Multilayered cognitive control for Unmanned Ground Vehicles

Dipartimento di Ingegneria Informatica, Automatica e Gestionale
ANTONIO RUBERTI
Dottorato di Ricerca in Ingegneria Informatica – XXV Ciclo

Candidate
Mario Gianni
ID number 792264

Thesis Advisor
Prof. Fiora Pirri

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Ingegneria Informatica
March 2014
Thesis defended on 26 September 2014
in front of a Board of Examiners composed by:
Prof. Paolo Ciancarini (chairman)
Prof. Andrea D’Ambrogio
Prof. Marco Montali

Multilayered cognitive control for Unmanned Ground Vehicles
Ph.D. thesis. Sapienza – University of Rome
ISBN: –
© 2014 Mario Gianni. All rights reserved

This thesis has been typeset by \LaTeX and the Sapthesis class.

Version: –
Author’s email: gianni@dis.uniroma1.it
Dedicated to myself
Acknowledgements

I would like to express my sincere and genuine gratitude to my advisor professor Fiora Pirri, whose patience, competence, care and immense knowledge have been really important for my research progress.
I also thank to my parents. They continuously believed in me. I am grateful to them for their important support. Mom and Dad, thank you.
Special thanks go to my colleagues, for creating an amazing work environment which I am proud to join every morning.
My last note of thanks is for my friends. I am grateful to them for encouraging me to keep going and for helping me unwind.
Abstract

Rescue robots have the potentials to assist responders in searching for survivors, in rescuing victims, in providing the responders with a general situation awareness, in creating a reference of the destroyed environment, in sampling suspicious substances from hazardous sites, in navigating through those areas, inaccessible for humans. In the last decades, rescue robots participated in many of the most critical environmental disasters around the world, exhibiting extraordinary abilities in terms of mapping, vision and navigation. In June 2012, at Mirandola, a city of Northern Italy, hit by a tremendous earthquake, we deployed a team of humans and robot to assess damage to historical buildings and cultural artifacts located therein. This in-field experience has been really important because it led us to a better understanding of what are the main research challenges which are not yet widely addressed in rescue robotics. The research work of this thesis aims to investigate, in more detail, some of these challenges, providing solutions and methodological approaches to the research problems, still opened. In particular, we address the problem of building a meaningful, higher level representation of unstructured and dynamic environments, from raw data, coming from the robot sensor suite. We also take care of how to formulate this representation into a domain where decision making and action planning can take place. We tackle with the problem of learning the skills required for a robot to perform a rescue task and formulating such skills into robot actions and plans. Here, the novelty is to use a wearable device, namely, the Gaze Machine (GM), to address the correspondence issues between the physical embodiments of the firefighter, wearing the GM, and the robot. Further, this thesis investigates the problem of increasing the level of autonomy of the robot, in low-level, semi-active and cognitive control. In low-level control, we propose an approach to design and develop a controller, which endows the robot with the ability to autonomously traverse harsh terrains, climbing stairs, surmounting obstacles, adapting the configuration of the robot to the underlying surfaces. In semi-active control, we propose an approach to coordinate the low-level capabilities of the robot and the interaction between the human and robot, under a mixed-initiative planning setting. In this approach the main components and activities of the robot are explicitly represented as well as the cause-effect relations and the temporal constraints among the activities. This control model is based on a logical framework which combines temporal constraint reasoning and action planning. This framework provides us with a solid logical structure on which to build the set of cognitive functions of the robot. Such functions endows the robot with the ability to flexibly adapt its behavior in response to environmental demands and stimuli. To model this ability, we propose a method for learning the dynamic processes regulating the human inspired paradigm of shifting and inhibition, underlying the task switching mechanism. Finally, this thesis proposes an alternative view of Augmented Reality, as a framework to augment the perceptual model of the robot as well as to build mixed-reality simulation environments, where to validate the performance of the robot, in terms of vision, motion planning and control.
Contents

1 Introduction .............................................. 1
   1.1 Challenges in understanding unstructured environments ....... 2
   1.2 Challenges in learning new skills .................................... 3
   1.3 Challenges in low-level autonomous control ......................... 5
   1.4 Challenges in semi-autonomous control ............................... 7
   1.5 Challenges in robot cognitive control ............................... 8
   1.6 Challenges in designing and evaluating robot behaviours .......... 9
   1.7 Contributions and thesis structure .................................. 11

2 Rescue Robots at Earthquake-Hit Mirandola, Italy: a Field Report 17
   2.1 Sites .................................................... 18
   2.2 Deployment .............................................. 19
       2.2.1 System & network infrastructure .............................. 19
       2.2.2 UGV ................................................... 20
       2.2.3 UAV .................................................... 21
       2.2.4 Human-robot teaming ...................................... 21
   2.3 Experience ................................................. 23
       2.3.1 Human-robot teaming ...................................... 23
       2.3.2 UGV .................................................... 25
       2.3.3 UAV .................................................... 27

3 A higher level representation of the robot low-level perception for reasoning and action planning 29
   3.1 Topological Segmentation of Metric Map ............................. 30
   3.2 Object Detection and Annotation of Topological Graph .............. 32
   3.3 Dialogue ................................................... 34
   3.4 The Logic-based Robot Control ..................................... 35

4 Terrain Traversability in Rescue Environments: 39
   4.1 Proposed Methodology ........................................... 40
       4.1.1 3D space structure ......................................... 40
       4.1.2 Terrain parameters ......................................... 43
       4.1.3 Traversability and cost Map .................................. 45
   4.2 Results ..................................................... 47
## Help me help you: how to learn intentions, actions and plans

### 5.1 Visual Localization 51

### 5.2 Segmentation and Action Maps 52

### 5.3 From actions to flexible plans and plan recognition 54

## An Augmented Reality approach for trajectory planning and control of tracked vehicles in rescue environments

### 6.1 The robot model 57

### 6.2 Robot Localization System 57

### 6.3 Trajectory tracking control 58

### 6.4 Trajectory planning and generation 59

### 6.5 Experiments 61

## Real-Time 3D Motion Planning and Control for Autonomous Navigation

### 7.1 3D motion planning 65

#### 7.1.1 Trajectory tracking controller 66

#### 7.1.2 Flippers position controller 67

### 7.2 Contact sensor model 68

#### 7.2.1 Data-set collection and labeling 68

#### 7.2.2 Feature representation 69

#### 7.2.3 Learning the parameters of $h_B$ 69

### 7.3 Point cloud segmentation and labeling 70

### 7.4 Feasible Path 73

### 7.5 Experiments 74

## Learning the dynamic processes of shifting and inhibition in cognitive control

### 8.1 Executive Summary 81

### 8.2 The stimuli model 82

### 8.3 The stimulus-response model 85

### 8.4 States, processes and goals 91

### 8.5 Task Selection 96

### 8.6 Implementation 98

#### 8.6.1 Robot functionalities 98

#### 8.6.2 Task switching interactive interface 99

### 8.7 Experiments 100

#### 8.7.1 Outliers rejection evaluation 101

#### 8.7.2 Performance of the stimuli model 101

#### 8.7.3 Generalization ability of the recommendation system 102

#### 8.7.4 Correctness of the stimuli-response model 102

#### 8.7.5 Effectiveness of the rule-based task selection model 103

## ARE: Augmented Reality Environments for mobile robots

### 9.1 The AR-based simulation framework 106

#### 9.1.1 The real environment representation 106

#### 9.1.2 The Artefacts model 106

#### 9.1.3 The AR-builder server 108

### 9.2 Experimental results 110
Contents

10 Conclusions 113

Bibliography 117
Chapter 1

Introduction

Urban Search & Rescue (USAR) robots play a crucial role in assisting first disaster responders. Such robots can go where rescue workers are not allowed [41]. They can provide responders with a preliminary assessment of the conditions of the disaster area in which they are deployed. Moreover, they might make decisions and consequently act so as to facilitate the rescue of the victims, got involved in an environmental disaster [40]. Further, compared with rescue workers and trained rescue dogs, rescue robots have advantages. Unlike their human counterparts, rescue robots will not become stressed or fatigued [29]. Rescue robots can be developed in large quantities, while experienced rescue professionals and trained rescue dogs are sparse resources [41]. Robots are expendable but humans and rescue dogs are not. Robots can be easily repaired or replaced, but the loss of rescue workers or dogs could be very difficult due to their relationship within society [40].

In the last decade, robots participated in the rescue and recovery operations of many past and recent devastations, such as the 2001 World Trade Center (WTC) collapse [183], the 2004 Mid Niigata earthquake in Japan [188], the 2005 Hurricanes Katrina, Rita, and Wilma in the United States [188], as well as the 2011 Tohoku earthquake and tsunami in Japan [111, 110].

In May 2012, two major earthquakes occurred in the Emilia-Romagna region, Northern Italy, followed by further aftershocks and earthquakes in June 2012. This sequence of earthquakes and shocks caused multiple casualties, and widespread damage to numerous historical buildings in the region. Although the Italian National Fire Corps deployed disaster response and recovery of people and buildings, some of the main devastated churches and historical buildings were not accessible even by firefighters. It was necessary an assessment of the state of risks. To this end, we deployed, in the red-area of Mirandola, Emilia-Romagna, a team of humans and robots to assess damage to historical buildings, and cultural artifacts located therein [144, 146].

Although robots have exhibited promising potential in USAR applications, the experience, at Mirandola, at a disaster scenario of highest realism, re-marked what still are the limitations of the actual robots for search and rescue, in terms of perception, decision making, autonomous navigation and control. To cope with these limitations new research challenges have to be addressed not only in rescue robotics but also in other research fields, such as machine perception, cognitive control and neuroscience.
1.1 Challenges in understanding unstructured environments

The problem of understanding unstructured and dynamic environments mainly concerns with (i) extracting relevant features from raw data, coming from the robot sensors; (ii) building a meaningful, higher level representation of the extracted information and, (iii) formulating such a representation into a domain where both reasoning and decision making can take place [43][205]. An accurate 3D representation of the environment, based on 3D point cloud data registration, coming from rolling laser sensors, has been described in [211]. This representation can be efficiently used by an operator for navigating disaster areas, but does not provide a semantic structure of the environment. An approach to build a richer metric representation of the environment, based on 3D Occupancy Grid mapping, has been proposed in [121]. This representation provides information about occupied, free and unknown space within the environment. The distinction between free and occupied space facilitates safe robot navigation, while information about unknown areas supports autonomous exploration. However, this approach provides a binary interpretation of obstacles, overly simplifying the complex interactions between the robot and the environment [23][219]. In fact, obstacles can be either surmount or avoided, depending by both the model and the kinematic constraints of the robot. Moreover, stairs or ramps can be identified to provide the robot with information about the accesses to the several levels of the environment. Building such a meaningful interpretation of the environment, on top of the low-level perception of the robot, is a challenging task. Recently, several authors have considered the problem of adding semantic information to regions [233][179][226][176]. In [179], the authors address the problem of classifying regions of the environment of a mobile robot, using range finder data. Regions are labeled by classifying the position of the robot in the environment, on the basis of simple geometric features, extracted from laser observations. In [176] Morisset and colleagues proposed an interesting approach to build a polygonal model of the environment, annotated with semantic labels, using disparity images, from a stereo camera attached to the robot. The polygonal model is obtained by efficiently merging local planar patches, representing the surfaces which better approximate the point cloud data, into a global one. After merging the global model, each polygon is labeled according to rules that account for local geometrical constraints. The main advantage of the mentioned approaches is to facilitate the formulation of the perceived knowledge about the environment into a suitable logic representation that can be maintained into the knowledge base of the robot. Moreover, semantic labeling specifies the interaction modes between the robot and the environment. Although the approaches to semantic mapping, based on either point cloud segmentation or model fitting, work well for both structured and semi-structured environments, where stairs or walls are clearly recognizable, they fail to provide a suitable representation of unstructured environments, such as a completely collapsed building. Modeling the characteristics of a cluttered disaster scenario requires real-time methods for terrain surface reconstruction. Indeed, terrain reconstruction is fundamental for both traversability assessment and morphological adaptation of self-reconfigurable tracked robots since it provides the real topology of the terrain. The topological graph of the terrain directly enables graph search methods to plan paths over the surface [23]. Terrain surface reconstruction has been widely approached with both laser data [253][164] and video sequences [256][33][262][52]. The main limitation of these approaches is the management of occlusion effects, strongly affecting the reconstruction [256]. To cope with this limitation, several approaches, based on depth maps merging, have been proposed [21][154]. Another
1.2 Challenges in learning new skills

Skill learning refers to the ability of humans to identify important behavioral components, retain them as skills, refine them through practice, and apply them in new task contexts [59, 74, 264]. Skill learning underlies both the human ability to choose to spend time and effort to specialize at particular tasks, and the ability to collect and exploit previous experience to cope, over time, with harder and harder problems, with less cognitive effort. In robotics, as well as in Artificial Intelligence and in cognitive science, skill learning has been widely studied with the aim to provide a theoretical basis to design intelligent systems, more specifically robots, capable to autonomously acquire new skills and to refine existing ones [130, 246, 127, 105, 119, 14, 12]. These studies have led to the formulation of two main methodologies for robot skill learning: (1) Learning from Demonstration (LfD) [191, 108] and (2) Reinforcement Learning (RL) [112, 241, 128, 268, 251]. In LfD-based paradigms, learning consists of behavior demonstration, followed by policy derivation from the recorded dataset via machine learning techniques. Many candidate representations for the learned behavior policy exist; one common representation is a mapping from observations of world state to robot actions. Policy development via demonstration has many advantages, including the generation of focused datasets and intuitive use by human developers [5]. LfD-based approach has been applied to a variety of behaviors, at control levels ranging from low to high. In [191], Nicolescu and colleagues described an approach for teaching robots, similar to the one people use when teaching each other. After a preliminary demonstration of the task, provided by the teacher, the robot can refine the acquired capabilities by practising under the teacher’s supervision. Depending on the quality of the learned task, the teacher can either demonstrate it again or provide specific feedback during the practice trial of the robot, for further refinement. Moreover, the teacher can provide the robot with simple instructions and informative cues, through speech, thus increasing the performance of learning. This feedback allows the robot to learn a graph-based representation of the demonstrated task. However, the correspondence issues between the physical embodiments of the teacher and the robot constitute the main limitation of the approaches based on LfD. Still, LfD techniques fail to maximize the generalizability of the learned skill to unseen situations. In [62], Dong and Williams proposed an approach for learning robot motions from observations of human performance, focused on determining acceptable variance in execution and generalization over conditions and perturbations. In this approach, relevant variables of a particular human motion are automatically determined through statistically analyzing demonstrated sequences. The idea behind this identification process is that the variables relevant to the motion are preserved over different demonstrated trials, while other variables of the motion can vary due to changes in the environment or the human’s movement. During autonomous execution of the demonstrated task in a new environment, the robot extracts those states that are relevant, based on the identified motion variables. Given the demonstrated data sequences, the robot identifies those relevant states that are similar to the states, extracted by the new environment.
Dynamic time warping is applied to both temporally match the new states with the observed ones and create a probabilistic flow tube representation of the human-generated motion trajectories. Such a representation enables the robot to optimize additional performance criteria or recover from disturbances. An alternative solution to the problem of generalizing the learned skills to new environments has been proposed in [32]. The main idea behind this approach is to provide the robot with the ability to make queries to the teacher, to retrieve more information about the demonstrated task. The aim of robotic systems asking questions is to reduce uncertainties in task demonstration and to mitigate unintended variances between the demonstrations given to different teachers. However, humans, as teachers, could not provide successful examples of executed tasks, during demonstration. In the mentioned LfD-based approaches, data from failed human task demonstrations are just discarded. In [109], the authors argued that it is possible to build models, from failed demonstrations, that can guide a robot system to discover a successful way to perform a novel task. In fact, learning from failed demonstrations has educational worth in at least three respects: (i) failed demonstrations specify what not to do, so replication should be avoided; (ii) they are indicative of what the human thinks a successful performance should be, so new attempts should explore around them and, (iii) multiple attempts indicate an appropriate breadth of exploration. Therefore, learning from failed demonstrations leads to a trade-off between the quality of the learned robot controller, the skill level of the demonstrator, the number of demonstrations and the time spent for learning [108]. RL mainly concerns with the problem of learning to maximize a numerical reward signal over time in a given environment. As a robot interacts with its environment, it receives a scalar reward for each action taken, and its goal is to learn to act so as to maximize the cumulative reward it receives over time [246]. The basic idea is that actions correlated with high reward have higher repeated probability, while those correlated with low reward have lower repeated probability. In the last decade, RL has been broadly applied to address mobile robot navigation and obstacle avoidance problems in structured environments, because of its self-learning and on-line learning abilities [241, 149, 268, 128]. Although RL is an effective tool to implement mobile robot reactive navigation, it fails to meet real-time requirements when the complexity of the learning problem increases. In particular, when applied into navigation area, the computational complexity exponentially increases, as the domain of the problem is inflated by an increase of states and actions. Moreover, learning by trial-and-error, when the robot interacts with the environment, may lead to risk for navigation system [260]. In [268], the authors proposed an approach, based on RL and rough sets, for autonomous navigation in unknown environments, which both reduces risks of learning by trial-and-error and improves the obstacle avoidance capability of the navigation system of the robot. In [128], Jeni and colleagues described an RL-based navigation framework which uses a hierarchical state space decomposition to speed-up the learning process, in large-scale robotic domains. Unlike LfD, both traditional and hierarchical RL techniques have been used to develop robots capable to learn optimal behavior in search and rescue applications [65, 159, 272]. In [65], Doroodgar and Nejat implemented a MAXQ hierarchical RL method within the context of semi-autonomous control for both robot exploration and victim identification problems. The objective was to allow the robot to learn, from its own previous experiences and those of a human operator, how to effectively perform tasks in USAR environments. The controller provides the robot with the ability to make decisions regarding which rescue tasks have to be carried out, at a given time, and whether another team member (e.g., an autonomous robot or a human controlled robot) can perform these tasks more quickly and efficiently without
1.3 Challenges in low-level autonomous control

compromising the safety of the victims, rescue workers and the rescue robot. By using a MAXQ approach, the overall search and rescue task, is decomposed into a finite set of smaller more manageable subtasks that can be concurrently learned. These subtasks define the MAXQ hierarchy of the overall task. The method supports state, temporal and subtasks abstraction. Temporal abstraction favours the flexibility of the execution of the actions of the different sub-tasks. State abstraction specifies what is relevant for the execution of a task, thus reducing the number of state variables that have to be taken in to account in the learning process. Further, subtask abstraction reduces the learning time as well as facilitates sub-task sharing [159]. In [272] the authors proposed an RL-based algorithm to learn the motor skills required to autonomously adapt the morphology of a self-reconfigurable tracked vehicle, in terrain traversability tasks. This algorithm takes as input the incoming sensory data (e.g., such as laser data, motor torque signals, inertial data, tracks odometry) and outputs a robot control mode, defined by either a binary speed decision (stop or go) or a combination of discrete flipper configuration and their stiffness (hard or soft). Control modes are selected automatically in such a way to maximize the expected sum of discounted rewards, defined by a reward function. This function is defined as a weighted sum of (i) user denoted reward, reflecting robot safety, (ii) high tilt angles penalty, negatively rewarding the control mode, (iii) excessive flipper mode change penalty, (iv) robot forward speed reward and, finally, (v) motion roughness penalty. Despite the many impressive skills exhibited by the actual rescue robots, the general approach to the generation of such behaviors has shown little changes over the last decades. The roboticists model the task dynamics as accurately as possible while using human insight, in order to create the desired robot behavior, as well as to eliminate all uncertainties of the environment. In most cases, such a process boils down to recording a desired behavior in an environment suitably designed. Such highly engineered approaches are feasible in particular environmental conditions or in research environments. However, to overcome this limitation, the strong reliance on models of both the environment and the robot, needs to be reduced. Recently, the rise of machine learning approaches, such as those mentioned above, has led to significant results in robot skill learning. However, such approaches are not yet adequate for robot skill learning as these methods (i) do not rely on an understanding of behavioral systems (ii) do not scale into the high-dimensional domains and, (iii) do not fulfil the real-time requirement of the domain. To cope with the complexities, involved in robot skill learning, and with the inherent problems of task representation, learning and execution should be addressed separately in a coherent framework employing a combination of imitation, reinforcement and model learning. This framework is needed to allow robots to learn their tasks with minimal programming and in less structured, uncertain environments [138],[199]. Finally, this framework requires the support of a suitable interface, enabling both human-robot interaction and collaboration, for the acquisition of the data needed for learning, from the human experience.

1.3 Challenges in low-level autonomous control

To date, the majority of rescue robots used in the field are tele-operated [183],[111],[146]. Under this control mode, search and rescue operations require a human operator in the loop, in order to remotely guide a robot. The main limitation of such a control is that a human operator effectively has perceptual difficulties in trying to understand a highly cluttered environment via remote visual feedback. Therefore, he/she can become disoriented, thus
losing situation awareness. The loss of situation awareness is the main reason why rescue robots frequently got stuck in the rubble-filled environment [41]. Moreover, remote visual perception does not allow human operators to clearly understand whether the robot could pass through openings or traverse over obstacles, due to either scale ambiguities or key-hole effects [267]. Research efforts have been made to improve the level of autonomy of rescue robots. 3D simultaneous localization and mapping (SLAM) based algorithms have been developed, from both laser and camera sensor data, to allow the robot to autonomously map a disaster area and localize its own position within the physical environment [148, 70, 176, 211]. Such SLAM systems, even though are quite accurate, are not sufficient to face all the navigation difficulties of an harsh environment due to the presence of stairs, ramps, collapsed walls or rubbles. Therefore, controlling the robot to either climb the stairs or to traverse an harsh terrain can be very demanding for a human operator. Mourikis and colleagues in [178] proposed a vision-guided approach for autonomous stair climbing. This approach relies on the determination of the rotational velocity of the robot, using a 3-axial gyroscope, and of the location of the stair edges, with a 2D camera, to estimate both the heading and the pose of the robot, relative to the stair boundaries. A two-tired controller, consisting of both a centering and a heading module, is used to guide the robot upstairs. Steplight and colleagues in [243] developed a hierarchical control structure to allow a tracked vehicle to autonomously climb stairs. The controller is based on an over-arching broker which estimates the heading angle of the robot from different sensors (e.g., visual and acceleration sensors) and sends this information to a proportional-derivative (PD) path-tracking controller. The PD controller is responsible of generating the suitable steering commands to drive the robot upstairs. In [192] the authors described a sensor-based autonomous sub-track controller to manage swingable sub-tracks of rescue robots, which are used to negotiate steps and uneven terrain. The controller allows an operator to manually control the main tracks, while the autonomous controller is used for the sub-tracks. LIDAR sensors on a robot provide terrain information to the controller for position control of the sub-tracks. The mentioned approaches do not cope with the wide variety of terrain surfaces which the robot has to traverse, in terms of control. Indeed, in many cases, classification in terms of slopes, ditches or sand, pavement, mud or gravel might not be feasible, as in a disaster area the terrain dramatically changes and most often it cannot be sorted out in terms of visual features. Further, vision-based approaches are affected by both bright lighting and dusty conditions [146]. However, a crucial problem is to prevent the robot from both tip and roll overs, in the process of traversing harsh terrains. One promising solution is to use self-reconfigurable tracked robots. Still, reconfiguring such robots for different terrains is a big burden for an operator, who has limited camera views [192]. Autonomous tip-over prediction and prevention algorithms have recently been developed for tracked vehicles, to address this issue [158, 155]. For example, in [155], the authors proposed an on-line control method for the stair-climbing of both tracked and isomerism-modules robots. On the basis of the kinematics analysis of the stair-climbing process, the coordination of the motions of the modules reduce the slippage between the tracks and the stairs, increasing the control efficiency of the stair-climbing. Further, the stair-climbing and the avoiding interference criterions, based on the force analysis for the motion planning and the identification for the stair, help the robot to predict the climbing capacity before the remaining stages, thus enhancing the safety of the robot in the stair-climbing process [155]. Although the proposed solutions for tip-over prediction and prevention are very promising, they have not been yet tested in disaster scenarios. Moreover, the effectiveness of these control methods strictly depends by the feedback information
coming from either proximity or contact sensors, embedded into the robot articulations. When such sensors are not available, terrain modeling is a pre-requisite to prevent the robot from instabilities [160, 262]. Autonomous stair-climbing, traversability analysis, as well as terrain modeling are important milestones for rescue robots, towards autonomy in both 3D motion planning and navigation [176, 23, 49]. In [176], the authors described a system for real-time 3D mapping and motion-planning for RHex mobile robots [228]. The motion planner uses an alternative decomposition technique to convert a 3D model into locally 2D regions. On the basis of the classification of the regions, a multi-region planning technique is used to generate paths toward target poses. Breitenmoser and colleagues in [23] proposed a method for navigating inspection robots on curved surfaces, modeled by triangle meshes. Each mesh represents an embedded graph, enabling graph search methods to plan paths of triangle strips over the surface. Discrete path planning and continuous robot control are combined to steer the robot smoothly along the triangle strip to the target regions on the surface. The authors clearly mention what are the main limitations of the proposed approaches, that is, the sensitivity to both the representation of the environment and the presence of dynamic obstacles [49].

1.4 Challenges in semi-autonomous control

Semi-autonomous control allows for task sharing between a robot and an operator [25, 259, 64]. Under this setting, the robot focuses on low level tasks, such as terrain traversing, whereas the human operator is in charge of high-level control and supervisory tasks [79, 192, 198]. However, some approaches to semi-autonomous control do not allow operators to choose the level of autonomy of the robot, according to the task at hands, neither before deployment nor on the fly, that is, during deployment [236, 198]. These approaches mainly lack the flexibility needed to control the robot in cluttered environments. When the robot gets physically stuck, the operator is not able to take the control of the low level operations in order to assist the robot. To overcome this limitation, Bruemmer and colleagues, in [25], proposed a semi-autonomous controller which allowed an operator to set four different control modes: (1) tele-mode; (2) safe-mode; (3) shared-mode and, (4) auto-mode. In tele-mode, the operator manually controls the robot motion. In safe-mode, both navigation and object detection tasks are managed by the operator, while collision avoidance is performed by the robot. In shared-mode, the robot is responsible of generating optimal paths for navigation, given the target positions, instructed by the operator. In auto-mode, the operator is in charge of high-level task, such as defining a point to navigate to or searching a selected region, while the robot manages navigation and obstacle avoidance. Such control mode effectively minimizes the workload of the operator in time-critical disaster scenes. However, this control schema does not allow the operator to change the level of autonomy of the robot on the fly, during a search and rescue operation [182]. An interesting approach to semi-autonomous control with dynamic adjustment of the level of autonomy of the robot has been described in [79, 36]. In this approach, the control system is responsible of coordinating the interventions of the human operator and the low level robot activities, under a mixed-initiative planning setting. More precisely, the control system is endowed with a declarative model of the activities of the robot, specified in the Temporal Flexible Situation Calculus (TFSC) [78]. The model explicitly represents the main components and activities of the robot, the cause-effect relationships as well as the temporal constraints among the
activities. Further, the model integrates a representation of the activities of the human operator, enabling the control system to supervise his/her operations. A reactive planning engine (i) monitors the consistency of the robot and operator activities, with respect to the model, managing failures and, (ii) incrementally generates plans, allowing the operator to locally assess the robot operations. The designed control schema endows the robot with some hybrid operative modalities lying between autonomous and teleoperated modes. Each of these modes is determined by the way in which the operator interacts with the control system. As an example, the human operator can manually control some functional activities of the robot, scheduled by the reactive planning engine (e.g., the operator can take the control of the motion, suspending the autonomous navigation task of the robot, to explore an interesting location or escape from difficult environments). Still, the operator can modify the control sequence produced by the reactive planning engine by skipping some tasks or inserting new operations. In such a case, the engine validates the operator choices, recovering from misalignments between the expected control plan and the state of the robot. This control system has been tested with a mobile robot in the National Institute of Standard Technology (NIST) yellow arena [248]. The results showed that the semi-autonomous control, under mixed-initiative, improved the performance of the robot, enhancing both the operator situation awareness and human-robot interaction [66, 183, 11]. Despite the mentioned approaches to semi-autonomous control resulted to be very effective in minimizing the workload of the operator, they are still far from being really considered trustworthy by firefighters. Trust is a crucial aspect for effective teamwork, especially in disaster situations [9, 122, 89]. The main reason why humans still do not fully rely on the autonomous capabilities of the actual rescue robots is the lack of an intuitive explanation of the robot behaviors, in task execution. Further, the lack of standard operating procedures, specifying the policies of intervention in presence of robots, as well as the roles of both robots and rescue workers, in disaster response, affects the process of engagement/disengagement of each rescue unit in task allocation.

1.5 Challenges in robot cognitive control

Robots are expected to operate, under disaster circumstances, by continuously tailoring the information, coming from both internal and external perception, into rules or goal-oriented behaviors, in a coherent manner. These abilities are well-known to exists in biological systems, especially in humans. Indeed, humans have the capacity to receive and process enormous amount of sensory information from the environment, exhibiting integrated sensorimotor associations [87]. Humans have the ability to organize the resources necessary for the task at hands, selecting and maintaining the required information to avoid disruption from other influences [86]. Moreover, they have the ability to selectively respond to stimuli or to inhibit inappropriate urges, preserving focus on the current task [174, 171, 8, 44, 250]. Designing a control system for rescue robots, capable of identify stimuli, from the amount of raw data, coming from sensors, and flexibly adapt robot behaviors in response to environmental demands, is a challenging task. Unfortunately, classical AI approaches of robot design do not provide compelling techniques for modeling such a control [222]. Let us consider, as an example, the problem of deciding when the robot should recharge the battery. A standard approach in the literature and in commercial robots is to set a fixed threshold and recharge whenever the energy supply of the robot drops
1.6 Challenges in designing and evaluating robot behaviours

Programming complex robotic systems, operating in dynamic and unpredictable environments, is a challenging task. The cognitive roboticists are more and more exposed to hard problems requiring a great amount of possible real world situations that are difficult to predict, when most of the tests require to go beyond laboratory experiments. Moreover, as the complexity of the robotic system is inflated by an increase of robot components and
1. Introduction

... functionalities, testing the whole set of interconnections between hardware and software components becomes exponentially difficult. Under this perspective, Augmented Reality (AR) has become a compelling technology providing cognitive robotics modellers with a development tool for constructing, at real-time, complex planning scenarios for robots, eliminating the need to model the dynamics of both the robot and the real environment as it would be required by whole simulation environments (e.g., Player/Stage/Gazebo [254], GaTAC [242], MAPSIM [24], V-REP [83], the DSHELL spacecraft simulators [13]). AR Research has been active in robot applications such as maintenance [190], manual assembly [173] and computer-assisted surgery [233]. One of the first robotic applications of AR was in telerobotic control [170]. In these applications operators were provided with visual feedback of a virtual wire-frame robot superimposed over an actual physical robot located at its remote working environment [221]. The virtual robot executed a task for evaluation by the operator and the task was transferred to the real robot if it was satisfactory. AR has been applied also in robots monitoring applications [3, 26]. In these applications, views of both the real robot and the environment were synthesized graphically from on-board sensory data, such as camera images, and presented to remote operators, increasing their situation awareness. Bischoff and Kazi [18] reported an AR-based human robot interface for applications with industrial robots. Their work focused on visualizing work-flows that could help inexperienced users to cope with complex robot operations and programming tasks. AR has also been used to program painting robots [200]. In this application, a spray gun, hand-held by a user, is tracked and the corresponding user movements are converted into robot programs. Giesler et al. [101] applied AR for prototyping warehouse transport robotic systems. This application enabled the user to interactively construct a topological map of paths and nodes in the warehouse by walking around and pointing at the floor. The robot was instructed to move along the map and the planned path was visualized to the user as an augmentation of the real world view. The user inspected possible intersecting points with workers paths, thus obtaining an estimation of the efficiency of the solution. Stilman et al. [244] created an AR-based simulation environment for decoupled testing of robot subsystems. Results from motion planning and vision algorithms are visualized over real world images provided by both external cameras and a camera mounted on the robot. Collett and MacDonald [50] applied ARE for debugging robot applications. They overlaid virtual information such as robot sensory and internal state data onto real world images of the robot environment, in so providing robot developers with a better understanding of the robot world view. Chong et Al., in [48], introduced an AR-based robot programming (RPAR-II) system to assist users in robot programming for pick and place operations with an human robot interface, enriching the interactions between the operators and the robot. In their system, a virtual robot, which was a replicate of a real robot, performed and simulated task planning processes. Green and colleagues, in [104], proved AR to be a suitable platform for human-robot collaboration. Similarly, Billinghurst et Al., in [16], show that AR allows to share remote views (ego-centric view) visualizing the robot view, to the task space (exo-centric view). Furthermore, they show that AR can provide support for natural spatial dialog by displaying the visual cues necessary for a human and a robot to reach common ground and maintain situation awareness. Billinghurst and colleagues show also that the robot can visually communicate its internal states to its human collaborators, by graphic overlays on the real world-view of the human [15]. The mentioned approaches for both designing and evaluating robotic applications are really very appealing, but, apparently, they do not go beyond the development of interfaces for overlaying virtual objects into the real...
robot scene. Virtual objects can be endowed with simple intelligent behaviors. Such virtual intelligent objects can be perceived by real robots as well as can interact with them. The complexity of the real environment can be increased not only by adding virtual objects, but also by making vary the behavior of the added objects. Still, complex robot behaviors can be designed and evaluated, on the basis of the dynamics of the virtual objects. Under this new perspective, AR-based simulation environments (ARE) constitutes an important research test-bed for robots meeting the needs to test and experiment complex robot behaviors using such a dynamic and rich perceptual domain.

1.7 Contributions and thesis structure

This thesis investigates, in more details, some of the key aspects of the research issues, mentioned in the previous paragraphs. We propose a bottom-up approach to convert raw sensor data into a meaningful higher level perception. We describe how such a perception can lead to the formulation of a model, representative of the knowledge that the robot has about the environment. In addition, we show how, on top of this perception, it is possible to develop both new robot skills and intermediate level models of control. The thesis continues by describing a top-down approach to model the cognitive control of an unmanned ground vehicle, for USAR applications. This control orchestrates the processes running across all the levels of the robot architecture. We propose a novel approach to learn the dynamic processes regulating the human inspired paradigm of shifting and inhibition, underlying the task switching mechanism. The developed cognitive control embodies the learned model of task switching, thus endowing the reactive behaviour of the robot with a proactive behaviour component. The thesis concludes with the description of a simulation framework, based on Augmented Reality (AR), developed with the main purpose of providing cognitive robotics modelers with a tool for constructing, at real-time, complex planning scenarios for robots. The framework builds a world model representation that served as ground truth for training and validating the proposed model for task switching, as well as for evaluating the behaviour of the designed cognitive control, under several simulated environmental contingencies.

• In Chapter 2 we describe the experience gained in the deployment of a human-robot team, under the EU-founded project NIFTi, at the core of sites that are heavily hit by earthquakes at the northern Italy from May until June 2012. Among the distinguishing contributions of this mission, we highlight the challenges as they are encountered within a USAR scenario of highest realism and the means by which the human-robot team in collaboration with the National Fire Corps (CNVVF) managed to jointly and effectively address.

• In Chapter 3 we presents a robotic system that employs high-level control in order to operate in a real-world setting where the main task is human-assisted exploration of an environment. In this system, we integrate multi-modal perception from vision and mapping with a model-based executive control. We also show how the system allows the interaction between the human operator and the robot platform via the dialogue-based communication. In this framework, action planning is performed using a high-level representation of the environment that is obtained through topological segmentation of the metric map and object detection and 3D localization in the map. This representation has the form of a graph where all the information related to the
spatial characteristics of the environment is stored into properties that are annotated to the nodes and the edges of the graph that is used by the planner to generate task-dependent plans. The control system monitors the execution of the action sequences and communicates the status through the dialogue. This work contributes in providing an effective method to (i) build a logical representation of the low-level perception of the robot and, (ii) compile the robot perception into knowledge and, finally, (iii) integrate this knowledge with a model-based executive control. The overall system enables the robot to infer strategies in so generating parametric plans that are directly instantiated from perception.

• In Chapter 4 we propose a new preliminary method for point cloud structuring, leading to the definition of a traversability map labeled with a cost that specifies how far is the considered region from a traversable one. The representation comes with a real-time algorithm that can be used for the safe navigation of a specific robot, according to its own limitations or constraints. Here, by robot constraints, we mean the length, height, weight of the robot, together with its kinematics constraints (in terms of ground mobility). We present results of the method with experiments taken on different scenarios, furthermore we illustrate the pros and cons of relying only on points cloud data set, without resorting to a surface reconstruction. The main contributions of the proposed methodology are dynamic assessment of point cloud levels, in the height direction, and static traversal affordability cost estimation with a new paradigm based on a neighbourhood path philosophy.

• In Chapter 5 we describe a novel framework to learn actions, intentions and plans of a firefighter, executing a rescue task in a training car accident scenario. This framework is based on a model of human-robot collaboration. This model interprets the collaboration as a learning process mediated by the firefighter prompt behaviours and the apprentice collecting information from him to learn a plan. Novel is the use of the Gaze Machine, a wearable devise which can allow to gather and convey visual and audio input from the firefighter while executing a task. This device creates a strong communication between the firefighter and the robot, by allowing an agent to observe, over time, not only what effectively the firefighter is doing and communicating it, but also how he adapts his behaviors, by instantiating with common sense the prescribed laws, that is, those usually regulating his conduct in disaster circumstances. In this work we describe the process through which such a rich information, delivered by the Gaze Machine, is transformed into plans. More specifically, how to obtain a map of the instructor positions and his gaze position, via visual slam and gaze fixations; further, how to obtain an action map from the running commentaries and the topological maps and, finally, how to obtain a temporal net of the relevant actions that have been extracted. The learned structure is then managed by the flexible time paradigm of flexible planning in the Situation Calculus for execution monitoring and plan generation. This work contributes in providing a new approach for robot skill learning. This approach intermediates between action recognition and learning by imitation.

• In Chapter 6 we describe a framework for trajectory planning and control of tracked vehicles for rescue environments, based on Augmented Reality (AR). The framework provides the human operator with an AR-based interface that facilitates both 3D path planning and obstacle negotiation. The interface converts the 3D movements
of a marker pen, handheld by the operator, into trajectories feasible for the tracked vehicle. The framework implements a trajectory tracking controller to allow the tracked vehicle to autonomously follow the trajectories, decided by the operator. This controller relies on a localization system which provides, at real-time, position feedback. The localization system exploits the performance of a Dead Reckoning System together with the accuracy of an ICP-based SLAM in pose estimation, to determine the pose of the tracked vehicle within the 3D map. We demonstrate the application of the planning framework in autonomous robot navigation for evaluating the robot capabilities in rescue environments. The main contribution of this work is to provide an effective solution for closing the loop between perception, motion, planning and control for tracked vehicles. Moreover, an operator is an integral part of such a loop.

- In Chapter 7 we describe a framework for real-time 3D motion planning and control of tracked robots, for autonomous navigation in harsh environments. This framework is composed of three main modules: (1) a semantic mapper, (2) a path builder and, (3) a motion controller. The semantic mapper is responsible of both segmenting and labeling, at real-time, point cloud data, registered by an ICP-based SLAM. This process comprises three main steps: (1) point cloud filtering; (2) estimation of normals to the surface and curvature and (3) clustering and merging the filtered point cloud. Clusters are labeled on the basis of the surface normals, the mean curvature and the points location. The path builder takes as input the labeled clusters and incrementally builds a connectivity graph, according to both the model and the kinematic constraints of the robot. These constraints takes into account the morphology as well as the capabilities of the robot to surmount obstacles. In parallel, the path builder estimates both the boundary and the inflated obstacle regions of the environment. Upon the estimation of the boundary regions, the edges of the connectivity graph are annotated with a weight which depends by the distance of the vertexes from such boundaries, the density of the neighbourhood of the vertexes and the length of the edge. This traversability structure used by the path builder to find minimum cost feasible paths toward target locations. The motion controller makes effective the motions of the tracked vehicle to autonomously accomplish the navigation task. This controller is divided into two decoupled control modules: (1) a trajectory tracking controller and, (2) a flippers position controller. These modules work in parallel and are synchronized so as to generate the control commands needed to track a given 3D path and to simultaneously adapt the position of the flippers to the surfaces on which the path lies, namely the planes tangent at each path point. Since the flippers are neither endowed with contact sensors nor with proximity sensors, the contact sensor estimating the touch of each flipper with the underlying surface has been statistically modeled. The flippers position controller activates these sensors to both correct the estimation of the flippers position command and to ensure that the robot has a better traction on the harsh terrain. The overall framework has been widely tested into three different scenarios: the Italian Fire Fighters Rescue Training area, at Prato (IT), the Fire Escape Stairs of our Department and a full 3D scenario, suitably designed. The latter scenario demonstrates the ability of the robot to navigate along alpha-shaped (non-planar) paths. The main contribution of this work is in the design of the 3D motion controller of the robot which has been proved to be able to cope with all the navigation difficulties,
even when it is not possible to build a structured representation of the environment.

- In Chapter 8, we propose a new approach to robot cognitive control based on modeling robot stimuli, the stimulus-response and the resulting task switching or stimulus inhibition. The proposed framework contributes to the state of the art on robot planning and high level control as it provides a novel view on robot behaviors and a complete implementation on an advanced robotic platform. Our approach is inspired by Gibson’s constructive view of the concept of a stimulus, translated to robots and by the paradigm of executive cognitive functions responding selectively to stimuli. This is the shifting or inhibiting paradigm that either inhibits inappropriate urges, by preserving focus on the current task, or selects the best response, when the stimuli are compelling. In an autonomous system these cognitive skills assess a well-regulated proactive behavior, which is of particular relevance for planning in critical circumstances. In this paper we concentrate on modeling the robot choice of what to do next in the presence of a stimulus, in three stages. We start by defining the stimuli as perceptual functions yielded by the active processes, and learned via an informed logistic regression. Then we model the stimulus-response problem, leading to the selection of a set of possible responses to the current selected stimuli, as a recommendation system. Finally we model a decision rule on the basis of an interference cost, which is the cost of discontinuing the current robot mental state to activate the processes needed for switching to a new task, where this activation is the reconfiguration cost, that takes into account the urgency of the stimulus. The method is validated by several experiments proving both the learning models and the general robot behavior. Because processes play such a crucial role both in the stimuli model and in the stimulus-response model, and because processes are activated by actions, we model processes based on a theory of actions.

- In Chapter 9, we present an AR-based simulation framework which allows robot developers to build on-line an Augmented Reality Environment (ARE) for real robots, integrated into the visualization interface of Robot Operating System (ROS). The system we propose goes beyond an interface for drawing objects, as the design exploits a stochastic model activating the behaviors of the introduced objects. Objects, people, obstacles, and any kind of structures in the environment can be endowed with a behavior. Furthermore, a degree of certainty of their existence and behaviors, with respect to what the robot perceives and knows about its space, can be tuned according to the experiment needs. The framework also builds a world model representation that serves as ground truth for training and validating algorithms for vision, motion planning and control. The main advantages and benefits of ARE for robot design and experimentation, as opposed to a complete simulation framework in scenario design and test, are illustrated in evaluating the capability of the robot to plan safe paths to goal locations in real outdoor scenarios, while the planning scene dynamically changes, being augmented by virtual objects.

Chapters have a uniform structure throughout the thesis: a brief introduction initiates the reader to the dissertation, then the proposed approach is presented, followed by technical results, including discussions. An additional chapter has been included, at the end of the thesis, summarizing the overall research work. Part of this thesis has already been published in [94, 96, 97, 134, 144, 136, 95, 194, 169, 30, 92, 90].
Chapter 2

Rescue Robots at Earthquake-Hit Mirandola, Italy: a Field Report

On May 20 2012, in the middle of the night, northern Italy was hit by an earthquake with epicenter in Finale Emilia, in the region of Emilia Romagna. On May 29 at 09:00 AM local time, a 5.8 magnitude earthquake struck the already damaged area again. Overall, 246 seismic events with magnitudes between 3 and 6.1 occurred from May 20 until June 18 within a radius of 50km of the original epicenter, see Fig. 2.1 and [215], and affected some 900,000 people across six provinces, with a rich cultural heritage.

The National Fire Corps (CNVVF), the Italian Department of firefighters, public rescue and civil defense, has been in charge of disaster response and recovery for the area. The undertaking involves a national effort to ensure search and rescue of people, evacuation of several centers and recovery of valuable works of art, and to secure many of the damaged buildings and surveying them. For intervening in the main devastated churches and historical buildings, some of which were not accessible even by the CNVVF, and for art-works recovery, it was necessary to assess location and state of risks.

This is where NIFTi comes in. NIFTi is an EU-funded project, focusing on human-robot teams for exploring disaster sites in Urban Search & Rescue settings [71]. NIFTi adopts a user-centric approach to developing its models, from autonomous robot behavior (UGV, UAV) to human-robot collaboration, working together with several end user organizations. With the end users NIFTi sets up requirements, experiments with prototypes, and evaluates overall systems performance on a yearly basis. The CNVVF is one of the project partners involving both the Instituto Superiore Antincendi (ISA) and the Scuola di Formazione Operativa (SFO) in Montelibretti. Since the beginning of the project in early 2010, this meant that the CNVVF has built up a close working relationship with the research partners in NIFTi. Having this experience with the systems developed by the project, and knowing the potentials of the work done, the CNVVF has requested NIFTi to aid in structure damage assessment by deploying a team of humans, UGVs, and UAVs in one of the most damaged towns of the whole region, namely Mirandola, by entering the red area and intervening at the Duomo and at the San Francesco church; here, together with his dynasty, Pico della Mirandola is buried. Pico was author of On the Dignity of Man considered the Manifesto of the Renaissance, and known for his prodigious memory.

This chapter describes the experience gained in the deployment, in July 2012 and it is organized as follows. §2.1 describes the sites of Mirandola red area where the team ran
missions. §2.2 outlines the team structure, the robots, and the infrastructure used in the deployment. §2.3 describes various aspects of the experience gained, in human-robot team workflow, and UGV and UAV technology.

Robots have been deployed in real-life disasters before. Robin Murphy and her colleagues have been in the field with a wide variety of robots, aiding first responders across the world (cf. e.g. [186][187][185]), and Satoshi Tadokoro and his team recently conducted (and concluded) a long-term deployment at the Fukushima nuclear power station [270][166], to cite the main contributions which, however took place in US and Japan. Within Europe, the only officially requested involvement of robots during a disaster we are aware of was at the Cologne city archive collapse in 2010 [156] – without running missions though, the robots were held on standby. The deployment described in this chapter can thus (presumably) be seen as the first deployment of a large human-robot team in Europe, fielding multiple types of robots.

### 2.1 Sites

During the deployment, the team surveyed two sites in Mirandola red area: San Francesco church and the Cathedral (Duomo).

San Francesco church dates back to the thirteenth century, and is one of the first Franciscan churches in Italy. It houses the suspended arks (sarcophagi) of the Pico family, who ruled the Duchy for four centuries (1310-1711). Severely damaged during the earthquakes, only the façade and some of the walls are still standing. The lateral nave, where are the Pico’s arks, is of particular cultural importance, and the Italian Cultural Heritage wanted to recover them, though the Gothic vaults were very dangerous. Indeed, the central nave and the eastern aisle are mostly destroyed, as the bell tower collapsed over the church roof. Fig. 2.2 (l.) shows the nave and a bit of the eastern aisle (in the back): A rubble heap by and large inaccessible to our UGV. The western aisle is still somewhat intact, but structurally highly unstable, see Fig. 2.2 (r.). The ceilings within most of the vaults are damaged, having holes and large unstable pieces of masonry about to come down. Throughout the aisle, there...
are rubble heaps with larger pieces of masonry fallen from the ceilings. At the end of the western aisle there is a heap of rubble due to a roof cave-in. The western aisle was accessible through the lateral coffered door. Size-wise, the western aisle was about 35 meters deep, and 6 meters wide.

Figure 2.2. San Francesco nave (l.) and western aisle (r.).

Our mission targets for San Francesco church were to assess structural damage to pillars and the vaults, to identify passages to the altar so as to recover paintings and to record the state of the Pico’s arks and the vaults over them. We ran missions at the church on Tuesday July 24 and Wednesday July 25, for a total of 5 UAV flights (27 minutes) and 2 UGV runs (1:05h).

The Duomo is a large cathedral finished in the 1470. Also here, the earthquake caused substantial damage. The façade of the cathedral, including a large clock, has largely fallen down – blocking access to the cathedral through its main entrances. The roof over the nave and the northern aisle has caved in, causing massive damage, as illustrated in Fig. 2.3 (l.). The bell tower of the Duomo is still standing, though structurally severely damaged. After consulting with the local CNVVF commander, and the cathedral’s padre vicarious, we managed to get access to the Duomo through the vicarage. This provided direct entry to the southern aisle, still intact though with damage to pillars and to the cross vaults, and another entry in face of the chorus. The southern aisle, shown in Fig. 2.3 (r.), proved to be relatively easy to traverse for the UGV, much more difficult was to reach the S.S. Sacramento chapel at the end of the northern aisle, were the painting of Sante Peranda was held. The top of the nave was covered by a large rubble heap of masonry from the roof and supporting structures.

Figure 2.3. The Duomo in Mirandola, nave (l.) and entrance to the S.S. Sacramento chapel behind the altar (r.).

For the Duomo, we again helped assessing structural damage, outside (bell tower) and inside to access the S.S. Sacramento chapel, to report on the state of the highly valuable paintings. Further missions were assessed to establish the state of the two wooden ancons
covered in gold, although the one on the northern aisle could not be clearly assessed as the UGV had to stop at the exit of the S.S. Sacramento chapel towards the northern aisle, because of the very high heap of rubble. The chapel had been reached from the door accessing the back of the altar, though almost the whole vault was collapsed. We ran several missions at the cathedral on Thursday July 26, for a total of 4 UAV flights (15 minutes) and 3 UGV missions (about 1:20h).

2.2 Deployment

The entire setup deployed in Mirandola has been developed within NIFTi, modulo the basic robot middleware (ROS). We deployed a subset of the available NIFTi functionalities, described in more detail in [71]. We focused on robust functionalities for robot control, video streaming from different omni-directional and monocular cameras, and laser-based 3D reconstruction of the environment, coupled to the NIFTi multi-modal Operational Control Unit (OCU).

2.2.1 System & network infrastructure

As middleware we run the Robot Operating System (ROS) [216]. ROS is used for running processes on the robot, streaming data over WiFi to an operator control unit (OCU) and other visualization tools (RViz), and for logging purposes (rosbag’s). Off-board computers were used for processing 3D laser range data (point clouds), and for the OCU and visualization. We used a mixture of laptops, monitors, and a desktop computer.

We use a 2.4GHz WiFi network. We set up an antenna nearby the entrance to the actual deployment area, fixed to a tripod, and connected by ethernet cable to a router and DHCP server in the command post. The antenna is 50cm long, has 14dBi gain, and is extended with a Ubiquiti high power bullet enabling a transmission power of maximally 28dBm. Each robot (UGV and UAV alike) is also equipped with a bullet, and an omnidirectional rod antenna with a 9dBi gain. As we were mostly working in large open spaces, we did not experience substantial problems with network coverage.

Throughout the deployment, electricity was provided by a Honda 20i portable power generator, provided by the CNVVF. The power generator was capable of generating 230V (±1%) – provided it had enough fuel. Occasionally, fuel would run out causing a shut down of the desktop computer and the monitors, though fortunately never during a mission.

All of these systems worked reliably, in outside working temperatures between 35 to 40 degrees centigrade, and dusty conditions. The gazebo-style roofing over the command post protected the staff, monitors, computers, and other equipment from direct sunlight.

2.2.2 UGV

We deployed two NIFTi UGV platforms in Mirandola: One as main system, and one as back-up should something go wrong. Fig. 2.4 shows the UGV platform used.

The UGV platform has been developed within NIFTi, in close collaboration between research partners and end users. The platform provides a mixture of passive and active morphological adaptivity, to provide good mobility even in harsh terrain. Its bogeys are

1 In retrospect, a sufficiently powerful UPS back-up would have been useful.
connected by a differential, allowing for passive adaptivity (following terrain contour) and active configuration (by blocking the differential). The four flippers can be independently controlled. The bogeys and flippers are constructed such that the platform has a ground clearance of over 15cm (which proved to be very useful in crossing “complex” rubble). The platform can traverse a wide variety of terrain, and per requirement climb up to 45 degrees; practice has shown that we can climb up 60 to 70 degrees inclines. Size-wise, the platform weighs in at about 25kg, and is airline compatible ($L + W + H < 158cm$). The entire body is IP53, it can drive through puddles (IP57), and is conceived for operating temperatures between -10 and 40 degrees centigrade.

The platform comes with a basic sensor suite consisting of a rotating laser (SICK LMS100) mounted in front of the robot, a LadyBug3 omnidirectional camera mounted on top of the robot, as well as an IMU and a GPS sensor. In addition, we mounted a 25cm-tall static mast on the battery compartment of the robot. On top of the mast was a pan-tilt unit with a Kinect camera. This provides a chase-style view of the robot, which is highly useful when navigating (tele-operating) the robot in tight or complex spaces – cf. also the recent experience with Quince reported in [270].

The platform has a quad-core on-board computer (mini-ITX type motherboard), and is powered by a battery pack with an operating time of between 2 and 4 hours. During the deployment we had the robot running all day long, only requiring a battery recharge in the evening.
2.2.3 UAV

Two different types of UAVs were prepared for the mission (see Fig. 2.5). The first platform has been developed within NIFTi according to requirements from end-user and research partners. It has a housing to protect from rain and dust, an interchangeable lower sensor compartment, and removable and easily exchangeable arms with motors. Disassembled it can fit into standard airline baggage. The UAV’s standard configuration includes a Hokuyo UTM-30LX range-finder mounted on top, a sonar and a pressure sensor-based altimeter, an IMU module, a 3D magnetic compass, GPS module, and two cameras - one forward-looking (15 degrees tilted) and downward-looking. The on-board computer is a 1.6GHz Intel Atom-based PC. Operation time is between 10 and 15 minutes depending on operational mode.

The second vehicle is a research platform, based on a construction kit. It has eight high-power engines and can lift up to 2 kg additional payload. It has neither rain- nor dust protection, but it provides a high level of configurability. It can carry a high resolution camera on an IMU-stabilized tilt unit that can be controlled by the pilot, and it can be deployed with all the sensors described for the first platform. The camera is equipped with a high-power video signal transmitter (5.8 GHz, 1.5W). Instead of the Hokuyo laser scanner, a Kinect-like camera (ASUS Xtion Pro) can be mounted on top of the platform; see Fig. 2.6 for example output data. The vehicle is equipped with an Intel i7 2.6 GHz based single-board PC. All components can be easily removed or additional are added.

Because of the flexible configuration of the second platform, and the functionality it could thus make available, it was used as the primary UAV, with the first drone as backup. The UAV had to be navigated in a GPS-denied environment, turned on spots to acquire a better view, and the pilot could only remain on one spot next to the entrance into the area being surveyed. Crucial in this case was the UAV’s ability to memorize its original orientation in space, and to translate the pilot’s movement control commands relative to his position. This functionality was only tested on this UAV prior the mission.
2.2.4 Human-robot teaming

Before the actual deployment we set up an organizational- and communications structure for the team to be deployed. The organizational structure is based on previous experience \[185\] and aims to set up chains of command which reflect responsibility (and ultimately, liability). The overall responsibility for each mission lays with the CNVVF, determining mission targets is resolved between NIFTi, the CNVVF and Cultural Heritage responsible, watching over missions targets.

![Organizational structure for geographically distributed human-robot team, including a UGV and a UAV, and reflecting responsibility/charges in the chain of command.](image)

Fig. 2.7 shows the structure, tied to roles. In practice, a single person can play multiple roles. The Mission Cmd (CNVVF) is in charge of the entire mission.

The UAV team largely deploys in the field. The UAV team consists of the UAV Operator,
piloting the UAV in-field; a UAV Mission Spc, watching the UAV video streams and guiding the UAV Operator to mission targets; and a Safety Cmd (CNVVF) safeguarding the UAV team. During the deployment, the UAV Mission Spc mostly operated with the UAV Operator in-field, to provide the UAV Operator with an extra pair of eyes on the UAV. See Fig. 2.8. The UAV team assessed video material afterwards. The information gained from video material was provided directly to the CNVVF, and was also used in the briefings for follow-up UGV missions.

Figure 2.8. UAV team with an Operator (l.), Mission Spc (m.) and Safety Cmd (r.) operating near the Duomo, assessing damage to the Duomo bell tower.

Similarly, the UGV team consists of a UGV Operator, and a UGV Mission Spc. The UGV team is located “remotely” in a command post. Both teams are backed up by a System Spc and an Infrastructure Spc operating from the command post, who ensure the network- and system infrastructure remains alive. The intention was for the UGV and UAV Mission Spcs to collaborate with a domain expert, to establish mission targets before and possibly during the mission. During the deployment, however, mission targets were always determined beforehand, together with members of the CNVVF and others involved in recovery (vicarious and cultural heritage representative). The UGV team therefore “reduced” to a UGV Operator, Mission Spc, and a person doubling as Infrastructure/System Spc.

The UGV team uses the NIFTi OCU and RViz, to display video streams, and visualize incoming sensor information. The UGV is tele-operated using a gamepad (or alternatively, the OCU), and the PTU on the mast is controlled using a simple widget. The UAV team uses an R/C control to pilot the UAV, and can use an instance of the NIFTi OCU to watch streaming video from the UAV. Both teams, and the Mission Cmd, have the possibility to use Augmented Reality goggles to watch video streams. In the end, the UAV Mission Spc used this option to great effect. Given that the UGV team relies heavily on face-to-face communication, the goggles were impractical.

2.3 Experience

Below we describe first-hand experience with deploying complex human-robot teams in an earthquake-hit disaster area.
2.3 Experience

2.3.1 Human-robot teaming

Deploying robots in an Urban Search & Rescue mission is a team effort. This holds just as much for operating a UGV, as it does for flying a UAV. There is too much information to attend to, ranging from sensor information to system- and infrastructure-related monitoring, to be handled safely by a single person, cf. also [185]. The organizational structure shown in Fig. 2.7 reflects this, and as we already indicated in §2.2 this structure was pretty much implemented “as-is” in the field.

Nevertheless, both the UGV Operator and the UAV Operator suffered from cognitive overload. UGV missions typically lasted about half an hour, and were characterized by interleaving driving, and observing. (At the moment we cannot yet say whether the ratio reflects earlier experience as reported e.g. in [28, 152].) This interleaving made it possible for the UGV Operator to relax, momentarily – a luxury the UAV Operator did not have. The UAV did have some degree of autonomous flight control, but circumstances demanded that the UAV Operator continuously attended to the UAV.

This provides a first insight in, or rather perspective on, possible roles of “robot autonomy.” In human-robot teams, humans and robots are (inherently) interdependent [129]. Robots can go where humans need to but cannot, whereas humans can aid robots in better understanding and operating in the environment. Both humans and robots are problem-holders – with the obvious “but” though that the human users are the stake-holders. Robot autonomy is ultimately to be in service of the human to collaborate with the robot as if “operating the world rather than the robot” [193]. We saw this over and again during the deployment, see also §2.3.3 and §2.3.2. Autonomy is to make life easier for the human to understand the environment, (and not for the robot to bugger off on its own in “look ma no hands” mode).

The UAV serves as a good example here. The UAV Mission Spc used augmented reality eyewear (Vuzix WRAP 920AR+) to watch the video stream from the camera mounted in a tilt-unit under the UAV. This quickly led to a pseudo-immersive experience, and the desire to look left-and-right and have the UAV and/or the tilt-unit follow suit. More (and better) flight control autonomy, enabling the UAV to simply hover and turn on the spot, would have
facilitated this.

Further insights concern the flow of information between the UAV team and the UGV team, in terms of tactical (team-level) situation awareness (tacSA) and mission planning. During the entire deployment, the UAV team and the UGV team never operated in the same area simultaneously. Partly, the reasons were technical (network) and environmental (dust). Another reason regarded the use, the workflow which emerged in using information from the different teams in establishing further missions. Based on in-field LOS observations of the area to be deployed in, and a first set of recon missions by the UAV team, we would establish a first sketch of the environment. Most importantly, we would identify important landmarks to navigate by, establishing explicit names for them (e.g. “column 4”), and determining targets for future missions. These targets typically included areas and objects to be observed, and how these observations were to be made. Targets were discussed together with members of the CNVVF.

Follow-up missions then helped detail out tacSA and revise mission targets. Since tacSA was built up from operational SA coming from the different teams, we occasionally found mismatches in expectations which then required further missions; (as was to be expected, cf. [227]). For example, video from initial UAV recon missions in San Francesco church gave the impression that the top of nave would be reachable from the western aisle, either from between the fourth and fifth columns, or the opening behind that. This would then make it possible for the UGV to drive close to the altar, and provide close-up video. As it turned out at the end of the second UGV mission, what seemed accessible terrain from the viewpoint of the UAV, was not so in UGV-reality. The UGV did manage to take video of the altar, but an additional mission was then planned for the UAV to fly in over the main nave and record video from that viewpoint.

The UAV and the UGV thus supported each other, but indirectly so. It did result in the required tacSA for the team, and the other stake-holders. At the same time, it also opened new questions as for how to optimally transfer data from one mission to the next, to make the tacSA consolidated so far available online. Before the deployment, we had developed a basic
viewer for post-mission analysis. During a mission, a Mission Spc could take snapshots in an OCU, annotate them with a description. Snapshots were stored with the text annotation and robot position information. For post-mission analysis, the viewer could then load snapshots and a 2D map, mark the snapshots on the map, and enable the user to browse snapshots. We did use some of this functionality, particularly to get high-definition snapshots of cultural artifacts, but what was missing was the possibility to correlate geo-referenced video from one mission, and show this during another mission in a context-aware fashion, i.e. show previously recorded video of the environment in which the robot in the current mission is located. This is a form of information fusion to provide continuous tacSA across different missions within a single area. We made similar observations about map information. The UAV could be deployed to gather a 3D reconstruction of the environment. This map would not need to be so detailed as to enable the UGV to localize itself in it. All the map would need to make possible is a form of forward mapping/scouting for the UGV team to determine the optimal path amongst different alternatives. While operating in a harsh environment like the ones in Mirandola we would have greatly benefited from such functionality, as it could have saved time, or have indicated paths where none were obvious (like a traversal from the western aisle to the nave in San Francesco church). See [2.10] and [2.11]. Coupling the UAV 3D information to the dense 3D metrical representation for the UGV could improve situation awareness for the Operator as well as the robot.

In summary, we observed several issues regarding the operations of a geographically distributed human-robot team, with team members operating both in-field and at a remote command post. As the UGV and UAV teams operated asynchronously, maintaining and transferring tactical situation awareness between missions was an issue to the extent that system automatization could help (in the future) to make aspects of operational situation awareness from one team available to the next in an operational context-aware fashion.

2.3.2 UGV

UGV piloting does not require keeping the LOS on the robot, due to the sophisticated visualization facilities offered by the OCU and RViz, providing the remote operator with situation awareness, as mentioned in §2.2. The UGV exploration is carried out by alternating two phases. The first phase corresponds to navigation and it is functional to terrain under-
standing: the pilot retrieves information on how to change the robot morphology in order to choose the best route and overcome or avoid obstacles. The second phase is *observation*. This is functional to the collection of data: the Kinect sensor is properly oriented in order to perform data acquisition towards the locations of interest. The UGV Mission controlled the alternation of these two phases and communicated to the UGV Operator the items of interest to inspect and whether to stop the navigation in order to acquire data in details.

Four different flipper configurations have been pre-defined within the OCU to support the navigation phase (e.g. for flat terrain or to approach an obstacle) that are manually adjusted by the UGV Operator according to the information provided by the camera feedbacks, the rotating laser and the robot 3D model within the computed 3D map (Figure 2.13). The latter was eventually deemed as the main source of information in order to suitably adjust the configuration of the UGV rear flippers. The 3D robot model provides an immediate visual feedback of the current flippers configuration with respect to the terrain, which is displayed on the 3D map. Adequately setting the UGV rear flippers is mandatory to prevent the UGV from tipping-over whenever overcoming rubble or climbing stairs was the case. Continuously switching among the camera feedback and the 3D visualization of the UGV in the map increased the UGV Operator’s cognitive load, which increased the time required to accomplish the mission, indeed.

During the *observation* phase, the UGV Operator was guided by the mission specialist to acquire data on the items of interest, by changing the orientation of the pan-tilt camera. This had a negative effect on the awareness of the UGV operator regarding the relation between the current coordinate frame of the camera orientation and the robot pose. To alleviate this problem, a set of pan-tilt configurations were pre-set in order to allow the UGV Operator to quickly recover the views related to the pre-defined reference poses.

Within the UGV sensor suite, the pan-tilt camera could not be used to inspect the back of the UGV. When this was required, the stream coming from the LadyBug3 together with the RViz 3D visualization constituted the only source of information about the robot status. Another issue with the pan-tilt-mounted Kinect concerned the dynamic adaptation in response to changes in the lighting conditions. In the interior of San Francesco church the lighting conditions were such that even within a shadow, the Kinect camera provided a qualitatively good output. In the case of the Duomo church, the situation was quite different as the Operator has to drive the UGV inside the church through a corridor with very constrained lighting. In that situation, a staircase had to be traversed and the most valuable information came from the LadyBug3 camera and the 3D scanner.

The above mentioned situation at the Duomo highlighted another important issue, namely
the fact that the presence of the static mast increased the risk of tipping over. This was particularly harmful when the UGV was operated to overcome the staircases and the large rubble heap occupying the way to the western aisle.

Nonetheless, according to the UGV Operator, the pan-tilt camera proved to be the highest priority perception instrument and most immediate choice among the sensor feedbacks and visualizations. Such a configuration-adaptive camera turned to be the most valuable sensor in the navigation phase because of the easiness in the interpretation of its output but it was also used in the observation phase to collect data regarding the surrounding of the UGV.

2.3.3 UAV

Piloting a UAV in a confined space is a challenging task by itself, even if the pilot has LOS to the UAV. In addition, it was clear that if the UAV looses control or crashes into an obstacle, we will not be able to retrieve it. All these factors negatively impacted the UAV Operator’s stress levels: No mistake was allowed.

The first deployment was in San Francesco church. It was required to provide images of structural damage to arcs and ceilings inside the western gallery of the church as well as to capture the condition of coffins and art, to observe the altar, and the end of the eastern gallery. The only access to the church was provided via a door of the western gallery with a less than 1 by 1 meter surface free of rubble for take-off and landing or via the caved in roof.

Before the actual deployment all UAV functionalities were tested, including GPS-aided hovering next to the church, flying along two galleries of a seminary that was intact, testing turning maneuvers, and testing the range of the video transmitter. After being satisfied with the performance, missions were undertaken.

Each deployment had a clearly identified task, which with a high level of stress and cognitive load helped the pilot to stay concentrated. One example of such a task is to reach the end of the western gallery and look to the right to see if a UGV could traverse there. During the first two missions the Mission Spc was sitting in a distant location from the Operator and viewing the video stream from the on-board camera in goggles (see above). Since the Mission Spc can only observe the area from the video and cannot intervene, and two successful missions were carried out without accident, it was decided that the Mission Spc would better assist the Operator, standing next to him. This was especially needed because of the amount of dust produced by propellers. Without high-contrast dark lenses it is hard to see the UAV. The video from the camera provides a better view but it is impossible
to control a flying vehicle in such a confined space with the camera view only. Even having a direct LOS, it is hard to understand how close you are to obstacles or walls, especially if the drone is far away. Our eyes are losing clear depth perception with long distances.

It is important to mention that even though the above mentioned functionality of translating control commands independent on the UAV orientation was of a great help, it did not function properly. The problem was in that the orientation calculation relied strongly on the on-board magnetic compass. In an open space this worked perfectly, but in the church there were numerous metal bars that influenced the compass thus causing sudden turns of the drone to several degrees. This dramatically increased the cognitive load of the Operator, forcing him to concentrate more on maintaining the UAV’s position. Even if each mission lasted in average no longer than 5 minutes, the stress level of the Operator at the end was so high that it could easily provoke making control mistakes. This was clearly seen by the landing maneuvers which later on were no longer performed smoothly, and caused hitting the entrance door several times, breaking several propellers (3 in total).

Another important mission to mention from the point of view of the obtained experience is the structural inspection of the bell tower of the Duomo. The total flight lasted slightly less than 6 minutes while the first 4 minutes appeared to be almost unusable. The reason was simple - the pilot could not see what he was filming. The task was to concentrate on a defined part of the tower in which CNVVVF were particularly interested. The pilot had a plan to reach a desired altitude and distance to the tower, fix the UAV by means of GPS, and then manipulate the camera and the copter orientation. After reaching the required altitude the pilot could not achieve a stable hovering of the UAV due to appeared wind. The pilot takes a decision to operate fully manually and approach the point of interest. The task seemed to be accomplished but shortly before landing the pilot noticed that the camera was facing upwards. That means a wrong point of interest was filmed and had to make a second approach.

Summarizing the experience of piloting the UAV for such a mission following conclusions can be made:

• Only functionality that has been tested, and therefore can be trusted, should be used.

• A high level of stress in combination with cognitive load can provoke pilot mistakes.

• A minimum level of autonomy is required even for pure tele-operation, as it provides the possibility for the UAV to autonomously maintain its position when no movement commands are received.

• A better situation awareness for the pilot is of high importance, particularly to provide information about close obstacles, and better depth perception.

• Observation camera control should be independent from the pilot, unless the vehicle is fully autonomous.
Chapter 3

A higher level representation of the robot low-level perception for reasoning and action planning

In search and rescue domains, where environmental constraints are minimal, highly efficient multi-modal perception is a prerequisite for action planning and execution [261, 189]. Nowadays, robot are endowed with a standard set of perception capabilities, even more efficient, such as 2D/3D simultaneous localization and mapping (SLAM) and object detection and localization [68, 181, 103, 136]. Furthermore, human-guided operation of the robot can be performed in a natural way using dialogue, wherein the robot receives instructions by the human operator using speech recognition that translates natural language into predefined tasks. The main problem to be addressed translates into building meaningful, higher-level representations of the real-world from incoming raw data that are acquired from cameras and laser scanners and formulating these representations into a domain where reasoning and goal generation takes place [184, 167, 37]. In this chapter, we present a robotic system that integrates into the Robot Operating System (ROS) [217], multi-modal perception from vision and mapping with action planning. With respect to vision, object detection is performed using several learnable detectors that are rapidly updated while 3D localization is estimated via several object detections that are used by a greedy algorithm based on RANSAC [123]. This information is used to augment a graph-based representation of the metrical map that is dynamically constructed during exploration using an unsupervised topological segmentation method [157]. The result of this process is a set of nodes that are annotated with properties related to the detected objects wherein connections between the nodes determine the traversability between the corresponding areas. By formulating the perceived knowledge about the environment into a suitable logic representation that is maintained into the knowledge base [208, 202], the logic-based planner can build a set of tasks, whose goal is communicated by the operator, through dialogue. The planner both verifies the consistency of their executability and monitors its execution reporting possible failures.

The chapter is organized as follows. Paragraph 3.1 describes the topological segmentation of metric maps. In Paragraph 3.2 we describe the approach proposed to both detect and localize movable objects in the 3D scene. Paragraph 3.3 introduces the human-robot interface for both control and communication, through spoken dialogue. In Paragraph 3.4
3. A higher level representation of the robot low-level perception for reasoning and action planning

Figure 3.1. (a) Topological decomposition of a metrical map using the proposed methodology; (b) Graph-based representation of the topological map.

we describe the logic-based control system of the robot, responsible of both compiling the perception into the domain knowledge of the robot, inferring parametric plans, instantiated by perception, monitoring and executing the inferred plans.

3.1 Topological Segmentation of Metric Map

The perception of the topology of the environment through mapping is initially represented within a metric layer and in the following as a higher-level topological layer, used for action planning.

Using the sensors of the robot, we first build a metric map using standard SLAM algorithm\(^1\). From wheel odometry and a 180° 2D laser scanner, we obtain an occupancy grid using Rao-Blackwellized particle filtering [107].

Based on this occupancy grid, we perform a topological segmentation of the free space in the metric layer. Instead of working at the discretization level of the occupancy grid, we down-sample it to zones of size 1\(m^2\) that will be referred to as “nodes” in the rest of this paragraph.

We use spectral clustering [27] as the segmentation method of the metrical map. The general algorithm of spectral clustering requires the neighborhood graph together with the corresponding adjacency matrix \(W\) with \(n \times n\) elements \(W(i, j) = \omega_{ij}\), where \(n\) is the number of nodes in the graph. Among the different approaches that have been considered in the definition of the weights \(\omega_{ij}\) between nodes, in our work, we define it as \(\omega_{ij} = e^{-\frac{l(i, j)}{\sigma^2}}\) where \(l(i, j)\) is the distance between the centers of nodes \(i\) and \(j\). Following the notation of our previous work in [157], the algorithm of spectral clustering is shown in Alg. 1.

In order to enable the robot to execute plans that remain consistent during the discovery of new, previously unexplored areas, we need to ensure the consistency of the topological map segmentation in time. In other words, a new segmentation of the map should build upon the last instance of segmented map instead of employing a segmentation of the complete map.\(^1\)

---

\(^1\)We use the GMapping software in ROS: \url{http://www.ros.org/wiki/gmapping}.\n
Algorithm 1: Spectral Clustering on Topological Segmentation

**Input:** $W \rightarrow$ Adjacency matrix, where $W(i, j)$ indicates the weight between two nodes $i$ and $j$; $k \rightarrow$ The number of clusters-areas; $T = \{T_1, T_2, \ldots, T_k\} \simeq \{1, 2, \ldots, k\}$;

**Output:** The list of indices corresponding to the nodes.

1. Calculate the normalized graph Laplacian using $L_{\text{sym}} := I - D^{-1/2}WD^{-1/2}$ or $L_{\text{rw}} := I - D^{-1}W$, where $D = \text{diag}\{d_1, \ldots, d_n\}$ and $d_i = \sum_{j=1}^{n} w_{ij}$;
2. Calculate the $k$ smallest eigenvectors $u_1, \ldots, u_k$ of $L$ (either $L_{\text{sym}}$ or $L_{\text{rw}}$), form the matrix $U = [u_1 \ldots u_k] \in \mathbb{R}^{n \times k}$;
3. Set $\hat{U}$ to be $U$ with rows normalized to the unit $L_2$ norm;
4. Use $k$-means clustering on the rows of $\hat{U}$;
5. Assign label $T_i$ to cluster $j$ if and only if row $j$ of $\hat{U}$ is assigned to cluster $j$.

Algorithm 2: Incremental Segmentation

**Input:** $M_t \rightarrow$ Decomposition results of the occupancy grid map; $\text{Obs}_t \rightarrow$ Threshold determining sufficient new observations;

**Output:** The updated list of topological regions $T$.

Retrieve the newly updated free nodes from $M_t$, with a total surface area $\text{Area}_t$;

if $\text{Area}_t < \text{Obs}_t$ then
   Return;
else
   new_list_of_regions = Do_SpectralClustering($M_t$);
   Clean_up();
   Register($T$, new_list_of_regions);
end if
3. A higher level representation of the robot low-level perception for reasoning and action planning

In Figure 3.1 we show an example of topologically segmented metrical map based on the described methodology.

In order to use the topological decomposition of the metrical map for plan generation, we extract the centroid of each area and use it to denote the position of a node within the graph-based representation of the topological map while the edges between the nodes are determined according to the proximity of one node with respect to another based on a distance threshold (see Fig. 3.1(b)). The topological segmentation of the metrical map is a continuous process running in parallel with object detection and localization that in turn, augments the domain knowledge stored in the topological graph with incoming information from vision.

3.2 Object Detection and Annotation of Topological Graph

Visual perception of the environment by the mobile robot is implemented by object detection and localization using an omnidirectional camera. The information about the detected objects is then introduced into the graph-based representation of the topologically segmented map in the form of graph nodes around the object that are attached onto the graph and by storing information related to the detected object.

Online learnable object detector We use an online learnable object detector \cite{123} that is based on an efficient combination of \cite{153} and \cite{131}. There are two key ideas we are using:
Algorithm 3: 3D localization of objects

Input: $D$ → Set of unprocessed object detections consisting of position and direction in common reference frame; $D_{\text{min}}$ → minimum number of consistent detections for object to be localized; $i_{\text{max}}$ → maximum number of iterations; $O$ → Set of detected objects $O$ with resolved position and set of consistent detections.

Initialize set of objects $O \leftarrow \{\}$

while $|D| \geq D_{\text{min}}$ do

Initialize new object $(p^*, D^*)$ with empty consensus set $D^* \leftarrow \{\}$

for $i \leftarrow 1$ to $i_{\text{max}}$ do

Compute position $p$ from two detections $D_0$ randomly chosen from $D$

Find detections $D_1$ from $D$ consistent with $p$

Initialize refinement counter $j \leftarrow 0$

while $|D_{j+1}| > |D_j|$ do

Set $j \leftarrow j + 1$

Refine position $p$ using all detections $D_j$

Find detections $D_{j+1}$ from $D$ consistent with $p$

end while

if $|D_j| > |D^*|$ then

Update new object $p^* \leftarrow p$, $D^* \leftarrow D_j$

end if

end for

if $|D^*| \geq D_{\text{min}}$ then

Accept new object $O \leftarrow O \cup \{(p^*, D^*)\}$

Remove processed detections $D \leftarrow D \setminus D^*$

else

return $O$

end if

end while
3. A higher level representation of the robot low-level perception for reasoning and action planning

Kalal et al. [131] showed that a tracker allows for efficient boosting of a detector and Lepetit et al. [153] showed that multiple detectors using the same set of features can run almost at the same rate as a single detector. In contrast to [131], we use several rapidly updated detectors instead of a tracker that use the same features but are updated with different parameters, yielding similar boosting ability as a tracker, while running in real-time.

**Category car detector** The above described online learnable detector essentially detects *instances* of a certain object. What is crucial is the visual similarity of the instance to the object that has been selected for learning. The individuality of the learnable detector makes general car detection hard. Car surface is usually highly specular while changing illumination influences its appearance significantly. We applied Adaboost detector from OpenCV library [22] for detection of rear car part. For learning we partly used available datasets and partly assembled our own.

**3D localization of detected objects** Object detectors localize objects of interest in captured images. This information needs to be transformed into the corresponding position in the 3D world so that the robot can infer strategies in approaching or avoiding objects depending on the task.

For object localization in 3D, a new ROS component has been developed. This component depends on the robot pose estimated from odometry and 2D laser scan data, on static transforms which relate the omnidirectional (Ladybug3) camera to the robot base and on internal camera calibration. These transforms are used to resolve directions in which the objects were detected within a particular reference frame. From several detections of a particular object class, a number of objects and their poses are estimated using a greedy algorithm based on RANSAC. In this way, the system tries to interpret all these detections following the minimum description length principle. An example of car detection and localization within the map is shown in Figure 3.2 while the localization algorithm is given in Alg. 3.

After the 3D localization of an object we associate a set of nodes around the object that correspond to the areas that the robot can reach (left, right, back, front) in order to approach the object. In this way, the topological graph representation of the metric map is enriched with perceptual information coming from vision giving an augmented graph-based representation that the planner is using to generate complex plans.

3.3 Dialogue

In this chapter, we consider settings in which a human operator is at a remote location, away from the disaster area into which the robot is deployed. The human and the robot interact through an Operator Control Unit (OCU). The OCU is illustrated in Figure 3.3.

The OCU provides the operator with a visualized map, camera stream, as well as plan and dialogue histories. The OCU facilitates multi-modal interaction: The operator and the robot can use spoken dialogue to interact, and the operator can use pointing gestures (mouse) to select waypoints or landmarks for the robot to go to. A gesture can but need not be accompanied by an utterance. Selecting a waypoint, or saying "Go here [select waypoint]" have in principle the same interpretation, namely that the operator intends the robot to go to the selected location.
3.4 The Logic-based Robot Control

The operator and the robot are working together on a task – namely to jointly explore an environment. This makes the interaction inherently task-driven. The idea of interpreting communicative acts in terms of their underlying intention (operator goal) therefore plays a fundamental role in the OCU design. It enables us to connect interaction to action in the world, in a way as proposed by the GUI design guidelines of [102], and models of situated dialogue processing like in [143]. The interpreted intention is grounded in the “world” by resolving the references in the utterances to aspects in the robot’s situation awareness that is maintained and updated into the knowledge base of the planner. These references can be referring expressions like “the car” as in “go to the car,” but can also be deictic references such as “here” or selected waypoints or detected objects. The resulting representation provides a smooth bridge to planning by stating what (abstract) goal the robot is to achieve, and relative to what locations.

3.4 The Logic-based Robot Control

The robot control is endowed with a declarative temporal model of the controllable activities and a planning engine. The structure of the architecture is shown in Figure 3.4.

The declarative temporal model is specified in the Temporal Flexible Situation Calculus (TFSC) [80] and explicitly represents the main components and processes of the robot system, the cause-effect relationships as well as the temporal constraints among the processes. The TFSC extends the language of a basic theory of actions in the Situation Calculus [206, 224], combining temporal constraint reasoning and reasoning about actions. It intermediates between Situation Calculus formulae and temporal constraints networks [57].

Within this framework, the system is modeled as a set of components specifying their activities over timelines. In the implementation that is presented in this chapter, we make
3. A higher level representation of the robot low-level perception for reasoning and action planning

Figure 3.4. The robot control architecture.

use of three components specified as \textit{slam}, \textit{navigation} and \textit{vision} that trigger a set of processes, according to their role. In detail, the \textit{slam} component performs a \textit{toposeg} process to segment the metric map of the environment, the \textit{navigation} component executes a \textit{goto\_node(nodeld)} process to reach a target node \textit{nodeld} and the \textit{vision} component performs a \textit{detect(object)} process that detects objects in the acquired video stream and localizes them within the map.

These processes are explicitly represented through fluents and instantaneous starting and ending actions which are defined in terms of preconditions and effects. For example, the \textit{toposeg} process is modeled by the fluent \textit{process\_\textit{\textit{slam}},\textit{toposeg},\textit{s}} and both the actions \textit{start\_toposeg(t)} and \textit{end\_toposeg(t)}. The effects are defined by the following successor state axiom:

\[
\text{process(slam, topos[gamma], do(a,s))} \equiv \exists t \ a = \text{start\_toposeg}(t) \lor \\text{process(slam, topos[gamma], s)} \land \\
\neg \exists t' (a = \text{end\_toposeg}(t'))
\]

where the action preconditions are:

\[
\text{Poss(start\_toposeg(t),s)} \equiv \text{idle(slam,s)} \land \text{time(s)} \leq t \\
\text{Poss(end\_toposeg(t),s)} \equiv \text{process(slam, topos[gamma], s)} \land \text{time(s)} \leq t
\]

and the hard time constraints among activities are managed by the TFSC model using Allen-like temporal relations \cite{2}.

The planning engine is composed of two main logical modules: the plan generator and the execution monitoring. The plan generator relies on a library of Prolog scripts designating the set of tasks which the mobile robot can perform, according to the specified processes, their temporal constraints (compatibilities), and preconditions. For example:
3.4 The Logic-based Robot Control

- **Go here**: navigate to a position within the metric map.
- **Move left, right, forward**: execute simple motion commands.
- **Visit graph**: visit all the nodes of the graph-based representation of the metrical map.
- **Visit node**: visit a specific node of the graph-based representation of the metrical map, following the optimal path within the graph according to a selected criterion (e.g. the distance between the nodes).
- **Approach object**: visit a node (left, right, front or back) around a detected object.

These tasks are based on the perceived information of the environment using *vision* and *slam*, which is represented in the knowledge of the planning engine in the form of a graph (see Fig. 3.5). The nodes of the graph correspond to topologically segmented regions or approachable areas around a detected object and edges determine the traversability between the regions.

The execution-monitoring is a continuous process which ensures that the set of action sequences, generated by the plan generator according to the TFSC model and the current state of the domain knowledge, are consistently executed. Concurrently, at regular time intervals, the execution-monitoring reads the system state and monitors the execution of the activities, in order to detect system malfunctioning that may result in action failures [56, 201].

Both the TFSC model and the planning engine are implemented in ECLIPSE Prolog [4] which optimally combines the power of a constraint solver (for the time and compatibility constraints) with inference in order to generate the set of action sequences, and also enable the continuous update due to incoming new knowledge (using finite progression).

**The planner embedded into the ROS architecture** The declarative temporal model and the planning engine are fully embedded into ROS. The underlying implementation involves the following ROS nodes:

![Figure 3.5](image.png)

*Figure 3.5*. An instance of topological map segmentation together with the corresponding node centers and connections within the nodes that is used by the robot to generate plans.
3. A higher level representation of the robot low-level perception for reasoning and action planning

1. A ROS Service node (RSN);

2. A ROS Robot node (RRN).

The RSN allows for RPC request-reply interactions with the user interfaces while the RRN takes care of the communication between the perceptual and physical robot components and the planning engine.

The RSN manages the communication with the human interfaces and embeds into the ROS language the logical part of the control system. It enables the human operator to interact with the control system during the computational cycle. In fact, the operator can interact with the system by posting goals, and he/she can also interrupt the execution of a plan, generated by the planning engine, or directly control the perceptual and physical robot components.

On the other hand the RRN receives the information from the different perceptual modalities (mapping and vision) in order to build the domain knowledge of the robot. The RRN is also responsible for sending task activation signals to the robot components in order to perform the sequence of actions generated by the planner, according to the operator's requests.

More specifically, the RRN takes as input the segmented metric map and creates a logical graph-based representation of the explored area. Nodes and edges of the graph are specified by predicates together with additional information about the edges (capacity, distance) and the nodes (surface area, convexity). According to this representation the plan generator is able to infer shortest paths along the graph, inducing from this a plan, whenever a node-to-node navigation task is requested by the operator.

Similarly, the visual perception provides the RRN with information about detected objects. Whenever an object is detected and localized within the metric map, a new set of nodes is added to the topological graph, circumscribing the object, and the corresponding properties are specified in the action theory. The orientation of the object can be used to determine the correspondence of these nodes with respect to the areas at the front, back, left and right of the object. This allows the plan generator to make a plan, to suitably approach the detected object, and also it enables the model-based planner to reason about the augmented logical structure.

The RSN receives a task request from the user, creates a logical representation of the request and sends the corresponding Prolog term to the execution-monitoring. The execution-monitoring asks the planner to generate an executable set of action sequences, according to the current state of the knowledge base. Once the plan is generated, the execution-monitoring sends these actions for execution to the RRN. Upon receipt of an action, the RRN takes the control managing the physical execution of the action. When an action is performed, the execution-monitoring retrieves the state of the robot, whence it verifies whether the action was successful or not. In the former case, the execution-monitoring sends the next action to the RRN, otherwise it aborts the execution of the remaining plan and notifies the failure to the RSN. In turn, the RSN returns the result of the execution of the action to the interface in so yielding the control to the operator.
Chapter 4

Terrain Traversability in Rescue Environments

Understanding terrain structure by taking into account specific traversability affordance is of primary importance in rescue scenarios, and for the Urban Search & Rescue Robotic (USRR) community. There is no way to go toward autonomy or semi-autonomy as far as a terrain structure interpretation is not adequately faced and approximatively solved, so as to ensure the robot to move on and move under it.

In this work, we approach this problem with only laser point cloud (or scan) data and provide some preliminary new methods to establish where the terrain provides a stable ground for the robot. Most of the earlier works in literature have approached this problem by assuming that the UGV (unmanned ground vehicle) can only move on a plane. However, in search & rescue scenarios, the UGV has to move in very complex environments in which terrain has a primary role. There are three main problems though, with terrain structure estimation. On board computational facilities, real-time terrain structure estimation, risk and cost evaluation in case of lack of information. A robot that can afford a harsh terrain, might not have on board the computational power to process all the data, therefore point cloud processing, to appraise some primary conditions, like stability, has to be real-time. Finally, because only an approximate estimation of the stability conditions can be provided, a terrain map should also provide risk levels, according to the robot different kinematics constraints.

In this chapter we introduce a preliminary account of terrain estimation, that is implemented with a real-time algorithm. In our approach we first provide a structure for the point cloud that does not need to estimate a surface, and that can maintain dynamically both filled and empty spaces, namely positive and negative obstacles. The structure also maintains the different levels of a point cloud, according to the robot structure, namely height and width, considering that a rescue environment has several levels due to collapsed structures. The method introduces some terrain classification based on slope morphology, density and parameters for them accordingly. Using the particular geometry of point structure introduced we arrive to the definition of a traversability map and cost map. This last is defined by relaxing some constraints on terrain stability.

The chapter is organized as follows. In Paragraph 4.1 we present the method for building the point cloud structure needed for terrain interpretation. Here, we also introduce the traversability map and the cost map. In Paragraph 4.2 we discuss the results obtained with the proposed algorithm within two testing scenarios. The first testing scenario considers a
building with staircase, and the other scenario is taken from the earthquake training scenario of the Italian Fire Fighters in Prato, Italy.

4.1 Proposed Methodology

4.1.1 3D space structure

In this paragraph we introduce the 3D space structure on which we shall model the terrain traversability map, given the continuously updating point cloud $\mathcal{P}(t)$, for $t=0,1,\ldots,T$.

We define a partition of the 3D space, according to the point cloud updating process, by taking care of the full and empty spaces, with respect to both the robot dimensions and the points distribution. The construction of the space is based on two steps, one relative to the $x$-$y$ plane and one, dynamic, with respect to the $z$-axis.

Let us define at $z=0$ a grid of size $400\times400$ meters and resolution $r > 0.8m$, and let $\gamma_{\text{min}}$ and $\gamma_{\text{max}}$ be the grid bounds. We begin by collecting all points $p=(x,y,z)^T$ of the current point cloud, within grid cells bounds and with $z$ varying freely (see Figure 4.3). To this end we introduce the following definitions.

Definition 1 (Column Map) Let

$$\mathcal{P}^+(t) = \{ p=(x,y,z)^T \mid p \in \mathcal{P}(t) \text{ and } p \notin \mathcal{P}(t-1) \}$$

(4.1)
4.1 Proposed Methodology

Figure 4.3. 3D space structure. Given a grid of size 400 × 400 meters a column collects all the points \( p=(x,y,z)^\top \) such that \( (x,y) \) falls into a grid cell \((i,j)\), with \( z \) free to vary. Levels are dynamically assessed as the point cloud is updated, and each level is defined within \( Z_{\min} \) and \( Z_{\max} \).

and assume \( \mathcal{P}(0)=\emptyset \). Then, the column map \( C(i,j,t) \) is inductively defined as follows:

\[
\begin{align*}
C(i,j,t) &= \mathcal{P}(t), \text{ for } t=0 \\
C(i,j,t) &= C(i,j,t-1) \cup \\
& \quad \{ p=(x,y,z)^\top \in \mathcal{P}(t) \mid (i,j) = \left( \text{sgn}(x) \left\lceil \frac{|x|}{r} \right\rceil, \text{sgn}(y) \left\lceil \frac{|y|}{r} \right\rceil \right) \} \quad \text{for } t > 0
\end{align*}
\]

Here \(|x|\) indicates the absolute value of \( x \), \( \text{sgn}(x) \) the sign, and \( \lceil x \rceil \) is the ceiling operator, mapping \( x \) to the smallest following integer. Note that the point cloud is continuously updated and new detected points at time \( t \) are mapped to the appropriate column maps.

The set of points falling into the column map \( C(i,j,t) \) is further partitioned into a set of levels. Each level is an interval of the form \([Z_{\min}, Z_{\max}]\) obtained by partitioning the \( z \) according to height parameter \( RH \) (see Figure 4.2).

Definition 2 (Levels set) Let a level be defined as:

\[
\ell=[Z_{\min},Z_{\max}]
\]

let \( p=(x,y,z)^\top \in \mathcal{P}(t) \), \( \setminus \) be the set difference operator, and let us abbreviate min and max with \( M=\max \), \( m=\min \), just in this definition for sake of space. Then, the set of levels \( \mathcal{H}_{ij}(t) \) for \( C(i,j,t) \) are defined, inductively, as follows:
for \( t = 0 \)
\[ H_{ij}(t) = \emptyset \]
for \( t > 0 \)
\[ H_{ij}(t) = \]

1. \( H_{ij}(t-1), \) if \( \exists \ell \in H_{ij}(t-1) \text{ s.t.} \)
\[ z \in [Z_m, Z_M] \]

2. \( (H_{ij}(t-1) \setminus \{[Z_m, Z_M]\}) \cup \{[z, \gamma]\} \)

3. \( (H_{ij}(t-1) \setminus \{[Z_m, Z_M]\}) \cup \{[Z_m, z]\} \)
\[ \exists \ell \in H_{ij}(t-1) \text{ s.t.} \]
\[ 0 \leq z - Z_m \leq R_H \text{ and} \]
\[ \forall \ell' = [Z_m', Z_M'] \in H_{ij}(t-1), \]
\[ Z_m' - z > R_H \]
\[ (4.4) \]

4. \( (H_{ij}(t-1) \setminus \{[Z_m, Z_M]\}) \cup \{[z, \gamma]\} \)

5. \( (H_{ij}(t-1) \setminus \{[Z_m, Z_M]\}) \cup \{[Z_m, Z_M']\} \cup \{[z, \gamma]\} \)
\[ \text{if both } 0 \leq z - Z_m \leq R_H \text{ and} \]
\[ 0 \leq Z_m - z \leq R_H \]

6. \( H_{ij}(t-1) \cup \{[z, \gamma]\}, \) otherwise

The 3D space structure of \( \mathcal{P}(t) \), at time \( t \) is given by the set \( S(t) \) of all non empty column maps and their levels.

We need to show that the above definitions capture the dynamic update of the point cloud. Namely, if a new point is detected at \( t \), the 3D structure is eventually transformed by allocating new column maps and new levels, if needed. The following proposition ensure this. Let \( \mathcal{P}(t) \) be the point cloud at time \( t \), let \( S(t) \) be its space structure, and consider a new point \( p = (x, y, z) \), with \( \gamma_{\min} \leq x, y \leq \gamma_{\max} \) and \( p \in \mathcal{P}^- \) (see eq. 4.1). Then

**Proposition 1** There exists a \( (i, j) \) such that \( p \in C(i, j, t+1) \), \( C(i, j, t+1) \subseteq S(t+1) \) and there is a levels set \( H_{ij}(t+1) \) and a level \( \ell \in H_{ij}(t+1) \) such that \( z \in \ell \).

**Proof.** By induction on \( t \).

If \( t = 0 \) then \( \forall (i, j) \subset C(i, j, t) = \emptyset \), by Definition 1. As \( p \in \mathcal{P}(t+1) = \mathcal{P}^- \setminus \{t+1\} \), then there exists a \( (i, j) = ([\frac{x}{2}], [\frac{y}{2}]) \) by the hypothesis that \( x, y \) are within the bounds of the grid. Hence \( p \in C(i, j, t+1) \). Since \( C(i, j, t) = \emptyset \) then \( H_{ij}(t) = \emptyset \). Then \( H_{ij}(t+1) = \emptyset \cup \{[z, \gamma]\} \), with \( z = Z_{\min} = Z_{\max} \), by Definition 2 item 6. Since \( C(i, j, t) = \emptyset \subseteq S(t) \) then \( C(i, j, t+1) \subseteq S(t+1) \).

Let now assume \( t > 0 \). Let \( (i, j) \) be a grid index such that \( (i, j) = ([\frac{x}{2}], [\frac{y}{2}]) \), by the hypothesis that \( x, y \) are within the bounds of the grid. If, for \( t > 0 \), \( C(i, j, t) = \emptyset \) then \( C(i, j, t+1) = \emptyset \cup \{p\} \) and for the levels set the proof is analogous to the one above (by item 1 of Definition 2). Otherwise there exists at least one point in \( C(i, j, t) \) and \( C(i, j, t+1) = C(i, j, t) \cup \{p\} \). Now let us enumerate with an index \( h \) all the levels in \( H_{ij}(t) \), specified by \( \ell^h = [Z_{\min}^h, Z_{\max}^h] \). If there exists a level \( \ell^h \in H_{ij}(t) \) with \( Z_{\min}^h \leq z \leq Z_{\max}^h \) then \( H_{ij}(t+1) = H_{ij}(t) \), namely the levels set is not changed (by item 2 of Definition 2). Otherwise, for each \( h \)-th level compute \( U^h = z - Z_{\max}^h \) and \( u^h = Z_{\min}^h - z \). If there exists some pair at level \( \ell^h \) such that \( U^h \leq R_H \) (resp. \( u^h \leq R_H \)), and for all \( \ell^k \in H_{ij}(t) \) \( Z_{\min}^k \leq z > R_H \) (resp. \( z - Z_{\max}^k > R_H \)) then, the corresponding level is extended toward the max (resp. the min).
4.1 Proposed Methodology

We can note that by the above proposition and the inductive construction, the following

\[ \text{item 5 of Definition 2}. \]

4.1 Proposed Methodology 43

\[
\ell \text{ such that } Z_{\ell} \text{ for all levels } \ell \text{ and } z \text{ is close to both, namely both } 0 \leq Z_{\min} - z \leq R_H \text{ and } 0 \leq z - Z'_{\max} \leq R_H \text{ then the two levels are collapsed, and the } Z'_{\max} \text{ of one level becomes the } Z_{\min} \text{ of the other, removing the previous two levels (item 5 of Definition 2).} \]

Otherwise, if for no levels \( \ell \) the above conditions are verified, it means that a new level needs to be set and thus \( H_{ij}(t+1) = H_{ij}(t) \cup \{(z,z)\} \), with \( z = Z_{\min} = Z_{\max} \), by item 6 of Definition 2. Since \( C(i,j,t) \subseteq S(t) \) then \( C(i,j,t+1) \subseteq S(t+1) \).

(qed)

We can note that by the above proposition and the inductive construction, the following results are obtained:

**Corollary 1** Given a point \( p = (x,y,z)^\top \), if \( p \in C(i,j,t) \) then there is a unique level \( \ell = [Z_{\min}, Z_{\max}] \) in \( H_{ij}(t) \) such that \( Z_{\min} \leq z \leq Z_{\max} \).

Furthermore:

**Corollary 2** Given a column map \( C(i,j,t) \) and its levels set \( H_{ij}(t) \), each level \( \ell \in H_{ij}(t) \) has an upper bound \( Z_{\max} \) which is a maximum and a lower bound \( Z_{\min} \), which is a minimum.

**Corollary 3** Given a column map \( C(i,j,t) \), the levels in \( H_{ij}(t) \) can be totally ordered according to the natural order of the reals.

It follows from the above proposition and corollaries that at any time step \( t \) the empty space between two levels must be greater than \( R_H \), the following proposition states this.

**Proposition 2** Let \( C(i,j,t) \) be a column map at time \( t \), \( H_{ij}(t) \) its levels set and let \( \ell_1, \ldots, \ell_s \) be a total ordering of \( H_{ij}(t) \), according to Corollary 3. Let \( \ell_n = [Z_{\min}, Z_{\max}]_n \) and \( \ell_{n+1} = [Z'_{\min}, Z'_{\max}]_{n+1} \) be two contiguous levels in the order, with \( Z'_{\min} \geq Z_{\max} \). Then \( Z'_{\min} - Z_{\max} > R_H \).

**Proof.** We prove it by contradiction. Suppose that \( Z'_{\min} - Z_{\max} \leq R_H \), then there exist two points \( p = (x,y,z)^\top \) and \( p' = (x',y',z')^\top \) in \( C(i,j,t) \), such that \( z = Z_{\max} \) and \( z' = Z'_{\min} \) and \( z' - z < R_H \).

Since \( p \) and \( p' \) are in \( C(i,j,t) \) there are times \( t' \leq t'' \leq t \) such that the two levels \( \ell_n = [Z_{\min}, Z_{\max}]_n \) and \( \ell_{n+1} = [Z'_{\min}, Z'_{\max}]_{n+1} \) have been modified or built using \( z \) and \( z' \). We consider only the cases of extensions of Definition 2 (items 3 and 4) since by the hypotheses that \( z = Z_{\max} \) and \( z' = Z'_{\min} \) and that the two levels are distinct, all other cases can be reduced to this one. Let us also assume, without loss of generality that \( Z_{\max} \) is obtained at time \( t' \) and \( Z'_{\min} \) is obtained at time \( t'' \), \( t' \leq t'' \leq t \).

Case \( Z_{\max} \) (item 3 of Definition 2): Let at time \( t' \), \( \ell_n = [Z'_{\min}, Z'_{\max}]_n \) and by the hypothesis that \( z = Z_{\max} \) let us assume that \( z \) extends \( Z'_{\max} \). Then \( 0 \leq z - Z_{\max} \leq R_H \) hence, by item 3 of Definition 2 for all levels \( \ell_k \), \( Z'_{\min} - z > R_H \).

Case \( Z'_{\min} \) (item 4 of Definition 2): Let at time \( t'' \), \( \ell_{n+1} = [Z''_{\min}, Z''_{\max}]_{n+1} \) and by the hypothesis that \( z' = Z'_{\min} \) let us assume that \( z' \) extends \( Z''_{\max} \). Then \( 0 \leq Z'_{\min} - z' \leq R_H \) hence, by item 3 of Definition 2 for all levels \( \ell_k \), \( z' - Z''_{\max} > R_H \).
Now, since at time $t''$, $z = Z_{\text{max}}$, and by Corollary 1 and Corollary 2 each level has a max and a min, it must be in particular that $z' - Z_{\text{max}} > R_H$, and since at time $t''$, by item 4 of Definition 2, $z' = Z_{\text{min}}'$ it follows that $Z_{\text{min}}' - Z_{\text{max}} > R_H$, a contradiction. (qed)

This result is quite relevant for the 3D space structure. In fact, it tells that levels, in a single column map, either maintain a specified distance $R_H$, compatible with the robot height, or they collapse into a single level. Therefore, we can now use the column maps and the levels to specify the surface references hence the terrain map.

### 4.1.2 Terrain parameters

In this paragraph we introduce concepts necessary to establish what can be considered terrain or ground and what can be considered top. These concepts are preliminary to further introduce (see next paragraph) the concept of traversability map. This is a quite hard concept in 3D, especially in the case of rescue environments in which there is a non-precise structure of floor and ceilings, as many parts of the geometry of the environment can be collapsed. We shall mainly concentrate on ground structures, since given the above Proposition 2, the results on the ground reference can be extended to model the upper reference too.

In geoscience there are precise criteria to assess terrain stability. According to these criteria, the terrain can be divided into classes usually specified in terms of slope gradient, superficial materials, material texture, material thickness, roughness, slope morphology, moisture conditions and ongoing geomorphic processes. Relying solely on the point cloud information, we can only use three of the above mentioned criteria. Namely, slope morphology, which specifies how harsh is the terrain, terrain density, in terms of amount of points that have been recovered for the specific region and, finally, slope gradient. Note that this classification is given without resorting to the definition of a surface, that is more space and time consuming, as we show in Paragraph 4.2.

We, thus, introduce parameters accounting for these three stability classes, for a general sets of points $W \subseteq \mathcal{P}(t)$.

**Definition 3 (Terrain parameters)** Let $\|W\|$ be the cardinality of $W$, namely the number of points falling into it, with $\|W\| > 3$, and at least three of them non collinear:

1. **Slope.** Let $n = (n_x, n_y, n_z)^T$ be the normal to the plane $\Pi = (n^T, d)^T$ fitted to $W$, via SVD, $d$ the distance to the global coordinates origin. Let $n_G$ be the normal to the plane parallel to the global $x$-$y$ coordinates plane, then the slope $\theta(W) = \cos^{-1}\left(\frac{n^T n_G}{|n||n_G|}\right)$. 

2. **Roughness.** Let $\sigma_W$ be the sample variance of $W$. Let $p_{\Pi}$ be a point on $\Pi$ closest to $p_i$, namely $p_{\Pi} = \frac{d - n^T p_i}{|n|^2} n + p_i$ where its distance from $p_i \in W$, is $\text{dist}(p_i, p_{\Pi}) = \frac{|d - n^T p_i|}{|n|}$. Then the roughness $\kappa(W) = \sigma_W \sum_{p \in W} \text{dist}(p_i, p_{\Pi})$.

3. **Density.** Let $\Pi$ be the above defined plane, the radius $s$ of $W$ is defined as follows:

$$s = \max\{v \mid v = (\arg\min_{p_i}(\text{dist}(p_i, p_{\Pi}))^2 + \arg\min_{p_i}(\text{dist}(p_j, p_{\Pi}))^2)^{1/2}\} \quad (4.5)$$
4.1 Proposed Methodology

Now, let us order $W$ according to $z$, let $Z$ be its upper bound and $U$ its lower bound, and define $\zeta = Z - U$. The density is $\rho(W) = \frac{||W||}{f(\pi^2 \zeta)}$, here $f$ is a normalization function transforming the approximated volume into number of points.

Then $(\rho, \theta, \kappa)$ are the terrain parameters for a set $W$ of points in $\mathcal{P}(t)$. Now, we have to evaluate these parameters once the points $W$ are mapped within the defined structures of $S(t)$, and within new structures of $S(t)$ that we define in the sequel, to generate traversable paths and traversable maps.

Indeed, for each column map and level, as defined in the previous paragraph, we want to define a primitive reference ground surface according to constraints on the parameters, and an 8-connected region around a fixed level. Let us consider a column map $C(i, j, t)$, its levels set $H_{ij}(t)$, a fixed level $\ell = [Z_{\text{min}}, Z_{\text{max}}]$ and the eight connected regions of $(i, j)$, namely $Q = \{(i-1, j-1), \ldots, (i+1, j+1)\}$, with $(i, j) \notin Q$ and the corresponding column maps at time $t$. The neighbors for $\ell$ are:

**Definition 4 (Neighbors of $\ell$)** Let $Z_{\text{max}}$ be the upper bound of $\ell$ and let $u \in Q$. We define:

$$N_0(Z_{\text{max}}, Q, t) = \{ \ell' u \in H_{ij}, u \in Q | \ell' u = [Z_{\text{max}} - Z_{\text{min}}, |Z_{\text{max}} - Z_{\text{max}}| < \eta R_H) \}$$ (4.6)

Here $\eta$ is a robot physic parameter ensuring its static stability, when moving between levels. We can note that the neighbors of a level $\ell$ are well defined since for any $Z_{\text{max}}$ there exists only one level in a levels set $H_{ij}(t)$, for a column map $C'(i, j, t)$, which can satisfy the constraint $|Z_{\text{max}}' - Z_{\text{max}}| < R_H$, because of Proposition 2, therefore in particular for $\eta R_H$, as $\eta R_H \leq R_H$.

Therefore, given the neighbors of a level for a specific column map, we can now define when a neighborhood is considered a stable ground terrain. Note that all the parameters should be suitably accommodated for a specific UGV.

**Definition 5 (Neighborhood)** Let $\xi$ be a fixed threshold for supporting the density of the points. Let $N_0(Z_{\text{max}}, Q, t)$ be the neighbors of $\ell$ at time $t$. We define:

$$\mathcal{N}_{ij}(Q, \ell, t) = \{ p \in C(u, t) | u \in Q, z \in \ell u \in N_0(Z_{\text{max}}, Q, t), 0 \leq Z_{\text{max}} - z \leq \xi \cup \{ p \in C(i, j, t) | 0 \leq Z_{\text{max}} - z \leq \xi \}$$ (4.7)

This set forms a stable neighborhood if the following conditions on the parameters $(\rho, \theta, \kappa)$ are satisfied, note that these values can be adjusted according to the UGV structural constraints:

1. $n2\pi - \alpha \leq \theta(\ell) \leq n2\pi + \alpha, n = 0, 1, \ldots$ Here $\alpha$ is defined according to the flippers affordable angle.
2. $\rho(\mathcal{N}_{ij}(Q, \ell, t)) > 0.4$.
3. $\kappa(\mathcal{N}_{ij}(Q, \ell, t)) < 0.5$.

A stable neighborhood is the primitive element for specifying a path hence a traversability map, which is discussed in the next paragraph, together with the cost of a traversability map. Note that the condition that the $z$ values of the points in this set do not vary on the whole level $\ell$ but on a subset around the upper bound is essential to effectively determine the existence of a supporting region.
4. Terrain Traversability in Rescue Environments

Figure 4.4. (a) The point cloud at time \( t \) and the ramp path with cost map. Colormap: from red to blue denotes from more to less stable path; (b) the robot facing the ramp, here the operator has used the cost map to draw a path given the affordable cost.

4.1.3 Traversability and cost Map

In this paragraph we introduce the concept of traversability map, as the surfaces that can be segmented out from the point cloud to indicate areas that the robot can stand on and traverse. To this end we give first a definition of path, then of traversable region and finally of traversability map and its associated cost map.

**Definition 6 (Stable Path)** Let \( p = (x, y, z) \top \in C(i, j, t) \) and \( p' = (x', y', z') \top \in C(w, k, t) \). Let \( z \in \ell \) and \( z' \in \ell'_{m' M'} \). A stable path between \( p \) and \( p' \), denoted \( \Gamma(p, p') \) is defined inductively as follow:

1. \( \Gamma(p, p) \) is a stable path.
2. \( \Gamma(p, p') \) is a stable path if \( p, p' \in \mathcal{S}_{ij}(Q, \ell, t) \) for some \( (i, j) \) and \( Q = \{(i-1, j-1), \ldots, (i+1, j+1)\} \), with \( (i, j) \notin Q \).
3. \( \Gamma(p_a, p_b) \) is a path if there exists a point \( p_k \) and paths \( \Gamma(p_a, p_k) \) and \( \Gamma(p_k, p_b) \).

We see that to build a path from the current level, it is required to build neighbors and verify their stability and proceed inductively. It follows that a traversable region is simply the collection of all paths from a point and the level of its \( z \)-value and the points that belongs to affordable levels. Note that we have not introduced the time in the path definition. Indeed, as the point cloud \( \mathcal{P}(t) \) is updated, clearly new paths can be created and old path might be removed, and this depends from the neighbors and stability update.

**Definition 7 (Traversable region)** A traversable region is a graph \( G(A, B, t) \) such that, for each node \( a \in A \), there exists a column map \( C(i, j, t) \) which is the center of a stable region \( \mathcal{S}_{ij}(Q, \ell, t) \) and for each vertex \( (a_i, a_j) = b \), with \( b \in B \) there is a path between \( a_i \) and \( a_j \).

We can finally introduce the definition of traversability map, as follows.

**Definition 8 (Traversability Map)** A traversability map is the set of all traversable regions obtained from a point cloud \( \mathcal{P}(t) \) at time \( t \).
4.1 Proposed Methodology

The inductive construction we have provided so far, clearly facilitates the map updating at any fixed time interval, and it allows for very simple implementations.

However, parameters might be more nasty than expected because there is a great amount of noise in the point cloud, and random points can fill empty space blocking navigation. On the other hand we might want a traversability estimation yet when the map would not be clearly established. Therefore, to overcome these drawbacks we can relax the stability constraints and rely also on learned distributions, for fixed neighbors numbers, for example from 4 to 8, and for the specified parameters \((\rho, \theta, k)\), see Definition 3. More precisely, we introduce a cost map that provides a risk function on a neighborhood \(T\) whose stability (see Definition 5) can be approximately established.

We consider that a multivariate Gaussian mixture is learned for sets of terrain observations, for the terrain parameters \((\rho, \theta, \kappa)\), and with different number of components, according to the number of neighbors considered for a specific level. More precisely, let us denote \(\theta \otimes 1_n\) the Kronecker product that maps \(\theta\) into a vector of \(n\) elements and let \(p_n\) be the vector \(n \times 3\) of the points \(p\) in a set \(W\), with \(n\) the number of points in \(W\). The training set of observations is:

\[
V_n = (p_n, \theta \otimes 1_n + \epsilon_1, \rho \otimes 1_n + \epsilon_2, \kappa \otimes 1_n + \epsilon_3)
\]  

(4.8)

Here \(\epsilon_1, \epsilon_2, \epsilon_3\) are noise vectors with variance 1 and mean \(\mu_i\).

Given number of components \(k \in \{3, \ldots, 8\}\), according to the required number of neighbors for a level, the mixture parameters \(\Theta^0 = (\Sigma^0_1, \mu^0_1), \ldots, (\Sigma^0_k, \mu^0_k)\) are estimated via maximum likelihood using as initial mean values \(\mu_\xi, \mu_\theta, \mu_\rho, \mu_\kappa\). Here \(\mu_\theta, \mu_\rho, \mu_\kappa\) are the mean of the admissible ranges for the stability parameters, while \(\mu_\xi\) is related to the spread of the terrain to be evaluated around the upper bound of each level in the neighbors (see Definition 5), for a fixed value \(\xi\). Namely for points \(p = (x, y, z)^T \in W\),

\[
\mu_\xi = (1/n) \sum_{i=1}^n \{p = (x, y, z)^T : z_\xi \leq z \leq Z_{\text{max}}\}
\]

Then, for each component \(i = 1, \ldots k\)

\[
\mu_i = (\mu_\xi, \mu_\theta, \mu_\rho, \mu_\kappa)^T
\]  

(4.9)

The sample variance is initialized accordingly. It follows that the so learned multivariate mixtures, for each component size, namely number of neighbors, estimates the probability for a set of points to be within the stability requirements.

Figure 4.5. (a) Staircase Scenario. Stairs are indicated in blue, the cost is high as the stability is uncertain. The operator is visible at the bottom of the stairs; (b) traversable map overview, including stairs and second floor accessibility. The robot is facing the stairs: it is at the second level.
Now, given a neighborhood \( \mathcal{T}(Q, \ell, t) \) the probability that the neighborhood is stable depends on the probability that each point in the neighborhood belongs to a stable set. This probability is given by the learned multivariate mixture, where the mixture weights \( w_i \) are computed on line, considering the number of points of each neighbors.

The above evaluation can be extended to the set of points of the neighborhoods \( \mathcal{T}' \) of a traversable region \( G(A, B, t) \), at time \( t \), by juxtaposing the observations of each neighborhood into a single matrix \( V \) of size \( (\sum_i n_i) \times 6 \). Note that a traversable region is always referred to by a level, because of the inductive definitions rooted in the set \( \mathbb{N}_0 \), see Definition 4.

Therefore, for each node of the graph \( G(A, B, t) \) the probability that a point belongs to a stable neighborhood is computed, and the cost map is simply defined as follows.

**Definition 9 (Cost Map)** Let \( V \) be as defined above, let \( G(A, B, t) \) be the graph of all the neighborhoods of the fixed traversable region, and let \( \Theta = (\Sigma_1, \mu_1), \ldots, (\Sigma_k, \mu_k) \) be the estimated mixture parameters with \( k \) components. The mixture \( g(v|\Theta) \) returns the probability that an observation \( v \) belongs to a stable region. Hence, the cost of the observation is \( 1 - g(v) \) and of the map is \( 1 - g(V) \).
4.2 Results

The algorithm is implemented for the tracked vehicle described in Chapter 2, Figure 2.4. Tests have been performed on different scenarios including stairs, slopes, ramps and rugged terrain. All tests performed were teleoperated. The operator never anticipated a path and followed the direction of the cost map, illustrated in the figures. In particular, for the experiments on the ramp (see Figure 6.6(a)) the operator has used the space segmentation provided by the algorithm to trace a path, which the robot has faithfully followed. The algorithm takes as input the point cloud every 0.1 seconds, according to the 10 Hz refresh rate of the robot pose estimation. Terrain estimation is updated at the same rate. Processing starts at the current robot location and iteratively increases the path (see Definition 5) in the direction of motion and it switches to other levels when updates of the current path converge.

For each experiment different resolution choices have been tested and the results are illustrated in the graph of Figure 4.8. Best comprise processing-time, number of points and accuracy with the widest band of neighbors between 6 and 8 is obtained with a resolution between 0.8m and 1.2m. Segmentation results according to the terrain structure, and labeled with the estimated cost, with color code from red, lowest cost to blue highest cost are illustrated in Figures 6.6(a) and 4.5 and taken on the earthquake training area of Prato (see Figure 7.5).

We compared our results also with respect to surface reconstruction. In Figure 4.7 we show the variation of processing time with two algorithms. The first one is our proposed algorithm (shown in red), and the other algorithm correspond to the work done by Marton et al. [165], with varying parameters. We have considered the number of neighboring points as varying parameter (over 50, 100 and 200 elements) of Marton and colleagues work.

As can be seen from the figure 4.7 the required time for triangulation of the point cloud
Figure 4.8. The graph illustrates varying resolution (in red), accuracy (in blue) and number of points processed (in green), and the processing time required while these parameters vary. Number of points are normalized with respect to resolution namely \( N_p = N(p) \cdot \max(r) / \max(N(p)) \) and analogously for the accuracy. The strips indicate the neighbors size: 2 (blue), 4 (green), 6 (yellow) and 8 (red).

is at least 0.2 seconds, which is not close to real-time. Whereas, the proposed algorithm performs around 0.03 seconds, which is close to optimal, considering real-time delay for a robot sending information to the remote base. Moreover, this step is only for the triangulation of the point cloud. If we consider the successive steps, that is, merging the continuously updating point cloud, within the triangulated surface, and estimation of static traversability cost, then the algorithm incurs in further delay and time consuming steps.
Chapter 5

Help me help you: how to learn intentions, actions and plans

In this chapter we outline a collaboration model between human-robot in which the final goal is to learn the best actions needed to achieve the required goals (in this case, reporting hazards due to a crash accident in a tunnel, identifying the status of victims and, possibly, rescuing them). The collaboration is here viewed as a learning process involving the extraction of the correct information from the instructor behaviours. The instructor communicate his actions both visually and with the aid of his comments delivered while executing the actions. In particular, actions and intentions are obtained by elaborating on the instructor path, while inspecting the accident place, what a fire fighter instructor looks at, together with his running commentaries recorded via the Gaze Machine (GM). The GM, worn by the instructor, is a complex wearable device, illustrated Figure 5.1, that allows to gather several perceptual data from a subject executing a task [12, 162].

The extraordinary vantage point obtained by the Gaze Machine enables an agent to observe, at any time step, not only what effectively the tutor is doing and communicating it but also how the tutor adapts his behaviours, by instantiating with common sense the prescribed laws, that is, those usually regulating his conduct in similar circumstances. It allows to get his intentions, tracking the relationship between saccades and motion towards a direction, namely something interesting in the scene. Finally, by the joint localisation of

![Figure 5.1. The instructor Salvo Candela with the Gaze Machine, rescuing a victim](image-url)
the instructor’s gaze, his current position and his running commentaries and the noise in the scene, it is possible to infer affordances, namely a well defined sequence of the preferred interactions between the instructor and the surroundings.

From these extremely rich source of information an agent is in the condition of learning a well temporised sequence of actions and thus, to generate a suitable plan, in order to correctly operate in a difficult and hazardous environment. In this chapter we describe at a very general level, the following aspects of this learning and generation process:

1. We define two paths, the instructor path in the scene obtained by visually localising the instructor via the GM (Paragraph 5.1) and the instructor gaze path, obtained via the stereo pairs mounted on the GM and the cameras staring at the eye pupils.

2. From the two paths and a suitable segmentation and clustering of motion directions (of both body and head), both the motion and vision actions are obtained and labelled. On the other hand, as the instructor actively (and benignly) comments his behaviours, all the manipulation actions are identified by the running commentaries and the association of head motions and body position (Paragraph 5.2). Indeed, actions are defined as processes with a start and an end action, and with time varying.

3. A plan library of possible activities and affordances, according to the context, is defined a priori with the contribution of the instructor. In particular, the “what to do in such a situation” can be earlier formulated. According to the prior plan library and the effective sequence accomplished, following the instructor behaviours induced by common sense a flexible plan of action processes is generated, where the timelines are settled according to the flexible instantiation provided by the difference between coded rules and common sense (Paragraph 5.3).

A schema of the model is given in Figure 5.2.

The problem of inferring a plan from the observations of actions, in the context of knowledge representation, is called plan recognition, and it has been earlier introduced by [231]. For a review of the consistency based and probabilistic based approaches to plan recognition see [7]. Geib in [88] introduced a method of plan-recognition where plan-library is first converted to a lexicon similar to that used in combinatory categorical grammar. By this way author is able to introduce concept of headedness, which avoids early commitments to plan and goal hypothesis in the process of plan-recognition, which eventually results in increased speed of the plan-recognition system. On the other hand in the realm of learning and computer vision the analogous concepts of acting based on observations have been specified as action recognition, imitation learning or affordances learning, as mainly motivated by the neurophisiological studies of Rizzolatti and colleagues [197, 84] and by Gibson [100, 125]. Reviews on action recognition are given in [172, 213] and on learning by imitation in [229, 6].

The two approaches have, however, evolved in completely different directions. Plan recognition assumed actions to be already given and represented, in so being concerned only in the technical problems of generating a plan, taking into account specific preferences and user choices, and possibly interpreting plan recognition in terms of theory of explanations [45]. On the other hand action recognition and imitation learning has been more and more concerned with the robot ability to capture the real and effective sequence and to adapt it to changing contexts. As noted by Kruger and colleagues in [141] the terms action and intent.
5.1 Visual Localization

Two paths can be obtained by the instructor running in the disaster theatre. The first concerns the position of his body and the second the position of his gaze, not mentioning the position and direction of his head obtained via the inertial sensor placed on the GM.

For the instructor position we relied on both the extended Kalman filter (EKF)-base visual slam introduced by [55] and the particle filter ones introduced by [214], but suitably extended to cope with the specific head motions and consequent change blindness and and the advantages of the calibrated stereo pairs. The challenges that we face in this procedure stem from the inherent scene peculiarities of rescue environments as well as the loosely constrained movement of the camera setup which follows the movement of the instructor’s head. In detail, the scene characteristics of a rescue environment include a wide

Figure 5.2. Schema of the flow of information and processing to learn actions from the collaboration instructor-robot, starting from the gaze machine.

recognition, in plan recognition, often obscure the real task achieved by these approaches. In fact, as far as plan recognition assumes an already defined set of actions the observation process is purely indexical. On the other hand the difficulties with the learning by imitation and action recognition approaches is that they lack important concepts such as execution monitoring, intention recognition and plan generation.

The contribution of this chapter fosters a more tight integration between the two approaches wherein the actions are segmented via the Gaze Machine and the instructor running commentary and the consequent plan recognition that is based on these actions.

5.1 Visual Localization

The contribution of this chapter fosters a more tight integration between the two approaches wherein the actions are segmented via the Gaze Machine and the instructor running commentary and the consequent plan recognition that is based on these actions.

Two paths can be obtained by the instructor running in the disaster theatre. The first concerns the position of his body and the second the position of his gaze, not mentioning the position and direction of his head obtained via the inertial sensor placed on the GM.

For the instructor position we relied on both the extended Kalman filter (EKF)-base visual slam introduced by [55] and the particle filter ones introduced by [214], but suitably extended to cope with the specific head motions and consequent change blindness and and the advantages of the calibrated stereo pairs. The challenges that we face in this procedure stem from the inherent scene peculiarities of rescue environments as well as the loosely constrained movement of the camera setup which follows the movement of the instructor’s head. In detail, the scene characteristics of a rescue environment include a wide
5. Help me help you: how to learn intentions, actions and plans

range of lighting conditions and a plurality of solid but also non-solid obstacles (such as smoke). The position-orientation of the camera setup is also highly variable as the instructor rushes within the accident area due to looming hazards. We can note that because most of the computational effort is carried out off-line, we can take advantage of the techniques developed in the context of Structure and Motion recovery in order to deal with the higher variance in the camera motion and particular lighting conditions. The selection of a stable sequence of frames, that turn out to be key frames not only for the localisation and mapping process but also for action segmentation is a crucial step (see Figure 5.3).

Furthermore, we rely on well known methods for feature extraction and optical flow to predict the displacement of the tracked features and bundle adjustment between pairs of stereo images for motion estimation. In SAM problems, 3D structure is used to estimate the camera pose by resectioning. Thus, the computation of the motion is also complemented by the usage of dense disparity maps. The process goes through three main steps:

1. build a Viewing Graph (see Fig. 5.4) and compute the Essential Matrices between each pair of views, obtained from the key frames set [116];

2. given the estimated position at time \( t \), factorise the Essential Matrices to produce observations for an EKF, which provides the current position at time \( t' \);

3. bundle adjust among the estimated sequence of 3D structure and camera motion.

**Figure 5.3.** Key frames extraction. First row: acquired frames through time. Second row: acquired accelerometer absolute mean amplitude through time. Third row: frames corresponding to accelerometers peaks (movement from a LOI to another) are discarded. Forth row: features are extracted by the different key frames. If a lot of features match between the key frames of different scenes, this means that those scenes are the same and thus they are grouped together under the same label.
5.2 Segmentation and Action Maps

In this paragraph we discuss how we can segment the data acquired using the Gaze Machine to obtain a sequence of performed actions. We shall also discuss the intention recognition via the coup d’oeil, i.e. how it is possible to extract the instructor’s intention on the basis of his fixations and the spoken running commentaries.

A library of possible activities and affordances has been compiled in advance with the contribution of the instructor but, due to the high changeability of the scenario, the instructor will not follow a predefined, prioritised sequence of actions. The decision on

Steps 1-2 provide a local consistency between different temporal frames. On the other hand they do not take into account sudden movements, which are filtered out in the key frame selection. In order to maintain a global consistency a bundle adjustment step is required where the re-projection error is minimised. Using the above described visual-based SLAM we are able to obtain an estimate of the instructor’s path which, in turn, is used to derive the gaze path within the scene. It is interesting to note that due to inhibition of return, typical of the gaze when a salient feature come up hiding previous saliency levels, often a large amount of images are required in order to effectively track features. The viewing graph will tell on which configurations it is possible to rely in so avoiding the constraints induced by a sequence of pairs of images. Given the position of the instructor the localisation of his gaze is immediately obtained by the stereo pair.

In the following paragraph the two paths are going to be segmented according to the recognition of actions from (i) the running commentary and (ii) the video sequence. The recognition of the actions will in turn enable to infer the spatio-temporal information of an action, Indeed, the two path prove to be essential for action segmentation as they can correctly specify the where and when an action is performed as well as the corresponding spatio-temporal information of the instructor’s gaze: what and when the instructor is gazing at, during a particular action.

**Figure 5.4.** Example of the Viewing Graph used for the visual localisation. $L_t$ and $R_t$ are respectively the left and the right scene cameras at time $t$. $E_i$ is the Essential Matrix between different views.
5. Help me help you: how to learn intentions, actions and plans

what to do next is taken on the run, according to the task (i.e. plan the rescue) and the affordances characterising the scenario. The current instructor’s intention involves what he is actually able to capture via attention. A saccade that is directed toward a location that is not involved in the current action may indicate a shift in the instructor’s attention; depending on the associated saliency, this may or may not fire a head movement. However, also the information provided by the peripheral vision is enough to increase the situation awareness and take decisions. We, thus, introduce the concept of coup d’oeil to refer to those time instants in which something in the peripheral view fires a running commentary reporting something relevant in the scene.

The Gaze Machine records the instructor’s saccade sequence by tracking his gaze in space. This is accomplished by projecting in the 3D scene the estimated point of regard. Scene structure is recovered via the Gaze Machine stereo rig while both pupils are tracked to extract visual axes (see Figure 5.1). A first kernel-based segmentation is performed to extract the fixation scan path from the acquired sequence of 3D points of regard. The main problem we address in this step is taking into account the instructor 3D position, as the 3D fixated points changes if the instructor moves.

The segmentation of the image flow acquired from the experienced firefighter is needed as a prior to further analysis of his actions. Key frame selection has been thoroughly investigated in the context of SAM recovery [210]. In this work we face the problem in the case of wearable cameras and unpredictable human motions. When performing some activity, a person is acquiring information (by gazing) in some important location in order to perform actions and then he moves to another location of interest (LOI). During the movement between two LOIs the acquired images are of little interest as they are most of the time fuzzy and very unstable. Moreover, the extensive visual disruptions due to the firefighter fast motion imply a high probability of change blindness [240] which decrease again the usability of the gaze data acquired during those periods. It is thus important to discard the frames which are recorded during the LOI change in order to extract the more stable scene (see Figure 5.3, third row). Finally, the firefighter can move from one LOI to another and then come back to the first LOI, or he can also be disturbed by some important bottom-up distractor which makes him turn his head and then he can look again to the previous scene. This shows that two stable scenes are not necessary different scenes or LOIs (Figure 5.3, forth row).

A two-step approach can be used to extract meaningful scenes or key frames from the video flow: first the data from the accelerometer can provide cues on the head stability and then computer vision techniques are able to recognise already seen scenes or novel scenes. Figure 5.3 illustrates the process. The top row shows the successive frames through time. The shape of the absolute mean amplitude of the accelerometers located in the gaze machine is presented on the second row and shows picks during the firefighter movements between two LOIs and valleys during his stay in the same LOI. By discarding the frames which correspond with the accelerometer peaks, it is possible to keep only the stable scenes. Feature extraction and matching between those different scenes provide information to group together the scenes which are the same. If the features extracted from some key frames of one scene match a number of features above a given threshold on some key frames from another scene, this means that the two scenes are the same as it can be seen on Figure 5.3, forth row. In that case the two scenes are labelled with the same label.

Along with the instructor changes in position, pose and the running commentaries, 3D fixations are used to detect the starting/ending of an action. We are interested in producing
5.3 From actions to flexible plans and plan recognition

The Actions map is constituted by a timeline indicating the time stamp of each action, the temporal relations among actions and the spatial cluster they belong to. The spatial cluster is obtained by the instructor path (see Paragraph 5.2). Using the rules specified in the plan library and the Action Map the instructor plan execution can be suitably labelled for planning.

For example, according to the plan recognition algorithm of ([88]), and using the
specified plan library, we first generate a combinatory categorical grammar (CCG) type plan lexicon which maps observations to CCG categories. The algorithm results in a set of explanations, mentioning a goal and an ordered sequence of actions. The choice of assigning categories to observations is made according to specified head value. Headedness is a powerful method of controlling the space of possible explanations to be considered during the plan-recognition procedure.

In any case we mainly base the mapping from the Action Map to possible plans via a temporal network compiling constraints and compatibilities within the Situation Calculus. Temporal relations specify how activities, such as looking at a victim and of opening the car door are correlate along time. For modelling both temporal constraints and cause-effect relations between activities we adopt, in fact, Temporal Flexible Situation Calculus ([79, 203]), accommodating Allen temporal intervals, multiple timelines among actions and concurrent situations. It intermediates between Situation Calculus formulae and temporal constraint networks. For example, the temporal relations illustrated in Figure 5.5, first row, last image, can be expressed by the compatibilities

\[ T_c = \text{comp}(\text{lookingAt(victim, } t)\text{, } [\text{during, openingDoor(car, } t)]) \]

Here the compatibility states that the activities look a victim, involving vision actions, and opening the car door have to be performed according to the during temporal relation. The temporal network associated with the compatibilities \( T_c \) is represented in Figure 5.6. Therefore a way to generate a plan is to exploit the obtained temporal network and the flexible plan in the Situation Calculus.
Chapter 6

An Augmented Reality approach for trajectory planning and control of tracked vehicles in rescue environments

Tracked vehicles are currently used in search and rescue, military, agricultural and planetary exploration applications where terrain conditions are difficult and unpredictable. They are better suited for such tasks than wheeled vehicles due to the larger contact area of tracks with the ground, which provides better traction on harsh terrains [54, 42]. These environments are often inaccessible or considered too dangerous for humans to operate in, thus requiring the tracked vehicle to be endowed with autonomous navigation, safe locomotion and human-robot interaction capabilities to assist humans in complex tasks such as rescue, scouting or transportation.

Autonomous navigation in harsh terrains is still a daunting challenge because of the difficulty in capturing and representing the variability of the environment. It relies on both 2D and 3D models of the environment to cope with perception and motion planning [252, 140, 113, 176]. Moreover, autonomous navigation requires the ability to decide whether parts of the environment can be traversed or have to be bypassed [161, 63]. Although these approaches have been demonstrated to work online, on real terrain data, their complexity significantly reduced the performance of the activities of the robot, which is an important factor in a search and rescue mission.

Tracked vehicles are difficult to control since they are steered by skid steering [260]. The skid steering principle is based on controlling the relative velocities of both tracks, much in the same way as differential drive wheeled vehicles. However, control of tracked locomotion poses a more complex problem because variation of the relative velocities results in slippage between the tracks and the ground. The slippage generates large accumulated positioning errors, thus affecting the accuracy of the odometry as well as the performance of the trajectory tracking controller of the tracked vehicle. Under this perspective, trajectory tracking for safe locomotion plays a crucial role in making effective the motions of the tracked vehicle to autonomously accomplish the navigation task in such unstructured environments. Several methods to improve the accuracy of the odometry, based on slip-compensation, and to enable the tracked vehicle to follow a given trajectory have been proposed [237, 163, 73]. These
6. An Augmented Reality approach for trajectory planning and control of tracked vehicles in rescue environments

Figure 6.1. Rescue scenarios

methods require the accurate measurements of the vehicle velocity using internal sensors (e.g., inertial sensor) and friction coefficients that change according to the ground surface. In addition, these methods are difficult to implement on a robot.

For rescue tasks human-robot interaction (HRI) is essential [183, 145]. HRI models robot and human perception in a reciprocal design maximizing the information provided to the human and the feedback given to the robot. Joint perception is very helpful to accomplish tasks [47, 150]. In fact, sharing knowledge is the basis of both robot and human situation awareness [66, 232].

In this work we propose a framework for trajectory planning and control for tracked vehicles in rescue environments, based on Augmented Reality (AR). This framework provides the human operator with an AR-based interface that facilitates both 3D path planning and obstacle negotiation. The interface offers the capability to track the movements of a marker pen, handheld by an operator. These movements can be converted into the trajectory a tracked vehicle can follow. The drawn trajectories are immediately overlaid on the displayed 3D map. The operator can modify the drawn path by partially changing it or by completely replanning the path. The framework implements a trajectory tracking controller, based on input output linearization via feedback, to allow the tracked vehicle to autonomously follow the potential path, committed by the operator [238]. The feedback of the controller is provided by a localization system which exploits the performance of a Dead Reckoning System together with the accuracy of an ICP-based Simultaneous Localization and Mapping in pose estimation [212], to determine the pose of the tracked vehicle on the 3D map, in real-time.

The chapter is organized as follows. In Paragraph 6.1 we introduce the kinematics model of the tracked vehicle. In Paragraph 6.2 we describe the robot localization system, taking care of the necessary feedback to the vehicle tracking controller. The control strategy for the trajectory tracking, is described in Paragraph 6.3. In Paragraph 6.4 we present the AR-based approach for trajectory planning and generation. Validation experiments are discussed in Paragraph 8.7.5.

6.1 The robot model

The tracked vehicle can be approximated to a differential drive robot, which is modelled as a unicycle. The higher level control inputs to the robot are specified in terms of the linear
velocity $v$ and angular velocity $\omega$. The differential drive model is given as follows:

$$
\begin{bmatrix}
\dot{x}_R \\
\dot{y}_R \\
\dot{\theta}_R
\end{bmatrix} =
\begin{bmatrix}
v \cdot \cos \theta \\
v \cdot \sin \theta \\
\omega
\end{bmatrix}
$$

(6.1)

Figure 6.2 shows the position of the centre of mass of the robot in a 2D plane. The orientation of the robot is given by the angle between the local robot reference frame $x'$ axis and the global $x_G$ axis. The linear and angular velocity are given as input to the lower level control which computes the velocities $v_l$ and $v_r$ of the left and right tracks of the robots, according to the following inverse kinematics:

$$
v_l = \begin{cases} 
v - \frac{L \omega}{2} & \text{if } |v_l| \leq v_{max} \\
v_{max} & \text{if } v_l > v_{max} \\
-v_{max} & \text{if } v_l < -v_{max}
\end{cases}
$$

(6.2)

$$
v_r = \begin{cases} 
v + \frac{L \omega}{2} & \text{if } |v_r| \leq v_{max} \\
v_{max} & \text{if } v_r > v_{max} \\
-v_{max} & \text{if } v_r < -v_{max}
\end{cases}
$$

(6.3)

Here $L$ is the distance between the robot tracks and $v_{max}$ is the maximum track velocity. Both Eq. (6.2) and Eq. (6.3) bound the velocity inputs of the tracked vehicle.

### 6.2 Robot Localization System

The localization system (LS) provides to the trajectory tracking controller, the pose $q_R(t)$ of the robot, in a global coordinate frame. This system relies on two main localization sub-systems: (1) a dead reckoning navigation system (DRNS) and, (2) an ICP-based simultaneous localization and mapping (ICP-SLAM). In addition, DRNS is endowed with a complementary filter (CF) which provides an estimation of the Euler angles of the body of the robot from data fusion of odometry and inertial data coming from the IMU sensor. The localization system compensates the pose estimation error, accumulated by DRNS, by integrating the laser odometry provided by ICP-SLAM and so it determines the robot pose correctly. The pose refinement process can be summarized as follows.
6. An Augmented Reality approach for trajectory planning and control of tracked vehicles in rescue environments

Figure 6.3. 3D Pose estimation based on (a) IMU sensor data, provided by DRN, and (b) point clouds from 3D Laser sensor from ICP-SLAM.

DRNS provides the localization system with an estimation $\hat{q}_{\text{odom}}(t)$ of the robot pose, with respect to the robot odometry reference frame, at $f_{\text{DRNS}} = 10$ Hz. This estimation is subject to a cumulative error, which increases with the travelled distance (see Figure 6.3(a)). The error is due to the inaccurate measurements in heading rotation and to the lack of a global reference.

In parallel, on the basis of the laser odometry information, ICP-SLAM provides the localization system with an estimation $\hat{q}_{\text{map}}(t)$ of the pose of the robot, referred to the map reference frame, as well as the coordinate transformation $T_{\text{map}}(t)$, between the robot odometry reference frame and the map reference frame. The map reference frame is assumed to be the global reference frame. Due to the drift and the slippage this transformation is not fixed. Both $\hat{q}_{\text{map}}(t)$ and $T_{\text{map}}(t)$ are updated by ICP-SLAM at $f_{\text{ICP-SLAM}} = 1$ Hz. The output frequency $f_{\text{ICP-SLAM}}$ is limited by the scanning speed of the laser sensor.

Given the estimation $\hat{q}_{\text{odom}}(t)$ of the pose of the robot, provided by DRNS, and given the current transformation $T_{\text{map}}(t)$, from ICP-SLAM, LS computes the pose $q_{\text{map}}(t)$ of the tracked vehicle, at time $t$ as follows

$$q_{\text{map}}(t) = T_{\text{map}}(t) \cdot \hat{q}_{\text{odom}}(t) \tag{6.4}$$

LS refines the robot pose updating the transformation $T_{\text{map}}(t)$, as soon as ICP-SLAM updates it. Figure 6.3(b) shows the effects of the pose refinement process. This approach allows LS to provide the pose of the robot $q_{\text{map}}(t)$, as a feedback to the trajectory tracking controller, at $f_{\text{DRNS}}$.

6.3 Trajectory tracking control

Let $q_d(t) = \begin{bmatrix} x_d(t) & y_d(t) & \theta_d(t) \end{bmatrix}^T$ be the position and orientation of a virtual reference frame on the desired Cartesian trajectory, with respect to a global reference frame. Let $q_R(t) = \begin{bmatrix} x_R(t) & y_R(t) & \theta_R(t) \end{bmatrix}^T$ be the pose of the local robot reference frame expressed in the global reference frame, provided by the localization system. The trajectory tracking controller of the robot has to generate the control inputs $u(t) = \begin{bmatrix} v_d(t) & \omega_d(t) \end{bmatrix}^T$ in order to asymptotically stabilize to zero the tracking error $e(t)$ between $q_d(t)$ and $q_R(t)$, which is defined as follows:
6.4 Trajectory planning and generation

The proposed approach to the design of trajectory tracking controller of the tracked vehicle is based on input-output linearisation via feedback [135, 238].

Let \( B = [x_B(t) \ y_B(t)]^T \) be a point displaced at distance \( b > 0 \) along the main axis of the tracked vehicle (see Figure 6.2), whose Cartesian coordinates are

\[
x_B(t) = x_R(t) + b \cos \theta_R(t) \\
y_B(t) = y_R(t) + b \sin \theta_R(t)
\]

(6.6)

Given Eq. (6.1) and Eq. (7.2), the kinematics of the point \( B \) on the tracked vehicle is given by

\[
\begin{bmatrix}
\dot{x}_B(t) \\
\dot{y}_B(t)
\end{bmatrix} =
\begin{bmatrix}
\cos \theta_R(t) & -b \sin \theta_R(t) \\
\sin \theta_R(t) & b \cos \theta_R(t)
\end{bmatrix} \begin{bmatrix}
v \\
\omega
\end{bmatrix} = T(\theta_R(t)) \begin{bmatrix}
v \\
\omega
\end{bmatrix}
\]

(6.7)

Since \( T(\theta_R(t)) \) is not singular for any \( b > 0 \), then the input-output linearization via feedback can be obtained by computing the transformation of the control inputs as follows

\[
\begin{bmatrix}
v \\
\omega
\end{bmatrix} = T^{-1}(\theta_R(t)) \begin{bmatrix}
u_1 \\
u_2
\end{bmatrix}
\]

(6.8)

Let \( \mathbf{q}_{d,B}(t) = [x_{d,B}(t) \ y_{d,B}(t) \ \theta_{d,B}(t)]^T \) the pose of the virtual reference frame on the desired Cartesian trajectory, displaced at the same distance \( b > 0 \) along the tangent to the trajectory at \( \mathbf{q}_d(t) \). Let \( \mathbf{\dot{q}}_{d,B}(t) = [\dot{x}_{d,B}(t) \ \dot{y}_{d,B}(t) \ \dot{\theta}_{d,B}(t)]^T \) be the velocity of the displaced virtual reference frame, according to the timing law, defined on the desired trajectory. The trajectory tracking controller, designed for the tracked vehicle, has the following linear form

\[
\begin{align*}
\dot{u}_1 &= \dot{x}_{d,B}(t) + k_1 (x_{d,B}(t) - x_B(t)) \\
\dot{u}_2 &= \dot{y}_{d,B}(t) + k_2 (y_{d,B}(t) - y_B(t))
\end{align*}
\]

(6.9)

Here \( k_1 > 0 \) and \( k_2 > 0 \) are the controller gains. The linear feedback controller makes the point \( B \) track any desired trajectory, even with discontinuous tangent to the path (e.g., a square without stopping at corners) without requiring the robot to stop and reorient itself at those points, as long as \( b > 0 \) [238].

Figure 6.4 depicts the convergence to zero of the Cartesian tracking error \( e_p(t) = [x_d(t) - x_R(t) \ y_d(t) - y_R(t)]^T \), with decoupled dynamics on its two components, induced by the linearization of the controller.

6.4 Trajectory planning and generation

Let \( \mathcal{C} = \mathbb{R}^3 \times SO(3) \) be the configuration space of the tracked vehicle. Let \( \mathcal{O} = \bigcup_{i=1}^p \mathcal{O}_i \subset \mathbb{R}^3 \) be the region occupied by the obstacles. Let \( \mathbf{q}_I \in \mathcal{C} - \mathcal{O} \) and \( \mathbf{q}_G \in \mathcal{C} - \mathcal{O} \) be the initial and the goal configuration of the robot, respectively. The trajectory planning problem can be broken down into the following two sub-problems:

- generating a collision-free path \( \tau : [0,1] \rightarrow \mathcal{C} - \mathcal{O} \), such that \( \tau(0) = \mathbf{q}_I \) and \( \tau(1) = \mathbf{q}_G \).
• defining a timing law \( s(t) \) on the path, such that \( s(t_i) = 0 \) and \( s(t_f) = 1 \), with \( \dot{s}(t) > 0 \), for \( t \in [t_i, t_f] \).

An approach based on Augmented Reality (AR) is proposed to address the problem of planning safe paths for the robot in the real environment \([271, 104, 48, 91]\). This approach enables the human operator to plan collision-free paths on the 3D map of the environment. For this purpose, the human operator is provided with an interface, visualizing the 3D map of the real environment, from the 3D laser range data, as well as the 3D model of the real robot in the correct pose, with respect to a global reference frame. The AR-based interface visualizes the motions of the robot overlaid on to the real environment, from different perspective views. The user interacts with a marker pen, changing its position on the 3D map, clicking on it or selecting an action event from a context menu assigned to it. The marker pen implements a function \( \hat{\tau}: [0, 1] \to \mathbb{R}^3 - O \), which generates a sequence of points, corresponding to the points of the trajectory curve. An action event in the context menu allows the operator to determine the starting point \( \hat{\tau}(0) = \hat{q}_I \) of the trajectory curve. The marker pen is tracked by the interface which, in parallel, converts the movements of the human operator into the sequence of points \( \hat{\tau}(1) = \hat{q}_G \) of the trajectory curve the current position of the marker pen on the 3D map.

Upon the commitment of the path, generating the trajectory of the robot requires to specify, for each point \( \hat{q}_i \in \hat{\tau} \), the orientation \( R(\phi) \in SO(3) \) of the virtual reference frame, with respect to the global reference frame, having as origin \( \hat{q}_i \), and the velocity \( \hat{\dot{q}}_i \). Let \( v_{d} \in \mathbb{R} \) be a constant, specifying the velocity profile. Let \( f \in \mathbb{R} \) be the minimum rate of the trajectory tracking controller. Given the path \( \hat{\tau} \), the AR-based interface generates the trajectory \( \tau \) as well as the velocities \( \dot{\hat{q}}_i \), for all \( \hat{q}_i \in \hat{\tau} \), according to the following function:

\[
 f_{3D}: \mathbb{R}^3 \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}^3 \times SO(3) \times \mathbb{R}^3
\]  

(6.10)

At \( i^{th} \) iteration, the function computes for the pair of consecutive points \( \hat{q}_i, \hat{q}_{i+1} \in \hat{\tau} \), the Euclidean distance \( d_i = |\hat{q}_{i+1} - \hat{q}_i| \). This distance is compared with the desired traveled distance \( d_{i,d} = v_d \cdot \frac{1}{f} \). Whether \( d_{i,d} \leq d_i \) a new point \( \bar{q}_i \) is added, at distance \( d_{i,d} \) along the segment connecting \( \hat{q}_i \) to \( \hat{q}_{i+1} \), to the trajectory \( \tau \), and \( \hat{q}_i \) is updated to \( \bar{q}_i \). Otherwise,
Figure 6.5. Trajectory control schema

\( \hat{q}_i \) is not updated, a new pair of consecutive points on the path is considered, the desired traveled distance is updated to \( d_{i+1,d} = v_d \cdot \frac{1}{2} - d_i \) and the new point, in the \((i+1)^{th}\) iteration will be added, according the updated distance \( d_{i+1,d} \). Moreover, for each point \( q_i \in \tau \), the velocity \( \dot{q}_i = \begin{bmatrix} v_d \cdot \cos(\phi) & v_d \cdot \sin(\phi) & 0 \end{bmatrix}^T \), where \( \phi \) is the yaw angle defined by the vector \( q_{i+1} - q_i \). This function generates the 3D trajectory, with the effect of smoothing the path to make it more feasible for the robot to follow.

However, in order to apply the proposed control strategy a 2D representation of the trajectory \( \tau \) is required. As the tracked vehicle is always constrained to the surface which it is traversing, the 2D trajectory can be obtained from \( \tau \) by computing the following mapping

\[
f_{2D} : \mathbb{R}^3 \times SO(3) \times \mathbb{R}^3 \rightarrow \mathbb{R}^2 \times SO(2) \times \mathbb{R}^2
\]  

(6.11)

This function projects each way point \( q_i \in \tau \) on to the 2D reference frame constrained to the traversed surface. In addition, for every point \( q_i \), it computes the heading reference \( R(\phi_i) \in SO(2) \) as well as the x-component and the y-component of the velocity \( \dot{q}_i \). Algorithm 4 shows an implementation of the trajectory generator function in (6.10), integrating the mapping function in (6.11).

### 6.5 Experiments

The capability to describe the navigation task using the proposed path planning system and the effectiveness of the trajectory tracking control method are validated by the experiments settled in real scenarios, with the rescue rover in Fig. 2.4. Chapter 2. Figure 6.6(a) shows a 3D representation of the wheelchair ramp, connecting the basement with the ground floor of a building. Colours of the point clouds specify the static traversability cost assessment of the extracted terrain. According to the assumed stability criteria, the terrain point cloud coloured by red or yellow can be safety traversable, while, conversely, green or blue terrain point cloud do not guarantee the static stability of the robot. The wheelchair ramp is 50 m long, with 8% grade of slope. The human operator draws the path to be followed by the robot along the ramp, starting from the basement, up to the courtyard, at the ground floor (see the blue thick line in Figure 6.6(a)). The trajectory generator algorithm takes as input the path and the velocity profile and suitably samples a sequence of points, with the direction, representing the desired trajectory. Upon the trajectory has been generated,
Algorithm 4: 2DtrajectoryGenerator

Data: $\hat{\tau}, v_d, f$

Result: $\tau, \dot{\tau}$

begin

d_i := 0; d_{i,d} := 0; \Delta d := 0; \varphi := 0; q_i := \hat{\tau}(0); q_i^{new} := \hat{\tau}(0); changePoint := \bot;
i := 1;

while $i < \text{length} (\hat{\tau})$ do

if changePoint then

d_{i,d} := \left( v_d \cdot \frac{1}{\tau} \right) - \Delta d;
d_i := \text{getDistance}(\hat{\tau}(i), q_i^{new});
\varphi := \text{computeYawAngle}(q_i^{new}, \hat{\tau}(i));

else

d_{i,d} := v_d \cdot \frac{1}{\tau};
d_i := \text{getDistance}(\hat{\tau}(i), q_i);
\varphi := \text{computeYawAngle}(q_i, \hat{\tau}(i));
setYaw(q_i, \varphi);

end

if $d_i \geq d_{i,d}$ then

$q_{i,x}^{\text{temp}} := q_{i,x} + d_{i,d} \cdot \cos(\varphi);
q_{i,y}^{\text{temp}} := q_{i,y} + d_{i,d} \cdot \sin(\varphi);
q_{i,z}^{\text{temp}} := q_{i,z};

if changePoint then

\varphi := \text{computeYawAngle}(q_i, \hat{\tau}(i));
setYaw(q_i, \varphi);

end

$q_{i,x} := v_d \cdot \cos(\varphi); q_{i,y} := v_d \cdot \sin(\varphi);
add(\tau, q_i); add(\dot{\tau}, q_i);
q_{i,z} := 0; q_{i} := q_{i}^{\text{temp}}; changePoint := \bot;

else

\Delta d := \Delta d + |d_i - d_{i,d}|;
q_i^{new} := \hat{\tau}(i);
changePoint := \top; i := i + 1;

end

end

end
6.5 Experiments

Figure 6.6. Trajectories followed (blue thick lines) by the robot along a wheelchair ramp (a) and along a sloppy terrain (d), respectively. Colours in the point cloud represent the static traversability cost assessment of the extracted terrain. (b) and (c) Cartesian trajectory error of the controller in following the wheelchair ramp, subjected to two different velocity profiles $v_d$. (e) Variation of the norm $\|\mathbf{e}_d(t)\|$ of the Cartesian trajectory error of the controller in following the trajectory along the slopyy terrain. (f) Variation of the orientation error $\Delta e_\theta(t)$ of the controller tracking the path along the slopyy terrain.

the controller computes the input velocity commands of the robot to track the reference trajectory. Figure 7.9 and Figure 7.9 report the convergence of the Cartesian trajectory error $\mathbf{e}_d(t)$, during the navigation task, under two different settings of the velocity profile of the trajectory, $v_d = 0.1 \text{ m/s}$ and $v_d = 0.25 \text{ m/s}$, respectively. In both these experiments the gains $k_1$ and $k_2$ of the controller have been set to 0.5 and the distance $b$ of the point $B$ from the center of the robot to 0.2 m.

Figure 6.6(d) shows the 3D representation of an area, of 100 $m^2$, with rubbles, stairs, gaps and ramps. Supported by the path planning interface, the human operator identifies a possible route through the rubbles, and instructs the tracked vehicle to autonomously follow the path. The gains $k_1$ and $k_2$ of the controller have been set to 0.3, the velocity profile $v_d$ to 0.25 m/s and the distance $b$ of the point $B$ from the center of the robot to 0.2 m. In this experiment the skid-steering of the vehicle resulted in slippage between the tracks and the terrain as well as track-soil shearing, thus making difficult to predict the exact motion of the vehicle on the basis of track velocities. Therefore, the accumulated positioning error affected the estimation of the robot pose provided by the dead reckoning navigation system. Figure 6.6(e) and Figure 6.6(f) show how the linear and the angular velocity commands, given as inputs to the robot, affected both the variation of the norm $\|\mathbf{e}_d(t)\|$ of the Cartesian error and the variation of the orientation error $\mathbf{e}_\theta(t)$ of the controller tracking the path along the slopyy terrain. The error on the estimation of the real pose of the robot increases, as the input velocities increase.
Chapter 7

Real-Time 3D Motion Planning and Control for Autonomous Navigation

Tracked vehicles are designed for search and rescue applications where terrain conditions are difficult and unpredictable. They are better suited for such domains than wheeled vehicles due to the larger contact area of tracks with the ground, which provides better traction on harsh terrains. These robotic platforms are usually similar to space rovers with two tracks on the sides linked to a central body. Each track can be extended with two active flippers. Moreover, mechanical differential systems allow the rotation of the tracks around the body. These systems further increase the traction of such robots, thus improving their stability on sloppy surfaces. Several sensors can be installed, such as rotating 2D laser scanners for 3D point cloud acquisition, mapping and localization, vision systems, IMU and GPS for inertial navigation systems. Despite such robots are well-equipped to face all the navigation difficulties of an harsh environment, their level of autonomy is still not sufficient to operate without the supervision of a human operator. The main challenges of autonomous navigation are plan and effective motion of the tracked vehicle to safely traverse the rough surfaces of an unstructured environment, thus leaving flatlandia.

In this chapter we describe a real time 3D path and motion planner that tries to overcome some limitation of the current path planners.

The chapter is organized as follows. Paragraph 7.1 describes the trajectory control. Paragraph 7.2 describes our approach for estimating torque at the flippers links. Paragraph 7.3 describes the segmentation and clustering of the point cloud. Paragraph 7.4 describes the boundaries estimation of traversable surfaces and the estimation of a traversability graph leading to a path. Finally, Paragraph 8.7.5 describes the experiments proving the effectiveness of the proposed methods.

7.1 3D motion planning

In this paragraph we describe the main components of the 3D motion planning controller designed for the tracked vehicle in Figure 2.4 Chapter 2. The robot configuration state is defined by the vector $q = (x_r, y_r, z_r, \psi, \phi, \theta, v_r, \omega_r, \alpha_1, \ldots, \alpha_4)^T$, with $x_r, y_r, z_r, \psi, \phi, \theta$ the 6D pose of the robot, $v_r$ and $\omega_r$ the linear and angular robot velocities, respectively. Here, $\alpha_1, \ldots, \alpha_4$ are the configurations of the robot flipper. The state can be separated into two controllable parts, assuming that the control of the robot pose is independent of the flippers...
control. Under this assumption, the 3D motion planning controller can be divided into two decoupled control modules: (1) a trajectory tracking controller, and (2) a flippers position controller. These modules work in parallel and are synchronized so as to generate, at time stamp, the control commands needed to track a given 3D path and to simultaneously adapt the position of the flippers to the surfaces on which the path lies, namely to the planes tangent at each path point.

### 7.1.1 Trajectory tracking controller

The trajectory tracking controller receives as input a path $\mathcal{P}$ (see Paragraph 7.4), generated on the 3D labelled map of the environment (see Paragraph 7.3) and compute trajectory control along the path. Let the global reference frame be given, by localization, and let $q_d(t) = [x_d(t) \ y_d(t) \ \theta_d(t)]^T$ be the position and orientation of a virtual reference frame defined at tangent point $p \in \mathcal{P}$, with respect to a global reference frame. Similarly, let $q_r(t) = [x_r(t) \ y_r(t) \ \theta_r(t)]^T$ be the pose of the local robot reference frame expressed in the global reference frame. Under the assumption that the tracked vehicle can be approximated to a differential drive robot, modeled as a unicycle, the trajectory tracking controller has to generate the control inputs $c_d(t) = [v_d(t) \ \omega_d(t)]^T$ in order to asymptotically stabilize to zero the tracking error $e(t)$ between $q_d(t)$ and $q_r(t)$, which is defined as follows:

$$e(t) = \begin{bmatrix} \cos \theta_r(t) & \sin \theta_r(t) & 0 \\ -\sin \theta_r(t) & \cos \theta_r(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d(t) - x_r(t) \\ y_d(t) - y_r(t) \\ \theta_d(t) - \theta_r(t) \end{bmatrix} \quad (7.1)$$

The control strategy underlying the trajectory tracking controller is based on input-output linearization via feedback \([91]\).

Let $B = [x_B(t) \ y_B(t)]^T$ be a point displaced at distance $b > 0$ along the main axis of the tracked vehicle, whose Cartesian coordinates are

$$x_B(t) = x_r(t) + b \cos \theta_r(t)$$
$$y_B(t) = y_r(t) + b \sin \theta_r(t) \quad (7.2)$$

Given the kinematic model of the robot, the kinematics of the point $B$ on the tracked vehicle is given by

$$\begin{bmatrix} \dot{x}_B(t) \\ \dot{y}_B(t) \end{bmatrix} = \begin{bmatrix} \cos \theta_r(t) & -b \sin \theta_r(t) \\ \sin \theta_r(t) & b \cos \theta_r(t) \end{bmatrix} \begin{bmatrix} v_d(t) \\ \omega_d(t) \end{bmatrix} = T(\theta_r(t)) \begin{bmatrix} v_d(t) \\ \omega_d(t) \end{bmatrix} \quad (7.3)$$

Since $T(\theta_r(t))$ is nonsingular for any $b > 0$, the linearization of the control law can be obtained by computing the transformation of the control inputs as follows

$$\begin{bmatrix} v_d(t) \\ \omega_d(t) \end{bmatrix} = T^{-1}(\theta_r(t)) \begin{bmatrix} \xi_1 \\ \xi_2 \end{bmatrix} \quad (7.4)$$

Let $q_{d,B}(t) = [x_{d,B}(t) \ y_{d,B}(t) \ \theta_{d,B}(t)]^T$ be the pose of the virtual reference frame at $p \in \mathcal{P}$, displaced of the same value $b > 0$, along the trajectory tangent at $q_d(t)$.
7.1 3D motion planning

\( \dot{q}_{d,t}(t) = \begin{bmatrix} \dot{x}_{d,B}(t) & \dot{y}_{d,B}(t) & \dot{\theta}_{d,t}(t) \end{bmatrix}^T \) be the velocity of the displaced virtual frame, according to the timing law \( \sigma(t) \), defined on \( \mathcal{P} \). The trajectory tracking controller, designed for the tracked vehicle, has the following linear form

\[
\begin{align*}
\xi_1 &= \dot{x}_{d,B}(t) + k_1 (x_{d,B}(t) - x_B(t)) \\
\xi_2 &= \dot{y}_{d,B}(t) + k_2 (y_{d,B}(t) - y_B(t))
\end{align*}
\] (7.5)

Here \( k_1 > 0 \) and \( k_2 > 0 \) are the controller gains. Given the linear feedback controller in eq. (7.5), the point \( B \) can track any desired trajectory, even with discontinuous tangent to the path (e.g., a square without stopping at corners) without requiring the robot to stop and reorient itself at those points, as long as \( b > 0 \). The input-output linearization guarantees the convergence to zero of the Cartesian tracking error \( e_{p}(t) = [x_{d}(t) - x_r(t), y_{d}(t) - y_r(t)]^T \), with decoupled dynamics on its two components [135].

7.1.2 Flippers position controller

Each robot flipper is controlled in single-feedback position-control mode, by Elmo servo drive [51]. The flippers position controller receives the position commands \( \alpha_{com}(t) = (\alpha_1(t), \ldots, \alpha_4(t))^T \), as input, generating suitable internal speed commands \( \alpha_{com}(t) \) to asymptotically stabilize to zero the flippers position error \( e_{pos}(t) = [\alpha_{com}(t) - \alpha_{feedback}(t)] \). The speed commands are directly injected as torque (motor current) commands to the servo drive. Torque commands are accepted in the range allowed by the actual torque command limits, which can also be manually set, via software commands. Some notation is introduce to explain how the internal speed commands are computed. Let \( p_i(t) \) be the current robot position with respect to the global reference frame, provided by the localization system, and let \( \gamma \) defined as in the following table:

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>( -\pi/4 )</th>
<th>( 3/4 \pi )</th>
<th>(-3/4 \pi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi/4 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then \( p_1(t), p_2(t), p_3(t) \) and \( p_4(t) \) are the contact points of the flippers, with respect to a global reference frame:

\[
p_i(t) = R(\theta_r(t) + \gamma_i, \phi_r(t), \psi_r(t)) \lambda + p_i(t)
\]

Here \( \lambda = (\lambda, 0, 0)^T \) is the displacement vector, with \( \lambda \) obtained from the robot size. The points position changes as the robot pose changes, during the trajectory tracking task. The flippers position controller estimates the angles \( \hat{\alpha}_i(t) \) between the normals \( n_i(p_i(t)) \), of each \( p_i(t) \) (see Paragraph 7.3), with the \( z \) axis of the global reference frame. Upon the estimation of \( \hat{\alpha}_i(t) \), a preliminary test is performed, checking \( \hat{\alpha}_i(t) \) with respect to a predefined set of flippers configurations (see Table 7.1) according to a set of empirical rules defined as follows. Let us define two configurations:

\[
A = n(p_1(t)) \parallel \cdots \parallel n(p_4(t)) \quad \text{all flippers are parallel.}
\]
\[
B = n(p_1(t)) \parallel n(p_2(t)) \parallel n(p_3(t)) \parallel n(p_4(t)),
\]
\[
\text{\quad \quad \quad \quad \quad \text{\quad \quad \quad \quad \quad } n(p_1(t)) \parallel n(p_3(t)), \quad \text{flippers are pair-wise parallel.}
\]

Together with the following cases for the angle ranges:

\[
\begin{align*}
Q_{11} : & \quad \hat{\alpha}_i \in [\pi/4 - \varepsilon, \pi/4 + \varepsilon] \\
Q_{12} : & \quad \hat{\alpha}_i \in [3\pi/4 - \varepsilon, 3\pi/4 + \varepsilon] \\
Q_{13} : & \quad |\hat{\alpha}_i| \leq \varepsilon
\end{align*}
\] (7.7)
Table 7.1. Predefined flippers configurations

<table>
<thead>
<tr>
<th>Label</th>
<th>Configuration</th>
<th>$\alpha_1$</th>
<th>$\alpha_2$</th>
<th>$\alpha_3$</th>
<th>$\alpha_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_1$</td>
<td>FLAT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\alpha_2$</td>
<td>DRIVE</td>
<td>$-\frac{11}{12}\pi$</td>
<td>$-\frac{11}{12}\pi$</td>
<td>$\frac{\pi}{2}$</td>
<td>$\frac{\pi}{2}$</td>
</tr>
<tr>
<td>$\alpha_3$</td>
<td>CLIMB</td>
<td>$\frac{\pi}{3}$</td>
<td>$-\frac{\pi}{3}$</td>
<td>$\frac{\pi}{3}$</td>
<td>$\frac{\pi}{3}$</td>
</tr>
<tr>
<td>$\alpha_4$</td>
<td>GET-OFF</td>
<td>$-\frac{\pi}{18}$</td>
<td>$-\frac{\pi}{18}$</td>
<td>$\frac{\pi}{4}$</td>
<td>$\frac{\pi}{4}$</td>
</tr>
<tr>
<td>$\alpha_5$</td>
<td>CONVEX</td>
<td>$\frac{\pi}{6}$</td>
<td>$\frac{\pi}{6}$</td>
<td>$-\frac{\pi}{6}$</td>
<td>$-\frac{\pi}{6}$</td>
</tr>
</tbody>
</table>

Then the commands $\alpha_{\text{com}}(t)$ are computed together with the steering commands $c_d(t)$ as follows, $\epsilon > 0$ is the offset angle:

$$
\alpha_{\text{com}}(t) = \alpha_2 \leftarrow A \& Q_{i3}, \quad i=1,\ldots,4 \quad (7.8)
$$

$$
\alpha_{\text{com}}(t) = \alpha_1 \leftarrow A \& Q_{i1} \text{or} Q_{i2}, \quad i=1,\ldots,4 \quad (7.9)
$$

$$
\alpha_{\text{com}}(t) = \alpha_3 \leftarrow B \& Q_{12} \& Q_{22} \& Q_{33} \& Q_{43} \quad (7.10)
$$

$$
\alpha_{\text{com}}(t) = \alpha_4 \leftarrow B \& Q_{13} \& Q_{23} \& Q_{31} \& Q_{41} \quad (7.11)
$$

$$
\alpha_{\text{com}}(t) = \alpha_5 \leftarrow B \& (Q_{11} \& Q_{12}) \text{or}(Q_{32} \& Q_{42}) \quad (7.12)
$$

For the remaining cases not covered by the previous control rules, then $\alpha_{\text{com}}(t) = (\hat{\alpha}_1(t),\ldots,\hat{\alpha}_4(t))^T$. However, during the control loop of the position controller, two feedbacks can be directly measured: (1) the actual angles $\alpha_{\text{feedback}}(t)$ of the flippers (see Fig. 7.1(a)) and, (2) the electrical currents $i(t)$ measured on the flippers motors (see Fig. 7.1(b)).

Since the flippers are neither endowed with contact sensors nor with proximity sensors, it is quite hard to determine and model the contact between the flippers and the surfaces, just from the measurements of both $\alpha_{\text{feedback}}(t)$ and $i(t)$. To face this limits the contact sensors of the flippers have been statistically modeled, as described in the next Paragraph 7.2. The controller activates these sensors to both correct the estimation of the position commands and to ensure that the robot has a better traction on the harsh terrain.

### 7.2 Contact sensor model

In this paragraph we describe the model of the contact sensor of each flipper. This model is based on learning a function $h_{\theta^*}$ assessing the optimal position of the flipper.

#### 7.2.1 Data-set collection and labeling

Several experiments have been performed with the robot climbing ramps and stairs, surmounting obstacles, overcoming rubble piles with different shapes, and several measurements of both $\alpha_{\text{feedback}}(t)$ and $i(t)$ of the flipper have been taken, over time. The gathered data have been manually labeled with a label $l(t) = 1$ and $l(t) = 0$, denoting the touch and the detach of the flipper from the obstacles surfaces, respectively, thus enriching the feedback information provided by the position controller (see Fig. 7.1(c)). This data have
7.2 Contact sensor model

Figure 7.1. (a) trend of the position of the front right flippers (b) trend of the component of the current signal of the front right flippers, in amperes, producing the magnetic field in the desired direction (c) labeling of the measurements of both $\alpha_{feedback}(t)$ and $ii(t)$ from the position controller of the front right flipper.

been further interpolated to build a data-set $D = \{ (\alpha(t), ii(t), l(t)) : t = 0, \ldots, T \}$ suitable for the training of the contact sensor model.

7.2.2 Feature representation

The data-set $D = \{ (\alpha(t), ii(t), l(t)) : t = 0, \ldots, T \}$ has been processed to extract the relevant features for tuning the parameters of the function $h_{\theta}^*$ to be learned. Let $t \leq T$ be a sampled time point and let $W = [t', t]$ be a fixed sliding time window, with $t' < t$, the following features, inferred from $D$, can be considered as relevant features:

$x_1 = \frac{1}{t - t'} \sum_{k=t'}^{t} |ii(k)|$

$x_2 = \frac{1}{t - t'} \sum_{k=t'}^{t} \left| \frac{\alpha(k+1) - \alpha(k-1)}{2} \right|$

$x_3 = \text{sgn} \left( \sum_{k=t'}^{t} ii(k) \right) \cdot \text{sgn} \left( \sum_{k=t'}^{t} \frac{\alpha(k+1) - \alpha(k-1)}{2} \right)$

$x_4 = \frac{1}{t - t'} \sum_{k=t'}^{t} |ii_{TDF-II}(k)|$

Here $ii_{TDF-II}(t)$ is the component $ii(t)$ of the current signal, filtered according to the Transposed-Direct-Form-II (TDF-II) digital filter. The filter has been applied to reduce the oscillations of $ii(t)$ during the transient conditions of the servo drive. The label associated with a feature vector $x = (x_1, \ldots, x_4)^T$ is computed as follows:

$y = \begin{cases} 
1 & \text{if } \frac{1}{t-t'} \sum_{k=t'}^{t} l(k) > 0.5 \\
-1 & \text{otherwise}
\end{cases}$

Let $N$ be the number of sampled time points, then $D' = \{ (x_i, y_i) \}_{i=1}^{N}$ is the data-set of features-labels pairs generated from $D$, with $x_i \in X \subseteq \mathbb{R}^4$ and $y_i \in L \equiv \{-1, 1\}$.

7.2.3 Learning the parameters of $h_{\theta}^*$

Let $\mathcal{H} = \{ h_{\theta}(x) : \theta \in \Theta \}$ be the space of the decision functions, such that $h_{\theta} : X \rightarrow L$, $\Theta$ the parameter space. Given the set $D' = \{ (x_i, y_i) \}_{i=1}^{N}$ of labeled features samples, assumed
to be drawn from an unknown distribution $P(x, y)$, we want to find a function $h_{\theta^*}$ which gives the lowest upper bound of the expected risk:

$$R(\theta) = \int |h_{\theta}(x) - y| P(x, y) dx dy$$

(7.13)

The lowest upper bound of such a loss function can be found by applying a non-linear classifier based on the Support Vector Machines (SVM). Therefore the problem of estimating the function $h_{\theta^*}$ can be considered as the problem of finding the decision surface that better separates the data. This can be formulated as follows:

$$\max_{\theta \in \Theta} \sum_{i=1}^{N} \theta_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \theta_i \theta_j K(x_i, x_j)$$

subject to $\sum_{i=1}^{N} y_i \theta_i = 0, 0 \leq \theta_i \leq 1, \forall i = 1, \ldots, N$.

Here $K(x_i, x_j) = (x_i^T x_j + 1)^d$. The polynomial kernel has been introduced due to the non-linear separability of the data-set $D'$. Given a new instance of a feature vector $x$, the decision function $h_{\theta^*}$ modeling the contact of the flipper with the underlying surface classifies $x$ as follows

$$h_{\theta^*}(x) = \text{sgn} \left( \sum_{i=1}^{N_S} y_i \theta_i^* K(s_i, x) \right)$$

(7.14)

Here $s_i$ are the support vectors and $\theta_i^*$ are the optimal parameter values associated to $s_i$, with $i = 1, \ldots, N_S$. A cross-validation technique has been used to evaluate the performance of the contact sensor model as both the degree $d$ of the polynomial kernel and the sliding time window $W$ change. Figure 7.2 shows that for $d = 13$ and $W = 5$ the model correctly classifies 90.7% of the contacts of the flipper with the underlying surface.
7.3 Point cloud segmentation and labeling

We describe a real time segmentation and labelling of a point cloud $PCL$ registered by an ICP-based SLAM [211]. The point cloud $PCL$ stores the geometric position $(x_i, y_i, z_i)$ of each point $p_i \in PCL$, approximating a surface manifold $M$, our goal is to establish traversability of the manifold. As the computation serves the path control, the segmentation and labelling of the surface has to be real-time. To this end we do not need to take into account all the points of $PCL$, but only those that fall into interesting categories. More specifically, we define four categories: ground, walls, ramp or stairs, surmountable obstacle. These have been devised to match the effective robot overcoming abilities: 30 cm steps with rounded edge; 40 cm gap; 20 cm stairs at 40° slope; spiral staircases; 45° slope / 15° roll; 80cm × 80cm cavities. These categories are captured by three main features, the surface normals, the mean curvature and points location.

The approach proposed for segmentation and labelling is specified by the following steps (1) point cloud filtering; (2) estimation of normals to the surface and curvature and (3) cluster and merging of the filtered $PCL$ and labelling of the resulting clusters. Note here that we do not consider the time argument to points, as in Paragraph 7.1.

Filtering Let us consider for each point $p \in PCL$ a neighborhood $\mathcal{N}(p) = \{y \in \mathbb{R}^3; \|p-y\| < r\}$, with $\|\cdot\|$ the Euclidean norm and $r$ increasing according to the neighborhood density $\rho = |\mathcal{N}|/\rho^*$, here $|\cdot|$ is the cardinality. An optimal density $\rho^*$ is specified by a radius of $r_{\text{min}} = 0.1\sqrt{2}$ meters and a neighborhood of 15 points. If $\rho < \rho^*/2$ then $r$ is increased to catch more points. To control the radius growth and filter out noise, a noise function is specified according to the density as follows. Let $f(p) = \exp(-\pi(r_{\text{min}} - r))$, then the probability that a neighborhood $\mathcal{N}(p)$ is affected by noise is:

$$p(\text{noise}(p)) = 1 - f(p)$$ (7.15)

The noise function is zero when $r = r_{\text{min}}$ and it is around 0.5 when $r$ is about 0.3, over this value the noise increases rapidly. All points with noise value greater than 0.5 are removed from $PCL$.

Normal estimation The problem of determining the normal to a local surface patch, approximated by $PCL$, at a point $p$ is equivalent to the problem of estimating the tangent plane $T_p S$ to $S$ at $p$, and it can be reduced to a least-square plane fitting problem. Let $p \in \mathbb{R}^3$ and $\mathcal{N}_{p^\circ}$ its neighborhood. The normal is found by minimizing the error term $\text{err}(n,d) = \sum_{n=1}^{N} (n^\top x_n - d)^2$, with $n^\top n = 1$. In [195] Pauly noticed that neighbors closer to $p$ should be weighted more than distant ones and introduced a Gaussian kernel as weighting function. Accordingly we define an orthogonal transformation weighted by a kernel function. Given a point $p$ and $N$ sampled neighbors, the sample covariance is defined as:

$$K = \frac{1}{N} \sum_{n=1}^{N} k(p_n, \bar{p}) \cdot (p_n - \bar{p})(p_n - \bar{p})^T$$ (7.16)

Here $p_n$ is a neighbor of $p$, with $\|p_n - p\| < r$, $\bar{p}$ is the center of mass of $\mathcal{N}(p)$, and $k(\cdot,\cdot)$ is the Gaussian kernel function (GKF):

$$k(x_n, \bar{x}) = \exp\left(-\frac{\|x_n - \bar{x}\|^2}{\xi}\right)$$ (7.17)
Here $\zeta$ is a smoothing factor, affecting the kernel resolution and it depends on the radius $r$. Clearly $K$ is symmetric, $K = V\Lambda V^T$ and $Kv_i = \lambda_i v_i$, with $v_i$ the $i$-th unit eigenvector and $\lambda_i$ the $i$-th eigenvalue of $K$. Since $K$ is positive definite, let $\lambda_1 > \lambda_2 > \lambda_3$, then $v_i$, $j \in \{1, \ldots, 3\}$ is a basis for the tangent plane, and we choose $n(p) = v_1 \times v_2$, with $v_1 \times v_2 \neq 0$. ($v_i$ unit vectors). Hoppe et al. [120] note that the sign should vary according to the constraint that geometrically close fitted planes have the same orientation, namely $n_i \cdot n_j \approx 1$, which can be reduced to a graph cut problem, which is NP-hard [120]. In the present framework the problem is simplified by the fact that the orientation of the tangent plane $T_pM$ is subject to the robot point of view, hence it is chosen to be in the range of the robot orientation, see Paragraph 8.7.5 on the experiments. The curvature at $p$ is given by the curves on the plane spanned by $n(p)$ and a vector $v$ lying on the tangent plane. In particular, if we consider the two principal directions given by the unit eigenvectors $v_1$ and $v_2$, then letting $S_p(v_1) = -\nabla_{v_1} n \cdot v_1$ and $S_p(v_2) = -\nabla_{v_2} n \cdot v_2$, with $\nabla_{v_i}$ the directional derivative in the direction of $v_i$, then we obtain the two principal curvatures $\pm \kappa_1$ and $\pm \kappa_2$ as the eigenvalues of $S_p(v_j)$, $j = 1, 2$ such that $\kappa = \frac{1}{2}(\kappa_1 + \kappa_2)$ is the mean curvature. For the discrete directional derivatives and the curvature computation, we exploit the simple meshing described in Paragraph 7.4 together with the method described in [132].

**Segmentation** Given normals $n(p)$ and curvatures $\kappa_1(p), \kappa_2(p)$, points are collected into clusters according to an initial condition and boundary, i.e. stop condition. The initial condition is given by the minimal curvature $\kappa_1 \kappa_2$ and assigned kinetic energy $E_k$; while the stop condition is given by a function $g(p)$ which stops the front collecting points, at a given direction of growth, whenever there is a jump in energy level or the curvature is getting higher. The energy level is $E_k(p) = \arccos(n(p)z)$, with $z = (0, 0, 1)$. Let:

$$
\varphi(p_0, p) = s(|\kappa(p) - \kappa(p_0)|) + s(|E_k(p_0) - E_k(p)|) \tag{7.18}
$$

Here $|\cdot|$ is absolute value, $s$ is the logistic function $y = c/(1 + a \exp(-bx))$, with $0 \leq x \leq \pi$, $b = 25$ and $a = \pi^2$. We have chosen for the limiting upper bound $c = 100$, these values ensures that when the difference in energy level/curvature increases beyond a threshold 0.08 then the function fires high values up to $c$. Then $d_{\text{trans}}(p_0, p)$ is the length of the longest path between $p_0$ and point $p$, in any direction, such that $\varphi(p_0, p') \leq 0.16$, for each point $p'$ in the path. Then the kinetic energy of the point $p_0$ is $E_k(p_0) = d_{\text{trans}}(p_0, p)$. Let

$$
g(p) = E_k(p_0) - \varphi(p_0, p) \tag{7.19}
$$

Points $p$, around $p_0$, are collected as far as $g(p) > 0$: Therefore when $g \leq 0$ then the front expansion stops for that point. To resume:

1. **Input**: the PCL, the normals, the curvatures.
2. For each neighborhood with minimal curvature, compute the length of the path to a farthest point such that for every point in the path the difference in energy level and curvature is low, namely $\varphi(p_0, p) \leq 0.1$.
3. Assign kinetic energy $E_k = d_{\text{trans}}(p_0, p)$ to the chosen points $p_0$ with minimal curvature.
4. Move the neighborhood front in all directions, checking for each direction the stopping criterion $g(p)$ (eq. 7.19).
5. For each $p_0$, the points it collected form a cluster $C$.

6. Empty the PCL from the points collected into the formed clusters, if PCL is empty stop, else go to item 2).

7. Output: the list of clusters.

Note that we choose a furthest point together with the path, and define the kinetic energy in place of a cost, confront with [230].

**Merging and Class labelling** Classes of interest are obtained by merging the clusters according to the energy levels and the $z$ of the collected points. This is obtained by discretizing the energy levels into 5 classes and by defining height thresholds as follows. Let $C$ be a cluster of points collected as described in the previous paragraph. Its energy level is:

$$
\hat{E}_i(C) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{5} E_{\ell_j}(p_i) \delta_{j, \ell_j} \quad \forall p_i \in C
$$

(7.20)

Here $N$ is the cardinality of the cluster and the levels are $\ell_1 = (1/4\pi, 3/4\pi), \ell_2 = (1/16\pi, 1/4\pi), \ell_3 = (7/8\pi, \pi), \ell_4 = (0, 1/16\pi), \ell_5 = (1/8\pi, \pi)$, and $\delta_{j, \ell_j} = 1$ if $E_{\ell_j}(p) \in \ell_j$ and 0 otherwise. The levels are thought for the defined classes, namely *ground, ramp or stairs, walls and surmountable obstacle*, and to comply with the robot overcoming abilities. The height thresholds are defined as follows. Let $z_{\text{max}} = \arg \max \limits_{p}(pz)$ and $z_{\text{min}} = \arg \min \limits_{p}(pz)$ and let $\text{height} \approx 0.3$. Then $H_1 = ||z_{\text{max}} - z_{\text{min}}|| \geq \text{height}, H_2 = ||z_{\text{max}} - z_{\text{min}}|| < \text{height}$. Then the 4 classes are defined as follows:

<table>
<thead>
<tr>
<th>Classes labels</th>
<th>wall</th>
<th>surmountable obstacle</th>
<th>stairs or ramp</th>
<th>ground</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_i(C)$</td>
<td>$\ell_1$</td>
<td>$\ell_1$</td>
<td>$\ell_2$ or $\ell_3$</td>
<td>$\ell_4$ or $\ell_5$</td>
</tr>
<tr>
<td>height</td>
<td>$H_1$</td>
<td></td>
<td></td>
<td>$H_2$</td>
</tr>
</tbody>
</table>

Then clusters $C_i$ and $C_j$ are labeled according to the above specified classes and merged if they belong to the same class as specified in the above table.

### 7.4 Feasible Path

In this paragraph we describe how a feasible path is obtained, beyond the local steps computed by the flippers control. The proposed method requires three steps: (1) first the
region formed by all boundaries, from the labeled clusters, computed as illustrated in the previous paragraph are obtained; (2) then a graph $G$ connecting points is defined and, finally, (3) the graph is suitably weighted to return the traversability graph. These steps are illustrated in the following.

**Boundary region** Let $\mathcal{A}$ be the set of clusters, labeled with the class they belong to. Let $\mathcal{A}' = \{(C_k, c_k) : c_k = \text{wall}\}$ and $\mathcal{A}^* = \mathcal{A} \setminus \mathcal{A}'$. We introduce the points belonging to boundary regions $\mathcal{B}$, which are used to specify a feasible path, beside the flippers control (described in Paragraph 7.1). Let $\nu_{xy}$ and $\nu_z$ be the projection functions of a point returning respectively the $x$, $y$ and $z$ coordinate, and let $u: \text{pcl} \mapsto \text{pcl}$ be the function:

$$u(p) \triangleq \arg\min_{p_j \in C_j} \{ \|p - p_j\| \} \quad \text{s.t.} \quad (C_j, c_j) \in \mathcal{A}^* \tag{7.21}$$

Then the set of boundary points $\mathcal{B}$ is defined as follows:

$$\mathcal{B} = \{ p : p = (\nu_{xy}(p'), \nu_z(p''))^T, \exists (C, c) \in \mathcal{A}', \ p' \in C, p'' = u(p') \} \tag{7.22}$$

We can note that the estimation of the boundary regions preserves a 3D discrete representation.

**Connectivity and traversability path** Let $G(\mathcal{N}, \mathcal{E})$ be the graph whose nodes are all the points in $\mathcal{A}^*$ and whose edges $E \in \mathcal{E}$ are defined as follow:

$$E(p_i, p_j) = \begin{cases} \text{true} & \text{if } \mu(\Delta p) \leq \eta \\ \text{false} & \text{otherwise} \end{cases} \tag{7.23}$$

Here $\mu(\Delta p) = \|\Delta p^T, (1, 1, \delta)^T\|$, $\eta$ is set equal to half the robot length and $\delta = \frac{\eta}{d_{max}}$ is set according to both the robot morphology and the robot overcoming capabilities. The graph $G$ is further weighted to build the traversability graph $wG$, where the weights are defined as follows. Let $\rho$ be the density of a point $p$ neighborhood, as specified in Paragraph 7.3 (filtering), let $u$ be the function specified in equation (7.21) above, and let $r_{infl}$ be the inflated radius, a parameter that can be freely set, then the weights labeling an edge $E$ are defined as follows:

$$w = w_{\text{length}} + w_{\text{boundary}} + w_{\text{density}} \tag{7.24}$$

Where, given $E(p_i, p_j), p_i, p_j \in \mathcal{N}$:

- $w_{\text{length}} = \|p_j - p_i\|
- w_{\text{boundary}} = \begin{cases} \frac{1}{u(E)} & \text{if } u(E) < r_{infl} \\ \infty & \text{otherwise} \end{cases}$
- $w_{\text{density}} = \frac{2}{\rho(p_i) + \rho(p_j)}$

The weighted graph $wG$ is the thought for traversability graph. Then, given a goal specifying a point on the labeled map, a feasible 3D path $\mathcal{P}$ can be computed by a generic optimization algorithm such as Dijkstra [61], A* [114] or D* lite [139]. Here, in our experiments, we have been using the Dijkstra algorithm.
7.5 Experiments

In this paragraph we present the experiments performed to evaluate the accuracy of the segmentation of the 3D map of the environment and the feasibility of the path generated on the 3D labeled map. The segmentation has been evaluated with respect to the time required to both segment and label the point cloud, and with respect to its accuracy. The segmentation accuracy has been measured by manually assigning a label to each cluster and by comparing these labels with the labels returned by the segmentation process. The percentage of the clusters correctly classified states the accuracy of the segmentation. The feasibility of the 3D path is measured in terms of the time required to build the graph structure representing the traversable areas of the environment, with respect to the value of the path smoothing algorithm, and in terms of the time needed to the robot to complete the path, namely the journey time. This time has been taken by hand with a stopwatch. To test the effectiveness of the autonomy capabilities of the overall system three different scenarios has been considered.

The Italian Fire Fighters rescue training area in Prato (IT) First experiment has been performed at the Italian Fire Fighters rescue training area in Prato (IT), during the final review meeting of the EU project NIFTi (247870). In this experiment the robot traversed the harsh terrain of the rescue area, though not climbing any ramp or stairs, overcoming small obstacles, following different paths toward several target poses, manually posted by an operator on the 3D map. Figure 7.5 illustrates the segmented map with the inflated region in red, the generated path toward the goal (the gray cube) and screen shots of the robot following the path. The dialog window, is related to automatic goal generation which is not treated here. Table 7.2 reports the accuracy of the segmentation of the 3D map of the environment and the feasibility of the path generated on the 3D labeled map, values are averaged over eleven trials towards the same goal position. On average the map had about 52 thousand points.

Fire Escape stairs Second experiment has been performed on the fire escape stairs of the Department. The goal is located on the landing at the end of the second stairway, when the robot is up the goal is moved to the ground. The robot has simply to climb up to the goal and then turning on its self and coming back to reach the new selected goal. Because there are some localization problems, while turning around itself, in some trials the robot is not able to re-descend autonomously. Figure 7.3 show the climbing steps on the graph in which inflated regions are in red. Figure 7.4 shows the segmentation of the courtyard.

Figure 7.4. On the left point cloud segmentation of the fire escape stairs, with path drawn in magenta, and detail of the robot climbing the stairs (from Fire Escape stairs experiment). On the right and composed path of the robot from the gallery up to the top of the ramp and the bridge (from the Full 3D experiment).
where the fire escape stairs are located and, superimposed a detail of the robot climbing the stairs. Table [7.3] reports, for three trials, the average segmentation time, the percentage of the clusters correctly classified, the average time needed to generate the 3D path on the stairs, with respect to the value of the path smoothing and the journey time. On average the map has 41 thousand points.

**Full 3D designed scenario.** The third experiment has been designed purposefully to test the effective 3D autonomy, within an environment whose structure is composed of multiple levels. A gallery, surmounted by a ramp, extended with a bridge, is built and it lies between a step on the floor and a wall of bricks (see Figure 7.6). The robot has to first traverse the gallery and then climb the ramp passing over the gallery and continue up to the end of the bridge. The goal is located at the end of the bridge. The robot is constrained to pass under the gallery by obstacles. It is interesting to note that in this experiment the space is fully 3D since the robot has to face both the levels: under and over the construction (see Fig. 7.7). Table [7.4] reports, for three trials, the accuracy of the segmentation of the scenario as well as the feasibility of the complex 3D path. On average the map has 15 thousand points.

**Computational time performance** In addition to the previous experiments, several trials has been performed to evaluate the computational performance of both the segmentation and the path planning algorithm with respect to the size of the point cloud. In these trials the robot was teleoperated in a wide area to acquire, at real-time, the point cloud. The goal was

<table>
<thead>
<tr>
<th></th>
<th>Segmentation</th>
<th>Graph</th>
<th>Smoothing</th>
<th>Journey</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point Cloud</strong></td>
<td>0.27</td>
<td>0.86</td>
<td>1.19</td>
<td></td>
</tr>
<tr>
<td><strong>Path</strong></td>
<td></td>
<td></td>
<td>0.2</td>
<td>357</td>
</tr>
</tbody>
</table>

Table 7.2. Prato Experiments

<table>
<thead>
<tr>
<th></th>
<th>Segmentation</th>
<th>Graph</th>
<th>Smoothing</th>
<th>Journey</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point Cloud</strong></td>
<td>0.32</td>
<td>0.91</td>
<td>1.45</td>
<td></td>
</tr>
<tr>
<td><strong>Path</strong></td>
<td></td>
<td></td>
<td>0.2</td>
<td>210</td>
</tr>
</tbody>
</table>

Table 7.3. Fire escape stairs experiment
Figure 7.6. On the left segmentation of the third experiment map. On the right a screen shot of the generated graph.

Figure 7.7. Full 3D experiment: the two images show the robot passing under the gallery and climbing the ramp autonomously. The point cloud is segmented and the path is drawn. It is interesting to note that the robot passes under the gallery still following its path, up to the goal.

<table>
<thead>
<tr>
<th></th>
<th>Segmentation</th>
<th>Graph</th>
<th>Smoothing</th>
<th>Journey</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time (s)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Point Cloud</strong></td>
<td>0.12</td>
<td>0.57</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td><strong>Path</strong></td>
<td></td>
<td></td>
<td>0.2</td>
<td>143</td>
</tr>
</tbody>
</table>

Table 7.4. 3D designed experiment
fixed and the robot computed a new path every new point cloud. The results, in terms of time of computation, of the different algorithms are reported in Fig. 7.8. Note that the robot was able to elaborate a point cloud composed by 55000 points and to generate a path in less than 3.5 s.

**Control performance** Several experiments have been also performed to evaluate the performance of the robot control. The performance of the control can be measured by considering the average of the tracking errors of the controller, during the execution of the paths generated on the 3D map. Several paths have been computed on the 3D labelled map of both the fire stairs of the Department and the rescue training area at Prato. The trajectory tracking controller computed the input velocity commands of the robot to track the reference trajectory along these paths. Figure 7.9 reports the convergence of the average of the modules $\|e(t)\|$ of the tracking errors of the controller with respect to the 3D paths on the stairs, under a constant velocity profile $\sigma(t) = 0.1 m/s$. Figure 7.9 reports the convergence of the average of $\|e(t)\|$ with respect to the 3D paths on the harsh terrain at Prato, under $\sigma(t) = 0.15 m/s$. In both these experiments the gains $k_1$ and $k_2$ of the controller have been set to 0.3 and the distance $b$ of the point $B$ from the center of the robot to 0.2 m.
Chapter 8

Learning the dynamic processes of shifting and inhibition in cognitive control

A robot interacting with people and with the environment is subject to several stimuli events, like any living organism. Robot stimuli representation and stimulus-response modeling have not yet received the necessary attention, though.

Everyone knows what a stimulus is in the human and animal sense. The concept of the stimulus has been studied by and large since the eighteenth century, with the work of Galvani and Volta on frogs, and further in psychology, physiology and psychophysics. Modern physiology considers a stimulus applied to a receptor and capturing information from both inside and outside the body, which psychologists define as proximal and distal stimuli, respectively. On the other hand following the pioneering work of Weber [69], Gustav Fechner has introduced psycho-physics to model the relation of physical stimuli to the resulting mental facts. On collecting several quotation about “what a stimulus is”, Gibson [99] proposes a constructive view:

I think the central question is the following. Is a stimulus that which does activate a sense organ or that which can activate a sense organ? Some writers imply that a stimulus not currently exciting receptors is not a stimulus at all. Others imply that a stimulus need not excite receptors to be called such [99].

Gibson’s implicit question is about the possibility to give a structure, a definition, of the relationship between objects and stimuli and between stimulus and response:

Perhaps there is an invariant stimulus for the invariant response, after all. Many sorts of higher order variables of energy may exist, only awaiting mathematical description. They will have to be described in appropriate terms, of course, not as simple functions of frequency and amount. We must not confuse a stimulus with the elements used for its analysis. [99]

As a matter of fact we do not have a model of robot stimuli and still most researchers accept that any sensory input from the environment, or from the internal states, can be considered a stimulus. This simplification is of no help, as the wide literature on human and animal stimuli witnesses. In fact, both in psychology and psychophysics the studies on stimuli lead
to the stimulus-response model, as Gibson writes *does a stimulus motivate the individual or does it merely trigger a response?*.

In this chapter we address the problem of providing a representation of what a stimulus is for a robot. To get an idea consider that while for a living organism pain is a stimulus that has a direct effect on his/her survival awareness, for a robot analogous stimuli could be the lack of WiFi connection or the battery exhaustion, both indicating a loss of vitality. Therefore, in providing such a representation the main problem is: how do we build the stimulus reflex within the robot structure and language, and how could the robot learn what the best response is according to the context.

We can note that where in living organisms there is a stimulus transduction eliciting a reflex, through a receptor, in a robot organism the stimulus transduction is operated by a process. Consider the robot motion system components, each of which is driven by processes controlling velocity, stir, obstacle avoidance, terrain adaptation, tracking and so on. Here the stimulus transduction is the *yield of a process*. In fact, the processes are the structure driving the information both from the environment and from the internal states. Tasks and processes define the context. Just consider that an animal elicits different responses to stimuli whether it is hunting or escaping or resting.

Therefore, together with the stimulus model, we study the response model under a learning perspective (an early work on this is [17]). The stimulus-response is learned on the basis of several operators trials scoring the response to each potential stimulus, according to the circumstances and the contexts impacting the stimulus stir. The model develops on the basis of the selection matrix introduced in recommendation systems.

To capture the context and hence to be able to establish a response cost hinging upon the interferences with other stimuli, current task, and previous and further duties, we define the action structure governing the robot activities. We do so specifying the underlying theory of actions that models how tasks are chosen and how a switch to a new task can occur as a result of a stimulus-response.

However, another important problem to be resolved, is the switching decision which cannot rely solely on the information obtained by the robot’s own actions. Experts of task switching claim that this decision undergoes a *switching cost*. This cost results from the interplay between the resources needed to reconfigure a mental state and the resources needed to resolve interference with the current state [174]. We model this cost considering both the reconfiguration of the mental state, and the interference with the current state. The reconfiguration and interference costs are defined in terms of the amount of processes that need to be inferred in order to initiate a new task, and their congruency with the current task.

To summarize our approach, in this chapter we aim to provide a preliminary stimulus-response model for robot cognitive control. This model leads to the decision of shifting to a new task or inhibiting the train of stimuli on the basis of a cost taking into account two efforts, the effort for reconfiguring the current state and the effort to resolve the interference with the current state and processes [174]. The stimulus-response model illustrated in these pages indicates a new direction in robot cognitive control exploiting several methodologies from different research areas. We treat action theories to model robot processes and actions, in a logical formalism; we treat learning, classification, regression and surface fitting, for modeling stimuli, the stimulus-response problem and the decision on switching and inhibition.

The chapter is organized as follows. In the next paragraph we provide a brief description of the schema of the workflow of the proposed mechanism of task switching. In Paragraph
we introduce the representation of the stimuli. In Paragraph 8.3 we describe the stimulus-response model. Further, in Paragraph 8.4 we describe the fundamental formalism defining concepts such as tasks and processes which are needed to model the stimuli transduction, via the information a process yields; they are also needed to define the cost, which is influenced by the task currently executed and the new task that should be initiated as a response to the stimulus. In Paragraph 8.5 we illustrate the method for choosing the minimal cost best response. The chapter concluded with experiments conducted to evaluate the performance of the proposed model of task switching and to effectively assess the correctness of the behavior of the robot, in the presence of different stimuli, under the proposed paradigm of shifting and inhibition.

### 8.1 Executive Summary

In this Paragraph we describe the workflow of the proposed mechanism of task switching, through which the cognitive control well regulates the behavior of the robot (see Fig. 8.1). The execution of a task activates a subset of the hardware and software components of the robot. Each of these components runs a subset of processes which will be active during the entire life-cycle of the task. The dynamic model of the robot, specified in a first-order logic-based language \( L \), is responsible of the formal verification of the preconditions of the execution of the robot actions as well as of the checking of the consistency of the constraints regulating the interaction among the running processes (Paragraph 8.4). The running processes carry information about both the internal state of the robot and the environment, within which the robot is operating, namely the yields of the processes. These yields can be represented as features vectors \( z_1, \ldots, z_n \), with \( z_i \in \mathbb{R}^d \). The stimuli model takes as input these features and identifies whether a feature vector \( z \) is a stimulus or not, on the basis of the value returned by a learned function \( h(\beta)(z) \) (Paragraph 8.2). Upon the identification of the stimulus, both the value of the feature vector \( z \) and the value returned by \( h(\beta)(z) \) have to be associated with the corresponding process, in the robot dynamic model, which has triggered the stimulus. A special functional \( \& \) binds, at execution time, the process terms of the language \( \mathcal{L} \), with the corresponding values of both the yields and the stimuli, detected from yields. The function \( \& \) effectively interconnects the model of the stimuli with the dynamic model of the robot (Paragraph 8.4). The stimulus-response model takes as input the triggered stimuli and maps, on the basis of a learned function \( \gamma \), such stimuli into mental states, thus returning those subset of the set of mental states, which are the states that are more likely to be effective to respond to the stimuli. Each of these responses represents the set of processes needed to be activated to initiate the corresponding task, specified into the task library of the robot (Paragraph 8.3). The decision to shift from the current task to a new task, in so responding to the stimulus, or to inhibit the stimulus, depends upon the following two costs: (1) the cost to reconfigure the current mental state of the robot to the mental state, associated with a new task and, (2) the cost required to resolve the interference between the current mental state and the new mental state (Paragraph 8.4). The rule-based task selection model evaluates the costs of reconfiguration, the costs of interference of each new mental state, the urgency of the stimuli, and determines the final decision to either shift to a new mental state, namely a new task, or inhibit the stimuli, thus preserving focus on current task (Paragraph 8.5).
8. Learning the dynamic processes of shifting and inhibition in cognitive control

Figure 8.1. Workflow of the proposed mechanism of task switching, through which the cognitive control well regulates the behavior of the robot

8.2 The stimuli model

In this paragraph we face the following problems:

1. What a stimulus is for a robot.

2. How a stimulus transduction is applied by an active process.

3. How potential stimuli information is transformed and brought to the robot mental state.

To begin with, let us consider the following events applied to a robot: the battery is getting low, the WiFi has very weak or no signal, saliency is detected in a sequence of frames, one track does not respond to the control command, the memory is surcharged, the communication with the operator is clogged, a request pops-up on the interface, someone is talking at the dialog box. There are hundreds of events like these ones a robot has to deal with. These events are stimuli and are applied to a specific robot component, such for the example: the battery, the WiFi, the attention system, the mapping system, the engines, the dialog system, the interface, and the obstacle avoidance system. Three aspects are at the core of stimuli modeling: 1) the information carried by a robot process, applied to a robot component; 2) the objects and events that may induce a stimulus within the processed information, and 3) the rule by which a stimulus is detected to be so:

it is reasonable to assume that stimuli carry information about the terrestrial environment. That is, they specify things about objects, places, events, animals, people, and the actions of people. The rules by which they do so are to be determined, but there is at least enough evidence to warrant discarding the
Table 8.1. Examples of stimuli features

<table>
<thead>
<tr>
<th>Organ</th>
<th>Features</th>
<th>Stimulus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battery</td>
<td>Level, Power Consumption, Time, Location</td>
<td>Exhaustion</td>
</tr>
<tr>
<td>WiFi</td>
<td>Strength, Dist. Source, Speed, Traffic</td>
<td>Disconnection</td>
</tr>
<tr>
<td>Laser scan</td>
<td>Resolution, Surface, Density, Smoothness</td>
<td>Lack of data, Obstacle</td>
</tr>
<tr>
<td>Mapping</td>
<td>Height, Slope, Intricacy, Depth</td>
<td>Terrain roughness, hostile scene</td>
</tr>
<tr>
<td>Visual perception</td>
<td>Shape, Dimension, Location, Distance</td>
<td>Identification, Detection</td>
</tr>
<tr>
<td>Attention</td>
<td>Light, Color, Motion, Contour</td>
<td>Saliency</td>
</tr>
<tr>
<td>Interface</td>
<td>Sound, Id Voice, Utterance, Meaning</td>
<td>Request</td>
</tr>
<tr>
<td>Sound</td>
<td>Scale, Frequency, Pitch, Time</td>
<td>Who, mute, noise</td>
</tr>
</tbody>
</table>

We shall first provide a simple representation of the information received by a component. This information can be defined by several features (see the examples given in Table 8.1). Further we shall determine when this information triggers a stimulus. This is the rule referred to by Gibson, quoted above [99]. For an initial step we consider stimuli those which do activate a robot component, and all its processes, and for which we have information specified by some signal. Each of these signals is defined on a domain $U \subset \mathbb{R}^d$, $d \geq 2$. Let us consider first the problem of learning the rules governing the stimuli. Let an observation $z$ be specified as a vector of features, namely $z = (z_1, \ldots, z_d) \in \mathbb{R}^d$, $d \geq 2$. Given a certain amount of observation samples $(z_1, \ldots, z_n)$, obtained from several trials, a sequence of pairs $(z_1, y_1), \ldots, (z_n, y_n)$ is returned, where $z_j \in \mathbb{R}^d$ is the input signal and $y_j \in \mathbb{R}$ is the outcome telling whether $z$ is a stimulus or not, and the goal is to predict the value $h_{\beta}(z^*) = y^*$, given a new observation $z^*$, using the learned parameters $\beta$. Hence $h_{\beta}$ is the function, in a function space, learned using the training data. A set of samples, taken from a trial, is a matrix $n \times d$, where $n$ is the number of observations and $d$ the size of the feature domain. Note that, to choose the outcome value $y_j$ given the features $z$, an operator uses an interface tracing all the signals yielded by the active processes, and can select the value of interest (see Figure 8.7 in Paragraph 8.6.2). These trials and the interface supporting them are described in Paragraph 8.7.5.

Before learning the function $h_{\beta}$ we need to provide a model of the operator choice, as we think that the choice of the operator might be subject to bias and outliers. Given $n$ observations $\{z\}_{j=1,n}$, let each observation $z_j \in \mathbb{R}^d$ be normalized so that $\|z_j\|^2 = \sum_{i=1}^{d} z_{ji}^2 = 1$. Consider the graph of the function $f_i$ as the set $\{(z_1, \ldots, z_j-1, z_j+1, \ldots, z_d, f_i(z_1, \ldots, z_j-1, z_j+1, \ldots, z_d))\}$, where we have chosen a feature $z_j$ as the response variable, thus leaving one feature out. We approximate $f_i$ with an hyper-surface by fitting a polynomial model to the observations. Note that the features are now $z \in \mathbb{R}^{d-1}$. For example, if the domain of the signal $f$ is in $\mathbb{R}^4$, meaning that it has 4 features in its domain, then

$$z_4 = f(z_1, z_2, z_3) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{ijk} z_1^{n-i} z_2^{n-j} z_3^{n-k}$$

(8.1)

And fitting returns the coefficient $c_{ijk}$ values. Why is this hyper-surface interesting? We note that to learn how to predict whether an observation $z^*$ is a stimulus or not we need to learn a function in the space of functions mapping features to the Boolean value $y$. The function is
8. Learning the dynamic processes of shifting and inhibition in cognitive control

Figure 8.2. On the left the features strength, speed, and traffic of the WiFi, and on the right the fitted surface, the normality ellipsoid in magenta and the normality plane in cyan.

learned under the hypothesis that the training set, made of the trials \((z_1, y_1), \ldots, (z_n, y_n)\) is correct. If this set is biased meaning that there are inputs that might not be well scored then the function learned would not be the optimal one. In the context of learning the stimulus model, we rely on several operators opinions gathered while the robot is executing different tasks, these opinions may be corrupted by noise, and difficult to assess, and so on. Therefore we need to learn how to discard outliers, before proceeding to classification. The surface fitting and the method for learning the stimuli rules have exactly this goal, keep outliers in the training data under control and find a rule to see if a trial set is good or not. Let us assume that the normal behavior of a component has a normal distribution, and consider the sample mean \(\mu\) and the sample variance \(\Sigma = V J V^\top\) of the data, and estimate a confidence region around the hyper-ellipsoid centered in \(\mu\), with iso-surface values:

\[
\mathcal{F}_d = \{ z \in \mathbb{R}^{d-1} \mid (z - \mu)^\top \Sigma^{-1} (z - \mu) = D^2 \} \tag{8.2}
\]

Here \(D\) is a scalar. The hyper-ellipsoid has axes length \(\sqrt{D^2 j_i / n}\), with \(j_i\) the \(i\)-th element of the diagonal of \(J\), and directions \(v_i\), with \(v_i\) the \(i\)-th column of \(V\). The 100\((1 - \alpha)\)% confidence region indicates the probability that the region will cover the true center of the (normality) data, here \(0 \leq \alpha \leq 1\), is the statistical significance level. The confidence region is:

\[
\text{ConfReg} = \{ z \in \mathbb{R}^{d-1} \mid (z - \mu)^\top \Sigma^{-1} (z - \mu) \leq \frac{d - 1}{n - (d - 1) - (d - 1) F_{d-1, n-(d-1)}} \} \tag{8.3}
\]

Here \(F_{d-1, n-(d-1)}\) is the Fisher-Snedecor distribution. The iso-surfaces can be chosen according to the distance:

\[
D^2 = \frac{(n - 1)(d - 1) F_{d-1, n-(d-1)}(\alpha)}{n - (d - 1)} \quad \text{providing hyper-ellipsoid axis length} \quad D \sqrt{\frac{j_i}{n}} \tag{8.4}
\]

Using \(\mu, V, J\) and the orthogonality of the eigenvectors we define also a normality plane \(\mathcal{P}\) parallel to the principal axis of the hyper-ellipsoid and passing through \(\mu\). The ellipsoid, the fitted surface and the plane are illustrated in Figure 8.2, right panel. These are used as follows. By construction, if a point \(z_i\) lies inside the confidence hyper-ellipsoid it is normal hence the response variable \(y_i\) is 0. For all the remaining points a rule \(\mathcal{R}\) is defined by relating the number of points having outcome \(y = 1\) with a local critical point.
More precisely, let $f$ be the fitted surface, and consider all those points $z \notin \text{ConfReg} \cup \emptyset$ for which $\nabla f(z) = 0$. Let us count the stimuli at critical points $cp$ outside the confidence region:

$$cp = \sum_{i=1}^{\frac{\partial f}{\partial y_i}} \frac{\partial f}{\partial y_i}$$

where $m_1 = \begin{cases} 1 & \text{if } \frac{\partial f}{\partial y_i} = 0 \\ 0 & \text{otherwise} \end{cases}$ and $m_2 = \begin{cases} 1 & \text{if } y = 1 \\ 0 & \text{otherwise} \end{cases}$

(8.5)

It turns out that $cp > 0.5$. Therefore the rule for stimuli amounts to count the number of critical points that are maxima, minima or saddle points. This rule is preliminary and it serves only to discard those training points which can affect a correct classification, namely the choice of the function $h_\beta$ in the function space. Let $H_f(z) = WAW^\top$ be the non degenerate Hessian of $f(z)$, namely has eigenvalues $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_d| > 0$. Let $k$ be the index of eigenvalues of a minor of $H_f(z)$, then the index of a critical point is defined as follows. $\text{index}(f(z)) = 0$ if $\lambda_1 \ldots \lambda_k \neq 0$ and $\forall k. \lambda_1 \ldots \lambda_k > 0$, then 0 is the index of a minimum. $\text{index}(f(z)) = 1$ if $\lambda_1 \ldots \lambda_k \neq 0$ and $\forall k. (-1)(\lambda_1 \ldots \lambda_k) > 0$, then 1 is the index of a maximum. Finally, $\text{index}(f(z)) = 2$ if either 1) $\lambda_1 \ldots \lambda_n \neq 0$ and neither $\text{index}(f(z)) = 0$ nor $\text{index}(f(z)) = 1$ holds, or 2) there $\exists k < n$ such that $\lambda_1 \ldots \lambda_k < 0$; at index 2 $z$ is a saddle point. The rule $\mathcal{R}(f, b, \lambda_1, \ldots, \lambda_n)$, $b \in \{0, 1, 2\}$ is defined as follows. Let $(\langle z_1, y_1 \rangle, \ldots, \langle z_n, y_n \rangle)$ be the training set and let $b \in \{0, 1, 2\}$:

$$\mathcal{R}(f, b, \lambda_1, \ldots, \lambda_n) = \begin{cases} 1 & \text{if } z_i \text{ is a critical points of } f \text{ and } z_i \notin (\text{ConfReg} \cup \emptyset) \text{ and } \frac{\sum_{i=1}^{\text{index}(f(z_i)) \neq b}}{\sum_{i=1}^{\text{index}(f(z_i)) = b}} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

(8.6)

Therefore a set of samples $(\langle z_1, y_1 \rangle, \ldots, \langle z_n, y_n \rangle)$ in a trial is good for classification of stimuli if for all $\langle z_i, y_i \rangle$, such that $y_i = 1$, $\text{index}(f(z_i)) = b$ iff $\mathcal{R}(f, b, \lambda_1, \ldots, \lambda_k) = 1$.

Given a complete set of good samples $(\langle z_1, y_1 \rangle, \ldots, \langle z_n, y_n \rangle)$ a model for stimuli prediction can be provided considering logistic regression:

$$h_\beta(z) = g_{\text{logistic}}(\beta^\top z)$$

(8.7)

Here $g_{\text{logistic}}$ is the logistic function taking as arguments the observed features $z$, the parameters $\beta$ to be estimated, and returns a value between 0 and 1, namely $1 \left/ \left(1 + \exp(-\beta^\top z)\right)\right.$

Given a new observation $z^\star$, $h_\beta(z^\star)$ is the hypothesis returning the binary response within the variable $y$:

$$y = \begin{cases} 1 & \text{if } h_\beta(z^\star) \geq 0.5 \\ 0 & \text{if } h_\beta(z^\star) < 0.5 \end{cases}$$

(8.8)

The logistic regression coefficients $\beta = (\beta_1, \ldots, \beta_{d-1})$, are obtained from the good trial set $(\langle z_1, y_1 \rangle, \ldots, \langle z_n, y_n \rangle)$, by minimizing the following cost function:

$$L(\beta) = -\frac{1}{n} \left( \sum_{i=1}^{n} y_i \log h_\beta(z_i) + (1 - y_i) \log(1 - h_\beta(z_i)) \right) + \frac{\xi}{2n} \sum_{j=1}^{d} \beta_j^2$$

(8.9)

Here $\xi$ is the regularization parameter and $\sum_{j=1}^{d} \beta_j^2$ the regularization term, which are used just in case the features induce a high-dimensional space. Newton-Raphson gradient descent
Learning the dynamic processes of shifting and inhibition in cognitive control

is used to obtain a minimum. In general $\xi$ is 0 whenever the features number is less than 4, such as for the battery or the WiFi, while for certain vision tasks it is greater than 1.

We have thus provided a model to estimate the stimuli for a given input signal. The stimulus, once predicted, is transduced to a process, namely to its representation, through an environment term, which will specified in Paragraph 8.5.

8.3 The stimulus-response model

In this paragraph we introduce the following problem: how the system learns to associate a mental state to a stimulus.

Having introduced the stimulus model and having clarified how the stimulus is brought to the robot current mental state, defined in eq.(8.20) via the active processes, we present here the learning model for the stimulus-response. We are interested in the association of a stimulus with processes defining a robot mental state $\tau(\Delta t)$. Given the features $z$, the value $y$ returned by the learned function $h_\beta(z)$, and given the environment term $E$ carrying the realization of the features and of the outcome $y$, we introduce a stimulus domain as:

$$Y = \{y_{\pi_C}|\delta^+_{\pi_C}(h_\beta(z),t) = 1, t \in \Delta t\}, \quad \forall \pi_C \in \tau(\Delta t) \quad (8.10)$$

Note that here $y_{\pi_C}$ is a term naming a stimulus, and it is different from the value $y \in \{0, 1\}$. In the following we can use $\pi_j$ instead of $\pi_C$, as an index for $y_{\pi_C}$. In particular, as in the tables processes take indexes specifying both the process number, according to the enumeration of each component processes, and the component name, then $y_{\pi_C}$ appears as $y_j, \pi_C$ in Table 8.3, Table 8.5 and Table 8.6.

Our goal is to model a mapping $\gamma: Y \rightarrow \mathcal{T}$ from stimuli to mental states, with $\mathcal{T}$ the set of all mental states specified by the robot task library (see Table 8.4). Because in this paragraph we deal with a simplified notation of the mental state, for the purpose of presenting the learning model, we shall refer to a mental state by denoting it task state.

We consider a finite and limited set of components and processes, illustrated in Table 8.2 and in Table 8.3. The stimulus-response mapping function $\gamma$ is based on a content-based recommendation system, which takes advantage of a selection matrix $A = \{a_{ij}\}$, $\forall i = 1, \ldots, |Y|$, $\forall j = 1, \ldots, |\mathcal{T}|$ to build a relation between stimuli and task states. Indeed, each entry $a_{ij}$ of the matrix $A$ represents the rating given by the stimulus $y_{\pi_C}$ to the task state $\tau_i \in \mathcal{T}$. However, most of these entries are unknown so that the selection matrix $A$ turns out to be sparse. The recommendation system can predict the unknown ratings for the stimulus-task pairs of $A$. The model is used to score the task states, with respect to the stimulus triggered according to the stimuli model, and sort them out according to their scores. Task states whose score value is greater than a threshold $\kappa$ can be selected as possible responses to the triggered stimulus.

The domain of the content-based recommendation system is composed of the two classes of entities, reported in Table 8.3 and Table 8.4, which we refer to as Stimuli and Task States, respectively. The ratings $a_{ij}$ of the stimulus-task pairs in the selection matrix $A$, is obtained for a number of entries by trials as follows.

A graphical user interface monitors all the processes running on the robot as well as the occurrences of stimuli, at real-time. The interface enables the operator to monitor the robot (mental) task state and to select a task state $\tau_i \in \mathcal{T}$, whenever a stimulus $y_{\pi_C} \in Y$ is presented. The interface is presented in Paragraph 8.6.2, Figure 8.7 and the experiments are described...
### Table 8.2. Robot Components and Processes

<table>
<thead>
<tr>
<th>Organ Id</th>
<th>Organ Description</th>
<th>Process Id</th>
<th>Process Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>Battery</td>
<td>$\pi_{1,C_1}$</td>
<td>Power Level Monitoring</td>
</tr>
<tr>
<td>$C_2$</td>
<td>WiFi</td>
<td>$\pi_{1,C_2}$</td>
<td>Connection Quality Checking</td>
</tr>
<tr>
<td>$C_3$</td>
<td>3D Laser Scan</td>
<td>$\pi_{1,C_3}$</td>
<td>ROS Drivers Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_3}$</td>
<td>Laser Rotating</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_3}$</td>
<td>Scanning</td>
</tr>
<tr>
<td>$C_4$</td>
<td>SLAM</td>
<td>$\pi_{1,C_4}$</td>
<td>ROS Packages Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_4}$</td>
<td>2D Mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_4}$</td>
<td>3D Mapping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{4,C_4}$</td>
<td>Localizing</td>
</tr>
<tr>
<td>$C_5$</td>
<td>Vision System</td>
<td>$\pi_{1,C_5}$</td>
<td>ROS Drivers Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_5}$</td>
<td>Calibrating</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_5}$</td>
<td>Streaming</td>
</tr>
<tr>
<td>$C_6$</td>
<td>Visual Perception</td>
<td>$\pi_{1,C_6}$</td>
<td>ROS Packages Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_6}$</td>
<td>Detecting Object/Person</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_6}$</td>
<td>Recognizing Object/Person</td>
</tr>
<tr>
<td>$C_7$</td>
<td>Robot Engine</td>
<td>$\pi_{1,C_7}$</td>
<td>ROS Drivers Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_7}$</td>
<td>Locomotion System Monitoring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_7}$</td>
<td>Temperature Monitoring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{4,C_7}$</td>
<td>Locking Differential Drive</td>
</tr>
<tr>
<td>$C_8$</td>
<td>Trajectory Control</td>
<td>$\pi_{1,C_8}$</td>
<td>ROS Packages Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_8}$</td>
<td>Planning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_8}$</td>
<td>Tracking</td>
</tr>
<tr>
<td>$C_9$</td>
<td>Operator Control Unit</td>
<td>$\pi_{1,C_9}$</td>
<td>Human to Robot Exchanging msg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_9}$</td>
<td>Robot to Human Exchanging msg</td>
</tr>
<tr>
<td>$C_{10}$</td>
<td>Microphone</td>
<td>$\pi_{1,C_{10}}$</td>
<td>ROS Drivers Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_{10}}$</td>
<td>Hearing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_{10}}$</td>
<td>Speech Recognizing</td>
</tr>
<tr>
<td>$C_{11}$</td>
<td>Memory</td>
<td>$\pi_{1,C_{11}}$</td>
<td>Data Reading</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_{11}}$</td>
<td>Data Writing</td>
</tr>
<tr>
<td>$C_{12}$</td>
<td>IMU</td>
<td>$\pi_{1,C_{12}}$</td>
<td>ROS Drivers Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_{12}}$</td>
<td>Robot Pose Estimating</td>
</tr>
<tr>
<td>$C_{13}$</td>
<td>ROS Master Core</td>
<td>$\pi_{1,C_{13}}$</td>
<td>ROS Core Running</td>
</tr>
<tr>
<td>$C_{14}$</td>
<td>Central Processing Unit</td>
<td>$\pi_{1,C_{14}}$</td>
<td>Payload Monitoring</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_{14}}$</td>
<td>Temperature Monitoring</td>
</tr>
<tr>
<td>$C_{15}$</td>
<td>Terrain Analysis</td>
<td>$\pi_{1,C_{15}}$</td>
<td>ROS Packages Running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{2,C_{15}}$</td>
<td>Traversability Analysing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{3,C_{15}}$</td>
<td>Ground Subtracting &amp; Analysis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{4,C_{15}}$</td>
<td>Obstacle Detecting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\pi_{5,C_{15}}$</td>
<td>Adapting Robot Morphology</td>
</tr>
</tbody>
</table>
8. Learning the dynamic processes of shifting and inhibition in cognitive control

Table 8.3. Stimuli associated to the yields of the robot processes

(a)

<table>
<thead>
<tr>
<th>Process Id</th>
<th>Stimulus Id</th>
<th>Stimulus Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_1 c_1$</td>
<td>$y_{\pi_1 c_1}$</td>
<td>Battery Exhaustion</td>
</tr>
<tr>
<td>$\pi_1 c_2$</td>
<td>$y_{\pi_1 c_2}$</td>
<td>WiFi Disconnection</td>
</tr>
<tr>
<td>$\pi_1 c_3$</td>
<td>$y_{\pi_1 c_3}$</td>
<td>3D Laser Device Error</td>
</tr>
<tr>
<td>$\pi_2 c_1$</td>
<td>$y_{\pi_2 c_1}$</td>
<td>3D Laser Rotation Stuck</td>
</tr>
<tr>
<td>$\pi_3 c_1$</td>
<td>$y_{\pi_3 c_1}$</td>
<td>Scan Data Not Available</td>
</tr>
<tr>
<td>$\pi_1 c_4$</td>
<td>$y_{\pi_1 c_4}$</td>
<td>SLAM ROS Nodes Died</td>
</tr>
<tr>
<td>$\pi_2 c_4$</td>
<td>$y_{\pi_2 c_4}$</td>
<td>2D Map Not Available</td>
</tr>
<tr>
<td>$\pi_3 c_4$</td>
<td>$y_{\pi_3 c_4}$</td>
<td>3D Map Not Available</td>
</tr>
<tr>
<td>$\pi_4 c_4$</td>
<td>$y_{\pi_4 c_4}$</td>
<td>Coordinate Frames Transformation Error</td>
</tr>
<tr>
<td>$\pi_1 c_5$</td>
<td>$y_{\pi_1 c_5}$</td>
<td>Vision System Device Error</td>
</tr>
<tr>
<td>$\pi_2 c_5$</td>
<td>$y_{\pi_2 c_5}$</td>
<td>Calibration Error</td>
</tr>
<tr>
<td>$\pi_3 c_5$</td>
<td>$y_{\pi_3 c_5}$</td>
<td>Camera Images Not Available</td>
</tr>
<tr>
<td>$\pi_1 c_6$</td>
<td>$y_{\pi_1 c_6}$</td>
<td>Visual Perception ROS Nodes Died</td>
</tr>
<tr>
<td>$\pi_2 c_6$</td>
<td>$y_{\pi_2 c_6}$</td>
<td>Detected Object/Person</td>
</tr>
<tr>
<td>$\pi_3 c_6$</td>
<td>$y_{\pi_3 c_6}$</td>
<td>Recognized Object/Person</td>
</tr>
<tr>
<td>$\pi_1 c_7$</td>
<td>$y_{\pi_1 c_7}$</td>
<td>Engine Device Error</td>
</tr>
<tr>
<td>$\pi_2 c_7$</td>
<td>$y_{\pi_2 c_7}$</td>
<td>Locomotion System Stuck</td>
</tr>
<tr>
<td>$\pi_3 c_7$</td>
<td>$y_{\pi_3 c_7}$</td>
<td>Temperature Level</td>
</tr>
<tr>
<td>$\pi_4 c_7$</td>
<td>$y_{\pi_4 c_7}$</td>
<td>Differential Drive Stuck</td>
</tr>
<tr>
<td>$\pi_1 c_8$</td>
<td>$y_{\pi_1 c_8}$</td>
<td>Trajectory Control Nodes Died</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Process Id</th>
<th>Stimulus Id</th>
<th>Stimulus Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_2 c_8$</td>
<td>$y_{\pi_2 c_8}$</td>
<td>Planning Error</td>
</tr>
<tr>
<td>$\pi_3 c_8$</td>
<td>$y_{\pi_3 c_8}$</td>
<td>Tracking Error</td>
</tr>
<tr>
<td>$\pi_1 c_9$</td>
<td>$y_{\pi_1 c_9}$</td>
<td>Human to Robot Message</td>
</tr>
<tr>
<td>$\pi_2 c_9$</td>
<td>$y_{\pi_2 c_9}$</td>
<td>Robot to Human Message</td>
</tr>
<tr>
<td>$\pi_1 c_{10}$</td>
<td>$y_{\pi_1 c_{10}}$</td>
<td>Microphone Device Error</td>
</tr>
<tr>
<td>$\pi_2 c_{10}$</td>
<td>$y_{\pi_2 c_{10}}$</td>
<td>Signal Noise Ratio</td>
</tr>
<tr>
<td>$\pi_3 c_{10}$</td>
<td>$y_{\pi_3 c_{10}}$</td>
<td>Recognized Speech</td>
</tr>
<tr>
<td>$\pi_1 c_{11}$</td>
<td>$y_{\pi_1 c_{11}}$</td>
<td>Read Access to Missing Data</td>
</tr>
<tr>
<td>$\pi_2 c_{11}$</td>
<td>$y_{\pi_2 c_{11}}$</td>
<td>Write Access to Memory Full</td>
</tr>
<tr>
<td>$\pi_3 c_{11}$</td>
<td>$y_{\pi_3 c_{11}}$</td>
<td>Memory State</td>
</tr>
<tr>
<td>$\pi_1 c_{12}$</td>
<td>$y_{\pi_1 c_{12}}$</td>
<td>IMU Device Error</td>
</tr>
<tr>
<td>$\pi_2 c_{12}$</td>
<td>$y_{\pi_2 c_{12}}$</td>
<td>Accuracy</td>
</tr>
<tr>
<td>$\pi_1 c_{13}$</td>
<td>$y_{\pi_1 c_{13}}$</td>
<td>ROS Master Connection</td>
</tr>
<tr>
<td>$\pi_1 c_{14}$</td>
<td>$y_{\pi_1 c_{14}}$</td>
<td>Central Processing Unit Overloaded</td>
</tr>
<tr>
<td>$\pi_2 c_{14}$</td>
<td>$y_{\pi_2 c_{14}}$</td>
<td>Temperature Level</td>
</tr>
<tr>
<td>$\pi_1 c_{15}$</td>
<td>$y_{\pi_1 c_{15}}$</td>
<td>Terrain Analysis Nodes Died</td>
</tr>
<tr>
<td>$\pi_2 c_{15}$</td>
<td>$y_{\pi_2 c_{15}}$</td>
<td>Stability Index</td>
</tr>
<tr>
<td>$\pi_3 c_{15}$</td>
<td>$y_{\pi_3 c_{15}}$</td>
<td>Terrain Roughness</td>
</tr>
<tr>
<td>$\pi_4 c_{15}$</td>
<td>$y_{\pi_4 c_{15}}$</td>
<td>Detected Obstacle</td>
</tr>
<tr>
<td>$\pi_5 c_{15}$</td>
<td>$y_{\pi_5 c_{15}}$</td>
<td>Flippers Configuration Error</td>
</tr>
</tbody>
</table>
### Table 8.4. Processes required to perform the robot tasks

<table>
<thead>
<tr>
<th>Task Id</th>
<th>Task Description</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>Robot System Recovery</td>
<td>${ R_{e1}, R_{r1}, R_{e2}, R_{r2}, R_{e3}, R_{r3}, R_{e4}, R_{r4} }$</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>Request for Path</td>
<td>${ R_{e1}, R_{r1}, R_{e2}, R_{r2}, R_{e3}, R_{r3}, R_{e4}, R_{r4}, R_{e5}, R_{r5} }$</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>Track Trajectory</td>
<td>${ R_{e1}, R_{r1}, R_{e2}, R_{r2}, R_{e3}, R_{r3}, R_{e4}, R_{r4}, R_{e5}, R_{r5}, R_{e6}, R_{r6} }$</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>Recovery WiFi Signal</td>
<td>${ R_{e1}, R_{e2}, R_{e3}, R_{e4}, R_{r2}, R_{r3}, R_{r4}, R_{e5}, R_{r5} }$</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>Return to Base</td>
<td>${ R_{e1}, R_{e2}, R_{e3}, R_{r1}, R_{r2}, R_{r3}, R_{r4}, R_{r5} }$</td>
</tr>
<tr>
<td>$\tau_6$</td>
<td>Store Data</td>
<td>${ R_{e1}, R_{e2}, R_{e3} }$</td>
</tr>
<tr>
<td>$\tau_7$</td>
<td>Cool Central Processing Unit</td>
<td>${ R_{e1}, R_{e2}, R_{e3}, R_{r1}, R_{r2}, R_{r3}, R_{r4}, R_{r5} }$</td>
</tr>
<tr>
<td>$\tau_8$</td>
<td>Label Map</td>
<td>${ R_{e1}, R_{e2}, R_{e3}, R_{r1}, R_{r2}, R_{r3}, R_{r4}, R_{r5} }$</td>
</tr>
<tr>
<td>$\tau_9$</td>
<td>Serve Operator Request</td>
<td>${ R_{e1}, R_{e2}, R_{e3}, R_{e4}, R_{r1}, R_{r2}, R_{r3}, R_{r4} }$</td>
</tr>
<tr>
<td>$\tau_{10}$</td>
<td>Approach Object/Person/Sound Source</td>
<td>${ R_{e1}, R_{e2}, R_{e3}, R_{e4}, R_{e5}, R_{r1}, R_{r2}, R_{r3}, R_{r4} }$</td>
</tr>
<tr>
<td>$\tau_{11}$</td>
<td>Overcome Obstacle</td>
<td>${ R_{e1}, R_{e2}, R_{e3}, R_{e4}, R_{r1}, R_{r2}, R_{r3}, R_{r4}, R_{r5} }$</td>
</tr>
<tr>
<td>$\tau_{12}$</td>
<td>Data Retrieval</td>
<td>${ R_{e1}, R_{e2} }$</td>
</tr>
</tbody>
</table>

### Table 8.5. Entries of the selection matrix (a)

<table>
<thead>
<tr>
<th>Task Id</th>
<th>$\tau_{i}$</th>
<th>Stimuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>0 (0.6069) 0.4169</td>
<td>0.5 0.444 0.4555 0.125 0.3375 0.1669 0.2 0.222 0.1669 0 0 0.1875 0 0 0.3 0.5 0.75</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0 (0.6069)</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_6$</td>
<td>0.1428 0.2 0.25 0.1669 0.2222 0.0599 0.145 0 0 0.0839 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_7$</td>
<td>0.1669 0.0839 0.0839 0.1669 0 0.1818 0.125 0.0314 0 0 0.0839 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_8$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_9$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_{10}$</td>
<td>0.0839 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_{11}$</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>$\tau_{12}$</td>
<td>0.1669 0.0839 0.0839 0.1669 0.2222 0.2222 0.1669 0.2222 0.1669 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

### Table 8.6. Entries of the selection matrix (b)

<table>
<thead>
<tr>
<th>Task Id</th>
<th>$\tau_{i}$</th>
<th>Stimuli</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>0.125 0.375 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_5$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_6$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_7$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_{10}$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_{11}$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>$\tau_{12}$</td>
<td>0 0 0</td>
<td>0.125 0.1111 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>
in Paragraph 8.7.5. Data from each trial, reporting the occurrences of the pairs \( \langle y_{\pi_j}, \tau_i \rangle \) have been collected and used to fill the entries of the selection matrix A (see Table 8.5 and Table 8.6). The zeros entries of A indicate that the human operator has not assigned a rating to the corresponding stimulus-task pair. As expected, the matrix turns out to be sparse due to the missing information about all the entries. The content-based recommendation system is further extended with features of task states, affecting the prediction of the unknown ratings. For each task state \( \tau_i \in \mathcal{T} \), a vector of task features \( q_i \in \mathbb{R}^d \), \( d > 1 \) the features space dimension, is defined by:

1. The reward of the task.
2. The computational payload required to accomplish the task.
3. The performance of the robot to accomplish the task.
4. The rate of failing to accomplish the task.

Predicting unknown ratings in the selection matrix requires to estimate the parameters \( \rho_j \in \mathbb{R}^d \) of the model, for each \( y_{\pi_j} \in Y \). Let \( q_i \) be the feature vector for task state \( \tau_i \) and let \( \rho_j \) be the estimated model parameters, then the prediction \( \hat{a}_{ij} \) of the rate that the stimulus \( y_{\pi_j} \) assigns to task state \( \tau_i \), is obtained as follows:

\[
\hat{a}_{ij} = \rho_j^\top q_i \quad (8.11)
\]

The problem of learning \( q_i \) and \( \rho_j \) amounts to minimize the following cost function, in which \( q_i \) and \( \rho_j \) are simultaneously considered. Let \( n \) be the number of tasks, and let \( m \) be the number of considered stimuli, then:

\[
J(q_1, \ldots, q_n, \rho_1, \ldots, \rho_m) = \frac{1}{2} \sum_{(i,j): r_{ij} = 1} \left( \rho_j^\top q_i - a_{ij} \right)^2 + b_1 \frac{1}{2} \sum_{i=1}^n \sum_{k=1}^d q_{ik}^2 + b_2 \frac{1}{2} \sum_{j=1}^m \sum_{k=1}^d \rho_{jk}^2 \quad (8.12)
\]

Here the first term of eq. (8.12) corresponds to the square error term and the last terms are the regularization terms of \( q_i \) and \( \rho_j \), respectively. The parameter \( r_{ij} \) is the indicator function of the matrix A, and \( b_1, b_2 \) the regularization parameters. The vectors \( q_i \) and \( \rho_j \),...
for all \( y_{\pi j} \in Y \) and for all \( \tau_i \in \mathcal{T} \), are obtained as the solutions of the following optimization problem:

\[
\min_{\mathbf{q}_1, \ldots, \mathbf{q}_n, \mathbf{\rho}_1, \ldots, \mathbf{\rho}_m} J(\mathbf{q}_1, \ldots, \mathbf{q}_n, \mathbf{\rho}_1, \ldots, \mathbf{\rho}_m) \tag{8.13}
\]

The local optima of the objective function in eq. (8.13) can be found with gradient descent. The update rules, for every \( j = 1, \ldots, n \) and for every \( i = 1, \ldots, d \) are given by the following equations:

\[
q_{ik} := q_{ik} - \zeta_1 \left( \sum_{r_{ij}=1}^{\rho_{j}^T \mathbf{q}_i - a_{ij}} \rho_{jk} + b_1 q_{ik} \right)
\]

\[
\rho_{jk} := \rho_{jk} - \zeta_2 \left( \sum_{r_{ij}=1}^{\rho_{j}^T \mathbf{q}_i - a_{ij}} q_{ik} + b_2 \rho_{jk} \right)
\]

This is a collaborative filtering model with both the stimuli and the task states biases defined together with their properties [118, 196]. As this model is very simple, the parameters can be estimated well, even under sparsity. However, the empirical prediction quality is typically low (see Figure 8.3(a)). Moreover, as the number of features of task states increases, the Mean Squared Error (MSE) considerably decreases and, consequently, the empirical prediction quality gets higher (see Figure 8.3(b)). Note that the criteria used for the convergence of the gradient descent algorithm as well as the accuracy of the content-based recommendation system affect the behavioral modes of the robot.

In conclusion, let \( \mathbf{\hat{A}} \) be the completed selection matrix, according to the estimated parameters \( \mathbf{P} = \{ \rho_{j}^T ; j = 1, \ldots, n \} \) and according to content-based recommendation system features \( \mathbf{Q} = \{ \mathbf{q}_i^T ; i = 1, \ldots, d \} \).

The function \( \gamma : Y \rightarrow \mathcal{T} \), mapping stimuli to task states is defined as follows. Let \( \gamma_1 : Y \times \mathbb{R}^2 \rightarrow \mathbb{R}^n \) abbreviate the estimation of a column \( a_j \) of \( \mathbf{\hat{A}} \), that is, \( \gamma_1(y_{\pi j}, \mathbf{\hat{A}}) = a_j \). Let \( \gamma_2 \) be such that \( \gamma_2(a_j, i) = \tau_i \). Then \( \gamma \) is the mapping selecting all the task states that have a rate \( a_{ij} > \kappa \):

\[
\gamma(y_{\pi}) = \{ \tau_i | \gamma_2(\gamma_1(y_{\pi}, \mathbf{\hat{A}}), i) > \kappa \} \tag{8.14}
\]

In conclusion, if \( y_{\pi c} \) is a term denoting a stimulus triggered during a task execution, within a time lapse \( \Delta t \), then the mapping \( \gamma \) relating stimuli to task states, takes as input the term \( y_{\pi c} \), looks up the pre-estimated selection matrix \( \mathbf{\hat{A}} \) and returns a subset of the set of task states \( \mathcal{T} \), which are the states that are more likely to be effective to respond to the stimulus.

### 8.4 States, processes and goals

In this paragraph we introduce and model the following concepts:

1. Actions and processes.
2. The mechanism generating processes to execute a chosen task.
3. The structure of constraints allowing processes to work in parallel.
4. The robot current mental state.
5. The interference and reconfiguration costs for changing state.

The above concepts form the basic structure inside which the stimulus-response model can be described. We note that by metonymy we interchange the term task with the term goal, where a task is the set of processes to reach a goal, and a goal is a state of the world. But as long as processes are involved in the state of the world (or the robot mental state) the two terms refer to the same state. To model tasks and processes we resort to the action theory $\mathcal{A}$ based on the formalism of the Situation Calculus. In this formalism the semantic domain is partitioned in sorts $(A, S, T, U, W)$ where $A$ stands for actions, $S$ for situations, $T$ for time (the reals), $U$ for the set of boolean values, and $W$ for every other object. Given the sorted domain the representation language $\mathcal{L}$ is made of predicates and terms (variables and functions of arity $n \geq 0$) and ruled by the first order predicate calculus with the exception of an inductive axiom, we refer the reader to [207] for a full description of the formalism. The representation language is used to provide the robot with a reasoning and knowledge structure about actions, with their preconditions and effects, about processes and about the constraints linking the parallel processes.

A process is a predicate $\pi(x, u, t, do(a, s))$ taking as arguments, together with terms $x = (x_1, \ldots, x_n), n \geq 0$ of any sort, a single term of sort boolean for the stimulus, a term of sort time and a single term $do(a, s)$ of sort situation. The special function $do$, of sort situation is an action builder, that is, it defines a situation as a list of actions. More precisely $do$ is a function taking as argument a term of sort action and a term of sort situation, and returning a term of sort situation $do : A \times S \rightarrow S$, which is a list of actions. Through its arguments a process carries the information about time $t$ and about the history of all processes of a component via all its start and end actions, activated up to the time $t$. For example, the process governing the battery status is represented by the following transition statement:

$$\pi_{\text{batt}}(x, u, t, do(a, s)) \equiv [a = \text{start}_{\text{batt}}(x, t) \lor \exists t'.(t' \leq t \land \pi_{\text{batt}}(x, t', s) \land \forall t''.(t'' > t' \land t'' \leq t) \rightarrow a \neq \text{end}_{\text{batt}}(x, t''))] \land \exists t' t''(t' < t'') = u$$

(8.15)

The above transition axiom tells the robot that the battery process is active if either the action start battery $\text{start}_{\text{batt}}$ is executed at time $t$ or the process was already active at some previous time $t'$, and at any time between $t'$ and $t$ (the current time) it was not ended. The axiom tells also the robot whether a stimulus about the battery status occurred or not, by the special function $\delta$. This term takes as input the realization of the yields of the process and returns, when the process is active, the value $u = 1$ if the signal elicits a stimulus, and $u = 0$ otherwise. The function $\delta$, namely the environment term, is the mapping between the representation language $\mathcal{L}$, in which the robot processes are defined, and the stimuli identified by the stimuli model (Paragraph 8.2). More specifically:

**Yields of a process.** Let $\pi_C$ be a process active at time $t$. Let $\delta^*_\pi_C : \mathbb{R} \times T \rightarrow U$ be a function, mapping the outcome $y \in \mathbb{R}$ of $h_B(z)$, at time $t$, to the term $u$ of sort $U$ of the language $\mathcal{L}$. Let $X_{\pi_C} = (X_1, \ldots, X_k)$, $k \geq 2$ be a random variable whose realizations are the yields that the process $\pi_C$ can collect, during its execution, from its sensory system. Let $\delta^{+}_\pi_C : \mathbb{R}^d \times T \rightarrow \mathbb{R}^k$ be a function, mapping the features $z \in \mathbb{R}^k$ to the realizations of $X_{\pi_C}$. The pre-images of $X_{\pi_C}$ are the variables of the representation language $\mathcal{L}$, namely $\text{re}(\delta^+_\pi_C) = \{x \in \text{Var}(\mathcal{L}) | x \text{ is a realization of } X_{\pi_C}, \delta^+_\pi_C(z, t) = X_{\pi_C}\}$. Then, the mapping between $\mathcal{L}$ and the stimuli is given by the identity $\delta(\text{re}(\delta^+_\pi_C(z, t)), t) = \delta^*_\pi_C(h_B(z), t)$ (see Fig. 8.4). □
The relevance of the mapping can be appreciated by considering the above transition axiom. When a process is inferred, the time variable $t$ is not instantiated, and thus $u$ is not grounded, therefore it cannot be determined whether $E(x,t) = 1$, the stimulus is fired, or $E(x,t) = 0$, no stimulus is fired, because the state of the process needs to be measured, via the realization of $X_{\pi}$, and this can be done only while the process is active.

Note that these transition axioms, one for each process, interpret also an important law, namely the persistence of a process. For example the above transition states that the battery is working up to when an action $end\_batt$, that ends the battery process, is activated. The axiom also returns the current time during which the process is active, as an argument of the process predicate. To activate a $start$ or $end$ action there is a special predicate $Poss$ telling the robot whether the action can be done given the current situation $s$ and time $t$, for example

$$Poss(end\_batt(x,t),s) \equiv at(home,t,s) \land stop(t,s)$$

(8.16)

tells the robot that the battery process can be ended if it is at home and not moving. A statement like the above one is called a precondition axiom. Note that a state is defined by all the predicates (namely processes and facts) that are verified at the specific situation and time, in the intended model. Namely, if $\mathcal{M}$ is the model of the language satisfying the whole theory, we might say that $\mathcal{M} \models \varphi(t,s)$ to indicate that the finite set of sentences $\varphi$ holds at time $t$, in situation $s$ in the model, in so specifying a finite state. On the other hand we can also write $\mathcal{A} \models \exists x,u,t,s. \mathcal{P}(x,u,t,s)$ to indicate that all the models (actually the intended one) of the action theory $\mathcal{A}$ are also models of the process $\mathcal{P}$ at some time $t$ and situation $s$, in so indicating that the situation $s$, $s = do(a_n, do(a_{n-1}, \ldots, do(a_1,S_0)))$ returns a sequence of actions, which governs the processes that can be activated in $S_0$ to perform some task, in which the component $C$ is involved.

Robot component are both hardware devices and software modules such as engine, dialog, vision, mapping, planning, path planning, and so on, handling several processes and needing to be regulated not only by transition axioms and by precondition axioms, but
also by time constraints. These constraints govern processes relations. For example, the following eq. (8.17) says that to start the process of approaching a certain location, the path planner process needs to be active, which in turns requires the mapping to be active:

\[
\pi_{\text{path-planning}}(x,u,t,s) \text{ during } \pi_{\text{mapping}}(x',u',t',s') \text{ before } \pi_{\text{approach}}(x'',u'',t'',s'')
\]  

(8.17)

Here \(s, s', s''\) are situations, the variables arrays \(x, x', x''\) identify objects of interest for the processes, and their features, such as for example location and temperature, while \(u, u', u''\) are the boolean terms yielded, respectively, by each of the three mentioned processes, and \(t, t', t''\) are time variables. Where no quantifier is mentioned, variables are universally quantified.

A constraint such as \((\pi_C \text{ before } \pi_{C_2})\) is defined by a formula setting the conditions on the components, that have to be satisfied. Different approaches for defining constraints in a first order language have been proposed in 10, 19, 177, 168, 82, 38, 263. A definition for each time constraint between processes, according to the linear-interval based time [2] is provided in [81]. The definition exploits a parallelization of situations into several timelines, and each timeline specify the evolution of all the processes of a component. A timeline is a special situation taking into account only the start and end actions of the processes of a component. Moreover, to allow parallel situations a transformation of the situation argument into a set of situations is introduced. For example the time relation \(\text{during}\) is based on the definitions of greatest start time and least end time, that specifies a process activity within a start and end time, up to a fixed time \(t\). The greatest start time is defined as:

\[
greatestStart_\pi(x,a,t) \overset{\text{def}}{=} \exists t'. a = \text{start}_\pi(x,t') \land \neg \exists t'', a', t' \leq t'' \leq t \land a'' = \text{end}_\pi(x,t'').
\]  

(8.18)

An analogous definition is given for \(\text{leastEnd}\), the least end action, in so characterizing the \(\Delta t\) inside which a process is certainly active or certainly not-active. Given the above definition, the time relation \(\text{during}\) is defined as follows:

\[
\{\pi_C(x,u,t,d(o,a,s)) \text{ during } \pi_{C_2}(x',u',t',d(o',a'),s')\} \overset{\text{def}}{=} \begin{cases} 
\begin{array}{l}
s \in s' \land s' = s' \land \pi_C(x,u,t,d(o,a,s)) \land \pi_{C_2}(x',u',t',d(o',a'),s') \\
\left[\greatestStart_{\pi_C}(x,a,t) \geq \greatestStart_{\pi_{C_2}}(x',a',t')\right] \land \\
\left[\text{leastEnd}_{\pi_C}(x,a,t) \leq \text{leastEnd}_{\pi_{C_2}}(x',a',t')\right]
\end{array}
\end{cases}
\]  

(8.19)

Here \(s\) (and \(s'\)) is a bag of timelines, namely timelines of different robot components. Note that in the formulas specifying constraints (as in the above formula (8.19)) the variables have a lambda binding, which means that the variables are bound by the quantifiers inside the sentence in which they are embedded, because the time constraints are macro sentences of the language. Temporal constraints are typically those defined in [2] among which are \(Op = \{\text{during}, \text{before}, \text{meet}, \text{start}, \text{finishes}, \text{overlaps}, \text{equal}\}\). Similarly to temporal constraints, spatial constraints are defined following the topological relations introduced in [220]. Spatial and time constraints are integrated (see [265, 76]). The operators are included in \(Op\) and the set of constraints is indicated by \(\mathcal{C}\) and are special formulas of the language as only the definiens (namely the l.h.s of the definitions) are expressed in the language \(\mathcal{L}'\).

The mental state of the robot is the set of processes that are active at a specific interval time \(\Delta t = [t^-, t^+]\). Let \(\mathcal{S}\) be the set of bag of timelines, up to \(t \in \Delta t\). Given the intended model \(\mathcal{M}\) of the theory \(\mathcal{A} \cup \mathcal{C}\), the current mental state of the robot is:

\[
\tau(\Delta t) = \{x, u.\pi_C(x,u,t,s) | s \in s, s \in \mathcal{S}, t \in \Delta t, \text{ and } \mathcal{M} \models \exists x, u.\pi_C(x,u,t,s)\}
\]  

(8.20)
8.4 States, processes and goals

If the robot has to infer the processes needed to execute a task (here we already begin to think about switching and inhibition) it may happen that for some process the starting action $\text{start}_{\pi_c}$ or the ending action $\text{end}_{\pi_c}$ are not derivable as possible, namely, $\mathcal{A} \cup \mathcal{C} \not\models \exists x, t, s \text{Poss}(\text{start}_{\pi_c}(x, t), s)$. To modify the current execution and make the activation of a process possible, it is desirable to obtain an explanation for enabling a specific action $a$. This is a formula $\psi \in \mathcal{L}$, such that:

$$\mathcal{A} \cup \mathcal{C} \not\models \psi \text{ and } \mathcal{A} \cup \mathcal{C} \cup \{\psi\} \models \exists x, t, s \text{Poss}(\text{start}_{\pi_c}(x, t), s) \quad (8.21)$$

With the proviso that $\mathcal{A} \cup \mathcal{C} \cup \{\psi\} \not\models$, meaning that is does not imply the contradiction. Here $\mathcal{A}$ is the action theory, $\mathcal{C}$ is the set of temporal-spatial constraints and $\psi$ is the formula to be added, to ensure the inference. We show that a formula $\psi$, which needs to be added to infer processes, their preconditions, transitions and constraints, have a reconfiguration cost and an interference cost, as defined below.

**Reconfiguration cost**: this is the cost to reconfigure the current robot mental state, which amounts to activate new processes $\pi_{c_1}, \ldots, \pi_{c_m}$. More precisely, the reconfiguration cost is about the effort needed to infer all the transitions axioms and all the precondition axioms that are required to make the processes $\pi_{c_1}, \ldots, \pi_{c_m}$ executable by the robot at the current state.

**Interference cost**: this is the cost needed to terminate the processes $\pi_{c'_1}, \ldots, \pi_{c'_k}$, specifying the current mental state.

For example, suppose the robot mental state includes a number of active processes, among which audio-interface, people-tracking, face-recognition, and attention. If the robot has a stimulus, such as WiFi disconnection, requiring to shift to another urgent goal, such as recover communication, some of the active processes need to be shut down, because they interfere with the new goal. On the other hand, new processes need to be activated, like for example the process of recovering from memory the previous path, together with its preconditions. Therefore the cost is here defined in terms of formulas of the language, that have to be inferred.

Let $\text{TERM}$ be the set of terms and let $\text{WFF}$ be the set of formulas of the language $\mathcal{L}$\textsuperscript{[72]}. The cost of a term $\eta \in \text{TERM}$ is inductively given by the function $\nu : \text{TERM} \to \mathbb{R}$, defined as follows:

$$\nu(\eta) = \begin{cases} 
0 & \text{if } \eta \text{ does not mention terms of sort action} \\
1 & \text{if } \eta = \text{start}_\pi(x, t) \\
1 & \text{if } \eta = \text{end}_\pi(x, t) \\
\sum_{i=1}^k \nu(a_i) & \text{if } \eta \text{ mentions } a_1, \ldots, a_k \text{ terms of sort action}
\end{cases} \quad (8.22)$$

Here $t$ is a term of sort time and $x$ is a list of terms of any sort, with the exception of sort situation. The function $\text{cost} : \text{WFF} \to \mathbb{R}$ returning the cost of a formula $\psi \in \text{WFF}$ can be inductively defined, extending the function $\nu : \text{TERM} \to \mathbb{R}$ over terms of $\mathcal{L}$ as follows. Let $\overline{\psi} \equiv Q_1 x_1 \ldots Q_n x_n [\phi_1 \lor \ldots \lor \phi_n]$ be a formula in prenex disjunctive normal form and equivalent to $\psi$. Here the quantifier string $Q_1 x_1 \ldots Q_n x_n$ is the prefix and $\phi_1 \lor \ldots \lor \phi_n$ is the matrix of the prenex form. Each sub-formula $\phi_i$ of $\overline{\psi}$ is a conjunction $L_1(x_1) \land \ldots \land L_m(x_m)$ of literals of $\mathcal{L}$. A prime implicant $\gamma$ of the formula $\overline{\psi}$ is a conjunction of literals such that $\gamma \models \overline{\psi}$ and there exists no other implicant $\gamma'$ such that $\gamma \models \gamma'$. Let $PI$ be the set of prime implicants of $\overline{\psi}$. Then, the cost of $\psi \in \text{WFF}$ is inductively computed as follows:
8. Learning the dynamic processes of shifting and inhibition in cognitive control

\[ \text{cost}(\psi) = \begin{cases} 
1 + \nu(x) & \text{if } \psi(x) \text{ is a literal} \\
\prod_{j=1}^J \text{cost} \left( L_{ij}(x_{ij}) \right) & \text{if } \psi \equiv Q_1 x_1 \ldots Q_j x_j \left[ L_{1j}(x_1) \land \ldots \land L_{kj}(x_k) \right] \\
\min_{\gamma \in PL} \{ \text{cost} (\gamma) \} & \text{if } \psi \in WFF \text{ and} \\
\end{cases} \]

Here each \( x_{ij} \) is a term of \( \mathcal{L} \) of any sort. Note that, under the formulation of the above function, it follows that \( \text{cost}(\emptyset) = 0 \) and \( \text{cost}(\mathcal{L}) = \infty \) (by considering the infinite set of literals in \( \mathcal{L} \)). Therefore, if \( \psi \equiv \top \), then the cost of \( \psi \) is the cost of the empty set of prime implicants, that is \( 0 \). On the other hand, if \( \psi \equiv \bot \), then the cost of \( \psi \) is the cost of the language \( \mathcal{L} \), that is \( \infty \) [204]. Given a set of formulas \( \Psi \), the cost of \( \Psi \) is the sum of the costs of each \( \psi \in \Psi \).

Having defined the cost of a formula of the language and of a set of formulas, we can introduce the reconfiguration cost of a mental state, requiring to compute the cost of the transition axioms, of the precondition axioms and of the constraints for all the processes that need to be activated to build a new mental state. We are given the following set of formulas postulating properties of processes. The transition axioms abbreviated as \( \pi_C(x, u, t, \text{do}(a, s)) \equiv \mathcal{A}_\text{trans}(x, u, t, a, s) \); the precondition axioms, abbreviated as \( \text{Poss}(a(x, t), s) \equiv \mathcal{A}_\text{pre}(a(x, t), s) \), and the constraints, abbreviated as \( \pi_C \cap \pi_C' \equiv \mathcal{A}_\phi(\pi_C, \pi_C') \), with \( \mathcal{A}_\phi \) the set of constraints. Let \( \Phi, \Psi \) be set of formulas of the language needed to infer the above indicated transitions, preconditions and the constraints (according to eq. (8.21)) above. The cost of reconfiguring a new mental state, at time \( t \) and situation \( s \), is defined by:

\[ \text{cost}_{\text{rec}}(\Phi) = \frac{1}{m} \sum_{i=1}^m \text{cost} (\mathcal{A}_{i}(x_i, t, a_i, s)) + \frac{1}{n} \sum_{j=1}^n \text{cost} (\mathcal{A}_{i}(x_i, t, s)) + \frac{1}{|\pi_C|} \sum_{b \in \pi_C} \sum_{i=1}^k \text{cost} (\mathcal{A}_\phi(\pi_C, \pi_C')) \]  

(8.24)

On the other hand, given a set of end actions \( Q = \{ \text{end}_C_1, \ldots, \text{end}_C_k \} \), terminating a set of active processes, together with the corresponding set of formulas \( \mathcal{A}_\text{pre} \), defining their preconditions, the interference cost of the set of formulas \( \Psi \) added to the theory in order to infer them is given by:

\[ \text{cost}_{\text{interf}}(\Psi) = \frac{1}{|Q|} \sum_{\text{end}_C \in Q} \nu(\text{end}_C) + \frac{1}{|Q|} \sum_{\text{end}_C \in Q} \text{cost} (\mathcal{A}_{\text{pre}}(\text{end}_C, s)) \]  

(8.25)

Note that we are treating the cost of the formulas that need to be added to the action theory \( \mathcal{A} \) in order to infer some fact, we are not treating the inference of the formulas to be added, this is specified in eq. (8.21). The cost of formulas are necessary to support the decision on the stimulus-response model treated in Paragraph [8.5] where they are used to resolve whether to shift to a new goal, in so responding to the stimuli, or to inhibit the stimuli.

Examples of processes as given above are reported in [81] [38] [79] [209] [204], in the framework of the situation calculus [207]. However, processes are defined in several other frameworks such as for example in STRIPS, Graphplan and PDDL [20] [168] [82] [75] each one with its own advantages in terms of expressive power, computational feasibility, and
8.5 Task Selection

The stimulus-response model chooses a set of task (mental) states as response to the incoming stimulus occurred within the time interval $\Delta t$, which should be activated (reconfiguration) if a stimulus occurs, according to the constraints imposed by the current mental state (interference). In this paragraph we address the problem of defining a decision rule whereby the robot can choose to shift its activities to one of the selected task states or to inhibit the stimulus, keeping its current state. The decision rule is based on both the reconfiguration and the interference costs, defined in Paragraph 8.4.

The basic idea behind the decision rule is to be conservative with respect to the current task unless either 1) a relevant stimulus is triggered, or 2) the current task is just initiated or has a very low cost, meaning that the interference of the stimulus and the response is not a big cost issue. This concept can be further extended to capture more circumstances but this will not be done here.

Let $\tau_{\text{curr}}(\Delta t)$ be the current task state (see eq. 8.20). By the definition of cost of a formula (see eq. 8.23), we are interested only on the action terms. This implies that the longer the time the robot is involved in the current task state, the longer is the history of actions that are counted to get the interference cost, and therefore the interference cost would become quite high. Furthermore, the longer the task state is being carried on, the more processes hold in parallel in the task state, therefore there are many processes to be shut down to reconfigure the current state into a new task state. On the other hand for any task state $\tau$ in the set $\gamma(y_{\pi C})$ the reconfiguration cost is about the cost of the preconditions for each start action for each process $\pi_C$ in $\tau$, the preconditions for the transitions of $\pi_C$ and the time and space constraints $C$ linking the processes.

We want to model a decision rule that suitably interprets the conservative idea expressed above. To this end we introduce a decision plane, which is divided into regions: the shifting regions and the inhibition regions. The decision plane axes are the interference cost ($x$-axis) and the reconfiguration cost ($y$-axis), noting that the interference cost is related to the current task $\tau_{\text{curr}}(\Delta t)$ while the reconfiguration cost is related to each task $\tau \in \gamma(y_{\pi C})$. Note that we say related because as shown in Paragraph 8.4 the real cost is about the formulas that are added to the theory (see eq. 8.23) to infer how to terminate the present task and how to activate the processes of the new task, that is, the interference and the reconfiguration.

The decision rule looks for the task $\tau$ in $\gamma(y_{\pi C})$ for which there exist a set of formulas $\Phi$, with cost $\text{cost}_{\text{rec}}$, and a set $\Psi$, with cost $\text{cost}_{\text{interf}}$, whose cost is minimal. We thus introduce two functions $g_{\text{cost}}$ and $g_{\text{cost}}$ which take the present task state $\tau_{\text{curr}}(\Delta t)$ and a task $\tau$ in $\gamma(y_{\pi C})$ and returns the interference and reconfiguration costs of the set of formulas $\Psi$ and $\Phi$. 


constraints expressible in the language. Where robot planning requires to orchestrate all the components, according to temporal constraints, the planning system usually takes advantage of temporal networks to manage temporal constraints as in \[2, 58, 257, 19, 177]. Here we have used the Situation Calculus to model the robot high level control of the processes because of its expressive power both for managing parallel processes, time-space constraints and for defining the interference and reconfiguration cost. The stimuli and stimulus-response model proposed in this work could be built on any action theory handling robot components, processes and constraints and able to express the robot mental state at a precise time, together with all the cause-effect relations that are affected by the response.
so that we can draw points on the decision plane, namely:

\[
\Gamma_{\text{cost}}(\tau_{\text{curr}}(\Delta t), y_{\pi_C}) = \{ p_i = (\text{cost}_{\text{rec}}^i, \text{cost}_{\text{interf}}^i) | \text{cost}_{\text{rec}}^i = \text{cost}_{\text{rec}}(\tau_{\text{curr}}(\Delta t), \tau_i) \text{ and } \text{cost}_{\text{interf}}^i = \text{cost}_{\text{interf}}(\Psi) = \text{cost}_I(\tau_{\text{curr}}(\Delta t), \tau_i) \}
\]

(8.26)

On the other hand we have a function \( g_{\text{dec}} \) which, given one of the computed points \( p_i \), on the decision plane, returns the original task namely, \( g_{\text{dec}}(p_i) = g_{\text{dec}}(\text{cost}_{\text{rec}}^i, \text{cost}_{\text{interf}}^i) = g_{\text{dec}}(\text{cost}_{\text{rec}}(\tau_{\text{curr}}(\Delta t), \tau_i), \text{cost}_I(\tau_{\text{curr}}(\Delta t), \tau_i)) = \tau_i \). Now, given a point \( p_i \) on the decision plane, if \( p_i \) is in the shifting region, and the choice falls on this \( p_i \), then the decision will be that of shifting to \( g_{\text{dec}}(p_i) = \tau_i \). On the other hand if the choice falls on a point lying an an inhibition region, the decision would be to inhibit the stimulus \( y_{\pi_C} \). Therefore we only need to explain how these regions are determined and how the choice is made. The decision plane is illustrated in Figure 8.5. The basic idea of these regions, being the decision principle conservative, is that the regions where the interference cost is increasing are inhibition regions, likewise regions where the expected weight of the stimulus is low are inhibition regions, all other regions are shifting regions.

To model this fact we introduce a weight on the stimuli and a way to make stimuli affect the position of a point to create boundary for these conditions. For weighting stimuli we simply consider a priori weights, here denoted \( w_{\pi_C}, w_{\pi_C} \geq 1 \), defined by a rule of thumb on each stimulus urgency. To define the stimuli boundary we introduce two rectangular hyperbolas: \( y = c^2/x \) and \( y = -c^2/x \) with asymptotes the x-axis and the y-axis, with \( c \geq 0 \) a threshold for urgent stimuli. The two hyperbolas specify the boundary of regions that are always shifting regions.

The meaning of these constraints and the resulting decision rule are specified by the following algorithm. Let \( \mathcal{D} \) be the decision plane and \( \Gamma_{\text{cost}}(\tau_{\text{curr}}(\Delta t), y_{\pi_C}) \) be the set of points obtained by the current task and the tasks inferred by the stimulus-response model defined in eq. (8.26). We distribute the points on the plane as follows. Let \( p \in \Gamma_{\text{cost}}(\tau_{\text{curr}}(\Delta t), y_{\pi_C}) \) we transform \( p \) into \( p' = (\text{cost}_{\text{interf}}^i, \text{cost}_{\text{rec}}^i) \), with \( \text{cost}_{\text{rec}}^i = \pm \frac{1}{w_{\pi_C}} \sqrt{\text{cost}_{\text{rec}}} + \epsilon \) and \( \text{cost}_{\text{interf}}^i = \pm \sqrt{\text{cost}_{\text{interf}}} + \epsilon \), to weight the reconfiguration cost according to the stimulus urgency. Note
that \( \varepsilon \) is normally distributed with mean \([0,0]^{\top}\) and variance \( \theta \) defined as follows:

\[
\theta = \begin{cases} 
\begin{bmatrix} 2\pi/3 & 0 \\ 0 & 2\pi/3 \end{bmatrix} & \text{if } cost_{\text{interf}}' > cost_{\text{rec}}' \\
\text{The identity matrix } I & \text{otherwise}
\end{cases}
\]  

The decision rule is then established by the following algorithm:

1. Let \( p' = (cost_{\text{interf}}', cost_{\text{rec}}') \) be a point on the decision plane \( \mathcal{D} \) and let \( g'_{\text{dec}} \) be like \( g_{\text{dec}} \) extended to cope with the transformation of \( p \) to \( p' \).

   (a) if \( -\sqrt{2}c \leq cost_{\text{rec}}' \leq \sqrt{2}c \) then decide for shifting, independently of the interference cost. Choose the task state \( g'_{\text{dec}}(p') \) that minimizes \( |cost_{\text{interf}}'| \).

   (b) if \( -\sqrt{2}c \leq cost_{\text{interf}}' \leq \sqrt{2}c \) then decide for shifting, independently of the reconfiguration cost. Choose the task state \( g'_{\text{dec}}(p') \) that minimizes \( |cost_{\text{rec}}'| \).

2. Otherwise, if no point lies in the region between the two hyperbolas, let \( r = (cost_{\text{interf}}'^2 + cost_{\text{rec}}'^2)^{1/2} \) and choose the point \( p' \) that minimizes \( r \). If the point is within a shifting region then shift for the task \( g'_{\text{dec}}(p') \), otherwise inhibit the stimulus \( y_{\pi C} \) and continue with task state \( \tau(\Delta t) \).

Figure 8.5, right, shows how points generated from different stimulus-response set \( \gamma(y_{\pi C}) \) with \( y_{\pi C} \) varying, are distributed on the decision plane. This simple rule allows, indeed, to deal with several circumstances, though the weights, for specifying the degree of urgency a stimulus elicits, have to be established a priori, so far. Experiments in Paragraph 8.7.5 incorporate results concerning the modeled decision rule.

### 8.6 Implementation

#### 8.6.1 Robot functionalities

The robot is provided with a real-time 3D ICP-based simultaneous localization and mapping (SLAM) system [211]. The robot is endowed with a Dead Reckoning Navigation System (DRNS), based on a complementary filter (CF), estimating the Euler angles of the body of the robot from data fusion of the odometry and the IMU inertial data [249, 33, 147]. Both ICP-based SLAM and DRNS provide, at real-time, the pose of the robot as a feedback to the Trajectory Tracking Controller (TTC) of the robot [93]. Static traversability cost assessment of the ground, from point cloud data, is performed by both a physics-based terrain [245, 194] and physical constraints analysis [31]. Visual perception of the environment is performed by object (e.g. signs, cars and persons) detection and localization through the omnidirectional camera [124].

The robot control is endowed with a declarative temporal model of the controllable activities and a planning engine. The declarative temporal model is specified in the Temporal Flexible Situation Calculus (TFSC) [79, 38] and explicitly represents the main components and processes of the robot system (e.g. SLAM, vision, differential, flippers), the cause-effect relationships as well as the temporal constraints among the processes [98]. The planning engine is composed of two main logical modules: the plan generator and the execution monitoring. The plan generator relies on a library of Prolog scripts designating the set of
tasks which the robot can perform, according to the specified processes, their temporal constraints (compatibilities), and preconditions. The execution-monitoring is a continuous process which ensures that the set of action sequences, generated by the plan generator according to the TFSC model and the current state of the domain knowledge, are consistently executed. Concurrently, at regular time intervals, the execution-monitoring reads the system state and monitors the execution of the activities, in order to detect system malfunctioning that may result in action failures. Furthermore the execution-monitoring integrates the model of task switching to flexibly regulate the behaviour of the robot in response to the incoming stimuli. Both the TFSC model and the planning engine are implemented in Eclipse Prolog, which optimally combines the power of a constraint solver (for the time and compatibility constraints) with inference in order to generate the set of action sequences, and also enable the continuous update due to incoming new knowledge (using finite progression).

An hybrid CAST subarchitecture has been deployed for the robotic system in order to fully embed the TFSC model, the planning engine and the model of task switching [117]. The subarchitecture interconnects the planning engine with the ROS Operating System (ROS) which is responsible of managing, at lower level, the activation of the processes on the robotic platform [217] (see Fig. 8.6). Moreover, the CAST subarchitecture for the planning allows the human operator to switch between several operational modalities lying between autonomous and teleoperated modes during the execution of a task. It provides communication interfaces with the human in both the directions: from the human to the robot and vice versa [142][151].

8.6.2 Task switching interactive interface

An interactive interface has been designed with the main purpose of building data set for training and validating the proposed model of task switching (see Figure 8.7). The layout of the main window of the interface comprises of the following four main group frames: (1) the stimuli frame, (2) the robot mental state frame, (3) the task state frame and, (4) the control frame. The stimuli frame includes a set of vertical tabs, each of which is associated with a robot component. Each tabs is composed of a set of frames. Each frame is related to a component process. This frame displays, over time, the trend of the function representing
Figure 8.7. Task switching interactive interface

the process as well as the process yields. An additional button is included in the frame of every component process to allow a human operator to manually notify the occurrence of a stimulus, simply observing the process yields. The human operator can individually disable a frame associated to a component process through a check-box button. By selecting a tab into the stimuli frame a human operator can observe the state of each robot component as well as the yields of the processes associated with that component. The robot mental state frame is composed of both a text box reporting the task which the robot is currently accomplishing and a table, listing all the running processes, required to perform that task. The task state frame is provided with a set of buttons, each of which is associated with a robot task. Through this frame, the operator can instruct the robot with the tasks which it has to perform. The control panel provides the human operator with two horizontal tabs: (1) robot steering tab and, (2) flipper configuration tab. The first tab is endowed with a set of control commands for steering the robot. The latter comprises a set of control buttons to change the configuration of the flippers and lock/unlock the differential drive. This frame enables the operator to manually control the locomotion system of the robotic platform. In addition, the main window reports both the reconfiguration and interference costs, computed with respect to the current task and with respect to each different task in the robot task library. Furthermore, the interface is endowed with three radio buttons setting the interface for three different user interaction modalities: (1) passive mode, (2) active mode and, (3) learning mode. In the first interaction modality, the interface can be used, by the operator, to monitor, at-real time, the state of the robot components and the processes running on the robot. This mode allows the operator to manually notify the occurrence of the stimuli, by observing the yields carried by the processes. In the active interaction modality, the interface displays the data yielded by process and, in addition, notifies, at-real time, the operator of the occurrence of a stimulus. This mode enables the operator to explicitly interact with
the robot, by selecting and executing the task which the robot has to perform, according to
the stimuli that have been notified, or by ignoring the stimuli in so allowing the robot to
accomplish the previous task. The latter interactive modality is suited for collecting data
needed to assess the behaviour of the robot with respect to the behaviour of the human
operator. In this mode, the interface notifies the human operator of both the occurrence
of the stimuli and the choices taken by the cognitive control of the robot to respond either
shifting to a new task or continuing the execution of the task at hands (inhibiting the stimuli).
The interface allows the human operator to simultaneously select the tasks which he/she has
choosen to respond to that stimuli as well as the inhibition choices, without affecting the real
choices of the cognitive control.

8.7 Experiments

In this paragraph we present the experiments performed to evaluate (1) the ability of the
stimuli model to discard uninformative samples from a data-set of features-stimulus pairs;
(2) the performance of the stimuli model (Paragraph 8.2); (3) the generalization ability
of the recommendation system and (4) its correctness (Paragraph 8.3) and, finally, (5) the
effectiveness of the rule-based task selection model assessing the behavior of the robot, with
respect to the behavior of a human operator (Paragraph 8.5). The experiments have been
carried out into a suitably designed environment, simulating a real partially destroyed area
(see Fig. 8.8(a)). Further to this setting, a set of detectable objects together with a set of
obstacles have been evenly distributed in the area. Some of the obstacles can be overcome
by the robot (see Fig. 8.8(b)). The following stimuli have been selected to be reproduced
within the scenario: (a) the power level of the battery, (b) the WiFi signal quality, (c) write
access to full memory and, (d) read access to missing data.

8.7.1 Outliers rejection evaluation

The ability of the stimuli model of discarding uninformative samples from a data-set of
features-stimulus pairs is measured with respect to the ratio between the number of samples
of the data-set, effectively selected to estimate the parameters $\beta$ of the model (see eq. (8.7),
Paragraph 8.2), and the number of samples of the data-set, manually labeled by the operators.
The evaluation is carried out by varying the statistical significant level $\alpha \in [0, 1]$, eq. (8.4),
affecting the estimation of the confident region, eq. (8.3), used to discard the outliers from a
data-set. The data-set has been collected from two active processes, monitoring the WiFi

Figure 8.8. (a) Simulated disaster area, (b) Obstacles located into the disaster area, (c) Operator
interacting with the robot through the task switching interface.
8.7 Experiments

Figure 8.9. Evaluation of the ability of the stimuli model to discard uninformative samples from a data-set of features-stimulus pairs with respect to the parameter $\alpha$, affecting the confident region in which trials are considered good for classification for stimuli on (a) WiFi and, (b) Memory.

Figure 8.10. Performance of the stimuli model: ROC curve of the true and false positives, as the time interval $\Delta t$ changes.

signal quality and checking the state of the memory, as follows. 10 operators have been selected to perform several short-term missions with the robot. Each operator started the mission by instructing the robot to explore the area. The operators were provided with the task switching interface, under the passive mode, to manually label the yields of the running processes as stimuli (see Fig. 8.8(c)). Each short-term mission terminated when the operator notified the occurrence of a stimulus, either on the WiFi signal quality or on the memory state. Only one stimulus of the two can be advised by the operator, for each short-term mission. Figure 8.9(a) and Figure 8.9(b) show the ratio between the number of accepted samples and the number of gathered samples, as $\alpha \in [0, 1]$ changes. The devised values $\alpha_{\text{WiFi}} = 0.3$ and $\alpha_{\text{Mem}} = 0.9$ are, indeed, considered the reference values for the statistical significance level of the confident region of the WiFi signal and of the confident region of the memory state, respectively.
8. Learning the dynamic processes of shifting and inhibition in cognitive control

Figure 8.11. Learning curve of the recommendation system under two different feature space dimensions (a) $d = 4$ and, (b) $d = 10$, derived from a k-fold cross-validation technique.

8.7.2 Performance of the stimuli model
The performance of the stimuli model is evaluated by analysing the ROC curve of both the true and false stimuli, as the size of the time interval $\Delta t$ changes, within which the occurrence of the stimuli has been considered. The stimuli model has been trained with the data-set experimentally inferred from the previous experiment, tough filtered with respect to the reference values of $\alpha$ parameter. The performance of the model has been measured on a test-set, collected as follows. 5 operators have been instructed to perform a patrolling mission with the robot, lasting 10 min. As in the previous experiment, the operators manually labeled the yields of the running processes, upon the occurrence of a stimulus. The yields, labeled by the operators, have been compared with the results of the classification of the stimuli provided by the model (Paragraph 8.2). Both correct classifications and mismatches have been counted by considering the time interval $\Delta t$, centred at the time in which the operator notified the occurrences of the stimuli, and by varying the size of $\Delta t$. Figure 8.10 shows the ROC curve of the stimuli model, with respect to the size of $\Delta t$. The trend reports the percentage of the true positives vs. the percentage of the false positives, for every considered time interval. Then, $\Delta t = [-2, 2]$ can be taken as the reference time interval to evaluate the correctness of the stimuli-response model, due to the trade-off between the false and the true positives.

8.7.3 Generalization ability of the recommendation system
The generation ability of the recommendation system (RS) has been measured by analysing the convergence of the RS Mean Square Error (MSE), in both the training and the validation phases, with respect to two different values of the feature space dimension $d$ (Paragraph 8.3). The data-set of stimulus-task pairs, for both training and validating the model, has been collected as follows. 20 operators have been asked to tele-operate the robot into the designated environment, looking for all the objects and labeling the map with their location. The experiment ended when each operator found all the objects placed into the environment. During each experiment, several instances of the predefined stimuli have been simulated at different time lapses. Given the stimuli detected by the learned stimuli model, the operators selected the task the robot had to perform, via the task switching interface, set
8.7 Experiments

8.7.4 Correctness of the stimuli-response model

The correctness of the stimuli-response model is evaluated by analysing the ROC curve of both true and false switches, as the threshold $\kappa \in [0, 1]$ in eq. (8.14), Paragraph 8.3, changes. Both true and false switches can be computed according to the following criteria. Given the occurrence of a stimulus, the model assigns a ranking $a_{ij}$ to each task $\tau_i \in T$. As the threshold $\kappa$ changes, the subset of the set of tasks states $T$, returned by the recommendation system, varies too. If the task selected by the operator, responding to the stimuli, is included into the subset of best task choices, then the response of the recommendation system is considered as a true switch. The following experiments have been performed to evaluate the correctness of the stimuli-response model. 10 operators have been selected to explore with the robot the designated environment, finding all the objects and labeling the map with their location, as in the previous experiment. Operators were provided with the task switching interface, set to the learning mode. Without a real interaction with the robot, the operators selected either the task to allow the robot to shift from the one it was currently performing, or the same task in so inhibiting the stimuli, detected by the stimuli model. The gathered data have been compared with the results obtained by the recommendation system. Figure 8.12(a) depicts the ROC curve of the percentage of both true and false switches, as $\kappa$ changes. The trend shows that, for $\kappa = 0.8$, the model correctly selects those subsets of tasks which include the actual responses of the operators, with no mis-classifications. Figure 8.12(b) shows the accuracy ratio of the recommendation system, with respect to $\kappa$. The accuracy ratio is, as usual, defined as the ratio between the sum of true switches and true inhibitions, and the sum of actual switches and of actual inhibitions, performed by the
operators. As a result $\kappa = 0.8$ can be considered as the minimum value of the score that a
stimulus can assign to a task state $\tau$, in order to be included into the set of best task choices.

### 8.7.5 Effectiveness of the rule-based task selection model

The effectiveness of the rule-based task selection model can be measured in terms of both
correct and wrong switches decisions assessed by the rule, specified in Paragraph 8.5, with
respect to the decisions effectively taken by the operators. This evaluation has been per-
fomed by making vary the parameter $c \in [0, 2]$ affecting the boundaries of both the shifting
and the inhibition regions, on the decision plane $\mathcal{D}$ (see Fig. 8.5 in Paragraph 8.5). Without
loss of generality and without introducing biases in the experiments, data collected from
the previous experiments have been used to compare the outcomes of the rule underlying
the task selection model with the actual responses of the operators to the incoming stimuli.
Figure 8.13(a) shows that, for $c = 1.2$, the decision rule assesses all the correct switches. On
the other hand, having fixed $c = 1.2$, the rule takes 1.19% of wrong switches decisions, with
respect to the operators choices (see Fig. 8.13(b)). Therefore, $c = 1.2$ can be considered as
the reference value of the threshold, setting the boundaries of the decision regions of $\mathcal{D}$.

These experiments have been crucial to determine the minimum values of the param-
eters of the proposed model of task switching. Under this parameter setting, the model
demonstrates to achieve good performance in terms of both correctness and effectiveness.
Indeed, the experiments proved that the model well-regulates the behavior of the robot, by
identifying what is a stimuli, by recognizing when a stimulus is occurred and, finally by
flexibly adapting the robot behavior in so either selectively responding or inhibiting such
stimuli. The performance show that the proposed approach is very promising in assessing
the robot behavior with respect to the behaviour of a human operator.
Chapter 9

ARE: Augmented Reality Environments for mobile robots

Augmented Reality (AR) has recently stepped beyond the usual scope of applications like machine maintenance, military training and production. AR is a compelling technology allowing cognitive robotics developers to design a variety of complex scenarios involving real and virtual components. Indeed, programming complex robotic systems, operating in dynamic and unpredictable environments, is a challenging task, when it is required to go beyond laboratory experiments. Furthermore, as the complexity of the robotic system is inflated by an increase of robot components and functionalities, testing the whole set of interconnections between hardware and software components becomes exponentially difficult. The cognitive roboticists are more and more exposed to hard problems requiring a great amount of possible real world situations that are difficult to predict, when most of the test require laboratory experiments. In this context, Augmented Reality (AR) facilitates robot programming, providing to robot developers a technology for testing the robot system which is much more flexible than a complete simulation environment, as it allows developers to design a variety of scenarios by introducing any kind of virtual objects into real world experiments. AR-environments can facilitate experiments bypassing complex simulation models, expensive hardware setup and a highly controlled environment, in the various stages of a cognitive robot development.

In this chapter, we present an AR-based simulation framework which allows robot developers to build on-line an Augmented Reality Environment (ARE) for real robots, integrated into the visualization interface of Robot Operating System (ROS) [218]. The system we propose goes beyond an interface for drawing objects, as the design exploits a stochastic model activating the behaviors of the introduced objects. Objects, people, obstacles, and any kind of structures in the environment can be endowed with a behavior; furthermore, a degree of certainty of their existence and behaviors, with respect to what the robot perceives and knows about its space, can be tuned according to the experiment needs.

To illustrate the advantages of an AR based simulation framework for robot design and experiments, as opposed to a complete simulation framework in scenario design and test, and to show the benefits of our approach, we set up several path-planning experiments, with increasing complex environments. In particular we show that the framework allows to compare the performance of the path-planner with respect to several parameter sets in any simple outdoor settings augmented with ARE.
The chapter is organized as follows. In Paragraph 9.1.3 we describe the main components of the AR-based simulation framework, detailing on the dynamic model. Paragraph 9.2 describes the robot setup and reports the results obtained by a paradigmatic application of ARE for evaluating the capabilities of the robot in navigation tasks.

9.1 The AR-based simulation framework

The AR-based simulation framework registers virtual objects, such as robots, cars, people, pallets and other kind of obstacles into the real environment (see Figure 9.1). The model for the life-cycle of each virtual object is stochastic. It formally structures the interaction between the virtual objects and the real environment by both avoiding collisions and handling occlusions effects. The AR-based simulation framework includes: (1) the model of the real environment, (2) the model of the virtual objects, namely artefacts and, (3) the AR-builder server.

9.1.1 The real environment representation

The real environment representation concerns both the 2D occupancy grid map $M_{2D}$ and the octree-based 3D map $M_{3D}$ on the left of the panel in Figure 9.1. This representation has been proved to be compact and easy to update incrementally, thus accounting for dynamic obstacles and changing scenes. In fact, a polygonal mesh $S_E$ is used to geometrically represent the environment. The polygonal mesh renders, together with the basic environment structure, also the 3D models of the artefacts, so as to correctly place the artefacts in the real scene, thus handling the occlusions with the existing objects.

9.1.2 The Artefacts model

An artefact is a dynamic object defined by three main structures, illustrated in Figure 9.2 bottom-right of the panel: the properties, selected by a mixture of Poisson distributions, a life cycle specifying the arrival and leaving time and, finally, a Markov decision process specifying the artifact main behaviors, given the life cycle and the selected properties. We are given a space $Q$ of all possible tuples defining an artefact type; actually we can think of $Q$ as the set of all possible matrices $Q$ where each row specifies a particular tuple $Q$ of properties of an artefact $A$, with $Q = \{l, b, p(.,.t), q(.,.t), \Phi\}$. Here $l$ is a label denoting the virtual object type, $b$ is the bounding box of the artefact, and $p(.,.t)$ and $q(.,.t)$ denote the position and orientation operators of the artefact, taking temporal values in the stochastic Poisson model, described in the next sub-paragraph. $\Phi$ is a set of additional
9.1 The AR-based simulation framework

Figure 9.2. On the left top the ROS message specifying an instance of an artefact. On the left, bottom, a Poisson Mixture of 5 components with parameters vector $(15, 7, 3, 21, 5)^\top$. On the right the artefact structure schema.

features. Each artefact is geometrically specified by a polygonal mesh $\mathcal{S}_A$, representing the virtual object as a set of faces and a set of vertices, according to the face-vertex model, along with additional information such as color, normal vector and texture coordinates. We assume that the geometry of the mesh data structure is not morphable and warpable so that the bounding box is fixed for its life-cycle. This representation allows the AR-builder server to manage the collisions between the artefacts populating the real environment. Within the framework, an artefact is implemented by a ROS message, see Figure 9.2, top-left panel.

In this sub-paragraph we introduce the simulation model regulating the spatio-temporal behavior of the artefacts populating the real environment. Namely we introduce both how properties are sampled in $\mathcal{Q}$ and the arrival and leaving time of the artefacts. This model is based on a marked Poisson process $\{77, 223, 255\}$, whose marks correspond to an identifier selecting a tuple from a matrix $Q \in \mathcal{Q}$ and the artefact life-cycle. A marked Poisson process is a Poisson Process which has each point labeled with a mark. These marks are used to connect the artefact properties with its specific life-time cycle. Indeed, the model estimates the arrival and exit time of the artefacts as well as the labels governing the choice of the artefact properties.

Let the variable $X$, selecting a tuple $Q$ in the space $\mathcal{Q}$, be a stochastic variable sampled from a mixture of $n$ Poisson distributions, with $n$ the number of object classes, namely:

$$Pr(X = k|\lambda_1, \ldots, \lambda_n) = \sum_{i=1}^{n} \pi_i \frac{\lambda_i^k}{k!} \exp(-\lambda_i)$$

(9.1)

Here $\sum \pi_i = 1$, $\lambda_i$, $i = 1, \ldots, n$ are the Poisson distribution parameters for each mixture component. Note that since $n$ is given a priori from the classes of buildable objects (e.g. cars, pallets, girls, boys,..) parameters estimation is set by the EM $[60]$. The choice of the tuples is specified upon arrival of a group of artefacts. This is detailed below.

Let $M_{2D} \subseteq \mathbb{R}^2$ be the occupancy grid designating the flat structure of a generic environment taken as base of the representation (e.g. the courtyard of the department, a set of corridors). The arrival time $t_a$ of artefact $a$, in $M_{2D}$ can be assigned according to a Poisson process. Here arrival time $t \in [0, T]$ has an average rate $\lambda \in \mathbb{R}$. The times between successive arrivals are independent exponential variables having mean $\frac{1}{\lambda}$. Upon arrival, an artefact is supposed to leave $M_{2D}$ after a certain time. Similarly to arrival time, the time last of each
artefact is an independent exponential variable with mean $\frac{1}{\mu}$, where $\mu \in \mathbb{R}$ is the average rate artefact leaving time. From these assumptions it follows that arrivals and leavings time $t$ are ruled by a time-homogeneous irreducible, continuous-time Markov chain $\{N(t) | t \geq 0\}$ with state space $\mathbb{N}^+$. Let $v(M_{2D})$ be the area of $M_{2D}$, let $o(\Delta t)$ be an infinitesimal of a higher order than $\Delta t^1$, then the stationary transition probabilities $p_{ij}(\Delta t)$ between state $i$ and state $j$, for infinitesimal lapses of time $\Delta t$, are given by

$$p_{ij}(\Delta t) := P(N(t + \Delta t) = j | N(t) = i) = \begin{cases} \lambda v(M_{2D}) \Delta t + o(\Delta t) & \text{if } j = i + 1 \\ 1 - (\lambda v(M_{2D}) + \mu) \Delta t + o(\Delta t) & \text{if } j = i \\ \mu \Delta t + o(\Delta t) & \text{if } j = i - 1 \\ o(\Delta t) & \text{if } |j - i| > 1 \end{cases}$$

$\{N(t) | t \geq 0\}$ can be seen as a special case of a birth-death process \[106\], with birth rates $\lambda_i = \lambda v(M_{2D})$ and death rates $\mu_i = \mu$, for each $i \in \mathbb{N}^+$. Let the arrivals of new artefacts in $M_{2D}$ occur according to a Poisson process with intensity $\lambda v(M_{2D})$ and let, upon arrival, the lifetime of each artefact be independent and exponentially distributed with mean $\frac{1}{\mu}$, then $N(t)$ returns the number of artefacts alive at time $t$ (see Figure 9.3(a)).

Given the life-cycle, artefacts populate the environment and, given the Poisson mixture, upon arrival they draw their specific properties, including features such as being a car or being a pallet, from the distribution. The final component of the artefact structure is its characteristic behavior determined by a motion planning algorithm that would govern what it can actually do in the scene. To model the specific chosen behavior a finite-horizon Markov Decision Process is introduced:

$$\mathcal{H}_A(t) = \{D, S, \mathcal{A}, p_t(\cdot | s, \alpha), r_t(\cdot | s, \alpha) : t \in D, s \in S, \alpha \in \mathcal{A}_s\}$$

Here $D = \{0, \ldots, d\}$ is the set of decision epochs, depending on the life time $d$ of the artefact, $S = M_{2D} \times \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi\}$ is the set of poses that it can take on during its life cycle, $\mathcal{A}_s$ is the set of possible actions when the artefact is in the state $s$, $p_t(\cdot | s, \alpha)$ is the transition

---

1 A function $f(t)$ is an infinitesimal of a higher order than $t$, namely $o(t)$, if $\lim_{t \to 0} \frac{f(t)}{t} = 0$. 

---

**Figure 9.3.** (a) Simulation of the waiting times as well as of the trajectory of the Poisson process, regulating the arrivals of the artefact to $M_{2D}$ (b) Simulation of the marked Poisson process, regulating the distribution in space of the artefacts, with the corresponding arrival time.
probability from state $s$ at time $t$ to the next state $s'$ at time $t'$ when action $\alpha$ is selected at the decision epoch $t$, and $r_t(\cdot | s, \alpha)$ is the reward earned when the artefact is in state $s$ and action $\alpha$ is selected at decision epoch $t$. Actions are choices about both behaviors and motion planning. In other words an action $\alpha$ is a function $\alpha : B \times \Pi \times S \mapsto (k_1, \ldots, k_m)^T$ mapping a set of predefined behaviors $B$, the set of motion planning algorithms $\Pi$, supporting the behaviors, and the set of states $S$, into a vector of numbers accounting for the choice. Behaviors specify what the artefact can do within the environment, i.e. walking, running, following another artefact, stopping, according to the identifier chosen with the mixture of Poisson distributions. Similarly, the motion planning algorithms $\Pi$ take into account the current positions of the artefacts already in $M_{2D}$ \cite{180} and the gridmap to consistently move the object in the scene. Note that the finite-horizon Markov decision process $H_A(t)$ selects the motion actions up to the time horizon $d$, according to the underlying action policy. Given the position of the other artefacts within $M_{2D}$, the motion planning $\Pi$ plans collision-free short trajectories to reach the new position and orientation of the artefact, at time $t'$, provided $H_A(t)$ has selected a set of behaviour $b_k \in B$.

### 9.1.3 The AR-builder server

The AR-builder server interconnects the real environment model together with the simulation model of the artefacts. The AR-builder server relies on the \texttt{tf} software library. Namely, it keeps track, over time, of the transformations between the global reference frame $\mathcal{F}_E$, attached to the real environment model, the base frame $\mathcal{F}_R$, attached to the robot base, the base laser frame $\mathcal{F}_L$, attached to the laser sensor, and the camera frame $\mathcal{F}_C$, associated with the camera system, mounted on the robot. Given the \texttt{tf} library, the AR-builder server can determine the current pose of the reference frame $\mathcal{F}_A$ attached to the center of mass of the artefact and map it into the frame $\mathcal{F}_E$, rather than $\mathcal{F}_L$ or $\mathcal{F}_C$. Upon the registration of the artefact, the AR-builder server correctly places it within the real environment, by both projecting the bounding box $b$ of the artefact on $M_{2D}$ and concatenating the vertexes of the polygonal mesh $\mathcal{M}_A$ to the voxels of $M_{3D}$ (see Figure 9.4(a)). On the basis of these computational steps, the AR-builder server constructs the augmented model of the real environment (see Figure 9.4(b)).

The AR-builder server implements a collision detection algorithm according to the dynamic model $H_A(t)$. The algorithm, performing pairwise hit-testing, determines whether the bounding box of an artefact intersects the bounding box of another one or, alternatively, whether an artefact becomes embedded in the polygonal mesh $\mathcal{F}_E$ of the real environment. In such a case, the builder simply resolves the collisions by either moving back the artefact to its last known safe pose or by allowing the artefact to move up to a safe distance. The model of the augmented real environment comprises only the artefacts which are not occluded, with respect to the robot field of view. These artefacts can be effectively ray-traced out from $M_{2D}$, thus not affecting the robot path-planning. The AR-builder server checks occlusion effects by implementing a ray tracing version of the z-buffer algorithm. Let $\mathcal{F}_{\text{view}} = \{ P, \{W, d_{\text{max}}\}\}$ be the view model of the real robot, where $P$ is the camera matrix of the real robot with center $C$ and $W$ is the polyhedral cone of the field of view, such that $n(h^{(i)}) \times m(h^{(i)})$ is the dimension of the viewing plane at distance $h$, with $h_{\text{max}}$ the maximum viewing ray length. We recall that the points $X$ on the ray joining a point $X$ and its projection on the image plane is $x = PX$, where $X$ is actually a ray of points, that is why along this ray only the free objects are seen by the robots while the others are occluded. Finally the vector in the direction of
the principal axis is defined by  \( \det(M) \) \( m^3 \), with  \( P = [M] p_4 \) [115]. To consistently deal with occlusions, an acclusion matrix is designed as follows. Let  \( d_{ij}^0, d_{ij}^1, \ldots, d_{ij}^n \) be the parameter values indicating where intersections with the implicit surface of the polygonal mesh \( S_{3D} \) of the real environment occur; let  \( d_{ij}^k \) be the first positive root, for each pixel \((i, j)\). We define a matrix  \( Z_{S_{3D}} \in \mathbb{R}^2 \) such that

\[
z_{ij} = \begin{cases} 
d_{ij}^k & \text{if } d_{ij}^k \in [0, d_{\text{max}}], \\
\infty & \text{otherwise}
\end{cases}
\]

Likewise, for each artefact \( a \) a matrix  \( Z_a \), mentioning the first positive roots  \( d_{ij}^k \in [0, d_{\text{max}}] \) is obtained by computing, for each pixel, the intersection of the corresponding ray with the bounding box \( b \) of the artefact. The set of artefacts perceived by the real robot is, therefore, given as follows:

\[
\mathcal{A}_f = \{ a | \text{count}((Z_a - Z_{S_{3D}})_{ij} \leq 0) > \tau \}
\]

Here the function count(·) returns the number of the elements \((i, j)\) which satisfy the above condition, and \( \tau \) is a threshold for assessing partial occlusions.

### 9.2 Experimental results

In this paragraph we illustrate the applicability of ARE to robot development and evaluation.

We embedded the AR-based simulation framework into a ROS package. We deployed the robotic platform, in Fig. 2.4 Chapter 2 in a wide outdoor area, and set up two experiments, where ARE has been used to populate the real surroundings with artefacts.

In the first experiment, we wanted to check the robot ability to replan the path towards a goal location, as the frequency of the arrivals of the artefacts into the environment changes. Different parameter settings of the path-planner have also been settled, further affecting the robot behavior into the navigation task (see Figure 9.5).
9.2 Experimental results

Figure 9.5. Parameter names and range (note: x-scale is logarithmic)

Figure 9.6. (a) The graph illustrates on the Y-axis the value of eq. (9.3), against scene update frequency. This shows how the parameter settings of the path-planner affect, in terms of time, the ability of the robot to replan the path towards the goal location. (b) Here the graph illustrates the number of goal that can be achieved with increasing mission time, and given an increasing spatial complexity, with thresholds indicating the parameters limits. Mission time is specified in minutes, spatial complexity is an index, as indicated in eq. (9.4).

During the experiment the path-planner component computes a new path each time the scene is updated. To measure the robot ability to replan the following time ratio is introduced

\[ \rho = \frac{\rho_t}{\rho_t + G_t} \]  \hspace{1cm} (9.3)

Here \( \rho_t \) and \( G_t \) denote, respectively, the time needed to the path-planner to replan the path, and the estimated time to reach the goal location. Figure 9.6(a) shows how the time frequency at which the scenario is updated, with the arrivals of new artefacts, affects the replanning time, under different parameter settings of the path-planner, hence it affects the robot short-term navigation capabilities.

In the second experiment we tested the long-term capability of the robot to navigate the cluttered environment in order to reach several goal locations. In this experiment the space complexity of the environment, as well as the parameters of the path-planner related
to the goal bias, have been taken into account. To measure the space complexity of the environment the following space ratio has been introduced:

\[
\nu = \frac{n_A}{n_{\text{free}}}
\]  

(9.4)

Here \(n_{\text{free}}\) and \(n_A\) denote respectively the number of free cells of the 2D occupancy grid \(\mathcal{M}_{2D}\) of the mapped area and the number of the cells occupied by the set \(A\) of artefacts within the environment. The robot is instructed with the task to reach multiple goal locations. The path-planner computes the initial path to reach each goal and it replans a path from the robot current position to the current goal pose, whenever an artefact arrives into \(\mathcal{M}_{2D}\), so as to find a collision free path, if one exists. Upon the receipt of the safe path, the execution component must be able to move the robot to effectively reach the current goal. In this experiment, the overall time needed to accomplish the task is measured together with the percentage rate of the reached goal locations. Figure 9.6(b) reports the performance of the robot in the navigation task with respect to different values of the space complexity of the environment.
Chapter 10

Conclusions

In this thesis we described the research work performed in the field of search and rescue robotics. Our experience, in May 2012, at Mirandola, at a disaster scenario of highest realism, led us to a better understanding of which were the main research challenges to be addressed. Among these challenges, only some of them have been deeply investigated. In particular, we addressed the problem of understanding, interpretation and representation of a disaster scenario, from both a human and a robot centric view. Indeed, we proposed an approach to build, at real-time, on top of uninformative point cloud data, a semantic representation of semi-structured environments. Building such a richer representation has three main advantages: (1) enhancing both human and robot situation awareness; (2) facilitating the formulation of a domain knowledge which can effectively be exploited by both human and automated reasoning and, finally, (3) providing a model of interaction between the robot and the environment in which it has to operate. This approach resulted to be very promising also for structuring complex cluttered environments. Indeed, the graph-based structure, underlying the semantic representation of the environment, allows us to capture, at a lower level of granularity, the morphology of the terrain, which the robot is traversing. Moreover, this structure provides both the robot and the human with information about the connectivity as well as the traversability of the different areas of the environment. On the other hand, our complementary research work on terrain traversability has been performed to face the problem of building a topology of the terrain surface, which the robot has to traverse, when classification in terms of slopes, stairs, pavement or walls is not feasible. Combined with the graph-based structure of the environment, this representation resulted to be crucial for traversability assessment, morphological adaptation of the robot and 3D path planning. To enable the robot to effectively track 3D paths, inferred from such a representation of the environment, we designed a motion planning system which is responsible of generating the velocity commands needed to follow the path and simultaneously adapting the position of the flippers of the robot to the surfaces on which the path lies. The main idea, behind the design of the controller, is to divide the 3D trajectory tracking problem in two sub-problems: (i) tracking the path by assuming that locally the robot is moving on 2D planes, namely the planes tangent to the surface, at each point of the path and, (ii) controlling the position of the flippers, on the basis of the normals to the surfaces, on which the robot is moving. According to this idea, the motion planning system can be divided into two decoupled control modules: (1) a trajectory tracking controller, and (2) a flippers position controller. The trajectory tracking controller implements a control strategy based on input-output linearization via
feedback. The controller takes as input the current pose of the robot, obtained by fusing laser data with odometry and inertial data, the pose of a virtual reference frame, on the desired trajectory, a velocity profile, and suitably generates the linear and angular control command, in order to asymptotically stabilize to zero the trajectory error. The main advantage of the proposed control strategy is to consider the motion of a point, displaced along the main axis of the tracked vehicle, instead of the motion of the centre of mass of the robot. As a consequence, the controller allows the robot to track any trajectory, even with discontinuous tangent to the path without requiring the robot to stop and reorient itself at those points. The flipper position controller estimates the normals of four points, on the surface on which lies the current segment of the path, representative of the contact points of the flippers with the surface. On the basis of the orientation of the normals with respect to the global reference frame of the robot, the controller suitably generates the positions commands of the flippers. However, the estimation of the normals is not accurate. Moreover, the flippers are neither endowed with contact sensors nor with proximity sensors. Therefore, it is quite hard to correct the estimation as well as to determine the contact between the flippers and the surface. To face this limit, we proposed a model of contact sensor. This model is based on a learned function, assessing the touch and the detach of the flippers from the surface. The flipper position controller activates this contact sensor to both correct the estimation of the position commands and to ensure that the robot has a better traction on the harsh terrain. The designed motion planning system endows the robot with the ability to autonomously climb stairs and to flexibly surmount obstacles. Besides the research challenges, related to environment understanding, 3D path planning, morphological adaptation and autonomous navigation, in this thesis, we tackled the problem of adapting the behaviors of the robot to the environmental contingencies, under the mechanism of task switching. In this context, we proposed a novel framework to model the stimuli of the robot, the stimuli-response mapping and the resulting task switching or stimulus inhibition robot decisions. Our approach has been inspired by the wide psychophysical and psychological literature on human patterns in stimulus-response and task-switching. In this framework, stimuli are defined as perceptual functions yielded by the processes, running on the robot, and learned via an informed logistic regression. In building the stimuli model the main difficulties we have encountered were due to the experiment setting and data collection. This led us to provide a control model for the selection of the training set, on the basis of the manifold of the features eliciting the stimuli. This turned out to be a good solution to transfer training data acquired by human operators to the robot training set. Stimuli-response mapping, leading to the selection of a set of possible responses to the current selected stimuli, is modeled as a content-based recommendation system. This model has proved to be suitable to estimate, under sparsity, the latent parameters underlying the stimuli-response mapping, as well as to predict those weights, given by each stimulus to each response, which were missing in the data acquired by the experience of human operators. The final best response behavior of the robot is based on a decision rule, which takes into account two different costs: (1) the cost to reconfigure the current state of the robot to the state, associated with the new task response and, (2) the cost required to solve the interference between the current state of the robot and the new state. To structure the whole framework we resorted to logical inference of the Temporal Flexible Situation Calculus (TFSC), which provided the grounds for the causal and constraints relations among processes carrying the information driven by stimuli. The logical structure has also been the framework within which processes and the robot states have been defined. This thesis also investigates another important research
problem in rescue robotics, namely, learning by demonstration, from skilled firefighters, the actions, the intentions, the affordances and the temporal relations among firefighter activities of rescue tasks. Moreover, how to formulate the learned concepts into plans, executable by the robot. A crucial role in the process of acquisition of the data coming from the experience of the firefighter, in a car accident scenario in a tunnel, with victims, has been played by the Gaze Machine. This device, worn by a firefighter, allowed us to gather and convey the visual and audio inputs from the firefighter, during the execution of the rescue task. More specifically, we obtained, from the information delivered by the Gaze Machine, a map of the positions and a path of the gaze of the firefighter. Both the positions map and the gaze path have been used, together with a running commentary, to learn the actions performed by the firefighter. The result of this process is a map of basic actions, divided in (1) body motions, (2) vision actions and, (3) manipulation actions, labelled with the corresponding starting/ending times. According to this map, we have defined the compatibility conditions for generating a flexible robot plan. For this purpose, we have defined a plan library of possible activities and affordances, according to the context, with the support of the instructions, provided by the firefighter. Based on this plan library, we generated a combinatory categorical grammar type plan lexicon, mapping observations to categories. The resulting categories, parsing the sentences within the running commentary, have been compiled into temporal constraint networks of robot activities. Here, for modeling both temporal constraints and cause-effect relations between activities we exploited the expressiveness and the flexibility of the TFSC logical framework. Finally, we proposed a framework, based on Augmented Reality (AR), to test, evaluate and validate a robotic system, during every phase of the design and development processes. Here, AR is view from a complete different perspective. Namely, rather than being a view of a human physical, real-world environment, augmented by computer-generated sensory input, AR is view as a tool to augment the perceptual model of robots. Indeed, virtual objects, placed into the real environment, can be perceived by the robot sensors. Moreover, virtual objects are modeled as intelligent agents, namely they are endowed with natural behaviors. Under this perspective, AR is a compelling technology. It allows us to design a variety of complex scenarios as well as to evaluate the behavior of a robotic system, while the real environment dynamically changes, being augmented by animated virtual objects. This framework allows us to also build a world model representation that serves as ground truth for training and validating algorithms for vision, motion planning and control.
123

Bibliography


Bibliography


[203] PIRRI, F. The well-designed logical robot: Learning and experience from observations to the situation calculus. Artificial Intelligence, 175 (2011), 378.

[204] PIRRI, F. The well-designed logical robot: Learning and experience from observations to the situation calculus. Artificial Intelligence, 175 (2011), 378.


