has an excessive cost to expand nodes that are a little bit far from the line between s and g. In order to compensate this fact, we modify the value of α as a function of the map size. This is show in eq. 3.6, taking into consideration a map with NxN nodes.

$$\alpha(s,p,g) = \alpha(s,p,g) \cdot \frac{\mathrm{N}}{100} \tag{3.6}$$

However, the relative weight of α can modify the behaviour of the search algorithm. Then, we have considered to multiply its initial value by a weight factor, α_w, as in eq. 3.7. As this factor increases, the relative weight of α over the search algorithm grows up. That is, the heuristic of moving away from the line that connects the start and goal nodes is bigger and force the algorithm to expand less nodes. This means that the α_w factor is inversely proportional to the number of expanded nodes during the search. It must be taken into consideration that, if $\alpha_w = 0$, the algorithm does not change its behaviour (due to the definition of the heuristic as in eq. 3.5). So, we can modify the behaviour of the search algorithm using α_w as a parameter. In the following section we discuss how the value of α_w affects the different search algorithms employed, that is, A*PS and Theta*. To summarize, we can say that a high value of α_w makes the search algorithm greedy, and values near to 0, slightly changes the original behaviour. As well, experimental results shows that values higher than 1 do not involve a better performance. Also, the degradation of path length is directly proportional to the value of the α_w factor.

$$\alpha(s,p,g) = \alpha_w \cdot \alpha(s,p,g) \quad \text{with } \alpha_w \in [0,1] \tag{3.7}$$

3.3 Heading changes heuristic experimental evaluation

In this section we provide a comparison between A*, A*PS and Theta*, and A*PS and Theta* using α as part of the heuristic function. For the experiments we use 0.25, 0.50, 0.75 and 1.00 for the α_w factor (when $\alpha_w = 0$ the heading change is not considered, i.e., the algorithm behaves like the original one). Figure 3.5 summarizes[1] the results for the execution of the different algorithms over 5000 random generated maps (see appendix A for details about the map generation algorithm) of 500x500 nodes with different number of blocked cells (5%, 10%, 20%, 30% and 40%, with 1000 maps for each group). For each map, we set the start and goal points in the following way: the start node is the south-west corner and the goal node is randomly chosen at the bottom from the column of the east. For all tests, the map generation algorithm and the initial conditions design guarantees that all problems have at least one solution.

The different path planning algorithms evaluated are implemented in Java and use the same methods and structures to manage the grid information. The execution is done on a 2.5 GHz Intel Core i7 with 8 GB of RAM under Ubuntu 14.04. We have performed the same tests for grids of 100x100 and 1000x1000 nodes (results not shown) getting similar results to the exposed here, so the presented results are representative independently of the map size. The algorithms that employ the α

[1] Although in path planning it is typically to present average values (graphically using barplots), we introduce the boxplot to provide a deeper representation of the generated data, showing the median, quartiles and variability outside the upper and lower quartiles. We think that exploit only average values for assessment looses relevant data. In this regard, initial researches that aim to improve the assessments performed in path planning are presented by Muñoz et al. [114–116].

value in the heuristic computation are denoted with H-Alpha followed by the α_w value. For the experiments, we have taken into consideration the values for the classical parameters analysed in path planning, replacing the number of heading changes with the β parameter introduced in sec. 3.1: (i) the length of the path, (ii) the total turns (β), (iii) the processor time or search runtime and, (iv) the number of expanded nodes during the search.

We compare the results obtained for A*PS and Theta* with and without α. For reference, we also include the results for A*. First, we take into consideration the path length. So let us start with A*PS. In terms of path length, the use of α

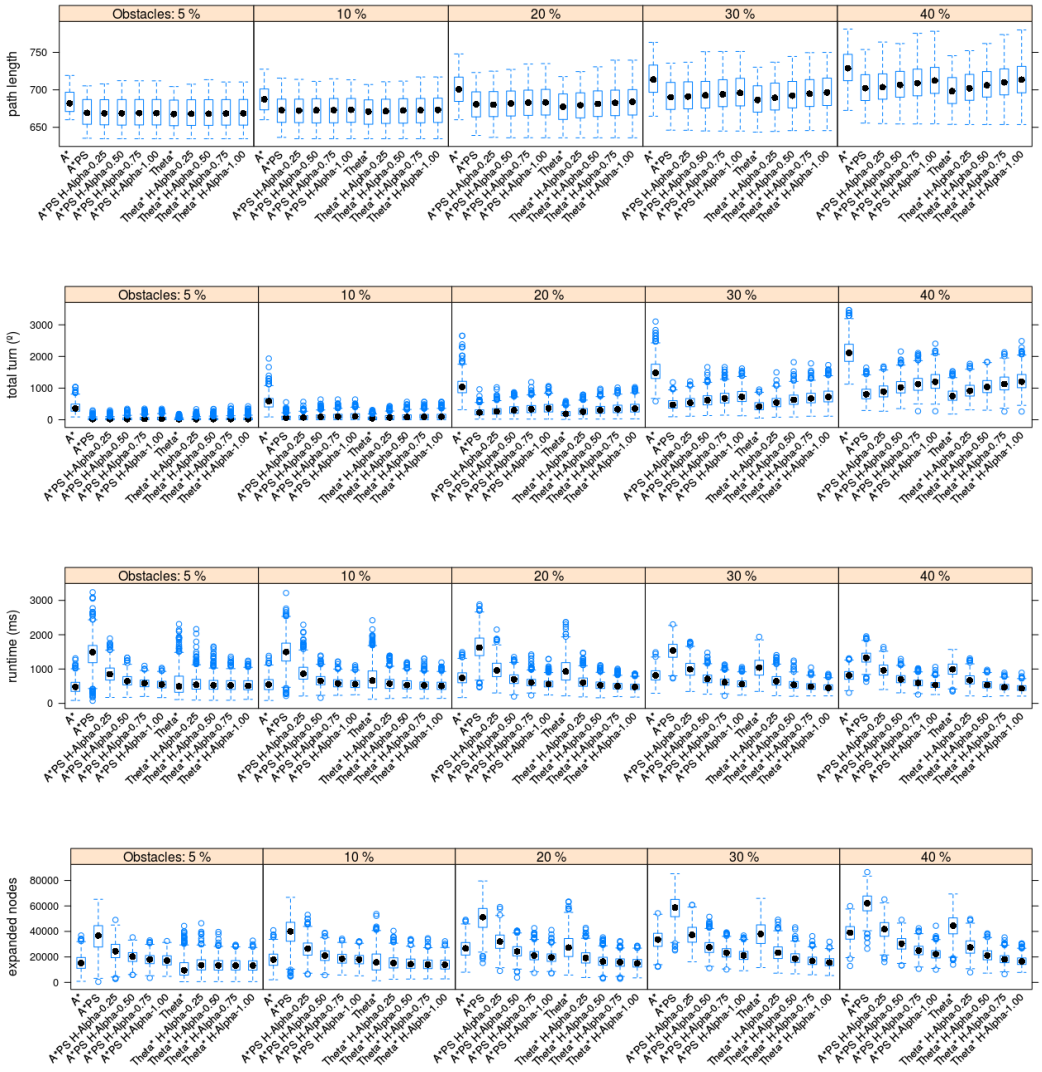


Figure 3.5: Results for the execution of different path planning algorithms over 5000 random generated maps (each obstacle group has 1000 maps). From top to bottom: path length, total turn in degrees, runtime in milliseconds and number of expanded nodes.

has little effect with less than 20% of blocked cells and with low α weights (that is, $\alpha_w = 0.25$). The best cases correspond to few blocked cells and low α_w values, getting similar paths than the original algorithm. The worst case, 40% blocked cells and $\alpha_w = 1.00$, gets, in average, 1.16% longer paths. Similarly, the total turns of the path increases proportionally to the α_w value, being more notorious the increases of the path length. However, the runtime using α is always better, decreasing between 1/3 (less blocked cells) and 1/2 (more blocked cells) when $\alpha_w = 1.00$. This can be explained due to the number of expanded nodes: A*PS H-Alpha expands less nodes as the α_w factor increases. Also, we can observe that A*PS H-Alpha with $\alpha_w \geq 0.50$ decreases both the path length, nodes expanded and the runtime compared to A*.

For Theta* using α the results are similar to the obtained with the modified A*PS, but the degradation of the path length is more notable in the cases with less obstacles. For instance, the path length worst case (40% blocked cells and $\alpha_w = 1.00$), gets, in average, 1.83% longer paths than the original Theta*. As well, the total turns of the path increases as α_w increases. The runtime is always lower using Alpha, but with $\alpha_w = 0.25$ it needs near 65% of the original time to get a solution, and with $\alpha_w = 1.00$ the speed-up is near to 50%, so Theta* using α works better in terms of runtime. Same as above, Theta* H-Alpha and $\alpha_w = 0.25$ achieves better runtimes than A*. For the expanded nodes the behaviour is the same as in A*PS; with $\alpha_w = 0.50$ it expands near the half of nodes than Theta* and with $\alpha_w = 1.00$ it decreases to the third part.

We are pursuing the objective of reducing the total turns for a path. In this sense, we can see that the algorithms with H-Alpha provides similar results in such parameter with low number of obstacles, but tends to perform more turns when the percentage of blocked cells grows more than 20%. Notwithstanding, even with the higher number of obstacles there are better solutions using $\alpha_w = 1.00$. In table 3.1 we provide the number of solutions with lower β values for the original path planning algorithms (A*PS or Theta*), for the algorithm using Alpha (with different α_w values), and the number of solutions in which β has the same value for both algorithms (that is, with and without α). For A*PS with Alpha we obtain good results for maps with less than 30% blocked cells. For instance, with 10% of blocked cells and $\alpha_w = 0.50$ we have the same β value for near 5% of the maps and better values than the original algorithm in about 40% of the maps. That is, in 45% of the maps the modification has no effect or has positive effects in this parameter. With more blocked cells, the original algorithm has better β values, but always we can found maps in which α has positive impact in the path planning algorithm. For Theta* we can observe a higher degradation as a consequence of both the percentage of blocked cells and the α_w factor. In this regard, the Alpha values work better in A*PS rather than in Theta*.

Table 3.1: Solutions with better β value over 5000 maps of 500x500 nodes clustered by the number of obstacles (each obstacle group has 1000 maps). For each obstacle group we present the number of maps in which the original algorithm (A*PS or Theta*) obtains better β values than the modified algorithm (for different α_w values in each column), or equals when both algorithm obtain the same β value.

5% blocked cells							
α_w	**0.25**	**0.50**	**1.00**	α_w	**0.25**	**0.50**	**1.00**
A*PS	311	378	439	**Theta***	478	525	569
A*PS H-Alpha	422	478	439	**Theta* H-Alpha**	79	78	71
Equals	267	144	122	**Equals**	443	397	360
10% blocked cells							
α_w	**0.25**	**0.50**	**1.00**	α_w	**0.25**	**0.50**	**1.00**
A*PS	431	545	655	**Theta***	685	751	798
A*PS H-Alpha	412	407	319	**Theta* H-Alpha**	118	108	93
Equals	157	48	26	**Equals**	197	141	109
20% blocked cells							
α_w	**0.25**	**0.50**	**1.00**	α_w	**0.25**	**0.50**	**1.00**
A*PS	574	751	850	**Theta***	816	868	912
A*PS H-Alpha	352	239	148	**Theta* H-Alpha**	153	112	79
Equals	74	10	2	**Equals**	54	20	9
30% blocked cells							
α_w	**0.25**	**0.50**	**1.00**	α_w	**0.25**	**0.50**	**1.00**
A*PS	681	826	920	**Theta***	816	893	935
A*PS H-Alpha	291	170	80	**Theta* H-Alpha**	165	103	64
Equals	28	4	0	**Equals**	19	4	1
40% blocked cells							
α_w	**0.25**	**0.50**	**1.00**	α_w	**0.25**	**0.50**	**1.00**
A*PS	721	893	948	**Theta***	822	889	967
A*PS H-Alpha	264	107	52	**Theta* H-Alpha**	173	111	33
Equals	15	0	0	**Equals**	5	0	0

Figure 3.6 summarizes the results presented above using average values. Left charts show the degradation of the path length and the heading changes as the α_w increases. Meanwhile, right charts present the relationship between the α_w and the runtime and number of expanded nodes. This last case is related to the memory and processor usage of the algorithm, which are smaller when the α_w increases. These results are provided for two maps groups with different number of blocked cells: 20% (top) and 40% (bottom). Please note that the original algorithm (without the modified heuristic) corresponds to the case in which the Alpha weight is 0.

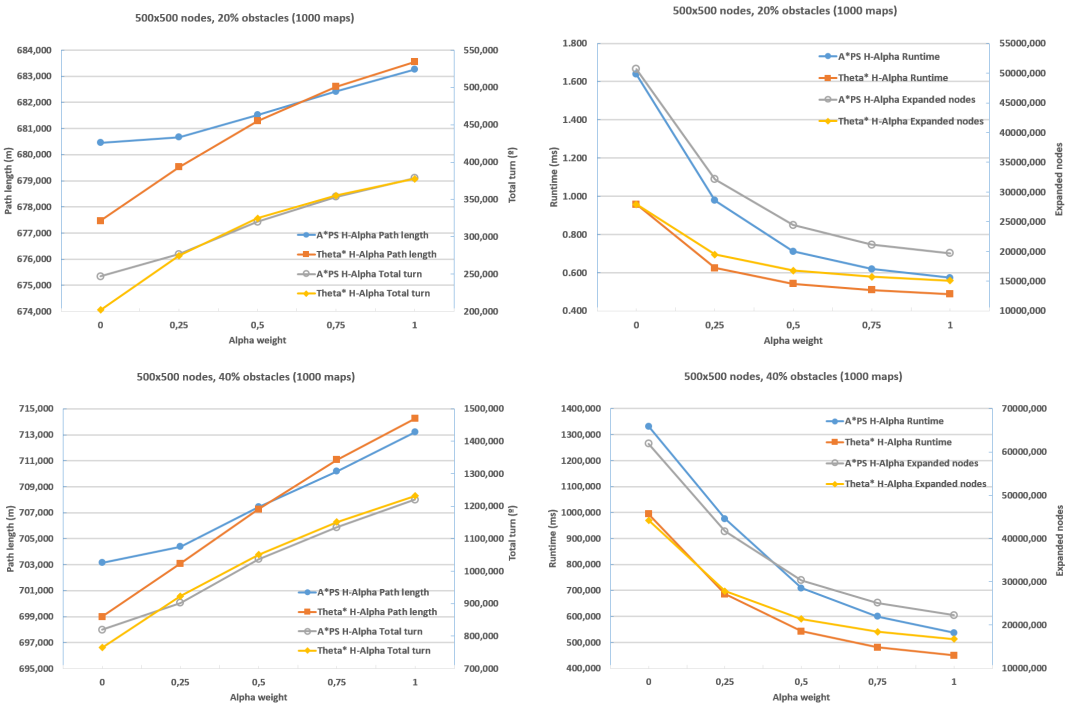


Figure 3.6: Trade-off between α_w and average values for path length/total turns (left) and runtime/expanded nodes (right), for groups of 1000 maps of 500x500 nodes with 20% of blocked cells (top) and 40% of blocked cells (bottom).

3.4 Heading changes as a cost function: the S-Theta* algorithm

If we consider the α value during the search as part of the cost function, G(p), we obtain a search algorithm that optimizes the combination of path length plus heading changes of the route, not only the path length as usually path planning algorithms do. An algorithm that provides smoother paths could be very desirable for some kinds of robots with limitations on turning. In the previous section we have employed the nodes p, s (start node) and g (goal node) to compute α, so the value obtained is fixed for each random node p. If we want to include the heading changes as part of the cost function we need to employ other nodes: the heading to reach a node is dependent on the path followed.

Taking into consideration that the cost function is the one that we try to minimize during the search, a good election on the nodes involved in the calculation of α could give us a minimization in the β_i value, and thus, in the total turn of the resultant path. Looking at the formulation of β_i in eq. 3.3, we could infer a good relationship between the three nodes: considering the expansion process, we want to reach the goal node g, and, from the current position, be it p, we are trying to expand the node t. The actual direction of the path is the one that follows the line which connects node p with its parent, q = parent(p). So, the deviation to reach the node t from the current position is defined by both, the actual heading and the turn required to go from p to t having in mind that we want to achieve the goal position.

Finally, we have that the relationship $q = \text{parent}(p)$ has important implications: employing α value as part of the cost function in A* does not improve the path: for A* we have that $q = \text{parent}(p) \Rightarrow p \in \text{successors}(q)$, so it is possible to locally reduce the zig-zag patterns but with a higher computational cost due to the computation of α. Then, we cannot apply it to A*PS neither.

So, we describe $\alpha(q, t, g)$ as the deviation in the trajectory measured from the current position, p, to reach the goal node g through the node t in relation to the current heading defined by the parent of its predecessor $q = \text{parent}(p)$ and the node $t \in \text{successors}(p)$. $\alpha(q, t, g)$ is graphically represented in fig. 3.7.

Taking into consideration the previous issues, we propose the Smooth Theta* (S-Theta*) algorithm [124] that we have developed from Theta*. It aims to reduce the amount of heading changes that a mobile robot should perform to reach the goal using the α value as part of the cost function. The evaluation function for the nodes, $\text{F}(t)$, in S-Theta* is computed as in eq. 3.8.

$$\text{F}(t) = [\text{G}(t) + \alpha(q, t, g)] + \text{H}(t) \tag{3.8}$$

The $\alpha(q, t, g)$ term (which is added to the accumulated path length) gives us a measure of the deviation from the optimal trajectory to achieve the goal as a function of the direction to follow, conditional to traverse a node t. Considering an environment without obstacles, the optimal path between two points is the straight line. Therefore, applying the triangle inequality, any node that does not belong to that line will involve both, a change in the direction and a longer distance. Therefore, α causes that nodes far away from that line will not be expanded during the search.

The result is that, once the initial direction has changed, the algorithm tries to find the new shortest route between the successor to the current position, t, and the goal node. The shortest route will be, if there are no obstacles, the one with $\alpha(q, t, g) = 0$, i.e., the route in which the successor of the current node belongs to the line connecting the parent node of the current position and the goal node. Figure 3.8 shows how the value of α evolves as the search progresses.

Algorithm 3 shows the pseudo-code of the function *UpdateNode* for S-Theta* (please note that the search process is the same as in A* or Theta*, see alg. 1 for more details). Alpha is included as a cost in the evaluation function of the nodes,

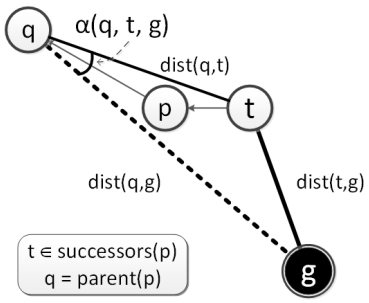


Figure 3.7: Graphical representation of $\alpha(q, t, g)$. Actual position is p with $q = \text{parent}(p)$. The successor considered is $t \in \text{successors}(p)$ and g is the goal node.

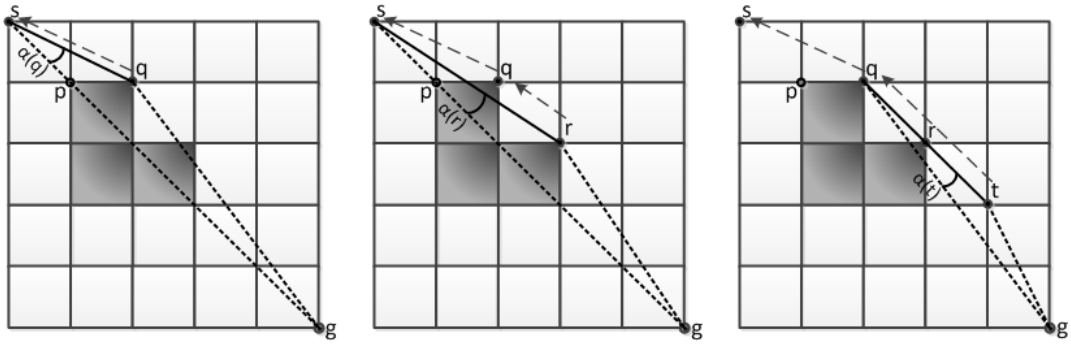


Figure 3.8: Representation of the evolution of α. Arrows are pointed to the parent of the node after expansion.

so the algorithm will also discriminate the nodes in the *open* list depending on the orientation of the search. Thus, a node in the *open* list may be replaced (which means that its parent will be changed) due to a lower value of α. In contrast, Theta* updates a node depending only on the distance to reach it, regardless of its orientation. As a result, the main difference with respect to Theta* is that S-Theta* can produce heading changes at any point, not only at the vertex of the obstacles. This difference can be seen in fig. 3.9, and in fig. 3.10, in which we also show example paths obtained for different algorithms in 500x500 nodes maps with 20% and 30% of blocked cells.

The results presented in the next section show that S-Theta* obtain better values for the total turn parameter than Theta*, with slightly longer path lengths.

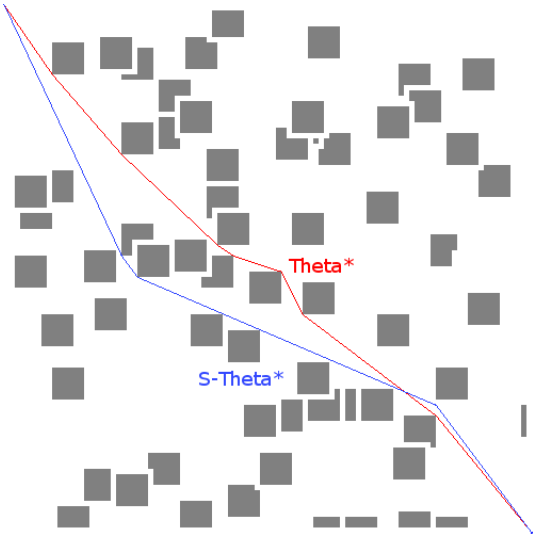


Figure 3.9: Resultant paths for Theta* (red) and S-Theta* (blue) in a random map. Theta* only has heading changes at vertices of blocked cells, while S-Theta* not. Path lengths are 142.28 and 147.82, and total turns 121.54° and 71.56° for Theta* and S-Theta* respectively.

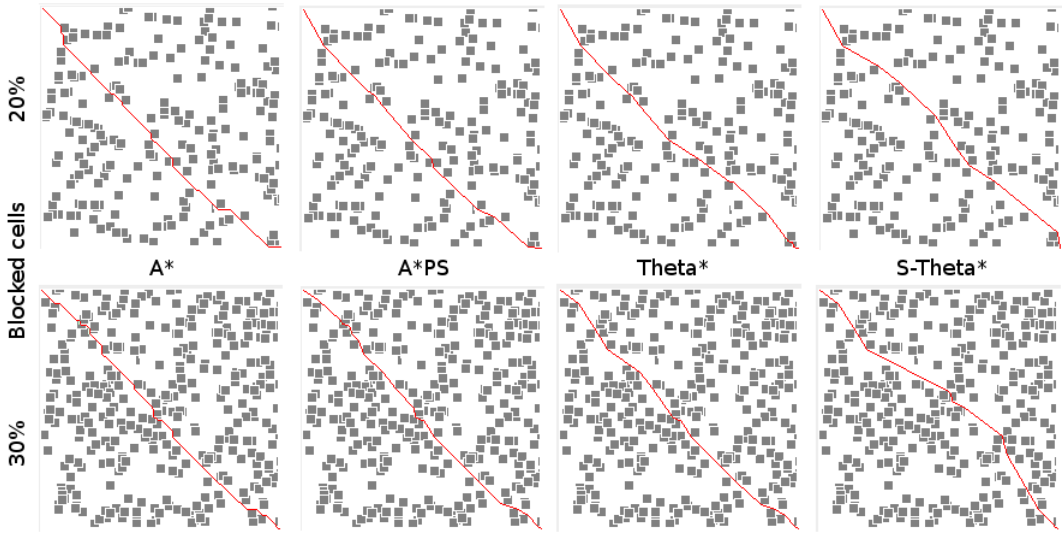


Figure 3.10: Solution paths for different algorithms in random generated maps.

Algorithm 3 Update vertex function for S-Theta*

```
1  UpdateNode(p, t)
2  if LineOfSight(parent(p), t) then
3      α_t ← α(parent(p), t, g)
4      G_aux ← G(parent(p)) + dist(parent(p), t) + α_t
5      if G_aux < G(t) then
6          G(t) ← G_aux
7          parent(t) ← parent(p)
8          if t ∈ open then
9              open.remove(t)
10         end if
11         open.insert(t, G(t), H(t))
12     end if
13 else
14     α_a ← α(p, t, g)
15     G_aux ← G(p) + dist(p, t) + α_a
16     if G_aux < G(t) then
17         G(t) ← G_aux
18         parent(t) ← p
19         if t ∈ open then
20             open.remove(t)
21         end if
22         open.insert(t, G(t), H(t))
23     end if
24 end if
```

3.5 S-Theta* experimental evaluation

In the same way as done in sec. 3.3, this section provides a comparison of heuristic search path planning algorithms in randomly generated maps in the same conditions as sec. 3.3 (we exploit the same generated maps and initial/goal positions). Particularly, we compare A*, A*PS, Theta* and S-Theta*. Also, A*PS and Theta* with the Alpha modified heuristic are included (considering only $\alpha_w = 1$). Figure 3.11 summarizes the results obtained for the resolution of 5000 random generated maps of 500x500 nodes, gradually increasing the percentage of blocked cells to 5%, 10%, 20%, 30% and 40% (each obstacle group has 1000 maps). For the experiments, we have taken into consideration the values for the following parameters: (i) the length of the path, (ii) the accumulated degrees by the heading changes (total turn or β) in degrees, (iii) the processor time or search runtime, and, (iv) the number of expanded nodes during the search. The number of heading changes is not provided as it is proportional in some way to the total turn parameter, while also this last one gives us more information about the objective of the analysis, i.e., minimize both, the path length and the total turn. The experiments are performed on a 2.5 GHz Intel Core i7 with 8 GB of RAM under Ubuntu 14.04.

As we can see, the algorithm that obtains the shorter paths is Theta* while A* obtains the worse performance (except for the case of 40% of blocked cells). From the modified algorithms, there is one that obtains closer path lengths to A* when the number of blocked cells increases: S-Theta*. For 40% of blocked cells, A* obtains an average path length of 730, Theta* gets 699 and S-Theta* gets 810. Moreover, S-Theta* has an important degradation in the path length respect to its former (Theta*) that grows from 0.45% with 5% of obstacles to 15.88% with 40% of blocked cells. Regarding the algorithms that employ Alpha as part of the heuristic function, we obtain less degradation than the obtained in S-Theta* for both cases, A*PS H-Alpha and Theta* H-Alpha. It is visible that the degradation is directly dependent on the number of blocked cells, being more remarkable with higher number of obstacles.

For the total turn parameter we obtain that S-Theta* has the best values (except for the minimum obstacles case) and, obviously, A* is the worst due to the constraint of heading changes to multiples of 45°. Here we can see that the difference in the total turns between Theta* and S-Theta* is bigger with more obstacles. Theta* requires 44.28% degrees more than S-Theta* for maps with 20% of obstacles (in average 202 degrees and 140 degrees for Theta* and S-Theta* respectively) and 320% degrees more when the blocked cells grow up to 40% (765 degrees for Theta* and 239 degrees for S-Theta* in average). This difference remarks that S-Theta* tends to maintain smoothed paths at the expense of the path length. For A*PS and Theta* with Alpha as part of the heuristic function we obtain the same behaviour of the path length: turn performance degrades as a function of both, the percentage of obstacles and the value of α_w factor.

In the case of the runtime, best values are achieved by Theta* using Alpha as part of the heuristic function. A*PS and Theta* using Alpha improves their performance in a factor close to 3 for the first algorithm and 2 for the second one. Both A*PS H-Alpha and Theta* H-Alpha have better runtime than A* in all cases (with the

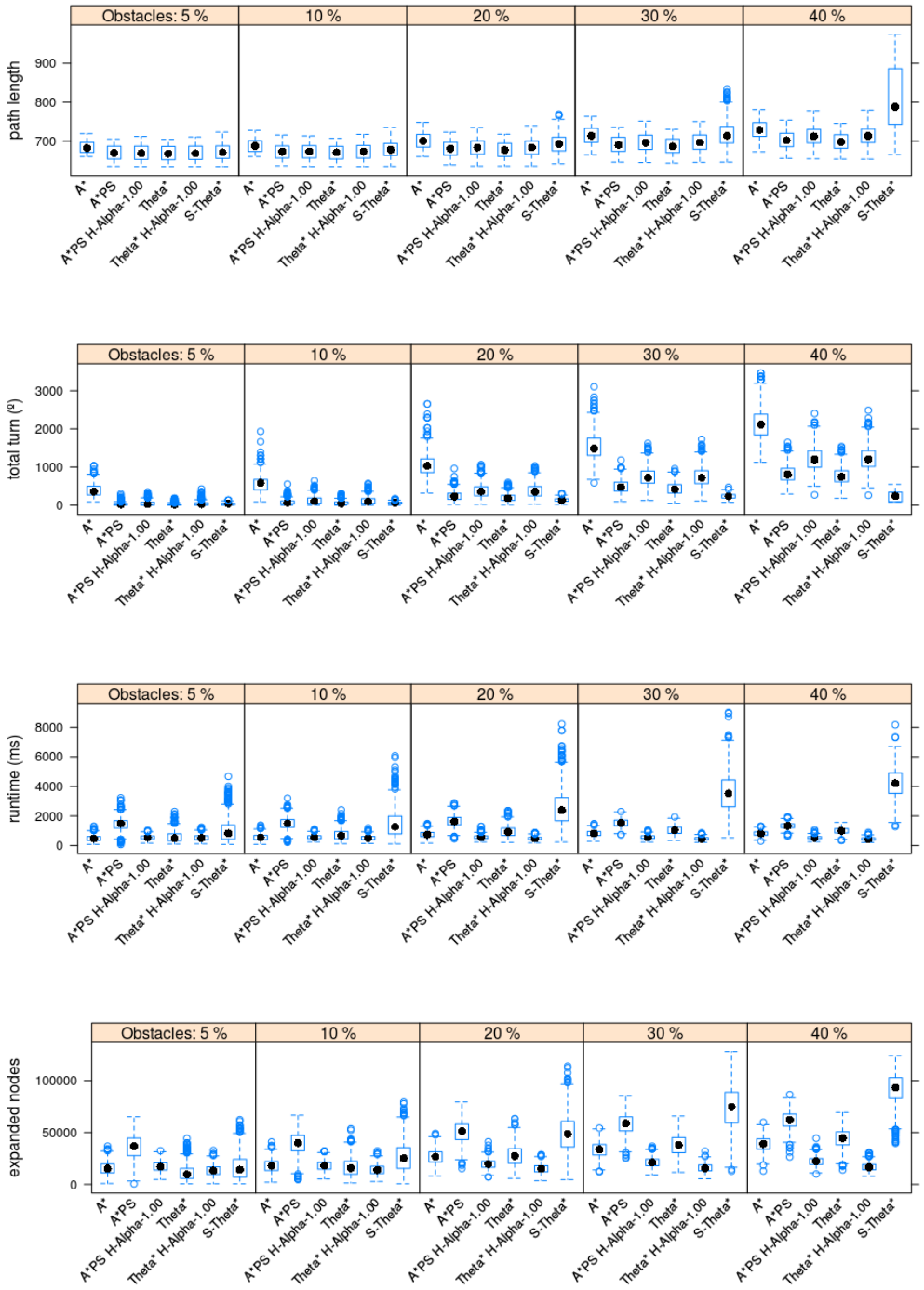


Figure 3.11: Results for the execution of different path planning algorithms over 5000 random generated maps (each obstacle group has 1000 maps). From top to bottom: path length, total turn in degrees, runtime in milliseconds and number of expanded nodes.

exception of A*PS H-Alpha and 5% of blocked cells). Also, we can observe that S-Theta* obtains the worse runtime. It is notorious a degradation in the performance when increasing the number of obstacles. This is due to the computational effort required to perform the Alpha computation and, as consequence of the minimization of the heading changes, S-Theta* requires to check the line of sight for longer segments, degrading more its performance. Moreover, as we see following, S-Theta* expands more nodes than the other algorithms. This is the result of trying to keep the heading during the search.

Finally, and directly proportional to the runtime, there is the number of expanded nodes. We want to remark the difference between the expanded nodes by A* and A*PS. This is because A*PS uses the *Euclidean* distance instead of the *Octile* one as A* does. Also, we can see that S-Theta* expands significantly more nodes than the former one, Theta*. As happens with the runtime, the number of expanded nodes significantly increases with the number of blocked cells.

To get a more accurate analysis of the optimality of the solution for the different path planning algorithms, in fig. 3.12 we graph the sum of the average path length and total turn against the percentage of blocked cells. We can observe that the worst result is for A*, followed by A*PS H-Alpha and Theta* H-Alpha (whose results are practically superposed). In fourth position there is A*PS. Then, for 5% and 10% of blocked cells the last two algorithms, Theta* and S-Theta*, have very similar results. However, for more obstacles, S-Theta* obtains better values, being more remarkable when the number of obstacles grow up to 40%. Then, S-Theta effectively reduces the heading changes of the path, at the expense of the degradation of the path length and the search runtime.

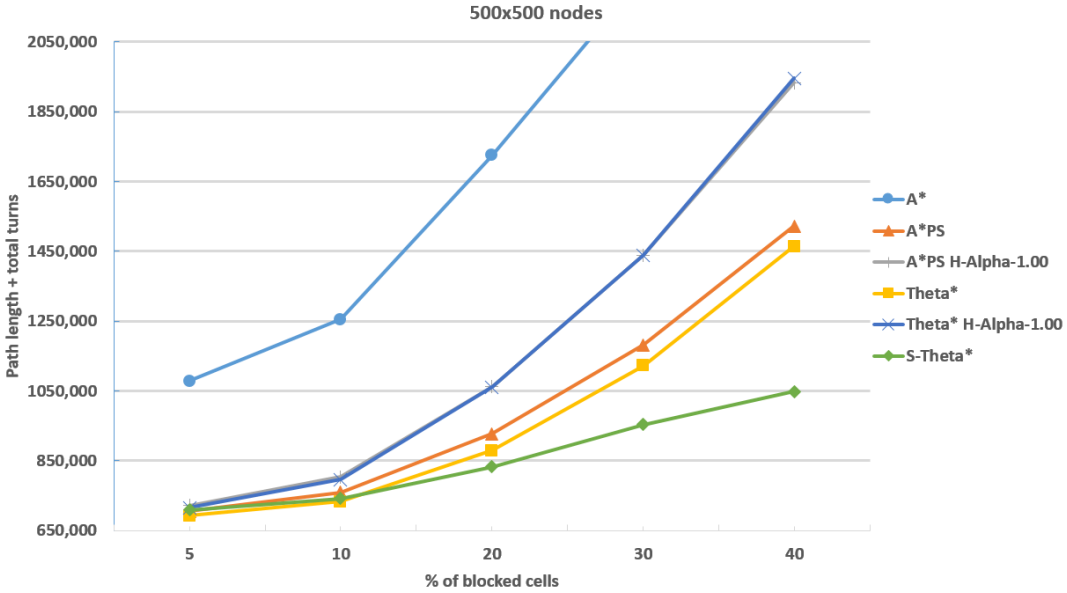


Figure 3.12: Evolution of path length plus total turn respect to the percentage of blocked cells for 5000 random generated maps.

3.6 Summary

Path planning for long term missions is a relevant feature in autonomous mobile robots. In this direction, the common objective is to reduce the distance travelled. However, in determinate circumstances, e.g., sand surfaces or robots with limited turning capability, the required heading changes of the path could also be relevant. For this reason, in this chapter we introduced the heading changes in heuristic search path planning algorithms. As result, we produced two outcomes. First, using the heading changes in the heuristic function in algorithms such as A*PS or Theta*, we can significantly reduce its runtime and memory required, at the expense of a degradation of the path length. And second, we developed the S-Theta* algorithm that reduces the number of heading changes and their amplitude with respect to its former algorithm, Theta*.

Chapter 4

Extending 2D path planning algorithms to 3D surfaces

As stated before, mobile robots require a path planner that can safely deal with realistic terrains, i.e., considering the relief or other characteristics [133]. The algorithms presented in the previous chapter works in flat terrain, which may not be enough to provide safe paths. Then, we propose the 3D Accurate Navigation Algorithm (3Dana) path planning algorithm, based on S-Theta* (see sec. 3.4), which integrates during the search the DTM information so it is able to avoid potentially dangerous areas. Particularly, it can discard movements that cross uneven terrains, generating safer routes. Besides, during the search process, the algorithm calculates the necessary turns needed to reach the next point taking into consideration the current heading and the position of the goal. This has two advantages: on the one hand, it could be used to obtain smoother routes; and on the other hand, it can avoid routes with abrupt heading changes, which is appropriate for soft terrains. Along this chapter we provide the description of 3Dana functionality, while at the end, an experimental evaluation is performed to demonstrate its capabilities under different maps, considering not only DTMs, but also traversability cost maps in either real and randomly generated terrains. Before presenting the 3Dana algorithm, the next section defines the terrain model and its mathematical representation.

4.1 Linearly interpolated DTM

In sec. 2.3 we have seen that 3D environments or DTMs have been used for path planning. However, some aspects about how the DTM is modelled remains unclear. In particular, there is not a common representation that enables path planning by means of heuristic search algorithms. Then, in this section we present a mathematical formulation for the DTM that can be used by heuristic search algorithms.

Starting from the 2D uniform grid we can transform it to a 3D grid or DTM by adding the elevation of each node. That is, for now on, any node p with coordinates (x_p, y_p) has an elevation z_p. Then, the terrain representation followed is a set of $k = n \times m$ spatial (three coordinates) points $(x_i, y_j, z_{i,j})$, where $1 \leq i \leq n$ and $1 \leq j \leq m$. We will call these points nodes as well. The ground projection of the terrain remains regular, i.e., the distance between two nodes with coordinates

(x_i, y_j) and (x_{i+1}, y_j), $1 \leq i < n$; $1 \leq j \leq m$ is constant and equally between two nodes (x_i, y_j) and (x_i, y_{j+1}), $1 \leq i \leq n$; $1 \leq j < m$. However, the distance between two nodes with coordinates $(x_i, y_j, z_{i,j})$ and $(x_{i+1}, y_j, z_{i+1,j})$ varies with the altitude of each node. A possible graphical representation can be seen in fig. 4.1, while a general outline is depicted in the following matrix:

$$\begin{matrix} (x_1, y_m, z_{1,m}) & (x_2, y_m, z_{2,m}) & \cdots & (x_n, y_m, z_{n,m}) \\ \vdots & \vdots & \iddots & \vdots \\ (x_1, y_2, z_{1,2}) & (x_2, y_2, z_{2,2}) & \cdots & (x_n, y_2, z_{n,2}) \\ (x_1, y_1, z_{1,1}) & (x_2, y_1, z_{2,1}) & \cdots & (x_n, y_1, z_{n,1}) \end{matrix}$$

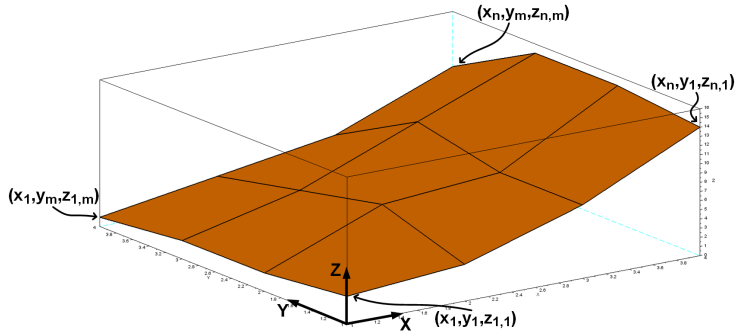


Figure 4.1: Representation of a DTM.

To fill the elevation data we can artificially generate each $z_{i,j}$ value, or we can use a DTM obtained, for example, from an on-board satellite instrument. In both cases, we get the elevation data for each point of the grid. This is enough to work with algorithms such as A* restricted to move between adjacent nodes. However, such restriction does not apply to any-angle algorithms such as Theta*. They can enter/exit cells at any point, in which the altitude is not known and can traverse long regions without crossing a node. This implies that we need to interpolate the elevation for points that not belong to the rectangular grid. The objective is to obtain the most approximate distance travelled by the robot when we are not forced to move between adjacent nodes.

First, considering the elevation, the cell formed by the four nodes $(x_i, y_j, z_{i,j})$, $(x_{i+1}, y_j, z_{i+1,j})$, $(x_i, y_{j+1}, z_{i,j+1})$ and $(x_{i+1}, y_{j+1}, z_{i+1,j+1})$, $1 \leq i < n$; $1 \leq j < m$ could not be (usually is not) a plane. Thus we need to define the geometry of the planes that conform the cell. We assume that each cell is composed by four triangles whose base is the connection between two nodes, while the two sides adjacent to the opposite angle are the joints between each node an the central point of the cell, $(x_{cij}, y_{cij}, z_{cij})$, as shown in fig. 4.2. The altitude at the central point is computed as the mean value of the heights of the four points that conform the square as in eq. 4.1. Coordinates x_c and y_c are computed as in eq. 4.2. This representation of the terrain is computationally complex when calculating the distance travelled, but unambiguous.

$$z_{cij} = \frac{z_{i,j} + z_{i+1,j} + z_{i,j+1} + z_{i+1,j+1}}{4} \tag{4.1}$$

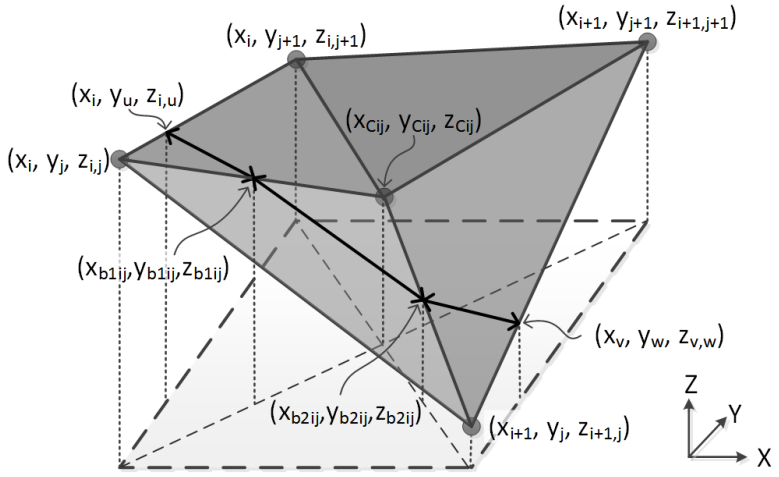


Figure 4.2: Lineal interpolation using four planes to define each cell.

$$x_{cij} = \frac{x_i + x_{i+1}}{2}; \quad y_{cij} = \frac{y_i + y_{i+1}}{2} \tag{4.2}$$

There are two possibilities to cross a cell. The first one is to move from one node p to another node t that is adjacent, i.e., $t \in \text{successors}(p)$. If the nodes are inside the same axis, the length is computed using the Pythagoras theorem. When the move is diagonal between opposite nodes, it is required to consider two segments: one from the node p to the central point, $(x_{cij}, y_{cij}, z_{cij})$, and from the central point to the t node. As the altitude of the central point is defined in eq. 4.1, the resultant length is, again, obtained using Pythagoras for the two segments.

The second one, when the movement starts in two arbitrary nodes p and t that are not adjacent, i.e., $t \notin$ successors (p), the path crosses more than one cell. This is a particularity of any-angle algorithms and requires a special treatment as the path entry/exit cells at intermediate points, not only at the vertex as A* does. Then, we need to compute the altitude for the entry and exit points to a cell. First, suppose that one of these entry/exit points is $(x_i, y_u, z_{i,u})$, $1 \leq i < n$, $u \in (j,\ j+1)$, then (x_i, y_u) will belong to the straight line between the two grid points (x_i, y_j) and (x_i, y_{j+1}), where $j = \lfloor u \rfloor$. Then, by triangle similarity, the altitude $z_{i,u}$ over (x_i, y_u) can be obtained as in eq. 4.3. This allows us to compute the elevation of any point between two adjacent nodes with the same x coordinate.

$$z_{i,u} = z_{i,j} + \frac{z_{i,j+1} - z_{i,j}}{y_{j+1} - y_j}(y_u - y_j) \tag{4.3}$$

If we suppose that the entry/exit point is $(x_v, y_j, z_{v,j})$, $v \in (i,\ i+1)$, $1 \leq j < m$, then (x_v, y_j) will belong to the straight line between the two grid points (x_i, y_j) and (x_{i+1}, y_j), where $i = \lfloor v \rfloor$. Then, the elevation $z_{v,j}$ is calculated as in eq. 4.4.

$$z_{v,j} = z_{i,j} + \frac{z_{i+1,j} - z_{i,j}}{x_{i+1} - x_i}(x_v - x_i) \tag{4.4}$$

So, let be $(x_i, y_u, z_{i,u})$ the entry point and $(x_v, y_w, z_{v,w})$ the exit point to a cell. Considering that $(x_i, y_u, z_{i,u})$ belongs to the line $\overline{(x_i, y_j, z_{i,j}), (x_i, y_{j+1}, z_{i,j+1})}$ as fig. 4.2 shows; then we can exit through one of the other three sides of the cell. According to the exit point, it is possible that we need to cross two or three planes of the cell. The possibilities are the following:

(a) Exit at the side defined by the straight line $\overline{(x_i, y_j, z_{i,j}), (x_{i+1}, y_j, z_{i+1,j})}$. Then the two planes formed by the points $(x_i, y_j, z_{i,j})$,$(x_{i+1}, y_j, z_{i+1,j})$, $(x_{cij}, y_{cij}, z_{cij})$ and $(x_i,\ y_{j+1}, z_{i,j+1})$ are crossed.

(b) Exit at the opposite side at a point that belongs to the line $\overline{(x_{i+1}, y_j, z_{i+1,j})}$, $\overline{(x_{i+1}, y_{j+1}, z_{i+1,j+1})}$ constrained to $\frac{y_v + y_w}{2} < y_{cij}$. Then three planes are crossed, and the one not crossed is the one formed by the points $(x_i, y_{j+1}$, $z_{i,j+1})$, $(x_{cij}, y_{cij}, z_{cij})$, and $(x_{i+1}, y_{j+1}, z_{i+1,j+1})$. This is the case presented in fig. 4.2.

(c) Exit at the opposite side at a point that belongs to the line $\overline{(x_{i+1}, y_j, z_{i+1,j})}$, $\overline{(x_{i+1}, y_{j+1}, z_{i+1,j+1})}$ constrained to $\frac{y_v + y_w}{2} = y_{cij}$. Then two planes are crossed, and the point $(x_{cij}, y_{cij}, z_{cij})$ belongs to the path. In this case the length of the path inside the cell can be computed using Pythagoras for the two segments.

(d) Exit at the opposite side at a point that belongs to the line $\overline{(x_{i+1}, y_j, z_{i+1,j})}$, $\overline{(x_{i+1}, y_{j+1}, z_{i+1,j+1})}$ and constrained to $\frac{y_v + y_w}{2} > y_{cij}$. Then three planes are crossed, and the one not crossed is the formed by the points $(x_i, y_j, z_{i,j})$, $(x_{cij}, y_{cij}, z_{cij})$ and $(x_{i+1}, y_j, z_{i+1,j})$.

(e) Exit at a point that belongs to the line $\overline{(x_i, y_{j+1}, z_{i,j+1}), (x_{i+1}, y_{j+1}, z_{i+1,j+1})}$. Then the two planes formed by the points $(x_i, y_j, z_{i,j})$, $(x_i, y_{j+1}, z_{i,j+1})$, $(x_{cij}$, $y_{cij}, z_{cij})$ and $(x_{i+1}, y_{j+1}\ z_{i+1,j+1})$ are crossed.

We can also consider entering the cell at a point that belongs to $\overline{(x_i, y_j, z_{i,j})}$, $\overline{(x_{i+1}, y_j, z_{i+1,j})}$, but it is equivalent to the previous cases presented and can be solved using symmetry. In the same way, following we present the formulation to compute the length traversed in a cell for cases (a) and (b); (d) and (e) are analogous, meanwhile (c) is trivial. Now, we need to compute the point(s) in which the straight line that connects the entry and exit points changes between the planes of the cell.

Focusing on the the enumerated cases with the points $(x_i, y_u, z_{i,u})$ and $(x_v, y_w$, $z_{v,w})$ we have that, in case (a), we cross two planes, so, only one point shall be computed, let it be $(x_{aij}, y_{aij}, z_{aij})$. For case (b) three planes are cut, so we need to compute two points, in the same way, be these points $(x_{b1ij}, y_{b1ij}, z_{b1ij})$ and $(x_{b2ij}, y_{b2ij}, z_{b2ij})$ as fig. 4.2 shows. To obtain these points we need the point (x_α, y_α), $\alpha \in \{aij,\ b1ij,\ b2ij\}$ and then interpolate the elevation at such point. The way to obtain the crossing points is to employ the equation of the line and obtain the point(s) in which the line $\overline{(x_i, y_u), (x_v, y_w)}$ and the diagonal(s) (defined by the lines $\overline{(x_i, y_j), (x_{i+1}, y_{j+1})}$ and/or $\overline{(x_i, y_{j+1}), (x_{i+1}, y_j)}$) intersects. For all cases, we defined a set of functions:

- $m(x_1, y_1, x_2, y_2)$ is the slope of a line which contains the points (x_1, y_1) and (x_2, y_2). It is computed as in eq. 4.5.

$$m(x_1, y_1, x_2, y_2) = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.5}$$

- $b(x_1, y_1, x_2, y_2)$ is the independent term of the line equation, computed as in eq. 4.6.

$$b(x_1, y_1, x_2, y_2) = y_1 - \frac{x_1 \cdot (y_2 - y_1)}{x_2 - x_1} \tag{4.6}$$

- $x_{cut}(m_1, b_1, m_2, b_2)$ is the x coordinate for the intersection of two lines defined by their slope and independent term. It is computed as in eq. 4.7.

$$x_{cut}(m_1, b_1, m_2, b_2) = \frac{b_2 - b_1}{m_1 - m_2} \tag{4.7}$$

- $y_{cut}(m_1, b_1, m_2, b_2)$ is the y coordinate for the intersection of two lines defined by their slope and independent term. It is computed as in eq. 4.8.

$$y_{cut}(m_1, b_1, m_2, b_2) = \frac{m_1 \cdot (b_2 - b_1)}{m_1 - m_2} + b_1 \tag{4.8}$$

Then, to compute the desired points, we define m_1, b_1, m_2 and b_2 as in eq. 4.9 for the diagonals of the cell. The intersection points for the path are computed as follows: for case (a) we only have the point $(x_{aij}, y_{aij}, z_{aij})$ as in eq. 4.10. For case (b), we have $(x_{b1ij}, y_{b1ij}, z_{b1ij})$ and $(x_{b2ij}, y_{b2ij}, z_{b2ij})$ as in eq. 4.11.

$$\begin{aligned} m_1 &= m(x_i, y_j, x_{i+1}, y_{j+1}); \quad b_1 = b(x_i, y_i, x_{i+1}, y_{i+1}) \\ m_2 &= m(x_i, y_{i+1}, x_{i+1}, y_j); \quad b_2 = b(x_i, y_{i+1}, x_{i+1}, y_j) \end{aligned} \tag{4.9}$$

$$\begin{aligned} x_{aij} &= x_{cut}(m_1, b_1, m(x_i, y_u, x_v, y_w), b(x_i, y_u, x_v, y_w)) \\ y_{aij} &= y_{cut}(m_1, b_1, m(x_i, y_u, x_v, y_w), b(x_i, y_u, x_v, y_w)) \\ z_{aij} &= (z_{cij} - z_{i,j}) \cdot \sqrt{\frac{(x_{aij} - x_i)^2 + (y_{aij} - y_j)^2}{(x_{cij} - x_i)^2 + (y_{cij} - y_j)^2}} + z_{i,j} \end{aligned} \tag{4.10}$$

$$\begin{aligned} x_{b1ij} &= x_{cut}(m_1, b_1, m(x_i, y_u, x_v, y_w), b(x_i, y_u, x_v, y_w)) \\ y_{b1ij} &= y_{cut}(m_1, b_1, m(x_i, y_u, x_v, y_w), b(x_i, y_u, x_v, y_w)) \\ z_{b1ij} &= (z_{cij} - z_{i,j}) \cdot \sqrt{\frac{(x_{b1ij} - x_i)^2 + (y_{b1ij} - y_j)^2}{(x_{cij} - x_i)^2 + (y_{cij} - y_j)^2}} + z_{i,j} \\ x_{b2ij} &= x_{cut}(m_2, b_2, m(x_i, y_u, x_v, y_w), b(x_i, y_u, x_v, y_w)) \\ y_{b2ij} &= y_{cut}(m_2, b_2, m(x_i, y_u, x_v, y_w), b(x_i, y_u, x_v, y_w)) \\ z_{b2ij} &= (z_{cij} - z_{i+1,j}) \cdot \sqrt{\frac{(x_{b2ij} - x_{i+1})^2 + (y_{b2ij} - y_j)^2}{(x_{cij} - x_{i+1})^2 + (y_{cij} - y_j)^2}} + z_{i+1,j} \end{aligned} \tag{4.11}$$

Finally, we can compute the distance travelled through the cell using Pythagoras. In case (b), the one presented in fig. 4.2, the length for a move between two given points $(x_i, y_u, z_{i,u})$ and $(x_v, y_w, z_{v,w})$ is computed as in eq. 4.12.

$$\begin{aligned} D &= \sqrt{(x_i - x_{b1ij})^2 + (y_u - y_{b1ij})^2 + (z_{i,u} - z_{b1ij})^2} \\ &+ \sqrt{(x_{b1ij} - x_{b2ij})^2 + (y_{b1ij} - y_{b2ij})^2 + (z_{b1ij} - z_{b2ij})^2} \\ &+ \sqrt{(x_{b2ij} - x_v)^2 + (y_{b2ij} - y_w)^2 + (z_{b2ij} - z_{v,w})^2} \end{aligned} \tag{4.12}$$

Besides the distance computation, we can provide also the terrain slope at each cell. To compute the slope we need to obtain the equation of each plane of the cell. Considering the plane formed by the points $(x_i, y_j, z_{i,j})$, $(x_{cij},\ y_{cij}, z_{cij})$ and $(x_{i+1}, y_j, z_{i+1,j})$, we can obtain the normal vector, $\overrightarrow{n_\pi}$, of the plane as in eq. 4.13. The angle obtained for such plane using the eq. 4.14, α_z, is the angle formed by the normal vector and the Z axis. This angle is the slope of the terrain. As each cell is divided into four planes, we can obtain four different slopes as fig. 4.3 shows.

$$\begin{aligned} \overrightarrow{n_\pi} &= (A, B, C) \quad \text{with:} \\ A &= (y_{cij} - y_j) \cdot (z_{cij} - z_{i+1,j}) - (z_{cij} - z_{i,j}) \cdot (y_{cij} - y_j) \\ B &= (z_{cij} - z_{i,j}) \cdot (x_{cij} - x_{i+1}) - (z_{cij} - z_{i+1,j}) \cdot (x_{cij} - x_i) \\ C &= (x_{cij} - x_i) \cdot (y_{cij} - y_j) - (y_{cij} - y_j) \cdot (x_{cij} - x_{i+1}) \end{aligned} \tag{4.13}$$

$$\alpha_z = \arccos \frac{C}{\sqrt{A^2 + B^2 + C^2}} \tag{4.14}$$

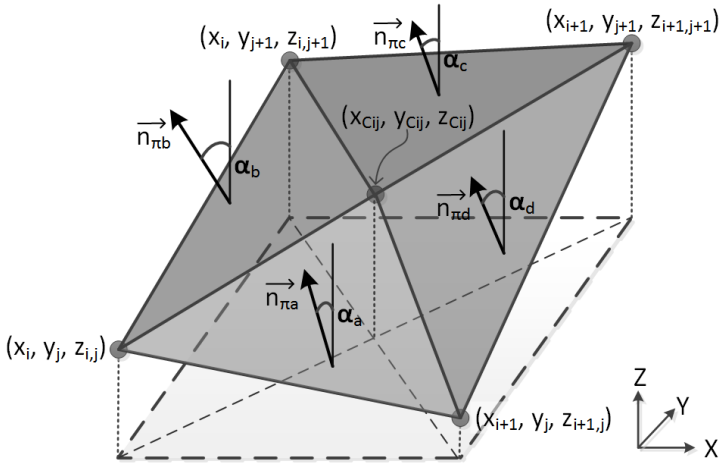


Figure 4.3: Four normal vectors for a cell in the lineal interpolation.

4.2 The 3Dana path planning algorithm

The 3D Accurate Navigation Algorithm (3Dana) [131] is a path planning algorithm developed to obtain safer routes based on heuristic search over a DTM and/or a traversability cost map. Its application scope is mobile robots in which the terrain characteristics (specially the surface relief) can affect their mobility. The main features of 3Dana are:

- Evaluation of potentially dangerous areas using a traversability cost map, as done in algorithms such as Field D*. A cost map defines one or more terrain

characteristics combined in a numerical factor. Using a cost map, the algorithm tries to minimize the path cost, avoiding high cost areas.

- Evaluation of the path distance using the terrain altitude. 3Dana performs path planning over a realistic surface model, using the DTM introduced in the previous section. The movement cost is a function of the distance between two points given their altitudes.

- Evaluation of heading changes during the search. Like the S-Theta* algorithm (see sec. 3.4), 3Dana calculates the necessary turns needed to reach the next position taking into consideration the current heading of the robot and the position of the goal. This has two advantages: first, it could be used to obtain smoother routes; and second, it is possible to avoid routes with abrupt heading changes.

- Evaluation of the terrain slope. 3Dana avoids paths that exceed the maximum slope allowed by the robot. This enables to obtain safer and feasible paths.

- Parametrization of the four above properties. 3Dana allows defining different configurations to use cost maps or DTMs alone or combined. Also, the slope and heading changes constraints can be set by the user.

The 3Dana algorithm is an evolution of the A* search algorithm that uses vertex re-expansion. Moreover, it takes advantage of the any-angle path planning algorithms such as Theta* or S-Theta*, applying them to a DTM and/or a traversability cost map. The search process in this new algorithm is similar to these algorithms, but adjusted to the mathematical model required to operate with the terrain relief. Next subsections introduce the algorithm functionality: first, the search and the nodes expansion processes are described. Following subsection shows how the line of sight between two nodes is calculated, which is directly related to the path length and cost calculation. Then, the slope consideration to avoid potentially infeasible paths is described. Finally, the heuristic computation and the heading changes evaluation is presented.

4.2.1 3Dana Search Process

The 3Dana search process is based on A* with node re-expansion. Algorithm 4 shows the pseudo-code for the search process employed. It is quite similar to A* but with two relevant differences. First, A* obtains better results using the *Octile* heuristic, while 3Dana uses a variation of the *Euclidean* distance. This last takes into consideration the altitude difference between the two nodes involved as explained in sec. 4.2.4. This difference is placed in line 6 of alg. 4. Second, the treatment of the nodes expansion (function *UpdateNode* at line 14) is different for both algorithms.

A* requires two nodes lists, *open* used to order the list of pending nodes to process (in ascending order, arranged by the F value), and *closed* with the expanded ones. However, 3Dana uses node re-expansion, so the *closed* list is no longer necessary; all nodes will be evaluated even if they are previously expanded. This may lead to

Algorithm 4 3Dana search algorithm

```
1  for p ∈ map do
2      G(p) ← ∞
3      parent(p) ← null
4  end for
5  G(s) ← 0
6  H(s) ← EuclideanZ(s, g)
7  open ← ∅
8  while open ≠ ∅ do
9      p ← open.pop()
10     if p = g then
11         return path
12     end if
13     for t ∈ successors(p) do
14         UpdateNode(p, t)
15     end for
16 end while
17 return fail
```

better paths, but increasing the runtime due to the possibility of expanding the same node several times.

At the beginning of the search, the *open* list only contains the start node, whose cost G is 0 and its heuristic is given by the *EuclideanZ* function (see sec. 4.2.4). The algorithm executes the search while there are nodes in the *open* list. When this list is empty, all the reachable nodes from the start position have been processed and none is the goal position. Thus, there is no feasible path between the desired points. Otherwise, the first node from the *open* list, that is, the most promising node, is extracted. If that node is the objective, the algorithm returns the path between the start and the goal through backtracking of the parents pointers of the nodes from the goal to the start. Otherwise, the successors(p) function returns a set with the visible adjacent nodes for the current node. In this point, A*-based algorithms without re-expansion discard nodes that are contained in the *closed* list: they have been expanded and, in a flat surface without traversal costs, expanding them again usually not rely on a better path. However, dealing with elevation and traversability cost maps implies that, maybe, the previous path is not the best. For instance, the algorithm can reach a point by climbing a hill, that can be more expensive that surrounding it. This usually implies to expand more nodes, and thus, the better path is discovered later during node re-expansion. Then, the adjacent nodes to the current position will be processed by the *UpdateNode* function.

The *UpdateNode* function shown in alg. 5 corresponds to the function employed by 3Dana. It is similar to the one used by Theta* but considering the possibility of node re-expansion. Theta* selects the route to follow (Path1 at line 11 or Path2, line 6) taking into consideration only the line of sight between the parent of the current position and the successor. This means that, if there is a line of sight between these two nodes, Theta* always chooses the Path2, the one that allows any-angle

Algorithm 5 Update node function for 3Dana

```
1  UpdateNode(pos, succ)
2  G_A ← AdyacentCost(pos, succ)
3  G_T ← SegmentCost(pos.parent, succ)
4  update ← false
5  if G_T > 0 and G_T ≤ G_A and G_T < G(succ) then
6      tentativeG ← G_T {Path2: any-angle}
7      tentativeH ← EuclideanZ(succ, g)
       +α(parent(pos), succ, g) · w_α
8      tentativeParent ← parent(pos)
9      update ← true
10 else if G_A > 0 and G_A < G(succ) then
11     tentativeG ← G_A {Path1: adjacent node}
12     tentativeH ← EuclideanZ(succ, g) + α(pos, succ, g) · w_α
13     tentativeParent ← pos
14     update ← true
15 end if
16 if update then
17     if (tentativeG + tentativeH) < F(succ) then
18         G(succ) ← tentativeG
19         H(succ) ← tentativeH
20         parent(succ) ← tentativeParent
21     end if
22     if parent(succ) ≠ null then
23         open.insert(succ)
24     end if
25 end if
```

routes. Obviously, this behaviour is undesirable for 3Dana, due to the necessity of considering the elevation of the terrain and the traversal cost of the region that it crosses.

3Dana needs to calculate the path cost for both possibilities, the any-angle path and the adjacent node path (i.e., the one followed by A*). This is required as the algorithm has to decide which path is better considering the terrain relief and cost. To do this we have implemented two functions that will be reviewed in sec. 4.2.2 and whose descriptions are as follows:

- *AdyacentCost* (alg. 5 line 2): is used to calculate the cost for the adjacent nodes, that is, those that follow Path1. This function returns a positive value since there is always line of sight between a node and its successors.

- *SegmentCost* (alg. 5 line 3): performs the line of sight check and calculates the cost associated to the line that connects two arbitrary nodes. It represents the Path2. When it is not possible to move between the desired nodes (the path is blocked by an obstacle), it returns a negative value.

When the costs of both paths have been calculated, the algorithm chooses the any-angle path (Path2) when there are no obstacles and the cost is lower than the one calculated for Path1. In both cases, if the cost to reach the successor node is less than the cost to reach that node from a different one (if it was previously reached), the heuristic of the successor must be calculated (as shown in sec. 4.2.4) and the cost and the parent have to be set. For Path1, the parent is the current position and the cost is calculated through the function *AdyacentCost*. While for Path2 the successor's parent is the parent of the current position and the cost is previously computed using the *SegmentCost* function. When the cost to reach the node under evaluation is lower than its previous value, its F value is updated accordingly to the path selected and, if its parent is *null* (that is, it is not in the *open* list), it is included in the *open* list.

4.2.2 Line of sight and cost calculation

In the *UpdateNode* function there are two methods to compute the path cost between two nodes: the one associated with the adjacent nodes (the path followed by A*); and the one that includes the line of sight check for arbitrary nodes (the any-angle path). In the first case the line of sight is guaranteed: the successors function returns a list with the reachable adjacent nodes. Then, we compute the path cost by obtaining the length between adjacent nodes (as in sec. 4.1) and multiplying it with the value of the crossed cell.

In the case of an arbitrary pair of nodes, i.e., any-angle path, we need to perform the line of sight check and the cost calculation. To do this, we employ an algorithm similar to the Cohen-Sutherland clipping algorithm [55]. The procedure is implemented in the *SegmentCost* function and divided in two phases. First, it is required to compute the points in which the line that connects the two nodes intersects with the horizontal or vertical axes. Second, for each segment formed by two consecutive points, it is required to compute the length between points and the cost related to the region that it crosses.

For the first step, we have implemented the algorithm shown in alg. 6 to compute the axes intersection points. It returns an ordered list of points that intersects the axes and belong to the segment $\overline{p_0, p_n}$. This algorithm allows us to compute the points in any situation, but if the line is parallel to one of the axes (i.e., $\Delta x = 0$ or $\Delta y = 0$) a faster computation using integers is performed (not shown here), improving the computational effort required. Figure 4.4 shows an example of the points list that the alg. 6 computes for the line that connects the nodes p_0 and p_6. The resultant point list is the ordered set of points $\{p_0, p_1, p_2, p_3, p_4, p_5, p_6\}$.

Once the points list has been calculated, the second step of the *SegmentCost* function is to process each segment formed by two consecutive points. Then, we need to perform the following computation for each two points, p_i and p_{i+1}:

1. Take the cost value of the cell that contains the segment $\overline{p_i, p_{i+1}}$. If the cell is an obstacle then the function ends, returning a negative value; there is no line of sight. Otherwise, the cell cost is stored in a variable called *cellCost*. If no cost map is provided, *cellCost* is always 1.

Algorithm 6 Algorithm to compute axes intersection points for a segment

```
1  SegmentPoints(p, t)
2  pointList ← ∅
3  step_x ← if t_x ≥ p_x then 1 otherwise -1
4  step_y ← if t_y ≥ p_y then 1 otherwise -1
5  m ← (t_y − p_y)/(t_x − p_x)
6  Xx ← p_x + step_x
7  Yy ← p_y + step_y
8  Yx ← p_x + (Yy − p_y)/m
9  Xy ← p_y + (Xx − p_x) · m
10 while Xx < t_x and Yy < t_y do
11     if (Xx ≤ Yx and step_x > 0) or (Xx ≥ Yx and step_x < 0) then
12         pointList.insert(Xx, Xy)
13         Xx ← Xx + step_x
14         Xy ← p_y + (Xx − p_x) · m
15     else
16         pointList.insert(Yx, Yy)
17         Yy ← Yy + step_y
18         Yx ← p_x + (Yy − p_y)/m
19     end if
20 end while
21 return pointList
```

2. If the user has defined a maximum slope, the algorithm computes the terrain slope of the cell using the normal vector for the terrain. If the slope is higher than the maximum defined, the function returns a negative value as if the way is blocked by an obstacle. More details about the maximum slope are presented in sec. 4.2.3.

3. Compute the length of the segment $\overline{p_i, p_{i+1}}$ using the lineal interpolation procedure. To do this, the algorithm employs the equations introduced in sec. 4.1. The length obtained is *long*.

4. The associated cost of the segment $\overline{p_i, p_{i+1}}$ is calculated as $long \cdot cellCost$.

Following this procedure we obtain a negative value if there is no line of sight (or the slope exceeded the user defined constraints), or a cost for the path that is the sum of all $\overline{p_i, p_{i+1}}$ segments. In the example of fig. 4.4, the total cost associated to the path between the nodes p_0 and p_6 is given by eq. 4.15. We consider that the number of the cells corresponds to the traversal cost matrix, and $A2$ is the value of the traversal cost associated to cross the cell A2.

$$\begin{aligned} SegmentCost(p_0, p_6) = \overline{p_0p_1} \cdot A2 + \overline{p_1p_2} \cdot B2 + \\ \overline{p_2p_3} \cdot C2 + \overline{p_3p_4} \cdot C3 + \overline{p_4p_5} \cdot D3 + \overline{p_5p_6} \cdot E3 \end{aligned} \tag{4.15}$$

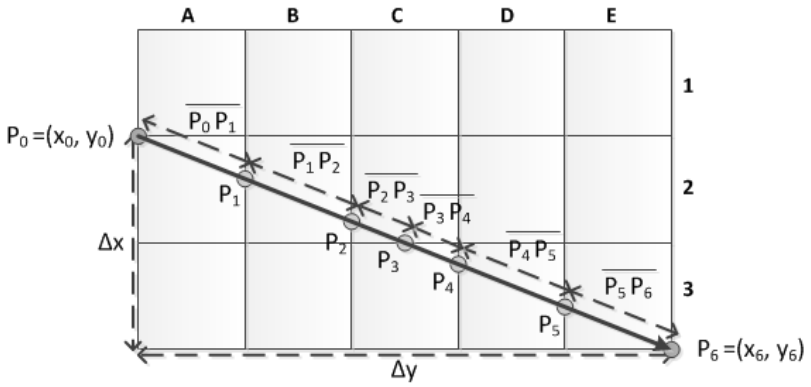


Figure 4.4: Line of sight evaluation.

4.2.3 Terrain slope consideration

When we deal with the terrain relief, 3Dana considers it in two ways: (i) computing the distance travelled, and (ii) analysing the slope to avoid dangerous areas. The objective is to provide an algorithm that is able to obtain feasible paths based on the DTM information. For example, it is possible to generate a cost map that integrates in some way the DTM information, but that approach simplifies the terrain and remove relevant data, i.e., we cannot provide paths constrained by the slope.

Consider the paths presented in fig. 4.5: both paths are obtained using 3Dana over a DTM file (without employing a cost map) with a variation of the maximum slope allowed. The left path, without slope limitation, is highly undesired: it crosses a crater. Meanwhile, when the maximum slope is setted to 15°, the algorithm avoids the crater and surrounds it, obtaining a longer but safer path.

To deal with the terrain slope, 3Dana computes the slope for each cell crossed by the path. In the *SegmentCost* function, the slope is evaluated during the line of sight checking. As each cell is formed by four planes (each one with its associated normal vector, see fig. 4.3), we need to analyse what planes are crossed during the movement. In this regard, 3Dana obtains the slope of each plane crossed and compute the average value as the slope for such cell. In the case that the movement belong to an axis, the value used is the average slope of the two planes that share the line that behaves to the movement. Then, if the computed slope is higher than the constraint imposed by the user, 3Dana discards such path. In the case that the path belongs to an axis, only the the vectors that are adjacent to the path are considered.

4.2.4 Heuristic and heading changes

The heuristic employed by 3Dana is implemented in the *EuclideanZ* function as in eq. 4.16. This is a variant of the *Euclidean* distance that takes into consideration the altitude difference between points, and thus, it prioritizes nodes without elevation changes.

$$EuclideanZ(p,t) = \sqrt{(x_t - x_p)^2 + (y_t - y_p)^2 + (z_t - z_p)^2} \tag{4.16}$$

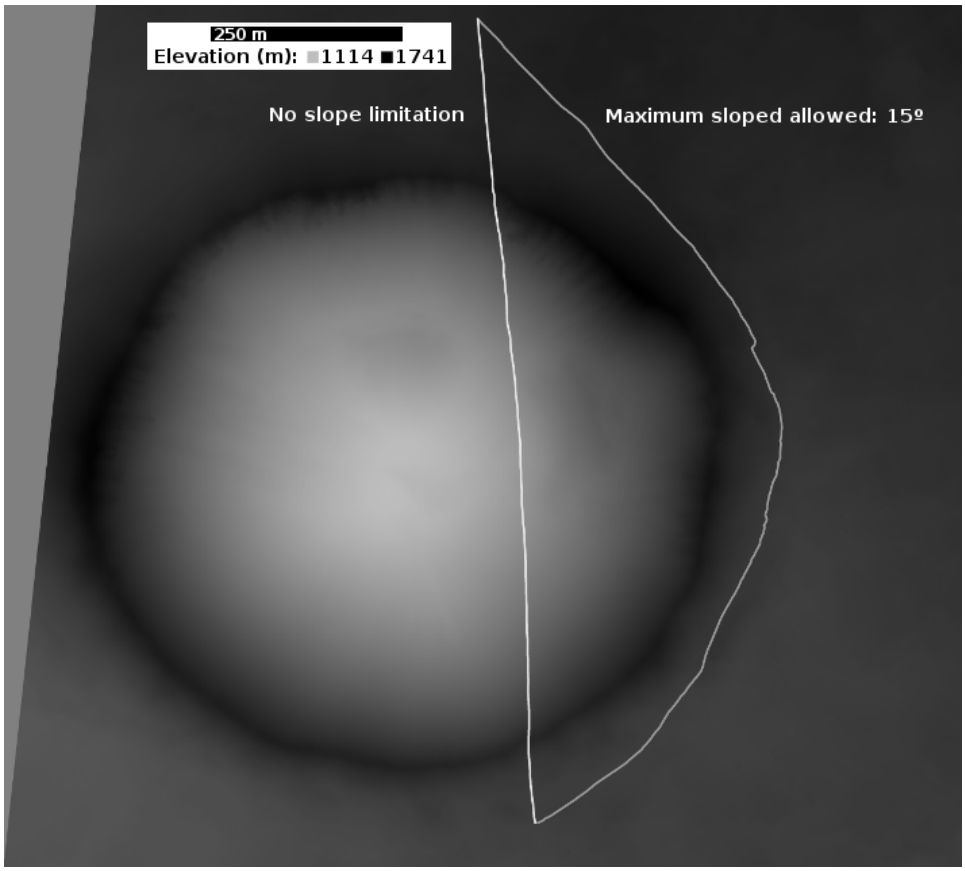


Figure 4.5: No slope limited path versus limited slope path over a DTM.

Besides this heuristic, 3Dana uses the value provided by the Alpha function (or α) inherited from S-Theta* to evaluate the heading changes during the path search. Given two arbitrary nodes, p, t, and the goal node, g, $\alpha(p, t, g)$ gives a measure of the heading changes necessary to reach the node t from p and facing it with the required heading to reach the goal node. This value is computed as in eq. 3.1 (see sec. 3.1). If this value is 0, then the three nodes involved in the calculation are in the same line, and the t node is nearest to the goal than p. When it returns 180, it happens the opposite: the three nodes are in the same line, but t is farthest from the goal than p.

Using this value in the heuristic enables 3Dana to consider the heading changes during the search process, delaying the expansion of nodes that require a high turn to be reached. This means that the algorithm tries to minimize the distance, but considering first paths with fewer turns. The Alpha function gives a value in the interval [0°, 180°], so, modifying the weight of this value we can change the order of the *open* list. Small weights imply a soft restriction in the heading changes, meanwhile bigger weights make the algorithm trends to follow smoother paths with less heading changes. We previously called this weight α_w (see sec. 3.2), which takes values in the range 0 (heading changes are not considered during search) and 1. Experimentally, we have seen that values higher than 1 do not rely on improvements in the heuristic.

4.3 3Dana experimental evaluation

In this section we present the assessment of 3Dana in different scenarios with randomly generated maps: exploring only traversability cost maps and dealing with both, elevation and traversability costs maps. The map generation algorithm for random maps can be seen in the appendix A. Please note that we do not consider maps with more than 10% of obstacles since increasing that number implies that the algorithm will avoid the obstacles rather than consider the terrain cost/relief. Finally, we assess 3Dana using only the DTM information from real Mars maps. All experiments are carried out on a 2.5 GHz Intel Core i7 with 8 GB of RAM under Ubuntu 14.04, except the ones presented in the next section, which are executed on a 2014 Mac Pro with a 3.9 GHz Intel Xeon E5 and 64 GB of RAM.

4.3.1 Random cost maps

First, an evaluation is performed considering only traversability cost maps. In this regard, we have compared 3Dana with A* and Theta*. These algorithms are not ready to work with traversability cost maps and/or DTMs, so we have adjusted both algorithms. To do this, in A* we have used a modified version of the *Octile* distance that considers the elevation difference between nodes. As well, we have adjusted Theta* by replacing the original line of sight evaluation with the one presented in sec. 4.2.2 and the heuristic with the *EuclideanZ*. Then, both algorithms use the terrain model presented in sec. 4.1. Also, thanks to a member of the Mars rover operation team, we have been able to use in our comparison the Field D* version employed at Jet Propulsion Laboratory (JPL) for the Mars rovers operations. In the case of 3Dana, different configurations for the heading change parameter (i.e., different values of α_w) has been considered. Figure 4.6 shows the results obtained for the resolution of 1500 randomly generated cost maps of 500x500 nodes, gradually increasing the percentages of blocked cells from 0% to 10% (each obstacle group has 500 maps). In these experiments, the DTM is not considered, i.e., the terrain is flat. In all cases the initial position corresponds to the coordinate (0, 0) and the objective is to reach a node in the last column randomly chosen the row from the bottom fifth (499, 460–499).

The algorithm that obtains better routes in terms of the path cost (i.e., the path length multiplied by the traversal cost) is 3Dana without considering the heading changes during path search ($\alpha_w = 0.00$), followed by 3Dana with increasing values of α_w, Field D*, A* and, finally, Theta*. This happens for all number of blocked cells, being quite similar the result obtained for Field D* and 3Dana 1.00 and 10% of blocked cells. It is remarkable that the path cost of 3Dana increases as α_w increases.

In the total degrees turned (the total amount of heading changes), we can observe that higher α_w values effectively reduce the total turns for 3Dana. Also, all algorithms provide smoother paths than Field D*, being the best one for Theta*, followed by the different configurations of 3Dana.

For the runtime, it is notorious that Field D* has a very low runtime to compute the path, while other algorithms require between 5 seconds and more than a minute in the worst case. Particularly, while all algorithms have been executed in the same

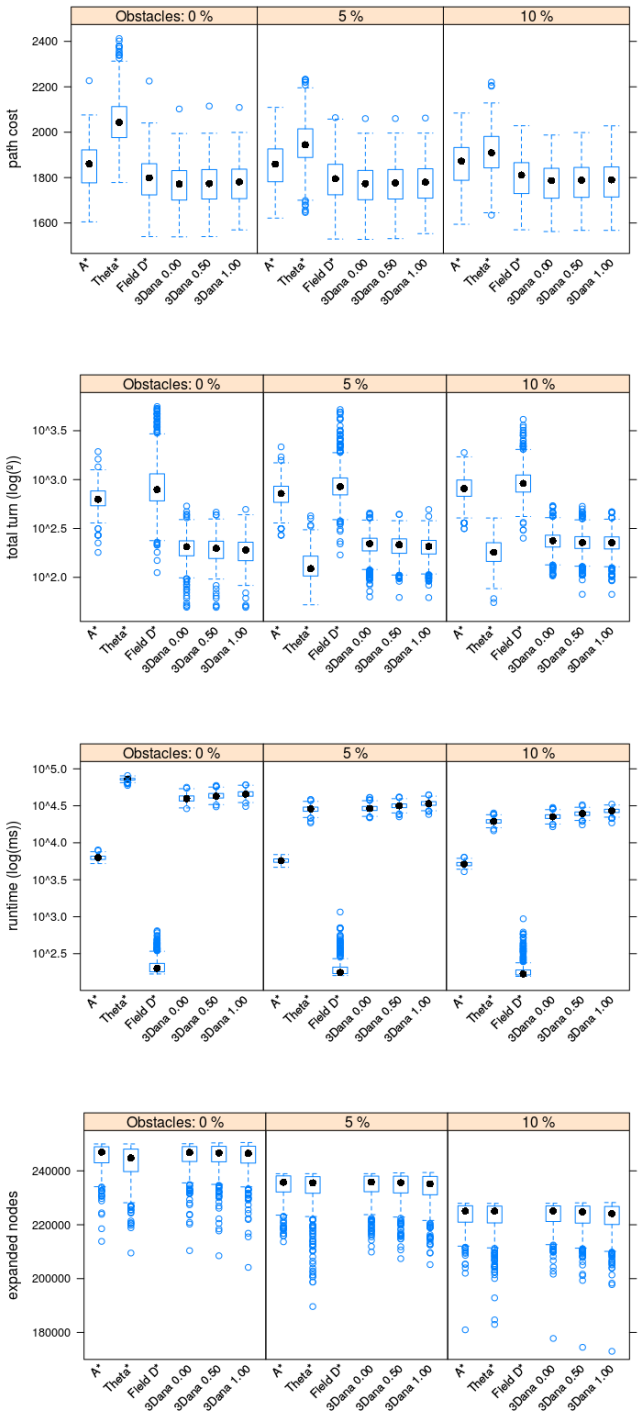


Figure 4.6: Results for the execution of different path planning algorithms over 1500 randomly generated cost maps with 0%, 5% and 10% of obstacles (each group has 500 maps). From top to bottom: path cost, total turn in degrees, runtime in milliseconds and number of expanded nodes (note that we do not have such value for Field D*). The number after 3Dana identifies the α_w value.

machine, Field D* is implemented in C++ and is very well optimized, while the others algorithms are implemented in Java. Then, apart from Field D*, A* obtains better runtime than the other algorithms, followed by Theta*, with the exception of the scenario without obstacles. In such situation, Theta* does not perform any heading change. Thus, each time a node is expanded, the line of sight check has to evaluate farther nodes, requiring more time for each evaluation. Then, the runtime is related to the line of sight check, not to the search process. Finally, 3Dana increases its runtime proportionally to the α_w value.

With respect to the expanded nodes, all the algorithms obtain similar values for each map group (note that the Field D* values are not displayed since the implementation used at JPL does not calculate this value). However, 3Dana expands nearly the same number of nodes independently of the α_w value. Although the runtime increases with α_w, the expanded nodes remain almost constant. The reason for this behaviour is that 3Dana re-expands some of the nodes in order to reduce the heading changes. This also has an important impact on the line of sight check: 3Dana has to check longer segments, degrading its performance.

4.3.2 Combined random cost maps and DTMs

Previous subsection has presented an evaluation of 3Dana in flat terrains. In this section we include the terrain relief, so the algorithm will search a path considering both, the cost map and the terrain elevation. In this regard, to create the terrain elevation we use a hill algorithm [139], normalizing the elevation in the range [0, 125]. As in the previous subsection, we compare 3Dana and the adjusted versions of A* and Theta*. However, we cannot adapt Field D* since we do not have access to the code, so those results are omitted.

Figure 4.7 shows the results obtained for the 1500 randomly generated maps. Each map consist of a traversability cost map and a terrain elevation, i.e., DTM. We have gradually increased the percentage of blocked cells from 0% to 10% (each obstacle group has 500 maps). As in the previous section, maps have a size of 500x500 nodes and the initial position corresponds to the coordinates (0, 0) and the objective is to reach a node in the last column randomly chosen the row from the bottom fifth.

Regarding to the path cost, 3Dana obtains better results in all obstacles group. Also, as happened with the results presented in the previous subsection, the path cost slightly increases with higher α_w values. As well, looking at the total turn value, it decreases as α_w increases. Then, the heading changes heuristic is providing significant smoother paths. However, the best algorithm for providing the lower heading changes is Theta*, but at the expense of higher path costs.

With respect to the runtime, A* outperforms the other algorithms. Focusing on 3Dana, the runtime grows as the α_w value increases in all the obstacles group. Nevertheless, Theta* obtains the worst runtime as the line of sight evaluation significantly degrades its performance.

Looking at the nodes expanded, we cannot appreciate significant differences; all algorithms expand nearly the same number of nodes. 3Dana behaves the same as in the previous experiments in relation to the runtime and the re-expansion of nodes.

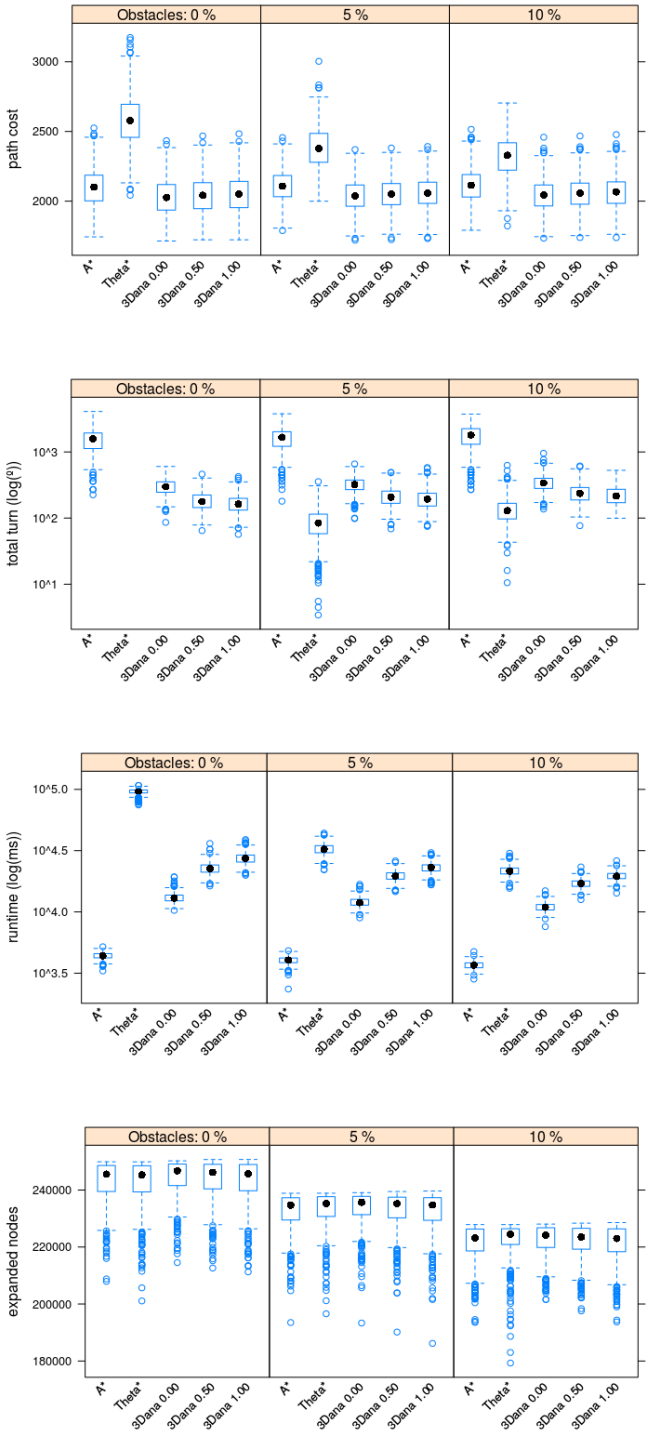


Figure 4.7: Results for the execution of different path planning algorithms over 1500 randomly generated maps (each obstacle group has 500 maps), considering either the cost map and the terrain altitude (DTM). From top to bottom: path cost, total turn in degrees, runtime in milliseconds and number of expanded nodes. The number after 3Dana identifies the α_w value.

4.3.3 Real Mars DTMs

In this section, we test the behaviour of the 3Dana algorithm using only the information of a DTM (without a traversability cost map) of real surfaces instead of randomly generated ones. The altitude data is obtained by the Mars Reconnaissance Observer (MRO) spacecraft probes using the High Resolution Imaging Science Experiment (HiRISE). With such data, it is possible to generate DTMs [90] that are publicly accessible[1]. The DTMs provide an elevation grid with a horizontal resolution from 0.25 to 2 meters and a vertical accuracy of 25 centimetres. All maps used in this article have a horizontal resolution of 2 meters, i.e., we have a uniform grid with elevation points every two meters. From the available DTMs we have selected five maps and, on them, we have run the modified A* algorithm and 3Dana with different α_w values (i.e., $\alpha_w \in \{0.0, 0.5, 1.0\}$).

For the slope limitation, we have considered from *no slope* limitation to decrement the maximum slope allowed from 30° to 10°. In this section we present two of these maps; the data for the others three can be checked in the appendix B. As well, we attempted to execute Theta*, but the runtime was excessive because there were no obstacles, and thus, Theta* trends to follow long paths without heading changes, degrading the performance due to the line of sight checking (as shown in sec. 4.3.1).

The first map considered presents a central structure and layered bedrock in a 30-kilometre diameter crater[2] in the Noachis Terra region[3]. The total area covered is near 40 km^2. The dimension in nodes is 3270 x 6636. In this map we set the initial point to the coordinates (700, 1500) and the goal to (2800, 6000). Then, we have run A* and 3Dana with the configurations defined above. The results obtained for the different paths are given in table 4.1, and some of them are depicted in fig. 4.8.

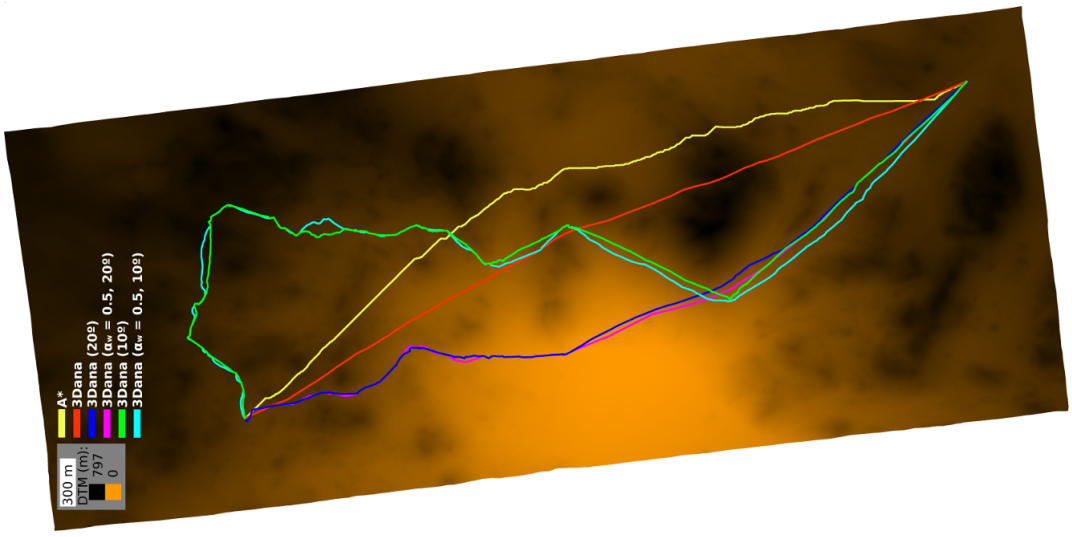


Figure 4.8: Paths obtained for the DTEEC_017147_1535 using A* and different configurations of 3Dana. Image rotated 90°.

[1]DTMs are available at `http://hirise.lpl.arizona.edu/dtm`
[2]`http://uahirise.org/dtm/dtm.php?ID=ESP_017147_1535`
[3]The location in Mars of the selected areas can be seen in fig. B.1.

Table 4.1: Paths data for DTEED_017147_1535. In bold: best path length plus total turns for each maximum slope.

Alg.	Max. slope	α_w	Length (m)	Turn (°)	Time (s)	Expanded nodes
A*	-	-	11010	31770	1120	2497903
3Dana	-	-	**10189**	**751**	776	2503970
A*		-	11079	37260	1347	2873747
	30°	0.0	10266	1676	1158	2560630
3Dana		0.5	10326	1402	4288	2689312
		1.0	**10340**	**1355**	6016	2687201
A*		-	11090	34245	1031	2427605
	25°	0.0	10354	2372	1056	2445298
3Dana		0.5	**10403**	**2203**	2770	2769846
		1.0	10467	2635	5147	2911966
A*		-	11427	39915	1023	3032953
	20°	0.0	10822	4918	1842	4087662
3Dana		0.5	**10887**	**3742**	4112	3870873
		1.0	11022	3720	5670	3990782
A*		-	12834	56475	1237	4902528
	15°	0.0	12116	7404	2196	5403052
3Dana		0.5	12235	7017	4039	5230152
		1.0	**12336**	**6887**	6332	5140151
A*		-	15434	73260	833	5213506
	10°	0.0	14703	10815	1617	5514326
3Dana		0.5	**14979**	**10355**	2485	5384046
		1.0	15193	10923	3147	5391994

Given the data, we can observe that the path length and the heading changes obtained by 3Dana are better than the values obtained by A* for all configurations. As we consider maximum slopes for the path, we can see that as higher is the restriction imposed (i.e., smaller slope allowed), both, the path length and the total turns increase. This is specially notorious when we restrict the slope to 10°, in which the path length is 1.35 times longer that the path restricted to 20° (without considering heading changes). Regarding to the heading changes, we can appreciate that, higher α_w values effectively reduces the total turn parameter. However, we can see cases in which using $\alpha_w = 1.0$ the values of the heading changes increases respect to its previous configurations. For instance, this happens with maximum slopes of 25° and 10°. While this parameter works fine in flat environments (see sec. 4.3.1), considering the elevation seems to affect negatively sometimes. Particularly, attempting to avoid heading changes can lead to follow longer paths, as we can discard better paths (as function of the slope) in spite of reducing the turns. Also,

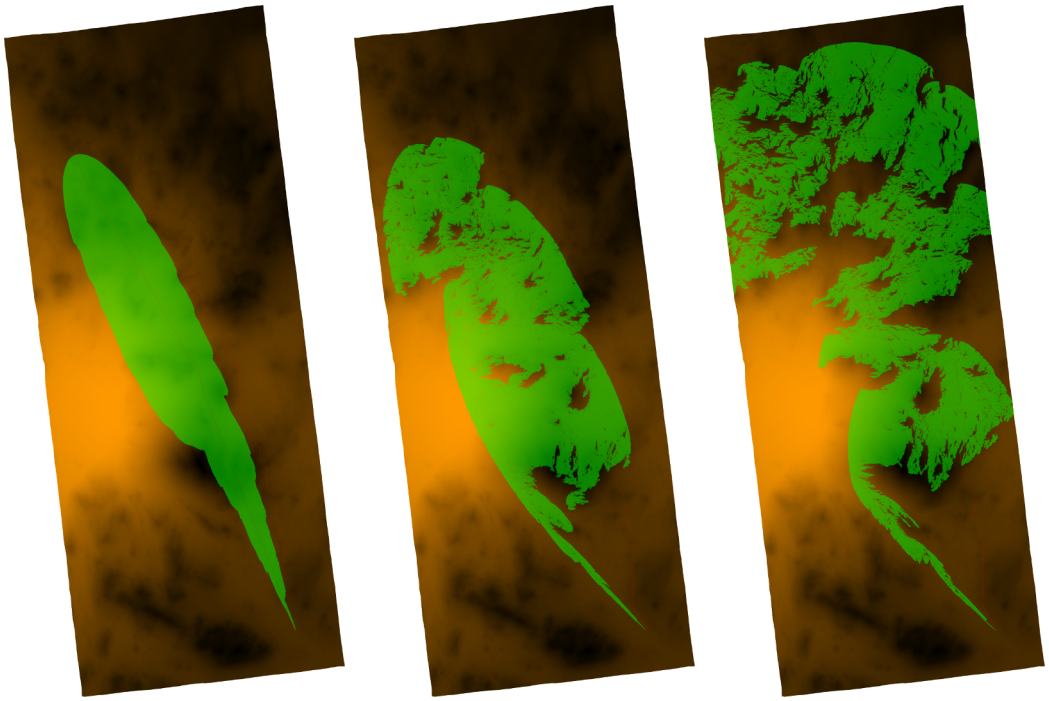


Figure 4.9: Area of the map expanded by 3Dana (with $\alpha_w = 0.0$) for the first experiment, considering different slopes. From left to right: no slope consideration, 20° and 10°.

we can observe that the runtime increases with higher α_w values. 3Dana requires some time to find paths as a consequence of both, the re-expansion process of the nodes and the computational cost related to the management of the DTM (note that the DTM used is quite wide).

Figure 4.9 provides a visualization of the expanded nodes for different configurations of 3Dana. In the left figure we can appreciate that the expanded nodes *follow* the path obtained by 3Dana $\alpha_w = 0$ with no slope in the fig. 4.8. Then, the center figure requires more expansions, as the maximum slope is restricted to 20°. Finally, the right figure requires to expand more areas from the top, but there are others areas that are not considered during the search (e.g., the crater), which correspond to zones with slopes higher than the limit imposed. Then, 3Dana selects a path that stays away from the crater, but also surrounds the hills. This path is longer, but safer and maybe more efficient as it does not climb hills.

The second map considered presents an uplift of a 30-kilometre diameter crater in Noachis Terra[4]. The total area covered is near 15 km^2. The dimension in nodes is 2960 x 2561. In this map we set the initial point to the coordinates (800, 600) and the goal in (1800, 2500). We have run the same experiments, i.e., $\alpha_w \in \{0.0, 0.5, 1.0\}$ and maximum slopes between 10° and 30°. However, in this map, there is no valid path that does not exceed the 10° slope limitation.

[4] `http://uahirise.org/dtm/dtm.php?ID=ESP_030808_1535`

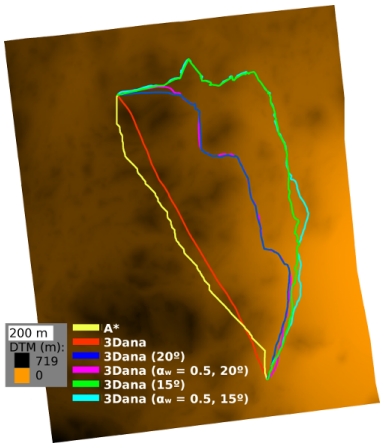


Figure 4.10: Paths obtained for the DTEED_030808_1535 using A* and different configurations of 3Dana.

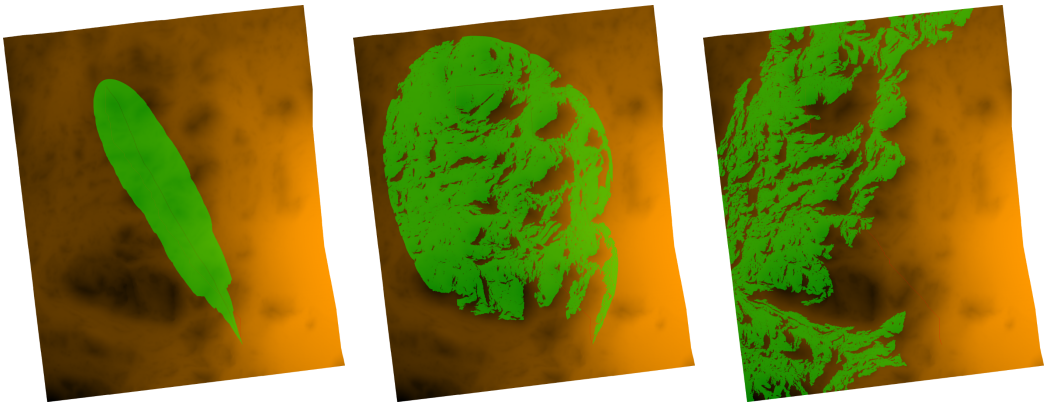


Figure 4.11: Area of the map expanded by 3Dana (with $\alpha_w = 0$) for the second experiment, considering different slopes. From left to right: no slope consideration, 20° and 10° (no path found in this case).

Table 4.2 presents the results of these executions. As for the first experiment, 3Dana outperforms A* in both, path length and total turns for all cases. As well, we can appreciate that the path length increases more than 1 km from the case with maximum slope of 20° to the case of 15°. If we analyse the heading changes, we can also appreciate that increasing the α_w value not always improves the heading changes. In fact, usually when more limited is the slope, higher α_w values trend to degrade the paths, while increasing the heading changes.

In fig. 4.11 we can appreciate the nodes expanded during the search for cases without considering maximum slope (left), maximum slope of 20° (center) and 10° (right). This last case is remarkable because neither A* nor 3Dana are able to found a path. For 3Dana, we can see that the algorithm expands several nodes, but there is no path that allows to safely traversing the desired goal. Then, fig. 4.11 (right) provides a vision of the reachable areas of the map when we restrict the maximum slope to 10°. This could provide an insight of the terrain that can be useful for human operators during the mission planning.

Table 4.2: Paths data for DTEED_030808_1535. In bold: best path length plus total turns for each maximum slope.

Alg.	Max. slope	α_w	Length (m)	Turn (°)	Time (s)	Expanded nodes
A*	-	-	4800	13995	49	407544
3Dana	-	-	**4493**	**478**	77	720744
A*		-	4986	20790	198	797942
	30°	0.0	4662	3145	235	1034084
3Dana		0.5	**4705**	**2221**	576	1062632
		1.0	4839	2388	904	1401925
A*		-	5440	27765	301	1577711
	25°	0.0	5077	3374	369	1705122
3Dana		0.5	5145	2703	701	1638401
		1.0	**5170**	**2658**	831	1566578
A*		-	5990	28080	343	2129666
	20°	0.0	5665	4442	433	2230932
3Dana		0.5	**5776**	**4241**	777	2150387
		1.0	5786	4428	983	2153923
A*		-	7387	41850	244	2712912
	15°	0.0	**7010**	**8569**	409	2789358
3Dana		0.5	7175	9066	749	2721862
		1.0	7450	9061	901	2774165
A*		-	No path			
3Dana	10°					

In the experiments carried out on the Mars maps, we have noticed a degradation on the path turns with high α_w values that are not reproduced with the synthetic maps (evaluating the DTM or in combination with a cost map). While the synthetic DTMs are mainly hills and valleys, real maps have different characteristics (e.g., hills, fissures). Higher α_w values (> 0.5) *force* the algorithm to follow the current heading. On the complex Mars surface, this can have an undesirable effect as may lead the path to an hazardous area in which many heading changes are required to arrive to a suitable terrain. In fact, it is remarkable that this behaviour is directly related to the slope limitation; without (or soft) slope restriction, α_w works as expected.

4.4 Summary

Path planning on flat surfaces is not enough when dealing with robotics such as rovers, which are designed to move on uneven terrains. Then, it is required to exploit more realistic terrain representations, and a path planning algorithm that can work with them. Then, in this chapter we first defined an accurate mathematical representation of a DTM. By means of lineal interpolation methods we can provide the elevation at any point, which allows using any-angle path planning algorithms. Exploiting such representation, we introduced the 3Dana path planning algorithm that takes advantage of previous algorithms such as S-Theta*. 3Dana is able to plan the paths having in mind the terrain elevation, allowing the user to define slope constraints. Then, the algorithm can discriminate paths that overcome the slope limitation imposed, providing safer and reachable paths. Moreover, it is also possible to use a traversability cost map in combination with the DTM to characterize other terrain properties as done with algorithms like Field D*. Finally, 3Dana allows defining the relevance of the heading changes, generating smoother paths in a similar way as done by S-Theta*.

Chapter 5

Interleaving path and task planning for deliberative layers

This chapter provides the description and functionality of the UP2TA planner. This system is intended to be a deliberative layer for autonomous controllers for mobile robotics. Specifically, its application is to those domains in which there are several goals allocated in different positions and the optimality of the solution depends on the total distance travelled. Thus, next section introduces the motivation of such system. Following, early PDDL models that integrates path and task planning to be solved by a PDDL planner are presented, while the models used by UP2TA are provided as well. Then, some concepts required to understand the new planner are defined. Section 5.5 presents the planner architecture. The chapter ends with an experimental section applying UP2TA to an exploration domain.

5.1 Integration of path and task planning

Most of the autonomous controllers presented in sec. 2.1 have been demonstrated in mobile robots, that move and perform tasks in a variety of environments. In order to do that, they have to avoid obstacles, find safe trajectories and plan the tasks. Generally, they move in known or partially known surfaces and the number of decisions is usually limited.

However, when dealing with increasing number of goals we require more capable systems to achieve the goals considering how to optimally order them. For instance, the mobility and science capabilities of new science missions such as the new rovers for planetary exploration (e.g., Mars Science Laboratory (MSL), ExoMars) increase from the previous missions, requiring more powerful tools to assist on-ground operators to plan the long term targets. In this regard, in previous missions plan the path followed by the rover was not a complex task due to the short distances they could travel per day. Nevertheless, a critical aspect in future missions is to try to optimally plan the path that a robot should follow while performing scientific tasks [125]. In addition, improvements for such domains can also be incorporated into commercial robotic applications, e.g., performing inventory tasks in a large warehouse or autonomous logistics domains.

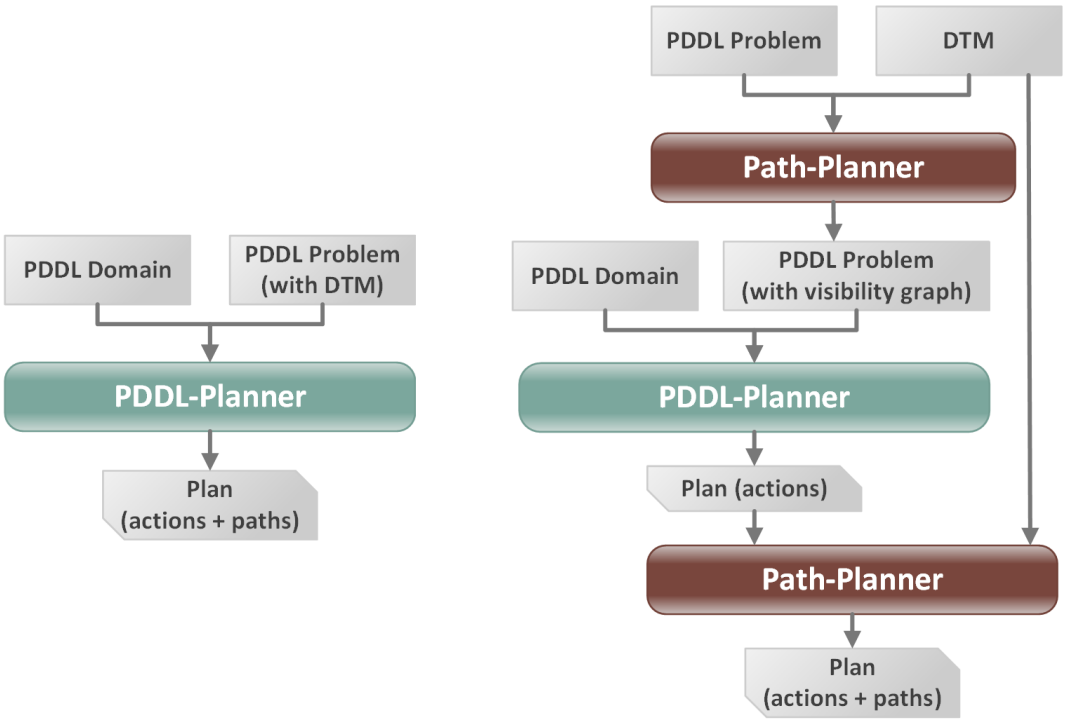
For this reason, we need a planner that integrates capabilities of path planning (introduced in the previous chapters) and task planning. The main idea is to take advantage of path planning heuristics and merge them with domain independent heuristics to generate better solutions in mobile robotic domains. Following this approach, we propose a planner, called Unified Path Planning and Task Planning Architecture (UP2TA) [129], which is able to plan paths considering the shortest path while performing the objective tasks in an efficient ordered way. A PDDL planner is responsible of ordering the tasks while a path planning algorithm searches for the route between tasks. In UP2TA, these planners are highly coupled, merging the heuristics of both planners to provide better solutions for mobile robotics domains.

5.2 PDDL models for interleaving task planning and path planning

In the literature we can find different approaches focused on integrating task planning and motion planning, which are more related to the manipulation of objects employing robotics arms (see sec. 2.4). However, our objective is to efficiently plan the paths that a mobile robot has to follow to achieve a set of targets. In this regard, while our objective is different, the techniques required to perform such integration are similar to the previous works. In this direction, the main issue is to integrate domain specific information in a task planner, interleaving task and path planning.

Our first approach to solve a mobile robotic domain was to integrate a DTM within the PDDL problem and provide a description of the movement actions in the domain. Then, we solved the problem using a PDDL planner (see fig. 5.1a). However, this solution did not provide good results. The problem of using a PDDL-based planner to integrate both path planning and task planning presents two drawbacks. First, the complexity of the domain and the problem. In our domain we used eight possible movement actions *north, south, east west, north-east, north-west, south-east*, and *south-west*. However, the critical part is in the problem, where we need to provide the DTM in a grid form. This means that we have to define all nodes and their connections. In other words, what nodes are adjacent with each other. Second, the size of the grids (at least 500 x 500 nodes). The planner must instantiate a huge number of actions. Particularly, the movement action is instantiated for every adjacent pair of nodes. Considering the possibility of movement in eight directions, we can instantiate eight possible move actions for each node. This results in an exponential explosion of the search space. The more locations we define in the problem, the more instantiated operators will be generated. So, even for medium size maps, the search is intractable and no state of the art planner could solve it (partially or fully grounded planners).

Solving small maps with this approach leads to routes with headings restricted to multiples of $\pi/4$ since we have defined movements in diagonal, horizontal and vertical, which may cause sub-optimal paths (as happens with A* in eight-connected grids). We could have defined more operators in order to smooth this restriction, but the search was already intractable. In this direction, we have first used FF [72], increasing the grid size. Results show that the planner can manage 50 x 50 grids



(a) Using a PDDL based planner (b) Interleaving both planners

Figure 5.1: Possible solutions to merge path planning and task planning.

with 6 and 9 tasks in 1 and 2 seconds respectively, but requires a high amount of memory (near 1 GB). Increasing the map size implies that FF is unable to manage the problem size, and thus, it cannot provide a solution. Since we wanted to test if the behaviour only occurred in FF, we did the same test with other fully grounded planners such as SGPlan [73], getting the same behaviour. We have also performed the same test for the partially grounded planner SatPlan2006 [157]. Results were even less encouraging since more time was needed to solve the basic problems that FF was able to solve.

A better solution is to decouple path planning and task planning using a three phases planning process (see fig. 5.1b). The idea is to include a path planner that generates a visibility graph considering all the waypoints given in a PDDL problem (that is, the initial and goal positions). This reduces the complexity of the PDDL domain because all the movement actions are reduced to a single one that traverses between waypoints. On the other hand, the PDDL problem includes only the relevant positions. By doing this, it is no longer required to include the DTM in the PDDL model, as the path planner considers it during the path search. This reduction in the number of actions and objects also decreases the workload of the task planner. Then, the PDDL planner provides a solution in which there are movements between waypoints, but without providing the real path between them. The last phase replaces the movements between waypoints with a valid path by executing again the path planner. Finally, we obtain a plan in which tasks and movements are properly interleaved.

This approach has two advantages: (i) it allows to easily interchange the PDDL planner and path planner to exploit different algorithms, and (ii) it eliminates zig-zag patterns and provides better routes when it uses an any-angle path planning algorithm. However, this schema presents a major drawback: as there is no shared information between the path planner and the task planner during the search phase, solutions generated can be highly inefficient. In particular, the selection of the task planner has a big impact in the solution optimality as the domain independent heuristic does not take into account the distance among tasks during the search.

In this sense, we have performed an experiment using SGPlan, OPTIC [15] and UP2TA in a classical exploration domain where each task performs a single action in a determined location. In particular, the test consists of 10 maps of 100 x 100 nodes (considering a distance between nodes of 1 meter) with 20% of blocked cells. For each map, two scenarios with 6 and 12 tasks randomly distributed are defined. For all planners, the path planning algorithm employed is S-Theta*, so the differences in the path length to accomplish all tasks are due to the task ordering rather than the path planning algorithm. The average results of this test is shown in table 5.1. Regardless of the reduced number of scenarios tested, the difference between the solutions provided is significant. SGPlan provides worse results than OPTIC, and UP2TA gives the best results. This is evidence that when we merge the heuristic of the path planner and the heuristic of the task planner we get better results than independently. A detailed discussion on this issue will be presented in sec. 5.6.

Table 5.1: Results for the execution of different maps (with a dimension of 100 x 100 m) with UP2TA and two different PDDL-based planners. All use S-Theta* as the path planner. In bold, best values.

# tasks	Planner	Runtime (ms)	Path length (m)
6	SGplan	2007	413.780
	OPTIC	2653	389.600
	UP2TA	**1661**	**164.363**
12	SGPplan	**8364**	1785.121
	OPTIC	69800	1386.400
	UP2TA	8917	**718.340**

5.3 Input files for UP2TA

The idea behind the UP2TA planner comes from our experience using a PDDL planner as the deliberative layer for the control of a mobile robot in exploration domains. We want a planner that gets a closer to optimal ordering of multiple tasks placed in a grid. In order to do that, we combine a modified FF planner and a path planning algorithm to work together in a coordinated way. The resultant system, called UP2TA, takes the benefits of (i) a PDDL planner for task planning using PDDL problems and domain independent heuristics, and (ii) a path planning algorithm for generating better routes using a DTM and domain specific heuristics.

To perform the integration, our first objective is to remove any information of the path planning operations from the task planner. That is, we do not want that the task planner provides the path, it just connects the different targets in an efficient order. For this reason, the PDDL domain and problem do not contain the DTM (it is only used by the path planner). Then, the files required by UP2TA are:

- *Domain and problem:* UP2TA accepts any PDDL version that is supported by the PDDL planner used in the integration. However, UP2TA requires in the domain description an action called `MoveTo` that allows the robot to move between waypoints (`wp`). Besides, the problem file does not need to provide a DTM, just the waypoints in which we want to perform a task or traverse through. These waypoints must have the form `Ca_b` to represent a position with coordinates x=`a` and y=`b`. Figure 5.2 shows a valid domain for UP2TA. As well, fig. 5.3 shows an example of a problem file that has three waypoints: `C1_1`, `C9_5`, and `C6_8`; the first one represents the initial position (which is also the desired final one), and the second and third ones represent where the goals are placed. An extended version of this domain is explained in sec. 6.2.

- *DTM:* this file contains the information of the terrain. The codification and content depend on the path planning algorithm used. If the algorithm works as the ones presented in chapter 3, a map with blocked and unblocked cells is required. However, more complex path planning algorithms require also a traversal cost map, which contains the costs related to move through each region of the map for algorithms like Field D*, or a more complex DTM for working in 3D environments, for instance, using 3Dana. There is a relationship between the DTM and the PDDL files: each location defined in the PDDL problem (`Ca_b`) corresponds to a position in the DTM with coordinates x=`a` and y=`b`.

Using these files, the UP2TA planner provides a sequence of actions with the (sub)optimal paths between the tasks, which achieves all the goals defined in the PDDL problem.

```
(define
 (:domain UP2TA-exploration)
 (:types wp subsyst ptu_aim - object)
 (:predicates
   (on ?s - subsyst) (off ?s - subsyst)
   (robot_at ?n - wp) (ptu_pos ?p - ptu_aim)
   (picture ?n - wp ?p - ptu_aim) ...
 )
 (:action MoveTo
  :parameters (?n1 ?n2 - wp)
  :preconditions (and (robot_at ?n1) (on gnc)
                      (ptu_pos P0_0))
  :effect (and (not (robot_at ?n1))
               (robot_at ?n2))
 )
 (:action TakePicture
  :parameters (?n - wp ?p - ptu_pos)
  :precondition (and (robot_at ?n) (off gnc)
                     (on cam) (ptu_pos ?p))
  :effect (picture ?n ?p)
 )
 (:action Drill
  :parameters (?n - wp)
  :precondition (and (robot_at ?n) (off gnc)
                     (on drill) (ptu_pos P0_40))
  :effect (sample ?n)
 )
)
```

Figure 5.2: Example PDDL domain file for the UP2TA planner.

5.4 Concepts and definitions for UP2TA

The following terms are defined in order to understand the integration explained in the next section.

- T_i is a task to be performed, defined as a goal in the PDDL problem. Each task takes place on a waypoint `Ca_b`.
- $h_{FF}(\mathrm{T}_i)$ is the FF heuristic function, which is computed using a relaxed plan graph. It returns the estimated number of actions required to reach a plan for a given goal.
- $dist(\mathrm{T}_i, \mathrm{T}_j)$ is a specific path planning heuristic. It is generally computed as the *Euclidean* distance (straight line distance between two waypoints).
- $greedy(\mathrm{T}_i, \mathrm{T}_j)$ is the cost of a fast computed path between two waypoints. This path is generated using the Alpha heuristic with an any-angle path planning

```
(define
(problem explor01)
(:domain UP2TA-exploration)
(:objects
     C1_1 C9_5 C6_8 - wp
     P0_0 P30_20 P0_40 - ptu_aim
     gnc ptu cam drill - subsyst
 )
 (:init
     (robot_at C1_1)
     (ptu_pos P0_0)
     (off gnc)
     (off ptu)
     (off cam)
     (off drill)
 )
 (:goal (and
  (picture C9_5 P30_20)
  (sample C6_8)
  (robot_at C1_1))
 )
)
```

Figure 5.3: Example PDDL problem file for the UP2TA planner.

algorithm (those presented in sec. 3.2), which returns a suboptimal path with lower runtime and memory usage.

- $path(T_i, T_j)$ is the path length between two waypoints. This path is computed using a classical path planning algorithm such as A* or Theta*.
- $H(T_i)$ is the heuristic function computed for a task. When a task requires a movement between waypoints, the heuristic is computed by adding the heuristic of the task planner (h_{FF}) and the heuristic from the path planner.
- $C(T_i)$ is the cost associated to perform T_i. This is value is computed by the task planner. Since we use FF, each action has unit cost (for non-movement actions).

As well, we have to define the two planners to use in our integration. For the path planner algorithm, as we discuss later, our integration makes easy to replace it with a different one. In any case, the algorithms presented in this dissertation will be used (i.e., S-Theta* and 3Dana).

The election of the task planner entails more discussion. There is a number of relevant PDDL heuristic planners that have shown good performance in recent International Planning Competition (IPC). Particularly, newer planners can deal with complex domains that include temporal constraints or preferences [57,64]. However, we have decided to use the well-known FF planner in our integration. It was the

winner of some competitions in the past. It is written in C, making it very compact and fast, suitable for real robots applications when the processor capacity is limited. Some newer planners such as Metric-FF and SGPlan rely on FF. Therefore, it seems to be possible that we can integrate our modifications without much effort in FF-based planners. Since we are more focused on evaluating the integration between a path planning heuristic and a domain independent heuristic, FF seems to be good enough for our purposes. It is worth underscoring that newer planners are usually more complex to be modified due to external modules, lexical analysers, among others.

5.5 The UP2TA deliberative

Figure 5.4 shows the UP2TA planner scheme. It works as a single system, but it has two different parts: the task planner and the path planner. The control of the system resides in the task planner, i.e., a PDDL planner. The process starts reading the PDDL data and instantiating the defined objects such as the robot initial position, the subsystems state, and the tasks to be performed. However, we can introduce other parameters and/or constraints into the PDDL domain such as the subsystems restrictions or hierarchical tasks. The DTM is given to the path planner.

Consider the domain and problem definitions shown in fig. 5.2 and 5.3. To achieve the goal `picture C9_5 P30_20` (denoted as T_1), the robot must move (action `MoveTo`) from the initial position `robot_at C1_1` to the waypoint `C9_5`. Next, the camera has to point at the angle `P30_20`, that is $pitch = 30°$ and $yaw = 20°$. Then, the robot can take the picture (action `TakePicture`). This action sequence achieves goal T_1. In general, the robot should perform a `MoveTo` and a the required action(s) (e.g., `TakePicture`) to reach a goal T_i. In this example, each task is performed at a particular position. Therefore, T_i denotes the task and the location. For instance,

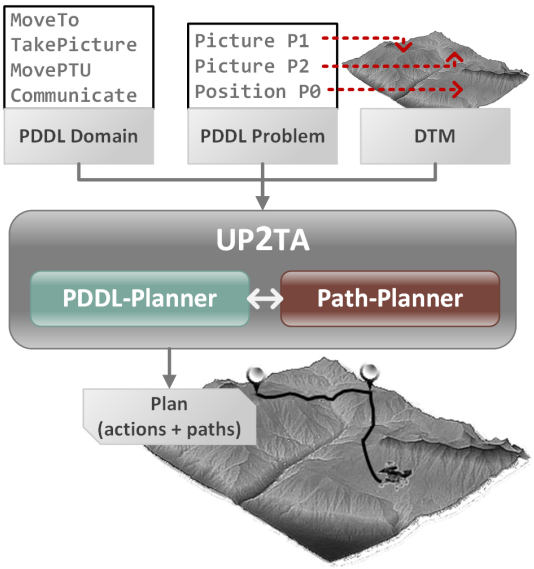


Figure 5.4: UP2TA general structure.

T_1 is located at `C9_5`, which corresponds to coordinates x=9 and y=5 in the DTM. Then, we need to plan for the second goal T_2 = `drill C6_8`, and finally the third goal T_3 = `robot_at C1_1`. Note that the robot ends at the same position where it starts.

As mentioned previously, the order in which we achieve the goals has a significant impact in the quality solution due to the total distance travelled. To optimally order the tasks, we have modified the task planner search algorithm to merge its heuristic with the path planner heuristic. For UP2TA, we use FF as the task planner and S-Theta* as the path planner algorithm. For FF, we have transformed the Best First Search algorithm into an A* algorithm. The only difference between Best First Search and A* is that the latter includes the heuristic computation in the node evaluation. In addition, we have disabled the Hill Climbing search to avoid local minimums. For the path planner, we do not need to perform any modification.

The search begins when UP2TA takes each task T_i that can be performed from the initial position (*start*) defined in the problem (denoted as `robot_at` in the example). It calculates the task planner heuristic, $h_{FF}(T_i)$, and the one provided by the path planning algorithm, $dist(start, T_i)$. The $h_{FF}(T_i)$ heuristic is a domain independent heuristic that computes an estimation of the actions required to accomplish a goal based on a relaxed planning graph. The $dist(start, T_i)$ heuristic is computed by the path planning algorithm, but it is only required when the task planner needs to instantiate an action `MoveTo`. In general, this heuristic is computed as the *Euclidean* distance, but others can be used. The way in which we have merged both heuristics in UP2TA is by adding them as in eq. 5.1, where $T_j = start$ at the beginning of the search. By doing this UP2TA can decide which goals to reach first based on the distance and not only in the number of actions required to accomplish it.

$$H(T_i) = h_{FF}(T_i) + dist(T_j, T_i) \tag{5.1}$$

The reason to combine the heuristics of the task planner and the path planner is because of the way FF performs the search. If h_{FF} is equal to 0, it means that the action under evaluation has reached the goal, and FF returns the plan. However, if the last action is a movement action, we need to consider the distance before returning the plan. That is the reason why we include the path planning heuristic in $H(T_i)$.

Once the heuristic value is calculated, we need to compute the $G(T_i)$ value. That is, the cost associated to reach task T_i from the previous task. If a movement action is required, we need to compute the distance between the two tasks. This step plays a fundamental role in the search process because obtaining a close to optimal ordering of the task depends on the environment. It is unusual to transverse routes between two points in straight-line because of the presence of obstacles or terrain features that make the path more costly. This is specially problematic when there are tasks that are close to each other, or with high number of obstacles in the map. In these cases, the real path length may be too far from the heuristic value and the selection of the actions to perform first depends on the presence of obstacles. Therefore, it seems to be a good idea to compute the real path when dealing with a pair of tasks. However, when we deal with a higher number of tasks, computing the

distance between them requires a higher computational effort. The solution adopted here is to use a greedy path planning algorithm (like the ones described in sec. 3.2) to calculate estimations of the path length between a pair of tasks. As we seen before, the heuristic modification in these path planning algorithms implies that fewer nodes will be expanded and, therefore, less memory and time are spent during the search of paths for partial plans.

Although this approximation does not generate the real path, the associated path length allows us to take into consideration the presence of obstacles without spending several time in obtaining paths between tasks that probably will not be used in the final solution. We define $greedy(\mathrm{T}_i, \mathrm{T}_j)$ as the path cost between tasks T_i and T_j using a greedy path planning algorithm. In addition, we need to compute the cost of the actions involved in performing a task T_i. This value is represented as $\mathrm{C}(\mathrm{T}_i)$ and it is given by the task planner. $\mathrm{G}(\mathrm{T}_i)$ is a cumulative cost that depends on the cost of reaching the previous task, the distance (estimated) between the two tasks, and the cost of performing a task i as in eq. 5.2.

$$\mathrm{G}(\mathrm{T}_i) = greedy(\mathrm{T}_i, \mathrm{T}_j) + \mathrm{C}(\mathrm{T}_i) + \mathrm{G}(\mathrm{T}_j) \tag{5.2}$$

The task planner keeps a list of partial plans (where a partial plan is a plan that achieves an arbitrary task T_i) ordered by the $\mathrm{F}(\mathrm{T}_i)$ values. So, each time that the task planner extracts a partial plan, it adds it to the list. Then, the search process continues with the extraction of the best promising partial plan. UP2TA generates all possible partial plans starting from the new location where the robot is located. The task planner continues extracting partial plans until there are no more goals (all objectives are accomplished), or there is no solution (unreachable task/s). When all task are performed, UP2TA does not give the solution directly. The paths computed between each pair of consecutive tasks are processed using suboptimal algorithm (the greedy ones). Therefore, the system takes each pair of tasks and requests the path planning algorithm for the real path (using the S-Theta* algorithm). Figure 5.5 shows the scheme integration between the PDDL planner and the path planner. The task planner is in charge of ordering the tasks and obtaining a plan, while the path planning module provides an estimation of the cost between partial plans and the routes for the final solution.

Figure 5.6 shows an example for the search process of the UP2TA planner for a problem with three goals. There are three possible tasks T_1, T_2, and T_3 at the *start* position. The first step is to calculate the heuristic and cost values $\mathrm{H}(\mathrm{T}_i)$ and $\mathrm{G}(\mathrm{T}_i)$ for each task. The heuristic $\mathrm{H}(\mathrm{T}_i)$ is calculated as in eq. 5.2 and it corresponds to the *Euclidean* distance between the task and the current position plus the heuristic given by the task planner, i.e., the estimated number of actions. The cost $\mathrm{G}(\mathrm{T}_i)$ of achieving the position of each task is calculated as in eq. 5.2. Once $\mathrm{H}(\mathrm{T}_i)$ and $\mathrm{G}(\mathrm{T}_i)$ are calculated, there are three partial plans to be evaluated, one for each task T_i.

UP2TA takes the one with lower $\mathrm{F}(\mathrm{T}_i)$ value. In our example, it is the partial plan that performs T_1. Next, it evaluates the other two possible tasks T_2 and T_3. In the same way, the system calculates the heuristic and cost to reach T_2 and T_3 from T_1, which in our case T_2 has the lower value. At this point, there is only one remaining task T_3, so the search finishes by adding that task. Once all tasks have

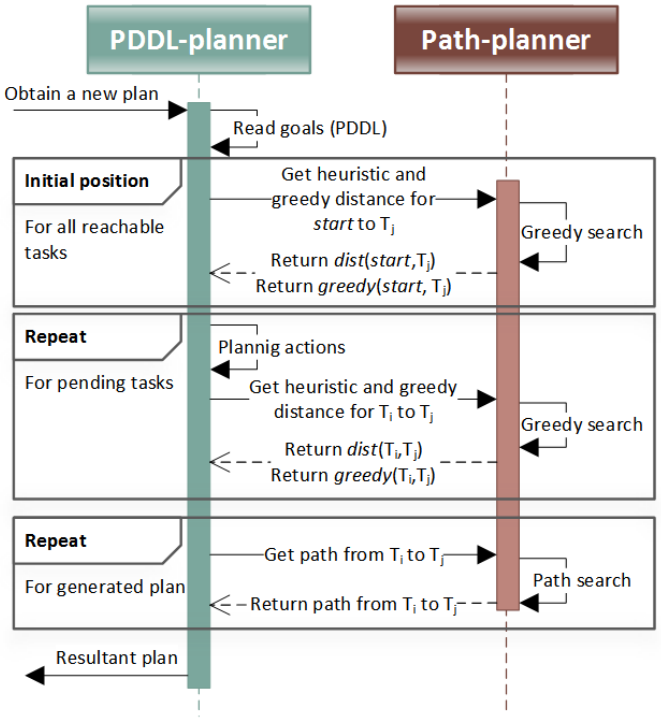


Figure 5.5: UP2TA algorithm integration.

been selected, the ordered tasks sequence for the solution is $T_1 \rightarrow T_2 \rightarrow T_3$. Figure 5.6 shows the most promising partial plans connected through a bold line (others have dotted lines). Then, UP2TA will calculate the real path between each pair of consecutive tasks that have been previously ordered. Finally, the system provides an ordered list of tasks and the paths between them.

5.6 UP2TA experimental scenario description

In this section, we describe the scenario that we use in the experimental evaluation. It consists of a mobile robot that must achieve a set of exploration tasks in different locations. As explained in sec. 5.3, we must provide three files to UP2TA: the PDDL domain and problem, and the terrain grid (DTM). The PDDL domain was partially presented in fig. 5.2. It describes the robot operations to manage the different subsystems (power on/off), move between points (`MoveTo`), and perform tasks (e.g., `TakePicture`, `Drill`, etc.) to accomplish mission goals. The robot is modelled to have different subsystems. In particular, it has a scientific/navigation camera mounted on top of a Pan-Tilt Unit (PTU), a navigation subsystem, and drilling equipment. Each subsystem can be *on* or *off* (must be *on* before operation). In addition, there are some constraints that must be considered when operating the different subsystems. Some of these constraints are presented in the `MoveTo`, `TakePicture`, and `Drill` operators. Concretely:

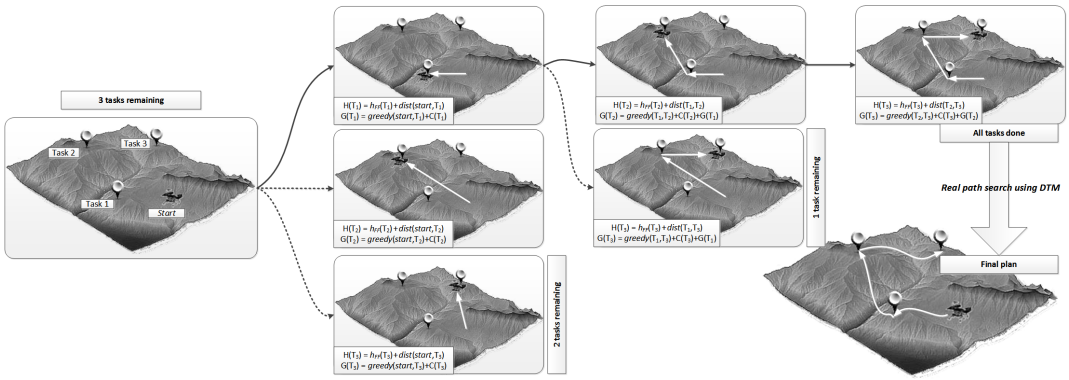


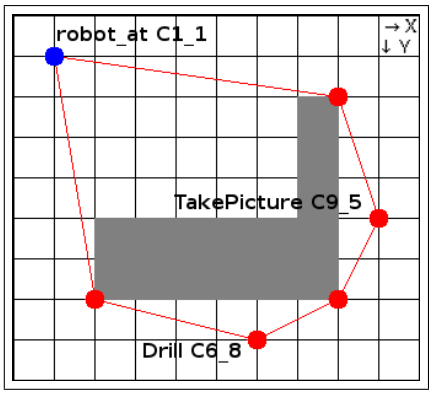
Figure 5.6: Representation of the UP2TA search process for a problem with three tasks.

- To move between waypoints, the rover must power on the navigation subsystem (`gnc`) and point the PTU at the front position (represented as `P0_0`) since the navigation camera is mounted on it. The other subsystems must be *off* to avoid mechanical problems and energy overconsumption.
- To take a picture, the rover must be stopped in the desired location and with a specific PTU position, which are defined in the goal. The camera must be active and the locomotion subsystem must be *off*.
- To get a sample from a desired location, the rover must use the drilling equipment to perforate the subsurface. To do this, the rover must be stopped in the desired position and the PTU must point at the surface (this allows the rover to take pictures during the drilling process).

Initially, all rover's subsystems are off with the PTU pointing to the front. The DTM, or terrain information, is in a different file. Particularly, the one that we used within our path planning algorithms provides a grid with blocked and unblocked cells (see fig. 5.7b). Given these data, UP2TA generates the solution shown in fig. 5.7a.

```
MoveTo C2_7
MoveTo C6_8
MovePTU P0_40
Drill C6_8
MovePTU P0_0
MoveTo C8_7
MoveTo C9_5
MovePTU P30_20
TakePicture C9_5 P30_20
MovePTU P0_0
MoveTo C8_2
MoveTo C1_1
```

(a) Plan given by UP2TA



(b) Plan graphical representation

Figure 5.7: Solution for the problem of fig. 5.3.

It is remarkable that the path planner expands the move actions between tasks to provide free collision paths, while the task planner orders the goals to minimize the distance travelled for the final plan.

For this domain, each task can be performed independently. That is, there is no relationship between tasks except for the robot position. For this reason, we have also defined a more challenging domain where tasks interfere with each other. This new domain is an extension of the previous one that includes extra requirements for the drilling process: it can only take one sample at a time, and it is necessary to deliver each sample before getting a new one. The delivering process implies going to a particular location and execute a deliver action. Once the robot delivers the sample, it can get another one. For this domain, the planner has to take into consideration the limited capacity of the drill to order the goals. The drill samples must be gotten one by one, but the pictures can be taken independently. This means that the planner must try to optimize the pictures acquisition interleaving them with the drilling/delivering operations, when possible.

5.7 UP2TA experimental results

In the following, we present an experimental evaluation on the mobile robot exploration domains described in the previous section. The test consists of running the UP2TA planner with different path planning algorithms (A* and Theta* (see sec. 2.2), and S-Theta* (see sec. 3.4)) over 100 uniforms flat surfaces of 500 x 500 nodes with 40% of blocked cells (the map generation algorithm can be found in appendix A) and randomly placed tasks. In addition, we try different heuristic configurations for both, task planning and path planning. For each test, we measure three parameters: (i) the runtime for the planning process, (ii) the total distance travelled to achieve all goals, and (iii) the total turns in degrees for the final path. The experiments were performed with a time limit of 2 minutes for each execution. The execution was conducted on a 2.5 GHz Intel Core i7 with 8 GB of RAM under Ubuntu 14.04.

We have performed two sets of experiments. In the first one, we use the domain shown in fig. 5.2 and different path planning heuristics combination. In the second one, we use the extended version of the exploration domain (with samples delivering and unitary capacity for the drill) in which we have substituted the drill operation with a collect and deliver task to analyse the importance of the task planner heuristic. In general, the objective is to evaluate the best configuration of UP2TA in terms of runtime and distance travelled (modifying the path planning algorithm), and to compare it with the results obtained with approaches that do not share information between the path planner and the task planner during the search.

For the first test we want to analyse how the path planner heuristic affects the tasks ordering by the task planner. We are also interested in analysing the paths generated to select the best performance configuration as function of the distance travelled. The way that we do this is by testing three different cost functions, $F(T_i)$=[$H(T_i)$] +[$G(T_i)$], for the integration of task planning and path planning:

1. We only use the FF heuristic and the cost is computed using the greedy path planning algorithm as shown in eq. 3.6. This technique does not share information between the task planner and the path planner. That is, it provides solutions in a similar way that the solution proposed in sec. 5.2 in which planners are interleaved without sharing information (see fig. 5.1b).

$$\begin{aligned} \mathrm{F}(T_i) &= [h_{FF}(\mathrm{T}_i)] + \\ &\quad [\mathrm{G}(\mathrm{T}_j) + greedy(\mathrm{T}_i, \mathrm{T}_j) + \mathrm{C}(\mathrm{T}_i)] \end{aligned} \tag{5.3}$$

2. We use the FF heuristic and the *Euclidean* distance as the task planner heuristic. The cost is obtained using a path planning algorithm as show in eq. 5.4.

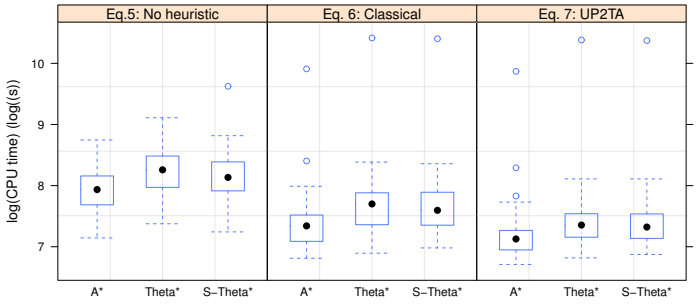
$$\begin{aligned} \mathrm{F}(T_i) &= [h_{FF}(\mathrm{T}_i) + dist(\mathrm{T}_i, \mathrm{T}_j)] + \\ &\quad [\mathrm{G}(\mathrm{T}_j) + path(\mathrm{T}_i, \mathrm{T}_j) + \mathrm{C}(\mathrm{T}_i)] \end{aligned} \tag{5.4}$$

3. We use the FF heuristic and the *Euclidean* distance as the task planner heuristic. The cost is given by a greedy path planning algorithm as in eq. 5.5.

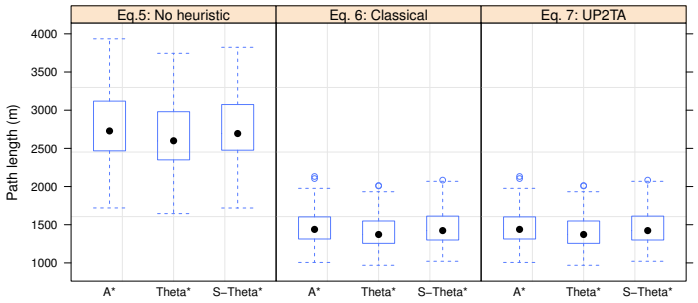
$$\begin{aligned} F(T_i) &= [h_{FF}(\mathrm{T}_i) + dist(\mathrm{T}_i, \mathrm{T}_j)] + \\ &\quad [G(\mathrm{T}_j) + greedy(\mathrm{T}_i, \mathrm{T}_j) + C(\mathrm{T}_i)] \end{aligned} \tag{5.5}$$

Figure 5.8 shows the results for the above mentioned approaches. For each case, 9 tasks will be executed. As we can see, when we only exploit the FF heuristic we obtain the worst results for all parameters. That is especially relevant in the case of the distance travelled. The reason why we obtain these results is that we do not provide any information about the distance in the heuristic, and then, the task planner will only take into consideration the number of pending tasks. This may lead to connect tasks that are located far from each other. When we include the path planning heuristic (eq. 5.4 and 5.5), we get the same results in path length and heading changes. The difference resides in the runtime: when we compute the path length using a non-modified path planning algorithm, the planner requires near 20-30% more time to generate a solution. In eq. 5.5 we use the greedy (or the modified) path planning algorithm that is faster since it provides an estimation of the path cost. Then, it requires less execution time. Therefore, the best configuration is the one that uses the evaluation function presented in eq. 5.5, being the one adopted for the UP2TA deployment. In the case of the path planning algorithm, A* provides the longer paths and the highest heading changes, while Theta* provides the best paths. However, S-Theta* obtains the best results in heading changes, and, also, the best paths when combining path length and heading changes. Thus, in the following we adopt S-Theta* as the path planner for UP2TA.

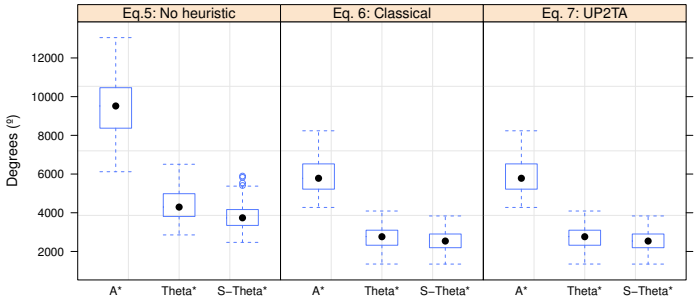
For the second test we have used the extended PDDL domain in which we have substituted 1/3 of the problem's goals by a collect and deliver task that requires 4 actions: (i) move to a location, (ii) collect a sample, (iii) move to the delivering location, and (iv) deliver the sample. In our test, the system has unitary capacity, which means that the current sample should be delivered before getting a new one.



a) Runtime in seconds



b) Path length in meters



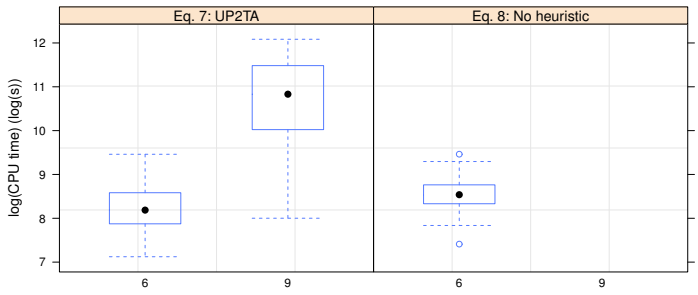
c) Heading changes in degrees

Figure 5.8: How the path planner affects the task ordering. Solutions of 100 problems with maps of 500 x 500 nodes with 40% of obstacles and 9 randomly placed pictures. Results are clustered by the path planning algorithm (A*, Theta* and S-Theta*) and the F function used (from left to right): no path planning heuristic (eq. 5.3); cost function using classical path planning algorithms (eq. 5.4); cost function using greedy path planning algorithms (eq. 5.5).

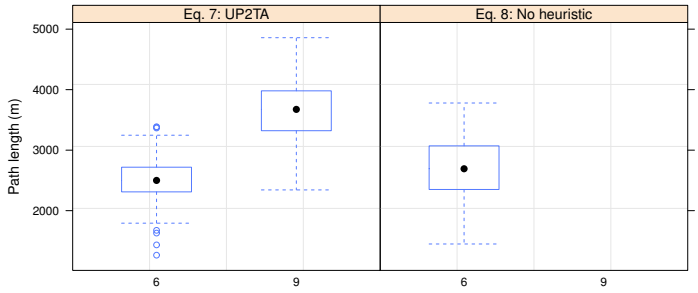
This implies that the planner should take such constraint into consideration to order the goals in the best way possible. This is done by the task planner. There two problems for this domain, which consist of delivering two and three samples and taking 4 and 6 pictures, being 6 and 9 the total number of tasks respectively. The rest of parameters are the same as the previous experiment (100 maps of 500 x 500 nodes with 40% of obstacles). To perform this test we use the UP2TA planner with eq. 5.5 to compute $F(T_i)$. In order to analyse the impact of the task planner heuristic, we have removed the FF heuristic and used the $F(T_i)$ shown in eq. 5.6. In any case, we use S-Theta* as the path planner.

$$\begin{aligned} \mathrm{F}(T_i) &= [dist(\mathrm{T}_i, \mathrm{T}_j)] + \\ &\quad [\mathrm{G}(\mathrm{T}_j) + greedy(\mathrm{T}_i, \mathrm{T}_j) + \mathrm{C}(\mathrm{T}_i)] \end{aligned} \tag{5.6}$$

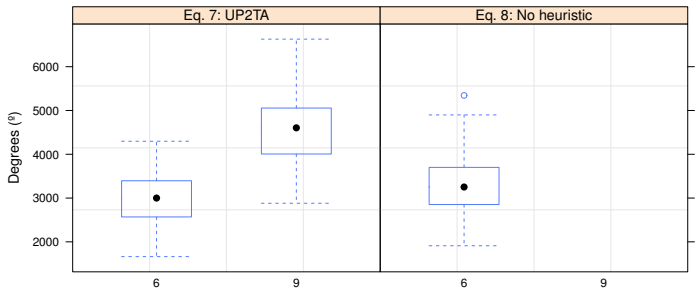
Figure 5.9 shows the results for the second test. We can see that when we remove the h_{FF} heuristic, the system is not able to find any solution for the 9 tasks case within the given time limit. Moreover, all parameters are worse than using UP2TA with $\mathrm{F}(\mathrm{T}_i)$ computed as in eq. 5.5. Conversely, UP2TA solves the 6 tasks and 9



a) Runtime in seconds



b) Path length in meters



c) Heading changes in degrees

Figure 5.9: How the task planner heuristic affects the solutions. Results for 100 sample and deliver problems with maps of 500 x 500 nodes with 40% of obstacles. Parameters clustered by the number of tasks (6 and 9) and the F function used (from left to right): cost function using greedy path planning algorithm (eq. 5.5) and; no task planning heuristic (eq. 5.6). In all cases employing S-Theta* as path planning algorithm.

tasks problems within 6 and 60 seconds respectively. The problem of removing the h_{FF} heuristic is similar to removing the path planning heuristic in the previous experiment. In this extended domain, the planner without the h_{FF} heuristic only tries to follow the shortest path between tasks. However, the problem is when a task has dependencies with other tasks. In that case, the system behaves like a breadth search algorithm, where the search space grows uncontrolled as a function of the number of tasks. The planner is not able to properly manage the tasks ordering to accomplish first the ones required to complete others actions. This generates undesirable behaviours such as movements to a location in which we want to take a sample when the previous sample is not delivered, delaying the solution.

5.8 Summary

In this chapter we presented an integration schema to interleave task planning and path planning, with the objective of generating optimal plans to achieve several objectives in different locations. Then, it is required to evaluate the paths between pairs of goals and properly sequence them, so the solution reduces the total distance travelled. Following that direction, we implemented the UP2TA planner, that integrates the path planning algorithms presented in ch. 3 and 4 with a state of the art PDDL planner. During the search phase, both planners share information, so it is possible to determine the tasks ordering by means of the distance among them. In this regard, we merge the domain independent heuristic of the task planner with the path planner heuristic. Then, UP2TA is able to provide plans that are shorter that others approaches in which planners are loosely integrated. Besides, using a PDDL planner provides the advantages of a modelling language, being possible to use in different domains. Moreover, the path planning algorithm is easily exchangeable, so we can select the best one that suits our application.

Chapter 6

A model-based autonomous controller

In this chapter we present an insight of the MoBAr autonomous controller, providing details about the technology and models employed for each layer. First, we introduce the controller from a global perspective. Then, the following three sections present the layers of the controller, i.e., deliberative, executive and functional. Finally, we show the results of the experimental evaluation of MoBAr when dealing with an exploration domain with both a simulated robotic platform, the ESA ExoMars rover, and a commercial robot, the TurtleBot 2 platform.

6.1 The MoBAr autonomous controller

The initial objective of MoBAr was to autonomously control the PTinto hexapod robot [117,130] for exploration domains. In this regard, we want that starting from a representation of the environment, a model of the robot subsystems and a set of goals, the robot could decide when and how to safely achieve the higher number of goals. This implies that the control architecture must take into consideration environmental restrictions, subsystems status, energy consumption and time, and the ability to plan and act under uncertainty and within dynamic environments. Then, we also feel the need to reuse it for further developments, since the architectures presented in sec. 2.1 rarely allowed that, i.e., the code is not publicly accessible or there is no enough technical documentation. In this regard, MoBAr provides an autonomous controller that enables testing the assets produced in this dissertation, i.e., the path planning algorithms (ch. 3 and 4) and the up2ta planner (ch. 5).

MoBAr corresponds to a hybrid 3T architecture, in which the top layer is in charge of the deliberative process and long-term memory. The middle level or execution system also has a short-term memory, as well as a series of rules that trigger the reactive behaviours implemented, in order to respond in a short time to eventual external or internal situations. Finally, the low level or functional level, is responsible for providing the hardware-related implementation details, and relaying the information collected by the sensors to the upper layers.

Each layer is based on a abstracted model that defines the properties, capabilities and constraints of the robot. The most detailed models are located in the functional layer; they contain the robot internal state plus the abilities that it has. The upper

layers have less detailed models, and thus, less coupled with the underlying hardware. In this way, the executor is in charge of taking high level actions coming from the deliberator and decomposing them into lower level commands supported by the functional layer. The executor has also the capability of managing the short term-memory reading data from the functional layer, and generating the symbolic knowledge used by the deliberator. We aim the executor to have little dependence on the hardware, for this reason, the high level model only knows the hardware in terms of available subsystems and high level abstraction of constraints. In fact, the deliberator works over a long-term memory using a symbolic formalism which includes actions that could be applied when its preconditions match with the current context, and their expected effects when they are applied. Also, the deliberator deals with the list of goals to achieve and the current state of the environment.

To minimize the time employed in coding the autonomous controller we have taken advantage of different general purpose technologies to implement MoBAr. This allows us to focus on the high level models rather than in implementation issues. High level models can describe the system constraints and capabilities in an upper level abstraction than the code, while also can provide supporting tools for simplifying the development and deployment. In this way, models provide a better way to define the platform behaviours and environmental constrains, which are usually easier to implement, understand and maintain.

For the deliberative layer we have used PDDL [106] and a PDDL planner (or the UP2TA planner). We have chosen the PLEXIL Executive (PE) and its language, Plan Execution Interchange Language (PLEXIL) [169] to model and control the executor. Finally the G$^{\text{en}}$oM framework [53] and Robot Operating System (ROS) [150] for the definition and implementation of the functional layer in different robotic platforms. The first two layers, deliberator and executor, use models (which are loaded at runtime) and thus, are easily interchangeable in order to adapt them to multiple robots. Instead, the functional layer is strongly coupled with the hardware but the philosophy of the modules generation framework (G$^{\text{en}}$oM) or nodes (ROS) allows building and testing functional modules faster than do it from scratch.

Figure 6.1 shows the MoBAr architecture and the connections between adjacent layers. These connections have great importance: each layer has a different vision of the world in terms of how the world is and what the robot can do. So, a direct channel of communication is not suitable, and the interconnection is done via interfaces, providing the possibility of sending requests and retrieve data from the adjacent layer. An interface has the capability to understand the request from its upper layer and to respond with data that fits the model of the destination layer.

The execution flow of the autonomous controller starts when the planner obtains a valid plan that satisfies the higher number of science targets that are set by the user. This plan is read by the executor, which takes each action and decomposes it in a group of lower level commands. For instance, a high level action can be *move to waypoint N*, so the executor must understand this task. Then, the executor decomposes the action into a sequence of commands executable by the functional layer that allows the robot to power on the locomotion subsystem and to move the desired distance. Finally, the functional layer executes the commands sent by the executor and relays the execution result and other data (such as the internal

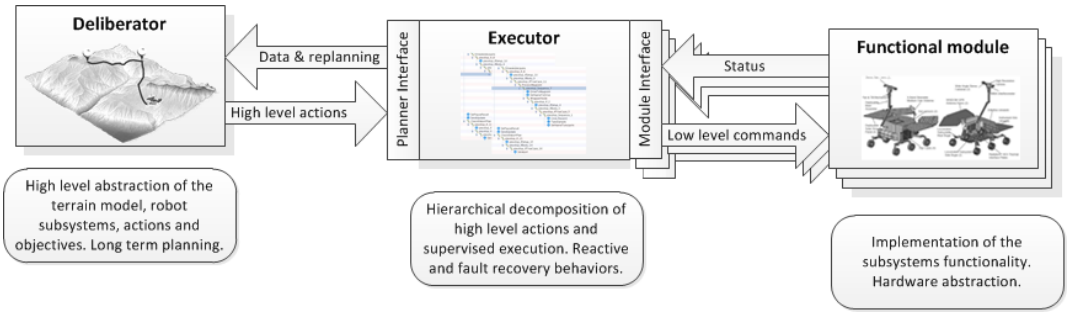


Figure 6.1: Conceptual vision of MoBAr and its connections between adjacent layers.

state of the subsystems or the science data) to the executor. The executor is in charge of analysing the results and acting accordingly. When everything is correct, the execution continues normally, but if there is a failure, the executor must take into consideration the failure type in order to entering into a safety state, or, more typically, detect a contingency in the plan execution (e.g., an unknown obstacle). In that case, the executor needs to update the information of the problem (as a previous step to call replanning at the deliberative level) or try to solve the error using a failure tolerant behaviour encoded for such purpose.

In MoBAr, the term *model* refers to high level models employed by the deliberative and executive to control the architecture execution flow. These models are written in the specific languages used by the respective systems. For the high level model, a PDDL domain and problem must be provided. These models are easy to deploy and understand, but the language version and planners employed could restrict the potential usage. For instance, using PDDL 1.0 [107] does not allow us to deploy models that take into consideration time for actions, while PDDL 2.1 [57] does. In the case of the executive, PLEXIL is semantically simple, but very expressive and provides a good way to define hierarchical tasks decomposition and basic resource models. The inherent problem to create PLEXIL models is that they must be hand coded and they require an important effort in checking that the coordination of parallel tasks is properly performed. Also, the PLEXIL model is dependent of the service provided by the interfaces, then, filling the gap between the executive and the adjacent layers implies a trade-off among the implementation of the interfaces and the coding of the PLEXIL plan. That is, as more services are provided by the interface, less effort shall be done to create the PLEXIL plan and vice-versa.

Instead, the benefits of using models could be summarized in:

- *The possibility of an incremental development.* This implies that we can make a basic model of the environment and the robot, and deploy it in a small period of time in order to test and verify the basic functionality. Then, we can improve the environment with a more realistic scenario and include models for secondary subsystems, fault tolerant behaviours, etc.

- *Same models for different robots.* The same high level models can be used for different rovers. For example, the locomotion or cameras subsystems operation is essentially the same in the deliberative layer since they are coded like *move* or *take picture* actions, and the executor models are in charge of a more detailed

description of the actions. Furthermore, the specific implementation of the subsystem operation resides only in the functional layer.

- *Improvements on the fly.* Models of the architecture could be modified during its execution adding or deleting functionality. For example, in the high level model we can include subsystems that are not available for the robot at the start of the mission. This potentially allows us to use this architecture with modular robots such as the S-Bots [42] increasing its abilities.

In the following sections each layer is described in detail.

6.2 The deliberative layer

For our 3T architecture we use a general purpose P&S system based on a standard language. Using a general purpose planner with a standard language enables the possibility of replacing the deliberator with a more powerful one with low cost, while also the models are reusable. In this direction, we have chosen PDDL, so we can exploit several planners that use such language. The same domain/problem files can be used without modification, and, if the planner support new PDDL features (e.g., time and metrics in PDDL 2.1 or constraints and preferences in PDDL 3 [64]) or extensions (like PIPSS [145], which implements resources as a PDDL extension), we will be able to extend the possibilities of the deliberator system in a short period of time.

Let's consider a planetary exploration scenario as an example to provide a description of MoBAr. In this regard, the deliberator goal is to obtain a close to optimal ordering of multiple scientific tasks placed in a grid, and the paths to follow between them. We can exploit a complex PDDL model that implements the scientific tasks to achieve, the rover's subsystems, and the model of the terrain as presented in sec. 5.2. However, as it has more cons than pros (complexity of modelling the DTM in the problem, zig-zag movements or planning requirements), we have used the interleaving schema presented in the previous chapter. Then, we can detach the DTM information from the PDDL model and exploit the UP2TA planner. However, UP2TA does not allows exploiting temporal actions as it uses PDDL version 1. For this reason, to also enable planning with time and resources, we have used the interleaving schema of task planning and path planning presented in sec. 5.2 (see fig. 5.1b) using planners that accept PDDL version 2.1 or higher. In this regard, the models used by both deliberatives are similar, removing only the temporary definition of the actions and the resources usage to adjust the model to the UP2TA planner.

Figure 6.2 shows part of the PDDL domain for the deliberator, that is, the set of actions that the robot can perform. These actions can only be executed by a robot if its model has the required subsystem involved in the action. The domain presented is used for the tested platforms, being only slightly modified in the case of the TurtleBot, which includes a battery recharge action (not shown here). In fig. 6.3, the problem definition for the ExoMars is presented. The TurtleBot is very similar and not shown. In this sense, both robots have cameras mounted on a PTU, but only ExoMars has a drill and communication equipment (some elements have been

removed for simplicity). However, the TurtleBot has the possibility of recharging its battery in locations in which a dock station is placed. Goals are defined as scientific tasks to be achieved in a specific location on the map. This model is evolved from the UP2TA template model presented in sec. 5.3 (see fig. 5.2 and 5.3), improved with the temporal behaviour and resource usage.

```
(:durative-action MoveTo
 :parameters (?r - rover ?p1 ?p2 - wp ?n - navmode)
 :duration (= ?duration (/ (distance_to_move ?p1 ?p2)(speed ?r ?n)))
 :condition (and (over all (has_locomotion ?r))
                 (over all (navigation_mode ?r ?n))
                 (over all (PTU_pos NavCam p0_0))
                 (at start (position ?r ?p1))
                 (at start (>= (energy ?r)(*(power_per_m ?r ?n)
                 (distance_to_move ?p1 ?p2)))))
 :effect (and (at start (not (position ?r ?p1)))
              (at end (position ?r ?p2))
              (at end (decrease (energy ?r)(*(power_per_m ?r ?n)
              (distance_to_move ?p1 ?p2)))))
)
(:durative-action MovePTU
 :parameters (?r - rover ?c - cam ?p1 ?p2 - ptupos)
 :duration (= ?duration (time_move_PTU ?p1 ?p2))
 :condition (and (at start (PTU_pos ?c ?p1))
                 (at start (>= (energy ?r)(*(PTU_energy)
                 (time_move_PTU ?p1 ?p2)))))
 :effect (and (at start (not (PTU_pos ?c ?p1)))
              (at end (PTU_pos ?c ?p2))
              (at end (decrease (energy ?r)(*(PTU_energy)
              (time_move_PTU ?p1 ?p2)))))
)
(:durative-action TakePicture
 :parameters (?r - rover ?p - wp ?c - cam ?a - ptupos ?m - mode)
 :duration (= ?duration (time_to_picture ?c ?m))
 :condition (and (over all (camera_mode ?r ?c ?m))
                 (over all (position ?r ?p))
                 (over all (PTU_pos ?c ?a))
                 (at start (>= (energy ?r) (camera_energy ?c ?m))))
 :effect (and (at end (picture ?p ?m ?a))
              (at end (decrease (energy ?r) (camera_energy ?c ?m))))
)
```

Figure 6.2: PDDL actions definition for the rover example.

In fig. 6.2 we present three high level actions that the robot could perform. First, the movement action (`MoveTo`) is based on the rover navigation mode (`fast` or `slow` as specified in the problem) and the travelled distance. The time to travel is dependent on the speed of the rover in the current navigation mode. One relevant condition prior to move is to check that the PTU of the navigation cameras is pointing to the front. The other actions are related to the picture acquisition: `MovePTU` and `TakePicture`. First one moves the PTU associated to a specific camera from a start pointing to a desired pointing. The duration of this action depends on the movement, and the energy required is a fixed value multiply by the required time. To take a picture, we need a camera with a determined mode, orientation of its PTU and the rover stopped in the desired location. Time and energy consumption of the action is a function of the camera and the operation mode.

To define the time and energy constraints, each subsystem could have different operation modes with different duration and energy consumption. These elements are defined in the problem and employed in the domain to compute the correct values during the plan search. For instance, the locomotion subsystem has defined two modes related to the speed of the rover, expressed in m/s. For each speed, we also define the energy consumed. The duration and energy consumption for the `MoveTo` action is, thus, calculated using these values as a function of the distance between the current location and the next waypoint to reach.

Then, fig. 6.3 shows an example problem in which we employ the ExoMars rover to perform a set of scientific tasks. First, we need to define the possible objects, that is, the relevant locations, the cameras and their operation modes, the PTU positions and, finally, the rover speed and its navigation modes. Next, there is the rover data. It contains the initial position, energy, navigation mode and the energy consumption of each subsystem. This includes the PTU (with the pointing positions that follows a similar schema to the locations, i.e., `p15_30` indicates that the PTU is pointing with angles $pitch = 15°$ and $yaw = 30°$). Attached to the PTU is the navigation camera (`NavCam`), which has the low resolution mode (`lowRes`). As well, the time required to operate the different subsystems are provided.

The last part of the problem is the goal(s) definition. It defines the tasks that the rover has to perform, such as, go to a desired location, or take an image in a specific location and with a particular orientation and mode. If the planner supports plan preferences, one or more targets could not be satisfied by the planner, assuming a penalization for each unsatisfied goal. The goals defined for the rover example are: take two pictures and finish the plan in the initial location.

In the particular case of the TurtleBot we have also included a set of actions that allows the deliberative to decide when an energy charge (using a dock station) is required to accomplish the objectives, and, as a common goal, end the execution at a dock station. In fact, the charge action specifies how long the robot shall charge its batteries by modelling the energy charged per time.

```
(define (problem RoverProb)
 (:domain Rover)
 (:objects
     C0_0 C6_0 C5_-5 - wp
     NavCam - cam  lowRes - mode
     p0_0 p15_30 - ptupos
     slow fast - navmode
     exomars - rover
 )
 (:init
  (position exomars C0_0)
  (= (energy exomars) 1.44)
  (has_locomotion exomars)
  (navigation_mode exomars fast)
  (= (speed exomars fast) 0.2)
  (= (speed exomars slow) 0.1)
  (= (power_per_m exomars slow) 0.0025)
  (= (power_per_m exomars fast) 0.0032)
  (camera_mode exomars NavCam lowRes)
  (= (camera_energy NavCam lowRes) 0.001)
  (= (time_to_picture NavCam lowRes) 15)
  (PTU_pos NavCam p0_0)
  (= (PTU_energy) 0.001)
  (= (time_move_ptu p0_0 p15_30) 10)
  (= (time_move_ptu p15_30 p0_0) 10)
 )
 (:goal (and
  (picture C6_0 lowRes p15_30)
  (picture C5_-5 lowRes p15_30)
  (position exomars C0_0) )
))
```

Figure 6.3: PDDL problem for the ExoMars rover.

6.3 The executive layer

An executor is a system that runs between the deliberator and the functional layer. Its purpose is to read the high level orders generated by the deliberator and translate them into a sequence of orders that the functional layer can execute. At the same time, the executor has to supervise the execution of these actions: if an error arises during the execution, it may produce that the rest of the plan cannot be executed. In this situation the system will most likely fall into a wrong state or, in the worst case, the entire system will fail. Then, when an error happens, the executor must have a response to control that event. In some cases it could try to execute an alternative command. In other cases, it will call back the deliberator with the current state so that it can return a new plan.

For the execution system we have used the couple PLEXIL and its associated executor, the PE. This system has been successfully applied to prototypes of planetary rovers, a drilling equipment [24, 168], and to demonstrate automation for some operations in the International Space Station (ISS).

PLEXIL is a structured language to make flexible and reliable command execution plans. It is portable (since it uses eXtensible Markup Language (XML) for the encoding), lightweight, deterministic (given the same sequence of events from the external world) and very expressive. PE is the interpreter, designed to facilitate the inter-operability of P&E frameworks.

The PE executes PLEXIL plans that are designed to perform one or more tasks. A PLEXIL plan is a nodes tree that represents a hierarchical breakdown of tasks. Each node has a set of conditions to control its execution and a section that describes its function. The set of conditions that control the execution of a node (at start, at end and during execution) can be triggered by the internal state of the executor or by an external event (such as time, failures or commands requests). Also as part of the conditions, we can implement a resource model for enabling/disabling commands execution. With all these elements we can express if-then branches, loops and parallel or batch processes. Finally, the PLEXIL language allows us to define and use sub-plans, known as PLEXIL libraries. These libraries are PLEXIL plans that can be attached to other plans creating an hierarchy of plans, which provides us a structured way to made complex behaviours.

The PE executor is divided into three modules as shown in fig. 6.4. First, the PLEXIL core that implements the PLEXIL syntax and the algorithms to execute the plans. Then, there is a module for the management of the resources in case there are usage conflicts. Finally, the lowest module in the figure, is an interface module to connect the executor with an external system. The PE core algorithm processes each node of the PLEXIL plan when it reaches its depth in the tree structure and its starting conditions are fulfilled. When an internal or an external state changes, this change is propagated in cascade until there are no more nodes that can be executed. Before executing a node, if it employs resources, the resource arbiter checks if there are enough resources available to start its execution. Commands that are executed by an external system are described with a function name and its associated parameters, which are accepted by an interface defined in an interfaces configuration file. This interface must be implemented in order to connect the executor with a library that provides the functionality requested by the PLEXIL plan. Multiple interfaces can be defined as shown in fig. 6.4, and the executor accepts both, synchronous and asynchronous command execution.

The executive model is a set of hierarchical plans written in PLEXIL. The top PLEXIL plan is in charge of the planning/replanning processes, and the interaction between the executor and the planner control. This is associated with the high level interface adapter which allows reading the plan obtained by the deliberator and accessing and modifying the information contained in the high level model, keeping the deliberative database updated during execution. When the deliberative generates a plan, the PE reads the actions one by one, and executes them. Each action must be associated with a PLEXIL node (normally it is a PLEXIL library) that manages the correct decomposition and execution of the commands through the functional

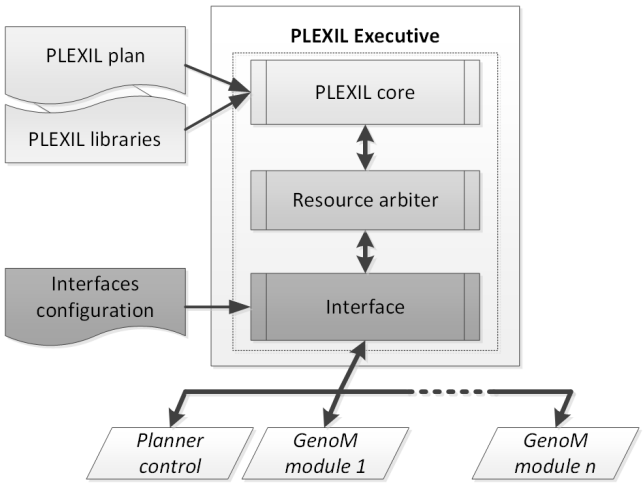


Figure 6.4: Representation of the PE modules and possible interfaces.

layer. This allows us to expand the functionality of the executor without modifying other components, for instance, by adding a new PLEXIL library to control a new functional command.

For each possible command, there is a functional module that executes it (currently G^{en}oM or ROS). Examples of possible actions obtained from the planner are *rotate* or *take a picture* as specified in the high level model. The *rotate* action corresponds to a rotation request of the *Locomotion* module and the picture acquisition to the *Camera* module. In the case of the *take a picture* action, some functional commands are required: change camera mode, take picture, compress it and save into a particular position of the memory to make it accessible. The executor is in charge of the correct execution of this sequence, and thus, it must monitor the command execution complying with the time and the resources constraints specified in the high level model.

In order to connect each G^{en}oM/ROS module with the executor, there is an interface adapter that communicates the module with the PE. This interface sends and monitors the *requests* to the G^{en}oM/ROS module, and catches the result sent by the module. If the execution is correct, the executor continues its plan without change, but if an error is reported, the PLEXIL plan is responsible for finding a solution. In our architecture, the reactive behaviours are designed to manage the failures of the plan execution at this level.

For the execution control, we have employed a top PLEXIL plan in charge of the planning/replanning process and a set of PLEXIL libraries that decompose the high level actions into low level commands, and execute and monitor them. The top plan is connected with the deliberator and can access the model data in order to read and update it when necessary. When a high level plan is obtained, the top PLEXIL plan reads the actions of the plan and executes them through the corresponding PLEXIL library. For example, the PLEXIL library responsible for the movement is connected to the *Locomotion* module (in G^{en}oM or ROS) and it manages the safety of the movement command. This library can react to unexpected problems during the movement, such as the presence of unknown obstacles or possible malfunctions

of the subsystem (e.g., overconsumption). In the first case, the executor updates the terrain information of the deliberator and requests a new path. In the second one, if the movement could continue (e.g., there is enough energy), the executor must take into account the new situation in terms of energy consumption and time needed to move, and adapt the resource model to ensure that the rover can continue its mission.

The basic execution flow for a general case of an action that causes a fault can be seen in fig. 6.5. When an action is executed, the outcome is read by the executor. If the action has been successfully executed, the plan continues nominally. However, when the action fails, a PLEXIL plan that implements failure tolerant behaviours shall be executed. These plans can attempt to execute the action several times or execute alternative commands trying to continue with the current plan. In the case that the failure tolerant behaviour cannot solve the problem, the executive updates the planner information, that is, the PDDL problem, in order to produce a new plan by invoking a replanning process in the deliberative layer.

Figure 6.6 shows the current PLEXIL plans using a graphical interface included in the PLEXIL distribution. The high level plan instantiates and runs the deliberative system, obtaining a plan. Then, the executor starts reading the actions and decomposes them into lower levels commands in order to relay them to the functional layer. The corresponding decomposition of the action into lower levels commands is guided by specific PLEXIL plans which are executed only when a match between the high level action and a PLEXIL library exists. This implies, for example, that the same executive model could be used to solve the previously presented domain and problem for the deliberator. So, the executive model can contain the operation, but only will be executed when the robot has defined the subsystem into the high level module, because a direct correspondence between high level actions and PLEXIL libraries to manage these actions exists.

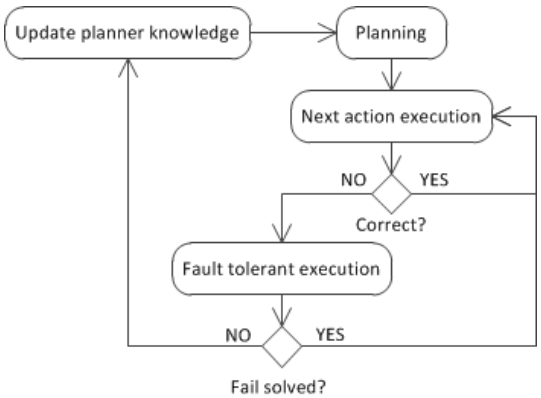


Figure 6.5: Definition of fault tolerant behaviours.

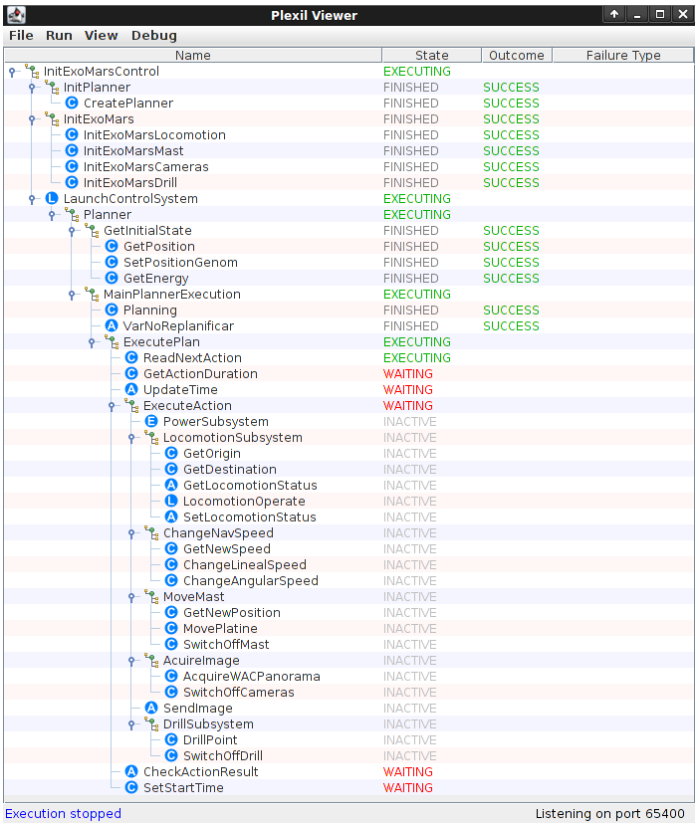


Figure 6.6: Graphical visualization of a PLEXIL plan during execution.

6.4 The functional layer

The functional level of the architecture must provide a good description of the capabilities of the hardware to control, and the implementation to safely operate it. Although every robot has different subsystems and technologies, there are possibilities to make standardized descriptions of subsystems functionality in order to obtain more adaptable and reliable implementations. Therefore, we have used G^{en}oM and ROS for this layer.

G^{en}oM [53, 105] is a framework that allows the definition of standardized modules. A module consists of a formal definition of the internal structures, the services that provides, and the associated software code which is responsible for the services execution. The framework allows us to model a subsystem into one (or more) module(s) and to perform the functionality of the module, which are invoked by external systems.

The creation of a module consists of two steps: a file definition (data structures, requests interface, etc.) and the services implementation of the *requests* into a piece of software called *codels* (which stands for code elements). When these elements are already generated (both can be refined a posteriori) the automated tools of the framework do the rest. The result is a server, which integrates the services and data of the module, and a series of interface libraries to access the *posters* (the module

data) and their *requests*. A G^{en}oM module provides a portable and reliable service, and allows us to integrate initialization services, interrupt routines, handle of failures, parameters checking and coordination of services. It is interesting to highlight that the module works in a server-client paradigm, that allows connecting more than one client to the module, which could be useful in order to monitor the module in real-time or enabling access to human controlled interfaces.

The MoBAr functional layer for the ExoMars simulation environment is implemented with G^{en}oM modules. Since every robot has different subsystems and sensors, the functional layer must be implemented for each particular one. Nevertheless, following a good design philosophy and using G^{en}oM, we can quickly adapt the MoBAr architecture for each robot. A G^{en}oM module first defines what it does, and then, how it does it. Then, if we correctly define the subsystems functionality, that is, the *requests* and *posters* of the module, we only need to implement the *codels* for each robot. This implies that the interface between the executor and the functional modules does not change, and thus, there is no need to modify the executor (except for the model to include/exclude functionality).

For instance, a typical *request* for the locomotion module is `rotate` θ `degrees`. The description of this service is the same regardless the locomotion mode, so we only need to change the associated *codel* that implements the particular locomotion method for every robot. If we use a legged robot, we implement the correct movements of the legs to perform the rotation, and if the functional module must manage a wheeled rover, the *codel* must control the velocities of each wheel. But the description of the module and the interface to access the functionality does not change.

Instead, ROS [150] has been used for the TurtleBot robot. ROS is a middleware placed on the operating system that provides a framework to develop reusable software for robotic systems. In essence, ROS operates as a distributed meta-operating system, offering a process execution environment and powerful interprocess communication mechanisms. In ROS terminology, processes are named *nodes*, and they interface the robot's sensors and actuators. In order to isolate potential errors, nodes are independent, and usually a robot control system involves a set of nodes.

In such a distributed processing environment, communication is essential and thus ROS provides a set of features to ease inter-node communication. The main tool is the *topic*, which implements a publish-subscribe pattern. In this way, for instance, a node controlling a sensor publishes its data through a certain topic and any node interested in getting those data listens to the topic. ROS also implements a one-to-one communication tool named *services*, where one node may offer a service to other nodes that consume it, similarly to a procedure call.

6.5 MoBAr as a black box

Using the previous domains and problems, we can consider MoBAr as a black box with a set of inputs (as shown in fig. 6.7) to define the specific behaviours and encapsulate the complexity of the underlying components. In order to specify a particular instance of the architecture, the following elements are required:

- A functional layer that provides the support to control the desired robot or simulator.

- The interfaces configuration file that defines the available operations given by the functional layer. This file also includes the operations to give access to the executor from the PDDL planner in order to obtain plans, analyse the output, and to read and/or modify the elements in the problem. This file is required by the PE.

- PLEXIL plans to manage the execution flow and decompose the actions into lower levels commands accepted by the functional layer. Our current top PLEXIL plan can be used as a template to manage the planning process.

- A configuration file for the planner. This file includes the domain, problem and DTM data, the PDDL planner to use, and the desired path planning algorithm to employ for the path search (only for UP2TA). As well, it is required some domain specific data to parse the planner output.

With these elements we can modify the behaviour of the architecture and adapt it to different robots without changing the implementation. We can define basic models and improve them later using an incremental schema. For instance, we can define a basic PDDL model to control the robot locomotion and the PLEXIL plan to manage it. Then, we can include cameras operations and the required task decomposition into the executor. Also, we can switch the path planner in the UP2TA system, or even, use only a PDDL planner, we can do it by simply modifying the planner configuration file employed by the PLEXIL plan. It is worth mentioning that our interface between PLEXIL and the planner is generic and can be easily adapted for different PDDL models and planners.

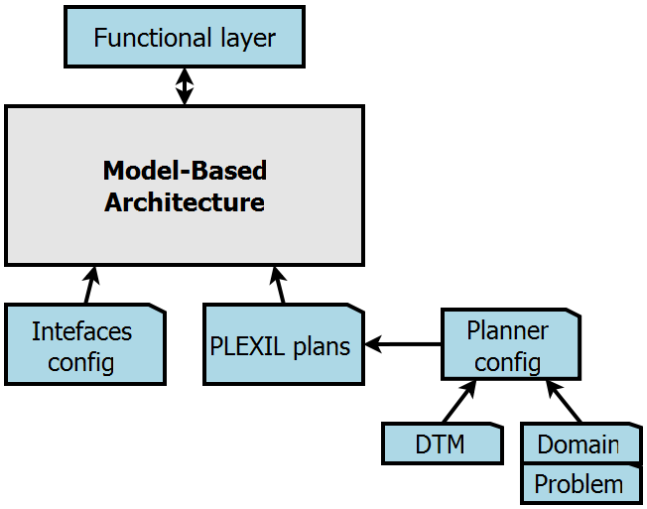


Figure 6.7: MoBAr as a black box.

6.6 MoBAr experimental evaluation

In the following subsections an experimental evaluation of MoBAr is provided. First, experiments have been performed using the ExoMars rover in a simulator suite. Second, some tests have been made using the TurtleBot robot. In both cases different PDDL planners have been exploited, including UP2TA.

6.6.1 Experiments with the ExoMars rover simulator

An important part of the investigation on Mars is the surface exploration and in-situ experiments. The ESA ExoMars rover (fig. 6.8) pretends to be the first robot to investigate the subsurface of Mars using a drill equipment and a subsurface sounding radar. It provides key mission capabilities: surface mobility, subsurface drilling (maximum 2 meters) and automatic sample collection, processing, and distribution to instruments, using solar panels to generate the required electrical power. It will host a suite of analytic instruments dedicated to exobiology and geochemistry research.

Moreover, the ExoMars has to exhibit autonomous capabilities: "*due to the infrequent communication opportunities, only 1 or 2 short sessions per sol (Martian day), the ExoMars rover has to be highly autonomous. Scientists on Earth will designate target destinations on the basis of compressed stereo images acquired by the cameras mounted on the Rover mast. The Rover must then calculate navigation solutions and safely travel 100 meters per sol*" [46].

The locomotion is achieved through six wheels. Each wheel pair is suspended on an independently pivoted boogie (the articulated assembly holding the wheel drives), and each wheel can be independently steered and driven. All wheels can be individually pivoted to adjust the rover height and angle with respect to the local surface, and to create a sort of walking ability, particularly useful in soft, non-cohesive soils like dunes. The camera system's images, combined with ground penetrating radar data collected while travelling, will allow scientists on-ground to define suitable drilling locations.

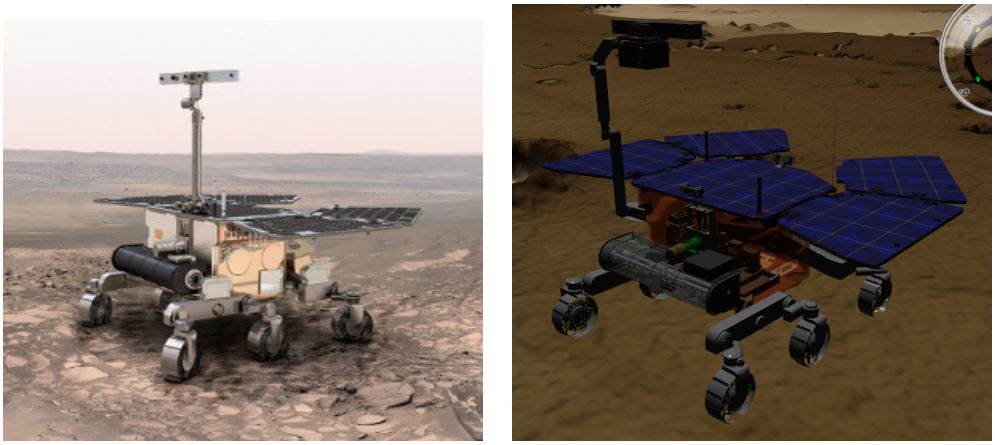


Figure 6.8: ExoMars artistic representation (left) and rover model in the 3DROV simulator (right).

To test the rover model and the control system, ESA has developed the 3DROV planetary rover simulation environment [146], which allows early-stage virtual modelling of terrain and mobile robots systems. This suite includes last model of the ExoMars rover as shown in fig. 6.8 (right). The simulation environment is composed of multiple modules connected through standardized interfaces, being the most important the *Simulation Framework*, that is the ESA's Simsat, responsible for the execution and scheduling of the simulation and the *Generic Controller* that manages the on-board flight software. This module allows us to connect software modules to control the rover. Also, it includes the *Environment block* in charge of the timekeeping, terrain and atmospheric conditions, and the *Visualization Environment*, a front-end that provides real-time visualization of the simulation progress.

ExoMars is a complex rover that requires engineering work to implement the control architecture, specially the functional layer. Thus, the incremental design possibilities of the MoBAr architecture allows making a first functional version of the autonomous control despite some of the subsystems are not functional [127,132]. Figure 6.9 (left) shows a conceptual vision of the MoBAr architecture deployed to control the ExoMars rover, and, in fig. 6.9 (right) there is a visualization example of a resolution of a small problem using the 3DROV simulation environment.

The functional layer deployed to control the rover is formed by some G$^{\text{en}}$oM modules, but others can be added in the future (such as the control of the scientific load for example):

- *Power:* this module is in charge of the energy of the rover. It takes into consideration the available power, the recharge rate of the solar panels and the consumption of the active subsystems to guarantee the correct thermal and energy state. In that sense, it can deny power on subsystems when the consumption overcomes the nominal design margin and it is able to send critical alerts to the executor in order to assure the safety of the vehicle.

- *Locomotion:* to control the locomotion and the position of the rover. This module implements a set of *requests* to move the rover and to update its position. The position data is exported via *posters* to grant direct access to the executor to this information.

- *Storage:* the memory management and the compression rate for the scientific data is performed by this module. All the scientific related modules that need to read or write data must do it through this module.

- *Communication:* this module contains another two modules, one to control the omnidirectional antenna and another one for the high gain antenna. The access to these two modules are controlled by the *Communication* module that has the visible interface with the executor. Also, it is in charge of the reception of the telecommands from the ground station and relay them (when appropriate) to the executor.

- *Cameras:* to access the multiple cameras of the rover, this module provides an interface to the executor. Each camera has its own module to implement the camera functionality (for example, shot mode or filter).

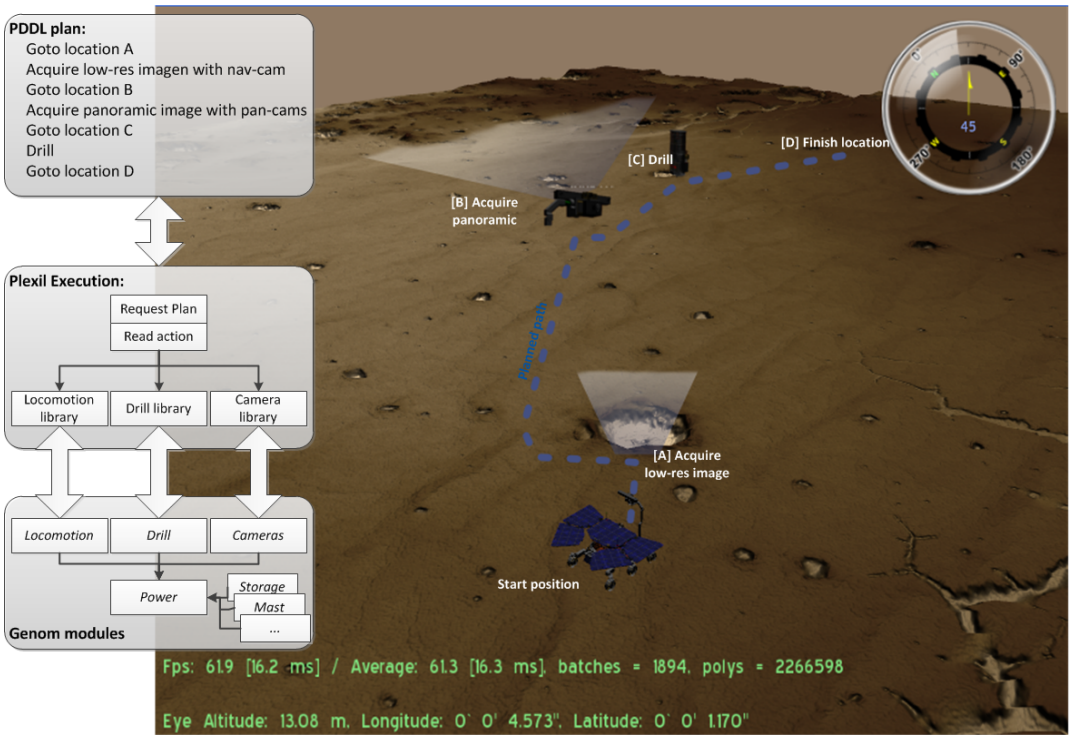


Figure 6.9: ExoMars MoBAr models and execution under the 3DROV simulator.

- *Mast:* this module implements the mobility of the mast.
- *Drill:* it controls the subsurface drill equipment.

Until now, we have described the ExoMars subsystems and the specific functional layer deployed to control them. In the following, we present the experiments performed using the models presented along this chapter, exploiting the next configuration for the three layers:

- The ExoMars G^{en}oM functional layer connected to the 3DROV simulator suite.
- The PLEXIL plans without replanning capabilities or reactive behaviours.
- The UP2TA planner and the task planning and path planning interleaving schema presented in sec. 5.2 (see fig. 5.1b).

Currently we are more interested in assessing the deliberative layer due to: (i) the functional and executive layers are more related to the engineering level, and its development is time consuming. Then, we only implement the required behaviours to enable the integration with the deliberative layer; and (ii) in this dissertation we have proposed different path planning algorithms and a new P&S system for mobile robots. Thus, we are interested in assessing how well these assets support autonomous operations when dealing with a robotic platform. For this reason, the experiments are only focused on the deliberative layer.

We present a comparison coupling the task planner with a path planner to ensure safe routes on a Mars-like terrain. We use the ExoMars rover and its simulation suite, which provides a realistic physics for the wheel-soil contact. Moreover, we use 3Dana for path planning as we are exploiting a rover with a suspension system in a complex terrain. Due to the complexity of the model, we have only used two planners that can deal in an efficient way with it: SGplan [73] and OPTIC [15]. We have employed both planners to solve the same problems with different number of tasks, interleaving with 3Dana for the path planning phase. We called these deliberatives SGplan$_{3D}$ and OPTIC$_{3D}$. Also, we have included UP2TA that uses FF, and thus PDDL version 1. In such case, we lose the possibility to model time and energy consumption in actions.

The results are presented in tab. 6.1, and show the runtime to obtain a plan and the total length travelled to achieve the mission goals. In the case of UP2TA, we obtain better results for the search time due to the simplicity of the model (it does not have the energy model neither temporal actions). However, it is remarkable the difference in the distance travelled: UP2TA clearly outperforms the interleaving schema of task planning and path planning used in SGplan$_{3D}$ and OPTIC$_{3D}$. In this way, the search integration of the task planner and the path planner in UP2TA, as happened in the standalone experiments (see sec. 5.7), demonstrates the effectiveness of the approach. In this regard, fig. 6.10 shows full routes for two problems obtained by UP2TA planner with 6 and 12 tasks. The path followed by the rover is marked with a line, while the tasks are denoted with a dot. In the problem definition, one of the tasks is marked to be the last one, in order to finish the route in that position.

Table 6.1: Results for the execution of different problems with the UP2TA system and two different PDDL planners combined with 3Dana using the ExoMars simulator.

Deliberative	**# tasks**	**Search time (ms)**	**Path length (m)**
SGplan$_{3D}$	6	2007	413.780
OPTIC$_{3D}$	6	2653	389.60
UP2TA	6	1661	164.363
SGPplan$_{3D}$	12	8364	1785.121
OPTIC$_{3D}$	12	69800	1386.400
UP2TA	12	8917	718.340

6.6.2 Experiments with the TurtleBot platform

Figure 6.11 shows the TurtleBot 2 [141] robot that we have used to test the MoBAr controller. The reasons to choose this platform among all the available in the market was twofold: the low-cost and the possibility to use open-source software (including ROS).

By default, the TurtleBot provides a mobility subsystem based on two motorized wheels with integrated encoders and its electronic support to provide odometry. It has a maximum lineal velocity of 65 cm/s and rotational of 110 deg/s. Also, it has

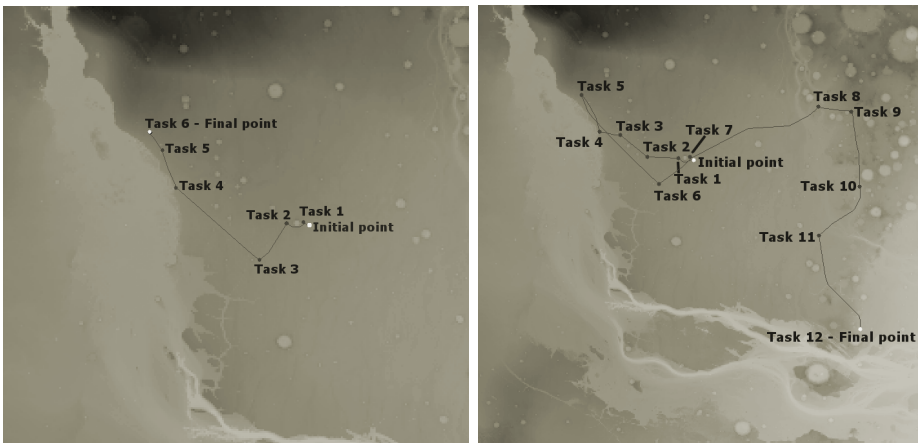


Figure 6.10: Example of solutions for a problem with 6 tasks (left) and 12 tasks (right) solved with UP2TA using 3Dana for path planning. We do not show paths for SGplan$_{3D}$ or OPTIC$_{3D}$ as the paths have several intersections that make hard to follow the route.

three infrared sensors and a frontal collision detector. Typically, it is used with an attached Microsoft Kinect camera, which provides optic and depth field images. In our robot we have mounted the Kinect on top of the robot with a PTU (formed by two servo motors) to enable camera pointing. The battery enclosed in the robot base has a capacity of 4.4Ah. The total mass including the control hardware is near 7 Kg with a dimension of 354 x 354 x 550 mm.

To control it, we have used a small factor PC endowed with a dual core Intel Atom processor, 2GB of RAM with Ubuntu 14.04 and ROS Indigo. Also, we have added an Arduino board to control the PTU and other sensors that will be added for specific applications.

With this configuration, we are able to perform autonomous navigation, picture acquisition and autonomous docking in recharge stations. Since we apply a long-term path planner with a map containing the environment, we only use the odometry for autonomous navigation, avoiding the Kinect based navigation and its high computational and energy requirements.

In opposition with the previous platform, the functional layer of the TurtleBot is implemented using ROS nodes and connected with the executive with two PLEXIL interfaces; one to connect to the ROS services (navigation, camera and energy) and the other one to Arduino. The services required to provide access to the functional layer are encapsulated into Remote Procedure Call (RPC) servers, that allows easy interfacing with PLEXIL, while also provide a simple way to deploy tele-operation systems.

The models for the MoBAr deliberative and executive are closely the same that the one exploited for the ExoMars simulator. In the case of the TurtleBot we have included in the PDDL model an action that allows the deliberative to charge the battery in a dock station if there is not enough energy to accomplish all goals. In particular, the duration for the recharge action is set to 10 seconds, recharging near 0.1% of the battery capacity.

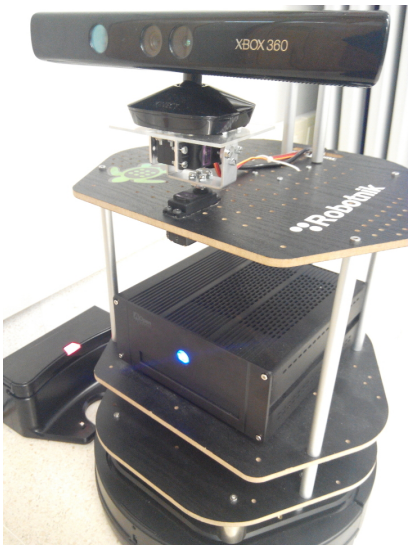


Figure 6.11: TurtleBot robot in our lab facilities. The dock station is visible at left-bottom.

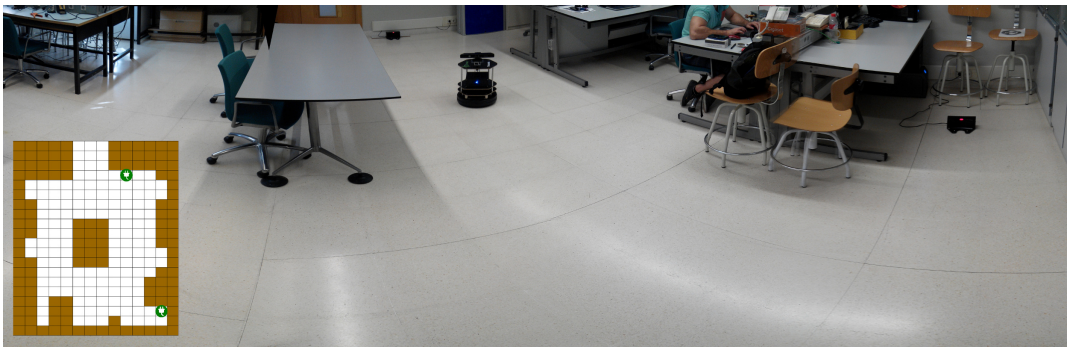


Figure 6.12: Panoramic view of the test area and its map representation.

Related to the path planning, we do not use 3Dana since we are not considering the DTM (the TurtleBot is designed to work in flat areas). Thus, we have exploited S-Theta* (see sec. 3.4) with an extension that defines a safety margin restriction to avoid movements close to obstacles. This means that the algorithm considers an area or margin surrounding every obstacle that the robot is enforced to avoid, but if there is no other possibility, it can ignore the area at the expense of performing low speed movements and moving only between the adjacent cells (i.e., behaving like A*).

Taking into consideration these facts, the evaluation consists of acquiring different pictures in a room (shown in fig. 6.12). The floor has a dimension of 8.0 x 5.6 m, modelled as 20 x 14 cells (each cell is 0.4 x 0.4 meters). In the test scenario there are two docking stations to recharge the battery as they can be seen in the figure and on the map (in green). At the end of each test, the robot shall finish in a dock station. However, for the initial position, the robot starts at different random locations, and, sometimes, over a dock station.

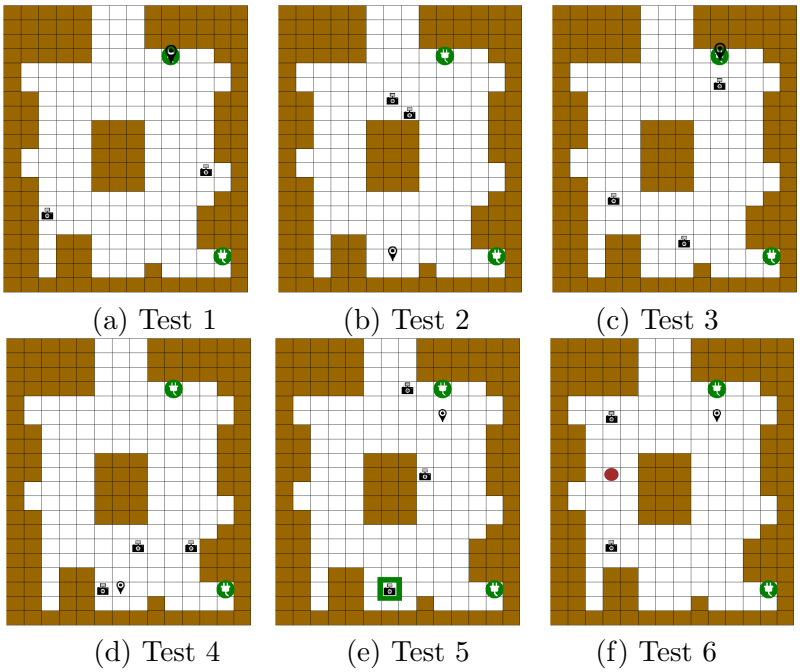
For the deliberative layer, we have tested the previous PDDL planners: SGplan and OPTIC. As we are interleaving them with the S-Theta* path planning algorithm, we call them SGplan$_{S\theta}$ and OPTIC$_{S\theta}$. Following, we present the different scenarios tested with the TurtleBot where we have measured the execution time, deliberation time, the distance covered by the platform and, for those scenarios in which there is no replanning, the estimated distance provided by the planner. We attempted to provide the battery consumption but the TurtleBot platform does not allow us to obtain accurate values, so we discard such data. For this reason, we model in PDDL the battery recharge when its level falls to 20%.

We present the results for the 6 different scenarios represented in fig. 6.13 with the following elements: the docking stations (green), the robot position (black marker), the initial goals (black cameras), the goals requested by the human operator during execution (camera surrounded by a green square) and initial unknown obstacles (red circle) if any. During the tests, the functional layer resides in the TurtleBot PC, while the deliberative and executive layers of MoBAr are running on a laptop connected via WiFi to the robotic platform. Next, each scenario is described in detail.

Test 1 (fig. 6.13a): starting at a dock station with full battery, the objective is to acquire two pictures. This is the easier scenario or nominal operation where the goals are achieved without any replan or recharge.

Test 2 (fig. 6.13b): this is similar to the previous one, but the initial position is located at the bottom of the map and the initial battery capacity is not enough to achieve all objectives. Thus, both planners generate a plan in which an intermediate recharge is required.

Test 3 (fig. 6.13c): in this case, the robot starts at the top dock station and the



(a) Test 1 (b) Test 2 (c) Test 3

(d) Test 4 (e) Test 5 (f) Test 6

Figure 6.13: Initial configurations for the TurtleBot test scenarios.

goal is to acquire three pictures. The initial battery charge does not allow to achieve all goals, and thus, both planners provide a plan in which a recharge is performed before achieving all objectives.

Test 4 (fig. 6.13d): this scenario is similar to the previous one, but the battery charge is very low and the pictures requested are quite close. Due to the low initial battery level, the generated plan requires, at least, one battery recharge. In particular, SGplan$_{S\theta}$ plans 4 intermediate recharges, while OPTIC$_{S\theta}$ only 2.

Test 5 (fig. 6.13e): this scenario starts with the robot near the top dock station and two goals defined. When both pictures are taken, during the path to the nearest dock station, the human operator requested a third picture. At that point, the system deliberates to generate a plan to take the new image and then, go to the dock station.

Test 6 (fig. 6.13f): in this scenario the robot initial position is close to the top dock station and there are two goals. The first picture is taken without problem, but, on the way to the second one, the robot hits an unknown obstacle. Then, the map is updated to include the obstacle and the deliberative obtains a new plan to overcome it and take the last picture. The robot ends at the dock station.

The parameters measured for each scenario are presented in table 6.2. We have measured the following:

- **Execution time**: the total time to achieve the mission goals in seconds.
- **Deliberation time**: time in milliseconds spent deliberating to generate the plan(s).
- **Distance planned**: predicted distance in meters that the robot shall travel to achieve its goals. It is provided by the deliberative layer. We only provide this value when there is no replanning.
- **Distance travelled**: is the total distance in meters measured by the robot odometry.

With these results we can observe that the robot is able to achieve the goals in different scenarios using various PDDL planners with the same model. However, OPTIC$_{S\theta}$ outperforms SGplan$_{S\theta}$ except for the time spent in deliberating. In this regard, we can see that it is preferable to deliberate during more time to achieve a better plan that, in some cases, can reduce the distance travelled (and thus, the execution time and battery required) to half. This is the case of scenarios 5 and 6, in which OPTIC$_{S\theta}$ requires more deliberation, but provides plans that better optimize the distance.

We can see that different planners take different routes to accomplish the objectives, even if the initial conditions and the path planner algorithm are the same. Then, evaluating and comparing deliberatives in real robotics platforms deserves a deeper analysis than evaluating them with a small set of variables acquired from a set of test scenarios. Meanwhile both planners achieve the solution, SGplan$_{S\theta}$ has better search times than OPTIC$_{S\theta}$, which obtains better executions times and distances travelled.

Table 6.2: Parameters measured for the TurtleBot test scenarios.

	Planner	Execution time (s)	Search time (ms)	Planned dist. (m)	Odometry dist. (m)
Test 1	SGplan$_{S\theta}$	505	47.416	15.609	15.663
	OPTIC$_{S\theta}$	332	51.629	9.5276	9.2895
Test 2	SGplan$_{S\theta}$	425	58.229	11.5572	11.5758
	OPTIC$_{S\theta}$	328	337.597	10.3276	10.2951
Test 3	SGplan$_{S\theta}$	750	48.303	19.26	19.222
	OPTIC$_{S\theta}$	618	221.204	15.0084	14.9527
Test 4	SGplan$_{S\theta}$	1079	57.393	24.4852	24.4593
	OPTIC$_{S\theta}$	736	2567.94	13.9848	13.843
Test 5	SGplan$_{S\theta}$	588	120.153	-	23.194
	OPTIC$_{S\theta}$	427	201.463	-	11.0791
Test 6	SGplan$_{S\theta}$	672	163.328	-	26.5437
	OPTIC$_{S\theta}$	380	268.921	-	12.3835

Moreover, we have performed another experimental evaluation using the TurtleBot without considering the temporal and energy constraints. This enables to compare UP2TA with the previous employed planners. The test consists of the two following scenarios:

- Acquiring six pictures using the previous domain.
- Acquiring three pictures, and getting and delivering a sample. This domain is an extension of the previous one that includes a collecting sample and deliver goal. In this regard, the robot has to reach a position in which a user puts an item on top of the robot and then, the robot shall deliver such item to another user. Also, there is only space for one item on the robot tray, so only one item is allowed.

We have performed each scenario three times to gather average values of the execution. We still use S-Theta* for the path planning side. The room used for the experiments has a dimension of 7.2 x 8.4 m modelled as 18 x 21 cells (each cell is 0.4 x 0.4 m).

Table 6.3 shows the results for the pictures acquisition scenario. The data shows that UP2TA obtains better results that the interleaving approach in terms of path length and execution time, but at the expense of a larger deliberation time. We can see that SGPlan$_{S\theta}$ and OPTIC$_{S\theta}$ generate two times longer paths than UP2TA. However, it is remarkable that the path provided by SGPlan$_{S\theta}$ requires more heading changes, slightly increasing the execution time. Figure 6.14 shows the location of the first scenario goals and the generated paths for each planner. The dots represent

Table 6.3: Parameters measured for 6 pictures acquisition scenario.

Planner	Search time (ms)	Execution time (s)	Planned dist. (m)	Odometry dist. (m)	Turn (°)
SGPlan$_{S\theta}$	47	395	23.44	23.16	527
OPTIC$_{S\theta}$	68	370	23.57	23.32	428
UP2TA	308	251	11.10	11.41	534

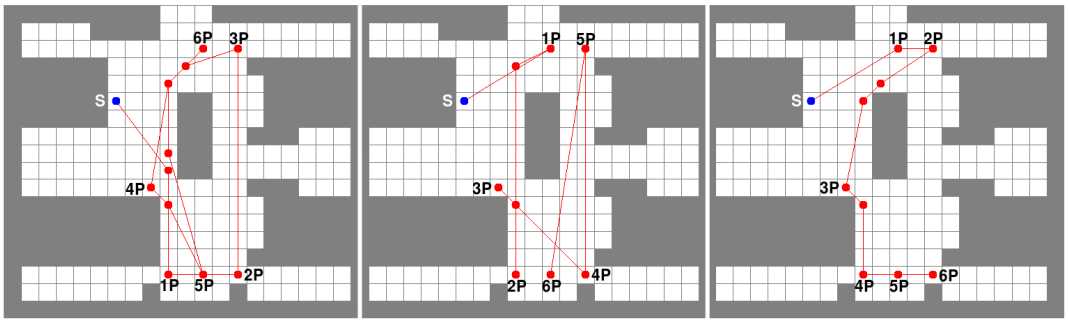


Figure 6.14: Planned path for 6 pictures acquisition scenario. From left to right: SGPlan$_{S\theta}$, OPTIC$_{S\theta}$ and UP2TA. #P identifies the order in which pictures are taken. S denotes the start position.

points in which the robot performs a heading change. The initial position of the robot is represented in the map by the S label. The #P labels identify the positions of the taking picture goals, preceded by a number that represents the order in the plan. We can see that UP2TA provides paths easier to follow compared to the ones generated by SGPlan$_{S\theta}$ and OPTIC$_{S\theta}$. These two systems provide paths that traverse from south to north and vice-versa several times. This increases the distance travelled and, therefore, the execution time. If we analyse the routes, the path generated by SGPlan$_{S\theta}$ first acquires the picture on the bottom left (labelled as 1P) and then traverses to the bottom-right where it acquires the second picture (2P). However, it would be better to complete task numbered as 5P than 2P, which is on the way. The same happens when trying to traverse from the start position to the first picture; there is a goal in the middle of the map (4P) that can be acquired, but it is achieved later on. OPTIC$_{S\theta}$ has a similar behaviour. Instead, UP2TA acquires first the picture on the top-middle (1P) of the map followed by the one at the top-right (2P). Then, it goes to the bottom of the map, but acquiring the picture in the middle (3P) before reaching the south side. Then, it takes the last pictures of the bottom from left to right (4P, 5P and 6P).

For the sample and delivering scenario, the results are presented in table 6.4. In this scenario UP2TA also requires more time to achieve a plan than SGPlan$_{S\theta}$ or OPTIC$_{S\theta}$. However, the length of the path is nearly 33% shorter than the one generated by OPTIC$_{S\theta}$. Also, OPTIC$_{S\theta}$ outperforms SGPlan$_{S\theta}$ in distance and execution time. The paths generated can be seen in fig. 6.15. In the same way as the previous scenario, paths are represented by lines and heading changes as dots. In this case, there are three labels: P for Pictures, T for Taking a sample and D

Table 6.4: Parameters measured for 3 pictures acquisition and 3 samples delivering scenario.

Planner	Search time (ms)	Execution time (s)	Planned dist. (m)	Odometry dist. (m)	Turn (°)
SGPlan$_{S\theta}$	72	672	41.21	41.71	814
OPTIC$_{S\theta}$	138	582	34.93	35.29	766
UP2TA	5041	480	26.55	26.82	615

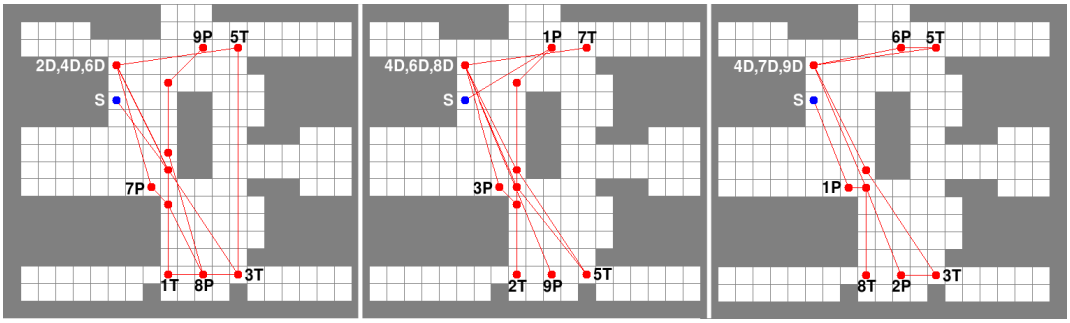


Figure 6.15: Planned path for 3 pictures and 3 samples delivering scenario. From left to right: SGPlan$_{S\theta}$, OPTIC$_{S\theta}$ and UP2TA. Actions in sequential order: P=picture; T=take sample; D=deliver sample.

for Delivering a sample (the deliver point is on top of the start position, S). If we analyse the paths followed by SGPlan$_{S\theta}$ or OPTIC$_{S\theta}$, we can observe that, usually, they do not interleave pictures acquisition with samples collecting/delivering. For example, SGPlan$_{S\theta}$ first takes the sample at the bottom-left (1T) and then delivers it (2D). Then it acquires the sample at the bottom-right (3T) and delivers it (4D). However, the picture at the bottom-middle (8P) or the one in the middle of the map (7P) are delayed to the end of the plan. This issue results in longer paths and suboptimal solutions. In a similar way, OPTIC$_{S\theta}$ does not properly interleave samples acquisition/delivering with pictures acquisition. It first acquires the picture at the top-middle (1P) and then acquires the sample at the bottom-left (2T). It could have been a better decision to take the sample on the right of the first taking picture (7T), but instead, it has decided to move to the south. Then during the traverse to deliver the sample (4D), it acquires the picture at the middle of the map (3P). Then, it acquires the sample at bottom-right (5T) and delivers it (6D). Again, it delays the action of taking a picture close to other goal to the end of the plan (9P). This is a consequence of the domain independent heuristic used by the PDDL planners that do not properly consider the distance travelled during the planning process, delaying goals even when the robot traverses nearby them. In the case of UP2TA, it performs picture acquisitions while moving to the sample acquisition goals or to the delivery position. It starts moving to the south of the map, acquiring the picture in the middle (1P) and then the one in the bottom-middle (2P). After, it takes the sample at the bottom-right (3T) and delivers it (4D). The next goal is the sample at the top-right (5T) but, before delivering it (7D) it acquires a picture

(6P). At this point, all pictures goals are achieved and there is only one task sample remaining. UP2TA takes the sample (8T) and delivers it (9D). We can conclude that sharing information between the path-planner and task-planner allows interleaving goals that are close to each other, which results in better solutions.

6.7 Summary

In this chapter we presented a description of MoBAr, an autonomous controller focused on models and high level behaviours. In this direction, the implementation of MoBAr is done by means of general purpose technologies. The deliberative and executive layers provide languages for modelling the behaviours for the deployed application (i.e., PDDL for the deliberative and PLEXIL for the executive), which reduces the time required for the implementation of the controller and allow focusing the efforts in defining the behaviours. Then, we presented the models for an exploration domain that is operationalised through a simulator and a real robotic platform. This shows that the controller enables the use of different P&S technologies, while dealing with different execution scenarios that require on-board replanning capabilities.

Chapter 7

A framework for autonomous controllers assessment

In this chapter we address the open issue of evaluating and characterizing autonomous controllers, particularly focusing on the P&S systems and their integration in robotics. For such purpose, we implement a software tool that, following the defined methodology and performance metrics, enables to generate objective and reproducible plan-based controllers assessment. First, we introduce the structure for assessing autonomous controllers. Following, we define a methodology for the testing process with the objective of generating objective and reproducible experiments. Then, a set of general and domain independent metrics that enables comparison among different P&S and P&E approaches for robotics applications are introduced. The metrics are formally defined and exemplified in MoBAr in sec. 7.5 for the planetary exploration domain introduced in sec. 7.4. Then, we evaluate the metrics in GOAC. In sec. 7.7 we summarize the software tool that allows automatic benchmarking. Finally, the framework is evaluated by comparing MoBAr and GOAC, demonstrating the effectiveness of the approach.

7.1 Toward autonomous controllers assessment

When we are interested in evaluating the performance of an autonomous control architecture, we need to take into account all the components of the architecture, not only the deliberative ones. The configuration of the architecture, that is, the hierarchy built on top of a set of different components and how they are connected, plays a fundamental role: some planning technologies generate a complete plan before executing it, while others generate partial plans, interleaving planning and execution in a loop. This implies questions such as the different delays interchanging data between components that affects the performance of the system. In order to analyse different deliberatives, we need to employ the same functional support, while employing different technologies for P&S over the same problems. Thus, we can obtain valuable information to better understand how the different components interact, and to select the best controller and configuration for a particular problem.

If we want to analyse and to compare autonomous controllers, we need to provide not only a valid methodology for the testing process, but also a tool that helps and

guides users during the different phases. Since in the current state of the art we cannot find any system that accomplishes both objectives, we need to start defining the requirements for such effort. In that sense, starting from lessons learned from the experiments reported of different autonomous controllers, and from the different works that try to characterize and evaluate autonomous controllers, we identify a set of requirements to cope with the purpose of defining a methodology and testing environment for objective evaluation of autonomous controllers. These requirements are the following:

- **Objective of the test**: first, we need to define what is the objective of the test. It is not the same to compare two autonomous controllers in the same scenario than evaluate the performance of an autonomous controller in different scenarios. For the first case we need to analyse general parameters applicable to both controllers, while in the second one we can be more specific.

- **Metrics**: in function of what is our objective, we need to define what we want to measure. This is a fundamental step: the result of the evaluation depends on the selected metrics. It is important to define at least a small set of metrics that are commonly applicable to all autonomous controllers. Moreover, if we want to focus on a particular autonomous controller, we can include specific metrics for that controller in our tests.

- **Scenarios**: the scenario can be defined as the initial set of constraints and goals that the autonomous controller takes as input. In general, we will have a domain that represents the interactions between the world and the robotic platform, and a problem, that defines the objectives to accomplish. However, to deal with uncertainty, this classical approach is not enough: autonomous controllers shall deal with challenging scenarios in which the environment is dynamic; external agents can send more goals during execution and even some failures can occur, requiring capabilities such as replanning and failure recovering schemes. Then, it is required that the description also includes different execution scenarios.

- **Instantiation**: usually it is required to generate different files that combine various sets of initial constraints and goals (scenarios), which are lately employed by the autonomous controller. Also some autonomous controllers can be deployed with different configurations, so, a tool that simplifies and automates this task to allow executing a large number of tests is necessary.

- **Execution**: an autonomous controller often requires some manual work to be executed. This is an important lack that shall be solved if we want to perform exhaustive tests campaigns. A tool that automates the process of instantiating, executing, monitoring, collecting and analysing the data of several executions of an autonomous controller is required. Also, that tool shall support modifications of the nominal execution, by automatically injecting goals or failures to test different operative scenarios.

- **Evaluation**: with the data gathered an objective evaluation shall be performed. Since a large amount of data can be produced during the execution,

we need a tool that compacts the information into standard reports to be shared along different research centres. This implies not only to create a tool, but also to define a general representation for the data produced.

- **Reproducibility**: if we want to provide valid experiments, they shall be reproducible by others researchers. In this way, providing the metrics, the scenarios definition and the configuration of the autonomous controller, each researcher must be able to reproduce the results by her/himself.

- **Costs**: next to the reproducibility are the costs. Typically the experiments are done using robotics platforms that are expensive or even unique and not accessible to all researchers. Thus, experiments done in this way are not reproducible. Then, a solution is to employ software simulators publicly available and/or accessible robotic platforms.

For this reason, we address the problem of creating a software environment and a methodology for testing the autonomous capabilities of robot controllers, in particular, those endowed with deliberative capabilities. This framework, called On-Ground Autonomy Test Environment (OGATE) [120–122], is intended to be an integrated environment to test features of autonomy software of different autonomous controllers. Moreover, its objective is to enable quantitative comparison based on accurate experiments and qualitative analysis allowed by inspection and visualization of software internal monitors of the controlled system. Also, the possibility of creating experiments concerning either single features or a combination of them is required to analyse and obtain a better comprehension of the interactions between the different layers of the architecture. These features are enclosed in a methodology for defining the testbench and a set of metrics that enable general autonomous controllers assessment. Both elements are operationalised in the OGATE software tool, which provides the automated support to instantiate, execute and evaluate autonomous controllers under a unified platform. It is proposed as a partially interactive tool to help designers and operators of autonomous controllers to deal with performance evaluation, while also provides a unified interface for in-execution control and metrics inspection of the controlled system. Focusing on the performance evaluation, OGATE acts as an automatic tool instantiating and executing the required components of the autonomous controllers and scenarios defined by the user, supervising the execution and generating a report with the metrics retrieved.

7.2 A methodology for autonomous controllers assessment

If we want to analyse the performance of a plan-based controller while generating reproducible assessments, we need to follow a common methodology. In the same way as Gertman et al. [65], we propose a methodology for autonomous controller assessments composed on sequential phases. Our methodology can be described as the composition of three sequential phases: evaluation design, tests execution and, report and assessment.

In general terms, given one or more autonomous controller to be assessed, a set of evaluation objectives should be isolated and some performance metrics must be formally identified. Then, a set of suitable experiments should be defined and performed in order to collect relevant information that will constitute a quantitative basis for the evaluation process. Finally, reports should be generated to point out measurements indicating the performance of the controller according to the evaluation criteria and metrics defined in the first phase. Particularly, a synthetic view of measurements can be generated, e.g., through graphical reports, to simplify the comprehension of the performance data. In the following, each phase is described in detail.

1) Evaluation Design. First, it is required to identify the evaluation objectives. In fact, according to the evaluation target different aspects may result relevant (or not). For instance, measuring the distance travelled or battery consumed could provide relevant information. In this case, very specific parameters can be considered and analysed. Regarding to the evaluation objectives, a metrics definition task is required to determine the parameters that should be measured during experiments execution. By means of a set of general metrics it is possible to compare performance of different control systems in the same operative scenario. In this way, for general assessments of autonomous controllers, metrics defined in sec. 7.3 will be applied. Besides, focusing on a particular autonomous controller or an operative scenario allow considering specific metrics in addition to the general ones. For each metric μ_i considered, a metric weight, μ_i^W, that represents the *relevance* of the metric in the assessment shall be defined. This is a key as the result of the evaluation strongly depends on the selected metrics and its relevance. In the following, considering a set of n metrics, the sum of all weights is supposed to be 100 as eq. 7.1 shows.

$$\sum_{i=0}^{n} \mu_i^W = 100 \tag{7.1}$$

Then, the definition of different scenarios and configurations to be tested should be implemented. The scenarios can be defined as the set of constraints and goals that the autonomous controller takes as input, i.e., the domain and problem.

However, to also deal with uncertainty, more than one scenario should be considered to investigate the behaviour of the autonomous controller under different conditions. For instance, external agents that can dynamically generate additional goals, or failures that may occur during execution are typically situations which an autonomous controller should deal with. This is done by means of autonomous capabilities, e.g., replanning or failure recovering schemes. In this way, we consider three general cases that an autonomous controller should deal with in real operative scenarios: (i) nominal execution, when everything goes as expected; (ii) dynamic goal injection, an extension of the nominal execution in which one or more goals are dynamically included during the system operation; and (iii) execution failure, when some components of the system induce a non-nominal behaviour. This forces the controller to adapt its plan to overcome the contingency. Failures in that case can be due to external perturbations, mechanical failures or degradation of the system over time.

2) Tests Execution. Performing tests entails the execution of each scenario to be monitored. In case of uncertainty and/or uncontrollable tasks are part of the problem, each scenario should be performed several times, to collect average behaviours and define metric values.

In this regard, a scenario instantiation step is required to generate the set of models needed to define a suitable set of planning domains and required goals. Also, autonomous controllers can be deployed with different internal settings and, thus, scenarios instantiation should also consider to enable the execution of tests with different configurations.

Then, the tests execution is needed. This is an important step for instantiating, executing, monitoring and collecting the data after several executions of an autonomous controller in a given scenario. During the tests execution, modifications of the nominal execution should be considered by automatically injecting goals or failures to also test non-nominal scenarios.

After execution, with the data collected, each metric defined has to provide a metric score, μ_i^S, ranged from 0 to 100 (better score) that defines the performance. In the case of the controller cannot accomplish the mission goals, all the metrics scores for that execution are 0. This means that failing to achieve the objectives will have a negative impact in the controller evaluation.

3) Report and Assessment. Once all the tests are completed, a report with the information gathered during several executions shall be provided. Reports contain an insight of the controller behaviours, providing values for each metric as well as generating synthesized views, e.g., by means of graphical representations to support the users while analysing the system performance. In fact, the information provided within the reports is to inform the users and to enable a performance assessment, allowing an objective evaluation of the control architecture in the different considered scenarios.

Within the assessment, we propose a controller evaluation by means of a synthesized value: the Global Score (GS). The GS provides a score between 0 and 10 considering the linear combination of all metrics defined in the controller assessment. Each metric is determined by two values: its relevance, μ_i^W, as defined in the first phase of the methodology (see again eq. 7.1); and its score, μ_i^S, obtained through tests execution. Then, the GS is computed as in eq. 7.2.

$$GS = \frac{\sum_{i=0}^{n} \left(\mu_i^S \cdot \mu_i^W\right)}{1000} \tag{7.2}$$

After execution, a huge amount of generated data is expected and thus, it is desirable to provide reports that are human comprehensible. For instance, in ALFUS [108] and PerMFUS [75], a three axis representation is presented. In a similar way, we propose a circular representation, such as the one depicted in fig. 7.1, to represent the autonomous controller assessment. In such graphical representation the metrics scores are divided into the four quadrants. The division of the metrics scores simplifies the comprehension of the evaluation, while also allows focusing on a particular

metrics group (see next section and fig. 7.2) to perform controller comparisons. The graphical representation has three different areas. Namely, starting from the center, the GS, the execution times and the metrics scores area.

The GS is shown at the center. Surrounding it, from one to three circular bars are depicted. These bars provide the average time required by the considered autonomous controller to complete each operative scenario defined in the evaluation design. Starting from the center, these bars represent: the execution time in (i) nominal execution, (ii) in dynamic goal injection and (iii) execution failures. At the bottom of the picture the execution times in seconds are also depicted.

Finally, the external ring in the chart presents the metrics scores. The smallest circumference (after the GS) of the ring represents the lower score for a metric, while the outside circumference is the best value ($\mu_i^S = 100$). Between both circumferences, dashed lines highlight the 25, 50 and 75 over 100 metrics scores. In each quadrant the metric scores are represented as a filled circular sector, that is proportionally wide to the metric weight. Then, better evaluations are those in which more ring partitions are filled.

This methodology constitutes a generic and reproducible process to evaluate autonomous controllers while considering varying execution scenarios. In this regard, the definition of metrics, weights and experimental cases, is the basic step on which we can rely to reproduce the evaluation results. However, a set of generally applicable metrics is required to achieve objective and reproducible evaluations.

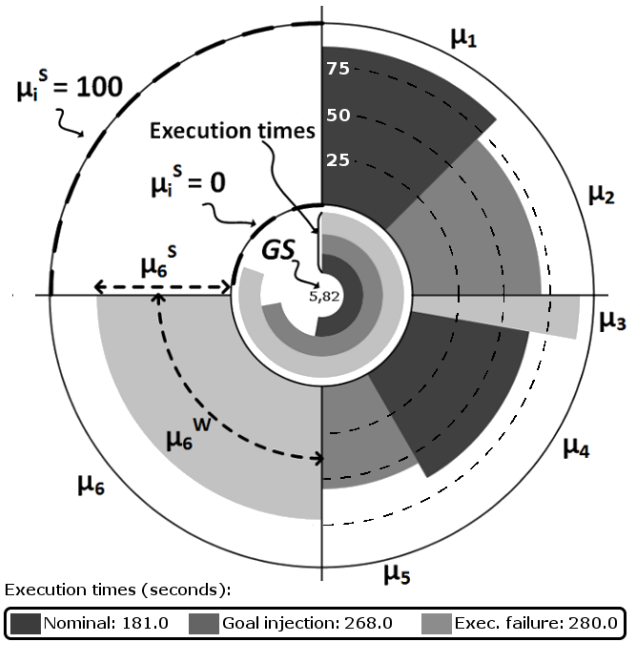


Figure 7.1: Controller assessment graphical report.

7.3 General metrics for autonomous controllers assessment

To provide autonomous controllers assessment, it is required to isolate general behaviours of autonomous controllers, and to formalize them in metrics. Then, using the data collected during execution, we can provide an objective evaluation.

Performing empirical evaluations is a common practice in research. Such evaluations allow characterizing and comparing different aspects of a system under study, by means of metrics that objectively describe the relevant performance features. Assessing classical planners is a well established practice [103], measuring parameters such as running time, memory usage or plan quality among others. However, the evaluation of P&S systems in real conditions is not yet widely considered. Integrating planners with different technologies to control a robotic platform (P&E) requires to evaluate general aspects and avoid domain dependent characteristics.

Analysing the execution of an autonomous controller, a plan guides the platform toward achieving the goals. Such plan is obtained by applying P&S techniques over a domain that represents the robot behaviours and the environment. Then, the performance will be greatly influenced by the deliberative and the representation used. Besides, the deliberative has to be connected to the functional layer to exploit the robot capabilities. This integration is usually done with an executive that is in charge of P&E integration. Then, how well these components are integrated, has to be assessed as well. For this reason, we have classified the metrics introduced following into four groups accordingly to the behaviours that they measure, as show in fig. 7.2:

Plan accuracy	**P&E integration**
Plan Time Accuracy$_{lb}$	Controller Processor Usage
Plan Time Accuracy$_{ub}$	Controller Memory Usage
Plan Effective Time	Controller Dispatching Time
	Controller Sensing Time
	Controller Monitoring Time
	Controller Reaction Time
Planner model adequacy	**Planner performance**
Command Time Discrepancy$_{lb}$	Planner Deliberation Time
Command Time Discrepancy$_{ub}$	Planner Deliberation Memory
Planner Model Analogy	Planner Deliberation Efficiency
	Planner Synchronization Ratio
	Planner Synchronization Frequency
Model $\leftarrow$	$\rightarrow$ ***Controller***

Figure 7.2: Clustering of the proposed metrics into four areas.

1. **Plan accuracy**. We want to evaluate the quality and accuracy of the plan when it is being executed. Assessing the plan generated will allow us to ensure that it is suitable for real applications. This evaluation requires to execute the plan; standalone test cannot isolate the uncertainty implicit in a robotic application. Then, to provide a general evaluation of the plan regarding to its execution, we introduce three metrics:

   - **Plan Time Accuracy Lower-Bound**: evaluates how well the minimal plan duration fits the execution time.
   - **Plan Time Accuracy Upper-Bound**: in a similar way that the previous metric, it assesses if the plan is time coherent with its execution, i.e., it finishes before reaching the maximum time planned.
   - **Plan Effective Time**: gives a measure of the time employed by the platform executing the plan, i.e., the time in which the platform is not idle or waiting for external events.

2. **Planner model adequacy**. Different domains can be used to solve the same scenario. In this way, we aim to analyse how well the actions defined in the domain fit the robot behaviours. As the plan generated is dependent of the domain employed, we need to check the correctness of the high level definitions of the robot behaviours. Assessing the domain will allow us to improve its adequacy. We have defined three metrics:

   - **Command Time Discrepancy Lower-Bound**: evaluates how accurate is the temporal definition of the modelled actions, by analysing the minimum duration defined in the model and facing it with the execution time.
   - **Command Time Discrepancy Upper-Bound**: measures the discrepancy between the time employed executing actions and their maximum defined time in the model, which enables (in addition to the previous metric) the evaluation of the temporal coherence of the model and the robot behaviour.
   - **Planner Model Analogy**: assesses if the controller has a closed loop between P&E, i.e., if each action carried out by the platform provides feedback to the planner to ensure correct action monitoring.

3. **Planner performance**. Usually, deliberation is an expensive task, so we want to characterize the planner performance. Moreover, different P&S systems can be used for the same domain. Then, to properly evaluate the planner performance we need to choose the best deliberative for our application. Moreover, it is also expected that the deliberative monitors the plan execution to ensure that the operative constraints remain safe. Then, how the planner monitors the plan execution is also object of study. We propose five metrics to assess the planner performance:

   - **Planner Deliberation Time**: evaluates how much time is required by the planner to generate the plan(s).

- **Planner Deliberation Memory**: assesses how much memory the planner uses when deliberating, facing to the required one during the plan execution.
- **Planner Deliberation Efficiency**: provides the efficiency of the generated plan by analysing domain independent features, such as the number of planned goals or the time required to achieve them regarding to the time required for the planning process.
- **Planner Synchronization Ratio**: measures how often the planer receives updates from the lower layers to keep its internal database updated.
- **Planner Synchronization Frequency**: evaluates how often the planner analyses the platform/environment status to check the coherence between the planned status and the current execution.

4. **P&E integration**. The plan execution is not usually done by the deliberative system. Typically, an executive entails plan execution by exploiting the functional support. In this regard, deliberative, executive and functional support must share information that rarely has a common representation. This entails translation processes between layers, but also synchronization so all layers work in a coordinated way. Then, evaluate the P&E integration will allow us to ensure that the different layers of the autonomous controller work properly. In this regard, to assess the performance of P&E, we introduce six metrics:

   - **Controller Processor Usage**: measures the processor usage of the controller.
   - **Controller Memory Usage**: measures the memory required by the controller during execution.
   - **Controller Dispatching Time**: evaluates the performance of P&E when dispatching commands, i.e., the effort required to translate and dispatch commands from the deliberative layer to the functional layer.
   - **Controller Sensing Time**: assesses the time required to translate raw data collected from the sensors to information that can be used by the other layers (e.g., deliberative model facts).
   - **Controller Monitoring Time**: provides a measure of the effort required by the controller to check the coherence between the planned status and the current status.
   - **Controller Reaction Time**: evaluates the capacity of the controller to react to unexpected events and external perturbations during the plan execution.

Furthermore, these four groups can be split in two classes as shown in fig. 7.2. By one side, *P&E integration* and *Planner performance* consider the assessment from the point of view of the *Controller*. That is, they evaluate the technologies used to implement the autonomous controller and how well they are coupled. Also, these metrics assess the performance of the controller over a particular computer. By the other side, *Planner model adequacy* and *Plan accuracy* are related to the correctness

of the *P&S model* employed, measuring the quality of the model when is applied to a real scenario. These metrics are independent of the hardware, but rely on the robotic platform capabilities, e.g., commands execution time.

All the metrics depicted in fig. 7.2 will be formally presented and exemplified in sec. 7.5 using the MoBAr autonomous controller as an study case when dealing with the planetary exploration scenario presented in the next section. As well, in sec. 7.6, we apply these metrics to the GOAC controller using the same exploration scenario.

7.4 The planetary exploration case study

From now on, we will present assessments of MoBAr and GOAC to formalise and exemplify the metrics. To do this, we exploit both controllers to solve the planetary exploration scenario. We have selected this scenario as some of the autonomous controllers referenced in sec. 2.1 exploit such domain for the experimental demonstration. The base of this scenario consists of achieving a set of targets (e.g., pictures or samples acquisition) in different locations defined by the human team in charge of the mission. After acquisition, the data gathered shall be communicated to a Ground Control Station (GCS) or a satellite that may not be visible for some periods. Although the scenario concept is quite easy, the implementation of an autonomous controller to deal with is technologically challenging. To achieve the mission goals, the robotic platform and the autonomous controller shall deal with navigation, perception and decision making, among other capabilities.

In order to enable reproducible results, we have to set some parameters, i.e., the operative rules and the robotic platform. The operative rules are typically defined in the P&S model and must be hold during the overall mission to maintain safe and effective configurations for the robot. For our scenario we have the following conditions:

C1: While the robot is moving the PTU must be pointing to the front.

C2: Pictures can only be taken if the robot is still in one of the requested locations while the PTU is pointing at the desired target.

C3: All acquired pictures shall be transmitted to the GCS.

C4: While communicating, the robot has to be still

C5: While communicating, the station has to be visible.

C6: We assume a flat terrain.

C7: There are no obstacles in the terrain.

We can exploit different robotics platforms to complete this scenario. By one side, GOAC was demonstrated using the DALA rover, which is one of the LAAS robotic platforms used for autonomous exploration experiments. By the other side, we have exploited the TurtleBot platform to test MoBAr.

DALA is an iRobot ATRV that provides a number of sensors and effectors. It can use vision based navigation (such as the one used by the MER rovers) by means of stereo cameras mounted on top of the PTU, as well as indoor navigation based on a Sick laser range finder. Its locomotion system is ready for uneven terrains and it has a communication facility. Then, DALA is able to autonomously navigate, take high-resolution pictures and communicate images to a GCS. However, we cannot exploit the robot for our experiments as it is property of LAAS and constructing a similar one is expensive. Then, we are only able to simulate it by means of the OpenPRS. This simulator provides the same behaviours as the real robot and enables customizing some of them (e.g., defining the duration of the different actions). Nevertheless, it does not have a Graphical User Interface (GUI) to show the execution in real time and it is not possible to modify the environment (e.g., we cannot place obstacles).

In the case of the TurtleBot we have a cheap and open-source robotic platform that can be easily customizable. In particular, we exploit the customization presented in sec. 6.6.2, so we have a robot that can be used for our scenario. In this sense, the TurtleBot provides good mobility in indoor environments, but it is not able to deal with uneven terrains; for this reason we have added the constraint C6. As well, the robot can perform autonomous navigation using Sample Localization And Mapping (SLAM) algorithms [49] with the data provided by the Kinect camera. However, for our tests, we do not enable such characteristic as the on-board computer is not powerful enough to exploit it without a significant latency. In this regard, we navigate using the information provided by the odometry sensors, assuming that we work on an open and free obstacle area (constraint C7). Instead, the TurtleBot is accessible in the GAZEBO simulation environment [140], that enables a good inspector panel of the robot status and its visualization in the environment (which is also customizable). Using an open-source platform will ease other researchers to recreate the experiments, either, with the real robot and/or with the GAZEBO simulator.

Then, in our experiments we use the simulated TurtleBot in GAZEBO. As a consequence, we have adjusted the GOAC functional support to work with the TurtleBot instead of DALA. For each execution, we consider the following data:

- The initial goal is to acquire two pictures in different locations. The acquisition of the picture implies: (i) move to a defined location; (ii) set the PTU; (iii) take the picture and; (iv) communicate the picture to the GCS.
- A third picture goal is injected during execution (at time 55 seconds).
- It is only possible to communicate the pictures after 50 seconds.
- In the domain, the actions have a defined maximum duration of: (i) 60 seconds for the movement action; (ii) 3 seconds for the PTU setup; (iii) 5 seconds for the picture acquisition and; (iv) 12 seconds to transmit the picture.
- The functional layer is executing in the robot mini-pc, while the other layers of the autonomous controller are running on a computer endorsed with a 2.5 GHz processor and 8 GB of RAM.

7.5 Formalising and applying the metrics to MoBAr

In this section we formally define the metrics introduced in sec. 7.3. To simplify the definition, we introduce an example to compute the scores. In this regard, we execute MoBAr with the TurtleBot robot for the exploration scenario presented in the previous section. Then, in addition to the initial data provided in the scenario definition, we obtain the following relevant values after the execution of MoBAr:

- The maximum predicted time to complete the three goals (or maximum planning horizon) is 228 seconds. This value is given by the MoBAr deliberative.
- The execution time to accomplish the scenario is 149 seconds. We call this value $\varepsilon_t = 149$.

For the experiments, we use MoBAr with the OPTIC planner as the deliberative layer instead of up2ta. The reason is that up2ta does not allow defining temporal behaviours for the actions, so the metrics related to the model will be very low.

The plans generated by the MoBAr deliberative are composed of fixed sequence of actions that are sequentially executed, i.e., it is not possible to temporary reallocate them during execution. Since the PDDL version used only manages one value for the actions durations, we have used it as the maximum duration. As a consequence, MoBAr is not able to provide a minimum duration for the actions neither a minimal end time for the generated plan.

It is worth mentioning that in MoBAr P&E are loosely coupled. The planner is only executed to generate the plan (i.e., at the beginning of the mission, when new goals are injected or when an action fails during execution), and then, the executive takes the control and executes and monitors the actions outcomes. Then, the only interaction between the planner and the executive is a component that provides access to the PDDL model and plan, while updating the problem with the information provided by the executive.

In the following subsections we formalize and exemplify the metrics starting from the *Plan accuracy* (top-left in fig. 7.2) and following the groups counter-clockwise. To follow the methodology each metric has to provide a score denoted by μ^S. To obtain such value, a formalization of the metrics is introduced, enabling to compute the score from the data gathered during the controller execution.

7.5.1 Plan accuracy

Various P&S systems can generate different solutions for the same domain. In this regard to assess the plan generated, classical planning performance measures (e.g., number of actions) are not generally applicable when dealing with uncertainty and dynamic environments. Then, as introduced in sec. 7.3, we evaluated these three metrics:

Plan Time Accuracy Lower-Bound – $\mathbf{PTA}^{lb}$. First, we want to evaluate if the plan is time coherent with its execution. In this way, the plan is temporary bounded

between a minimum and a maximum duration time. The objective of this metric is to measure the difference between the minimum plan duration (the lower plan horizon or plan horizon^{lb}) and the real execution time. A good plan will provide a plan horizon^{lb} that is smaller than the execution time; otherwise the planner or the model employed are not timely coherent with the real system. In such case is not possible to properly monitor the execution as the plan ends before expected. Then, if the execution time (ε_t) is lower than the plan horizon^{lb} the score for this metric is 0. On the contrary, the score is inversely proportional to the difference between the execution time and the plan horizon^{lb} as in eq. 7.3.

$$\frac{\varepsilon_t}{\text{plan horizon}^{lb}} \begin{cases} \leq 0 \Rightarrow \mu^S_{PTA^{lb}} = 0 \\ > 0 \Rightarrow \mu^S_{PTA^{lb}} = \frac{\text{plan horizon}^{lb}}{\varepsilon_t} \cdot 100 \end{cases} \tag{7.3}$$

The planner in MoBAr does not provide a lower planning horizon, as the actions are only temporary defined in their maximum duration. Thus, the metric score for the PTA^{lb} is 0. If we analyse the execution, it ends in 149 seconds. However, it can be faster. Due to the physical restrictions of the robot, the execution should be at least 109 seconds (since the robot cannot, for instance, move faster than the maximum velocity). Let's consider an execution in which a sensor provides bad information (e.g., the odometry returns higher distances than the real travelled), and the execution is completed in 90 seconds (which is physically not possible). Such issue cannot be detected if the planner does not provide a minimum plan duration, but the controller *believes* that the goal have been completed, which is actually false. For instance, in the case the pictures are taken in wrong places.

Plan Time Accuracy Upper-Bound – PTA^{ub}. Following the temporal correctness of the plan, we can also evaluate the maximum duration predicted by the planner. In this direction, the controller must provide a maximum plan duration (plan horizon^{ub}) that shall be bigger than the execution time. So, the plan has to finish its execution before reaching the plan horizon^{ub} time. Everything that happens after such time is out of the temporal scope of the planner, and thus, is not properly controlled. The score for this metric is 0 when the execution time is bigger than the plan horizon^{ub}. Otherwise, the score is the coefficient between the execution time and the plan horizon^{ub}, as in eq. 7.4.

$$\frac{\text{plan horizon}^{ub}}{\varepsilon_t} \begin{cases} \leq 0 \Rightarrow \mu^S_{PTA^{ub}} = 0 \\ > 0 \Rightarrow \mu^S_{PTA^{ub}} = \frac{\varepsilon_t}{\text{plan horizon}^{ub}} \cdot 100 \end{cases} \tag{7.4}$$

MoBAr's deliberative provides a maximum duration for the plan, so we can acknowledge that something goes wrong if the plan execution time becomes larger than the given planning horizon. For the initial two goals, the planning horizon is 196 seconds. Then, when the new goal is injected, MoBAr generates a new plan (discarding the previous one) to achieve the pending goals. This last plan has a (maximum) duration of 228 seconds. Then, the PTA^{ub} score is 65.35 over 100. This means that the maximum planning horizon is not very accurate, so the plan does not well fit the robot behaviours.

Plan Effective Time – PET. The plan has to take into consideration uncontrollable events that affect its performance. For instance, an action may be only applicable when a specific event occurs (e.g., communicate only when a satellite is visible). Thus, a good plan is that in which the actions are allocated having in mind such events, avoiding idle times. The objective of this metric is to measure how much time has been used performing actions to achieve the mission goals. To compute this metric we face the time spent executing actions (called cmd exec time) with the execution time. As well, the time generating the plan (deliberation time) is considered as proactive time, so it is deducted from the execution time as in eq. 7.5.

$$\mu^{S}_{PET} = \frac{100 \cdot \text{cmd exec time}}{\varepsilon_t - \text{deliberation time}} \tag{7.5}$$

If we analyse the data gathered during the MoBAr execution, we observe that we need 149 seconds to execute the plan, but only 144 seconds (cmd exec time) are used executing actions (i.e., moving, setting the PTU, taking pictures or communication to ground). As well, the planning time is less than 1 second (0.3 seconds in total), so for near 4 seconds the platform is idle since the communication can only start at time 50. In this scenario, MoBAr uses most of the time on the mission objectives, but it is relevant to analyse why the platform is idle for near 4 seconds. In fig. 7.3 we show the first 60 seconds of the execution[1]. We can find that the platform is idle before the transmission of the first picture. Analysing the plan, the planned movement action (`GoingTo`) duration is 60 seconds, but the robot takes 36 seconds in executing it. This discrepancy between the planned time and the execution time is relevant when dealing with synchronous tasks or external events, as we cannot continue executing the plan, in this case, until the communication opportunity. It is possible to overcome such issue by adjusting the plan during execution or using a planner that better deals with temporal constraints. However, as the planner in MoBAr is not well suited for temporal constraints neither supervises the execution, the plan is not modified to perform other actions, e.g., repositioning the PTU for the next movement. Then, the PET score is 96.84.

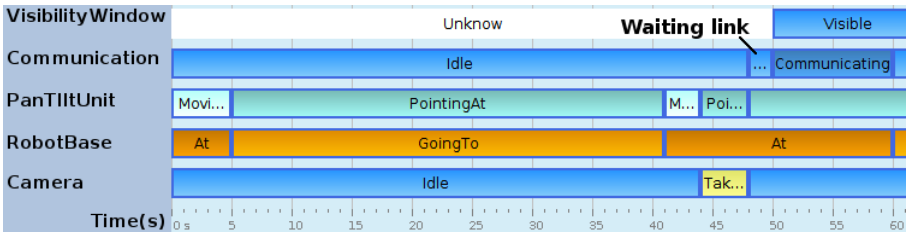


Figure 7.3: Execution for MoBAr, remarking the idle time waiting for the communication opportunity.

[1]MOBAR's planner is based on predicates. However, we use a timelines representation to show the execution. Each timeline represents the state of a particular subsystem over time.

7.5.2 Planner model adequacy

The previous metrics are dependent of both, the planner used and the domain employed. Furthermore, using different domains, the same planner can generate different solutions. The previous metrics assess the plan at a coarse granularity. Now, we are focused on evaluating the model at a fine granularity, i.e., we want to assess how the planner model fits the actual behaviours of the robotic platform. The definition of the three metrics is as follows:

Command Time Discrepancy Lower-Bound – CTD^{lb}. As we did with the plan, we want to evaluate if the actions defined in the domain have a correct temporal definition. The objective is to characterize the quality of the deliberative model employed in terms of how well it fits the robotic system during execution. In this regard, an important aspect to be covered is to analyse the difference between the time planned for a command and its execution time. In general, we need to properly model the actions duration so the planner can provide feasible plans. Actions that require less time to execute (we call this value cmd time$_{executed}$) than the minimum planned time (cmd time$_{min\ planned}$) entails that the model does not fit the execution. This metric provides a fidelity measure of the time planned for the commands and its real execution time. To compute this metric, for each command i we define CTD_i^{lb} as the difference between the execution time of a command and the minimum planned time for that command. In the case that the CTD_i^{lb} is negative, means that either the planner did not use a good model for the actions duration or there is a specific problem executing the command. In that case the CTD_i^{lb} is penalized multiplying its value by a factor δ. This factor is set by the user with values starting at 1 (i.e., no penalization), and increasing proportionally to the penalization desired. In our case we set δ to 2 (i.e., negative CTD_i^{lb} has double penalization in the evaluation than positive CTD_i^{lb}). The metric score is computed as the total time discrepancy for all actions (i.e., the sum of all CTD_i^{lb}) divided by the execution time as in eq. 7.6.

$$\begin{aligned} CTD_i^{lb} &= \text{cmd time}_{executed} - \text{cmd time}_{min\ planned} \\ \text{if } CTD_i^{lb} < 0 &\Rightarrow CTD_i^{lb} = CTD_i^{lb} \cdot (-\delta) \\ \mu^S_{CTD^{lb}} &= 100 - \frac{\sum_{i=0}^{n} CTD_i^{lb}}{\varepsilon_t} \cdot 100 \end{aligned} \tag{7.6}$$

As we stated before, MoBAr does not provide the minimum action duration, which has negative consequences for the execution monitoring. For instance, consider that there is a failure in the robot odometry and it gives higher values for the distance travelled than the real ones. If we properly model the time for the movement, the planner can supervise the temporal behaviour and determine that the movements end to fast, overcoming the robot physical constraints. Then, it is possible to determine that there is a failure. However, without such supervision, the planner receives a position that is not the true one, failing to complete the desired objectives, e.g., a picture acquisition in a determined location. For this reason, the score for this metric is 0.

Command Time Discrepancy Upper-Bound – CTD^{ub}. In the same way that we also measure the discrepancy between the minimum action duration, we measure the difference between the time employed executing a command and its maximum planned time. Actions that require more time than the planned time (cmd time$_{max\ planned}$) are out of its temporal scope, and thus, are out of the planner control. The metric is computed as the previous one, but defining the CTD_i^{ub} as the difference between the maximum planned time and the execution time. As well, we apply the same penalty factor of the previous metric for negative CTD$_i^{ub}$. Then, the score is computed as in eq. 7.7.

$$\begin{aligned} CTD_i^{ub} &= \text{cmd time}_{max\ planned} - \text{cmd time}_{executed} \\ &\text{if } CTD_i^{ub} < 0 \Rightarrow CTD_i^{ub} = CTD_i^{ub} \cdot (-\delta) \\ \mu^S_{CTD^{ub}} &= 100 - \frac{\sum_{i=0}^{n} CTD_i^{ub}}{\varepsilon_t} \cdot 100 \end{aligned} \tag{7.7}$$

In MoBAr the actions have a fixed duration defined in the domain. Then, during the execution we obtain a CTD$_i^{ub}$ for each executed action. We can depict these values in a chart generating the temporal profile presented in fig. 7.4, so the peaks of the chart provides the CTD$_i^{ub}$ value for each action. We can see that the first action has a negative CTD$_i^{ub}$, which is penalized in the evaluation. In this case the first repositioning of the PTU requires more time that the planned (the maximum planned time is 3 seconds and the execution is near 5.5 seconds). The reason for this behaviour is a problem related to the initialization of the PTU module at the executive level (we only observe this anomaly for the first PTU setup, the others PTU movements requires less time than the planned one). As well, the higher peaks correspond to the movement action. The total time discrepancy measured is 113 seconds, obtaining a score of 24.16 over 100. This means that, in general, the actions model do not fit well the robot behaviour.

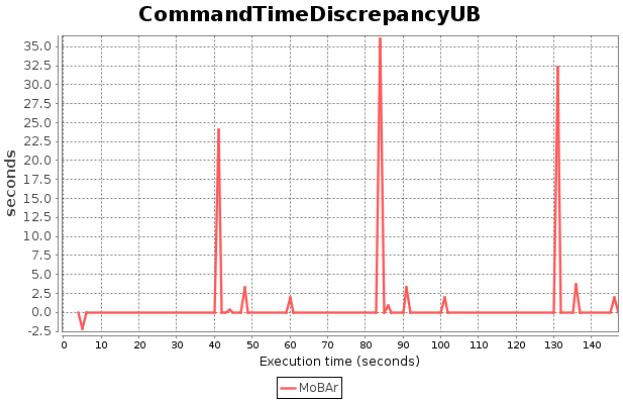


Figure 7.4: Temporal profile of CTD$_i^{ub}$ for MoBAr.

Planner Model Analogy – PMA. In a robotic application, the deliberative model can be defined with a variable number of actions. This means that the model can be more coupled with the functional layer (more lower level actions) or more abstracted

(higher level, typically less actions). For instance, considering the robot movement, a high level action can define the movement to a waypoint. On the contrary, it is also possible to define two movements, rotation and forward. In any case, it is required that each action provides feedback to the planner after execution. So, the objective of this metric is to analyse the similarity of the planner model with the real environment and platform, evaluating the feedback provided by the actions execution. It is desirable that, at least, each command produce an update in the P&S system, so the controller has a closed loop between P&E. This metric is computed facing the number of times that the planner receives feedback to keep the knowledge database up-to-date (planner states updates) with the number of commands executed. The score is computed as in eq. 7.8. When the score is higher than 100, it is normalized to 100.

$$\mu^{S}_{PMA} = \frac{\text{planner states updates}}{\text{commands executed}} \cdot 100 \tag{7.8}$$

In MoBAr the result of each action is defined in the PDDL problem to keep the platform state up-to-date. Then, if a replanning is required, the problem reflects the current state and achieved goals. In this regard, 33 planner updates are produced during the execution (which the temporal distribution of fig. 7.5), for 13 executed actions. This implies that each action has a reflect in the planner knowledge base. Then the metric score is 100. This implies that, in MoBAr, P&E works in a closed loop.

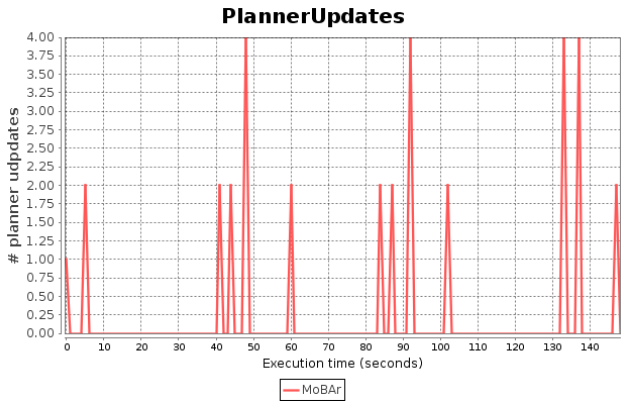


Figure 7.5: Temporal profile of the planner updates in MoBAr.

7.5.3 Planning performance

The generation of the plan that guides the execution is usually an expensive task. Then, it is relevant to measure the performance of the planning process. Classical measures used in the planning community such as the time employed and memory required during deliberation can be considered. Moreover, in a robotic application other aspects must be taken into account. In this direction, the planner has not only to generate the plan, but also to monitor its execution, maintaining the safety

constraints defined in the domain. Next, the five metrics to assess the planner performance are presented:

Planner Deliberation Time – PDT. We can apply a classical measure in the planning community: the time spent generating the plan. In this regard, it is desirable to generate a plan as fast as possible, as the environment can change invalidating the plan when the planning process takes excessive time. To compute the score for this metric, the deliberation time is faced with the execution time as in eq. 7.9. This gives better scores for lower deliberation times with longer execution times.

$$\mu^{S}_{PDT} = 100 - \frac{100 \cdot \text{deliberation time}}{\varepsilon_t} \tag{7.9}$$

For our example, MoBAr has a temporal profile for the deliberation time such as the depicted in fig. 7.6. This profile shows two peaks when MoBAr is deliberating. The first planning process (for the initial two goals) requires near 0.215 seconds. Then, at time 55 seconds a third goal is injected. If we observe the plan execution (see again fig. 7.3), we can see that at time 60 the first goal (i.e., taking and communicating a picture) is completed. At this moment, the third goal is planned, requiring near 0.085 seconds to generate the plan for the injected goal and the pending second goal. Then, with a total deliberation time of 0.3 seconds, the PDT score is 99.80 over 100. In this sense, the planner is fast and does not require a long deliberation time respect to the execution time.

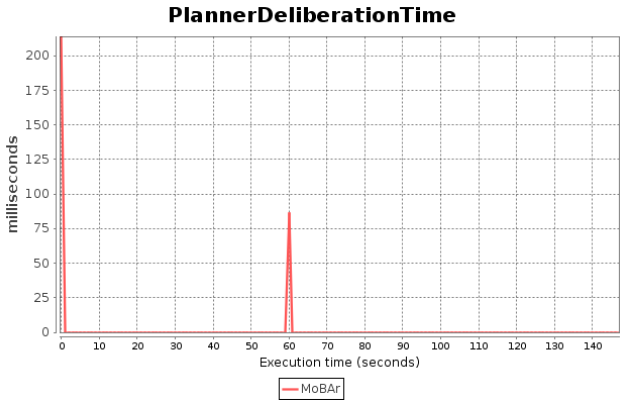


Figure 7.6: Temporal profile of the deliberation time in MoBAr.

Planner Deliberation Memory – PDM. During the plan generation, it is usual that the deliberative requires some extra memory. In this regard, it is not desirable that the planner allocates an excessive amount when planning. That is, we penalize planners that does not have a constant memory usage (which is desirable for real time systems). To characterize this point, this metric provides a score facing the average memory and the maximum memory used when planning, as in eq. 7.10.

MoBAr has a low memory requirement, except when the deliberative is generating a plan. In this sense, the temporal profile for the memory usage (for both the

deliberative and the executive) is shown in fig. 7.7. We can see a peak at the beginning, when the deliberative is generating the first plan. In that point, the memory required is near 0.1%. Instead, during execution the planner is not running. Then, the 0.00067% of memory used during the rest of the execution is employed by the executive. This means that the planner greatly increases the memory required when planning, and thus, the metric score is 0.67. We obtain such low score as this excessive allocation of memory for the planning process is undesirable. In this regard, it could impose restrictions in the design, e.g., forcing the system to reserve memory for the planning process or to perform memory reallocation during execution.

$$\mu^S_{PDM} = 100 \cdot \frac{\text{average memory used}}{\text{max memory used}} \tag{7.10}$$

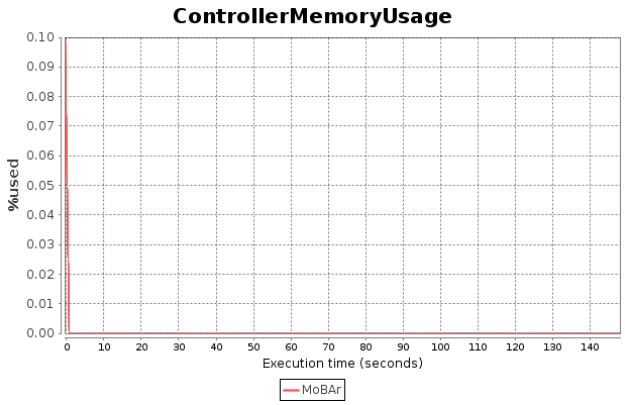


Figure 7.7: Temporal profile of the memory usage in MoBAr.

Planner Deliberation Efficiency – PDE. We can measure the efficiency of the planning process by analysing the generated plan, e.g., assessing the plan length respect to the optimal solution. However, we need to address the efficiency of the planner from a general perspective, avoiding domain dependent features. For this reason, we evaluate the deliberation efficiency in terms of the time employed in deliberating, the number of planned goals and the time for executing the plan (i.e., the time executing actions, defined previously as cmd exec time). In general, it is desirable to plan as much goals as possible in less time. Then, the score is computed as in eq. 7.11.

$$\mu^S_{PDE} = 100 - \frac{100 \cdot \text{deliberation time}}{\text{cmd exec time} \cdot \text{number of goals}} \tag{7.11}$$

For our example, MoBAr requires 0.3 seconds to generate the plans to achieve the three goals, and 145 seconds to execute the actions contained in the plan. Then, the PDE score is 99.93 over 100. This means that MoBAr requires much more time to execute the plan than the required to generate it.

Planner Synchronization Ratio – PSR. To maintain the safety constraints, it is usual that the planner monitors the plan execution. To do this, the planner has to keep its internal knowledge base up-to-date by receiving updated information from the lower layers (e.g., the robot position). In this direction the planner can receive updates each time that a parameter changes or with a defined frequency (depending of the implementation). Comparing these updates with the expected states, the planner effectively monitors the plan execution. The frequency of the updates must be enough to capture all changes of the monitored parameters, which is dependent of the environment/platform. Then, the objective of this metric is to link the number of updates received by the planner regarding to the scenario frequency (set by the user). The score for this metric is computed as in eq. 7.12.

$$\mu^S_{PSR} = \frac{\text{planner updates}}{\varepsilon_t} \cdot \frac{100}{\text{frequency}} \tag{7.12}$$

For the PSR metric we need to define the update frequency for our system. In this regard, we will use a frequency of 1 second. This means that we acknowledge any change in the environment/platform within 1 second. However, the robot position in MoBAr is only updated at the end of the movement action. Thus, the planner does not receive information with the current position during the movement, so it is not possible to trace the action execution at a fine granularity. This also happens with the other actions. So, in MoBAr we have that the temporal profile for the planner updates is the one previously presented in fig. 7.5, showing that 33 planner updates are generated. Then, the score for this metric is 22.15. This means that the MoBAr deliberative database is not frequently updated, which is a requirement to properly monitor the plan execution. For instance, monitoring the action movement each second allows detecting problems early during execution (e.g., if the robot is moving significantly slowly respect to its nominal speed), not only at the end of the action.

Planner Synchronization Frequency – PSF. The planner can be updated by receiving information from the lower layers after a command execution. However, it is useful that the planner has a periodic task to monitor the plan execution, analysing what is currently in execution. This task enables the planner to halt the system if an unexpected or unsafe state is detected. Then, this metric measures how often the planner analyses the system status to check the coherence between the planned status and the current execution. As well, the frequency defined in the previous metric is applicable to this one. To compute this metric is required to obtain how often the planner compares the current status and the planned one. We call this value planner synchronizations, and it can be obtained evaluating the temporal distribution of the planner updates, or using the CMT metric (as explained later). Then, we compute the PSF score comparing the planner synchronizations with the total execution time and the desired frequency.

$$\mu^S_{PSF} = \frac{\text{planner synchronizations}}{\varepsilon_t} \cdot \frac{100}{\text{frequency}} \tag{7.13}$$

As we stated before, in MoBAr the planner updates came only from the execution of an action, i.e., the planner is not actively supervising the execution. If we look again at fig. 7.5, each peak represents one planner synchronization (if the action outcome does not correspond to the expected one, the planner generates a new plan). Then, there are 13 planner synchronizations, which means that the planner is updated, in average, every 11.46 seconds, which is far from our 1 second desired frequency. Then, the PSF score is 8.72. MoBAr only supervises changes in the environment/platform at the end of a command execution. Thus, any change that occurs during execution may not be registered, which could lead to miss opportunistic objectives or environmental changes that affect the execution.

7.5.4 P&E integration

The integration between the deliberative and the other layers has a major impact in the overall performance of a plan-based controller. Typically, each layer of the architecture has a different temporal scope and abstraction to express the platform and environment status. While the planner manages high level data with a long term memory, the executor is usually more coupled with the functional level, with a short term memory to ensure current action monitoring. Then, assessing how the information is synchronized among layers is relevant. Also, identifying the computational requirements for the controller allows determining the processor and memory capacity for the robotic platform. The six metrics in this group are next explained:

Controller Processor Usage – CPU. As any other software, the computational resources usage is a relevant performance measure. Then, measuring the processor usage during execution will enable to determine if its capacity is enough to support the controller. To compute this metric, we consider the processor usage each second, CPU_i. This value is the quotient between the cpu time and the current execution time. Then, eq. 7.14 provides the score for this metric using the average processor usage.

$$CPU_i = \frac{\text{cpu time}}{\text{current execution time}} \cdot 100$$
$$\mu^S_{CPU} = 100 - \frac{\sum_{i=0}^{\varepsilon_t} CPU_i}{\varepsilon_t} \tag{7.14}$$

The temporal profile (which shows the combined CPU_i values for the deliberative and the executive) generated by the MoBAr execution for our scenario example is presented in fig. 7.8. From such data we can obtain that, in average, MoBAr uses 0.025% of the processor capacity. As well, there are visible the two planning processes, which require near 0.16% and 0.1% of the processor capacity. Then, the CPU score is 99.98 over 100. This means that the planner and executive are lightweight in terms of processor requirements, being suitable for lower resource computers than the actual exploited for our test.

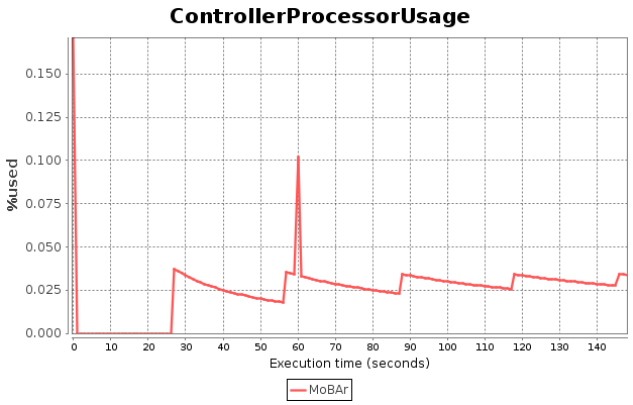


Figure 7.8: Temporal profile of the processor usage (CPU_i) in MoBAr.

Controller Memory Usage – CMU. Another relevant resource to measure is the amount of memory required by the controller during execution. We can obtain the percentage of memory used at any moment, CMU_i, facing the resident memory of the controller and the total memory of the platform. This enables to provide a temporal characterization of the memory usage. Then, we can compute the metric score using the average memory required as in eq. 7.15, considering that we obtain a memory measure each second.

$$
\begin{aligned}
CMU_i &= \frac{\text{resident memory}}{\text{total memory}} \cdot 100 \\
\mu^S_{CMU} &= 100 - \frac{\sum_{i=0}^{\varepsilon_t} CMU_i}{\varepsilon_t}
\end{aligned}
\tag{7.15}
$$

MoBAr requires a small amount of memory as can be seen in the temporal profile in fig. 7.7. In average, the controller requires 0.007% of the available memory, being the biggest usage corresponding to the initial planning. Thus, the score for this metric is 99.99. In this sense, MoBAr requires a small amount of memory, so it is possible to execute the controller in machines with much less memory.

Controller Dispatching Time – CDT. The plan generated provides high level actions that (commonly) are not executable by the platform. Thus, the actions have to be translated, and sometimes, decomposed into simpler commands to be executed. This is usually done by the executive. The objective of this metric is to measure the effort required to translate the plan into functional level commands. In this regard, this metric provides a measure of the performance of P&E when dispatching commands. To do this, we measure the time elapsed to translate and dispatch each planned action. The metric score is obtained considering the time spent for dispatching all the commands (dispatching time) and facing it with the total execution time for the mission, as in eq. 7.16.

$$
\mu^S_{CDT} = 100 - \frac{100 \cdot \text{dispatching time}}{\varepsilon_t} \tag{7.16}
$$

The dispatching time in MoBAr has a temporal profile such as the shown in fig. 7.9. In that profile we can observe that, during the action execution, the time is 0, while before the command execution there is a peak. Such peak represents the time required to (i) translate the action information (from PDDL objects to numerical parameters) and (ii), to select the corresponding behaviour in the PLEXIL plan to execute the action. In total, the dispatching time is 0.968 seconds, which gives a score of 99.35. In this regard, the time employed by MoBAr dispatching the commands to execution is negligible respect to the execution time.

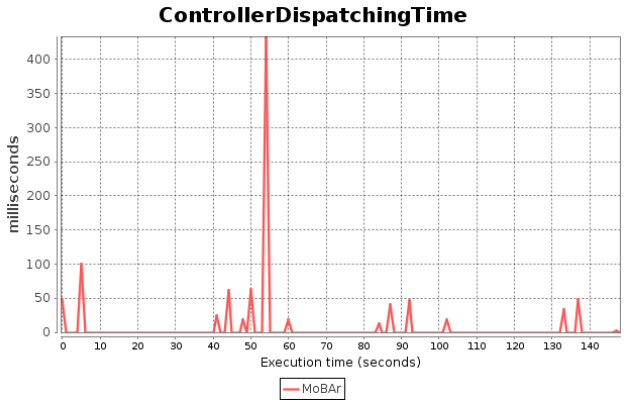


Figure 7.9: Temporal profile for the dispatching time in MoBAr.

Controller Sensing Time – CST. As happen with the commands, the sensor data has to be translated to be used by the different layers. In this sense, it is relevant to characterize the time required to deliver each sensor data to the other layers of the controller. This enables to evaluate how well the different layers share information. For instance, controllers in which P&E have a common abstraction will require less effort. The score for this metric is computed as in eq. 7.17, considering that the sensing time is the sum of the time employed to deliver all sensor data to the other layers.

$$\mu^{S}_{CST} = 100 - \frac{100 \cdot \text{sensing time}}{\varepsilon_t} \tag{7.17}$$

As in the case of the CDT, MoBAr only performs data gathering at the end of a command execution. Then, the temporal profile for the sensing time is the one presented in fig. 7.10. It shows that the sensing time during actions execution is 0. Then, the peaks provide a measure of the effort required to translate the sensor data into PDDL information, being in total 0.007 seconds. Then, the CST metric has a score of 99.99. This means that sharing the platform/environment data collected by the sensors to the other layers is a fast process.

Controller Monitoring Time – CMT. Typically, the planner receives updates after a command execution. However, environmental changes can affect the plan execution. For this reason, to keep the safety constraints defined in the model, the

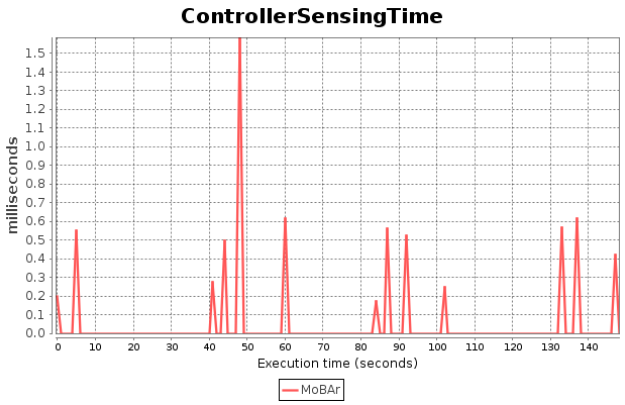


Figure 7.10: Temporal profile of the sensing time in MoBAr.

planner has to monitor the environment and platform status. Thus, a periodical task to check coherence between the execution and the plan is encouraged. Previously, we provide a performance measure based on the frequency of this task (see PSF). The objective of this metric is to assess the performance of such a task based on the time that it requires, which includes gathering sensor information and coherence checking. To compute the score we face the total time employed monitoring the plan execution and the execution time, as in eq. 7.18.

$$\mu^{S}_{CMT} = 100 - \frac{100 \cdot \text{monitoring time}}{\varepsilon_t} \tag{7.18}$$

In MoBAr there is no permanent supervision of the execution. Both the planner and the executive are not monitoring how the commands are being executed. For this reason, the score for this metric is 0. Then, it is not possible to detect problems during actions execution, being only possible at the end of the execution.

Controller Reaction Time – CRT. During the execution, external agents can produce new goals or failures may arise. Such unexpected events can broke the plan execution, requiring that the planner provides a new plan (replan) or repairs the current one to take into consideration the event. In this regard, it is possible to replan after the event detection or do it later. Nevertheless, delaying the plan generation could not be a good strategy: an execution failure has to be considered as soon as possible to avoid hazardous situations, while replanning new goals can be more efficient than executing them later (the new goal could be close to the current execution context). Then, this metric has the objective of measuring the capacity of the controller to react to unexpected events and external perturbations. Its score is determined by the time elapsed between an unexpected event is triggered and the time in which the planner starts the planning process to consider it. That is, for each event we define CRT_i as the time difference between an event i occurs at the subsequent planning process. Accumulating all CRT_i and comparing them to the total execution time, we obtain the metric score as in eq. 7.19. In case that there are no unexpected events, this metric has a score of 100.

$$CRT_i = \text{time}_{deliberation\ starts} - \text{time}_{event}$$
$$\mu^S_{CRT} = 100 - \frac{\sum_{i=0}^{n} CRT_i}{\varepsilon_t} \cdot 100 \tag{7.19}$$

If we look again at the deliberation temporal profile (see fig. 7.6), we can see that the injected goal is planned nearly at the same time that is added: the goal is injected at 55 seconds and the replanning process starts at time 60 seconds. We have that CRT_1 is 60-55 = 5 seconds, so, the score for this metric is 96.64 over 100. Particularly, MoBAr performs the replanning process after the current command in execution ends, i.e., it is not possible to interrupt commands execution. This high score means that MoBAr can rapidly take advantage of injected goals to adapt the plan to achieve to, for example, opportunistic goals.

7.5.5 MoBAr Assessment

Considering these metrics we can provide an assessment of any controller that enables to characterize relevant aspects of the P&S system and the model used, but also about the P&E integration. For our example scenario we can summarize the evaluation for MoBAr in the fig. 7.11, being the *GS* 5.61 over 10. P&E is well integrated in the controller, requiring low computation resources and showing a good performance in the layers interconnections. However, as there is no supervision of the plan execution, the score for the monitoring time is 0. Looking at the planner performance, we can see that the deliberation time and efficiency provides good scores. Notwithstanding, the memory allocated during deliberation is notoriously higher than during the execution, which negatively affects the evaluation. Moreover, the planner synchronization ratio and frequency have very low scores as there is no permanent supervision of the plan execution. For instance, commands that continuously change a parameter during execution (e.g., the robot position during a movement) are only covered at the end of the action.

Focusing on the model, its accuracy could be greatly improved. First, the modelling language does not support the definition of temporal intervals for the actions, only a single value for the duration is possible (which is used for the maximum duration). Thus, it is not possible to ensure the temporal correctness of the plan during execution. Furthermore, the maximum action duration is not very accurate for the movement action, which degrades more the evaluation. Finally, the planner does not provide a minimum plan duration and there is a significant difference between the actual execution time and the maximum duration planned. However, the platform spends most of the time executing the plan or planning, so the platform is rarely idle. In general terms, the strengths of MoBAr are the performance of the planning process and the low resource usage, meanwhile its weakness are more related to the model used and the plan monitoring.

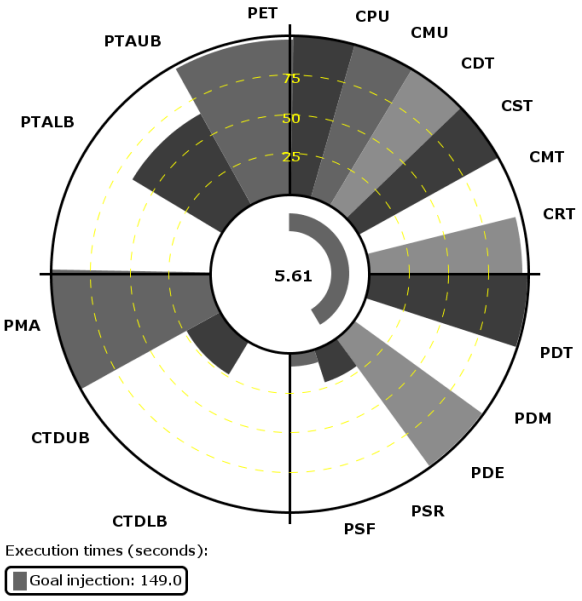


Figure 7.11: Summary report for the MoBAr example.

7.6 Applying the metrics to GOAC

In the previous sections we have defined a set of metrics for autonomous controller assessment and we have applied them for an execution of the MoBAr controller. Notwithstanding, we want to analyse if we can use such metrics to assess other controllers with different technologies for P&S and P&E. Then, in this section, we apply the metrics to the GOAC controller (see sec. 2.1.4).

For the experiments (and for the assessment done later in sec. 7.8) we exploit a GOAC instance with two reactors as shown in fig. 7.12: a *Deliberative reactor* using an APSI-TRF timeline-based planner and a *Command dispatcher* reactor. The *Deliberative reactor* is responsible of performing the deliberative tasks given a domain and a problem. The *Command dispatcher* reactor is in charge of executing commands and collecting execution feedback, exploiting the TurtleBot robot.

In the deliberative layer, the domain expresses the possible transitions for a set of timelines that represent the robot subsystems (robot position, camera, PTU

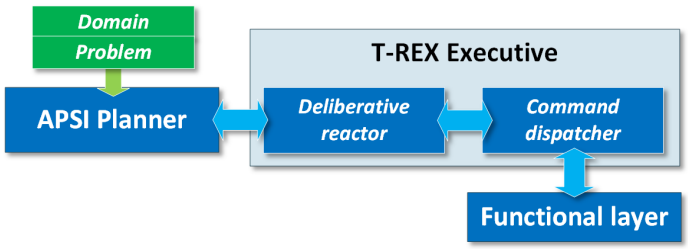


Figure 7.12: GOAC instance used.

and communications) and the environment (communication opportunities with the GCS). The transitions specify how and when the status of a timeline vary. Each state has its temporal interval, e.g., for a movement of the PTU we can define the minimum and maximum time required by the robot performing the action. As well, synchronizations between timelines shall occur to keep the mission constraints, e.g., the robot can move only when the PTU is pointing to the front.

Then, given a domain and a set of goals, the P&S system can generate a plan to guide the execution considering the allowed transitions and synchronizations among timelines. The plan corresponds to an assignment of the required states for each timeline. In the case of GOAC, the planner allows a flexible temporal allocation of tasks for a given planning horizon. The horizon represents the maximum time allowed to achieve all goals and it is given by the user. The first 60 seconds of the plan execution can be seen in fig. 7.14. In the figure we can see that, for the first 10 seconds, the system is still in the initial state. That time slot is dedicated for the planning process, which is set by the user.

Regarding to the P&E, in GOAC some of the timelines states correspond to actions that can be executed by the robotic platform (e.g., `GoingTo`, `TakingPicture`). These states are translated into functional commands by the *Command Dispatcher* reactor and executed through the functional layer. As well, this reactor is in charge of collecting and translating the sensors information. During the execution P&E are interleaved. This means that the planner is permanently observing the environment and robot status. This allows detecting execution flaws in a short time, allowing to exploit reactive planning schemes.

Then, to test the metrics we perform an assessment of GOAC for the same scenario of the previous section, and defined in sec. 7.4. Additionally to the information given in the scenario description, GOAC provides the following data for the execution:

- In the domain, the actions have a defined minimum duration of: (i) 20 seconds for the movement action; (ii) 1 second for the PTU setup; (iii) 1 second for the picture acquisition and; (iv) 8 seconds to transmit the picture.
- The GOAC maximum planning horizon is set by the user to 300 seconds.
- The GOAC planning slot is set by the user to 10 seconds.
- The execution time to accomplish the scenario is 196 seconds, i.e., $\varepsilon_t = 196$.

Next, each set of metrics are evaluated.

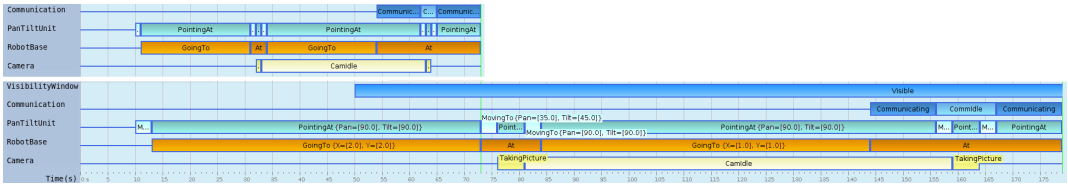


Figure 7.13: Plan generated by GOAC for the two initial goals. Considering minimum duration (top) and maximum duration (bottom) for all actions.

7.6.1 Plan accuracy

When GOAC generates a plan, the solution is temporary bounded. In fig. 7.13 the plan generated for the two initial goals is presented. The plan on the top shows the most *time optimistic* plan, in which each action has the smaller duration. Instead, fig. 7.13 bottom shows the plan considering the longest duration of each action. We can see that the minimum duration for the plan is 73 seconds, while the maximum is 179 seconds. However, the plan is flexible till a maximum time set by the user (in our example, 300 seconds). Then, GOAC is allowed to reallocate the starting time of the actions, so the plan can long till 300 seconds. In any case, the plan execution shall finish between the given interval, otherwise the plan generated and the execution are not time coherent. In such undesirable case, the model or the planner have flaws that lead to potentially uncontrolled situations. The score of the metrics in this group are next calculated:

Plan Time Accuracy Lower-Bound – PTA^{lb}. For our example, GOAC provides a plan with a minimum duration of 73 seconds for the first two goals plans. When these goals are completed, GOAC generates a new plan to achieve the injected goal, which has a minimum duration of 164 seconds. That is, the mission shall long more than 164 seconds to achieve all goals. In fact, the execution ends at time 196 seconds. Then, the metric score is 83.67 over 100. In this sense, GOAC is quite accurate predicting the minimum duration of the plan.

Plan Time Accuracy Upper-Bound – PTA^{ub}. For our scenario, GOAC planned a maximum execution time of 300 seconds and the real execution time is 196, obtaining a score of 65.33 over 100. This means that, the longer end time of the plan is significantly different to the real execution. However, this value is set by the user and can be better adjusted.

Plan Effective Time – PET. In our example, the total time employed executing actions (we called this value cmd exec time) is 150 seconds, and the total deliberation time is 0.341 seconds, providing a score of 77.88. GOAC obtains this score due to the planning slot and the actions delays (as shown in fig. 7.14). In this example, the planning slot is excessively long for the deliberation requirement, i.e., we have a total time of 20 seconds for deliberating and only 3.4 seconds are used, being the last 16.6 seconds unused. Notwithstanding, this value is set by the user and can be better set to avoid idle time. In the case of the actions delays, we have that some time is wasted due to the communication between the different reactors when notifying/receiving the updated status of the shared timelines.

7.6.2 Planner model adequacy

GOAC provides a modelling language that enables to provide temporary bounded actions as shown in fig. 7.15 for a movement action. This means that the planner can discriminate actions failures for premature or excessive actions duration. The values of the metrics in GOAC are next evaluated:

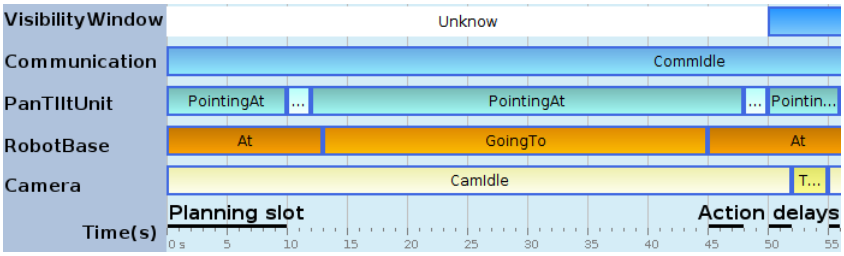


Figure 7.14: GOAC execution showing the idle time of the robotic platform due to the planning slot and delays in actions execution.

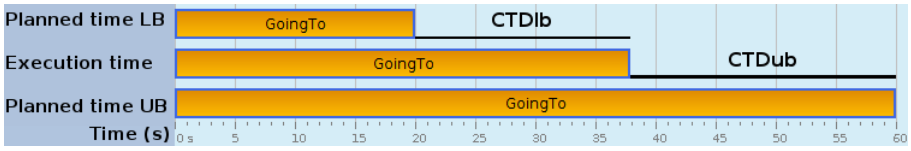


Figure 7.15: Times difference for an action planned time (lower and upper bounds) and execution time.

Command Time Discrepancy Lower-Bound – CTD^{lb}. Figure 7.16 provides a temporal evolution of the CTD$_i^{lb}$ values for GOAC, that are represented as peaks in the chart. The total time discrepancy between the minimum action duration and the real action duration is 57 seconds, obtaining a metric score of 70.91. In this sense, we can see that the minimum action duration is quite accurate with the exception of the movement action, that provides significantly higher differences.

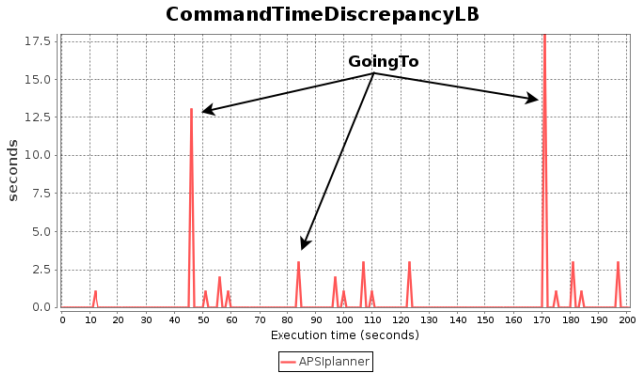


Figure 7.16: Temporal profile of the CTD$_i^{lb}$ for GOAC.

Command Time Discrepancy Upper-Bound – CTD^{ub}. Figure 7.17 shows the temporal profile of the CTD$_i^{ub}$ for the execution. From such data we obtain that GOAC has a total deviation of 101 seconds between the maximum planned time and the real execution time. Then, the score for this metric is of 48.47 over 100. This means that the model is not very accurate respect to the robot behaviours. As happens with the previous metric, the problem resided mainly in the movement action, whose maximum duration is set to a big enough value to enable long moves.

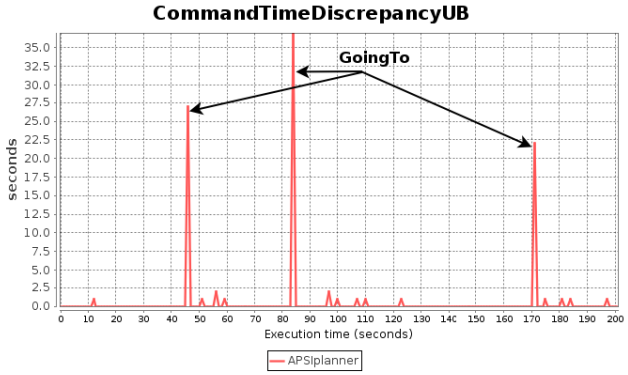


Figure 7.17: Temporal profile of the CTD_i^{ub} for GOAC.

Planner Model Analogy – PMA. In our example, we have that GOAC executes 16 actions to complete the objectives. The number of planner states updates are 72, and can be extracted through the temporal profile given in the fig. 7.18. The computed score is higher than 100, and thus, normalized to 100. This means that each action has a reflect in the planner database, closing the sense-plan-act cycle.

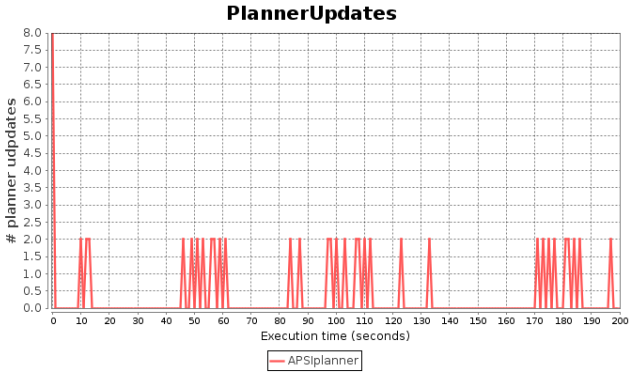


Figure 7.18: Temporal profile of the planner updates for GOAC.

7.6.3 Planning performance

For our example, GOAC has a fixed time slot (set by the user to 10 seconds) for the planning process. If the planner generates a plan before the time limit, the system will wait until the end of the time slot. Properly evaluating the planning performance will allow us to set this value for a better performance. The values for the metrics in this group are:

Planner Deliberation Time – PDT. Figure 7.19 presents the temporal profile of the deliberation time for our execution. It presents two planning processes, one at the beginning (to plan the two initial goals) and another one later (to plan the injected goal). From the data gathered in that chart, the deliberation time is the sum of the time employed for these two planning processes, i.e., 0.341 seconds. Then, the scores for this metric is 98.26 over 100. This implies that GOAC does not spend much time deliberating regarding to the time employed to complete its objectives.

Figure 7.19: Temporal profile of the deliberation time for GOAC.

Planner Deliberation Memory – PDM. In fig. 7.20 we show the temporal profile of the memory used by GOAC during execution, differentiating the memory required for each layer. We can see that there is a peak in the memory required for the deliberative layer (APSI planner) at the begin of the execution, as it is generating the first plan. In this case, the maximum memory allocated by the GOAC deliberative is 1.6% with an average value of 1.39%, providing a score of 86.88.

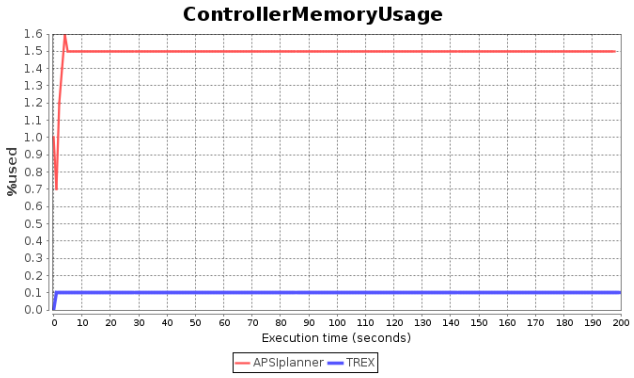


Figure 7.20: Temporal profile for the memory usage in GOAC.

Planner Deliberation Efficiency – PDE. In our execution scenario, GOAC properly plans and executes the three goals. The time executing commands is 150 seconds and the deliberation time is 0.341 seconds, giving a PDE score of 99.24. In this regard, the planner is efficiently deliberating in a small amount of time regarding to the time required to execute the actions contained in the plan.

Planner Synchronization Ratio – PSR. During execution, the planner receives updates of the environment/platform with the temporal distribution shown in fig. 7.18. From such data, we observe that GOAC acknowledges 72 changes of the environment/platform during the execution. As in the previous section, we want a frequency of 1 second for the environment monitoring. Then, the score for this metric is 36.73 over 100. This means that some states are only notified at the end of commands execution, e.g., we cannot track the movement action or the transmis-

sion of a picture at a fine coarse granularity. Then, it is not possible to monitor commands execution at the desired frequency.

Planner Synchronization Frequency – PSF. In our example, GOAC keeps its planner knowledge base up-to-date by means of a periodically task that monitors the environment/platform every second. Then, its score is 100.

7.6.4 P&E integration

If we analyse how GOAC performs during execution, we appreciate that the planner and the executive are highly coupled, sharing a common knowledge database. Then, it is expected that sharing information among layers is fast. The values of the metrics are next reported:

Controller Processor Usage – CPU. The temporal profile of the processor usage (CPU_i) for the execution is shown in fig. 7.21. GOAC has a 0.25% of average processor usage for the deliberative and 0.26% for the executive, providing scores of 99.75 and 99.73 respectively. Then, we have a score of 99.74 over 100 for the controller (average score for the components). This means that both layers require low processor usage and are suitable for lower resources computers than the used in the experiments.

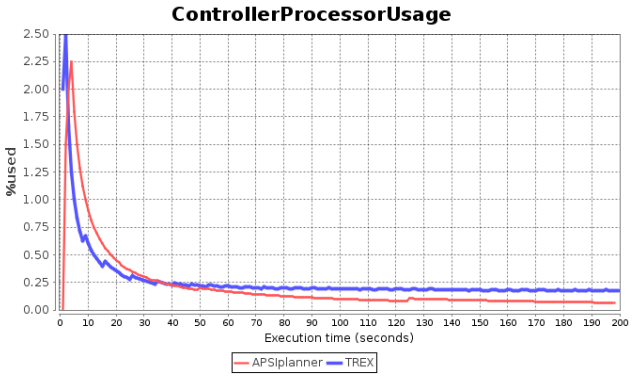


Figure 7.21: Temporal profile for the CPU_i in GOAC.

Controller Memory Usage – CMU. The temporal profile of the memory usage for the execution is presented in fig. 7.20. From the data in such profile, the average memory required by GOAC is 1.39% and 0.1% for the deliberative and executive respectively. The scores are 98.59 and 99.90 for the layers and 99.24 for the controller. This implies a very low memory usage of the computer used.

Controller Dispatching Time – CDT. During execution we obtain a temporal profile such as the presented in fig. 7.22, where each peak represents the time required by the controller to dispatch a command for execution. Then, the dispatching time is in total 0.003 seconds (APSI planner) and 0.001 seconds (TREX) for translating and

dispatching the commands. Then, the controller score is 99.99. As both layers share the data with a common representation, the only translation required is to transform the timeline attributes into numerical values for the functional layer, which is very fast given the results.

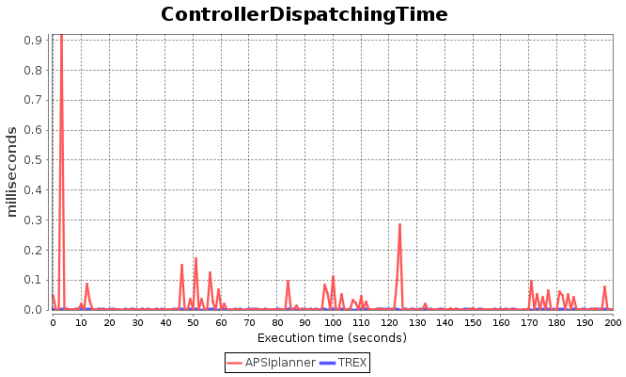


Figure 7.22: Temporal profile for the dispatching time in GOAC.

Controller Sensing Time – CST. Figure 7.23 presents the temporal profile of the sensing time for the GOAC execution. The sensing time is the sum of all peaks, i.e., the time employed in translating each sensor data into a planner fact. For our example, the sensing times are 1.546 seconds and 0.01 seconds for the different layers, providing a score of 99.20 over 100. In this way, the process of generating the timelines states from the data given by the platform is fast, and, as the executive and deliberative share the same database, it is only required to translate the sensor data once to permit the access to the other layers.

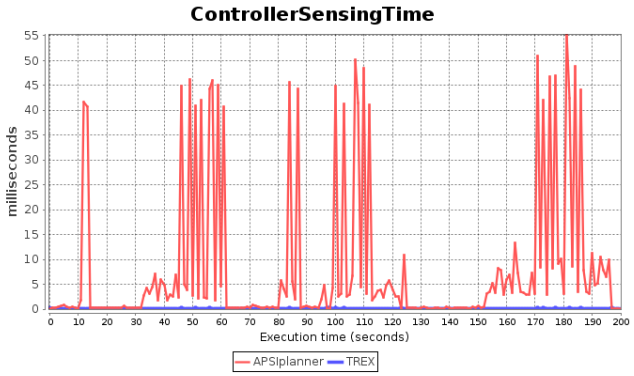


Figure 7.23: Temporal profile of the sensing time for GOAC.

Controller Monitoring Time – CMT. In the case of GOAC the different reactors spend some time each second to analyse the current state of the system. This is specially relevant in the case of the APSI planner: it uses this time to check the coherence between the plan and the execution status. As a consequence, GOAC generates a temporal profile when synchronizing its internal data such as the presented in fig. 7.24. In this execution the monitoring times for GOAC are 7.764 seconds and

0.016 seconds for the deliberative and executive respectively. Thus, the CMT score for the controller is 96.03. This means that the controller requires low time to check the coherence between the current execution status and the planned one.

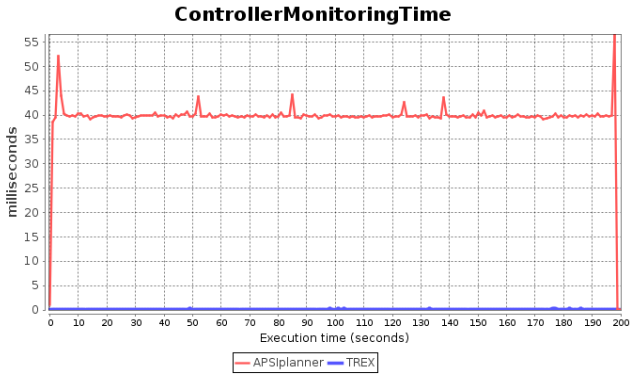


Figure 7.24: Temporal profile of the monitoring time for GOAC.

Controller Reaction Time – CRT. For our example, we have a new goal injected at time 55 seconds. Such goal is considered by GOAC in a planning process that starts at time 122 seconds (see again fig. 7.19). Thus, the CRT_i for that contingency is 122-55 = 67. There is only one contingency ($n = 1$), so the CRT score is 65.81 over 100. In this sense, GOAC delays the planning process until the previous plan has been executed, which may have a negative impact in the execution. For instance, failures have to be considered as fast as possible, while usually the plan performance could be increased when injected goals are rapidly taken into consideration. In this direction, if we inject a goal close to the current position, delaying the planning process could lead to a longer execution if the planned actions were farther away from the actual position.

7.6.5 GOAC assessment

With the metrics presented we are able to evaluate GOAC considering the execution of the controller in an exploration scenario. The GS for this execution is 8.16, which is notoriously better than the obtained by MoBAr. In this sense, we can see, given the different metrics scores, that GOAC properly integrates P&E. However, the strategy of delaying the injected goal until the previous goals have been achieved, reduces its evaluation. The planner performance also shows good scores, except for the synchronization ratio.

Focusing on the model adequacy, the model employed lacks on the actions duration, specially in the movement action. In any case, the evaluation allows improving the controller for our scenario. In the case of the plan accuracy, we observe that the lower planning horizon is quite accurate, but the upper bound could be improved. As well, by reducing the planning slot we can obtain a better score for the effective time. In this sense, the metrics not only allow us to evaluate the controller, but also to select the best controller parameters for our application. As well, the controller can be enhanced by modifying how it plans for injected goals. Probably, it is better to replan the new goals when they are injected, instead of delaying them.

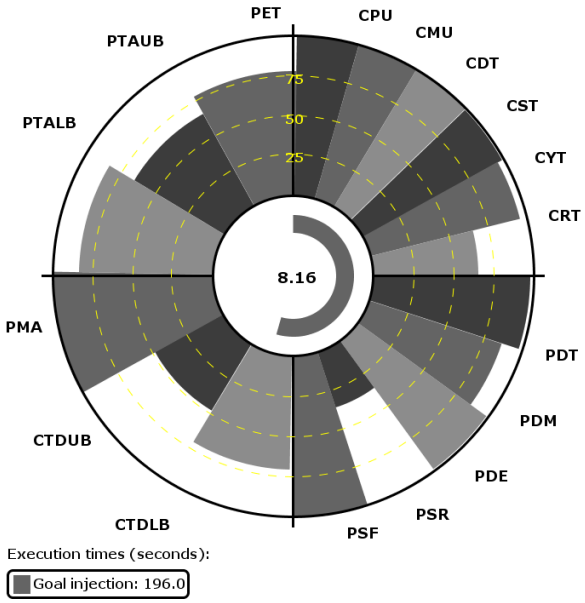


Figure 7.25: Summary report for the GOAC example.

7.7 The OGATE software tool

To perform the controllers assessment with the previous metrics requires some effort. Particularly, executing several times a controller and gathering the data generated is an onerous task. Moreover, following the methodology, it is required to perform the test several times (to collect average behaviours) and under different scenarios (to evaluate autonomous capabilities). It is possible to manually carry out such work, but it is impractical if we want to perform large testbenchs. Then, it is desirable a software tool to automate the controllers assessment through the methodology and metrics defined in the previous sections.

In order to support autonomous controllers assessment we have developed the On-Ground Autonomy Test Environment (OGATE) [118, 119]. It offers capabilities required for automated testing of autonomous controllers: controller instantiation within a scenario defined by the user, supervision of the plan execution and report generation with the information gathered during the test as shown in fig. 7.26. Then, OGATE can be exploited in order to implement a suitable sequence of evaluation steps for supporting the objective assessment of autonomous controllers, defining testbenchs that can be shared and reproduced by different researchers. Furthermore, OGATE also constitutes an interactive tool to help designers and operators of autonomous controllers providing an interface for in-execution control and inspection of the controlled system during execution.

To enable controller assessment with OGATE, it is required to include internal monitors in the system under study. The objective of the monitors is to measure the parameters required to compute the metrics scores (e.g., deliberation time, planning horizon) for the evaluation. Then, to provide the metrics values (and optionally, telemetry data) to OGATE a TCP/IP communication has to be set. This enables a data channel between OGATE and the controller, which can send the metrics values

at any time during execution. Moreover, multiple communication channels can be set, so it is possible to provide an insight of the metrics values for the different layers of the controller. With the data provided it is possible to generate temporal profiles for the metrics, providing charts such as the presented in sec. 7.5 and 7.6, which are accessible by the user in real time. As each layer can provide its own data it is possible to inspect the controller performance with more detail.

OGATE also allows defining a second interface to enable execution supervision of the autonomous controller. This interface provides functionality to manage the execution (start, pause or finish) while monitoring the controller status during the test (executing, paused, failing), and at the end (execution correct or failed) for the different layers. Additionally, this interface defines a telecommand function that allows OGATE to send instructions to be executed, for instance, goal injection during test execution. Implementing this interface in the autonomous controller enables automatic benchmarking and user-controlled execution.

Once the interfaces have been set in the controller under study, a previous step to perform assessment is to supply to OGATE a configuration file with the scenario definition. This file entails the evaluation design of the proposed methodology. In this regard, the configuration includes the scenarios description (i.e., domain and problem), user-defined metrics for assessing specific domain characteristic (the metrics introduced in sec. 7.3 are attached by default) and the controller configuration (executable files and their parameters). To enable assessment of different configurations for the same controller, it is possible to provide different values for each parameter defined in the configuration. For instance, different goals and missions durations can be set. Then, OGATE will set up the different combinations for all pairs of parameter/value defined.

The last step before the assessment is to define the non-nominal execution scenarios (i.e., dynamic goal injection and execution failure). To do this, a controller-dependent file is required. It includes the definition of the events to be sent to the autonomous controller, and the time in which they will be injected during execution. In this regard, OGATE only dispatches the events to the controller using the telecommand interface, i.e., OGATE does not manipulate that data.

Reaching the previous requirements enables automatic tests execution, data gathering and report generation. In this direction, multiple executions of the same controller can be done to collect average metrics behaviours. At the end, OGATE provides the user with a report about the collected data. It includes temporal pro-

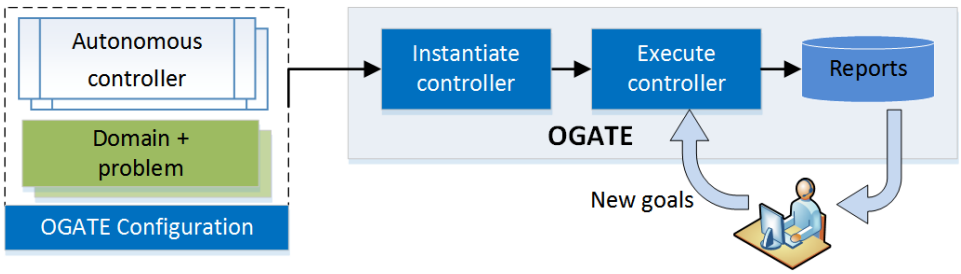


Figure 7.26: Tests execution through OGATE.

filing for the different metrics values and the graphical report presented in sec. 7.2. Moreover, such information is also available during execution through a GUI, which enables the user to inspect the evolution of the controller in real-time. As well, there is a dedicated slot in the GUI that can be customized by the user to represent the data gathered by the telemetry interface. This dedicated interface can also enables the user to interact with the controlled system, for instance, allowing injecting goals during execution, providing an initial support to integrate the HMI.

Technically, OGATE is composed of three different modules as depicted in fig. 7.27. Each module provides the services required to enable assessment through the presented methodology. Initially, the Mission Specification module takes the configuration files required to instantiate all elements, and then, the Mission Execution module executes all generated configurations for the autonomous controller(s), while retrieving the metrics and generating reports at the end of the execution. All steps can be controlled/monitored by the user through the OGATE GUI module. Next, these modules are explained.

- **Mission specification:** allows the system to deal with the configuration file and the instantiation process, that is, the description of the components that defines a control architecture and the platform over which they operate. In this way, the system provides a convenient mode to allow the user to configure the components of the autonomous controller to evaluate. For each controller, it is possible to define different values for a parameter, so this module is in charge of generating all possible instances for large test campaigns combining the parameters defined in the configuration file.
- **Mission execution:** this module provides the execution support to deal with the complexity of the autonomous controller under evaluation. It supports the automatic testbench execution of the user defined scenarios, meanwhile gathering the metrics data. In this regard, using specific defined interfaces, OGATE is able to access to the different layers of the autonomous controller to supervise and retrieve the metrics data. Also, during the execution, the user must be able to interact with the controlled system, for instance, including new goals to change the nominal execution, in order to test the robustness of the system and features of different components such as replanning capabilities of deliberative components.

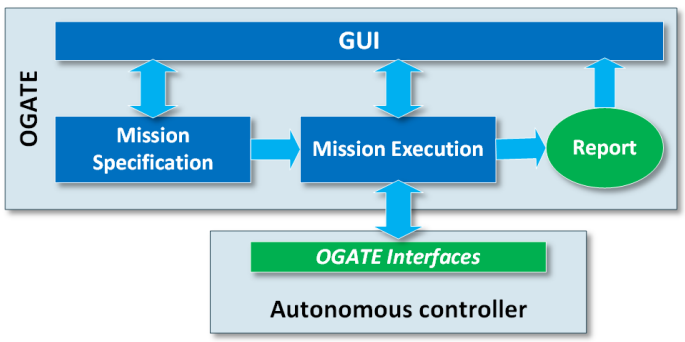


Figure 7.27: OGATE concept.

- **Report:** the collected information are exploited to generate detailed reports to support assessments based on the analysis of the considered metrics. In this way, OGATE provides a graphical report as the ones presented in fig. 7.11 and 7.25, but also the temporal profiles of the selected metrics and their representative values (minimum, maximum, average and aggregated value) in a Comma Separated Values (CSV) file that can be assessed with other analysis tools.

- **OGATE GUI:** this module contains graphical support for the previous modules to provide a user friendly environment. For the Mission Specification module the GUI gives features to create/modify the OGATE configuration file. For the Mission Execution it provides the execution status for the different components, graphical reports for the metrics, and controller specific controls (implemented by the user) to manage the execution through the graphical interface.

While some engineering effort shall be done to exploit the OGATE capabilities, a similar or higher effort should be dedicated in order to perform hand tailored tests campaigns. In this sense, the benefits of exploiting OGATE are the saving effort for preparing the tests campaign, the data gathering and analysis, and easing the dissemination of the required elements to reproduce the experimental setups.

Finally, OGATE has been designed to directly connect the planning and/or execution layers of an autonomous controller, and allows performing experiments with either real or simulated robotic platforms exploiting different robotic frameworks (e.g., ROS, ROCK [39], OpenPRS, PLEXIL).

7.8 Experimental evaluation: MoBAr and GOAC comparison

In this section we present a comparison between MoBAr and GOAC for the exploration scenario defined in sec. 7.4. The configuration of the two controllers is the same presented in sec. 7.5 for MoBAr and sec. 7.6 for GOAC. Both controllers relies on different P&S paradigms: MoBAr exploits an action-oriented planner, while GOAC is based on timelines [126]. As well, both controllers exploit different solutions for P&E integration. As stated before, the execution objective is to acquire and to communicate two pictures in different locations, with a communication opportunity that starts at time 50 seconds. Each execution has a maximum time allowed of 300 seconds. As well, the following execution scenarios are considered:

- Nominal execution, i.e., no perturbations affect the execution.

- Dynamic goal injection in which a new picture goal is included between time 50 and 60 seconds after the execution starts.

- Execution failure in which the second PTU movement is not properly executed.

For the presented scenario we have executed each controller 10 times for each scenario to collect average behaviours (30 total executions for each controller). The metrics that provide the performance assessment are those introduced in sec. 7.3. For the metrics *relevance*, we consider that each metric group contributes with 25% of the controller score and the weights are uniformly distributed for each metric group.

The test execution is automatically carried on by OGATE in a PC endowed with an Intel Core i7 2.5GHz processor with 8GB of RAM. After execution, OGATE provides the temporal profiles of the metrics for each execution (such as the one presented in sec. 7.5 and 7.6), as well as the summarized graphical report that enables the controller comparison (see fig. 7.1).

First, lets have a look at the average execution time for each scenario in table 7.1. It is remarkable that GOAC is unable to solve the execution failure scenario (as happened in the previous section). It is also notorious that MoBAr is faster than GOAC. However, assessing the *GS* in table 7.2, we can observe that GOAC outperforms MoBAr for the *GS* in the nominal and goal injection scenarios, even when achieving the objectives requires more time. In this regard, assessing an autonomous controller without providing an insight of the P&S integration does not allows us to extract relevant conclusions.

Figure 7.28 presents the graphical assessment for both controllers. This report will ease us to isolate different aspects of the controller that entail the performance differences. As well, the graphical representation contains the data presented in tables 7.1 and 7.2. The graphical reports for each execution scenario can be seen at the end of this section in fig. 7.29.

Then, starting the controllers comparison from the *GS*, we can observe that each controller has similar scores for the nominal and goal injection scenarios. This means that injecting goals during execution does not reduce their performance. As well, the *GS* for MoBAr remains nearly constant in the execution failure scenario. Instead, GOAC is not able to complete such scenario. In this regard, the GOAC deliberative component is able to generate a valid plan, but the controller fails in properly completing its execution. In particular, when the deliberative component receives the PTU miss-configuration, it does not correspond to its planned states, producing

Table 7.1: Execution time (seconds) for each scenario (average for 10 runs each).

	Nominal	Goal injection	Exec. failure
MoBAr	103.5	148.6	101.6
GOAC	128.6	201.6	–

Table 7.2: *GS* for 30 executions (10 runs each scenario).

	Nominal	Goal injection	Exec. failure	*Average*
MoBAr	5.62	5.64	5.61	*5.62*
GOAC	7.86	8.11	0.0	*5.33*

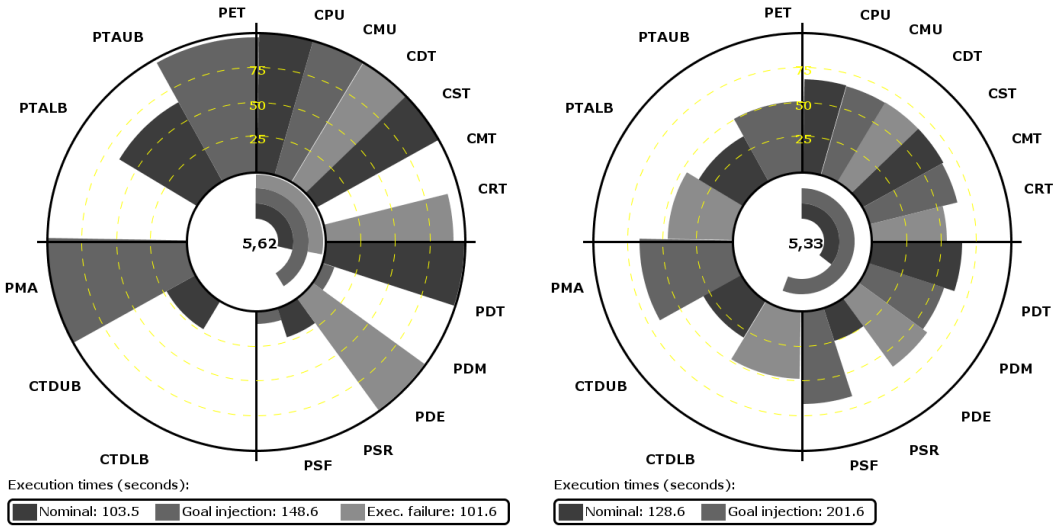


Figure 7.28: Average values for 30 executions of MoBAr (left) and GOAC (right).

a failure that leads to a system halt. By its side, MoBAr is able to solve this issue discarding the current plan and generating a new one. The failure to solve 1/3 of the scenarios significantly reduces the average *GS* for GOAC. Notwithstanding, the average evaluation for the 30 executions of GOAC is only 0.3 out of 10 less than the average *GS* for MoBAr.

Focusing on the external ring, i.e., the metrics scores, we can characterize each controller for the four metrics groups (see fig. 7.2). While in MoBAr (fig. 7.28 left) the metrics can fill all the ring (all executions are completed), in GOAC (fig. 7.28 right) the maximum score is 66.6 over 100 due to the inability to complete the execution failure scenario.

We start analysing the metrics in the first quadrant (top-right), which summarize the scores for the P&E integration. We can appreciate that both controllers do not require excessive resources, i.e., CPU and CMU have high scores. As well, the command dispatching and sensing time (CDT and CST) have scores close to the best value. This indicates that dispatching commands and perceptions among layers is fast, i.e., the planner and executive are well integrated. Then, we assess the metric that evaluates how the system is monitoring the environment/platform, i.e., the CMT. In MoBAr there is no monitoring of the plan execution, the internal state is only updated after a command execution. Thus, its score is 0. Instead, GOAC is permanently observing the execution, obtaining a good score (in average 98 over 100 for the nominal and goal injection scenarios). Finally, the CRT provides us a measure of the delay between a new goal is added or an execution failure arisen and the subsequent planning process. MoBAr outperforms GOAC in this metric with scores 92 and 81 (without considering the execution failure) respectively. The reason of this is how the replanning process occurs. In GOAC the goal added during execution is planned when the initial goals are executed. Instead, MoBAr replans after the current command executions ends, discarding the plan in execution and generating a new one.

For the second quadrant (the metrics that provide a measure of the planner performance) we can observe that the PDT is slightly better in MoBAr. This means that the planning process is faster. MoBAr requires near 214 milliseconds in average while GOAC employs 7287 milliseconds in average. However, for the PDM MoBAr obtains a significant low score. In fact, during execution, the planner is not running, and the memory employed in runtime is just used to store the plan. Then, during deliberation, it requires some extra memory in opposition with the used during plan execution. By its side, GOAC maintains the deliberative in execution, but it requires a low extra memory while deliberating. Regarding to the deliberation efficiency, MoBAr has a better score, 99.56, than GOAC 92.51 (without considering the execution failure scenario). The reason of this is the planning slot in GOAC: there is a fixed slot of 10 seconds for the deliberation process, even when the deliberation requires less time. Currently, the system is idle for near 6 seconds after the plan generation. The last metrics in this group analyse how the planner knowledge base is updated, i.e., PSR and PSF. In this regard, MoBAr has significant low values for both metrics as the planner is only updated at the end of a command execution. In the opposite, GOAC obtains the best score in the PSF as the deliberative is continuously monitoring the platform/environment during execution. However, the PSR has a lower score as the planner is not updated each second. For instance, during the movement action, the planner does not receive the current robot position; it is only provided at the end of the command execution.

Considering the metrics that analyse the model employed (in the third quadrant) we can observe a relevant issue. MoBAr has a 0 score for the CTD^{lb}. The reason is that the planner in MoBAr only provides one value for the action duration. Such value is set for the maximum duration of the action, and thus, it is used to compute the CTD^{ub}. Instead, GOAC provides a temporal range for each action, so it can properly monitor the temporal coherence of the plan during execution. Besides, GOAC outperforms MoBAr for the CTD^{ub} with scores 46.5 (without considering the execution failure scenario) and 23.6 over 100 respectively. In any case, this indicates that the temporal model of the actions is not very accurate. In particular, we have different actions: *move the PTU*, *take pictures*, *transmit the pictures* and *going to*. While most of them have small uncertainty in the time required for execution, the *going to* action is dependent of the distance travelled (assuming constant velocity). However, the models employed in both planners do not consider such issue as they do not integrate path planning capabilities. Then, the maximum duration for each *going to* action is set to a high value, big enough to reach all the different locations. For the PMA metric both controllers obtain the best score as the planner receives data for each command executed, i.e., the controller has a closed execution loop.

Finally, for the evaluation of the plan accuracy (fourth quadrant), MoBAr obtains a 0 score for the PTA^{lb}. Instead, the plan provided by GOAC is temporal bounded. The score for this metric in GOAC is 69.85 without considering the execution failure scenario. However, MoBAr has a better PTA^{ub} (64.25) than GOAC (55.03 without considering the execution failure scenario). This implies that the plan generated by MoBAr is more approximated to the execution time than the one produced by GOAC. The reason is because GOAC employs a temporal planning horizon (set by the user) that allows time flexible allocation of the actions, while

in MoBAr the plan is time fixed and the maximum planning horizon is set by the planner. As a consequence, the planning horizon in GOAC is longer and the metric score lower. The metric that expresses the time employed accomplishing the mission objectives (PET) shows that MoBAr has a good score (near 96 over 100). However, GOAC obtains 77 without considering the execution failure scenario. In fact, the performance degradation in GOAC is due to the fixed deliberation slot and several delays in command executions.

Assessing the data generated in the presented experimental campaign allows us to extract relevant facts about different aspects of the controllers under study. Particularly, MoBAr does not supervise properly the plan execution, which can be problematic in dynamic environments. By its side, GOAC is designed to work with temporal flexibility, but it requires some implementation work to complete the failure execution scenario. As well, the temporal model for both controllers can be enhanced, as the current ones are not so accurate (mainly in the `GoingTo` action).

7.9 Summary

In this chapter we presented a framework that enables autonomous controllers evaluation and comparison. The objective of such effort is to generate objective and reproducible experimental results for the integration of AI techniques in robotics. Particularly, we assessed the integration of P&S systems in robotics, characterizing the models employed and the performance of the planning process. As well, we evaluate how well P&E are merged, providing a better characterization of deliberative layers when dealing with real application scenarios. In this direction, we defined a methodology to guide the testing process and a set of metrics that are generally applicable to plan-based controllers, independently of the technologies used or the application domain. Both elements are operationalised in the OGATE software tool that automatically carries on with large testbench for different execution scenarios. Then, it is possible to evaluate and compare different autonomous controllers based on objective criteria.

To demonstrate the effectiveness of the framework, we performed a testbench evaluating the GOAC and MoBAr controllers. From the analysis of the results, we could compare the performance of different controllers, but also test different parameters and configurations to choose the best one for our application. In general, it is worth underscoring that performing the same evaluation without OGATE represents a significant effort in terms of coding, customization of specific metrics, collection of performance information, and generation of reports. The main effort required for using OGATE is related to the implementation of the interfaces for the specific autonomous controller. Thus, OGATE constitutes an off-the-shelf tool for carrying out a significant amount of work in an automated manner.

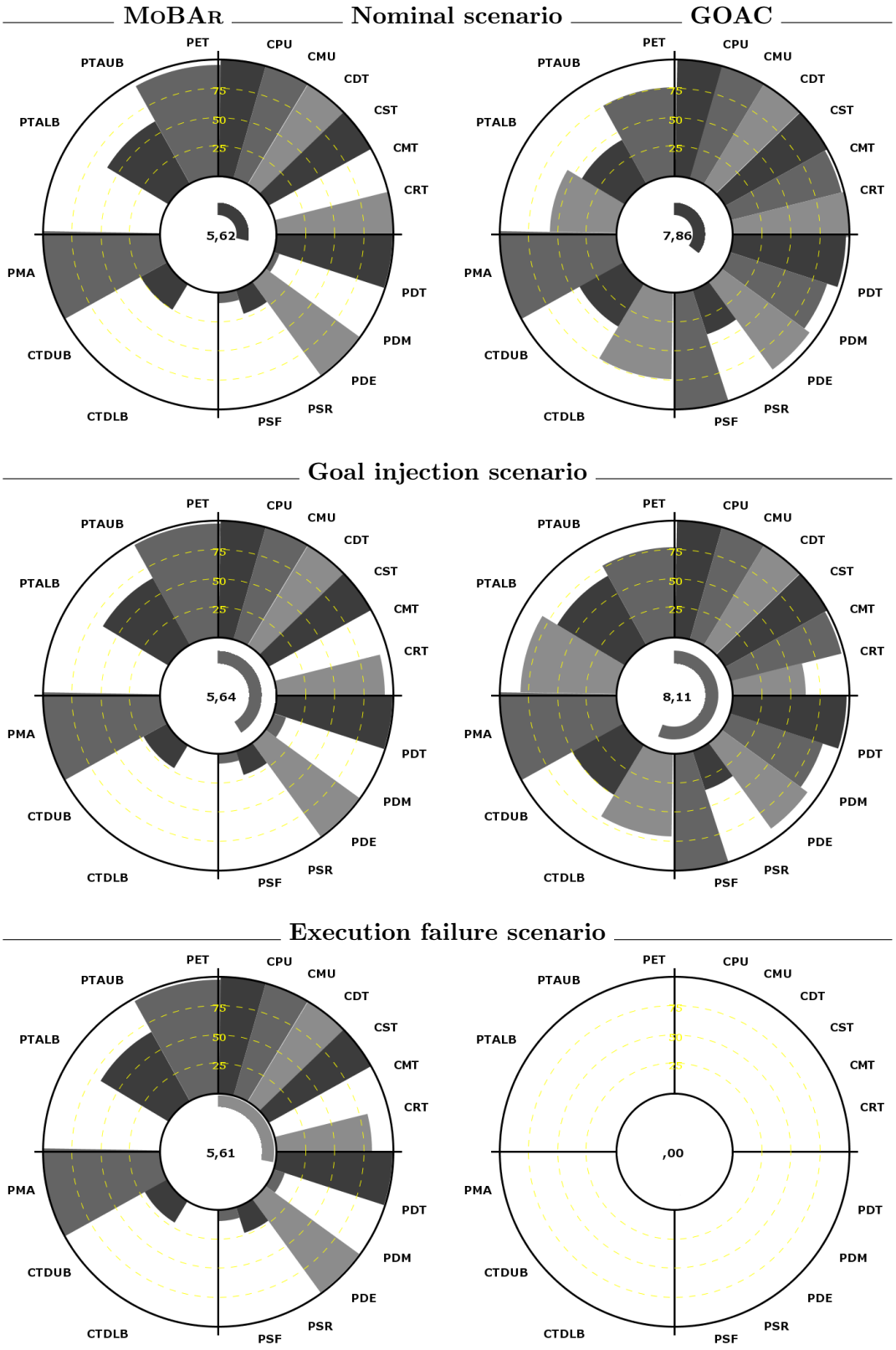


Figure 7.29: Evaluations for the different execution scenarios (10 runs each) of the controllers under study.

Chapter 8

Conclusions

This last chapter presents the conclusions of this dissertation. They are divided into three parts, corresponding to the three main aspects covered in this thesis: (i) the path planning algorithms; (ii) the MoBAr autonomous controller (including the up2ta planner) and; (iii) the evaluation of and characterization of autonomous controllers with the OGATE framework. The dissertation ends with a list of future research lines.

8.1 Path planning

We have introduced the angle deviation between nodes during the search phase. Applying this parameter in the evaluation function of the search process we can modify path planning algorithms in two ways: (i) improving its efficiency when we use it as a dependent domain heuristic, which reduces the computational resources used during the path extraction, but with a slight degradation of the generated path length respect to the original algorithms, and; (ii) producing a new algorithm, S-Theta*, that minimizes the heading changes during the search process, not only the path length as the former one does.

Using the angle parameter (named Alpha) as part of the heuristic function of a path planning algorithm has three advantages: first, it is easy to implement, and is valid for any heuristic search path planning algorithm, such as those based on A*. Second, we can modify the behaviour of the Alpha parameter using a factor to deal between the runtime and the degradation of the other path parameters. Finally, Alpha affects to the memory required: using this value makes that the search algorithm expands less nodes, and thus, less memory and time are required.

We have also presented the S-Theta* algorithm, based on Theta*, exploiting the Alpha value as part of the cost function. The new algorithm effectively reduces the number of heading changes of the generated path, as well as the total cost associated with these turns. As the experimental results show, the S-Theta* algorithm improves the former algorithm on the total turn, in exchange of a slight degradation on the path length. Notwithstanding, if we consider that an optimal solution is the one that minimizes both, the path length and the total turns, S-Theta* gets better results than Theta*.

Finally, we have proposed a new any-angle algorithm named 3Dana. It is designed with the purpose of considering the terrain relief during search, while also minimizing the heading changes of the generated paths. 3Dana integrates during the search the DTM combined with a traversability cost map, so it is able not only to avoid potentially dangerous areas, but also excessive terrain slopes in order to generate safer routes. Besides, during the search process, the algorithm calculates the necessary turns needed to reach the next position taking into consideration the current heading and the position of the goal, such as S-Theta* does. Additionally, 3Dana provides different parameters that allows customizing the search process to adapt the algorithm to each robotic platform and application requirements.

8.2 MoBAr autonomous controller

The autonomous control of a robot requires the deployment of P&S techniques, which usually constitutes a high effort to perform the modelling and P&E integration. In this dissertation we have presented the MoBAr autonomous controller. It is a 3T architecture that aims to provides a modular way to implement an autonomous control architecture using models. To design the models that abstract the underlying hardware of the robot, some general purpose technologies have been identified and exploited. The deliberative integrates a PDDL based planner and the path planning algorithms presented in this dissertation. Using a standard language it is possible to reuse the model with different deliberatives. The executive layer uses models codified in PLEXIL. These plans provide an abstraction to decompose, execute and monitor commands implemented in the functional layer. This last layer is the one that implements the particular hardware to control the robot, but using the G^{en}oM and ROS frameworks it is possible to implement the functionality in an incremental way, using standard interfaces given by the frameworks.

In mobile robot applications it is required that the planner provides optimal sequencing of goals. In this regard, we have included in MoBAr the up2ta planner that integrates task planning using a PDDL-based planner, and a path planning algorithm to generate the paths between goals. The motivation of this system is to obtain a deliberative layer for autonomous robots, that can interleave task planning and path planning in order to obtain a close to optimal ordering of tasks, taking into consideration the path between them. This approach allows the robot to take advantage of each planner goodness to deal with more complex scenarios ensuring a good performance and improved quality of the generated plan. Also, it is possible to replace the path planner, deploying different instances of up2ta adapted to the application needs.

We have shown evidences of how MoBAr provides an incremental design philosophy. The use of PDDL, a very extended language in the planning community, or PLEXIL, simple and easy to understand, allows MoBAr to work as a black-box which, connected with a particular functional layer, enables the deployment of functional prototypes in a short period of time. In this regard, we provide a general interface for connecting PDDL-based planners, which simplifies to swap between different planners, domain and problems. Also, a set of interfaces to connect different G^{en}oM modules is provided. If a different functional layer is used, such as ROS,

some engineering effort must be done in order to create the interface between the functional layer and the executor. Moreover, a deployment with ROS is provided as well.

To demonstrate the capabilities of MoBAr some experiments have been performed with either real and simulated robots with different characteristic. In this regard, the same models (for the deliberative and executive layers) have been tested in the ExoMars rover (simulated) and with a commercial platform supported on ROS, the TurtleBot 2 platform. In this way, an exploration mission consisting of pictures acquisition have been carried out, demonstrating the effectiveness of the controller in different execution scenarios, which include on-board replanning. Moreover, we have extended the up2ta experimentation providing a real testbench over the TurtleBot platform. Such experiments demonstrated that the task planning and path planning integration performed in up2ta is suitable for mobile robots scenarios, providing better solutions than approaches in which the planners are less coupled.

8.3 Autonomous controllers assessment

In the experimental section of the MoBAr controller we have demonstrated the effectiveness of the system. However, only a limited set of test were performed and we did not provide any comparison with other state of the art autonomous controllers. The reason is that there is not a standard way to perform experiments according to some structured methodology.

In this dissertation we have proposed a work towards producing a framework for plan-based controllers assessment. To deal with this open issue, a methodology to properly guide the testing phase to enable generation of reproducible testbench campaigns was proposed. The methodology is operationalised through sequential steps that cover the scenario definition, the test execution and the report and performance assessment. It also defines a compact graphical representation for the performance assessment, in order to simplify the comprehension and comparison of controllers performance over different operational scenarios. Such methodology is supported by a set of general applicable and formally defined metrics that allows us to characterize the performance of plan based controllers by means of the integration between the P&S systems, while also analysing the correctness and accuracy of the deliberative components and the models employed for abstracting the robotic platform and environment. These metrics are grouped into four performance areas, namely, P&E Integration, Planner Performance, Planner Model Adequacy and Plan Accuracy. Each group provides details about different aspects of the controller under study that provide an objective evaluation of the parameter under analysis. Then, exploiting the methodology and metrics, it is possible to assess autonomous controllers regardless the application domain or the technology employed.

These metrics and methodology are operationalised in a software environment, called OGATE, that automatizes the benchmarking process. OGATE provides a general testbench to enable easy deployment of autonomous controllers. Also, specialized graphical interfaces could be enabled to show the relevant data to the user, encapsulating the complexity of the underlying functionality, and allowing a better

interaction with the controlled system. Finally, OGATE also offers an automated testbench to obtain quantitative comparison based on accurate experiments.

By means of OGATE, an exhaustive analysis of different planning policies for the deliberative component of the GOAC controller has been performed. Assessment has involved inspecting internal measures of the GOAC deliberative component while executing scenarios with increasing complexity. Within these tests we have been able to obtain different reports describing the performance of the system. This allows us to obtain conclusions that were hard to be achieved performing standalone tests.

Finally, using OGATE a test campaign consisting of evaluating MoBAr and GOAC has been performed. For the tests, both controllers operate over the same simulated robotic platform to perform an exploration mission. Analysing the data generated during the testbench, we are able to characterize both controllers in different execution scenarios. Using the methodology and the metrics defined, it is possible to fairly compare the performance of the two autonomous controller, even when both rely not only in different representations for the deliberative layer, but also in different P&E schemas implemented with different technologies. Given these results we can acknowledge that the current metrics and methodology constitutes a step forward to improve the quality of the experiments performed in autonomous controllers, allowing to better characterize the performance of such systems. It also enables to reproduce the results by other researchers since the results are supported by an objective and formal description of the metrics involved.

8.4 Future research lines

In the following we present future research lines related to the works presented in this dissertation.

- Currently, in the path planning community the comparison and evaluation of the algorithms is performed using the average values for the different parameters measured, typically focusing on the path length and the runtime. However, the average value does not provide enough information to made strong assumptions. For this reason, it could be interesting to stablish better practices in the path planning community evaluating the algorithms based on statistical values, following the direction started in this thesis.

- Continuing with the path planning evaluation, the experimental assessment typically consists on performing extensive execution of different path planning algorithms over: (i) binary maps with increasing number of randomly placed obstacles; and (ii) games maps. However, several aspects remain unclear in the definition of the testbenchs. For instance, in the literature there is not a reference algorithm to create the maps. In this regard, we want to evaluate the possibility of properly defining a map generation algorithm (such as the presented in appendix A) that can be used for the experimental assessment. Then, we need to ensure that the generation algorithm is suitable to generate large maps sets that are relevant for the evaluation of path planning algorithms.

- Currently UP2TA does not allow defining the temporal behaviour of the controlled system. Then, the integration with a planner that supports temporal constraints is encouraged. In this regard it is not only interesting to exploit a PDDL 2.0 planner (or newer), but also testing the integration proposed in timelines based planers (e.g., APSI-TRF). In this direction, the temporal behaviour of the controlled system could entail new constraints when searching for the optimal solution.

- We argue that the average value could not be enough to made strong assumptions for large benchmarks. In this regard, the current definition and implementation of the metrics in the OGATE framework exploit average values. Notwithstanding, the software tool currently captures other values, i.e., minimum, maximum and aggregated. However, these values are not used in the evaluation. Then, a required step is to provide a statistical analysis of the metrics, which could rely in a better assessment.

- The current metrics set for autonomous controllers evaluation provides an insight of different aspects of P&S, P&E and the integration of these techniques in robotics. However, it is required to continue the research effort to isolate other aspects (e.g., resource usage, interaction with other agents, etc.) that could be relevant for the plan-based controllers assessment.

- The OGATE software could be enhanced by adding new modules to provide extra functionalities. For instance, a plan supervision module could be added to assist the human operators during the execution of plans. In this regard, a potential objective is to include a HMI to simplify the comprehension of the controller execution, while also providing a step toward the implementation of mixed initiative controllers.

- We have being able to assess two autonomous controllers through OGATE. Meanwhile both controllers relies on different P&S and P&E schemes, it is desirable to evaluate and compare other controllers that have similarities/differences with the previous ones. Then, we could better demonstrate the generality of our approach.

- As well, it could be interesting to integrate UP2TA in GOAC and performs a comparison between MOBAR and GOAC exploiting the same deliberative. This could rely in executing more complex domains in which path planning is required.

- We have currently evaluated the controllers over a particular domain. Thus, it is desirable to test the same controllers over different domains, assessing the adaptability of the controllers, meanwhile we can demonstrate that the metrics and methodology are fully domain independent.

Appendix A

Random maps generation

To perform comparative tests over several path planning algorithms two strategies are generally followed. The first one consist of using previously generated map sets, typically obtained from games. This approach is quite limited as the number of available maps is generally low, while also the maps are very limited in their properties (e.g. size, obstacles). The second testbench is based on random generation of large maps sets. This approach provides more possibilities as the maps can be generated using different parameters to set the maps dimension, number of obstacles, etc. However, the form in which these maps are generated is rarely indicated. Therefore, in this appendix we introduce a random map generation schema based on parameters. This method differentiates two parts: the DTM generation and the obstacles positioning, which could include the generation of a random traversability cost map. So, using the algorithms presented here it is possible to generate maps that can be fairly used to compare different path planning algorithms.

The elevation map or DTM are generated by using the Hill algorithm shown in alg. 7. The logic of this algorithm consists of selecting a point of the map randomly and increase the altitude of the surrounding area accordingly to a radius given by the user. What we obtain is a hill at such point. Repeating this procedure several times generates a map with valleys and hills. Then, a last steps is to normalize the altitude for the user defined range.

The obstacles and traversal cost map are generated in a separate file. Generally, an obstacle is defined by a cell that has a cost greater than a maximum defined value, c_{max}. Also, the minimum cell cost is 1, so, at least, the path cost is equal to the path length. The obstacle positioning and the cost map generation are done by the algs. 8 and 9 respectively.

The obstacle generation algorithm randomly places obstacles until it reaches a maximum percentage of blocked cells defined by the user. As well, each obstacle has a maximum dimension set by the user. After placing an obstacle its perimeter is protected. This means that further iterations of the algorithm cannot place an obstacle in such area, so there is no unreachable nodes in the generated map. This ensures that the generation algorithm produces maps in which it is possible to find a path between each pair of nodes. Moreover, the first/last column/row are protected. As a consequence of the generation algorithm, in function of the dimension of the obstacles and the percentage of blocked cells, it is possible that the algorithm does

not converge. In practice, with less than 50% of blocked cells, the algorithm usually generates a map in a negligible time.

If costs have to be included, these will be added after positioning the obstacles in the same map. First, a random position is selected along with a random cost value within a range from 1 to the maximum value defined by the user (c_{max}). From this point onwards, a rectangular region of dimensions specified by the user, will be created with the random cost value. The blocked cells will be ignored and the process will be repeated a user desired number of iterations. Those cells, which its associated cost has not been modified, will remain always with the initial value, being this value the unit.

The required parameters to define a testbench are the ones outlined in table A.1. N, R, z_{min} and z_{max} parameters correspond to the elements used to generate the DTM. O, DX and DY are required to generate the obstacles. Finally, and in case of including the transversal costs, M, CX, CY and c_{max} must be supplied. Additionally, it has to be specified whether the center-node or the corner-node schema is used. In the experiments carried out with random generated maps in this thesis, the corner-node schemas is used, while the parameters for the maps generation are presented in the last column of table A.1.

Table A.1: Map generation parameters.

Param	**Definition**	**Range**	**Value used**
N	Number of points to elevate DTM	$[1..\inf)$	$min(cols, rows)/4$
R	Radius of the circle to elevate	$[1..\inf)$	$min(cols, rows)/5$
z_{min}	Minimum altitude value	$(-\inf, \inf)$	0
z_{max}	Maximum altitude value	$[> z_{min}, \inf)$	$min(cols, rows)/4$
O	Percentage of obstacles	$[0\%, 50\%]$	$between\ 0\%\ and\ 40\%$
DX	First dimension of the obstacle	$[1, cols)$	$cols/25$
DY	Second dimension of the obstacle	$[1, rows)$	$rows/25$
M	Number of transversal costs regions to generate	$[0..\inf)$	$(cols \cdot rows)/(DX \cdot DY)$
CX	First dimension of the transversal cost region	$[1, cols)$	$DX \cdot 4$
CY	Second dimension of the transversal cost region	$[1, rows)$	$DY \cdot 4$
c_{max}	Maximum transversal cost	$[2, \inf)$	8

Algorithm 7 Hill algorithm for DTM generation

Require: $cols, rows$: map dimension
N: number of iterations
R: hill radius
z_{min}, z_{max}: normalized altitude range

```
1  dtm ← new empty map of cols × rows
2  for i ← 0 to N do
3      x ← random ∈ [0, cols)
4      y ← random ∈ [0, rows)
5      for zx ← x − R to x + R do
6          for zy ← y − R to y + R do
7              if zx ∈ [0, cols) and zy ∈ [0, rows) then
8                  nz ← R² − (zx − x)² + (zy − y)²
9                  if nz > 0 then
10                     dtm[zx][zy] ← dtm[zx][zy] + nz
11                 end if
12             end if
13         end for
14     end for
15     normalize dtm in[z_min, z_max]
16 end for
17 return dtm
```

Algorithm 8 Random obstacles generation algorithm

Require: : $cols, rows$: map dimension
O: percentage of blocked cells
DX, DY: obstacles dimension

```
1  costmap ← new empty map of cols × rows
2  blockedcells ← (O · cols · rows)/100
3  while blockedcells > 0 do
4      x ← random ∈ (0, cols)
5      y ← random ∈ (0, rows)
6      for ox ← x to x + DX do
7          for oy ← y to y + DY do
8              if ox ∈ (0, cols − 1) and oy ∈ (0, rows − 1) then
9                  if blockedcells > 0 and map[ox][oy] not protected then
10                     costmap[ox][oy] ← obstacle
11                     blockedcells ← blockedcells − 1
12                 end if
13             end if
14         end for
15     end for
16     for sx ← x − 1 to x + DX + 1 do
17         for sy ← y − 1 to y + DY + 1 do
18             if cx ∈ [0, cols − 1] and cy ∈ [0, rows − 1] then
19                 if costmap[cx][cy] empty then
20                     costmap[cx][cy] ← protected
21                 end if
22             end if
23         end for
24     end for
25 end while
26 return costmap
```

Algorithm 9 Random traversal cost map generation algorithm

Require: *costmap*: an obstacle map of $cols \times rows$
M: number of traversal cost regions generated
CX, CY: size of traversal cost regions
c_{max}: maximum traversal cost

```
1  for i ← 0 to M do
2      x ← random ∈ [0, cols)
3      y ← random ∈ [0, rows)
4      c ← random ∈ [1, c_max]
5      for cx ← x to x + CX do
6          for cy ← y to y + CY do
7              if cx ∈ [0, cols) andcy ∈ [0, rows) then
8                  if costmap[cx][cy] ≠ obstacle then
9                      costmap[cx][cy] ← c
10                 end if
11             end if
12         end for
13     end for
14 end for
15 return costmap
```

Appendix B

Path planning on Mars with 3Dana

In this appendix we present the solutions obtained with A* and 3Dana for three Mars maps obtained with the HiRISE instrument. In fig. B.1 we put in context the selected maps in a Mars topographical representation. The maps presented in sec. 4.3.3 (fig. 4.8 and 4.10) are located in the Noachis Terra region, corresponding to different areas of a 30-kilometre crater. The first map presented in this appendix (fig. B.2) is an structure of the Leighton crater, at the east of the Schiaparelli crater. The second map (fig B.3) is a region with compositional diversity in northern Hellas region. The last map (fig. B.4) shows the central uplift of the Ritchey crater, also located in the Noachis Terra region.

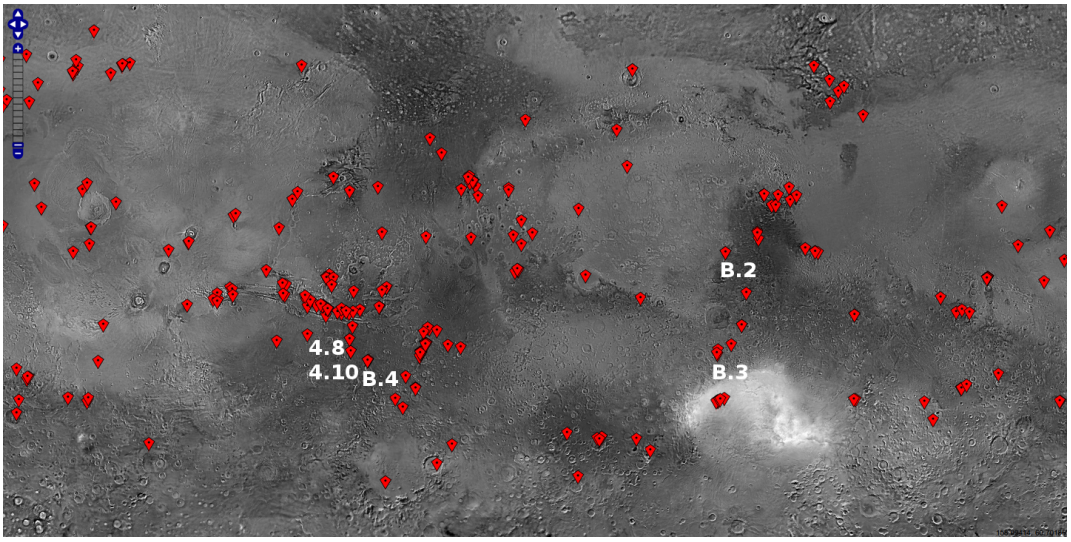


Figure B.1: DTMs available in the HiRISE web as August 2016. The labels identify the figure with the paths obtained by 3Dana for the maps of sec. 4.3.3 and this appendix.

The first map presents a structure in the Leighton Crater[1]. The total area covered is near 36 km^2 with a dimension of 5522 x 3257 nodes. Table B.1 and fig. B.2 present the data for a path between points (1000, 300) and (2300, 5200).

Table B.1: Paths data for DTEED_020492_1830. In bold: best path length plus total turns for each maximum slope.

Alg.	Max. slope	α_w	Length (m)	Turn (°)	Time (s)	Expanded nodes
A*	-	-	11007	14175	270	1246143
3Dana	-	-	**10306**	**516**	557	2058774
A*		-	11015	19035	279	1268072
	30°	0.0	10308	577	804	2001361
3Dana		0.5	10332	275	9685	2096684
		1.0	**10334**	**164**	11559	1920525
A*		-	11021	21555	274	1240419
	25°	0.0	10317	703	809	808631
3Dana		0.5	10336	527	6410	2036773
		1.0	**10347**	**497**	8435	2117518
A*		-	11073	24795	286	2180587
	20°	0.0	10360	1472	1004	2398166
3Dana		0.5	**10384**	**1019**	5302	2347419
		1.0	10399	1161	7235	2340256
A*		-	11187	34560	429	1740510
	15°	0.0	10459	2574	1049	2788701
3Dana		0.5	10504	2348	4653	2751384
		1.0	**10507**	**2169**	6617	2718648
A*		-	No path			
3Dana	10°					

[1] `http://uahirise.org/dtm/dtm.php?ID=ESP_020492_1830`

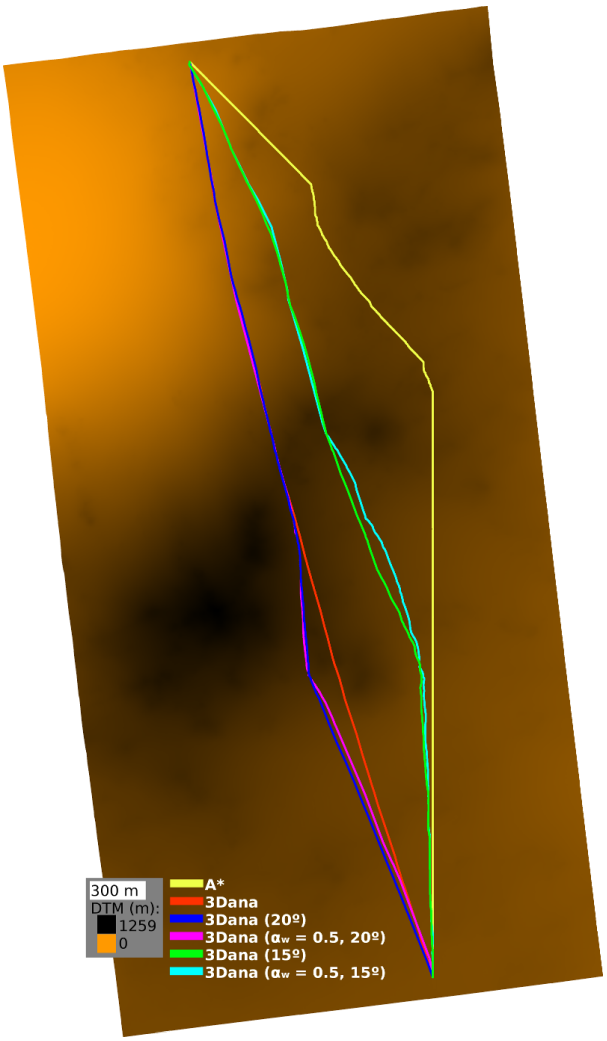


Figure B.2: Paths obtained for the DTEED_020492_1830 using A* and different configurations of 3Dana.

The second map is located in northern of the Hellas region[2]. The total area covered is near 53 km^2 with a dimension of 7955 x 3349 nodes. Table B.2 and fig. B.3 present the data for a path from (1200, 500) to (1800, 7000).

Table B.2: Paths data for DTEED_029815_1530. In bold: best path length plus total turns for each maximum slope.

Alg.	Max. slope	α_w	Length (m)	Turn (°)	Time (s)	Expanded nodes
A*	-	-	13694	24255	923	2806279
3Dana	-	-	**13140**	**592**	694	1956913
A*		-	13677	27180	804	2546422
	30°	0.0	13141	596	1192	1885863
3Dana		0.5	**13150**	**264**	6730	1665626
		1.0	13152	287	7573	1538329
A*		-	13677	23940	768	2426763
	25°	0.0	13144	593	1244	1982619
3Dana		0.5	**13151**	**275**	5695	1565942
		1.0	13152	309	6944	1577497
A*		-	13687	27675	7256	2125253
	20°	0.0	13164	685	1154	1733098
3Dana		0.5	**13174**	**353**	3024	1272223
		1.0	13179	460	2839	1055844
A*		-	14844	42075	1568	4111250
	15°	0.0	13823	1839	1950	2406437
3Dana		0.5	13862	1902	4428	2412699
		1.0	**13974**	**1460**	5457	2544489
A*		-	16141	45450	1788	6020368
	10°	0.0	**15305**	**2195**	4608	6621195
3Dana		0.5	15478	2533	11322	6983322
		1.0	Timeout		>5h	

[2] http://uahirise.org/dtm/dtm.php?ID=ESP_029815_1530

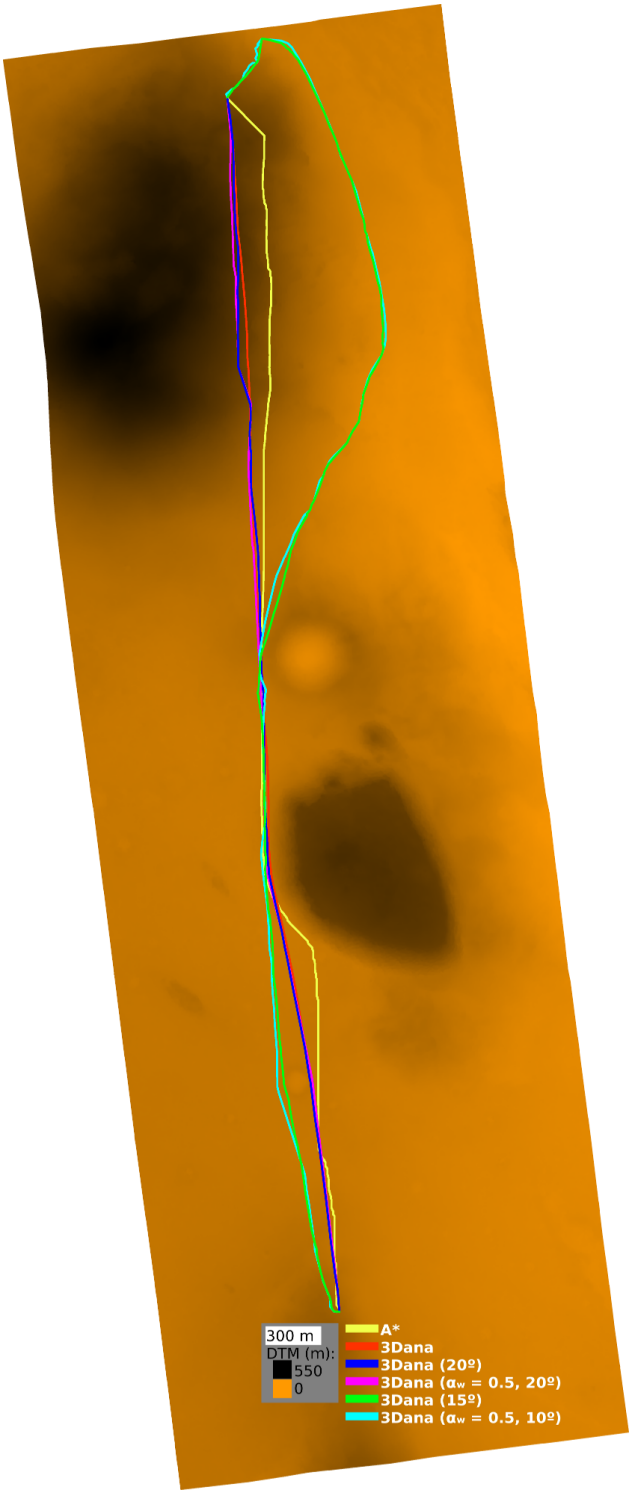


Figure B.3: Paths obtained for the DTEED_029815_1530 using A* and different configurations of 3Dana.

Finally, the third map shows the central uplift of the Ritchey Crater[3]. The total area covered is near 50 km^2 with a dimension in of 7758 x 3242 nodes. Table B.3 and fig. B.4 present the data for a path between points (1500, 1500) and (2000, 7000).

Table B.3: Paths data for DTEED_029964_1510. In bold: best path length plus total turns for each maximum slope.

Alg.	Max. slope	α_w	Length (m)	Turn (°)	Time (s)	Expanded nodes
A*	-	-	11499	19980	150	1103194
3Dana	-	-	**11133**	**964**	455	1796831
A*		-	11522	21330	157	1107588
	30°	0.0	11133	985	659	1764564
3Dana		0.5	**11147**	**304**	4044	1615257
		1.0	11149	357	4550	1431735
A*		-	11539	23040	164	1104231
	25°	0.0	11136	1305	679	1784577
3Dana		0.5	11153	667	3444	1709972
		1.0	**11148**	**478**	4647	1677322
A*		-	11616	26910	181	1138413
	20°	0.0	11168	1539	927	2136595
3Dana		0.5	11185	960	4116	2114850
		1.0	**11200**	**792**	4975	2030527
A*		-	11911	34020	530	2333058
	15°	0.0	11375	2578	1321	3011790
3Dana		0.5	**11447**	**2152**	4265	3127010
		1.0	11490	2283	5503	3984846
A*		-	12746	53325	1202	4577942
	10°	0.0	12129	9826	2209	5266664
3Dana		0.5	**12277**	**8166**	7031	5834310
		1.0	12372	6465	8976	5683667

[3] `http://uahirise.org/dtm/dtm.php?ID=ESP_029964_1510`

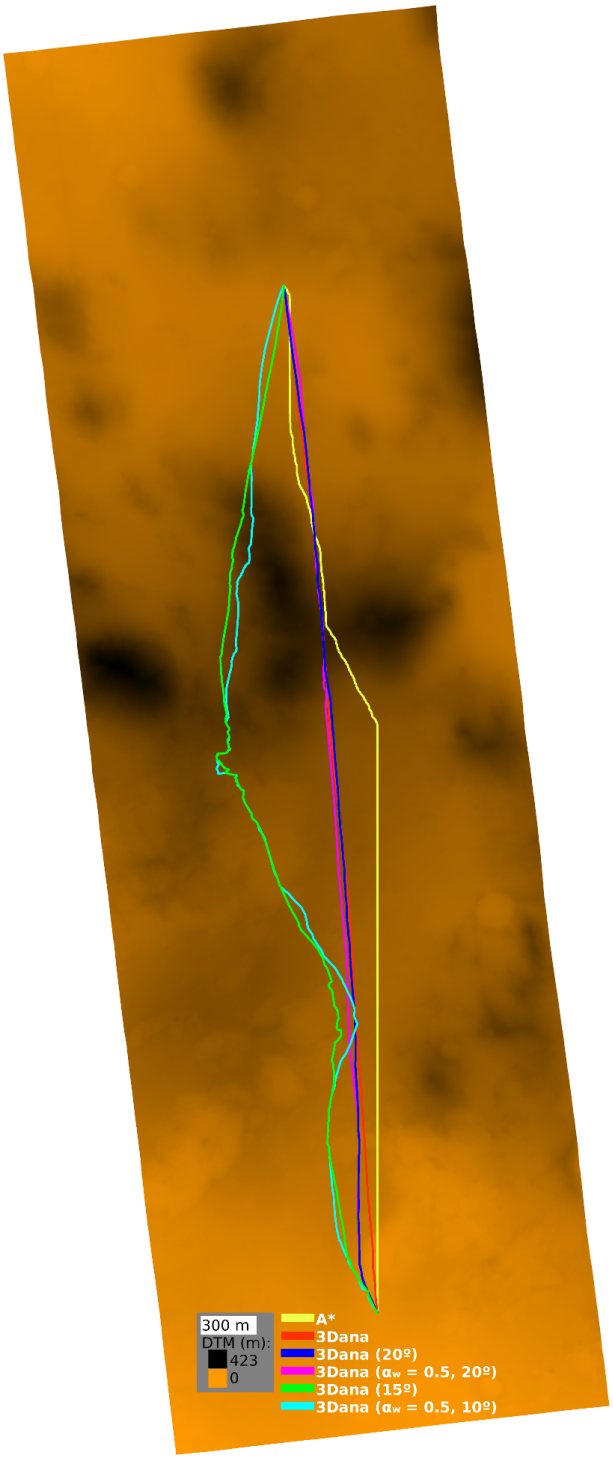


Figure B.4: Paths obtained for the DTEED_029964_1510 using A* and different configurations of 3Dana.

Bibliography

[1] Ad Hoc Autonomy Levels for Unmanned Systems Working Group. Autonomy levels for unmanned systems (ALFUS) framework – volume I: Terminology. Technical Report 1011-I-2.0, National Institute of Standards and Technology, October 2008.

[2] P. E. Agre and D. Chapman. What are plans for? *Robotics and Autonomous Systems*, 6:17–34, 1989.

[3] M. Akinc, K. Bekris, B. Chen, A. Ladd, E. Plaku, and L. Kavraki. Probabilistic roadmaps of trees for parallel computation of multple query roadmaps. *Algorithmic Foundations of Robotics VI, Springer Tracts in Advanced Robotics*, 14:80–89, 2005.

[4] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *Field Robotics, Special Issue on Integrated Architectures for Robot Control and Programming*, 17:315–337, 1998.

[5] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Quin. An integrated theory of the mind. *Psycological Review*, 111(4):1036–1060, 2004.

[6] R. C. Arkin. Navigational path planning for a vision-based mobile robot. *Robotica*, 7(1):49–63, 1989.

[7] R. C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6(1-2):105–122, 1990.

[8] R. C. Arkin and T. Balch. AuRA: Principles and practice in review. *Experimental and Theoretical Artificial Intelligence*, 9(2-3):175–189, 1997.

[9] R. C. Arkin and D. C. MacKenzie. Temporal coordination of perceptual algorithms for mobile robot navigation. *IEEE Transactions on Robotics and Automation*, 10(3):276–286, 1994.

[10] P. Aschwanden, V. Baskaran, S. Bernardini, C. Fry, M. D. R-Moreno, N. Muscettola, C. Plaunt, D. Rijsman, and P. Tompkins. Model-unified planning and execution for distributed autonomous system control. In *Association for the Advancement of Artificial Intelligence (AAAI) 2006 Fall Symposia*, Washington DC, USA, October 2006.

[11] G. Ayorkor, A. Stentz, and M. B. Dias. Continuous-field path planning with constrained path-dependent state variables. In *ICRA 2008 Workshop on Path Planning on Costmaps*, Pasadena, CA, USA, May 2008.

[12] T. Balch. Teambots. Online: www.teambots.org.

[13] A. Basu, M. Bozga, and J. Sifakis. Modeling hterogeneous real-time components in BIP. In *Procs. of the 4th IEEE International Conference on Software Engineering and Formal Methods*, Washington DC, USA, September 2006.

[14] S. Behnke. Robot competitions – ideal benchmarks for robotics research. In *2006 IEEE/RSJ International Conference on Robots and Systems (IROS) Workshop on Benchmarks in Robotics Research*, Beijing, China, October 2006.

[15] J. Benton, A. Coles, and A. Coles. Temporal planning with preferences and time-dependent continuous costs. In *Procs. of the 22nd International Conference on Automated Planning and Scheduling*, Atibaia, São Paulo, Brazil, may 2012.

[16] D. Bernard, G. Doraist, E. Gamble, B. Kanefskyt, J. Kurien, G. K. Man, W. Millart, N. Muscettola, U. Nayak, K. Rajant, N. Rouquette, B. Smith, W. Taylor, and Y. wen Tung. Spacecraft autonomy flight experience: The DS1 Remote Agent experiment. In *Procs. of the AIAA Space Technology Conference and Exposition*, Albuquerque, NM, USA, September 1999.

[17] R. P. Bonasso. Integrating reaction plan and layered competences through syncrhonous control. In *Procs. of the 12th International Joint Conference on Artificial Intelligence*, Sydney, New South Wales, Australia, August 1991.

[18] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack. Experiences with an architecture for intelligent reactive agents. *Experimental and Theoretical Artificial Intelligence*, 9(2):187–202, 1997.

[19] A. Botea, M. Muller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1:1–22, 2004.

[20] J. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4:25–30, 1965.

[21] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[22] R. A. Brooks. *How to Build Complete Creatures Rather than Isolated Cognitive Simulators*. Lawrence Erlbaum Assosiates, Hillsdale, NJ, 1991.

[23] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[24] M. Bualat, L. Edwards, T. W. Fong, M. Broxton, L. Flueckiger, S. Y. Lee, E. Park, V. To, H. Utz, V. Verma, C. Kunz, and M. MacMahon. Autonomous robotic inspection for lunar surface operations. In *Procs. of the 6th International Conference on Field and Service Robotics*, Chamonix, France, July 2007.

[25] L. M. Camarinha-Matos. Plan generation in robotics: State of the art and perspectives. *Robotics*, 3(3-4):291–328, 1987.

[26] S. Cambon, R. Alami, and F. Gravot. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research*, 28(1):104–126, 2009.

[27] J. G. Carbonell, O. Etzioni, Y. Gil, R. Joseph, C. Knoblock, S. Minton, and M. Veloso. PRODIGY: An integrated architecture for planning and learning. *SIGART Bulletin*, 2(4):51–55, 1991.

[28] A. Ceballos, S. Bensalem, A. Cesta, L. D. Silva, S. Fratini, F. Ingrand, J. Ocón, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. V. Winnendael. A goal-oriented autonomous controller for space exploration. In *Procs. of the 11th Symposium on Advanced Space Technologies in Robotics and Automation*, Noordwijk, the Netherlands, April 2011.

[29] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Developing an end-to-end planning application from a timeline representation framework. In *Procs. of the 21st Innovative Applications of Artificial Intelligence Conference*, Pasadena, CA, USA, July 2009.

[30] A. Cesta and A. Oddi. *DDL1: A Formal Description of a Constraint Representation Language for Physical Domains*. IOS Press, Amsterdam, 1996.

[31] S. Choi, J. Y. Lee, and W. Yu. Fast any-angle path planning on grid maps with non-collision pruning. In *Procs. of the IEEE International Conference on Robotics and Biomimetics*, Tianjin, China, December 2010.

[32] S. Choi, J. Park, E. Lim, and W. Yu. Global path planning on uneven elevation maps. In *Procs. of the 9th International Conference on Ubiquitous Robots and Ambient Intelligence*, Daejeon, Korea, Nov 2012.

[33] R. J. Clark, R. C. Arkin, and A. Ram. Learning momentum: On-line performance enhancement for reactive systems. In *Procs. of the IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

[34] J. H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Procs. of the IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

[35] J. W. Crandall and M. A. Goodrich. Measuring the intelligence of a robot and its interface. In *Performance Metrics for Intelligent Systems (PerMIS'03) Workshop*, Gaithersburg, MD, USA, September 2003.

[36] K. Daniel, A. Nash, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[37] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint network. *Artificial Intelligence*, 49:61–95, 1991.

[38] A. P. del Pobil. Why do we need benchmarks in robotics research? In *2006 IEEE/RSJ International Conference on Robots and Systems (IROS) Workshop on Benchmarks in Robotics Research*, Beijing, China, October 2006.

[39] DFKI Robotics Innovation Center. Rock: the robot construction kit. http://rock-robotics.org.

[40] M. B. Dias, S. Lemai, and N. Muscettola. A real-time rover executive based on model-based reactive planning. Technical Report 169, Robotics Institute, 2003.

[41] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[42] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. L. Christensen, A. Decugniere, G. A. D. Caro, F. Ducatelle, E. Ferrante, A. Forster, J. Guzzi, V. Longchamp, S. Magnenat, J. M. González, N. Mathews, M. A. M. de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Rétornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stuetzle, V. Trianni, E. Tuci, A. E. Turgut, and F. Vaussard. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 2012.

[43] M. Dorigo, E. Tuci, V. Trianni, R. GroB, S. Nouyan, C. Ampatzis, T. H. Labella, R. O'Grady, M. Bonani, and F. Mondada. *Swarm-Bot: Design and Implementation of Colonies of Self-assembling robots*, chapter 6, pages 106–135. Computational Intelligence: Principles and Practice. IEEE Computational Intelligence Society, NY, 2006.

[44] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras. Combining high-level casual reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Procs. of the IEEE International Conference on Robotics and Automation*, Shangai, China, May 2011.

[45] K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Procs. of the National Conference on Artificial Intelligence*, Seattle, WA, USA, July 1994.

[46] ESA. Exomars rover, October 2015. http://exploration.esa.int/mars/45084-exomars-rover.

[47] ESA-ESTEC Requirements & Standards Division. Space engineering: Space segment operability. Technical Report ECSS-E-ST-70-11C, European Coordination for Space Standardization, Noordwijk, The Netherlands, July 2008.

[48] D. Ferguson and A. Stentz. Field D*: An interpolation-based path planner and replanner. In *Procs. of the International Symposium on Robotics Research*, October 2005.

[49] J. A. Fernández-Madrigal and J. L. Blanco. *Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods*. IGI Global, 2012.

[50] N. B. Figueroa, F. Schmidt, H. Ali, and N. Mavridis. Joint origin identification of articulated robots with marker-based multi-camera optical tracking systems. *Robotics and Autonomous Systems*, 61(6):580–592, 2013.

[51] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem-proving to problem-solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[52] R. J. Firby. An investigation into reactive planning in complex domains. In *Procs. of the 6th National Conference on Artificial Intelligence*, Seattle, DC, USA, July 1987.

[53] S. Fleury, M. Herrb, and A. Mallet. *GenoM: User's Guide*. CNRS; LAAS, May 2010.

[54] L. Flückiger and H. Utz. Service oriented robotic architecture for space robotics: Design, testing, and lessons learned. *Journal of Field Robotics, Special Issue on Space Robotics*, 31(1):176–191, January 2014.

[55] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1992.

[56] G. Fontana, M. Matteucci, and D. G. Sorrenti. RAWSEEDS: Building a benchmarking toolkit for autonomous robotics. In F. Amigoni and V. Schiaffonati, editors, *Methods and Experimental Techniques in Computer Engineering*, Springer Briefs in Applied Sciences and Technology, pages 55–68. Springer International Publishing, 2014.

[57] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *AI Research*, 20:61–124, 2003.

[58] S. Fratini, F. Pecora, and A. Cesta. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

[59] A. Garcia, A. Barrientos, A. Medina, P. Colmenarejo, L. Mollinedo, and C. Rossi. 3D path planning using a fuzzy logic navigational map for planetary surface rovers. In *Procs. of the 11th Symposium on Advanced Space Technologies in Robotics and Automation*, Noordwijk, The Netherlands, May 2011.

[60] E. Gat. ALFA: A language for programing reactive robotic control systems. In *Procs. of the IEEE International Conference on Robotics and Automation*, Sacramento, CA, USA, April 1991.

[61] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Procs. of the 10th National Conference on Artificial Intelligence (AAAI)*, pages 809–815, San Jose, CA, USA, July 1992.

[62] E. Gat. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Working notes of the AAAI Fall Symposium on Plan Execution*, Cambridge, MA, USA, November 1996.

[63] E. Gat. *On Three-Layer Architectures*. AAAI Press, 1998.

[64] A. Gerevini and D. Long. Plan constraints and preferences in PDDL3. In *Procs. of the 5th International Planning Competition*, Italy, 2005.

[65] D. I. Gertman, C. McFarland, T. A. Klein, A. E. Gertman, and D. J. Bruemmer. A methodology for testing unmanned vehicle behavior and autonomy. In *Performance Metrics for Intelligent Systems (PerMIS'07) Workshop*, Washington, D.C. USA, August 2007.

[66] A. K. Goel, K. S. Ali, M. W. Donnellan, A. G. de Silva Garza, and T. J. Callantine. Multistrategy adaptive path planning. *IEEE Expert*, 9(6):57–65, 1994.

[67] Y. Guou, Y. Long, and W. Sheng. Globlal trajectory generation for nonholonomic robots in dynamic environments. In *Procs. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Rome, Italy, April 2007.

[68] K. Z. Haigh and M. M. Veloso. Interleaving planning and robot execution for asynchronous user requests. *Autonomous Robots - Special Issue on Autonomous Agents*, 5(1):79–95, 1998.

[69] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[70] K. Hauser and J. Latombe. Integrating task and PRM motion planning: Dealing with many infeasible motion planning queries. In *Procs. of the 9th International Conference on Automated Planning and Scheduling*, Thessaloniki, Greece, September 2009.

[71] J. Hoffman. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[72] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Artificial Intelligence Research*, 14:253–302, 2001.

[73] C. Hsu and B. Wah. The SGPlan planning system in IPC-6. In *In Procs. of the 6th International Planning Competition*, Sydney, Australia, September 2008.

[74] H.-M. Huang, E. Messina, and A. Jacoff. Performance measures framework for unmanned systems (PerMFUS): Initial perspective. In *Performance Metrics for Intelligent Systems (PerMIS'09) Workshop*, Gaithersburg, MD, USA, September 2009.

[75] H.-M. Huang, E. Messina, A. Jacoff, R. Wade, and M. McNair. Performance measures framework for unmanned systems (PerMFUS): Models for contextual metrics. In *Performance Metrics for Intelligent Systems (PerMIS'10) Workshop*, Baltimor, MD, USA, September 2010.

[76] H.-M. Huang, K. Pavek, J. Albus, and E. Messina. Autonomy levels for unmanned systems (ALFUS) framework: An update. In *SPIE Defense and Security Symposium*, Orlando, FL. USA, May 2005.

[77] A. R. Hudson and L. H. Reeker. Standardizing measurements of autonomy in the Artificially Intelligent. In *Performance Metrics for Intelligent Systems (PerMIS'07) Workshop*, Washington, D.C. USA, August 2007.

[78] F. Ingrand, R. Chatila, R. Alami, and F. Robert. PRS a high level supervision and control language for autonomous mobile robots. In *Procs. of the IEEE International Conference on Robotics and Automation*, Minneapolis, USA, April 1996.

[79] F. Ingrand, M. P. Georgeff, and A. S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(4):34–44, 1992.

[80] F. Ingrand, S. Lacroix, S. Lemai-Chenevier, and F. Py. Decisional autonomy of planetary rovers. *Field Robotics*, 24(7):559–580, 2007.

[81] F. Ingrand and F. Py. An execution control system for autonomous robots. In *Procs. of the IEEE International Conference on Robotics and Automation*, Washington DC, USA, May 2002.

[82] B. Innocenti, B. López, and J. Salvi. Design patterns for combining social and individual intelligences on modular-based agents. In *3rd International Workshop on Hybrid Artificial Intelligence Systems*, Salamanca, Spain, June 2008.

[83] B. Innocenti, B. López, and J. Salvi. Integrating individual and social intelligence into module-based agents without central coordinator. *Frontiers in Artificial Intelligence and Applications*, 179(1):82–93, 2008.

[84] International Electrotechnical Commission (IEC). Performance evaluation method of intelligent mobile robot platform for household and similar applications. Technical report, International Electrotechnical Commission (IEC), 2012.

[85] G. Ishigami, K. Nagatani, and K. Yoshida. Path planning for planetary exploration rovers and its evaluation based on wheel slip dynamics. In *Procs. of the IEEE International Conference on Robotics and Automation*, Roma, Italy, April 2007.

[86] L. Jaillet, J. Cortés, and T. Siméon. Transition-based RRT for path planning in continuous cost spaces. In *Procs. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, September 2008.

[87] A. K. Jonsson, P. H. Morris, N. Muscettola, K. Rajan, and B. D. Smith. Planning in interplanetary space: Theory and practice. In *Procs. of the 5th International Conference on Artificial Intelligence Planning Systems*, Breckenridge, Colorado, USA, April 2000.

[88] L. P. Kaelbling. Goals as parallel program specifications. In *Procs. of the National Conference on Artificial Intelligence*, Saint Paul, MN, USA, August 1988.

[89] M. Kanehara, S. Kagami, J. Kuffner, S. Thompson, and H. Mizoguhi. Path shortening and smoothing of grid-based path planning with consideration of obstacles. In *Procs. of the IEEE International Conference on Systems, Man and Cybernetics*, Montreal, QC, Canada, October 2007.

[90] R. L. Kirk, E. Howington-Kraus, M. R. Rosiek, J. A. Anderson, B. A. Archinal, K. J. Becker, D. A. Cook, D. Galuszka, P. E. Geissler, T. M. Hare, I. M. Holmberg, L. P. Keszthelyi, B. L. Redding, W. A. Delamere, D. Gallagher, J. Chapel, E. M. Eliason, R. King, and A. S. McEwen. Ultrahigh resolution topographic mapping of Mars with MRO HiRISE stereo images: Meter-scale slopes of candidate Phoenix landing sites. *Journal of Geophysical Research: Planets*, 113(E3):1–31, 2008.

[91] K. Konolige and K. Myers. The Saphira architecture for autonomous mobile robots. Technical report, MIT Press, November 1996.

[92] P. Laborie and M. Ghallab. Planning with sharable resource constraints. In *Procs. of the International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

[93] J. E. Laird. Extending the Soar cognitive architecture. In *Procs. of the Conference on Artificial General Intelligence*, Memphis, Tennessee, USA, March 2008.

[94] A. Lampe and R. Chatila. Performance measure for the evaluation of mobile robot autonomy. In *Procs. of the IEEE International Conference on Robotics and Automation*, Orlando, FL, USA, May 2006.

[95] P. Langley. Cognitive architectures and general intelligent systems. *AI Magazine*, 27(33):33–44, 2006.

[96] P. Langley and D. Choi. A unified cognitive architecture for physical agents. In *Procs. of the 21st National Conference on Artificial Intelligence*, Boston, MA, USA, July 2006.

[97] P. Langley, D. Choi, and S. Rogers. Interleaving learning, problem solving, and execution in the ICARUS architecture. Technical report, Computational Learning Laboratory, Standford University, 2005.

[98] S. M. LaValle and J. J. Kuffner Jr. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[99] C. Lebiere and J. R. Anderson. A connectionist implementation of the ACT-R production system. In *Procs. of the 15th Annual Conference of the Cognitive Science Society*, Colorado, CO, USA, June 1993.

[100] S. Lemai. *IxTeT-eXeC: Planning, Plan Repair and Execution Control with Time and Resource Management.* PhD thesis, LAAS-CNRS and Insitut National Polytecnique de Tolouse, France, 2004.

[101] P. Lima, D. Nardi, G. Kraetzschmar, J. Berghofer, M. Matteucci, and G. Buchanan. RoCKIn innovation through robot competitions. *IEEE Robotics & Automation Magazine*, 21(2):8–12, 2014.

[102] M. Lindström, A. Orebäck, and H. Christensen. BERRA: A research architecture for service robots. In *Procs. of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, USA, may 2000.

[103] C. L. López, S. Jiménez, and M. Helmert. Automating the evaluation of planning systems. *AI Communication*, 26(4):331–354, 2013.

[104] A. Mallet, S. Fleury, and H. Bruyninckx. A specification of generic robotics software components: future evolutions of GenoM in the Orocos context. In *Procs. of the International Conference on Intelligent Robotics and Systems*, Lausanne, Switzerland, september 2002.

[105] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, and F. Ingrand. GenoM3: Building middleware-independent robotic components. In *Procs. of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, May 2010.

[106] D. McDermott. The PDDL planning domain definition language. *The AIPS-98 Planning Competition Comitee*, 1998.

[107] D. McDermott. PDDL: the planning domain definitin language. *AI Magazine*, 21(2):35–55, 2000.

[108] G. T. McWilliams, M. A. Brown, R. D. Lamm, C. J. Guerra, P. A. Avery, K. C. Kozak, and B. Surampudi. Evaluation of autonomy in recent ground vehicles using the autonomy levels for unmanned systems (ALFUS) framework. In *Performance Metrics for Intelligent Systems (PerMIS'07) Workshop*, Washington, D.C. USA, August 2007.

[109] D. P. Miller. *Planning by Search Through Simulations.* PhD thesis, Yale University, 1985.

[110] I. Millington and J. Funge. *Artificial Intelligence for Games.* Morgan Kaufmann Publishers, 2 edition, 2009.

[111] B. G. Milnes, G. Pelton, R. Doorenbos, M. Hucka, J. E. Laird, P. Rosenbloom, and A. Newell. A specification of the Soar cognitive architecture in Z. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[112] N. Muscettola. *HSTS: Integrating Planning and Scheduling.* Intelligent Scheduling. Morgan Kaufmann, 1994.

[113] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams. Remote Agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103:5–48, 1998.

[114] P. Muñoz, D. F. Barrero, and M. D. R-Moreno. Run-time analysis of classical path-planning algorithms. In *Procs. of the 32nd SGAI International Conference on Artificial Intelligence*, Cambridge, UK, December 2012.

[115] P. Muñoz, D. F. Barrero, and M. D. R-Moreno. Statistic methods for path-planning algorithms comparison. *Künstliche Intelligenz*, 27(3):201–211, 2013.

[116] P. Muñoz, D. F. Barrero, and M. D. R-Moreno. A statistically rigorous analysis of 2D path-planning algorithms. *The Computer Journal*, 58(11):2876–2891, 2014.

[117] P. Muñoz, B. Castaño, and M. D. R-Moreno. Simulation of the hexapod robot PTinto walking on irregular surfaces. *International Journal of Simulation Modelling*, 14:1–12, March 2015.

[118] P. Muñoz, A. Cesta, A. Orlandini, and M. D. R-Moreno. Toward a test environment for autonomous controllers. In *5th Italian Workshop on Planning and Scheduling*, Turin, Italy, December 2013.

[119] P. Muñoz, A. Cesta, A. Orlandini, and M. D. R-Moreno. First steps on an on-ground autonomy test environment. In *Procs. of the 5th International IEEE Conference on Space Mission Challenges for Information Technology*, Maryland, USA, September 2014.

[120] P. Muñoz, A. Cesta, A. Orlandini, and M. D. R-Moreno. Evaluating autonomous controllers: An initial assessment. In *6th Italian Workshop on Planning and Scheduling*, Ferrara, Italy, September 2015.

[121] P. Muñoz, A. Cesta, A. Orlandini, and M. D. R-Moreno. A framework for performance assessment of autonomous robotic controllers. In *Proc. of the 2nd Workshop on Planning and Robotics (ICAPS-15)*, Jerusalem, Israel, June 2015.

[122] P. Muñoz, A. Cesta, A. Orlandini, and M. D. R-Moreno. The on-ground autonomy test environment: OGATE. In *Procs. of the 13th ESA Workshop on Advanced Space Technologies for Robotics and Automation*, Noordwijk, The Netherlands, May 2015.

[123] P. Muñoz and M. D. R-Moreno. Improving efficiency in any-angle path-planning algorithms. In *Procs. of the 6th IEEE International Conference on Intelligent Systems*, Sofia, Bulgaria, September 2012.

[124] P. Muñoz and M. D. R-Moreno. S-Theta*: Low steering path-planning algorithm. In *Procs. of the 32nd SGAI International Conference on Artificial Intelligence*, Cambridge, UK, December 2012.

[125] P. Muñoz and M. D. R-Moreno. Cooperative systems in mission planning. In *Procs. of the 12th ESA Workshop on Advanced Space Technologies for Robotics and Automation*, Noordwijk, The Netherlands, May 2013.

[126] P. Muñoz and M. D. R-Moreno. Deliberative systems for autonomous robotics: A brief comparison between action-oriented and timelines-based approaches. In *Procs. of the 1st Workshop on Planning and Robotics (ICAPS-13)*, Rome, Italy, June 2013.

[127] P. Muñoz and M. D. R-Moreno. Model-Based Architecture on the ESA 3DROV simulator. In *Procs. of the 23rd ICAPS Application Showcase*, Rome, Italy, June 2013.

[128] P. Muñoz and M. D. R-Moreno. *On Heading Change Measurement: Improvements for Any Angle Path-Planning*, chapter 6. Novel Applications of Intelligent Systems, April 2015.

[129] P. Muñoz, M. D. R-Moreno, and D. F. Barrero. Unified framework for path-planning and task-planning for autonomous robots. *Robotics and Autonomous Systems*, 82:1–14, 2016.

[130] P. Muñoz, M. D. R-Moreno, and B. Castaño. Integrating a PDDL-based planner and a PLEXIL-executor into the PTinto robot. In *Procs. of the 23rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: Next-Generation Applied Intelligence*, Córdoba, Spain, june 2010.

[131] P. Muñoz, M. D. R-Moreno, and B. Castaño. 3Dana: Path planning on 3D surfaces. In *Procs. of the 36th SGAI International Conference on Artificial Intelligence*, Cambridge, UK, December 2016.

[132] P. Muñoz, M. D. R-Moreno, and A. Martínez. A first approach for the autonomy of the Exomars rover using a 3-Tier architecture. In *Procs. of the 11th ESA Workshop on Advanced Space Technologies for Robotics and Automation*, Noordwijk, The Netherlands, April 2011.

[133] P. Muñoz, M. D. R-Moreno, A. Martínez, and B. Castaño. Fast path-planning algorithms for future Mars exploration. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Turin, Italy, September 2012.

[134] A. Nash, K. Daniel, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. In *Procs. of the 22nd AAAI Conference on Artificial Intelligence*, British Columbia, BC, Canada, July 2007.

[135] A. Nash, S. Koening, and M. Likhachev. Incremental Phi*: Incremental any-angle path planning on grids. In *Procs. of the International Joint Conference on Artificial Intelligence*, Pasadena, CA, USA, July 2009.

[136] N. J. Nilsson. A mobile automation: an application of artificial intelligence techniques. In *Procs. of the 1st International Joint Conference on Artificial Intelligence*, Washington, D.C, USA, May 1969.

[137] N. J. Nilsson. *Principles of Artificial Intelligence*. Palo Alto: Tioga, 1980.

[138] N. J. Nilsson. Shakey the robot. Technical report, AI Center, SRI International, Menlo Park, CA, USA, April 1984.

[139] B. Nystrom. Hill algorithm. http://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html.

[140] Open Source Robotics Foundation. Gazebo robot simulation. http://gazebosim.org.

[141] Open Source Robotics Foundation. Open-source robot development kit for apps on wheels. http://turtlebot.com.

[142] A. Orebäck and H. I. Christensen. Evaluation of architectures for mobile robotics. *Journal of Autonomous Robots*, 14:33–49, 2003.

[143] D. L. Page, A. F. Koschan, and M. A. Abidi. Ridge-valley path planning for 3D terrains. In *Procs. of the IEEE International Conference on Robotics and Automation*, Orlando, FL, USA, May 2006.

[144] R. Parasuraman, T. B. Sheridan, and C. D. Wickens. A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 30(3):286–297, May 2000.

[145] J. Plaza, M. D. R-Moreno, B. Castaño, M. Carbajo, and A. Moreno. PIPSS: Parallel integrated planning and scheduling system. In *27th Annual Workshop of the UK Planning and Scheduling Special Interest Group*, Edinburgh, UK, December 2008.

[146] P. Poulakis, L. Joudrier, S. Wailliez, and K. Kapellos. 3DROV: A planetary rover system design, simulation and verification tool. In *Procs. of the 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Hollywood, CA, USA, February 2008.

[147] F. Py and F. Ingrand. Dependable execution control or autonomous robots. In *Procs. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, September 2004.

[148] F. Py, K. Rajan, and J. McGann. A systematic agent framework for situated autonomous systems. In *Procs. of the 9th International Conference on Autonomous Agents and Multiagent Systems*, Toronto, Canada, May 2010.

[149] Y. Pyo, K. Nakashima, S. Kuwahata, R. Kurazume, T. Tsuji, K. Morooka, and T. Hasegawa. Service robot system with an informationally structured environment. *Robotics and Autonomous Systems*, 74, Part A:148–165, 2015.

[150] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, May 2009.

[151] M. D. R-Moreno, B. Castaño, M. Carbajo, A. Moreno, D. F. Barrero, and P. Muñoz. Multi-agent intelligent planning architecture for people location and orientation using RFID. *Cybernetics and Systems*, 42(1):16–32, 2011.

[152] M. D. R-Moreno, A. Cesta, and J. Kurien. Innovative AI technologies for future ESA missions. In *Procs. of the 10th Symposium on Advanced Space Technologies in Robotics and Automation*, Noordwijk, the Netherlands, November 2008.

[153] S. Rockel, B. Neuman, J. Zhang, K. S. R. Dubba, A. G. Cohn, Š. Konečný, M. Mansouri, F. Pecora, A. Saffiotti, M. Günther, S. Stock, J. Hertzberg, A. M. Tomé, A. J. Pinho, L. S. Lopes, S. von Riegen, and L. Hotz. An ontology-based multi-level robot architecture for learning from experiences. In *Designing Intelligent Robots: Reintegrating AI II, AAAI Spring Symposium*, Stanford, CA, USA, March 2013.

[154] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2009.

[155] K. R. Scherer. *Appraisal Considered as a Process of Multilevel Sequential Checking*, chapter Appraisal Processes in Emotion: Theory, Methods, Research. Oxford Universty Press, 2001.

[156] F. Schneider, D. Wildermuth, and H.-L. Wolf. ELROB and euRathlon: Improving search & rescue robotics through real-world robot competitions. In *10th International Workshop on Robot Motion and Control (RoMoCo)*, Poznan, Poland, July 2015.

[157] H. K. B. Selman and J. Hoffman. Satplan: Planning as satisfiability. In *Abstracts of the 5th International Planning Competition (IPC-5), hosted at the International Conference on Automated Planning and Scheduling (ICAPS)*, Cambria, UK, June 2006.

[158] G. Shaffer and A. Stentz. A robotic system for underground coal mining. In *Procs. of the IEEE International Conference on Robotics and Automation*, Nice, France, May 1992.

[159] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, 1994.

[160] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Procs. of the IEEE/RSJ International Conference on In Intelligent Robots and Systems*, Victoria, B.C., Canada, October 1998.

[161] R. Simmons and C. Fedor. Experience with a task control architecture for mobile robots. Technical report, Carnegie Mellon University, December 1989.

[162] R. Simmons and E. Krotkov. An integrated walking system for the Ambler planetary rover. In *Procs. of the IEEE International Conference on Robotics and Automation*, Sacrament, CA, USA, April 1991.

[163] A. K. Singh, K. M. Krishna, and S. Saripalli. Planning non-holonic stable trajectories on uneven terrain through non-linear time scaling. *Autonomous Robots*, pages 1–22, 2015.

[164] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Procs. of the IEEE International Conference on Robotics and Automation*, Hong Kong, China, June 2014.

[165] Z. Sun and J. H. Reif. On finding energy-minimizing paths on terrains. *IEEE Transactions on Robotics*, 21:102–114, 2005.

[166] C. E. Thorpe and L. H. Matthies. Path relaxation: Path planning for a mobile robot. In *Procs. of the OCEANS Conference*, Washington D.C., USA, September 1984.

[167] M. Veloso, J. G. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Experimental & Theoretical Artificial Intelligence*, 7(1):81–120, January 1995.

[168] V. Verma, V. Baskaran, H. Utz, and C. Fry. Demonstration of robust execution on a NASA lunar rover testbed. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Hollywood, USA, 2007.

[169] V. Verma, A. Jónsson, C. Pasareanu, and M. Iatauro. Universal Executive and PLEXIL: Engine and language for robust spacecraft control and operations. In *American Institute of Aeronautics and Astronautics Space Conference*, San Jose, CA, USA, september 2006.

[170] D. Vernon, G. Metta, and G. Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE Transactions on Evolutionary Computation*, 11(2):151–180, April 2007.

[171] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The CLARAty architecture for robotic autonomy. In *Procs. of the IEEE Aeroespace Conference*, Big Sky, MT, USA, March 2001.

[172] R. Volpe, I. A. D. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. CLARAty: Coupled layer architecture for robotic autonomy. Technical report, JPL, December 2000.

[173] D. Wettergreen, H. Thomas, and C. Thorpe. Planning strategies for the Ambler walking robot. In *Procs. of the IEEE International Conference on Systems Engineering*, Pittsburgh, PA, USA, August 1990.

[174] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Procs. of the 30th National Conference on Artificial Intelligence (AAAI)*, Portland, OR, USA, August 1996.

[175] J. Wolfe, B. Marthi, and S. Russell. Combined task and motion planning for mobile manipulation. In *Procs. of the 10th International Conference on Automated Planning and Scheduling (ICAPS)*, Toronto, Canada, May 2010.

[176] M. Woods, A. Shaw, D. Barnes, D. Price, D. Long, and D. Pullan. Autonomous science for an ExoMars rover-like mission. *Field Robotics*, 26(4):358–390, 2009.

[177] M. J. Woods, L. Baldwin, G. Wilson, S. Hall, A. Pidgeon, D. Long, M. Fox, R. Aylett, and R. Vituli. MMOPS: Assessing the impact of on-board autonomy for deep space missions. In *Procs. of the SpaceOps Conference*, Rome, Italy, June 2006.

[178] P. Yap. Grid-based path-finding. In *Advances in Artificial Intelligence*, volume 2338 of *Lecture Notes in Computer Science*, pages 44–55. Springer Berlin / Heidelberg, 2002.

[179] F. Zacharias, C. Borst, and G. Hirzinger. Bridging the gap between task planning and path planning. In *Procs. of the IEEE International Conference on Robotics and Systems*, Beijing, China, October 2006.