DR 4.1: Basic methods for collaborative planning

Federico Ferri*, Mario Gianni*, Matteo Menna*, Stefan Wilkes† and Fiora Pirri*

*Alcor Laboratory, Dipartimento di Ingegneria Informatica, Automatica e Gestionale "Antonio Ruberti"- Sapienza Università di Roma, via Ariosto 25, 00185 Rome, Italy

†Fraunhofer IAIS, Sankt Augustin, Germany

⟨pirri@dis.uniroma1.it⟩

Project, project Id:	EU FP7 TRADR / ICT-60963
Project start date:	Nov 1 2013 (50 months)
Due date of deliverable:	M14
Actual submission date:	March 2015
Lead partner:	ROMA
Revision:	final
Dissemination level:	PU

This document describes the progress status of the research on the development of formal basis of collaborative planning, focusing on multi-robot task allocation. The report also describes additional research work concerning both the consolidation and the improvement of the functionalities of the UGV and UAV, needed for collaboration. The research reported in this document concerns the WP4 for the Year 1 of the TRADR project. Planned work is introduced and the actual work is discussed, highlighting the relevant achievements, how these contribute to the current state of the art and to the aims of the project.

Executive Summary

The key objective of WP4 is to develop the formal methods needed to model knowledge exchange, knowledge maintenance, information sharing, common and individual decision structures in order to deploy collaborative planning. In Year 1, we focused on the multi-robot task allocation problem. In this context, we developed a model which manages the assignment of tasks to different robots, involved in a mission, under both time and resource constraints. The proposed task allocation model also deals with task reliability and failures. In addition, we extended the Augmented Reality framework, developed in NIFTi, for both training and validating the proposed task allocation model. In Year 1 we also concentrated on both the consolidation and improvement of the basic functionalities of the UGV and UAV, on top of which multi-robot task allocation has been deployed. In particular, we developed a framework for solving the autonomous 3D navigation task for the UGV. In this framework, we have faced the problem of 3D path planning, based on point cloud clustering and labeling, and motion control for flipper adaptation. We improved this framework with traversability mapping and dynamic obstacle removal. We developed a unified framework for trajectory tracking control design, based on both a direct and differential kinematic model of the UGV, correlating the motion of the robot body with the motion of the active flippers, in traversal task execution. We proposed a new approach to robot cognitive control design, based on a stimuli-response framework that models both robots stimuli and the robot decisions to switch among tasks. Finally, we developed a 3D motion planning and tracking algorithm for the UAV. Most of the research work concerning the improvements of the basic autonomous functionalities of the robots has been performed, together with other WPs, in order to increase both the degree of flexibility and reliability of the TRADR set-up.

Role of task allocation in TRADR

Task allocation is at the basis of multi-robot collaboration in TRADR. The proposed model establishes which task is assigned to which robot, as well as when a robot has to execute its assigned task, under uncertainties about task failures. Task allocation builds on the tasks which each robot can effectively perform. Each task is formulated on top of the functionalities every robot can exhibit. Therefore, while on the one hand it is quite important building a decisional structure for task allocation; on the other it is crucial to develop a set of basic functionalities which ensures that each robot of the TRADR team is effectively able to execute the assigned tasks. These considerations motivate part of the research work of WP4, jointly performed with WP1, WP2 and WP3, concerning the consolidation of those baseline

autonomous capabilities of both the UGV and UAV, required for implementing a basic structure for multi-robot collaboration, with the proviso of human-robot collaboration. In particular, the proposed traversability analysis enhances both the actual robot- and human-centric situation awareness of the TRADR system. The developed algorithms for 3D path planning and trajectory tracking control, complementary to those implemented by other WPs, provide the UGV with alternative strategies for planning and posture adaptation. These strategies, among those already developed by other WPs, can be selected on the basis of the terrain surface, topology and possible sources of failures. The main advantage of complementary strategy selection is to increase the flexibility, the robustness and the reliability of the UGV, under an autonomous setting. The flexibility of the UGV is even more increased by the proposed model for task switching. This model allows the robot to deal with critical situations, when it is required to take a decision whether to shift from a task to another or inhibit the urges, focusing on the task at hands. The implementation of a 3D motion planning and tracking algorithm for the UAV alleviates several crucial problems in TRADR, related to piloting the UAV in a confined space. Finally, the proposed Augmented Reality framework serves as a test bed for evaluating the effectiveness of every robot functionality, in both a coupled and decoupled setting. Moreover, the framework is used for collecting data about the reliability of the tasks, executed by the robots. These data are used for both task reliability analysis and the validation of the task allocation model.

Contribution to the TRADR scenarios and prototypes

The proposed model for multi-robot task allocation supports the decision making mechanisms, underling the human-robot teaming (WP5). The conceptual representation of the environment, based on both point cloud categorization and traversability analysis, extends the robot-centric world model, build by WP1. By resembling the way in which humans perceive the environment, this representation also enhances human-centric situation awareness for operational effectiveness (WP3). 3D path planning and trajectory tracking control for posture adaptation of the UGV, as well as 3D motion planning of the UAV, increase the level of autonomy of the TRADR system (WP1,WP2). The basic collaborative structure, supported by the actual autonomous capabilities of the UGV/UAV, allows WP7 to investigate more complex use cases and scenario evaluations with end-users, in Year 2 (WP3). The Augmented Reality framework contributed to the evaluation of the reliability of the robot's functionalities. Task failure rate has been analysed, across several different sorties, as the complexity of both the scenario and tasks increases. This assessment has been lifted to knowledge and then made

persistent, within the multi-robot task allocation model. The role of an in-field rescuer has also been investigated in the problem of allocating tasks among robots. The presence of an in-field rescuer has been modeled as a positive reward for the robots, in task assignment, as well as a low failure rate, in task execution. Several software packages have been implemented for traversability analysis, 3D path planning and trajectory tracking control for the UGV (analogously for the 3D motion planning for the UAV). These packages contribute to the set up of the TRADR prototype.

1 Tasks, objectives, results

1.1 Planned work

The planned work of WP4, in Year 1, concerning "Basic methods for collaborative planning" is described in Task T4.1. Task T4.1 achieves the objectives described in Milestone MS4.1. An excerpt of the description of both Task T4.1 and Milestone MS4.1 is reported in the following.

Task T4.1 The goal of Task 4.1 is to develop the formal basis of collaborative planning, focusing on the early collaborative issues not requiring a full knowledge management. T4.1 expected result at the end of Yr1, is the formalization and implementation of basic collaborative planning methods for the generation of a plan common to two or three robots and its execution on an uninstantiated horizon. The model includes a basic memories structuring for both individual and common evaluation of the plan while monitoring its execution. A contribution of Task 4.1 is also an augmented environment, with simulated representation, namely an Augmented Real Environment (ARE) meant to fill in the lack of a common representation of the perceptual data. The novelty of ARE is that it provides the real robot, operating in a real environment, with an augmented reality, by simulating other robots, people, objects, and the knowledge about them is shared and is made uncertain, replicating noise and incomplete information of real environments.

Milestone MS4.1 MS4.1 actuates very basic collaboration performance, providing the early execution model for collaborative planning. The execution to be operated both in real and augmented environment actuates the different levels of knowledge generating and executing a plan. This concerns a common goal for a group formed by one UAV and one UGV. The novelties MS4.1 intends to prove are (1) collaborative finding of an unknown target; (2) generation of a common plan handling each other role; (3) compilation of plan results into new knowledge. The last item anticipates major results of WP4 on persistence.

1.2 Actual work performed

The actual work performed supports the objectives of Milestone MS4.1. This work focused on the development of a model for multi-robot task allocation. This model establishes which tasks are assigned to robots, as well as when a robot has to execute its assigned task. Task assignment takes into account both task failure and reliability due to the occurrence of unknown exogenous events. The Augmented Reality framework, developed in NIFTi, has been extended with a probabilistic model for both generating virtual events and

regulating their dynamics. This model served as a ground truth for analysing the reliability of the tasks, executed by the robots, while the planning scene dynamically changed, being augmented by virtual events. However, multi-robot collaboration presupposed a model of perception, reasoning, planning and execution of the UGV (analogously of the UAV). Such a model built on the effective functionalities, which the robot can exhibit, such as path planning, given a suitable representation of the environment, accounting for the complexity of the terrain, trajectory control for path execution and morphological adaptation, resource management and task switching, dealing with the choice of the best task to be executed, when unexpected events occur. Some of these basic functionalities have been developed within NIFTi, jointly with other project partners. However, the actual capabilities of the robot resulted to be still very weak and unreliable to support a collaborative model of task planning. Therefore, in Year 1, we also concentrated on both the consolidation and improvement of the main basic functionalities of the UGV (analogously of the UAV), on top of which multi-collaboration has been implemented. In particular,

- we developed a preliminary framework for solving the autonomous 3D navigation task for the UGV. In this framework, we have faced the problem of 3D path planning, based on 3D map clusterization and labeling, and motion control for flipper adaptation.
- we improved the framework for 3D autonomous navigation with traversability analysis and dynamic obstacle removal. The framework also integrates an extended version of randomized $A^\star$, coping with difficult terrains and complex paths for non-holonomic robots;
- we formalized the problem of traversability analysis within a probabilistic framework;
- we proposed a unified framework for trajectory tracking control design, based on both a direct and differential kinematic model of the UGV, correlating the motion of the robot body with the motion of the active flippers, in traversal task execution;
- we developed a 3D motion planning and tracking algorithm for the UAV.
- we proposed a new approach to robot cognitive control based on a stimuli-response framework that models both robots stimuli and the robot decisions to switch among tasks in response or to inhibit stimuli;

In synthesis, this section reports the research carried out by WP4, more specifically the results in Task T4.1.

1.2.1 Multi-robot task allocation model

In the multi-robot task allocation problem, the goal is to allocate several tasks amongst members of a team of autonomous robots such that there are no conflicts, while maximizing the reward received for performing the tasks [36, 85]. The simplest forms of multi-robot task allocation problem in general are combinatorial problems, for which the optimal solution can be very difficult to be efficiently found [42]. A common approach to dealing with this complexity is to use a sequential greedy allocation algorithm, where tasks are allocated by iteratively finding the task-robot pair which results in the greatest net reward increase, and allocating that task to that robot. Sequential greedy solutions have been shown to provide acceptable approximations which are typically much faster to compute than the optimal solution [14].

However, in USAR domain applications several unknown exogenous events can occur. These events can either positively or negatively affect the performance of the tasks, executed by the robots. If we allow tasks to involve exogenous events the following issues arise: (1) the type of the exogenous events is unknown a priori; (2) the number of possible exogenous events which can occur is unknown a priori; (3) the behaviors which these exogenous events can exhibit are unknown a priori; (4) the number of possible behaviors is unknown a priori; (5) when and where an exogenous event can occur is unknown a priori; (6) the reward for tasks can no longer be assumed known a priori, since the exogenous events are likely to have influence upon them and, finally, (7) the reward received for doing a task involving one exogenous event may not contain any information about future task involving another exogenous event, as each exogenous event may have a different influence on the received reward.

In a multi-robot task allocation problem, all the possible exogenous events which can occur within a real environment can not be explicitly modeled. Moreover, even if we were able to detect these events, we would not be able to predict the behaviors they are going to exhibit. Both events and behaviors are not directly observable by the multi-robot team, in a real environment. What we can directly observe either is the failure or the success of the tasks executed, over time, by the multi-robot team. In order to allocate tasks among a team of robots, information from previous tasks successfully completed or failed can be effectively used. A possible approach to this issue is to model the task reliability, namely, the ability of the robots to execute tasks, under stated conditions for a specified period of time. For example if the failure rate of a task, executed by a robot, becomes very high, after a certain time period, probably that task should be assigned to another robot, whose failure rate, for that specific task and at that time period, is lower. Thus, by predicting both the life-cycle and the risks of failures of the tasks, the multi-robot task allocation model can use past tasks successfully

completed involving the robots of the team, to better distribute future tasks among them, over time.

There are several types of methods that are used for reliability prediction, mainly coming from industry [5]. In particular, we resorted to the Weibull analysis [72]. The primary advantage of this analysis is the ability to provide reasonably accurate failure analysis and failure forecasts with extremely small samples. In order to build the Weibull model for reliability prediction, we proceeded as follows. We deployed three robots, endowed with a different sensor suite, within a real simple scenario. Each robot was endowed with a finite set of functionalities, depending by the kind of built-in sensors. Each robot performed a finite set of tasks, designed by suitably combining its own functionalities. We augmented the real simple scenario in which these robot operated with dynamic virtual events, through the Augmented Reality framework. We instructed each robot to execute each task, while the simple scenario dynamically changed, being augmented by these events. For each task executed by each robot we measured its time-to-failure. After these experiments, we selected a statistical model that fitted the data and represented the life of the tasks based on the goodness-of-fit test. Then, we used the gathered data to estimate the parameters of this model, via Maximum Likelihood Estimation. The learned model has been used to calculate the probability that each task, performed by each robot, will operate successfully at a particular point in time. These probabilities have been used to weight the reward each robot receives for the execution of a task, within the formulation of the maximization problem for task allocation. The Augmented Reality framework, used to dynamically augment the real scenario of the robots with virtual exogenous events, is described below.

1.2.2 Modeling uncertainty of the augmented reality world in ARE

We extended the AR-based framework, proposed in [31], with a model regulating both the generation and the behaviors of virtual exogenous events, that will populate the real environment in which the robots operates. The extended framework is composed of two main building blocks: (1) the AR-Builder and, (2) the AR-Server. The AR-builder comprises three stochastic models: (1) the events model; (2) the behaviors model and, (3) the population model. The event model is based on a Hierarchical Beta Process (HBP) [92]. The HBP includes a beta process which is used as a prior over the probabilities of an event exhibiting each behavior. The model also incorporates a separate beta process prior for each type of exogenous event. The HBP allows us to stochastically model the generation of the exogenous events, without establishing a priori both the number and the types of events that can populate the real world model of the robots as well as the assign-

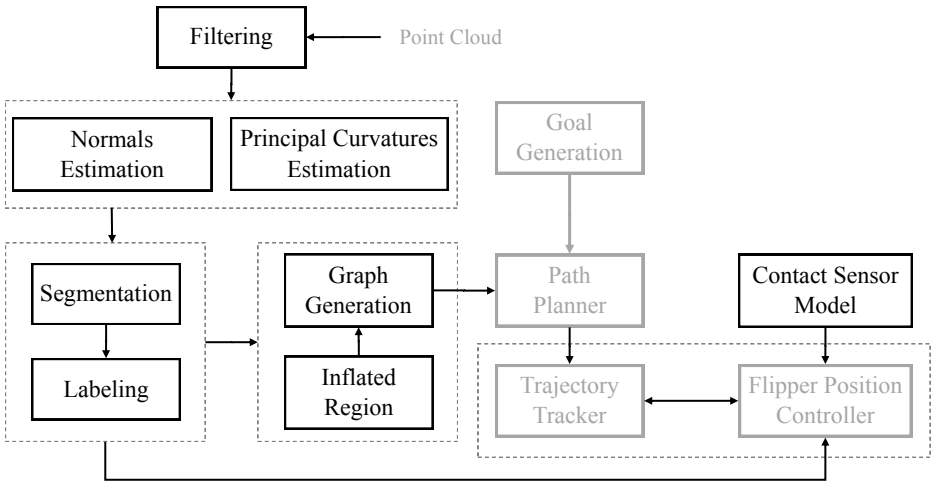
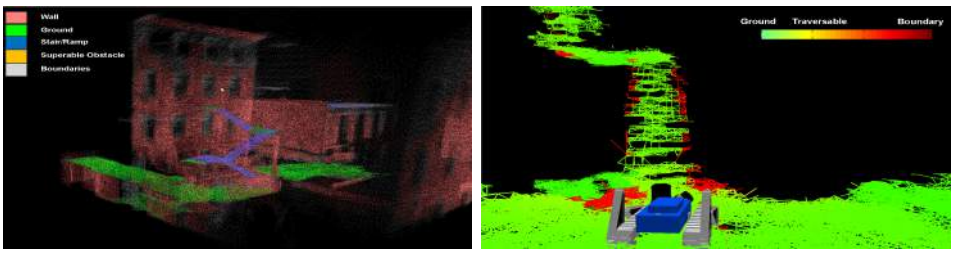


Figure 1: Framework overview. Note: solid blocks denote our contribution, while shaded blocks denote third party components.

ment of the behaviors to the events, without fixing a priori which behaviors each event can exhibit. The model also allows exogenous events within a class to share the probabilities of exhibiting each behavior, while allowing for different probabilities across classes. The behavior model is based on a Dirichlet Process Gaussian Mixture Model (DP-GMM) [66]. The DP-GMM allows us to model the observed behaviors of the generated exogenous events without assuming that the events exhibit a fixed number of behaviors. The population model relies on a spatio-temporal Poisson Process to model, in both time and space, the arrival and leaving of the exogenous events [24]. The AR-Server interconnects the real environment model together with the simulation model of the events [31]. The generated augmented world model serves as ground truth for training and validating the proposed multi-robot task allocation model, when task reliability of each robot, executing a task, over time, is affected by the presence of exogenous events.

1.2.3 Real-time 3D autonomous navigation framework for the UGV

We developed a framework for 3D path planning and motion control for the UGV. This framework has been integrated into the UGV navigation stack of TRADR. The main purpose was to extend the autonomous functionalities of the UGV, previously developed in NIFTi. On top of all the autonomous functionalities of the robot we have built a task library. Such a task library has been used to develop the proposed model of task allocation for multi robot collaboration.



(a) (b)

Figure 2: (a) Point cloud segmentation and labeling and (b) weighted graph representation of a fire escape stairs scenario.

The framework comprises three main building blocks: (1) point cloud segmentation and labeling; (2) graph generation and inflated region estimation and, finally, (3) flippers posture adaptation, based on contact sensor modeling. The overall schema of the proposed framework is illustrated in Figure 1. Point cloud segmentation and labeling provide a basic categorization of the environment, specifically defined for navigation purposes, that is, walls, ground, stairs, ramps, and obstacles that can be overcome. This process is made by the following steps: (1) point cloud filtering; (2) estimation of normals to the surface and curvature and, finally, (3) clustering and merging of the filtered point cloud. Clusters are labeled according to the geometrical constraints applied to the surface normals, to the mean curvature and to the 3D point coordinates. This results in a classification of the point cloud into walls, stairs or ramps, and ground and surmountable obstacles as illustrated in Figure 2(a). Points belonging to clusters, labeled as ground and stairs or ramps, are connected based on an iterative procedure taking into account both the model and the kinematic constraints of the UGV, namely its morphology as well as its ability to overcome obstacles. The result of this procedure is a graph connecting the different regions of the point cloud, denoting areas accessible by the UGV. In parallel, both boundary and inflated obstacles regions are estimated by projecting the points labeled as walls onto the planes tangent to the surfaces approximating ground, stairs or ramps. Upon the estimation of the boundary regions, the edges of the connectivity graph are weighted by a factor taking into account the distance of the graph vertexes from these boundaries, the density of the neighborhood of the vertexes and the arc length of the edge. This traversability structure is used by a graph-based planning strategy to find minimum cost feasible paths toward target goals (see Figure 2(b)). In order to allow the UGV to both track the 3D planned path and suitably adapt flippers to the terrain surface on which the path lies, the tracking controller of the UGV integrates a contact sensor model. This model is used to correct the morphology of the robot on the basis of the contact between the flippers and the surface. The

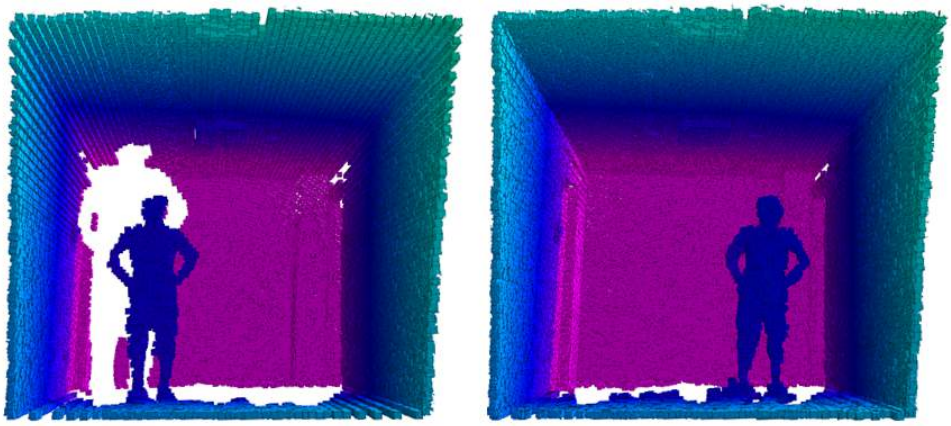


Figure 3: On the left a person is standing in front of a wall (dark blue blob on the left). The white blob behind the person indicates that the point cloud of the region of the wall has not been aggregated into the entire map of the area, due to occlusion. On the right the result of the dynamic correction algorithm: after the motion of the person, the map is updated by both deleting the blob of the person and by aggregating the point cloud belonging to the wall, previously occluded.

model is based on a learned function, assessing the touch and the detach of the flippers from the surface. This approach for flippers control ensures a better traction of the robot on the terrain, during the trajectory tracking task. For more details and results concerning this research work we refer to Annex §2.1.

1.2.4 3D path planning in cluttered and dynamic environments

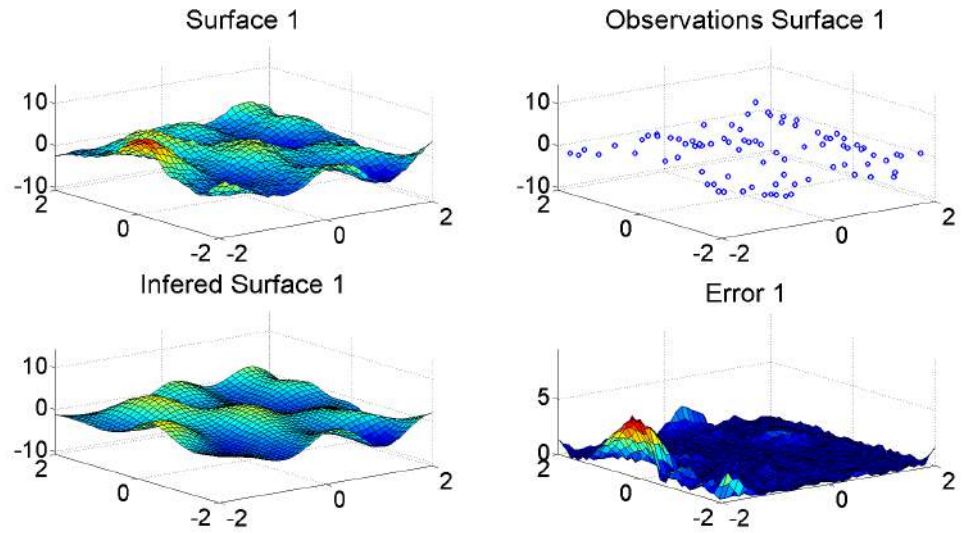
The framework for real-time 3D autonomous navigation, described above, assumes that the UGV navigates within a static 3D Map. Actually there are no special procedures which are responsible of merging new scans with the accumulated map, accounting for dynamic obstacle removal, into the mapping functionalities of TRADR. Therefore, after a short time period, the 3D Map becomes very cluttered, due to the presence of dynamic obstacles, making the UGV autonomous navigation impossible. Moreover, in the proposed framework, each traversable point within the point cloud was considered as a possible successor state by the planning algorithm. This assumption extremely increases the dimension of the search space, thus breaking down the performance of the algorithm. In order to face these two main drawbacks, we developed a 3D path planning framework which integrates a procedure for dynamic obstacle removal as well as a method for sampling candidate successor states so as to reduces the planning domain. The method for testing whether a point should be removed is based on a local signed distance

test against the nearest simplex of the convex hull of the scan, acquired from the rolling laser sensor. This methods tells us whether a point is inside the deletion volume [102], also in the presence of noise. The obtained result is illustrated in Figure 3. Sampling is governed by a probability density function induced by both traversability analysis and obstacle detection. For more details and results see Annex §2.2 and Annex §2.7. A further improvement of this research work is reported in Annex §2.3. In the context of the multi-robot collaboration, the proposed dynamic obstacle removal procedure makes both more reliable and effective the autonomous navigation of the UGVs in patrolling tasks.

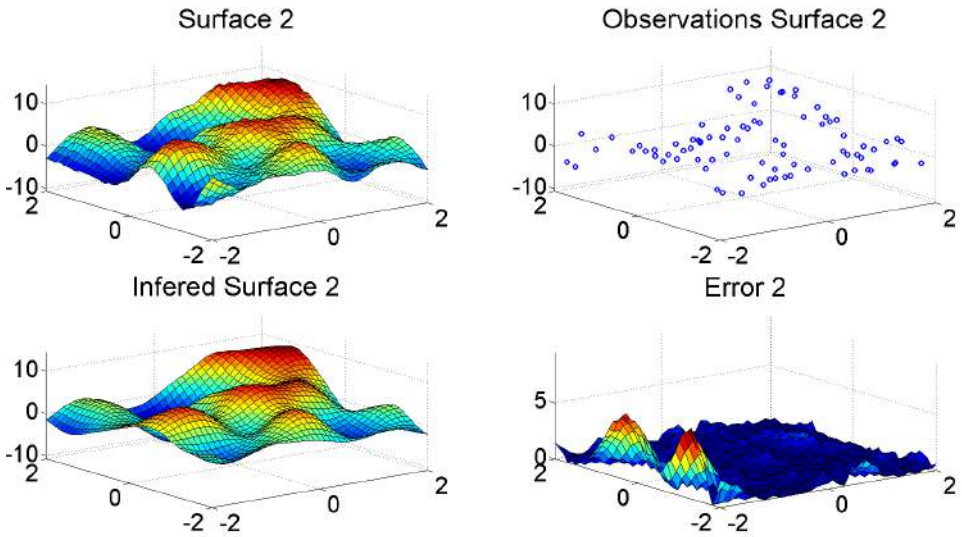
1.2.5 Probabilistic framework for traversability analysis

During the assessment of the reliability of the UGV navigation task, balancing tasks assignment in the proposed multi-robot task allocation model, we tested the performance of the different algorithms for 3D path planning and trajectory tracking, developed in NIFTi and in WP4 research work. In this evaluation we verified that having multiple planning strategies among which the UGV can choose significantly increases both the flexibility and robustness of the overall system. However, safety in navigation tasks was not completely ensured. In fact, when the task allocation model assigned the navigation task to the UGV towards a complex cluttered area of the environment, we were forced to manually interrupt the task, in order to prevent robot damages. Therefore, the reliability of the UGV navigation task turns out to be very low, thus bounding tasks assignment. This experience led us to analyse the problem of autonomous safe navigation. This problem foresees a robot-centric representation of the terrain assessing traversability. Therefore, the research work of WP4 partially pursues to develop a model for estimating terrain traversability. This work differs from the research work of WP1, concerning adaptive traversability, as it mainly focuses on building a traversability map of the environment, where path planning can take place, rather than training the controller to adapt the robot flippers to different terrain surfaces.

Traversabilty is a continuous scalar metric representing the cost to traverse a region. It is usually calculated either from maps or sensor data. Traversability provides a cost map allowing potential obstacles and difficult regions to be avoided at runtime. The cost of traversability is computed combining geometric features of the neighbourhood of each observation (e.g., terrain slope, roughness, obstacle presence). Prior works on traversability cost estimation do not account for uncertainty and missing information, in a statistically direct manner. Gaussian Processes (GPs) regression have recently become popular methods for traversability cost estimation. These methods handle uncertainty as well as appropriately represent spatial correlation resulting also effective for managing incompleteness of data. The



(a)



(b)

Figure 4: Figure shows the inference results on synthetic data for terrain surface (a) and traversabilty cost (b). Top left of each figure shows the original surface, top right shows the observations, bottom left is the inferred surface and bottom right is the error. The surfaces are jointly learned using the multi-task learning framework.

model of terrain produced by GPs is scalable, yielding a continuous domain representation of the terrain data. Moreover sampling can be performed at any desired resolution. However, GPs-based spatial stochastic processes are quite limited when dealing with traversability cost estimation in the presence of dynamic obstacles. In these settings, time plays an important role for a proper estimation of terrain traversability. In fact, traversabilty mapping of environments with dynamic obstacles requires an update at each time step. Spatial processes, intrinsically memory less, only provide a static representation of the environment, leading to unsatisfactory planning performance. In order to cope with these issues we developed a model for terrain traversability, based on spatio-temporal Gaussian Processes, in the context of multi-task learning probabilistic framework. Multi-task learning is an area of machine learning whose goal is to learn multiple related processes avoiding tabula rasa learning by sharing information between the different processes. The main advantage of this approach is to simultaneous learn both these processes in order to the performance over the no transfer case.

Given the set of measures provided by the robot 3D laser sensor and the traversabilty cost computed on the measured points we infer the model of both terrain surface and trasversabilty cost, also taking into account time relations between different observations. We approached the inference problem by placing a GP prior over the latent process for the terrain surface and the latent process for the traversabilty cost.

In this model, we defined a covariance function modeling the correlation between both the observations of each process and the inter-processes covariance among the different processes. The latter captures the space-time correlation between the processes. This covariance function is stationary with respect to space but not-stationary with respect to time, in order to manage local changes on both the terrain surface and the traversability cost, due to the presence of dynamic obstacles. Model parameters are learned by the maximization of the marginal likelihood of the observations, given both the inputs and the parameters. The optimization process is constrained in order to guarantee the properties of the covariance function (e.g., semi-definitive positiveness) as well as to reduce the search space, ensuring real-time reliability. A preliminary result on synthetic data is shown in Figure 4.

1.2.6 Adaptive Robust 3D Trajectory Tracking for the UGV

In Annex §2.1 we developed a preliminary trajectory tracking control model of the UGV. This control model has been evaluated at the Italian Fire Fighters rescue training area in Prato, during Year 4 of NIFTi, and at various fire-escape and ordinary stairs. In Prato we observed that the robot had locomotion difficulties in rotational motions, when it was autonomously traversing narrow passages, due to the flippers in a flat configuration. In

this situation, a better control strategy could have been the decision to lift the flippers to enhance the robot mobility, instead of maximizing the contact surface with the ground. Further, where the terrain was particularly unstable, the rotation of the belts of the tracks caused ditches in the sand, within which the robot got stuck, due to a loss of both friction and propulsion. Still, when the robot was passing over holes or gaps, the behavior of the flipper position controller was to lower the flippers, as much as possible, causing a bump at the end of the flippers with the negative side of the obstacle. If the slope of the negative obstacle was not so steep, the controller recovered from this situation. Otherwise, the flippers got stuck under the hole, causing damages to the vehicle. The flipper position controller was not able to suitably adjust the flippers posture due to the lack of a fast feedback about the structure of the perceived terrain. To cope with this limitation the trajectory tracking controller could have had to scale the robot velocity while waiting for the feedback to move the sub-tracks and correctly approach the slope of the negative obstacle. This drawback suggests to jointly model both the controllers for skid-steering and the sub-tracks posture adaptation.

The performance obtained at the fire escape stairs scenario, was more encouraging. The robot, endowed with the decoupled control modules, autonomously climbed the stairs, from the basement up to the landing of the second flight. However, we noted that oscillations of the heading direction of the robot frequently occurred, thus increasing both the lateral and longitudinal slippage between the tracks and the ridges of the stairs. The controller generated high values of angular velocity in order to accurately track the planned trajectory, without accounting for slippage compensation. Unfortunately, the kinematic model, underlying the trajectory tracking controller, did not take into account the slippage, when the control commands of the robot were generated. Moreover, the flipper position controller, even with the contact sensor model, was not able to reduce this effect.

On the basis of the lessons learned during this in-field experience, we developed a general preliminary solution for trajectory planning and control of the UGV. Trajectory planning combines the intrinsic robot characteristics with the geometric properties of the terrain model. The goal of trajectory planning is to negotiate collision-free trajectories taking into account the workspace of the active flippers of the robot. Trajectory control adapts the configuration of the flippers while simultaneously generating the track velocities, to allow the vehicle to autonomously follow a given feasible 3D trajectory. The control relies on both a direct and differential kinematic model of the UGV. The benefit of this approach is to allow the controller to flexibly manage all the degrees of freedom of the UGV as well as the skid-steering. The differential kinematic model has been designed to extend the differential drive robot model, described in [29] to compensate the slippage between the robot tracks and the terrain. Moreover, this model allows us to derive a feedback control law, which ensures the positional error of

the end points of the front flippers, to converge to zero. This control law dynamically accounts for both the kinematic singularities of the mechanical vehicle structure and those vehicle configurations in the neighborhood of a singularity. The designed controller also integrates a strategy selector to reduce both the effort of the flipper servo motors and the traction force on the robot body, recognizing when the robot is moving on an horizontal plane surface. According to this strategy, rotational motions of the robot, moving within narrow passages, are also facilitated.

The main idea behind the design of the controller is to apply both the concepts and methodologies of robot manipulator kinematics to the UGV, and to extend these methodologies with skid steering principles, accounting for slippage between the tracks and the ground. The role of the strategy selector is to replicate the behavior of a skilled operator, in situations such as navigating on flat terrains or traversing a narrow passage. For example, a skilled operator would lift up the robot sub-tracks to increase the mobility as well as to facilitate rotational motions. Optimization techniques have been applied to find a solution to the inverse kinematics of the UGV, in the presence of singularities [80]. To take into account the closeness of the robot configuration to a singular configuration, a heuristic is proposed [13]. This reduces the task error when the robot configuration is far from singularities. Finally, a pose refinement technique, exploiting the performance of a Dead Reckoning System together with the accuracy of an ICP-based simultaneous localization and mapping (SLAM), has been proposed to increase the rate of the control loop [29]. For more details and results see Annex §2.5.

1.2.7 Three dimensional motion planning for the UAV

In NIFTi, the UAV was essentially teleoperated by a human pilot. Conversely, in TRADR, the UAV is expected to perform tasks, under an autonomous setting, in order to jointly collaborate with the UGV during the execution of a common task. Therefore, the implementation of a 3D motion planning and tracking algorithm for the UAV is crucial in TRADR to investigate forms of basic collaborations among heterogeneous robots.

Path planning deals with the search for a valid configuration sequence, which securely moves the UAV through the three dimensional space, to reach a predefined goal position. To solve this problem several steps are necessary.

- Discretization of the configuration space, called C-Space
- Finding a search algorithm
- Tracking of a found path and avoiding obstacles

It is supposed that the estimated position of the UAV is provided as transformation and environment perceptions are continuously available via

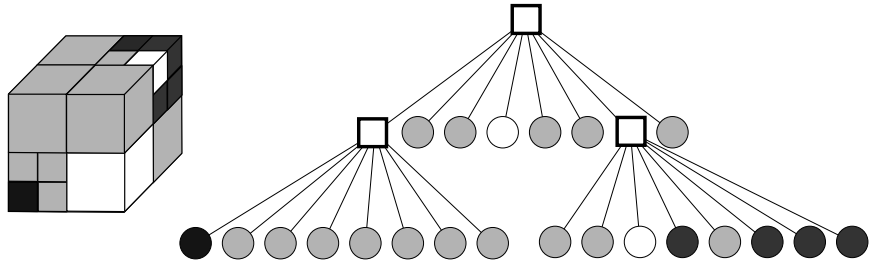


Figure 5: Example for an octree representation (right) of a cubic room (left). A node takes one of the follwing states: Free (white), occupied (dark gray) or undefined (gray), which needs to be split again.

a point cloud. Based on this requirements the following algorithms are hardware independent and can be used even in a simulated environment.

C-Space discretization The C-Space discretization is an essential step to optimize point cloud access times and makes search algorithms more efficient. It is known, that the classic three dimensional workspace, represented by the point cloud, needs a lot of space, because each fragment of the environment is stored as a three dimensional point. The configuration space is an extension of the workspace. It is the space which includes all possible configurations of the UAV and contains, along the position, the velocity, the rotation angle or other configuration vectors. So the C-Space takes a lot more dimensions than the workspace in general.

To reduce the C-Space, a three dimensional discretization technique called Octree is used. An octree is a tree based data structure which divides the configuration space in each dimension. The resulting octets are now representing a smaller part of the original space and are going to be classified. The octet can be a node, which is synonymic to an unknown space or a leaf. A leaf gets the state free or occupied, based on the perceived environment information. A node can not be classified into a free or occupied state and needs to be divided again. This step is iteratively called, till the whole configuration space is split into octets, which has a defined state.

Figure 5 shows the octree representation of a cubic based model. A classic voxel grid representation, which means a discretization using a fixed size, requires the storage of 512 datapoints. By building an octree based representation, which combines voxels with the same state, only 25 datapoints have to be stored. In this simple example, a compression rate of 96% is reached. Another advantage of the octree is the option to model a safety zone in the configuration space. By defining a minimal octet size based on the UAV diameter, even the smallest obstacle takes the place of the minimal

octet size. This operation is known as the Minkowski sum.

$$A = A \oplus B \tag{1}$$

The UAV can be handled as a single three dimensional point in further path planning and collision check operations. The C-Space itself is just a definition of a set containing possible configurations. Each configuration vector, or more exactly its position component, can be validated by simple checking the corresponding occupation state in the octree representation of the perceived environment. To visualize this model representation, all occupied voxels are going to be drawn.

Path planning using RRTConnect After the C-Space, respectively the workspace discretization, the method for finally planning the path through the environment can be presented. This planning task is also called global path planning. What the keyword global stands for, is shown in the next paragraph. Due to the high dimensional configuration space, a random based algorithm is used to plan the configuration sequence for the UAV. This algorithm doesn't find the optimal path, but it's able to calculate a valid path in a short period of time.

For random based movement planning in robotics Rapidly-Exploring Random Trees[52], short RRT, are frequently used. A RRT is a tree based data structure build by an incremental algorithm. For K iterations a random configuration is taken from the C-Space and added to the tree using the *extend* function. Algorithm 1 implements this extension, which is the main functionality of the RRT based path planning. After choosing a random configuration q_{rand} the nearest configuration q_{near}, which is already connected to the RRT, is determined. Afterwards the tree will be extended, as shown in Figure 6, from the nearest configuration to the random configuration with a previously defined metric ϵ, assumed that there are no obstacles between this extension q_{new} and the already connected configuration q_{near}.

```
1  def extend(tree, q):
2    q_near = nearest_neighbour(q, tree)
3    if new_config(q, q_near, q_new):
4      tree.add_node(q_new)
5      tree.add_edge(q_near, q_new)
6      if q_new = q:
7        return REACHED
8      else:
9        return ADVANCED
10   return TRAPPED
```

Algorithm 1: Extends a RRT by a new, randomly chosen, configuration.

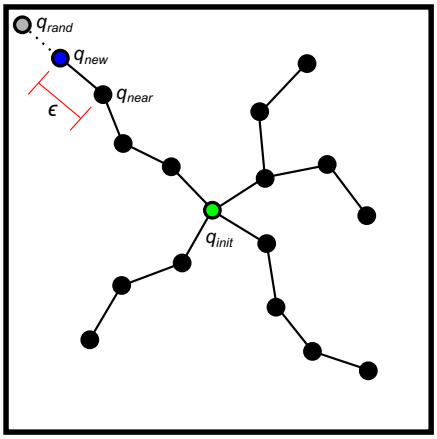


Figure 6: Single iteration of the RRT construction in a two dimensional configuration space.

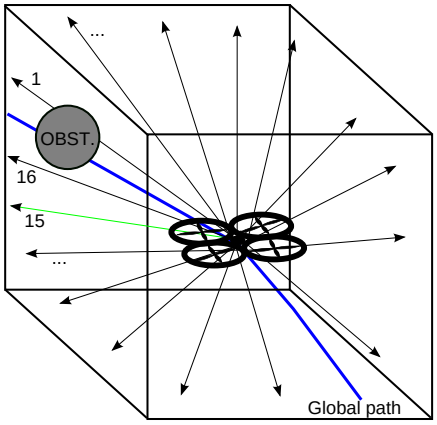


Figure 7: Highly discretized visualization of the DWA approach in three dimensional space.

The extension ends in one of three defined states. If we have added a new configuration q_{new} to the RRT, the state *ADVANCED* is returned. Otherwise no more configurations are available or can't be connected to the tree without passing obstacles. This path is in a *TRAPPED* state. A special case is, that the new configurations equals the randomly chosen configuration. In this case we have found a complete path, which is used in Algorithm 2 and not part of the basic RRT algorithm.

A modification of the RRT algorithm is presented in the RRTConnect[49] approach and shown in Algorithm 2. Two trees are generated parallel, where the first tree has its root in the start configuration and the second tree in the goal configuration. The algorithm doesn't search for a connection between start and goal anymore. It's searching for a connection between those RRTs. After initializing both trees, tree A is extended by a randomly chosen configuration as explained above. Now the algorithm tries to extend tree B also against this configuration, so the trees always grow into the same direction. The function *connect* repeats this extension, til no more extensions are possible towards this configuration. If an extension connects the configuration q_{new} from tree A and q_{new} from tree B, the function extends returns in the state REACHED, which was explained above. In this case a connection between the trees, and also a possible path was found.

```
1 def rrt_connect(q_start, q_goal):
2   tree_a.init(q_start)
3   tree_b.init(q_goal)
4   for k = 1 to K:
5     q_rand = random_config(C)
```

```
6       if not extend(tree_a, q_rand) == Trapped:
7         if connect(tree_b, q_new) == Reached:
8           return Path(tree_a, tree_b)
9       swap(tree_a, tree_b)
10    return Failure
```

Algorithm 2: RRTConnect, a parallel approach for finding a path between the start and end configuration.

The RRTConnect algorithm tries to extend the path in two direction, which results in fast execution times. Another big advantage is the metric definition. With this metric, also known as the extended step of the path, restrictions of physical motions or velocities can be modeled. On these grounds the RRTConnect algorithm was chosen as the path planning algorithm for the UAV. Due to the usage of octree based environment perceptions the path extensions can be validated fast, if a metric corresponding to the leaf size is defined. In this case an extension step equals always results in directly connected voxel of the origin configuration.

Path tracking and improvements due to dynamic navigation The RRTConnect algorithm has calculated a path regarding current static obstacles. To transfer this path to the controller of the UAV, a PID controller is used. Equation 2 shows the difference equation for a time discrete PID controller, which is used for each dimension of the configuration space. The term e denotes the error between the desired position of the configuration on the path and the actual configuration of the UAVs. Using the parameters P, I and D, the variable y, the velocity for the corresponding configuration dimension, can be sized, where these parameters stand for for the proportional, the integral and differential component. The parameters were determined for the simulation based on Gazebo (cf. DR2.1). For the pid control of the UAV three controllers were implemented, one for each translational motion in space. The rotation is not controlled at the moment, because roll and pitch motions are not needed for an autonomous navigation and the yaw position can be set directly at the target position. Has the UAV reached the current position of the path, the controller error tends to zero and the next position of the path can be set as the target position.

$$y_n = e_n * P + \sum_{i=0}^{n} e_i * I + \frac{e_n - e_{n-1}}{T} * D \tag{2}$$

The planning of the path is currently based on the planning of a collision-free path throughout the whole perceived environment. This is also called global path planning. In general, however, the environment is dynamic, that means the environment and its obstacles can change any time. Since

the dynamic detection of the complete environment is impossible due to technical difficulties also a second, local navigation approach is required.

Fox, Burgard and Thrun present a method for implementing such a dynamic local navigation: the Dynamic Window Approach, short DWA[26]. This strategy for collision avoidance simulates possible movements over a small time interval and creates a window of possible trajectories. A subsequent evaluation of this trajectories tells you which of the simulated control commands stays best on the path and avoids possible obstacles. The selected control command is transmitted to the robot, whereby a periodic loop is formed. The recently introduced pid controller for tracking the global path is omitted. Possible criteria for evaluation are the distance to an obstacle, the distance to the navigation target, the distance to the global path, or even velocity constraints. It is important to mention that for the assessment of the trajectories, only the current nearby environment of the UAV is used, the local map. For sensors such as the xTion this information is always available.

You can find a DWA implementation in the ROS navigation stack, which is designed for two-dimensional navigation. For the UAV, velocities in each dimension are necessary, because the UAV is able to perform holonomic movements in the three-dimensional space. However, the subsequent procedure is identical. The choice of the simulation time determines the size of the window of generated commands, as well as the control time in which the system can respond to external influences. Figure 7 shows an example of the evaluation of a control command for a UAV in three dimensional space. Corresponding to the distance to the global path, the first control vector would be chosen. However, this configuration is now blocked by a new obstacle that was not considered in the global planning of the path. The algorithm then selects control vector 15 as the first action to be performed, because it's the most secure path.

Experience has shown that the method always achieved good results in two dimensional space. According to the developers a robust dynamic navigation, for a robot with speeds of up to one meter per second, was already possible in 1997. If the final sensor system for the UAV is able to measure three dimensional space instantly, then the DWA method should be used.

1.2.8 The stimulus-response framework

Several unexpected events can occur within a rescue scenario. These events can induce a robot, collaborating with others robots in the execution of a common task, to either shift from the current task to another one or inhibit the inappropriate urges, preserving focus on the task at hands. For example, if during the execution of a task, the power level of the battery drops below a certain threshold, the robot has to decide whether to return

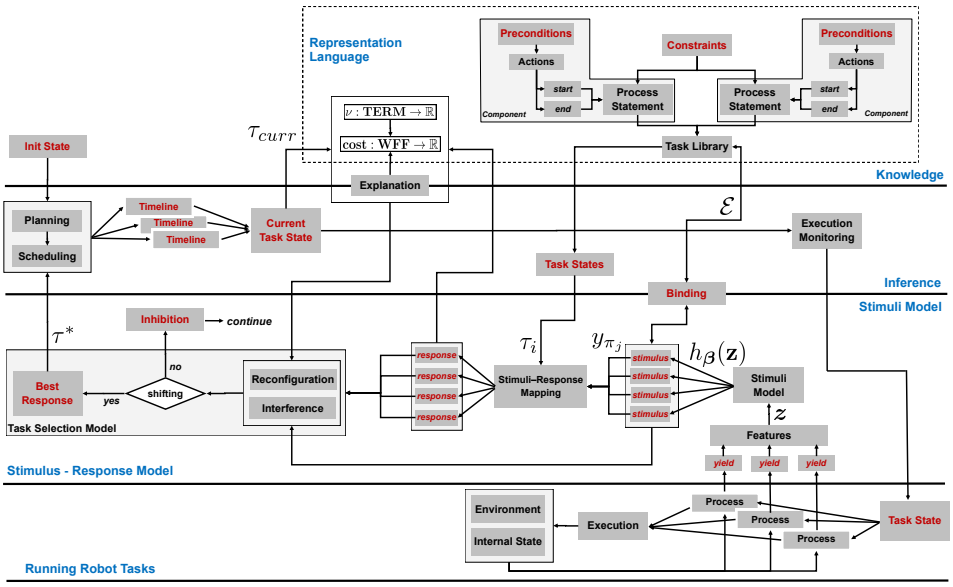


Figure 8: Work-flow of the proposed stimulus-response framework. From bottom to top: the lowest layer indicates a current active task, being executed. The active processes feed the processes yields collecting information from them, the stimuli model selects from the yields those which are stimuli. The stimulus-response model scores the response tasks. The decision is taken evaluating the payoff of switching to a task suggested by the stimulus-response matrix. In the middle panel, between knowledge and inference, the execution monitoring takes care of the actual task execution and its updating, both affecting the mental states, namely the decision of whether to consent a response to the stimulus. In the upper panel, a first-order logical formalism is used to model the robot processes, with action preconditions and effects, affecting activation costs and motivating causal constraints. This closes the loop between stimulus activation, reasoning, planning and decision.

to the command base station, to recharge the battery, or to continue the task. In the first case, the decision to shift to another task can lead to a re-allocation of the tasks among all the other robots involved in the common mission. Therefore, the task allocation model requires a mechanism which notifies it when a robot is deviating from the assigned task, due to its internal decision to switch to another task, in order to accordingly re-allocate tasks. This mechanism foresees that each robot is endowed with a cognitive executive control modeling the stimuli identification, the relation between such stimuli and the robot tasks and, finally, the switching decision. The ability to selectively respond to several stimuli, and to inhibit inappropriate urges, focusing on the task at hand, are well-known to exist in humans as shifting and inhibition executive functions [2, 60]. Modeling the dynamic processes regulating such cognitive executive functions, in the cognitive executive control of the UGV, is crucial to assess a well-regulated behaviour of the robot as well as to balance the work load distributed among the multi-robot team. For this purpose, we developed a preliminary framework to model the robot processes, their yields, the stimuli occurrences, and the decision underlying the response to the stimuli. The proposed framework contributes to the state of the art of robot planning and high level control as it provides a novel perspective on the interaction robot-environment. The main advantage of this approach is the fact that robot control does not need to be designed a priori but it can be drawn by the interaction with users, who teach the robot when a stimulus is so and what the possible alternatives are. Indeed, a robot that has learned a stimulus-response strategy by several humans, will most certainly be more usable than a robot that either has no stimuli at all or has no strategies to respond to stimuli, other than failures. In fact, we take into account also the context in order to be able to establish a response cost and we exploit a 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. An overview of the proposed robot stimulus-response framework is illustrated in Figure 8. During the execution of a task, each active robot process yields a quantum of information with characteristic features; this quantum of information is called the *yield* of an active process. The features of the yield are used by the stimuli model to learn a function establishing whether a stimulus occurred, during the specific process execution, or not. If the stimulus occurs, the robot has to choose a possible response or it can inhibit the stimulus, and continue its task. Therefore the robot has to (1) identify the task that is a possible response to the stimulus and (2) decide whether to go on with the current task or to switch to the identified response task. The first issue is dealt with by filling a score matrix whose values are estimated via factorization. On the other hand, the switching decision is based on the pay-off of switching. This pay-off is computed considering the risk of continuing the current task, without taking into account the stimulus, and the effort required to fulfil the stimulus. The

effort is, in turn, computed considering two costs: (1) the cost to *reconfigure* the current robot state to the new state that switching would lead to and (2) the cost to resolve the *interference* due to the interruption of the current task. These costs are computed by considering the preconditions and effects of each action involved in both the processes to be interrupted and in the ones to be activated. To bind the information yielded by a process to the domain of reasoning a special functional is used, mapping the process terms of the representation language to the corresponding values of both the yields and the stimuli, at execution time. For more details and results see Annex §2.4.

1.3 Relation to the state-of-the-art

Multi-robot task allocation Multi-robot Task Allocation (MRTA) problems seek to allocate tasks to robots such that the cost to complete all tasks is minimised. An extensive amount of work has been proposed to address multi-robot task allocation [28, 101, 48]. Methods for solving MRTA problems can be classified into centralized and decentralized approaches, depending by the multi-robot system architecture [9]. Deterministic and heuristic approaches based on numerical optimization, dispatching rules, simulated annealing, tabu search, genetic algorithms, evolutionary algorithms have been developed for centralized architecture [78]. On the other hand, behavior-based approaches [75] and market-based approaches [19] have been proposed for distributed systems [23]. However, an important aspect of MRTA for multi-robot systems is to take uncertainties in available information into account when making decisions. Uncertainty can enter the problem at various levels; for example, at the robot level, it may appear as modelling uncertainty resulting from inaccurate models of the robot. Uncertainty also enters at the mission level as a result of limited prior knowledge about the environment. For example, an accurate model of the environment may not be available a priori, or the environment might change, making it difficult to decide on the best course of action. Therefore, the proposed approach for multi-robot task allocation attempts to model the uncertainty at the robot mission level by investigating how a class of stochastic reliability models can be integrated into typical task allocation frameworks. Task allocation frameworks employing reliability models can results in improved planning performance when uncertainty enters in task assignment.

Augmented Reality Augmented Reality (AR) is a recent emerging technology stemming from Virtual Reality (VR). AR develops environments where computer-generated 3D objects are blended (registered) onto a real world scene [3]. This technology has been applied in robotics applications such as maintenance [67], manual assembly [61], computer-assisted surgery [84], telerobotic control [58], monitoring [1, 7], robot programming

[6, 76, 99, 15], human-robot interaction [98], prototyping [32] and debugging [88, 17]. The mentioned approaches for both designing and evaluating robotics 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. Further, 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, the AR-based framework we have developed 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. The framework goes beyond the state-of-the-art concerning AR-based robotic applications, as the design exploits stochastic models 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.

3D autonomous navigation The developed framework for real-time 3D autonomous navigation for tracked vehicles in rescue environments progressed the current state-of-the-art concerning autonomous 3D mapping and navigation [56, 45, 16]. Several state-of-the-art approaches make use of 2.5D elevation maps [38] or full 3D voxel maps [45], or point clouds [97], yet attempt to reduce the problem dimensionality by planning the paths in a 2D navigation map, as also in [47, 63]. The developed framework overcomes this issues by allowing the UGV to directly plan paths within the 3D map of the environment. The framework also contributes to the state-of-the-art concerning 3D Semantic mapping, by building a basic categorization of the environment, specifically defined for navigation purposes, based on point cloud segmentation and labeling [71, 51, 4, 20].

Traversability analysis The proposed model for traversability cost estimation overcomes the main issues concerning scalability, resolution and continuity in domain representation raised from using discrete representations for traversability analysis, based either on elevation maps [35, 37, 77] or on multi-level surface maps [93, 43]. The proposed approach exploits the versatility in dealing with uncertainty of Gaussian Process regression [57, 33, 94] together with the commonality property of Multi-Task Learning [50] in order to handle both uncertainty and data incompleteness, in a statistically direct manner. Other popular learning based approaches proposed binary classification, specifying whether terrain is locally traversable or not

[46, 86, 90]. However, a binary classification is valuable for hazard avoidance but does not provide any additional information about the cost of regions and it might be unusable for path planning strategy guidance.

3D Trajectory tracking and control Several research efforts in robotics have been made to increase the level of autonomy of articulated tracked vehicles, focusing on adaptation [34, 65, 73], stability [70, 74], self-reconfiguration [39, 53], track-soil interaction [54, 100] and control [87, 22, 64, 8, 29]. Most of the proposed solutions focus on a specific task, for example, stair climbing rather than rough terrain traversal [34, 65, 44, 73, 53, 70]. Furthermore, most of the proposed approaches to the design of a trajectory tracking controller lack a unified framework for modeling the differential kinematics, accounting for all the DOFs of the AATV. The research work, described in Annex §2.5, advances the state-of-the-art by providing the baselines for designing a general adaptive robust 3D trajectory tracking controller for actively articulated tracked vehicles for both skid-steering and sub-tracks posture adaptation, independent of specific tasks.

3D motion planning for UAVs Generally speaking, the state-of-the-art in motion planning is represented by ROS, which provides a comprehensive framework for the autonomous navigation of a robot of any kind[55][82]. However, these algorithms were designed to work in two-dimensional space only. There are several approaches that use these navigation algorithms also on a UAV[27] but to the disadvantage of restricted degrees of freedom. Using this methods, a UAV will only be able to navigate safely in one plane. Another framework for path planning is given by MoveIt![40]. Previously designed for classic arm kinematics and related path planning, MoveIt! provides several interfaces, which allow the integration of any robot. A UAV can also be described as a single point in free space, which can reach any position or configuration, physical restrictions not taken into account.

Localization is an essential pre-stage for the autonomous navigation of a UAV. At this time, a UAV is located either by a motion capturing system or by its GPS sensor. But in GPS-denied environments there is no position information; hence, other sensors have to be evaluated for their localization capabilities. This has been done in a recent Master's thesis[96] and the result is presented in more detail in DR2.1.

Cognitive control In the research work, described in Annex §2.4 we applied the well-known concepts of shifting and inhibition in task switching to develop the executive cognitive control of the UGV [81, 91, 12, 62, 83]. The theories on executive cognitive control processes and task switching, initiated in neuroscience, have strongly influenced cognitive robotics architectures since the eighties, as for example the Norman and Shallice [69] ATA schema and the principles of goal directed behaviors in Newell [68]. However only

recently cognitive control is becoming a hot topic in cognitive robotics to model a robot complex behavior in unknown environments. Earliest studies in robotics have been carried within brain-actuated interaction [59], mechatronic [10], learning [41] and planning [25]. More recently, several studies have highlighted the need to model task switching to cope with adaptivity and ecological behaviors in a dynamic environment [11, 89, 95, 21, 18, 30, 79]. The major problem to be resolved in most of the cited works, as noted in [95], is the switching decision. Our research work in cognitive control contributes to this problem by proposing a solution to model this decision. In particular, we modeled the switching decision on the basis of the cost resulted from the interplay between the resources needed to reconfigure the robot internal state for the execution of a new task and the resources needed to resolve interference with the current robot internal state [62].

2 Annexes

2.1 Menna, Gianni, Ferri, Pirri (2014), " Real-time Autonomous 3D Navigation for Tracked Vehicles in Rescue Environments"

Bibliography Matteo Menna, Mario Gianni, Federico Ferri, Fiora Pirri. "Real-time Autonomous 3D Navigation for Tracked Vehicles in Rescue Environments." In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '14)*, page 696-702. Chicago, Illinois, 2014.

Abstract The paper presents a novel framework for 3D autonomous navigation for tracked vehicles. The framework takes care of clustering and segmentation of point clouds, traversability analysis, autonomous 3D path planning, motion planning and flippers control. Results illustrated in an experiment section show that the framework is promising to face harsh terrains. Robot performance is proved in three main experiments taken in a training rescue area, on fire escape stairs and in a non-planar testing environment, built ad-hoc to prove 3D path planning functionalities. Performance tests are also presented.

Relation to WP This work consolidates the main functionalities of the UGV, on top of which collaborative planning, in Task, T4.1, is developed.

Availablity Unrestricted. Included in the public version of this deliverable.

2.2 Ferri, Gianni, Menna, Pirri (2014), "Point Cloud Segmentation and 3D Path Planning for Tracked Vehicles in Cluttered and Dynamic Environments"

Bibliography Federico Ferri, Mario Gianni, Matteo Menna, Fiora Pirri. "Point Cloud Segmentation and 3D Path Planning for Tracked Vehicles in Cluttered and Dynamic Environments." In *Proceedings of the 3rd IROS Workshop on Robots in Clutter: Perception and Interaction in Clutter*. Chicago, Illinois, 2014.

Abstract The paper presents a framework for tracked vehicle 3D path planning in rough areas, with dynamic obstacles. The framework provides methods for real-time point cloud interpretation, segmentation and traversabilty analysis tacking also into account changes such as dynamic obstacles and provides a terrain structure interpretation. Moreover the paper presents $R_2A^\star$ an extended version of randomized $A^\star$ coping with difficult terrains and complex paths for non-holonomic robots.

Relation to WP This work contributes to build the bases for developing formal methods of collaborative planning, in Task, T4.1.

Availablity Unrestricted. Included in the public version of this deliverable.

2.3 Ferri, Gianni, Menna, Pirri (2015), "Fast path generation in 3D dynamic environments"

Bibliography Federico Ferri, Mario Gianni, Matteo Menna, Fiora Pirri. "Fast path generation in 3D dynamic environments." Submitted paper. Alcor Laboratory, DIAG "A. Ruberti", Sapienza University of Rome.

Abstract In this article we present a method for planning paths in 3D environments with dynamic obstacles, which is a challenging problem for autonomous robots operating in unstructured environments. Our contribution consists of dynamic obstacles removal and merging of 3D maps, traversability analysis within a probabilistic framework, and a domain specific randomized 3D path planner, which is a further refinement of $R_2A^\star$. Results show a performance comparison with other path planners, and evaluation of the dynamic obstacle removal algorithm.

Relation to WP This work improves the main functionalities of the UGV, which constitute the ground for collaborative planning in T4.1.

Availablity Restricted. Not included in the public version of this deliverable.

2.4 Gianni, Kruijff, Pirri (2014), " A Stimulus-Response Framework for Robot Control"

Bibliography Mario Gianni, Geert-Jan M. Kruijff, Fiora Pirri. "A Stimulus-Response Framework for Robot Control." In *ACM Transaction on Interactive Intelligent Systems*, Volume 4, Issue 4, January 2015.

Abstract We propose in this paper a new approach to robot cognitive control based on a stimulus-response framework that models both a robots stimuli and the robots decision to switch tasks in response or to inhibit the stimuli. In an autonomous system, we expect a robot to be able to deal with the whole system of stimuli and to use them to regulate its behavior in real-world applications. The proposed framework contributes to the state of the art of robot planning and high-level control in that it provides a novel perspective on the interaction between robot and environment. Our approach is inspired by Gibsons constructive view of the concept of a stimulus and by the cognitive control paradigm of task switching. We model the robots response to a stimulus in three stages. We start by defining the stimuli as perceptual functions yielded by the active robot processes and learned via an informed logistic regression. Then we model the stimulus-response relationship by estimating a score matrix, which leads to the selection of a single response task for each stimulus, basing the estimation on matrix low-rank factorization. The decision about switching takes into account both an interference cost and a reconfiguration cost. The interference cost weighs the effort of discontinuing the current robot mental state to switch to a new state, while the reconfiguration cost weighs the effort of activating the response task. A choice is finally made, based on the payoff of switching. Because processes play such a crucial role both in the stimulus model and in the stimulus-response model, and because processes are activated by actions, we address also the process model, which is built on a theory of action. The framework is validated by several experiments, exploiting a full implementation on an advanced robotic platform, and compared with two known approaches to replanning. Results demonstrate the practical value of the system in terms of robot autonomy, flexibility and usability.

Relation to WP This work contributes to build the bases for developing formal methods of collaborative planning, in Task, T4.1.

Availablity Unrestricted. Included in the public version of this deliverable.

2.5 Gianni, Ferri, Menna, Pirri (2014), "Adaptive Robust 3D Trajectory Tracking for Actively Articulated Tracked Vehicles (AATVs)"

Bibliography Mario Gianni, Federico Ferri, Matteo Menna, Fiora Pirri. "Adaptive Robust 3D Trajectory Tracking for Actively Articulated Tracked Vehicles (AATVs)." In *Journal of Field Robotics, Special Issue on Safety, Security, and Rescue Robotics (SSRR '2014)*, December 2014 (to appear in print).

Abstract A new approach is proposed for an adaptive robust 3D trajectory tracking controller design. The controller is modeled for actively articulated tracked vehicles (AATVs). These vehicles have active sub-tracks, called flippers, linked to the ends of the main tracks, to extend the locomotion capabilities in hazardous environments, such as rescue scenarios. The proposed controller adapts the flippers configuration and simultaneously generates the track velocities, to allow the vehicle to autonomously follow a given feasible 3D path. The approach develops both a direct and differential kinematic model of the AATV for traversal task execution correlating the robot body motion to the flippers motion. The benefit of this approach is to allow the controller to flexibly manage all the degrees of freedom of the AATV as well as the steering. The differential kinematic model integrates a differential drive robot model, compensating the slippage between the vehicle tracks and the traversed terrain. The underlying feedback control law dynamically accounts for the kinematic singularities of the mechanical vehicle structure. The designed controller integrates a strategy selector too, which has the role of locally modifying the rail path of the flipper end points. This serves to reduce both the effort of the flipper servo motors and the traction force on the robot body, recognizing when the robot is moving on an horizontal plane surface. Several experiments have been performed, in both virtual and real scenarios, to validate the designed trajectory tracking controller, while the AATV negotiates rubbles, stairs and complex terrain surfaces. Results are compared with both the performance of an alternative control strategy and the ability of skilled human operators, manually controlling the actively articulated components of the robot.

Relation to WP This work provided a model for planning and execution of the UGV, in the context of T4.1.

Availablity Restricted. Not included in the public version of this deliverable.

2.6 Gianni (2014), "Multilayered cognitive control for Unmanned Ground Vehicles"

Bibliography Mario Gianni. "Multilayered cognitive control for Unmanned Ground Vehicles." PhD Thesis, September 2014.

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 func-

tions 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.

Relation to WP This work provides an in-field study of the main issues concerning both the deployment and the development of Urban Search and Rescue robots. The work also proposes a solution to address these issues and provides an unified model for planning, reasoning and control of UGVs. This model is at the basis of multi-robot collaboration in T4.1.

Availablity Unrestricted. Available for download: `http://www.dis.uniroma1.it/~gianni/mydoc/PhDThesis.pdf`

2.7 Ferri, Gianni, Menna, Pirri (2015), "Dynamic obstacles detection and 3D map updating"

Bibliography Federico Ferri, Mario Gianni, Matteo Menna, Fiora Pirri. "Dynamic obstacles detection and 3D map updating." Submitted paper. Alcor Laboratory, DIAG "A. Ruberti", Sapienza University of Rome.

Abstract We present a real time method for updating a 3D map with dynamic obstacles detection. Moving obstacles are detected through ray-casting on spherical voxelization of point clouds. We evaluate the accuracy of this method on a point cloud dataset, suitably constructed for testing ray-surface intersection under relative motion conditions. Moreover, we show the benefits of the map updating in both robot path planning and navigation in real world environments, populated by moving people.

Relation to WP This work improves the main functionalities of the UGV, which constitute the ground for collaborative planning in T4.1.

Availablity Restricted. Not included in the public version of this deliverable.

References

[1] P. Amstutz and A.H. Fagg. Real time visualization of robot state with mobile virtual reality. In *Proc. ICRA*, volume 1, pages 241–247, 2002.

[2] A.R. Aron. The neural basis of inhibition in cognitive control. *The Neuroscientist*, 13:214–228, 2007.

[3] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. Recent advances in augmented reality. *IEEE Comput. Graph. Appl.*, 21(6):34–47, 2001.

[4] P. Biber, H. Andreasson, T. Duckett, and A. Schilling. 3d modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera. In *IROS*, volume 4, pages 3430–3435, 2004.

[5] A. Birolini. *Reliability Engineering: Theory and Practice*. Springer, 2013.

[6] R. Bischoff and A. Kazi. Perspectives on augmented reality based human-robot interaction with industrial robots. In *Proc. IROS*, volume 4, pages 3226–3231, 2004.

[7] V. Brujic-Okretic, J.-Y. Guillemaut, L.J. Hitchin, M. Michielen, and G.A. Parker. Remote vehicle manoeuvring using augmented reality. In *Proc. VIE*, pages 186 – 189, 2003.

[8] M. Burke. Path-following control of a velocity constrained tracked vehicle incorporating adaptive slip estimation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 97–102, 2012.

[9] Yifan Cai and Simon X. Yang. A survey on multi-robot systems. In *World Automation Congress (WAC)*, pages 1–6, June 2012.

[10] G. Capi. Robot task switching in complex environments. In *Advanced intelligent mechatronics, 2007 IEEE/ASME international conference on*, pages 1 –6, 2007.

[11] G. Capi, G. Pojani, and S.-I. Kaneko. Evolution of task switching behaviors in real mobile robots. In *Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on*, page 495, 2008.

[12] Cindy Chamberland and Sébastien Tremblay. Task switching and serial memory: Looking into the nature of switches and tasks. *Acta Psych.*, 136:137–147, 2011.

[13] S. Chiaverini, B. Siciliano, and O. Egeland. Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. *IEEE Transactions on Control Systems Technology*, 2(2):123–134, 1994.

[14] Han-Lim Choi, L. Brunet, and J.P. How. Consensus-based decentralized auctions for robust task allocation. *Robotics, IEEE Transactions on*, 25(4):912–926, 2009.

[15] J. W. S. Chong, S. K. Ong, A. Y. C. Nee, and K. Youcef-Youmi. Robot programming using augmented reality: An interactive method for planning collision-free paths. *Robotics and Computer-Integrated Manufactoring*, 25(3):689–701, June 2009.

[16] Francis Colas, Srivatsa Mahesh, François Pomerleau, Ming Liu, and Roland Siegwart. 3d path planning and execution for search and rescue ground robots. In *IROS*, pages 722–727, 2013.

[17] T.H.J. Collett and B.A. MacDonald. Augmented reality visualisation for player. In *Proc. ICRA*, pages 3954–3959, 2006.

[18] D.B. D'Ambrosio, J. Lehman, S. Risi, and K.O. Stanley. Task switching in multirobot learning through indirect encoding. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2802 –2809, 2011.

[19] M. Bernardine Dias. *Traderbots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2004. AAI3126923.

[20] James Diebel and Sebastian Thrun. An application of markov random fields to range sensing. In *NIPS*, pages 291–298. MIT Press, 2005.

[21] K.T. Durkee, C. Shabarekh, C. Jackson, and G. Ganberg. Flexible autonomous support to aid context and task switching. In *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2011 IEEE First International Multi-Disciplinary Conference on*, pages 204 –207, 2011.

[22] D. Endo, Y. Okada, K. Keiji, and K. Yoshida. Path following control for tracked vehicles based on slip-compensating odometry. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007*, pages 2871–2876, 2007.

[23] A. Farinelli, L. Iocchi, and D. Nardi. Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(5):2015–2028, 2004.

[24] Barbel Finkenstadt and Leonhard Held. *Statistical Methods for Spatio-Temporal Systems*. Chapman & Hall/CRC, 2006.

[25] Alberto Finzi and Fiora Pirri. Representing flexible temporal behaviors in the situation calculus. In *In Proc. of IJCAI*, 2005.

[26] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 4(1):23–33, Mar 1997.

[27] M Gäbel, T Krüger, and U Bestmann. Design of a quadrotor system for an autonomous indoor exploration. In *IMAV 2014: International Micro Air Vehicle Conference and Competition 2014, Delft, The Netherlands, August 12-15, 2014*. Delft University of Technology, 2014.

[28] Brian P. Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The Intl. J. of Robotics Research*, 23(9):939–954, 2004.

[29] M. Gianni, G. Gonnelli, A. Sinha, M. Menna, and F. Pirri. An augmented reality approach for trajectory planning and control of tracked vehicles in rescue environments. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 1–6, 2013.

[30] M. Gianni, P. Papadakis, F. Pirri, and M. Pizzoli. Awareness in mixed initiative planning. In *AAAI Fall Symposium Series*, 2011.

[31] Mario Gianni, Federico Ferri, and Fiora Pirri. Are: Augmented reality environment for mobile robots. In Ashutosh Natraj, Stephen Cameron, Chris Melhuish, and Mark Witkowski, editors, *Towards Autonomous Robotic Systems*, Lecture Notes in Computer Science, pages 470–483. Springer Berlin Heidelberg, 2014.

[32] B. Giesler, T. Salb, P. Steinhaus, and R. Dillmann. Using augmented reality to interact with an autonomous mobile platform. In *Proc. ICRA*, volume 1, pages 1009–1014, 2004.

[33] R. Hadsell, J. A. Bagnell, D. Huber, and M. Hebert. Accurate rough terrain estimation with space-carving kernels. In *Proc. of Robotics: Science and Systems*, 2009.

[34] D.M. Helmick, S.I. Roumeliotis, M.C. McHenry, and L. Matthies. Multi-sensor, high speed autonomous stair climbing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 733–742, 2002.

[35] M Herbert, C Caillas, Eric Krotkov, In So Kweon, and Takeo Kanade. Terrain mapping for a roving planetary explorer. In *ICRA*, pages 997–1002, 1989.

[36] J. P. How, C. Fraser, K. C. Kulling, L. F. Bertuccelli, O. Toupet, L. Brunet, A. Bachrach, and N. Roy. Increasing autonomy of UAVs. *Robotics and Automation Magazine, IEEE*, 16(2):43–51, 2009.

[37] A. Howard, Michael J. T., L. Matthies, B. Tang, A. Angelova, and E. Mjolsness. Towards learned traversability for robot navigation: From underfoot to the far field. *J. Field Robotics*, 23(11-12):1005–1017, 2006.

[38] Karl Iagnemma, Frank Genot, and Steven Dubowsky. Rapid physics-based rough-terrain rover planning with sensor and control uncertainty. In *ICRA*, volume 3, pages 2286–2291, 1999.

[39] Karl Iagnemma, Adam Rzepniewski, Steven Dubowsky, and Paul Schenker. Control of robotic vehicles with actively articulated suspensions in rough terrain. *Autonomous Robots*, 14(1):5–16, 2003.

[40] Sachin Chitta Ioan A. Sucan. Moveit! http://moveit.ros.org. [Online; Checked 2015-01-28].

[41] Masato Ito, Kuniaki Noda, Yukiko Hoshino, and Jun Tani. Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model. *Neural Networks*, 19:323–337, 2006.

[42] L. B. Johnson, S. Ponda, H.-L. Choi, and J. P. How. Asynchronous decentralized task allocation for dynamic environments. In *Proceedings of the AIAA Infotech@Aerospace Conference*, St. Louis, MO, March 2011.

[43] D. Joho, C. Stachniss, P. Pfaff, and W. Burgard. Autonomous exploration for 3D map learning. In *Autonome Mobile Syst.*, 2007.

[44] A. Kalantari, E. Mihankhah, and S.A.A. Moosavian. Safe autonomous stair climbing for a tracked mobile robot using a kinematics based controller. In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1891–1896, 2009.

[45] Jochen Klaess, Joerg Stueckler, and Sven Behnke. Efficient mobile robot navigation using 3d surfel grid maps. In *ROBOTIK*, pages 1–4, May 2012.

[46] K. Konolige, M. Agrawal, M. R. Blas, R. C. Bolles, B. P. Gerkey, J. Sol, and A. Sundaresan. Mapping, navigation, and learning for off-road traversal. *J. Field Robotics*, 26(1):88–113, 2009.

[47] Kurt Konolige, Motilal Agrawal, Robert C. Bolles, Cregg Cowan, Martin Fischler, and Brian Gerkey. Outdoor mapping and navigation using stereo vision. In *Exp. Rob.*, volume 39, pages 179–190. Springer Berlin Heidelberg, 2008.

[48] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *Int. J. Rob. Res.*, 32(12):1495–1512, 2013.

[49] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001, 2000.

[50] Abhishek Kumar and Hal Daume. Learning task grouping and overlap in multi-task learning. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning*, pages 1383–1390, New York, NY, USA, 2012. ACM.

[51] Jean-François Lalonde, Nicolas Vandapel, Daniel F Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *JFR*, 23(10):839–861, 2006.

[52] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, 1998.

[53] N. Li, S. Ma, B. Li, M. Wang, and Y. Wang. An online stair-climbing control method for a transformable tracked robot. In *Proceeding of the IEEE International Conference on Robotics and Automation*, pages 923–929, 2012.

[54] Y. Liu and G. Liu. Track–stair interaction analysis and online tipover prediction for a self-reconfigurable tracked mobile robot climbing stairs. *IEEE/ASME Transactions on Mechatronics*, 14(5):528–538, 2009.

[55] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *International Conference on Robotics and Automation*, 2010.

[56] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian P. Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *ICRA*, pages 300–307, 2010.

[57] S. Martin, L. Murphy, and P. Corke. Building large scale traversability maps using vehicle experience. *Exp. Robotics*, 88:891–905, 2013.

[58] P. Milgram, S. Zhai, D. Drascic, and J. Grodski. Applications of augmented reality for human-robot communication. In *Proc. IROS*, volume 3, pages 1467–1472, 1993.

[59] Jose del R. Millan, Frederic Renkens, Josep Mourino, and Wulfram Gerstner. Brain-actuated interaction. *Artificial Intelligence*, 159:241–259, 2004.

[60] E.K. Miller and J.D. Cohen. An integrative theory of prefrontal cortex function. *Annual Review of Neuroscience*, 24:167–202, 2007.

[61] J. Molineros and R. Sharma. Computer vision for guiding manual assembly. In *Assembly and Task Planning IEEE Int. Symposium*, 2001.

[62] Stephen Monsell. Task switching. *Trends in Cognitive Sciences*, 7(3):134 – 140, 2003.

[63] Michael Montemerlo, Sebastian Thrun, Hendrik Dahlkamp, and David Stavens. Winning the darpa grand challenge with an ai robot. In *AAAI*, pages 17–20, 2006.

[64] S.A.A. Moosavian and A. Kalantari. Experimental slip estimation for exact kinematics modeling and control of a tracked mobile robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 95–100, 2008.

[65] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, D. M. Helmick, and L. Matthies. Autonomous stair climbing for tracked vehicles. *International Journal of Robotic Research*, 26(7):737–758, 2007.

[66] Radford M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

[67] U. Neumann and A. Majoros. Cognitive, performance, and systems issues for augmented reality applications in manufacturing and maintenance. In *Proc. IEEE Virtual Reality Annual International Symposium*, pages 4 –11, 1998.

[68] A. Newell. *Unified theories of cognition.* Harvard University Press, 1990.

[69] D. A. Norman and T. Shallice. *Consciousness and Self-Regulation: Advances in Research and Theory*, volume 4, chapter Attention to action: Willed and automatic control of behaviour. Plenum Press, 1986.

[70] M. Norouzi, J.V. Miro, and G. Dissanayake. Planning high-visibility stable paths for reconfigurable robots on uneven terrain. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2844–2849, 2012.

[71] A. Nuchter, H. Surmann, and J. Hertzberg. Automatic model refinement for 3d reconstruction with mobile robots. In *3DIM*, pages 394–401, 2003.

[72] P. O'Connor and A. Kleyner. *Practical Reliability Engineering*. Wiley, 2011.

[73] Y. Okada, K. Nagatani, K. Yoshida, S. Tadokoro, T. Yoshida, and E. Koyanagi. Shared autonomy system for tracked vehicles on rough terrain based on continuous three-dimensional terrain scanning. *Journal of Field Robot.*, 28(6):875–893, 2011.

[74] Panagiotis Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373–1385, 2013.

[75] L.E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, 1998.

[76] T. Pettersen, J. Pretlove, C. Skourup, T. Engedal, and T. Lkstad. Augmented reality for programming industrial robots. In *Proc. IEEE/ACM Int. Symp. on Mixed and Augmented Reality*, pages 319–320, 2003.

[77] P. Pfaff, R. Triebel, and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *Int. J. Rob. Res.*, 26(2):217–230, 2007.

[78] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

[79] Fiora Pirri and Matia Pizzoli. *Knowing, Reasoning, and Acting Essays in Honour of Hector J. Levesque*, volume 16, chapter Inference about Actions: Levesques view on action ability and Dirichlet processes. Kings College Publications, 2011.

[80] J. Pujol. The solution of nonlinear inverse problems and the levenberg-marquardt method. *Geophysics*, 72(4):W1–W16, 2007.

[81] R. Rogers and S. Monsell. The costs of a predictable switch between simple cognitive tasks. *J. of Exp. Psychology: General*, 124:207–231, 1995.

[82] Ros navigation stack. http://wiki.ros.org/navigation. [Online; Checked 2015-01-28].

[83] J. Rubinstein, D.E. Meyer, and E.Jeffrey. Executive control of cognitive processes in task switching. *J. of Exp. Psych.: Human Perception and Performance*, 27(4):763–797, 2001.

[84] Raj Shekhar, Omkar Dandekar, Venkatesh Bhat, Mathew Philip, Peng Lei, Carlos Godinez, Erica Sutton, Ivan George, Steven Kavic, Reuben Mezrich, and Adrian Park. Live augmented reality: a new visualization method for laparoscopic surgery using continuous volumetric computed tomography. *Surgical Endoscopy*, 24:1976–1985, 2010.

[85] T. Shima and S. Rasmussen. *UAV Cooperative Decision and Control*. Society for Industrial and Applied Mathematics, 2009.

[86] M. Shneier, W. Shackleford, T. Hong, and T. Chang. Performance evaluation of a terrain traversability learning algorithm in the darpa lagr program. Technical report, DTIC Document, 2009.

[87] S. Steplight, G. Egnal, Sang-Hack Jung, D.B. Walker, C.J. Taylor, and J.P. Ostrowski. A mode-based sensor fusion approach to robotic stair-climbing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 1113–1118, 2000.

[88] Michael Stilman, Philipp Michel, Joel Chestnutt, Koichi Nishiwaki, Satoshi Kagami, and James Kuffner. Augmented reality for robot development and experimentation. Technical report, Robotics Institute, 2005.

[89] S. Suzuki, T. Sasaki, and F. Harashima. Visible classification of task-switching strategies in vehicle operation. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, pages 1161 –1166, 2009.

[90] A. Talukder, R. Manduchi, R. Castano, K. Owens, L. Matthies, A. Castano, and R. Hogg. Autonomous terrain characterisation and modelling for dynamic control of unmanned vehicles. In *IROS*, volume 1, pages 708–713 vol.1. IEEE, 2002.

[91] Thea and Ionescu. Exploring the nature of cognitive flexibility. *New Ideas in Psych.*, 30(2):190 – 200, 2012.

[92] R. Thibaux and M. Jordan. Hierarchical beta processes and the indian buffet process. In *Proceedings of the 11th Conference on Artificial Intelligence and Statistics (AISTAT)*, Puerto Rico, 2007.

[93] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *IROS*, 2006.

[94] S. Vasudevan, F. T. Ramos, E. Nettleton, and H. F. Durrant-Whyte. Large-scale terrain modeling from multiple sensors with dependent gaussian processes. In *IROS*, 2010.

[95] J. Wawerla and R.T. Vaughan. Robot task switching under diminishing returns. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5033 –5038, 2009.

[96] Stefan Wilkes. 3D Navigation for UAVs in GPS denied environments. Master's thesis, Westphalian University of Applied Sciences Gelsenkirchen, January 2015.

[97] Oliver Wulf, Christian Brenneke, and Bernardo Wagner. Colored 2d maps for robot navigation with 3d sensor data. In *IROS*, volume 3, pages 2991–2996, 2004.

[98] J.E. Young, E. Sharlin, and J.E. Boyd. Implementing bubblegrams: The use of haar-like features for human-robot interaction. In *Proc. CASE*, pages 298–303, 2006.

[99] M.F. Zaeh and W. Vogl. Interactive laser-projection for programming industrial robots. In *Proc. ISMAR*, pages 125 –128, 2006.

[100] Karel Zimmermann, Petr Zuzanek, Michal Reinstein, and Vaclav Hlavac. Adaptive traversability of unknown complex terrain with obstacles for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5177–5182, 2014.

[101] Robert Michael Zlot. *An Auction-Based Approach to Complex Task Allocation for Multirobot Teams.* PhD thesis, Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, December 2006.

[102] G. Zolynski, C. Schank, and K. Berns. Point cloud gathering for an autonomous bucket excavator in dynamic surroundings. In *Proc. of the Comm. Vehicle Techn. Symp.*, 2012.

2014 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS 2014)
September 14-18, 2014, Chicago, IL, USA

Real-time Autonomous 3D Navigation for Tracked Vehicles in Rescue Environments

Matteo Menna and Mario Gianni and Federico Ferri and Fiora Pirri
ALCOR, Vision, Perception and Cognitive Robotics Laboratory
DIAG, University of Rome 'La Sapienza', Italy
{menna, gianni, ferri, pirri}@dis.uniroma1.it

***Abstract*— The paper presents a novel framework for 3D autonomous navigation for tracked vehicles. The framework takes care of clustering and segmentation of point clouds, traversability analysis, autonomous 3D path planning, motion planning and flippers control. Results illustrated in an experiment section show that the framework is promising to face harsh terrains. Robot performance is proved in three main experiments taken in a training rescue area, on fire escape stairs and in a non-planar testing environment, built ad-hoc to prove 3D path planning functionalities. Performance tests are also presented.**

I. INTRODUCTION

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 (see Figure 1). 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 paper we propose a real time 3D path and motion planner that tries to overcome some limitation of the current path planners.

The paper is organized as follows. The next section briefly summarizes the state of the art. Section III introduces the proposed approach for 3D motion planning. Section IV describes our approach for estimating the contact between the flippers and the surface. Section V describes the segmentation and clustering of the point cloud. Section VI describes the estimation of the inflated regions of the environment as well as the estimation of a traversability graph leading to a path, whose generation is discussed in Section VII.

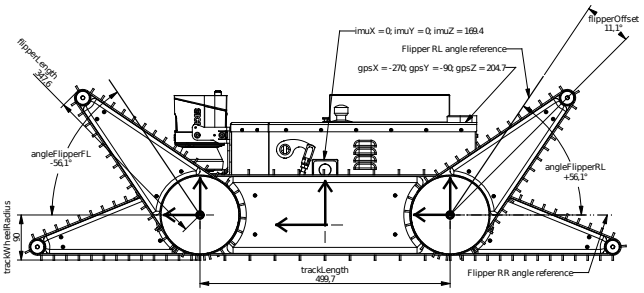


Fig. 1. The robotic platform.

Finally, Section VIII describes the experiments proving the effectiveness of the proposed framework. An overview of this framework is shown in Figure 2. Point cloud, coming from the rolling laser is filtered to remove the outliers. Then, both normals and principal curvatures are estimated for segmenting, labeling and estimating the inflated regions of the filtered point cloud. The resulting 3D labeled map is used for building a traversability graph of the environment, enabling the path planner to generate paths toward goal locations. Given both a path and the 3D labeled map, the flipper position controller relies on a contact sensor model to ensure the touch of the flippers to the surface on which the path lies.

II. STATE OF THE ART

We can classify existing literature about mapping and navigation in 2D and 3D approaches. 2D path planning has been widely studied and can be considered as a mostly solved problem [1]. Approaches going beyond 2D representations use 2.5D elevation maps [2] or full 3D voxel maps [3], or point clouds [4], yet attempt to reduce the problem dimensionality by planning the paths in a 2D navigation map, as also in [5], [6]. The so obtained cost maps can be coupled with a cost aware RRT implementation as in [7], [8] or a grid based planner as in [9]. A method based on elevation maps has been proposed in [10]. In all these solutions environment representations are intrinsically 2D and they cannot represent overhanging objects and environments with multiple overlapping levels.

3D Semantic mapping has been studied in [11], [12] where a 3D laser sensor has been used to create a map in which semantic labels are added in order to identify architectural

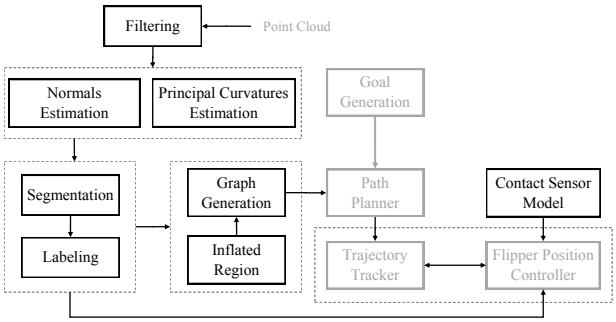
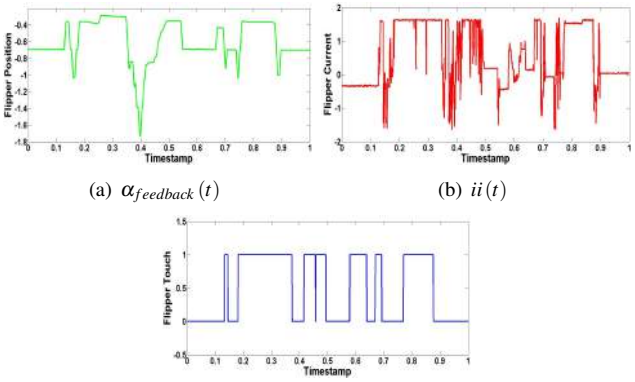


Fig. 2. Framework overview. Note: solid blocks denote authors' contribution, while shaded blocks denote third party components.



(a) $\alpha_{feedback}(t)$ (b) $ii(t)$

(c) $l(t)$

Fig. 3. (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.

elements via geometric features. Research on 3D mapping and reconstruction for mobile robots using laser range finder has been also explored in [13], [14]. These representations do not make use of any additional semantic label and have not proved to work in real time with a fully autonomous robot system, likewise [15], since it is possible to show that stairs are 2D reducible and flipper angles can be set in advance.

Approaches using vision, such as [16], [5], are not directly comparable to those using 3D sensors as requiring more computing effort.

III. 3D MOTION PLANNING

In this section we describe the main components of the 3D motion controller designed for a tracked vehicle such as the one illustrated in Figure 1. The robot configuration state is defined by the vector $\mathbf{q} = (x_r, y_r, z_r, \psi, \phi, \theta, v_r, \omega_r, \alpha_1, \ldots, \alpha_4)^T$, with $x_r, y_r, z_r, \psi, \phi, \theta$ the 6D pose of the robot, v_r and ω_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. The trajectory tracking controller receives as input a path $\mathscr{P}$ (see Section VI), generated on the 3D labeled map of the environment (see Section V) and computes the steering commands to allow the robot to follow the given path $\mathscr{P}$. The control strategy underlying the trajectory tracking controller is based on input-output linearization via feedback [17], [18]. The flippers position controller receives the desired flipper positions $\boldsymbol{\alpha}_d(t) = (\alpha_1(t), \ldots, \alpha_4(t))^T$, as input, generating suitable internal speed commands to asymptotically stabilize to zero the flippers position error, on the basis of a proportional-derivative (PD) control law. The positions $\boldsymbol{\alpha}_d(t)$ are selected from a set of predefined configurations, depending on both the current robot attitude and the information provided by the 3D labeled map of the environment [15]. These configurations allows the robot to climb stairs and to overcome both positive and negative obstacles. However, during the control loop, the flipper position controller is not able to detect, after positioning the flippers according to the selected posture, whether the flippers effectively are either in contact with the surface or with the obstacle to be surmounted. Still, the flippers are neither endowed with contact sensors nor with proximity sensors. To face this limit, the flipper position controller integrates a statistical model assessing the touch and the detach of the flippers with the surface. This model is based on a function, learned from a set of features, extracted from the direct measurements of both the actual angles $\boldsymbol{\alpha}_{feedback}(t)$ of the flippers (see Figure 3(a)) and the electrical currents $\mathbf{ii}(t)$ of the flipper servo motors (see Figure 3(b)). During the control loop, the controller adjusts the flippers posture $\boldsymbol{\alpha}_d(t)$, depending on the feedback provided by the contact sensor model.

IV. CONTACT SENSOR MODEL

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

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 $\alpha_{feedback}(t)$ and $ii(t)$ 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 Figure 3(c)). This data have been further interpolated to build a data-set $D = \{\langle \alpha(t), ii(t), l(t) \rangle, t = 0, \ldots, T\}$ suitable for the training of the contact sensor model.

Feature representation The data-set $D = \{\langle \alpha(t), ii(t), l(t) \rangle, t = 0, \ldots, T\}$ has been processed to

extract the relevant features for tuning the parameters of the function h_{θ^*} 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:

$$
\begin{aligned}
x_1 &= \frac{1}{t-t'} \sum_{k=t'}^{t} |ii(k)| \\
x_2 &= \frac{1}{t-t'} \sum_{k=t'}^{t} \left| \frac{\alpha(k+1) - \alpha(k-1)}{2} \right| \\
x_3 &= \operatorname{sgn}\left(\sum_{k=t'}^{t} ii(k) \right) \cdot \operatorname{sgn}\left(\sum_{k=t'}^{t} \frac{\alpha(k+1) - \alpha(k-1)}{2} \right) \\
x_4 &= \frac{1}{t-t'} \sum_{k=t'}^{t} |ii_{TDF-II}(k)|
\end{aligned}
$$

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 $\mathbf{x} = (x_1, \ldots, x_4)^T$ 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 $D' = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ is the data-set of features-labels pairs generated from D, with $\mathbf{x}_i \in X \subseteq \mathbb{R}^4$ and $y_i \in L \equiv \{-1, 1\}$.

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

$$
R(\theta) = \int |h_{\boldsymbol{\theta}}(\mathbf{x}) - y| P(\mathbf{x}, y) d\mathbf{x} dy \tag{1}
$$

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 $h_{\boldsymbol{\theta}^*}$ can be considered as the problem of finding the decision surface that better separates the data. This can be formulated as follows:

$$
\begin{aligned}
\underset{\boldsymbol{\theta} \in \Theta}{\text{maximize}} \quad & \sum_{i=1}^{N} \theta_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \theta_i \theta_j K(\mathbf{x}_i, \mathbf{x}_j) \\
\text{subject to} \quad & \sum_{i=1}^{N} y_i \theta_i = 0, 0 \leq \theta_i \leq 1, \forall i = 1, \ldots, N.
\end{aligned}
$$

Here $K(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^T \mathbf{x}_j + 1\right)^d$. 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 $\mathbf{x}$, the decision function $h_{\boldsymbol{\theta}^*}$ modeling the contact of the flipper with the underlying surface classifies $\mathbf{x}$ as follows

$$
h_{\boldsymbol{\theta}^*}(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{N_S} y_i \theta_i^* K(\mathbf{s}_i, \mathbf{x}) \right) \tag{2}
$$

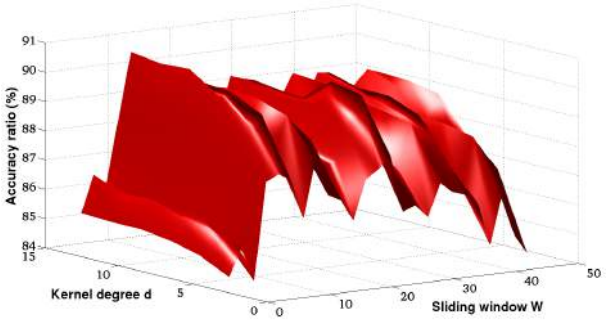


Fig. 4. Accuracy ratio of the contact sensor model as both the degree d of the polynomial kernel and the sliding time window W change

Here $\mathbf{s}_i$ are the support vectors and θ_i^* are the optimal parameter values associated to $\mathbf{s}_i$, 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 4 shows that for $d = 13$ and $W = 5$ the model correctly classifies 90.7% of the contacts of the flipper with the underlying surface.

V. Point Cloud Segmentation and Labeling

In this section we describe a real time point cloud *PC* segmentation and labeling, registered by an ICP-based SLAM [19]. The point cloud *PC* stores the geometric position (x, y, z) of each point $\mathbf{p} \in PC$, approximating a surface M. The objective is to establish traversability of the surface for the path control, requiring real-time computation. To this end we define just four categories for traversability: *ground, walls, ramp or stairs, surmountable obstacle*. These categories have been specified 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; $80cm \times 80cm$ cavities. Here we assume that segmentation is always performed on the incoming laser measures, and we consider the current one, in so avoiding to parametrize points with time.

We propose here a novel approach to simple categories segmentation based on geometric features. The approach is specified by the following steps (1) patch construction and filtering; (2) estimation of the normals and of the principal curvatures of the surface at all points and (3) segmentation and labeling of the *PC*.

Patch construction and Filtering An initial points filtering is already provided by the ICP algorithm, here we are mainly concerned with constructing a surface patch for each PC region, used for geometric features computation, and to discard ill-conditioned neighborhoods. A patch is defined by a mapping $\psi : (x, y) \mapsto M$, with $\psi(x, y) = \mathbf{p} \in \mathbb{R}^3$, and $(x, y) \in \mathbb{R}^2$, the parameters of the mapping. To simplify we denote $\psi(x, y)$ by $\mathbf{x}$ intending also the coordinates of the point $\mathbf{p}$ on the surface M; similarly, we identify the parameters with the x and y coordinates. The patch can be obtained by transforming the point representation of *PC* into

a voxel grid, building a suitable neighborhood for points that can be reordered into surface patches of contiguous points, and ensuring that there is none or little overlapping between patches. First we decimate the *PC* of a factor of 0.9 to obtain the downsampled PC_s, and then we initialize the neighborhood of each point $\mathbf{q}$ in PC_s with k-means taking the $\mathbf{q}$ as seeds, using only the points coordinates together with the Euclidean distance. Since k-means does not take care of ill-conditioned clusters we adjust each cluster by expanding it or reducing it, by splitting or merging, according to the cluster spatial frequency $\rho = {}^{|H|}/_{r^3}$, where $|\cdot|$ is the cardinality of the cluster H and $r = mean(d(H))$ is the radius defined by the mean distance within the cluster. An optimal density $\rho^\star$ is given by a radius $r_{min} = 0.5\sqrt{2}m$, (m, meters), and a neighborhood of 81 points. If $\rho < {}^{\rho^\star}/_2$ then r is increased to catch more points joining with the cluster that ensures to keep the optimal dimension, among the closer ones. To control the radius growth and filter out noise, a noise function is specified according to the spatial frequency of points as follows. Let $\varphi(H) = exp(-\pi(r_{min} - r))$, then the probability that a neighborhood H is affected by noise is:

$$p(noise(H)) = 1 - \varphi(H) \tag{3}$$

The noise function is clearly zero when $r = r_{min}$. Neighborhoods with noise value greater than 0.5 are removed from the *PC*. Once the clusters are stabilized then, each of them is labeled into a voxel grid, possibly resorting to interpolation, mostly required at the boundary of a patch, so that close points are contiguous. The grid representation leads to the surface patch representation of the three matrices $\mathbf{X}$, for the x coordinates of the points in the cluster, $\mathbf{Y}$ for the y coordinates, and $\mathbf{Z}$ for $\psi(x,y)$. These matrices have size $m \times n$ each.

Normal and principal curvature estimation Having transformed the original *PC* into a number of surface patches well approximating the surface M, using the neighbors construction and the grid, a very precise estimation of the normals, and likewise of the principal curvatures, can be obtained quite simply, using first and second derivatives. This allows us to avoid the computation based on least mean square (see for example [20], [21]), which can be quite sensitive to noise and it is computationally very expensive, since it requires to compute neighborhoods for each point of the cloud and to perform SVD decomposition.

Here we consider finite-size linear-phase separable kernels, introduced by [22], for first and second order differentiation of discrete multidimensional signals. These kernels are used to convolve the matrices $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ to obtain the derivatives, by first expanding the matrices at the boundaries. Then, differentiation for each coordinate x and y is obtained by collecting the convolution of each matrix in each direction. Namely, let f_x, g_x, h_x be the first-order derivatives of $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$, respectively, with respect to x, obtained with the convolution kernel, these are matrices of size $m \times n$ too. Therefore $\mathbf{x}_x$ is the tensor $(f_x, g_x, h_x)^\top$ and, analogously, $\mathbf{x}_y = (f_y, g_y, h_y)^\top$. In an analogous way we obtain the second-order derivatives $\mathbf{x}_{xx}$, $\mathbf{x}_{yy}$, $\mathbf{x}_{xy}$. Having defined the patches in explicit form, it is now straightforward to obtain the normals $\mathbf{n}s$ for all the points in a patch, by simply applying the cross product to the first derivative tensors, obtaining the tensor $3 \times n \times m$:

$$\pm \begin{pmatrix} g_x \cdot h_y - g_y \cdot h_x \\ f_y \cdot h_x - f_x \cdot h_y \\ f_x \cdot g_y - f_y \cdot g_x \end{pmatrix} \tag{4}$$

and normalizing to get the tensor $\mathbf{N}$ of unit normals, noting that in the above equation $\cdot$ is the element-wise matrix multiplication. Note that the sign can be actually determined by simply computing the smallest angle between the obtained normals and the laser source position, say $\mathscr{S}$. Namely, we look for the angle θ minimizing $\cos^{-1}(\mathbf{s}^\top \mathbf{n}'_{3\times mn}) / \|\mathbf{s}\| \|\mathbf{n}'_{3\times mn}\|$, where

$$\mathbf{s} = (\mathscr{S} \otimes \mathbf{1}_{1\times nm} - \mathbf{x}'_{3\times nm}) / \|\mathscr{S} \otimes \mathbf{1}_{1\times nm} - \mathbf{x}'_{3\times nm}\|$$

with $\mathscr{S}$ the source position, $\otimes$ the Kronecker product, $\mathbf{x}'_{3\times nm}$ and $\mathbf{n}'_{3\times nm}$ tensors of size $(3 \times n \times m)$ reshaped into a $3 \times nm$ matrix.

As for the principal curvatures, at each point $\mathbf{p}$ in a patch, we exploit the fact that with the first and second derivatives and the normals tensor $\mathbf{N}$ available, we can obtain the following elements, named according the I and II fundamental forms:

$$\begin{array}{lll} E = \Sigma_{dim=3} \mathbf{x}_x \cdot \mathbf{x}_x & F = \Sigma_{dim=3} \mathbf{x}_x \cdot \mathbf{x}_y & G = \Sigma_{dim=3} \mathbf{x}_y \cdot \mathbf{x}_y; \\ L = \Sigma_{dim=3} \mathbf{x}_{xx} \cdot \mathbf{N} & M = \Sigma_{dim=3} \mathbf{x}_{xy} \cdot \mathbf{N} & N = \Sigma_{dim=3} \mathbf{x}_{yy} \cdot \mathbf{N} \end{array} \tag{5}$$

In other words E, F, G, L, M, N are all matrices $m \times n$. Thus, k is computed as the solution of the quadratic equation

$$(EG - F^2)k^2 + (2FM - EN - GL)k + (LN - M^2) \tag{6}$$

And it is well-known that the discriminant of the above equation is greater or equal zero, therefore there are either two distinct real roots k_1 and k_2, namely the two principal curvatures, or a single one. While at umbilical points $L/M = M/F = N/G$. Given the principal curvatures we can also obtain both the mean curvature $(1/2)(k_1 + k_2)$ and the Gaussian curvature $k_1 k_2$. We observe that the only computational difficulty here is the construction of the patches. The main difficulties coming from far points, which are those mostly requiring merging and dropping, therefore it might be better for those noisy regions to delay the patch construction, according to the noise function defined in (3).

Segmentation and Labeling Segmentation still uses the matrices $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ yet all integrated into the voxel grid. We note that without hampering the information for the categories, the kernel convolution included a significant smoothing of the matrix $\mathbf{Z}$. Then we can obtain from the normals, the Gaussian curvature and the principal curvatures an initialization of the categories as follows. Let $|\mathbf{n}| = (|n_1|, |n_2|, |n_3|)^\top$ be the unit normal absolute value at a point $\mathbf{p}$, let $K = k_1 k_2$ be the Gaussian curvature, with k_1 and k_2 the principal curvatures, let $\theta_1 = 0.17$ and $\theta_2 = 0.95$ then the following table illustrates the initialization values:

	Parameters	
Classes	$\lvert n_1\rvert,\lvert n_2\rvert,\lvert n_3\rvert$	K, $k_1,k_2,\mathbf{Z}$
ground	$\lvert n_1\rvert,\lvert n_2\rvert<0.01$, $\lvert n_3\rvert\geq 0.8$	$\lvert K\rvert<0.1$
walls	$\lvert n_2\rvert,\lvert n_3\rvert<0.01$, $\lvert n_1\rvert\geq 0.8$	$\lvert K\rvert<0.1$
ramp	$\theta_1\leq\lvert n_3\rvert\leq\theta_2$	
	$\lvert n_1\rvert\leq 0.1$, $\theta_1\leq\lvert n_2\rvert\leq\theta_2$	$\lvert k_2\rvert<10^{-2}$, $\lvert k_1\rvert\geq 1$
	$\lvert n_2\rvert\leq 0.1$ $\theta_1\leq\lvert n_1\rvert\leq\theta_2$	$\lvert k_1\rvert<10^{-2}$, $\lvert k_2\rvert\geq 1$
surmountable obstacles	$\theta_1\leq\lvert n_3\rvert\leq\theta_2$	$\lvert K\rvert\geq 1$, $\mathbf{z}>0.3m$.

Because of noise these parameters produce scattered segmentation, to even the neighborhoods of the above defined categories we introduce an energy functional for each class as follows. All points in a class are assigned kinetic energy E_k; a stop condition is given by a function $g(\mathbf{p})$ which stops the front collecting points, at a given direction of growth, whenever there is a jump in energy level. Let the normal for the given classes be specified by $\mathbf{n}_e$ with $e = \{ground, wall, ramp, surm.obstacle\}$. The energy level for class e is $E_{\ell_e}(\mathbf{p}) = \cos^{-1}(\mathbf{n}^\top \mathbf{n}_e)$, with $\mathbf{n}$ the normal at the current point $\mathbf{p}$ considered, with $\mathbf{p}_0$ a starting point. Let κ denotes either the Gaussian curvature or the principal curvatures, according to the class and its constraints considered. Let:

$$\eta(\mathbf{p}_0,\mathbf{p}) = s(|\kappa(\mathbf{p}) - \kappa(\mathbf{p}_0)|) + s(|E_{\ell_e}(\mathbf{p}_0) - E_{\ell_e}(\mathbf{p})|) \quad (7)$$

Here s is the logistic function $y = c/(1+a\exp(-bx))$, with $b = 25$ and $a = \pi^5$. 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. Let

$$g(\mathbf{p}) = E_k(\mathbf{p}_0) - \eta(\mathbf{p}_0,\mathbf{p}) \quad (8)$$

Points $\mathbf{p}$, around $\mathbf{p}_0$, are collected as far as $g(\mathbf{p}) > 0$: Therefore when $g \leq 0$ then the front expansion stops for that point. To resume:

1) *Input*: the set of surface points, the normals, the curvatures.
2) Assign kinetic energy E_k to the chosen points $\mathbf{p}_0$ in the class.
3) Compute a path to a farthest point such that for every point in the path the difference in energy level and curvature is low, namely $\eta(\mathbf{p}_0,\mathbf{p}) \leq 0.1$.
4) Move the neighborhood front in all directions, checking for each direction the stopping criterion $g(\mathbf{p})$, eq. (8).
5) Remove the points collected into the formed clusters, if the set of surface points is empty stop, else go to item 2).
6) *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 [23]. The final obtained clusters are labeled according to the specified categories.

VI. Graph generation

In this section we describe how the connectivity and traversability graph is obtained. The proposed method requires three steps: (1) estimation of the inflated regions the environment; (2) definition of a graph G, connecting the points of the labeled cluster (3) weighting of the the graph, returning the traversability graph. These steps are illustrated in the following.

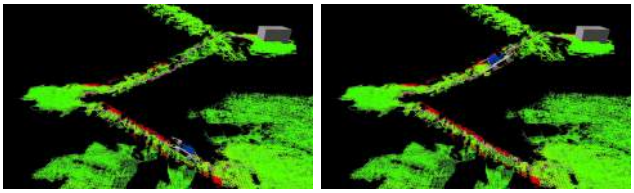


Fig. 5. Weighted graph and path (in magenta) on the fire escape stairs. The big gray cube is the goal. Graph and path computation time are described in Section VIII.

Inflated region estimation Let $\mathscr{A}$ be the set of clusters, labeled with the class they belong to. Let $\mathscr{A}' = \{(C_k, c_k) : c_k = wall\} \subseteq \mathscr{A}$ and $\mathscr{A}^\star = \mathscr{A} \backslash \mathscr{A}'$. We introduce the points belonging to inflated regions $\mathscr{B}$, which are used to specify a feasible path, beside the flippers control (described in Section III). Let $u : PC \mapsto PC$ be the function:

$$u(\mathbf{p}) \triangleq \underset{\mathbf{p}_j \in C_j}{\arg\min}\{\left\|\mathbf{p} - \mathbf{p}_j\right\|\} \quad \text{s.t. } (C_j, c_j) \in \mathscr{A}' \quad (9)$$

Then the set of inflated region points $\mathscr{B}$ is defined as follows:

$$\mathscr{B} = \{\mathbf{p} \ : \mathbf{p}=(\mathbf{p}'_x, \mathbf{p}'_y, \mathbf{p}''_z)^T, \exists (C,c) \in \mathscr{A}', \mathbf{p}' \in C, \mathbf{p}'' = u(\mathbf{p}')\} \quad (10)$$

We can note that the estimation of the inflated regions preserves a 3D discrete representation (see Figure 5).

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

$$E(\mathbf{p}_i, \mathbf{p}_j) = \begin{cases} true & \text{if } \mu(\mathbf{p}_i - \mathbf{p}_j) \leq \eta \\ false & otherwise \end{cases} \quad (11)$$

Here $\mu(\Delta\mathbf{p}) = \left\|\Delta\mathbf{p}^T \cdot (1,1,\delta)^T\right\|$, η is set equal to half the robot length and $\delta = \frac{\eta}{\Delta z_{max}}$ is set according to both the robot morphology and the robot overcoming capabilities. Here Δz_{max} is the maximum obstacle height surmountable by the robot.

The graph G is further weighted to build the traversability graph wG, where the weights are defined as follows. Let ρ be the density of a point $\mathbf{p}$ neighborhood, as specified in Section V. Let $u(\cdot)$ be the function specified in Equation 9 above, then the weights labeling an edge E are defined as $w = w_{length} + w_{inflated} + w_{density}$, where for each pair of points $\mathbf{p}_i$, $\mathbf{p}_j \in \mathscr{N}$ for which $E(\mathbf{p}_i, \mathbf{p}_j)$ is true:

- $w_{length} = \left\|\mathbf{p}_i - \mathbf{p}_j\right\|$
- $w_{inflated} = \begin{cases} \frac{1}{u(\mathbf{p}_j)} & \text{if } u(\mathbf{p}_j) < r_{infl} \\ \infty & otherwise \end{cases}$
- $w_{density} = \frac{2}{\rho(\mathbf{p}_i) + \rho(\mathbf{p}_j)}$

Here r_{infl} is a parameter which specifies the inflated radius.

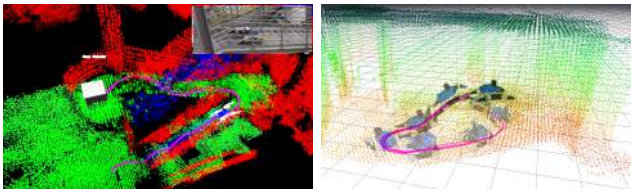


Fig. 6. On the left point cloud segmentation of the fire escape stairs, with path drawn in magenta, and detail of the robot climbing the stairs (from Fire Escape stairs experiment). On the right and composed path of the robot from the gallery up to the top of the ramp and the bridge (from the Full 3D experiment). It is interesting to note that the robot passes under the gallery still following its path, up to the goal

Fig. 7. The Italian Fire Fighters rescue training area in Prato (IT)

VII. Path planning

The weighted graph wG is meant for traversability graph. Then, given a goal specifying a point on the labeled map, a feasible 3D path $\mathscr{P}$ can be computed by a generic optimization algorithm such as Dijkstra [24], A* [25] or D* lite [26]. Here, in our experiments, we have been using the Dijkstra algorithm.

VIII. Experiments

In this section we present the experiments performed to evaluate the accuracy of the 3D map segmentation and the generated path feasibility, 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 have 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 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 accommodates automatic goal generation, which is not treated here. Table (a) reports the accuracy of the segmentation of the 3D map of the environment and the feasibility of the path, generated on the 3D segmented and labeled map. Here the values are averaged over eleven trials to obtain the same goal position. On average, the map had about 52 thousand points.

Fire Escape stairs The 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, and when the robot is up the goal is moved to the ground. The robot has simply to climb up the stairs to reach the goal and then turning on its self and step down back to reach the new selected goal. In some trials the robot is actually not able to get down autonomously, because localization problems might arise when the robot is turning around itself. Figure 5 shows the robot climbing the fire escape stairs, visualizing only the graph where the inflated regions are in red. Figure 6, left panel, shows the segmentation of the courtyard with the fire escape stairs and, in the window up right, a detail of the robot climbing the stairs. Table (b) reports the average segmentation time, first table; the percentage of the clusters correctly classified, middle table; and in the last table 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 8). 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 Figure 6, right panel). Table (c) 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 tests have 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. During these tests the robot was teleoperated, so as to explore a wide area and to acquire, at real-time, the point cloud. The goal was fixed and the robot

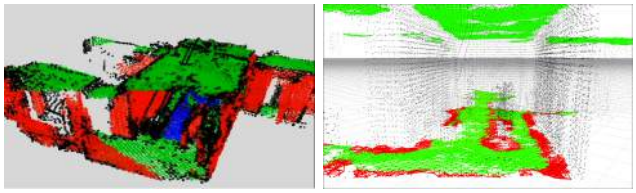


Fig. 8. On the left segmentation of the third experiment map. On the right a screen shot of the generated graph.

(a) Prato experiment

	Segmentation		**Graph**	**Smoothing**	**Journey**
	Time (s)	Accuracy	Time (s)		Time (s)
Point Cloud	0.27	0.86	1.19		
Path				0.2	357

(b) Fire escape stairs experiment

	Segmentation		**Graph**	**Smoothing**	**Journey**
	Time (s)	Accuracy	Time (s)		Time (s)
Point Cloud	0.32	0.91	1.45		
Path				0.2	210

(c) 3D designed experiment

	Segmentation		**Graph**	**Smoothing**	**Journey**
	Time (s)	Accuracy	Time (s)		Time (s)
Point Cloud	0.12	0.97	0.92		
Path				0.2	143

computed a new path every new point cloud. The results, in terms of time of computation, of the different algorithms are reported in Figure 9. Note that the robot was able to elaborate a point cloud composed of 55000 points and to generate a path in less than 3.5 *s*.

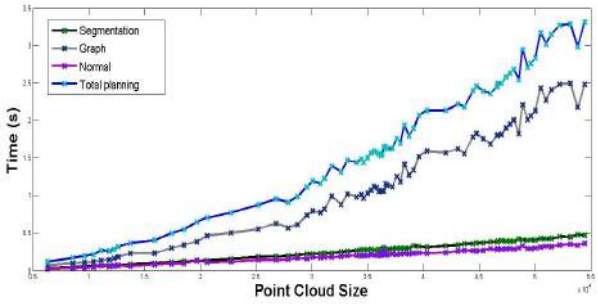


Fig. 9. Time of computation for both the segmentation and the path planning algorithm with respect to the size of the point cloud.

IX. Conclusions

We presented a framework for solving the autonomous 3D navigation task for tracked robots. In the paper we have faced the problem of 3D path and motion planning, including flippers control, and path generation via 3D map clusterization and labeling. We have discussed how our approach goes beyond the state of the art, and shown that it is promising. Indeed, the experiments prove the robot to be able to autonomously face any kind of structure within the limits of its overcoming abilities, and in real time.

X. Acknowledgement

This research was supported in part by EU FP7-ICT-Project NIFTi 247870 and it is currently supported by EU-FP7-ICT-Project TRADR 609763.

References

[1] E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *ICRA*, 2010, pp. 300–307.

[2] K. Iagnemma, F. Genot, and S. Dubowsky, "Rapid physics-based rough-terrain rover planning with sensor and control uncertainty," in *ICRA*, vol. 3, 1999, pp. 2286–2291.

[3] J. Klaess, J. Stueckler, and S. Behnke, "Efficient mobile robot navigation using 3d surfel grid maps," in *ROBOTIK*, May 2012, pp. 1–4.

[4] O. Wulf, C. Brenneke, and B. Wagner, "Colored 2d maps for robot navigation with 3d sensor data," in *IROS*, vol. 3, 2004, pp. 2991–2996.

[5] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. Gerkey, "Outdoor mapping and navigation using stereo vision," in *Exp. Rob.* Springer Berlin Heidelberg, 2008, vol. 39, pp. 179–190.

[6] M. Montemerlo, S. Thrun, H. Dahlkamp, and D. Stavens, "Winning the darpa grand challenge with an ai robot," in *AAAI*, 2006, pp. 17–20.

[7] J. Lee, C. Pippin, and T. R. Balch, "Cost based planning with rrt in outdoor environments." in *IROS*, 2008, pp. 684–689.

[8] D. Ferguson and M. Likhachev, "Efficiently using cost maps for planning complex maneuvers," in *ICRA Workshop on Planning with Costmaps*, 2008.

[9] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *ICRA Workshop on Path Planning on Costmaps*, 2008.

[10] T. Simon and B. Dacre-Wright, "A practical motion planner for all-terrain mobile robots." in *IROS*, vol. 2, 1993, pp. 1357–1363.

[11] A. Nuchter, H. Surmann, and J. Hertzberg, "Automatic model refinement for 3d reconstruction with mobile robots," in *3DIM*, 2003, pp. 394–401.

[12] J.-F. Lalonde, N. Vandapel, D. F. Huber, and M. Hebert, "Natural terrain classification using three-dimensional ladar data for ground robot mobility," *JFR*, vol. 23, no. 10, pp. 839–861, 2006.

[13] P. Biber, H. Andreasson, T. Duckett, and A. Schilling, "3d modeling of indoor environments by a mobile robot with a laser scanner and panoramic camera," in *IROS*, vol. 4, 2004, pp. 3430–3435.

[14] J. Diebel and S. Thrun, "An application of markov random fields to range sensing," in *NIPS*. MIT Press, 2005, pp. 291–298.

[15] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, "3d path planning and execution for search and rescue ground robots," in *IROS*, 2013, pp. 722–727.

[16] B. Morisset, R. B. Rusu, A. Sundaresan, K. Hauser, M. Agrawal, J.-C. Latombe, and M. Beetz, "Leaving flatland: Toward real-time 3d navigation," in *ICRA*, 2009, pp. 3786–3793.

[17] D.-H. Kim and J.-H. Oh, "Tracking control of a two-wheeled mobile robot using inputoutput linearization," *Contr. Eng. Prac.*, vol. 7, no. 3, pp. 369 – 373, 1999.

[18] M. Gianni, G. Gonnelli, A. Sinha, M. Menna, and F. Pirri, "An augmented reality approach for trajectory planning and control of tracked vehicles in rescue environments," in *SSRR*, 2013, pp. 1–6.

[19] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing icp variants on real-world data sets," *Aut. Rob.*, vol. 34, pp. 133–148, 2013.

[20] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, pp. 71–78, 1992.

[21] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross, "Shape modeling with point-sampled geometry," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 641–650, 2003.

[22] H. Farid and E. Simoncelli, "Differentiation of discrete multidimensional signals," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 496–508, 2004.

[23] F. Schindler and W. Förstner, "Fast marching for robust surface segmentation," in *PIA*, 2011, pp. 147–158.

[24] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[25] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[26] S. Koenig and M. Likhachev, "D*lite," in *AAAI*, 2002, pp. 476–483.

Point Cloud Segmentation and 3D Path Planning for Tracked Vehicles in Cluttered and Dynamic Environments

Federico Ferri and Mario Gianni and Matteo Menna and Fiora Pirri
ALCOR, Vision, Perception and Cognitive Robotics Laboratory
DIAG, University of Rome 'Sapienza', Italy
{ferri, gianni, menna, pirri}@dis.uniroma1.it

***Abstract*— The paper presents a framework for tracked vehicle 3D path planning in rough areas, with dynamic obstacles. The framework provides methods for real-time point cloud interpretation, segmentation and traversabilty analysis tacking also into account changes such as dynamic obstacles and provides a terrain structure interpretation. Moreover the paper presents $R_2A^\star$ an extended version of randomized $A^\star$ coping with difficult terrains and complex paths for non-holonomic robots.**

I. Introduction

Navigation in urban disaster environments is a challenging task. Though tracked robots (UGV) are well equipped to operate in clutter, search and rescue operations still require a human operator to remotely teleoperate the robot [1], [2]. That is why improving autonomy for these robots is a pivotal issue in search and rescue. To this end, terrain structure interpretation is a fundamental precondition to move toward autonomy or semi-autonomy and it is still an open problem. Indeed, to guarantee safe autonomous UGV navigation both path planning and trajectory tracking controllers require real-time interpretation of off-road heterogeneous harsh and complex outdoor terrains. The development of effective and accurate three dimensional maps provides, in fact, a good ground for 3D path planning.

An overview on terrain traversabilty methods can be found in [3]. Most of the methods use geometric features to interpret the terrain. For example, in [4], the authors use normal vector estimation from stereo vision to estimate inclination of a sector in an elevation map. In [5] the 3D path planning algorithm requires an initialization step to build a geometrical representation of the environment, from laser data. As the map of the environment changes, this step has to be re-iterated. Therefore, the approach does not seem to be appropriate for exploration missions, where map updates occur frequently. Moreover the approach works only in static environments.

In this work we propose a framework to deal with dynamic 3D environment mapping. We provide a general method to real-time point cloud interpretation, segmentation and affordance of the full 3D environments, managing changes, such as dynamic obstacles. Our approach differs from probabilistic ones [6], [7] needing multiple observations for a complete update of dynamic obstacles, as it relies on a compact representation, requiring to store just obstacle points with normal information, in so allowing to recover information about free space. The approach provides a terrain structure interpretation, via terrain transferability analysis, and introduces a path planner formalization based on randomized $A^\star$. The remainder of this paper is organized as follows. Section II describes the overall framework, introducing some notation too. Section III introduces the dynamic obstacles removal method. Section IV proposes the basic ideas for computing geometric features addressing both light and accurate segmentation. The terrain analysis is presented in Section V and the path planner is described in Section VI. We conclude with the description of the experiments performed to evaluate the proposed method in Section VII and some insight for future work is addressed in Section VIII.

II. Framework Overview

An overview of the framework is resumed in Figure 1. The framework is composed of two different pipelines. The first pipeline, shown at the bottom of Figure 1, comprises the computational steps needed for a global path planner to generate long distance paths toward the goal. The second one, shown at the top of Figure 1, illustrates the sequence of steps to compute local paths toward intermediate sub-goals, as described in Section VI.

The first pipeline has been designed with the main purpose of providing a fast and approximated interpretation of the environment, where a global planning can take place. In parallel the second pipeline has the main role of locally build a more accurate model of the environment, needed for traversability assessment and local path planning. The $Scan_t$, acquired from the rolling laser sensor of the robot at time t, is filtered for outliers removal and registered via an iterative closest point (ICP) algorithm [8]. The accumulated map of the environment Map_t, at time t, is clustered and labeled using both normals and surface variation features, estimated at large neighborhoods (see Section IV). The global path planner takes as input the goal position $\mathbf{q}_G$ and the labeled map $LMap_t$, returned by the segmentation process, and it provides the local 3D path planner, giving a preliminary information on the goal reachability. For points in $Scan_t$, which are close to the robot current position, both the normals and the principals curvatures k_1 and k_2 are estimated, providing an accurate classification for curvature. The labeled $SScan_t$ is further merged with the accumulated segmented map $SMap_{t-1}$, at time $t-1$. This process requires the following two steps: (1) removal of dynamic obstacles,

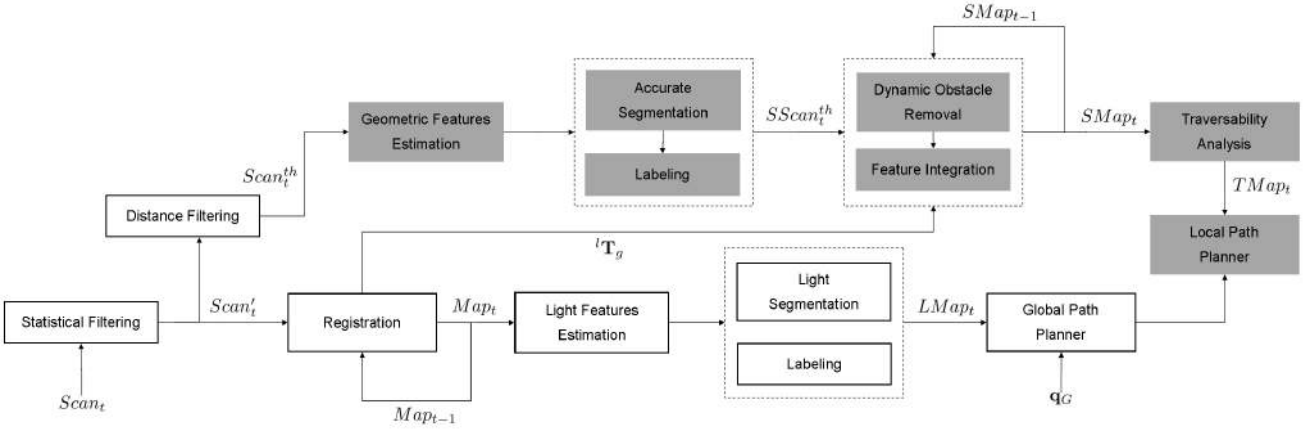


Fig. 1: The schema resumes the framework of the proposed method.

and (2) resolution of all the segmentation conflicts that are detected within the overlapping labeled clusters. The new obtained $SMap_t$ is used for traversabilty analysis. The local path planner, based on randomized $A^\star$, receives as input an intermediate sub-goal of the global path and generates a sub-optimal local path, according to the local traversabilty map $TMap_t$.

III. Dynamic Point Cloud

Scanning the environment is a continuous process which, due to the presence of dynamic obstacles, needs to be periodically repeated in order to ensure that the built map is up to date, and the robot can safely navigate on its own. In static environment conditions, mapping can be performed by simply registering and adding or merging the 3D scan to the accumulated map. However, in a dynamic environment, special procedures are needed for merging the new 3D scan with the accumulated map. Not performing dynamic obstacles removal would result in a map cluttered by outdated dynamic obstacles, making autonomous navigation impossible after a short time. We identify with *OD* the region of space concerning the points to delete from *Map*, corresponding to outdated dynamic obstacles, also known as *deletion volume* [9] as mentioned in Section II. The region of space *OD* is estimated from the *Scan* point cloud, that is, from what the sensor (3D laser scanner) is observing right now as the *free space*. A point in the old point cloud shall be removed when it lies inside *OD*. Our method for testing whether a point should be removed is based on a local signed distance test against the nearest simplex of the convex hull of *Scan*, which tells whether the point is inside *OD*, also accounting for noise. Note that this method may underestimate *OD*, i.e. it may fail to remove some points very close to *Scan* (see Figure 2), however we should note that this side effect is more preferable than erroneously removing points from *Map*. To minimize the effects of this underestimation error, we work with spherical partitions of *Scan* and *Map*. We partition the set of points in *Scan* (and *Map* respectively) according to both spherical coordinates' angles, and perform the removal test partitionwise. As the

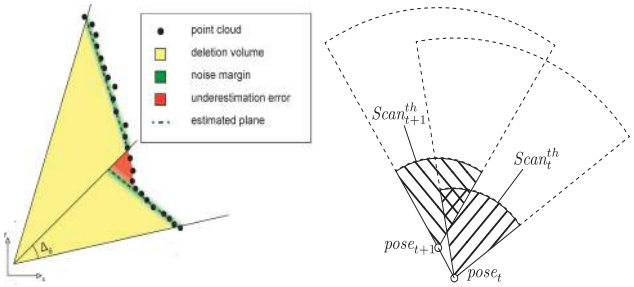


Fig. 2: On the left a 2D example of the **deletion volume** *OD* estimation, using 2 partitions: the planes (in blue dashed lines) are estimated from the convex hull (not shown in this figure) of the point cloud. A point of the old map should be removed if it is outside the noise margin and it is on the same side of the laser. It may happen that the deletion volume (in yellow) is underestimated; however, reducing the partition size reduces this error, without increasing the computational cost. On the right two consecutive scans and the overlapping region where conflicts might take place.

number of partitions approaches infinity, the region of *OD* matches the true deletion volume. However, there is a upper limit for choosing the number of partitions, i.e. when using narrow partitions (high number of partitions) smaller than the actual point cloud pitch, this approach fails to detect the surface of *Scan*, possibly resulting in the erroneous non-removal of points from *Map*.

IV. Geometric features and segmentation

Point cloud segmentation of large scenes, for the purpose of robot planning, requires robust and real-time evaluation of geometric features that can establish the meaningful components of the environment in terms of traversability. A point cloud can be locally considered to be the discrete representation of a surface patch. Despite this, point clouds are unorganized point sets, and differential operations are not easily performed. The only effective structure to base geometric features computation is the neighborhood of a point. For the computation of normals this is quite straightforward. Indeed, given the neighborhood $\mathcal{N}_m(\mathbf{p})$ of

a point, having cardinality m, where $\mathbf{p} = (x, y, z(x,y))^\top$, the computation of the normal amounts to the computation of the plane $\pi = (\mathbf{n}, d)$ tangent to $\mathbf{p}$, and minimizing the distance of each point $\mathbf{p}_i \in \mathscr{N}_m(\mathbf{p})$, under the constraint $\|\mathbf{n}\| = 1$. This turns out to be the vector $\hat{\mathbf{u}}$ corresponding to the smallest eigenvalue λ in D, given $\mathbf{V}\,\mathbf{D}\,\mathbf{U}^\top = (\mathscr{N}_m(\mathbf{p}) - (\mathbf{p} \otimes \mathbf{1}_m))^\top$, where $\otimes$ is the Kronecker product. Then for $\pm\hat{\mathbf{u}}$ the sign compatible with view point *view*, namely the sign ensuring smallest angle for $\cos^{-1}(view - \pm\hat{\mathbf{u}})/\|view - \pm\hat{\mathbf{u}}\|$, is chosen. Where *view* is assumed to be directed toward the viewer. Then the sought for normal $\mathbf{n}$ at $\mathbf{p}$ is taken to be $\hat{\mathbf{u}}$ with the chosen sign. This method, being subject to the vantage point cannot provide an instantaneous correct assessment of the normals of an object, therefore in several cases a number of conflicts need to be handled in the merging and integration of the point cloud features.

For the curvature computation, on the other hand, the problem is more involved, since normal curvature, principal curvatures and the Gaussian and mean curvatures obtained by the principal ones, are defined by the coefficients of the first and second fundamental forms, which are essentially differential operators.

To alleviate the difficulty of computing derivatives on an unorganized point set, many researchers (see [10], [11], [12]) resorted to statistical computation of the second fundamental form, via some suitable approximation. These statistical approximations capture somehow the curvature variation but often are not enough discriminative. One of the most well known approximation is the one introduced by Pauly in [13] based on the above computation of the normal, but using the symmetric form, which actually returns the covariance of the neighborhood, and the eigen-decomposition of this last. Then the surface variation is given by $\lambda_1 / \sum_{i=1}^{3} \lambda_i$ with $\lambda_1 \leq \lambda_2 \leq \lambda_3$. This quantity varies between 0 and $1/3$, and has the property of being high if the region is highly curved and it is small if the region is flat, however does not capture the principal curvatures and the induced classes. For these reasons we discriminate between close and far regions for segmentation, providing a more accurate estimation of features for points with a distance $\|\mathbf{p}, Robot\| < 3m$ and a light feature space for further points. For far points we consider both normals and curvature variation as defined above [13], which is adequate to characterize large regions and classify major obstacles. Note that for far points we compute the light features on the integrated map Map_t at time t, and consider large clusters of points with mean radius of $0.2m$.

On the other hand for the region right in front of the robot a more accurate evaluation, taking into account the classes induced by the principal curvatures, needs to be considered. Let $H \in \mathbf{R}^3$ be the point set with distance $< 3m$ from the robot. This set is partitioned into clusters $C_i \subset H$ with centroid $\mathbf{c}_i \in H$, such that each cluster is not greater than 50 points, and not smaller than 7 points. Then, given the normal to $\mathbf{c}_i$, computed as indicated above, the neighborhood is rotated and translated so that $\mathbf{n}$ is transformed to $\mathbf{n}_z$ parallel to the z-axis of the reference frame of the robot and $\mathbf{c}'_i = \mathbf{T}(\mathbf{c}_i) = (0, 0, z(0,0))$, with $\mathbf{T}$ the transformation.

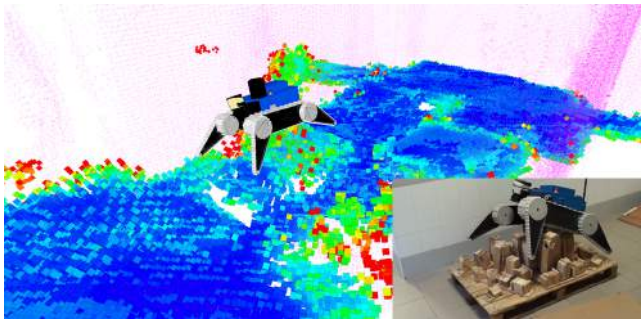


Fig. 3: Traversability map *TMap* computed real-time while the robot is traversing a home-made rubble pallet, in the lab. The colors indicate the traversabilty cost, from minimal cost (dark blue) to maximal cost (dark red).

Consider a point $\mathbf{q} \in \mathbf{T}(C)$, $\mathbf{q} = (x, y, z(x,y))$, the projection of the vector $\mathbf{cq}$ on the normal $\mathbf{n}_z$ is, from Taylor expansion:

$$(z(x+dx, y+dy) - z(x,y))\mathbf{n}_z = ((z_x dx + z_y dy) + (1/2)(z_{xx}dx^2 + 2z_{xy}dxdy + z_{yy}dy^2) + o(dx^2 + dy^2))\,\mathbf{n}_z \quad (1)$$

here dx and dy are the differentials w.r.t. the origin and are less than 0.2, by the choice of the cluster, then the last term can be ignored, and also $z_x\mathbf{n} = 0$ and $z_y\mathbf{n} = 0$. Furthermore, since $\mathbf{c}$ is centered at the origin we obtain: $2z(dx, dy)\mathbf{n} = (z_{xx}dx^2 + 2z_{xy}dxdy + z_{yy}dy^2)\mathbf{n}$ as the projection of the point $\mathbf{q}$ on the transformed normal, and this is, indeed, the second fundamental form. Therefore a principal direction is toward the point $\mathbf{q} \neq \mathbf{c}$ that maximizes this projection and the other principal direction is toward the point $\mathbf{q}' \neq \mathbf{c}$ that minimizes this projection. We can see that in this formulation the sign of the principal curvatures is maintained and the induced classes (elliptic, planar, paraboloid, ellipsoid[14]) for the points can be defined.

We have thus obtained two sets of features: (1) the light ones specified by surface variation and normals, on the integrated map Map_t, for points far from the robot, and using large and possible overlapping neighborhoods, and (2) the accurate ones, specified by the principal curvatures κ_1 and κ_2 and normals, on the scan map $Scan_t$, for points close to the robot, and using small clusters obtained by partitioning the close point set H.

Segmentation then is initialized with histogram distribution of normals and curvatures to highlight the sought for classes: *walls*, *terrain*, *surmountable obstacle*, and *stairs/ramps*. Further, clusters are refined and suitably combined according to the energy minimization algorithm illustrated in [15].

V. Traversability in Clutter

In this section we describe the method proposed to assess the traversability of the environment. Traversability is defined as a composite index depending on elevation statistics (e.g., height difference, slope, discontinuity, terrain roughness), physical properties of the terrain (e.g., hardness, slippery) and robot mobility (e.g., wheels rather than tracks, or legs, maximum superable step, steering efficiency). In the pro-

posed approach traversability is assessed with a cost function, which takes into account the following geometrical features of the surrounding environment: (1) terrain roughness; (2) terrain classification, based on both terrain slope and robot locomotion capabilities; (3) obstacle clearance and (4) point cloud density. This cost function has been defined as follows. Let $SMap_t$ be the segmented point cloud, at time t, introduced in Section IV. Let $\mathbf{p} \in SMap_t$ a point of the point cloud. The point-wise traversability cost function $f : SMap \mapsto \mathbb{R}$, returning the static traversability cost of the point $\mathbf{p} \in SMap_t$, is given by

$$f(\mathbf{p}) \quad := w_L^{(\mathbf{p})} \cdot \left(w_{CL}^{(\mathbf{p})} + w_{Dens}^{(\mathbf{p})} + w_{Rough}^{(\mathbf{p})} \right) \tag{2}$$

Here the term $w_L^{(\mathbf{p})}$ represents the contribution to the overall cost of $\mathbf{p} \in SMap_t$, computed according to the segmentation of the local map. This cost term is lower for points belonging to clusters labeled as *ground*, while increases for points belonging to clusters, which are more difficult to traverse, such as those labeled with *stairs* or *ramps*. $w_{CL}^{(\mathbf{p})}$ is the cost contribution provided by the obstacle clearance. Points belonging to clusters which are relatively close to obstacles, inflated by the inscribed radius of the robot, have been assigned with a higher cost, since they are unreachable for the robot. This cost term is computed as follows

$$w_{CL}^{(\mathbf{p})} = \frac{1}{\max\left\{\min\left\{||\mathbf{p}-\mathbf{p}'||\right\}, \Delta\right\} - \Delta} \tag{3}$$

for all $\mathbf{p}' \in SMap_t$, such that $\mathbf{p}'$ belongs to clusters labeled either as *wall* or *ceiling*. Here Δ is the inflated radius (see Figure 4). Note that this approach is similar to the artificial potential fields methodology, proposed in [16]. $w_{Dens}^{(\mathbf{p})}$ represents the contribution to the overall cost of $\mathbf{p} \in SMap_t$, computed with respect to the point cloud density, as follows

$$w_{Dens}^{(\mathbf{p})} = \frac{|\mathscr{N}_r(\mathbf{p})|}{\frac{4}{3}\pi r^3} \tag{4}$$

Here $|\mathscr{N}_r(\mathbf{p})|$ is the cardinality of the set of points belonging to the neighborhood of $\mathbf{p} \in SMap_t$. High values of $w_{Dens}^{(\mathbf{p})}$ are assigned to points $\mathbf{p} \in SMap_t$ which are relatively close to terrain regions not correctly segmented, due to unevenness of the terrain, holes or lack of laser scan data. This cost term has the effect of deterring navigation near such areas. $w_{Rough}^{(\mathbf{p})}$ measures the terrain roughness. Roughness is estimated by computing the average distance of the outliers from the plane fitted on the points belonging to the neighborhood $\mathscr{N}_r(\mathbf{p})$ of $\mathbf{p} \in SMap_t$. By applying the defined traversability cost function $f(\mathbf{p})$, $\forall \mathbf{p} \in SMap_t$, we obtain the traversability map $TMap_t$ of the environment.

VI. Path planner

The global path planer, which we do not discuss here, computes a main direction toward a goal, either given or generated. The local path planner computes the intermediate steps toward this goal. In this section we present the local path planner based on randomized $A^\star$ ($RA^\star$) [17] and extended to cope with non-holonomic robots and large point clouds. Similarly to $RA^\star$ the proposed algorithm $R_2A^\star$ samples the action space, uses a k-dimensional tree data structure for partitioning and adapts the branching factor according to a search radius. Differently from $RA^\star$ we propose a density function acting on the boundary of the current location neighborhood.

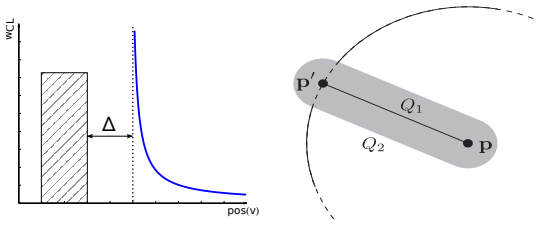


Fig. 4: On the left a plot of the term w_{CL} as a function of the vertex position. The dashed region represents an obstacle, and the continuous blue line is the cost. On the right the boundary of a point neighborhood, with highlighted the two sets Q_1 and Q_2.

Let $\mathbf{q}_{LG}$ be an intermediate goal, that is, a position in the traversability map $TMap_t$ at a distance less than a given threshold. If there exists a path from the current robot position to $\mathbf{q}_{LG}$, the local path planner computes it. The basic idea of the proposed algorithm is to generate a search tree $\mathscr{G} = \langle N, E \rangle$, N the set of nodes and E the set of the edges, by sampling candidate poses. Sampling is governed by a probability density function defined on the traversabilty map $TMap_t$, at time t. The novelty of the approach is to reduce the search space still ensuring to find a path if this exists.

We first introduce some notation. A path *path* is an ordered list of nodes, such that if two adjacent nodes n_i, n_{i+1} are in *path* then there exists an edge directly linking them, where an edge E between two nodes is a relation defined according to the robot length. A trajectory τ is a curve specified by arc length s, $s \in [0,1]$ and it is defined on the traversability map.

The function $pos : N \to \mathbb{R}^3$ returns the coordinates of the point $\mathbf{p} \in TMap_t$, associated with the node $n \in N$, the function $parent : N \to \mathbb{R}$ returns the coordinates of the point $\mathbf{p}' \in TMap_t$, associated to the parent of a node $n \in N$. We denote *visited* the data structure storing the visited nodes and denote *leaf* the data structure storing the current leaf nodes, namely, the frontier of the search tree $\mathscr{G}$.

Given a point $\mathbf{p} \in TMap_t$ such that $\mathbf{p} = pos(n)$, with $n \in N$, the neighborhood of $\mathbf{p}$ is denoted $\mathscr{N}_r(\mathbf{p})$, as in Section IV, where here r denotes the radius of the neighborhood; and with $\partial\mathscr{N}_r$ we denote the boundary of the neighborhood, namely $\partial\mathscr{N}_r(\mathbf{p}) = \{\mathbf{p}' \in \mathscr{N}_r | \|\ \mathbf{p} - \mathbf{p}'\| - r < \varepsilon\}$. Finally, the density for $\mathbf{p} \in TMap_t$ is $g_f = \alpha(1/f(\mathbf{p}))$, where f is defined in eq. (2) and $\alpha = \sum_1^m (1/f(\mathbf{p}_i)), \mathbf{p}_i \in \mathscr{N}_r(\mathbf{p})$ is a normalization term, m the cardinality of the neighborhood of $\mathbf{p}$. The density reinforces points with low traversability cost.

The parameters of a node are the followings:

- The euclidean distance $d : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ between the position of two nodes.
- The set $Q_{1,n} = \{\hat{\mathbf{p}} \in \mathbb{R}^3 | \hat{\mathbf{p}} = \lambda\mathbf{p} + (1-\lambda)\mathbf{p}'\}$, with $\lambda \in [0,1]$, $\mathbf{p}' = pos(n)$ and $\mathbf{p} = parent(n)$.

- The set $Q_{2,n} = \{\mathbf{p} \in TMap_t | d(\mathbf{p}, \hat{\mathbf{p}}) < \rho\}$, $\forall \hat{\mathbf{p}} \in Q_1$, with ρ half the robot length. See Figure 4.
- The traversability cost $f(pos(n))$, defined in eq. (2).
- The cost $f' : \mathbb{R}^3 \to \mathbb{R}$ defined as $f'(n) = \max\{f(\mathbf{p})\}$, $\forall \mathbf{p} \in Q_{2,n}$.
- The admissible goal heuristic $h^\star : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$, between the position of the current node and the goal node.
- The function $r : \mathbb{R}^3 \to \mathbb{R}$, which sets the search radius, finding the distance of a point $\mathbf{p} \in TMap_t$ from the nearest obstacle, defined as follows:

$$r(pos(n)) = \min_{\mathbf{p}' \in TMap_t} \left\{ \left\| pos(n) - \mathbf{p}' \right\|_2 \right\} \tag{5}$$

  $\forall \mathbf{p}.\mathbf{p} \in TMap'_t$ with $\mathbf{p}$ belonging to clusters labeled either as *wall* or *ceiling*.

The generation and expansion of the tree $\mathscr{G}$ is defined inductively as follows. For the basic step, the initial node $n_0 = pos(\mathbf{p}_{robot})$, namely the current robot position, is in N.

For the induction step. Let $n \in N$ and consider the data structures *visited* and *leaf*.

Generation Candidate nodes are sampled according to the density g_f. A node $n^\star$ is chosen among the set of candidate nodes if the following two conditions are satisfied.

1) $n^\star \notin visited$;
2) $\exists \mathbf{p} \in TMap_t$ such that $\mathbf{p} \in Q_{1,n^\star}$, with $\mathbf{p} = parent(n^\star)$

If $n^\star$ satisfies the above conditions then it is added to the *leaf* data structure.

Expansion Given a node $n \in leaf$, the node is expanded according to the above parameters, used in the following rules. Let us assign to each $\mathbf{p} \in \partial\mathscr{N}_r(pos(n))$ the value $g_f(\mathbf{p}) = \alpha \frac{1}{f'(\mathbf{p})}$, where the radius r defining the neighborhood has been computed as in eq. (5). Here, we are implicitly defining a discrete probability distribution over $\partial\mathscr{N}_r(pos(n))$ in which points with lower traversability cost have high probability.

In principle we are computing a continuous trajectory $\tau(s)$, between n_0 and n_G, where $s \in [0,1]$, and with the cost of the trajectory within the traversability map defined as:

$$\int_0^1 cost_{TMap}(\tau(s))ds \tag{6}$$

Since $R_2A^\star$ returns a discrete path we approximate this continuous cost with the following cost function defined for each node:

$$\widehat{cost(n)} = f'(pos(n)) + d(pos(n), parent(n)) + h^\star(pos(n), \mathbf{q}_{LG})$$

And, thus, the approximated trajectory cost is the cost of the expanded nodes, obtained by minimizing:

$$\sum_i \widehat{cost(n)}_i, \qquad \forall n_i \in path \tag{7}$$

The algorithm starts to expand the node in *leaf* with the minimum cost and ends as soon as the current neighborhood is the one of the goal node, and the distance to the goal node is less than the robot size.

We show, now, that if there exists an admissible path between two points which is found by $A^\star$ then there is a path computed by the proposed algorithm $R_2A^\star$. More precisely, let n_0 denote the start node and n_G the goal node and assume that $A^\star$ computes the path Q such that $n_0, n_G \in Q$ then:

Lemma 1: There exists a path *path* computed by $R_2A^\star$ such that $n_0, n_G \in path$ and $\sum_i \widehat{cost(n)}_i \leq \sum_i d(parent(n_i), pos(n_i)) + w_i h^\star(n_i, n_G)$.

Proof. First note that we have made explicit a penalty w_i added to $h^\star$ to balance the lack of the traversability cost term in the $A^\star$ cost function. Now, observe that in Q there must exists a node n_b that belongs to $\partial\mathscr{N}(n_0)$ since any path from n_0 to n_G has to pass from the boundary of the neighborhood of n_0. Therefore the proof can be done by induction on the number of boundaries traversed. As basic step, assume n_G belongs to the neighborhood of n_0 and is not on the boundary, this implies there are no obstacles and thus $A^\star$ and $R_2A^\star$ coincide finding a straight trajectory. Assume, now, that the current node is n, with $\partial\mathscr{N}(pos(n))$ the boundary of its neighborhood, and there exists a path from n to n_G, then there exists a node n_b with $pos(n_b) \in \partial\mathscr{N}(n)$ such that $cost(n_b) + cost(n_G) \leq cost(n) + cost(n_G)$. In fact $R_2A^\star$ finds the boundary node before $A^\star$ because of the condition of the expansion. Clearly also $A^\star$ must sooner or later pass through the boundary, but since it does not overestimate the cost and chooses straight lines, in the worst case it might cross the boundary after having expanded a greater number of nodes than $R_2A^\star$, hence the path cost must be greater. □

We have thus shown that $R_2A^\star$ behaves like $A^\star$ in the worst case and also that its cost cannot exceed $A^\star$. The proof that its cost is admissible is here omitted.

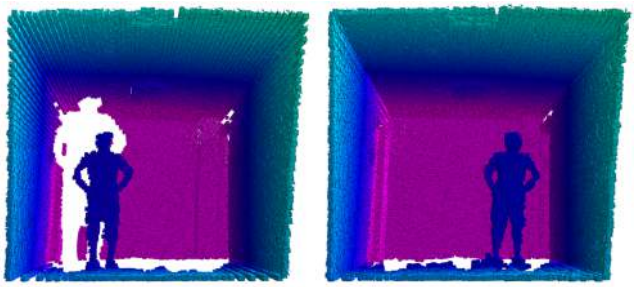


Fig. 5: On the left a person is standing in front of a wall (dark blue blob on the left). The white blob behind the person indicates that the point cloud of the region of the wall has not been aggregated into the entire map of the area, due to occlusion. On the right the result of the dynamic correction algorithm: after the motion of the person, the map is updated by both deleting the blob of the person and by aggregating the point cloud of the region of the wall, previously occluded.

	$R_2A^\star$ nodes in a path over 6 runs					
$A^\star$	run_1	run_2	run_3	run_4	run_5	run_6
138	72	69	75	81	71	87

TABLE I

VII. Experiments

We evaluated the framework on the dynamic point cloud and on the 3D path planner, also comparing $R_2A^\star$ and $A^\star$

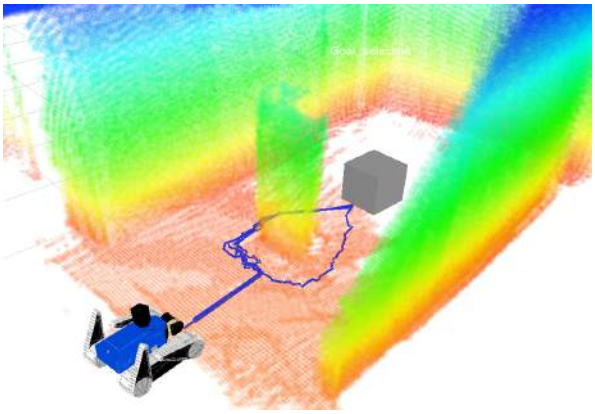


Fig. 6: Path planning based on the $R_2A^\star$ algorithm. In blue are shown sub-path toward the goal, highlighted by a large cube, and located behind such an obstacle.

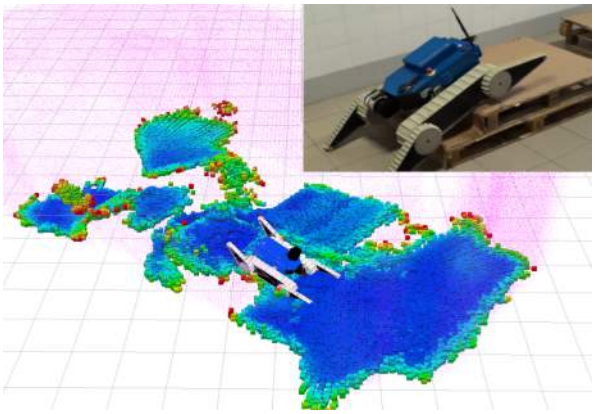


Fig. 7: Traversability experimented on a path interrupted by a ramp and steps.

see Table I above.

Figure 5 illustrates the results of dynamic correction, removing outdated dynamic obstacles from the current map of the environment. Observe that after the motion of the subject, the method correctly updates the map of the area, by both deleting the blob of the person and by aggregating the point cloud of the region of the wall. The region of the point cloud earlier occluded by the person is aggregated together with the point cloud representing the moved person, to the current map. Note that, in the current map the region of the wall, actually occluded by the blob of the person, still persists.

Figure 6 shows some traces of the local path planner, based on the $R_2A^\star$ algorithm. The algorithm generates intermediate sub-paths toward a given target pose, the sub-paths are shown in blue. The robot stands in front of a non superable obstacle, located in the middle of the area. A target pose has been posted behind such an obstacle. $R_2A^\star$ computes, over time, possible alternative paths (blue lines), generated by different runs of $R_2A^\star$. The cumulative cost of these solutions, measured with respect to the total number of nodes of the search tree expanded by $R_2A^\star$, has been compared to the number of nodes expanded by standard $A^\star$ algorithm. Table I reports the results obtained from this comparison. Traversability and the effective cost of the planner in clutter has been experimented in any kind of terrain, purposely built to prove the impact of the cost on the planner (see Figure 3 and Figure 7).

VIII. Conclusion

3D path planning in rough and cluttered areas, subject to dynamic changes, is a hard problem, we have presented a framework for point cloud segmentation and autonomous path planning tasks for tracked robots. There are still several open problems, such as detailed understanding of the environment to improve traversability, and continuous 3d path planning for complex tasks requiring long lasting activities. These are topics that we are facing and will be treated in future works.

References

[1] G. M. Kruijff, F. Pirri, M. Gianni, P. Papadakis, M. Pizzoli, A. Sinha, E. Pianese, S. Corrao, F. Priori, S. Febrini, S. Angeletti, V. Tretyakov, and T. Linder, "Rescue robots at earthquake-hit mirandola, italy: a field report," in *SSR*, 2012.

[2] E. Guizzo, "Japan earthquake: Robots help search for survivors," *Spectrum*, 2011.

[3] P. Papadakis, "Terrain traversability analysis methods for unmanned ground vehicles: A survey," *Eng. Appl. of A.I.*, no. 4, 2013.

[4] T. Braun, H. Bitsch, and K. Berns, "Visual terrain traversability estimation using a combined slope/elevation model." in *LNCS*, vol. 5243, 2008.

[5] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, "3D path planning and execution for search and rescue ground robots," in *ICRA*, 2013.

[6] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Aut. Rob.*, vol. 34, pp. 189–206, 2013.

[7] D. F. Wolf and G. S. Sukhatme, "Online simultaneous localization and mapping in dynamic environments." in *ICRA*, 2004.

[8] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp." in *Robotics: Science and Systems*, vol. 2, 2009.

[9] G. Zolynski, C. Schank, and K. Berns, "Point cloud gathering for an autonomous bucket excavator in dynamic surroundings," in *Proc. of the Comm. Vehicle Techn. Symp.*, 2012.

[10] J. Berkmann and T. Caelli, "Computation of surface geometry and segmentation using covariance techniques," *PAMI*, no. 11, 1994.

[11] G. Agam and X. Tang, "A sampling framework for accurate curvature estimation in discrete surfaces," *Visualiz. and Comp. Graph.*, no. 5, Sept 2005.

[12] E. Kalogerakis, P. D. Simari, D. Nowrouzezahrai, and K. Singh, "Robust statistical estimation of curvature on discretized surfaces," in *Symp. on Geom. Process.*, 2007.

[13] M. Pauly, M. H. Gross, and L. Kobbelt, "Efficient simplification of point-sampled surfaces," in *Visualization*, 2002.

[14] E. Abbena, S. Salamon, and A. Gray, *Modern Differential Geometry of Curves and Surfaces with Mathematica, Third Edition*, 2006.

[15] M. Menna, M. Gianni, F. Ferri, and F. Pirri, "Real-time autonomous 3d navigation for tracked vehicles in rescue environments," in *IROS*, 2014.

[16] E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *Rob. Aut.*, no. 5, 1992.

[17] R. Diankov and J. Kuffner, "Randomized statistical path planning," in *IROS*, 2007.

A Stimulus-Response Framework for Robot Control

MARIO GIANNI, ALCOR Laboratory, DIIAG, Sapienza, University of Rome, Italy
GEERT-JAN M. KRUIJFF, Nuance Deutschland GmbH, Aachen, Germany
FIORA PIRRI, ALCOR Laboratory, DIIAG, Sapienza, University of Rome, Italy

We propose in this article a new approach to robot cognitive control based on a stimulus-response framework that models both a robot's stimuli and the robot's decision to switch tasks in response to or inhibit the stimuli. In an autonomous system, we expect a robot to be able to deal with the whole system of stimuli and to use them to regulate its behavior in real-world applications. The proposed framework contributes to the state of the art of robot planning and high-level control in that it provides a novel perspective on the interaction between robot and environment. Our approach is inspired by Gibson's constructive view of the concept of a stimulus and by the cognitive control paradigm of task switching. We model the robot's response to a stimulus in three stages. We start by defining the stimuli as perceptual functions yielded by the active robot processes and learned via an informed logistic regression. Then we model the stimulus-response relationship by estimating a score matrix that leads to the selection of a single response task for each stimulus, basing the estimation on low-rank matrix factorization. The decision about switching takes into account both an interference cost and a reconfiguration cost. The interference cost weighs the effort of discontinuing the current robot mental state to switch to a new state, whereas the reconfiguration cost weighs the effort of activating the response task. A choice is finally made based on the payoff of switching. Because processes play such a crucial role both in the stimulus model and in the stimulus-response model, and because processes are activated by actions, we address also the process model, which is built on a theory of action. The framework is validated by several experiments that exploit a full implementation on an advanced robotic platform and is compared with two known approaches to replanning. Results demonstrate the practical value of the system in terms of robot autonomy, flexibility, and usability.

Categories and Subject Descriptors: I.2.9 [**Robotics**]: Autonomous Vehicles, Operator Interfaces; I.2.10 [**Vision and Scene Understanding**]: Perceptual Reasoning, Representations, Data structures, and Transforms; F.4.1 [**Mathematical Logic**]: Logic and Constraint Programming; I.2.4 [**Knowledge Representation Formalisms and Methods**]: Predicate Logic, Situation Calculus, Relation System, Representation Languages; I.2.6 [**Learning**]: Parameter Learning, Induction; I.2.8 [**Problem Solving, Control Methods, Search**]: Plan Execution, Formation, and Generation

General Terms: Design, Algorithms, Performance, Reliability, Languages

Additional Key Words and Phrases: Task switching, intelligent interaction, stimulus-response model, behavior adaptation

ACM Reference Format:
Mario Gianni, Geert-Jan M. Kruijff, and Fiora Pirri. 2015. A stimulus-response framework for robot control. ACM Trans. Interact. Intell. Syst. 4, 4, Article 21 (January 2015), 41 pages.
DOI: http://dx.doi.org/10.1145/2677198

This work is supported by the EU-FP7 ICT 247870 NIFTI project.
The reviewing of this article was managed by Lutz Frommberger, Heriberto Cuayahuitl, Nina Dethlefs, Antoine Raux, Matthew Marge, and Hendrik Zender.
Authors' addresses: M. Gianni and F. Pirri, ALCOR LAB, Dipartimento di Ingegneria Informatica, Automatica e Gestionale, "Antonio Ruberti", via Ariosto 25, 00185, Roma; emails: {Mario.Gianni, Fiora.Pirri}@dis.uniroma1.it; G.-T. M. Kruijff, Nuance Communications Deutschland GmbH, Kackertstrasse 10, D-52072 Aachen, Germany; email: gj@nuance.com.

ACM 2160-6455/2015/01-ART21 $15.00
DOI: http://dx.doi.org/10.1145/2677198

1. INTRODUCTION AND MOTIVATIONS

A robot interacting with people and with the environment is subject to several stimulus events, like any living organism. How these stimuli are to be represented and how a response should be modeled has not yet received the necessary attention, even though robot autonomy, beyond laboratory experiments, relies on an intelligent response to these stimuli.

Everyone knows what a stimulus is in the human and animal sense. The concept of *stimulus* has been studied the eighteenth century, with the work of Galvani and Volta on frogs, and was further explored in psychology, physiology, and psychophysics. Modern physiology considers a stimulus applied to a receptor and capturing information from both inside and outside the body; these are called *proximal* and *distal* stimuli, respectively. Following the pioneering work of Weber [1834], Gustav Fechner introduced psychophysics to model the relation of physical stimuli to the resulting mental facts. Gibson [1960] proposes the following a constructive view of a stimulus:

> 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. [Gibson 1960]

Gibson's implicit question is about the possibility of giving a structure, a definition, to 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. [Gibson 1960]

As a matter of fact, we do not yet have a model of robot stimuli, and most researchers accept that any sensory input from the environment or from the robot's internal states can be considered a stimulus. This simplification is of no help, as the wide literature on human and animal stimuli witnesses. If any sensory input can be considered a stimulus, how it would be possible for a robot to discern between a wall and a victim lying on the ground, between the light from a fire and the light of a lamp, between a scream and repeated noise, and so on? Not all inputs require a response, but a stimulus does.

In this article, we address the problem of providing a representation of what a stimulus is for a robot and how it affects the robot's behavior and its tasks under execution. Consider that whereas for a living organism pain is a stimulus that has a direct effect on its survival awareness, for a robot, a similar stimulus could be the lack of a WiFi connection or battery exhaustion, both indicating a loss of vitality. Therefore, in providing such a representation, we answer the question of how to model the stimulus reflex within the robot's structure and language, and we explore how the robot learns the best stimulus-response according to the context of its current task.

In living organisms, stimulus transduction elicits the stimulus reflex through a receptor. In a robot, this transduction is conveyed by its active processes. Consider the robot motion system, which is made of several components, each of which is driven by processes controlling velocity, steering, obstacle avoidance, terrain adaptation, tracking and so on. These processes collect information from the environment and from the robot's internal states. These processes elaborate the collected information and update it, eventually returning it to the robot's inner states to manage its current activities. In other words, the interaction with environment configures a space of information allocating to each process a quantum of information that is transduced to a stimulus at specific peaks of the information manifold. A stimulus induces a request with the

call for a response. A response might appeal to robot components and their processes not yet active at the very moment at which the stimulus is triggered.

In this article, we introduce a preliminary framework to model robot processes, their yields, the stimuli occurrences, and the decision underlying the response to the stimuli. The advantage of this approach is that robot control does not need to be designed a priori but instead can be drawn by its interactions with users who teach the robot when a stimulus occurs and what the possible alternatives are for handling the stimuli. These aspects are not dealt with in any other control methods, including planning, whether reactive, proactive, or declarative. Indeed, a robot that has learned a stimulus-response strategy from several humans will most certainly be more usable than a robot that either has no stimuli at all or has no strategy to respond to stimuli. We also take context into account in order to establish a response cost, and we exploit a theory of actions that models how tasks are chosen and how switching to a new task can occur as a result of a stimulus-response.

The ability to selectively respond to several stimuli and to inhibit inappropriate urges by focusing on the task at hand are well-known in humans as *shifting* and *inhibition* executive functions, respectively [Miller and Cohen 2007; Aron 2007]. The neuroscience theory of control uses inhibition to explain many cognitive activities, especially to explain how an individual presented with several stimuli responds selectively and is able to resist inappropriate urges [Tipper 2001].

Since the early work of Jersild [1927], several studies in neuroscience and cognitive science have led to a better understanding of many of the variables affecting task switching in the context of cognitive control (we refer the reader to Monsell [2003] and the citations therein). These theories have strongly influenced cognitive robotics architectures since the 1980s, as, for example the ATA schema [Norman and Shallice 1986], the principles of goal-directed behaviors in Newell [1990], and, more recently, the CogX architecture [Hawes et al. 2010; Wyatt et al. 2010]. (For a review on the earliest architectures in the framework of the task switching paradigm, see Rubinstein et al. [2001].)

Only recently has the need to model the stimulus-response, and hence the cognitive control underlying task switching and the cognitive functions of shifting and inhibition, become a hot topic in cognitive robotics. The need to model adaptation, to increase the flexibility in handling complex tasks, has led to modeling task switching for robot navigation, motion planning, and control in large-scale environments [Althaus and Christensen 2003], for path planning [Medina Ayala et al. 2012], for recognizing human intentions in conversation [Nakano et al. 2011], and for motor planning to improve tasks execution [Ganesh and Burdet 2013]. Task switching is also addressed in manipulation [Ajoudani et al. 2013; Ding and Fang 2013] and soft robot control [Tao et al. 2012; Li et al. 2013], for modeling multirobot systems [Sung et al. 2013], for learning task space control through the use of different tools [Jamone et al. 2013], for biped walking in rough terrain [Saglam and Byl 2013], and for multiple-space control [Li and Cheah 2012].

Most of the cited works are quite recent, showing increasing interest in the stimulus-response framework for coping with adaptivity and ecological behaviors in an interactive environment. Further examples of these studies exist [Capi 2007; Suzuki et al. 2009; Wawerla and Vaughan 2009; Finzi and Pirri 2010; Durkee et al. 2011; D'Ambrosio et al. 2011]. The major problem to be resolved, as noted in Wawerla and Vaughan [2009], is the switching decision. Experts on task switching claim that this decision incurs 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 [Monsell 2003]. We model this cost by considering both the reconfiguration of the mental state and the interference with the current state. The reconfiguration

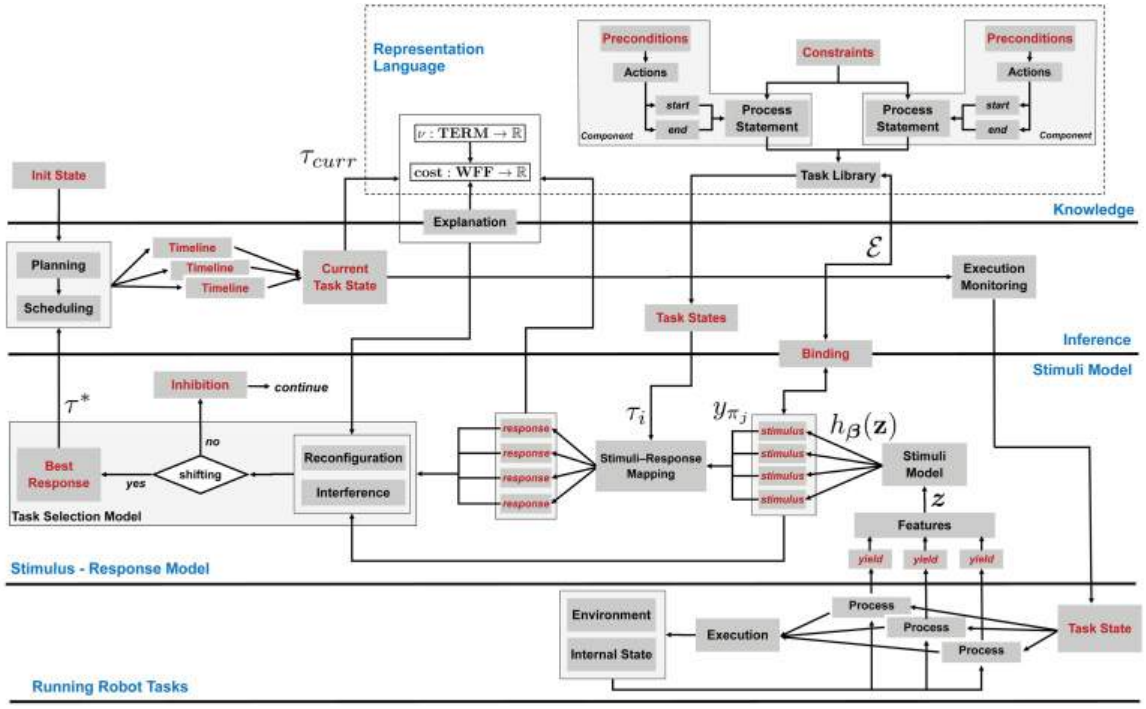


Fig. 1. Work flow of the proposed stimulus-response framework. From bottom to top: the lowest layer indicates a currently active task being executed. The active processes feed the process yields, collecting information from them; the stimulus model selects from the yields those which are stimuli. The stimulus-response model scores the response tasks. The decision is taken by evaluating the payoff of switching to a task suggested by the stimulus-response matrix. In the middle panel, between knowledge and inference, execution monitoring takes care of the actual task execution and its updating, both affecting the mental states; namely, the decision of whether to consent to a response to the stimulus. In the upper panel, a first-order logical formalism is used to model the robot processes, with action preconditions and effects affecting activation costs and motivating causal constraints. This closes the loop between stimulus activation, reasoning, planning, and decision.

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. The decision is made based on a payoff that is computed by considering the cost of updating the current robot state and the risk of continuing with the task under execution. The stimulus-response model illustrated in these pages indicates a new direction in robot cognitive control, and the tests and experiments presented here prove that this is a promising direction.

The stimulus-response framework introduces new emerging methodologies that build cognitive robot control via learning and perception, and we expect that it will spread to several applications that require cognitive control to resolve the continuous struggle a robot has to face in a real-world environment.

2. EXECUTIVE SUMMARY

The flowchart of the proposed robot stimulus-response framework is illustrated in Figure 1. The lower layer represents a robot task under execution. The next level illustrates the stimuli and the response models, described in Sections 3 and 4. The level enclosed between *Inference* and *Knowledge* delineates the work flow linking the decision level to the current execution level. The upper level provides a schema of the system representation in the formal language of Situation Calculus, where the mapping of the stream of information into the domain of reasoning takes place. These levels are described in Sections 5 and 6.

Table I. Examples of Yields of Processes and Their Features

Process \ Yields and Their Features					Stimulus Description
Battery	Level	Power Consumption	Location	-	Exhaustion
WiFi	Strength	Dist. Source	Speed	Traffic	Disconnection
Laser scan	Resolution	Surface	Density	Smoothness	Lack of data, device error
Mapping	Height	Slope	Intricacy	Depth	3D data not available, obstacle
Visual perception	Shape	Dimension	Location	Distance	Identification, Detection
Attention	Light	Color	Motion	Contour	Saliency
Interface	Sound	Id Voice	Utterance	Meaning	Request
Sound	Scale	Frequency	Pitch	Who	Mute, Noise

Each active robot process yields a quantum of information with characteristic features; this quantum of information is called the *yield* of an active process. The features $\mathbf{z}$ of the *yield* are used by the stimulus model to learn a function $h_{\beta}(\mathbf{z})$ (Section 3). This function establishes whether or not a stimulus occurred during the specific process execution.

Example 2.1. Consider the robot process π_{WiFi} managing the connection between the robot and a remote command base station. We expect that the stimulus occurs when a high signal becomes low or disappears. So the process π_{WiFi} yields a quantum of information specified by a set of features $\mathbf{z}_{WiFi}$, such as those indicated in Table I. The robot, with the help of users, learns to discriminate a stimulus from ordinary information via the function $h_{\beta}(\mathbf{z}_{WiFi})$, discerning between typical $\mathbf{z}_{WiFi}$ values and odd ones. Therefore, the outcome of the learned function h_{β} is a value $y = 1$ to fire the stimulus of WiFi disconnection and $y = 0$ otherwise.

If the stimulus occurs, the robot has to choose a possible response (e.g., going back to the last position where it was connected), or it can inhibit the stimulus and continue its task even though it is disconnected from the operator. Therefore, there are two issues that need to be modeled:

(1) identify the task that is a possible response to the stimulus;
(2) decide whether to go on with the current task or to switch to the identified response task.

The first item is dealt with by filling a score matrix $\mathbf{A}$ whose values are estimated via factorization, as described in Section 4.

On the other hand, the decision to either shift from the current task to a new task in so responding to the stimulus, or to inhibit the stimulus, is based on the payoff of switching. This is computed considering the risk of continuing the current task, without taking into account the stimulus, and the effort required to fulfill the stimulus. In turn, the effort is computed considering two costs: (1) the cost to *reconfigure* the current robot state to the new state that switching would lead to and (2) the cost to resolve the *interference* due to the interruption of the current task. These costs are computed by considering the preconditions and effects of each action involved in both the processes to be interrupted and in the ones to be activated (Section 6). To bind the information yielded by a process to the domain of reasoning, a special functional $\mathcal{E}$ is used, mapping the process terms of the representation language to the corresponding values of both the yields and the stimuli at execution time (Section 5).

Example 2.2. Suppose the robot is executing task τ_{10} (see Table VII, in Appendix A) that requires the robot to approach a victim in a rescue area. The processes composing this task are listed in Table VII. Assume that the yield of process π_{WiFi} (named

connection management in Table VII) fires a stimulus, namely, the current values of its yields $\mathbf{z}_{WiFi}$ induce $h_\beta(\mathbf{z}_{WiFi}) = 1$. Let the stimulus-response matrix $\mathbf{A}$ suggest that the response task is τ_4; namely, *recovery WiFi signal*. To start the task *recovery WiFi signal,* all the required processes have to be activated with their preconditions and postconditions; therefore, the reasoning system of the robot has to be informed of the stimulus in order to make a decision by computing the payoff of switching. The mapping $\mathcal{E}$, applied to the features, transforms them into values that are terms of the representation language, thus allowing the cost computation to take place.

The article is organized as follows. In the next section, we introduce a representation of the stimuli. In Section 4, we describe the computation of the stimulus-response matrix. In Section 5, we describe the fundamental formalism needed to define concepts such as tasks and processes that are needed to model the transduction between stimuli and processes. In Section 6, we illustrate both how switching costs are computed and how the payoff for switching is obtained. Section 7 evaluates, with some experiments, the performance of the proposed stimulus-response model at managing task switching when incoming information acknowledges new situations to be faced. In particular, results are confronted with replanning at execution time as proposed in the context of CRAM [Beetz et al. 2010] and DTPDDL, implementing POMDP [Talamadupula et al. 2010; Göbelbecker et al. 2011]. The experiments prove the correctness of the robot's behavior in the presence of different stimuli, cover many aspects that have not been adequately treated in the article, and present the whole picture as simply as possible. In the Appendix, we provide some implementation details of the robotic platform and its functionalities; we also use tables to illustrate the implementation of several robot components, their processes, and the stimuli yielded by those processes; finally, we show the stimulus-response matrix developed for a set of stimuli and tasks.

3. THE STIMULUS MODEL

In this section, we introduce what we define as a robot stimulus and how we estimate the presence of a stimulus given the information gathered by a process.

Consider the following events applied to a robot, not necessarily in parallel: the battery is getting low, the WiFi has a very weak or absent signal, saliency is detected in a sequence of frames, the memory is surcharged, the communication with the operator is clogged, a request pops up on the interface, and someone is talking at the dialog box. A robot has to deal with hundreds of events like these. These events are stimuli, in the sense that they trigger some alert on the information process transduced by an executive process of an active robot component such as, for the example, the battery, the WiFi, the attention system, the mapping system, the engines, the dialog system, the interface, or the obstacle avoidance system. Three concepts are at the core of stimulus modeling: (1) the information flow caused by an active robot process, (2) the states and events inducing a stimulus, 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 opposite assumption under which we have been operating for centuries that stimuli are necessarily and intrinsically meaningless. [Gibson 1960]

Consider a robot process π being executed among a number of others under execution. This process π has a domain of interest defined by the values it controls; some example are given in Table I. These values are the realizations of a stochastic variable $\mathbf{X} : \Omega^d \mapsto \mathbb{R}^d$, and we identify them as the *yield* of the active process π. At each instant time t,

if the process is active, then the stochastic variable is realized and the values it takes are known; either they induce a stimulus for the process π or they do not.

More precisely, the yield of a process π forms an observation vector $\mathbf{z} = (z_1, \ldots, z_d)^\top$, $\mathbf{z} \in \mathbb{R}^d$ and whether these values induce a stimulus or not is the objective of a function $h_\beta : \mathbb{R}^d \times \boldsymbol{\beta} \mapsto [0, 1]$ to be learned with parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_d)^\top$ and arguments being the yield of the process.

A trial is the set of observations gathered in a discrete time interval T ; if n trials are performed by n operators, we obtain a training set $\mathcal{T}$ of size $(nT \times (d+1))$. The training set is of the form $\mathcal{T} = (\langle \mathbf{z}_1, y_1 \rangle, \ldots, \langle \mathbf{z}_{nT}, y_{nT} \rangle)^\top$, with $\mathbf{z}_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}$.

The logistic regression model can be used to estimate the outcome y. We recall that the logistic regression model computes the probability of the outcome y as:

$$h_\beta(\mathbf{z}) = \mathcal{B}(g(a)), \tag{1}$$

where $\mathcal{B}(x)$ is the Bernoulli density $\mu^x(1-\mu)^{(1-x)}$ with $\mu \in [0, 1]$; a is the activation, namely, $a = \boldsymbol{\beta}^\top(1, \mathbf{z}^\top)^\top$; and g is the sigmoid function:

$$g(a) = \frac{1}{1 + \exp(-a).} \tag{2}$$

The goal is to estimate the parameters $\boldsymbol{\beta}$ of the function h_β.

It is important to note that for this model to predict accurately the occurrence of a stimulus with the outcome value y, given the features $\mathbf{z}$ of the specific process yields, it is required that the outliers introduced during data gathering are kept under control before the estimation of the parameters $\boldsymbol{\beta}$ takes place. In fact, in addition to the difficulty of obtaining comparable trials, the training set $\mathcal{T}$ is extremely noisy due to operators mistakes made while a trial is running.

Since the noise is mainly introduced by the operator in the difficult task of deciding whether an observation is or not a stimulus, we have devised a method of rejecting most of the outliers that severely affect parameter estimation. Indeed, we exploit some prior knowledge about stimuli/nonstimuli for each process, and we use this to define a function that is consistent with the *typical behavior* of the process and that rejects wrong data in the training set. The proposed function maps to a rejection region all those outcomes that are classified as stimuli but whose range of yields should in principle assign them to a nonstimuli classification since stimuli are extraordinary events. The function defining the rejection region is:

$$\gamma(\mathbf{z}, y) = \begin{cases} 1 & \text{iff } y = 1 \text{ and } \mathbf{z} \notin \mathit{RejRegion} \\ 1 & \text{iff } y = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Here, *RejRegion* is the rejection region that still need to be defined. This is done according to the following steps:

(1) Fit nonparametric kernel density to the data $(\mathbf{z}_1, \ldots, \mathbf{z}_{nT})$ using a Gaussian kernel.
(2) Find the modes of the density. The modes indicate where the data are more dense; in other words, they indicate the values identifying the most typical behavior of the process yield.
(3) Select the data falling in the regions around the modes $\{\hat{\mathbf{z}}_q\}, q = 1, \ldots, k_M$, with k_M being the number of modes, and lock these data into an hyperellipsoid whose dimension is estimated according to the Hotelling test, assuming that these data come from a Gaussian distribution.
(4) The confidence region for the mean of the selected data is the required rejection region *RejRegion*.

For the first step, we consider a smoothing kernel over the process yield on a d-dimensional grid. Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ be a point in the grid, and let $\mathbf{z}_k \in \mathbb{R}^d$ be an observed yield, $1 \leq k \leq nT = N$; let $\upsilon = [(\frac{\mathbf{x}-\mathbf{z}_k}{h^d})^\top(\frac{\mathbf{x}-\mathbf{z}_k}{h^d})]^{1/2}$, with $h > 0$ the kernel window. We consider a radial symmetric Gaussian kernel $K_{hd}(\upsilon) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}\upsilon^2)$. A nonparametric estimate for the observations that are close to point $\mathbf{x}$ is:

$$f_{hd}(x_1, \ldots, x_n) = \frac{1}{N} \sum_{k=1}^{N} c K_{hd}(\upsilon_k). \tag{4}$$

Here, c is the normalization term that, by the fact that $\int_{-\infty}^{\infty} K(\upsilon)d\upsilon = 1$, is $(h^{d^2}(2\pi)^{(d-1)/2})^{-1}$. The gradient of the density is

$$\nabla f_{hd}(x_1, \ldots x_d) = \left\{ \frac{c}{N} \sum_{k=1}^{N} -\frac{1}{h^{2d}} K_{hd}(\upsilon_k)(x_j - z_{kj}) \right\}_{j=1,\ldots,d.} \tag{5}$$

Then, by some simple algebraic transformation and hiding in c the constant term we obtain:

$$\nabla f_{hd}(x_1, \ldots, x_d) = \left\{ f_{hd}(x_1, \ldots, x_d) \left(\frac{\sum_{k=1}^{N} K_{hd}(\upsilon_k) z_{kj}}{\sum_{k=1}^{N} K_{hd}(\upsilon_k)} - x_j \right) \right\}_{j=1,\ldots,d.} \tag{6}$$

Local maxima of f_{hd} are computed by gradient ascent; these are the modes $\widehat{\mathbf{z}}$ of the function f_{hd}. Let us consider these modes $\{\widehat{\mathbf{z}}_q\}$, $q = 1, \ldots, k_M$, with k_M the number of modes. Assume these data come from a Gaussian distribution with means μ_q and covariances Σ_q, and let $\overline{\mathbf{z}}_q$ and $\mathbf{S}_q$ be, respectively, the sample mean and sample covariance of the q-th region under the q-th mode. The confidence region at level $(1 - \alpha)$ for the data mean μ_q is:

$$CR_q = \{\mu_q \in \mathbb{R}^d \mid (\mu_q - \overline{\mathbf{z}}_q)^\top \mathbf{S}_q^{-1} (\mu_q - \overline{\mathbf{z}}_q) \leq \frac{d}{N-d} F_{(1-\alpha);d,N-d}\}, q = 1, \ldots, k_M. \tag{7}$$

Here, $F_{(1-\alpha);d,N-d}$ is the $1 - \alpha$ quantile of the F-distribution with d and $N - d$ degrees of freedom. That is, we are confident at a level $(1 - \alpha)$ that CR_q contains μ_q; namely, $p(\mu_q \in CR_q) = (1 - \alpha)$. Now, $(\mu_q - \overline{\mathbf{z}}_q)^\top \mathbf{S}_q^{-1}(\mu_q - \overline{\mathbf{z}}_q), q = 1, \ldots, k_M$ are hyperellipsoids with center μ_q and semi-axes of length $\sqrt{D_q^2 \lambda_{sq}}$, where λ_{sq} is the s-th eigenvalue of S_q^{-1}. Since $D_q^2 = \frac{d}{N-d} F_{(1-\alpha);d,N-d}$ is the maximal ellipsoid surface, enclosing the q-th confidence region, it follows that these regions correspond to the *most typical* behavior of the yield of the process π under consideration. Hence, these are rejection regions for stimuli—regions where it is most unlikely that a stimulus can occur. Therefore, we are finally arrived at the definition of rejection region *RejRegion*:

$$RejRegion = \{\mathbf{z} \in \mathbb{R}^d \mid (\overline{\mathbf{z}}_q - \mathbf{z})^\top \mathbf{S}_q^{-1} (\overline{\mathbf{z}}_q - \mathbf{z}) \leq D_q^2, q = 1, \ldots k_M\}. \tag{8}$$

As noted earlier, k_M is the number of modes of the nonparametric density. Therefore, according to Equation (3), the set of samples for which $\gamma(\mathbf{z}_k, y_k) > 0, k = 1, \ldots, N$ (recall that $N = nT$) is the optimal one for classification.

Now, given a complete set of good samples $\mathcal{T}^\star = (\langle \mathbf{z}_1, y_1 \rangle, \ldots, \langle \mathbf{z}_N, y_N \rangle)$, the model for stimulus prediction is provided by logistic regression. Namely, the logistic regression coefficients $\boldsymbol{\beta}$ are obtained from the good trial set $\mathcal{T}^\star$ for the yield of process π by

Table II. Components, Processes, and Stimuli

Component Description	Process Description	Stimulus Description
Battery	Power Level Monitoring	Battery Exhaustion
WiFi	Connection Management	WiFi Disconnection
Robot Engine	Temperature Monitoring	Temperature Level
Visual Perception	Detecting Object/Person	Detected Object/Person
Memory	Data Writing	Write Access to Memory Full
Dialogue	Human to Robot Exchanging msg	Human to Robot Message

minimizing the following cost function:

$$L(\boldsymbol{\beta}) = -\frac{1}{n}\left(\sum_{i=1}^{N} y_i \log h_\beta(\mathbf{z}_i) + (1-y_i)\log(1-h_\beta(\mathbf{z}_i))\right) + \frac{\xi}{2n}\sum_{j=1}^{d}\beta_j^2. \tag{9}$$

Here, ξ is the regularization parameter and $\sum_{j=1}^{d}\beta_j^2$ the regularization term; these are used just in case the features induce a high-dimensional space, which in general is kept ≤ 5 to manage the nonparametric density. The Newton-Raphson algorithm used to minimize Equation (9) requires an update step expressed as a least square step, namely, as an iteratively reweighted least square. Then, given a new observation $\mathbf{z}^\star$, $h_\beta(\mathbf{z}^\star)$ is the hypothesis returning the binary response within the variable y. More precisely:

$$y = \begin{cases} 1 \text{ if } h_\beta(\mathbf{z}^\star) \geq 0.5 \\ 0 \text{ if } h_\beta(\mathbf{z}^\star) < 0.5. \end{cases} \tag{10}$$

We have thus provided a model to represent the stimuli and for learning when such stimuli occur. We have also introduced the principle that a large region of nonstimuli for each process yield can be identified around the most typical values of the yield to simplify noisy data collection. Note that this concept of typical values for the process yield grants the possibility of estimating h_β with quite few trials.

4. THE STIMULUS-RESPONSE MATRIX

Given a process π, we have introduced the function h_β that takes as input the yield of a process and determines whether a stimulus is triggered or not, returning a value 1 if this is the case and a value 0 otherwise. If a stimulus occurs, it is necessary to score all possible response tasks before any decision can be taken.

In this section, we show how to score the relation stimulus response, which amounts to scoring the task that would eventually be chosen to respond to the triggered stimulus. Here, a task is a set of processes; see Table II for an excerpt of the structures linking robot components, processes, and stimuli, and the more detailed Tables VI, VII, and VIII in Appendix A. Therefore, while a task is being executed, a stimulus can be triggered from any of the active processes of the task. To address each case, given a limited number of tasks for a robot, we introduce a stimulus-responses matrix $\mathbf{A}$ that scores each pair: The score matrix $\mathbf{A}$ is formed by two classes of entities, which we refer to as *Tasks* (rows) and *Stimuli* (columns), respectively (see Table VII and Table VIII in Appendix A). Note that a task is built out of many processes, yet each process can yield a single stimulus; namely, there is a bijection between each process and the stimulus it yields.

Matrix $\mathbf{A}$ is initially built using a number of experiments and further completed by estimating the unknown values for the stimulus-response pairs with the regularized low-rank matrix factorization method [Zhang et al. 2012; Chen 2008; Buchanan and Fitzgibbon 2005; Canny 2002].

The experiments for filling **A** are short-term missions performed by operators who are well informed about the task library, the processes, and the stimuli that the robot can handle. Each operator is assigned a task from the task library (see Table VII in Appendix A, as described in Section 7). During the trial, a stimulus is triggered for an active process π_j. Upon the occurrence of the stimulus, the operator selects the task τ_i that he or she considers the most appropriate to respond to the presented stimulus from among those reported in Table VII (Appendix A) using a graphical user interface (see Figure 11 in Appendix A). Note that, during the experiment, the operator teleoperates the robot. The experiment terminates when the operator signifies his or her task choice. Data from each trial, reporting the occurrence of the pairs $\langle \tau_i, y_{\pi_j} \rangle$, are collected, and the corresponding entries a_{ij} of matrix **A** (see Table IX and Table X) are filled. When operators did not select the corresponding stimulus-task pair, a *dash* sign is added.

The problem now is to complete the matrix by finding the missed values. These are recovered by factorizing matrix **A** into the product of two smaller matrices, $\mathbf{P} \in \mathbb{R}^{m \times d}$ and $\mathbf{Q} \in \mathbb{R}^{n \times d}$, of rank d[1]. Given that **A** has dimension $(n \times m)$—namely, n tasks and m process yields (here, $n = 13$ and $m = 42$)—and rank d, in order to complete **A**, it is necessary to find two factors explaining **A**: an $(n \times d)$ matrix **Q** and an $(m \times d)$ matrix **P**, such that $\mathbf{A} = \mathbf{Q}\mathbf{P}^\top$. Thus, it is possible to minimize the error norm on the known values.

Let **W** be an indicator matrix of the same size as **A** in which 0's correspond to missing elements of **A** and 1's correspond to observed data; then, the problem can be reformulated as:

$$\epsilon(Q, P) = \|W \odot (\mathbf{A} - \mathbf{Q}\mathbf{P}^\top)\|_F^2 + \lambda_1 \|\mathbf{Q}\|_F^2 + \lambda_2 \|\mathbf{P}\|_F^2. \tag{11}$$

Here, $\odot$ is the Hadamard component-wise matrix product, $\|\cdot\|$ is the Frobenius norm, and λ_1 and λ_2 are regularizing constants. Several approaches have been proposed to minimize this error, considering that if one of **Q** or **P** are known, there is a closed-form solution for the other that minimizes Equation (11) (see, e.g., the dumped Newton algorithm introduced by Buchanan and Fitzgibbon [2005] and the Wieberg algorithm of Takayuki Okatani and Deguchi [2006]). Here, we have chosen a version of the alternating-least-squares with weighted-λ-regularization [Zhou et al. 2008], in which the steps to obtain **Q** and **P** are summarized as follows. Initialize **Q** and **P** to small random values and let w_{ij} be the (i, j) element of the indicator matrix **W**; then, the schema (11) can be rewritten as follows:

$$\epsilon(\mathbf{q}_1, \ldots, \mathbf{q}_n, \boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_m)) = \frac{1}{2} \sum_{(i,j)} w_{ij} \left(\boldsymbol{\rho}_j^T \mathbf{q}_i - a_{ij}\right)^2 + \frac{b_1}{2} \sum_{i=1}^{n} \sum_{k=1}^{d} q_{ik}^2 + \frac{b_2}{2} \sum_{j=1}^{m} \sum_{k=1}^{d} \rho_{jk}^2. \tag{12}$$

Here, the first term of Equation (12) corresponds to the square error term, and the last terms are the regularization terms of $\mathbf{q}_i$ and $\boldsymbol{\rho}_j$, respectively. The terms b_1 and b_2 are the regularization parameters. Local optima of the objective function in Equation (12) are found via Newton-Raphson gradient descent. The update rules, for every $i = 1, \ldots, n$ and for every $j = 1, \ldots, m$ are given by the following equations:

$$\begin{aligned} q_{ik} &:= q_{ik} - \zeta_1 \left(\sum_j w_{ij} \left(\boldsymbol{\rho}_j^T \mathbf{q}_i - a_{ij}\right) \rho_{jk} + b_1 q_{ik} \right) \\ \rho_{jk} &:= \rho_{jk} - \zeta_2 \left(\sum_i w_{ij} \left(\boldsymbol{\rho}_j^T \mathbf{q}_i - a_{ij}\right) q_{ik} + b_2 \rho_{jk} \right) \end{aligned} . \tag{13}$$

[1]Here, the letter d is used for the matrix rank, while in the previous section it was used to define the size of observations.

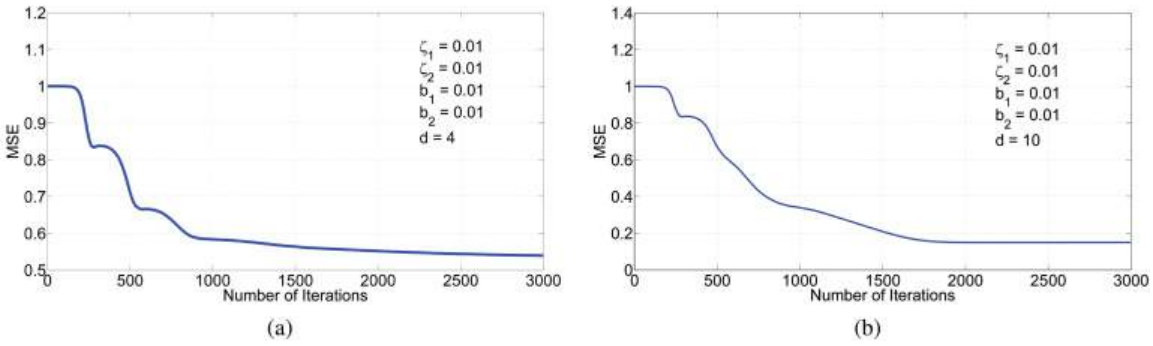


Fig. 2. Estimation of accuracy of the parameters ζ and b obtained in Equation (13) given a model with (a) rank of $\mathbf{A} = 4$ and (b) rank of $\mathbf{A} = 10$.

Once both $\mathbf{q}_i$ and $\boldsymbol{\rho}_j$ have been estimated, then the score matrix $\mathbf{A}$ can be completed by estimating the missing values of the rates $\widehat{a}_{ij}$ as

$$\widehat{a}_{ij} = \boldsymbol{\rho}_j{}^{\top}\mathbf{q}_i \qquad \forall w_{ij} = 0. \tag{14}$$

The empirical prediction quality is typically low (see Figure 2(a)). Note that, as expected, the Mean Squared Error (MSE) considerably decreases with a higher rank d of the matrix $\mathbf{A}$ and, consequently, the empirical prediction quality gets higher (see Figure 2(b)). Figures 2 shows the accuracy of the prediction when varying the rank of matrix $\mathbf{A}$.

An example of the score matrix $\mathbf{A}$ filled by the trials values for the implemented task library with their processes and stimuli is illustrated in Tables IX and X , in Appendix A. The completed stimulus-response matrix, with the whole score set completed, is shown in Tables XI and XII in the Appendix A.

5. STATES, PROCESSES, AND TASKS

The state of the robot is the set of processes that are active at a specific interval time $\Delta t = [t^-, t^+]$, and it coincides with the current task in execution (see the tasks given in Table VII in Appendix A). The decision to switch or not to a response task, chosen according to the score matrix $\mathbf{A}$ introduced in the previous section, is a matter of how difficult it is to switch, how many processes need to be interrupted, and how many new processes need to be activated.

To understand the complications intervening in this decision, we briefly introduce the representation of processes in the framework of their control, execution, and monitoring, together with the reasoning mechanism required to infer the facts that need to be appraised before eventually switching to a new task. Namely, switching requires us to infer an appropriate explanation for enabling all the preconditions for activating the response task. In this section, we introduce the following concepts to address this problem:

(1) The representation of actions and processes.
(2) The mechanism generating processes to execute a chosen task.
(3) The constraints between processes working in parallel.
(4) The representation of the robot's current task state.

Tasks and processes are modeled in the action theory $\mathcal{A}$ based on the formalism of Situation Calculus. Planning, the decision to switch, and execution monitoring are all expressed in 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, U for the set of boolean values $\{0, 1\}$, and W for every other object. Given the sorted domain, the representation language $\mathcal{L}$ (via its signature $\Sigma_{\mathcal{L}}$) 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 Pirri and Reiter [1999] for a full description of the formalism. The representation language is used to provide the robot with a reasoning and knowledge structure about actions by specifying their preconditions and effects, and about both the processes and the constraints linking their parallel execution.

A process is a predicate $\pi(\mathbf{x}, u, t, do(a, s))$ taking as arguments a single term u of sort boolean for the stimulus, a term t of sort time, a single term $do(a, s)$ of sort situation, and terms $\mathbf{x} = (x_1, \ldots, x_n), n \geq 0$ of any other sort. The special function do of sort situation is an action sequence builder that defines a situation as a list of actions. More precisely, do is a function taking as arguments a term of sort action and a term of sort situation and returning a term of sort situation $do : A \times S \mapsto S$, which is a list of actions. Actions are functions, and when the term of sort situation explicitly mentions the sequence of actions it builds, we denote the term with σ. Through its arguments, a process carries the information about the history of all processes of a component via all its start and end actions, activated up to the time t. For example, consider one of the processes given in Table VI, Appendix A: The *connection management* of the *WiFi component* is represented in language $\mathcal{L}$ by the following transition statement:

$$\begin{aligned}\pi_{WiFi}(\mathbf{x}, u, t, do(a, s)) \equiv [a = start_{WiFi}(\mathbf{x}, t) \wedge u = 0] \vee \big(\exists t'.t' \leq t \wedge \pi_{WiFi}(\mathbf{x}, u, t', s) \\ \wedge(\forall t''\forall u.(t'' > t' \wedge t'' \leq t) \rightarrow \mathcal{E}(\mathbf{x}, t'') = u \rightarrow a \neq end_{WiFi}(\mathbf{x}, t''))).\end{aligned} \tag{15}$$

Here, $start_{WiFi}$ and end_{WiFi} are actions, and t, t', t'' are variables of sort time. The function $\mathcal{E}$ maps terms of the representation language $\mathcal{L}$ to the values identified by the stimulus model (Section 3). This function takes as input the realization of the yields of the process when the process is active, and it returns the value $u = 1$ if the signal elicits a stimulus and $u = 0$ otherwise. This is called a *transition axiom,* and it specifies that the robot process managing WiFi connection is active if either the action $start_{WiFi}$ 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 by a disconnection notified by a stimulus nor it was ended by the action end_{WiFi} independently of the stimulus value. The transduction of a stimulus realization to a term of the language $\mathcal{L}$ requires the existence of a set of transformations binding terms and domains at execution time. This is better detailed by the following remark.

Remark 5.1 *(Transduction between stimuli and processes).* A process yield and its stimulus value are both linked to the signature $\Sigma_{\mathcal{L}}$ of the action language $\mathcal{L}$ by the mapping $\mathcal{E}$. The domain of $\mathcal{L}$ and of the processes' yields are linked by the interpretation I of $\mathcal{L}$.

To prove this, we first introduce some auxiliary mappings. Let L be a set of labels, and let T be time. We introduce the function $\mathcal{E}^+ : \mathbb{R}^n \times L \rightarrow \Sigma_{\mathcal{L}}$, the function $\mathcal{E}^\star : [0, 1] \times T \rightarrow \Sigma_{\mathcal{L}}$, and finally the mapping $re : \mathbb{R}^n \times T \mapsto \mathbb{R}^n$ and the mapping $\mathcal{E} : \Sigma_{\mathcal{L}} \times T \mapsto \Sigma_{\mathcal{L}}$. To these functions, we add the interpretation function I mapping terms of the language into the interpretation domain, and we assume that equality is interpreted by itself. We have to show that, via $\mathcal{E}^\star$, re, and $\mathcal{E}^+$, $\mathcal{E}$ links the feature terms with terms of $\mathcal{L}$ and that the interpretation links, via the mappings, the sample domain where the yields take values with the domain of interpretation for $\mathcal{L}$. Let us assign to each defined process $\pi \in \mathcal{L}$ a label, and let $\mathbf{X} : \Omega^n \mapsto \mathbb{R}^n$ be a multivariate random variable whose realizations are the observed values $\mathbf{z}$ of the yield of π at time t. The function re serves to specify the realization with respect to the process yields and time, namely, $re(\mathbf{X}, t) = \mathbf{z} \in \mathbb{R}^n$. Now $\mathcal{E}^+$ maps real numbers, labeled by the specific process of which they are the yield, into terms of the language such as

$\mathcal{E}^+([z_3, z_4], L_{WiFi}) = (signal\_strength, distance\_from\_source)$ and so on. Similarly, $\mathcal{E}^\star$ takes the outcome of the learning function $h_\beta(\mathbf{z})$ (see Section 3) and maps it to a term of the language. These terms have a semantics in the domain D^n of $\mathcal{L}$, so that real values labeled by a process name need to be paired with the interpretation of the terms of the specific process. So, for example, the interpretation of the term $signal\_strength \in \Sigma_\mathcal{L}$, taken as argument by the process π, has to be linked to the corresponding element of the sample domain Ω^n. Therefore, when the process is active, then t is instantiated, the transduction closes the loop between the different terms, and binding occurs on the appropriate instances in the respective domains according to the sorts requirements:

$$\begin{aligned}
&re(\mathbf{X}, t) = \mathbf{z} \in \mathbb{R}^n \\
&\mathcal{E}^+(\mathbf{z}, L) = \mathbf{x} \in \Sigma_\mathcal{L} \\
&\mathcal{E}^\star(h_\beta(\mathbf{z}), t) = u \in \Sigma_\mathcal{L} \\
&u^I \in \{0, 1\} \subset D \\
&\mathcal{E}(\mathcal{E}^+(re(\mathbf{X}, t), L), t) = \mathcal{E}(\mathbf{x}, t) = \mathcal{E}^\star(h_\beta(\mathbf{z}), t) = u \in \Sigma_\mathcal{L} \\
&\mathcal{E}^I(\mathcal{E}^+(re(\mathbf{X}, t), L), t)^I = \mathcal{E}^{\star,I}(h_\beta(\mathbf{z}), t)^I = u^I \in \{0, 1\} \subset D.
\end{aligned} \tag{16}$$

This shows that the mappings $\mathcal{E}^\star$, $\mathcal{E}^+$, re, and $\mathcal{E}$, together with the interpretation I of the language, are sufficient to define the transduction between stimuli and processes, both linking the terms of the language and mapping the outcome into $\{0, 1\} \subset D$.

A process π is active between a $start_\pi$ and an end_π action. To activate a *start* or *end* action, a special predicate *Poss* tells the robot whether the action can be done given the current situation s and time t, for example:

$$\begin{aligned}
&Poss(end_{WiFi}(\mathbf{x}, t), u, s) \equiv \\
&\quad (at(home, t, s) \wedge \neg ask(operator, t, s) \vee u = 1) \wedge \pi_{WiFi}(\mathbf{x}, u, t, s)
\end{aligned} \tag{17}$$

tells the robot that the WiFi process can be ended if it is at home and if, for example, there are no operator requests running or a disconnection signal is detected. This statement is called a *precondition axiom*. Note that a state is defined by all the predicates (processes and facts) that are verified at the specific situation and time in the intended model. Thus, if $\mathcal{M}$ is the model of the language satisfying the whole theory, we might say that $\mathcal{M} \models \bigwedge \varphi(t, \sigma)$ to indicate that the finite set of sentences $\bigwedge \varphi$ holds at time t in situation σ in the model, and so specify a finite state of the system. On the other hand, we can also write $\mathcal{A} \models \exists \mathbf{x}, u, t.\pi_C(\mathbf{x}, u, t, \sigma)$ to indicate that all the models (actually the intended model of the axiomatization) of the action theory $\mathcal{A}$ are also models of the process π_C at some time t and situation σ, thus indicating that the situation $\sigma = do(A_n(\mathbf{x}), do(A_{n-1}(\mathbf{x}), \ldots, do(A_1(\mathbf{x}), S_0))\ldots)$ returns a sequence of actions that governs the processes that can be activated in S_0 to perform some task involving the component C.

Robot components are both hardware devices and software modules such as engines, dialog, vision, mapping, planning, path planning, and so on, all handling several processes and needing to be regulated not only by transition and precondition axioms, but also by both temporal and spatial relations. To handle relations between processes, a parallelization of situations into several timelines has been introduced [Finzi and Pirri 2010], and each timeline assesses 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. Parallel situations are handled by a set of situations called *bags of timelines*. Time constraints are based on the definition of the greatest start time and least end time, thus bounding a process activity within a start and end time, up to a fixed time t. The greatest start time is defined as:

$$greatestStart_\pi(\mathbf{x}, a, t) \stackrel{def}{\equiv} \exists t'.a = start_\pi(\mathbf{x}, t') \wedge \neg \exists t'', a''.t' \leq t'' \leq t \wedge a'' = end_\pi(\mathbf{x}, t''). \tag{18}$$

An analogous definition is given for *leastEnd*, the least end time for an action, thus characterizing the Δt inside which a process is certainly active or certainly not active. Given this definition, a time relation such as *during* is defined as follows:

$$\begin{aligned}\{\pi_i(\mathbf{x}, u, t, do(a, \sigma))\ \mathbf{during}\ \pi_j(\mathbf{x}', u', t', do(a', \sigma'))\}[\mathbb{s}, \mathbb{s}'] \stackrel{def}{\equiv} \\ \sigma \in \mathbb{s} \wedge \sigma' \in \mathbb{s}' \wedge \pi_i(\mathbf{x}, u, t, do(a, \sigma)) \wedge \pi_j(\mathbf{x}', u', t', do(a', \sigma')) \wedge \\ [greatestStart_{\pi_i}(\mathbf{x}, a, t) \geq greatestStart_{\pi_j}(\mathbf{x}', a', t')] \wedge \\ [leastEnd_{\pi_i}(\mathbf{x}, a, t) \leq leastEnd_{\pi_j}(\mathbf{x}', a', t')].\end{aligned} \tag{19}$$

Here, $\mathbb{s}$ and $\mathbb{s}'$ are bags of timelines; specifically, timelines of different robot components. Note that in the formulas specifying constraints (as in Equation (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 Allen [1983], among which we consider the set $\{during, before, meet, start, finishes, overlaps, equal\}$.

For example, the Equation (20) 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:

$$\begin{aligned}[\pi_{path-planning}(\mathbf{x}, u, t, \sigma)\ \mathbf{during}\ \pi_{mapping}(\mathbf{x}', u', t', \sigma')] \\ \mathbf{before}\ \pi_{approach}(\mathbf{x}'', u'', t'', \sigma'')[\mathbb{s}, \mathbb{s}', \mathbb{s}'']\end{aligned} \tag{20}$$

A constraint formula is indicated by $\pi_i \mathbf{op} \pi_j[\mathbb{s}_i, \mathbb{s}_j]$, and a set of constraint formulas is indicated by $\mathcal{C}$. Note that constraints are special formulas because only the definiens (namely, the left hand side of the definitions) are expressed in the language $\mathcal{L}$.

Different approaches for defining constraints in a first-order language have been proposed [Baar et al. 2001; Block et al. 2006; Morris et al. 2001; McDermott 2003; Fox and Long 2003; Carbone et al. 2008; Wittocx et al. 2010]. A definition for each time constraint between processes, according to linear interval-based time [Allen 1983], is provided in Finzi and Pirri [2010].

Similar to temporal constraints, spatial constraints are defined following the topological relations introduced in Randell et al. [1992]. Spatial and time constraints (see Wolter and Zakharyaschev [2000] and Finger and Gabbay [1996]) are integrated in the framework.

Now we must specify what happens if the robot needs to switch to a different task and stop the current one. If at time t, the robot has to infer the processes needed to execute a task , it may be that for some process the starting action $start_\pi$ or the ending action end_π are not derivable; namely, $\mathcal{A} \cup \mathcal{C} \not\models \exists \mathbf{x}, t\ Poss(start_\pi(\mathbf{x}, t), \sigma)$. 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 the formula $\psi \in \mathcal{L}$, such that:

$$\mathcal{A} \cup \mathcal{C} \cup \{\psi\} \not\models, \text{ and } \mathcal{A} \cup \mathcal{C} \cup \{\psi\} \models \exists \mathbf{x}, t\ Poss(start_\pi(\mathbf{x}, t), \sigma), \tag{21}$$

where the proviso $\mathcal{A} \cup \mathcal{C} \cup \{\psi\} \not\models$ means that the explanation added to the theory does not imply the contradiction. Explanations can be computed using first-order abduction (see, e.g., Marquis [1991] and Cialdea and Pirri [1993]). Here, $\mathcal{A}$ is the action theory, $\mathcal{C}$ is the set of temporal-spatial constraints, and ψ is the formula explaining the observation $\exists \mathbf{x}, t\ Poss(start_\pi(\mathbf{x}, t), \sigma)$; that is, ψ is the formula to be added to ensure the inference.

Examples of such processes are reported by Finzi and Pirri [2005, 2010], Carbone et al. [2008], Pirri and Reiter [2000], and Pirri [2011] in the framework of Situation Calculus [Pirri and Reiter 1999]. However, processes are defined in several other frameworks, as, for example, in STRIPS, Graphplan, and PDDL [Blum and Langford

1999; McDermott 2003; Fox and Long 2003; Fikes and Nilsson 1971]. Each of these has its own advantages in terms of expressive power, computational feasibility, and constraints expressible in the language. When robot planning requires the orchestration of all components according to temporal constraints, the planning system usually takes advantage of temporal networks to manage temporal constraints [Allen 1983; Dechter et al. 1991; Vilain et al. 1986; Block et al. 2006; Morris et al. 2001]. Here, we use Situation Calculus to model high-level control of robot processes because of its expressive power both for managing parallel processes and time-space constraints and for defining interference and reconfiguration cost. The stimuli and stimulus-response model proposed in this work could be built on any action theory that handles robot components, processes, and constraints and is able to express robot mental state at a precise time, together with all cause-effect relations affected by the response. This need is clarified in the next section, where we compute the cost of an explanation.

6. TO SWITCH OR NOT TO SWITCH

We are now at a point in the formalization of the stimulus-response where we can illustrate how a decision can be taken about whether to respond to a stimulus. To illustrate the decision problem, we continue with the WiFi disconnection example.

Example 6.1. The robot is executing task $\tau_{10,}$ expecting to approach victims in a rescue area. At time t, the process π_{WiFi} get disconnected, $h_\beta(\mathbf{z}_{WiFi}) = 1$; hence $\mathcal{E}(\mathbf{x}_{WiFi}, t) = 1$. Therefore, the action end_{WiFi} becomes possible, and process π_{WiFi} is interrupted. From table **A**, the robot is advised to switch to task τ_4, which corresponds to *recovery WiFi signal*, so it can choose between continuing the current task without the WiFi signal or switching to task τ_4. To make the switch, it has to discontinue several active processes in τ_{10} (see the Table VII in Appendix A), and it has to activate only process $\pi_{1,C_{11}}$, which is *read access to missing data*, because all others required processes are already active in τ_{10}. However, the constraints of the kind $\pi_i \mathbf{op}_k \pi_j$ in the processes have changed, and they need to be recomputed in the new task τ_4. Therefore, the system has to rate the effort and risks that its choice implies.

Here, we assume that before any decision to switch is taken, the response tasks to switch to has been selected according to the scores given by matrix **A** (see Section 4). We first consider the switching cost, given the best scored task to switch to. The cost to switch from the current to a new task breaks down into a *reconfiguration* and an *interference* cost with respect to a specific task. To these costs we may add a second reconfiguration cost if we need to return to the current task.

Reconfiguration cost: This is the cost to reconfigure the current robot state, which amounts to activating new processes $\pi_{R_1}, \ldots, \pi_{R_m}$. More precisely, the reconfiguration cost calculates the effort the system has to invest to infer all the transition and precondition axioms that are required to make processes $\pi_{R_1}, \ldots, \pi_{R_m}$ executable by the robot at the current state.

Interference cost: This is the cost needed to terminate active processes $\pi_{S_1}, \ldots, \pi_{S_k}$ specifying the current task, which are either not needed or would hinder the newly chosen task.

Let τ_i be the current task under execution, and let τ_j be a chosen task. Let $\Pi_R = \{\pi_{R_1}, \ldots, \pi_{R_m}\}$ be all the processes that need to be activated to operate task τ_j and which are not yet active, and let $\Pi_S = \{\pi_{S_1}, \ldots, \pi_{S_k}\}$ be all the active processes that need to be turned off. Then, according to Equation (21), there is a set of formulas Ψ and a set of formula Φ that must be inferred to explain, respectively, how processes Π_R can be

activated and how processes Π_S can be turned off:

$$\begin{aligned}
&\mathcal{A} \cup \mathcal{C} \cup \Psi \models \exists \mathbf{x}, t \ \textstyle\bigwedge_{i=1}^{m} Poss(start_{\pi_{R_i}}(\mathbf{x}, t), \sigma) \wedge \bigwedge_{i \neq j} \pi_{R_i} \mathbf{op} \pi_{R_j}[\mathbb{s}_i, \mathbb{s}_j], \quad \forall \pi_{R_i} \in \Pi_R \\
&\mathcal{A} \cup \mathcal{C} \cup \Phi \models \exists \mathbf{x}, t \ \textstyle\bigwedge_{i=1}^{k} Poss(end_{\pi_{S_i}}(\mathbf{x}, t), \sigma) \wedge \bigwedge_{i \neq j} \pi_{S_i} \mathbf{op} \pi_{S_j}[\mathbb{s}_i, \mathbb{s}_j], \quad \forall \pi_{S_i} \in \Pi_S.
\end{aligned} \tag{22}$$

Note that Ψ and Φ are not necessarily minimal explanations (see Marquis [1991] and Cialdea and Pirri [1993] for the minimality problem in computing first-order explanations), but we might assume here that these explanations nevertheless minimize the action sequences.

The cost of reconfiguration and interference amounts to computing the cost of these statements. We define the cost considering first the terms of the language; namely, variables and functions of any arity. We exclude from the cost evaluation the terms mentioning variables of sort situation and variables of sort action: For example $do(a, s)$ is excluded, but not $do(A(\mathbf{x}), S_0)$ since S_0 is a constant of sort situation and the action term A is not a variable. Let η be a term of the language; the cost of η is inductively given by the function ν, defined as follows:

$$\nu(\eta) = \begin{cases} 0 & \text{if } \eta \text{ does not mention terms of sort action} \\ 1 & \text{if } \eta = start_{\pi}(\mathbf{x}, t) \text{ or } \eta = end_{\pi}(\mathbf{x}, t) \\ \sum_{i=1}^{k} \nu(A_i) & \text{if } \eta \text{ mentions } A_1, \ldots, A_k \text{ terms of sort action} \end{cases} \tag{23}$$

Here, t is a term of sort time and $\mathbf{x}$ is a list of terms (i.e., variables, constants, functions of any arity) of any sort. Now, to extend the cost computation to formulas, we consider the following restrictions. Let $\varphi \in \mathcal{L}$; we assume that φ either does not mention terms of sort action and situation or, if it does mention them, these are in the form $do([A_1, \ldots, A_m])$, with none of the action terms A_i as variables. Then, let $DNF(\varphi) = \overline{Q}\mathbf{x}[C_1(\mathbf{x}_1) \vee \cdots \vee C_m(\mathbf{x}_m)] \equiv \varphi$ be the equivalent transformation of φ into DNF; namely, into disjunctive normal form in which each clause $C_i(\mathbf{x}_i)$, $1 \leq i \leq m$, is a conjunction of literals, with $\mathbf{x}_i$ terms varying on all sorts. Variables mentioned in $\mathbf{x}_i$ are included in $\mathbf{x}$ and bound by the quantifiers matrix $\overline{Q}\mathbf{x}$. Clearly, any variable x appearing in $\mathbf{x}_i$ can neither be of sort action nor situation. Then the cost of the formula φ is inductively defined as follows:

$$cost(\varphi) = \begin{cases} \nu(\mathbf{x}) & \text{if } \varphi = L(\mathbf{x}) \text{ is a literal} \\ \sum_i cost(L_i(\mathbf{x}_i)) & \text{if } \varphi \equiv \overline{Q}\mathbf{x}\left(L_1(\mathbf{x}_1) \wedge \cdots \wedge L_k(\mathbf{x}_k)\right) \\ min\{cost(C_i(\mathbf{x}_i))\} & \text{if } \varphi \equiv \overline{Q}\mathbf{x}\ \left(C_1(\mathbf{x}_1) \vee \cdots \vee C_m(\mathbf{x}_m)\right) \end{cases} \tag{24}$$

Note that if φ is a tautology, then φ can be reduced to a tautology—namely $\varphi \equiv B \vee \neg B \vee C$, for any clause C. Hence, $min\{cost(\varphi)\} = 0$, since B does not mention terms of sort action. On the other hand, if φ is a contradiction, then $\varphi \equiv B \wedge \neg B \wedge \bigwedge_i \psi_i$ for any conjunction $\bigwedge_i \psi_i$ of literals of $\mathcal{L}$. Hence, $cost(\varphi) = min\{\sum_i cost(L_i)\}$, with $\sum_i cost(L_i)$ tending to infinity since we may take any literal from the language and with any sequence of actions as argument. Finally, given a set of formulas Ψ, the cost of Ψ is the sum of the costs of each $\psi \in \Psi$.

We are now ready to determine the cost for inferring the explanations needed to activate a preferred task τ_j in terms of the cost of the preconditions and constraints for τ_j, at time t, according to Equation (22) and in terms of discontinuing a number of processes occurring in the current task τ_i. Given Equation (22), the cost of inference for switching is:

$$switchCost(\langle \tau_j, \tau_i \rangle, t) = cost(\Psi, t) + cost(\Phi, t) + J\ cost(\Psi', t'), \quad t' > t, q \in \{0, 1\}. \tag{25}$$

	processes	π_{1,C_2}	$\pi_{1,C_{11}}$	$\pi_{1,C_{13}}$
	yields	y_{π_1,C_2}	$y_{\pi_1,C_{11}}$	$y_{\pi_1,C_{13}}$
max scored tasks				
τ_4		.24		
τ_6			.32	
τ_3				.48

Fig. 3. The switching costs for the task τ_{12}, *data retrieval* (see Table VII). The up-rows indicate the processes the task is assembled from and their yields. In the left column are the tasks that obtained the maximum switching score in score matrix **A**. The gray cells indicate the average normalized switching cost to the preferred task, indicated in the left column.

Here, Ψ' is like Ψ, but it is the cost of returning to the current task after the task the robot has switched to is completed. If $J = 0$, there is no need to return to the current task τ_i.

Having obtained the switching cost and knowing the response task for the triggered stimulus, we can compute the payoff. Here, the payoff is simply the culmination of the stimulus-response problem and it is, indeed, the payoff for switching. Let R_S be a function specifying the choice of switching and R_I the analogous function for inhibition. Then the payoff for switching is simply $R_S - R_I$.

Obviously, we expect to attribute the switching cost to R_S. However, if the switching cost is computed before any decision is taken, then this is like attributing the cost to both R_S and R_I. In fact, by its definition, computing the switching cost is the same as abducing the set of explanations needed to activate the new task and deactivating the current state; thus, it follows that this inference process ends up weighting the decision well before the decision is taken.[2]

Therefore, to both correctly define the two functions and to avoid burdening the decision, the switching cost can be replaced by a prediction of it. To predict the switching cost, as defined in Equation (25), two steps are needed:

(1) Define a trend of the switching cost according to time, in terms of the sequence of actions entering the σs of the two logical implications in Equation (22).
(2) Produce an estimate of the switching cost for different time durations of the current task.

To this end, we generate a number of action sequences for as many time lapses within a time range bounded by 120 minutes. For each of these samples, the cost given in Equation (25) is computed. An example of the averages of the normalized switching cost for task τ_{12}—namely, *data retrieval*—is given in Figure 3 with respect to each of the response tasks to the stimuli that can be triggered by each of the processes composing the execution of task τ_{12}.

The trends of the switching cost for a task are established with respect to each of its processes and for each of the response tasks to the stimuli of its process yields; therefore, it is defined for each pair $\langle \tau_i, \tau_j \rangle$. Once the trends are obtained, the switching cost is approximated by its frequencies with respect to time.

To estimate the cost distribution, we define a bivariate random variable $\mathbf{X}$ taking values in the domain $Times \times SwitchingCost$ for each pair $\langle \tau_i, \tau_j \rangle$. The realization of the bivariate variable $\mathbf{X}$ is computed by a nonparametric density f_X measuring the switching cost frequencies to discontinue the current task τ_i and activate task τ_j. Here, we use a Gaussian kernel to estimate kernal density, as we did in Section 3; here, however, we have a bivariate estimator (time and cost), so we define the kernel density

[2]Computing the explanations for switching can be viewed as the intellectual work that the system performs, as opposed to the practical and physical work of executing tasks.

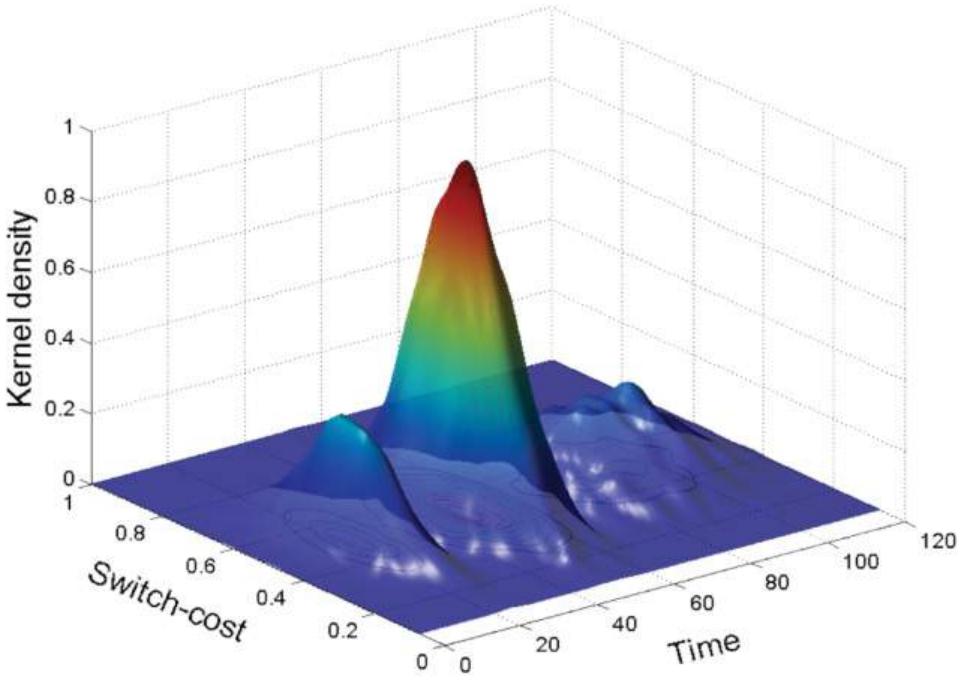


Fig. 4. Kernel density f_X for switching cost from the task τ_{12} to task τ_4, showing frequencies of switching cost in a time lapse of 120 minutes, taken with respect to 10 cost trends.

for a generic task as follows:

$$f_X(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{det(\mathbf{H})}K(diag(\mathbf{H}^{-1})^{\top}(\mathbf{x}-\mathbf{X}_i)). \tag{26}$$

Here, $K(\upsilon) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}\upsilon^2)$ and $\mathbf{H} = n^{-1/6}\Sigma^{1/2}$, where Σ is the diagonal covariance matrix of the samples; namely, $\mathbf{H}$ follows the Scott's rule of thumb, and $diag(\cdot)$ indicates the diagonal. The grid has resolution $\mathbf{H}$, and the bivariate $\mathbf{X}_i$ is built using samples from the switch cost trend taken at time t for different action sequences. The switching cost density for tasks (τ_{12}, τ_4) is illustrated in Figure 4, where time is presented in minutes. To compute the modes of kernel density, we proceed as in Section 3. Let $\upsilon_i = diag(\mathbf{H}^{-1})^{\top}(\mathbf{x}-\mathbf{X}_i)$ and $diag(\mathbf{H}) = (h_1, h_2)^{\top}$; then the gradient of f_X is:

$$\begin{aligned}\nabla f_X(x_1,x_2) \;=\; &\left\{\frac{1}{n}\sum_{i=1}^{n}-\frac{1}{h_j det(\mathbf{H})}K(\upsilon_i)\upsilon_i\right\}_{j=1,2} = \left\{-f_X(x_1,x_2)\left(\frac{x_1}{h_j^2}+\frac{x_2}{det(\mathbf{H})}\right)\right.\\ &\left.+\frac{1}{n}\sum_{i=1}^{n}\frac{1}{det(\mathbf{H})}K(u_i)\left(\frac{X_{i1}}{h_j^2}+\frac{X_{i2}}{det(\mathbf{H})}\right)\right\}_{j=1,2}\end{aligned} \tag{27}$$

As in Section 3, the modes $\widehat{\mathbf{x}}$ are collected by gradient ascent. The modes return the best prediction of the bivariate $\mathbf{x}$. Where the probability of $\mathbf{x}$ is below a threshold ϵ or where the probability is high but the estimated switching cost (given time t) is beyond the effort the robot system can afford, then we introduce a time failure. These failures, that can be collected for each pair $\langle\tau_i, \tau_j\rangle$, are best explained with an example.

Example 6.2. Continuing Example 6.1, consider the pair (τ_{10}, τ_4) at time $t = 72$. According to the cost trends, the nonparametric function $f_X(\mathbf{x})$ is drawn, and it has modes at $T = (35, 71, 110, 119)$. At these times, the cost rates are $(0.3, 0.5, 0.7, 0.9)$. Let $\epsilon = 0.6$ be the maximum cost the robot can afford to switch. Since $t = 72$, it follows

that the cost can be afforded. If t were 110, then the rate would be $0.7 > \epsilon$; hence, we would obtain a failure in the cost computation.

We collect the time to process failure by establishing the switching cost between $\langle \tau_i, \tau_j \rangle$ from the modes $\widehat{\mathbf{x}}$ of $f_X(\mathbf{x})$ as in the following set, where sc is $switching\_cost$, and ϵ is the threshold for the effort the robot can afford:

$$\{t_i \mid (t_i, sc_i) \in \widehat{\mathbf{x}},\ f_X(\widehat{\mathbf{x}}) < 10^{-2} \vee sc_i > \epsilon\}. \tag{28}$$

Time to failure of the switching cost amounts to the belief that, at the specific time, either the effort of computing the state update is not affordable, or the switching cost value cannot be predicted. This time to failure is independent of any previous decision. Therefore, its best approximation is the exponential distribution with a constant hazard function; that is, it has no burn-in cost. Let U_{sc} be a random variable with exponential distribution, with failure rate[3] $\alpha = 1/\theta'$:

$$\begin{array}{ll} F_{U_{sc}}(z) = & 1 - \exp(-\alpha z), \qquad CDF \\ f_{U_{sc}}(z) = & \alpha \exp -(\alpha z), \qquad PDF. \end{array} \tag{29}$$

The reliability for time to failure of the process establishing the switching cost is $R_{U_{sc}}(t) = 1 - F_{U_{sc}}(t) = \exp(-\alpha t)$.

At this point, we consider the time to failure of the current tasks τ_i and the chosen task τ_j. To estimate a task's time to failure, we introduce a random variable W_τ whose probability distribution indicates beliefs about the likelihood of failures time. The random variable W_τ is characterized by a Weibull distribution with parameters (β, θ), with θ the average time to failure and β the failure rate[4] (decreasing if $0 < \beta \leq 1$ and increasing for $1 < \beta < 3.5$):

$$\begin{array}{l} F_{W_\tau}(z; \beta, \theta) = 1 - \exp\left(-\left(\frac{z}{\theta}\right)^\beta\right), \qquad CDF \\ f_{W_\tau}(z; \beta, \theta) = \dfrac{\beta \exp\left(-\left(\frac{z}{\theta}\right)^\beta\right)\left(\frac{z}{\theta}\right)^{\beta-1}}{\theta}, \qquad PDF. \end{array} \tag{30}$$

Here, β is increasing, $1 < \beta \leq 3.5$, and θ ranges between 0 and 120 minutes. Then, the reliability of task τ_i is $R_{W_{\tau_i}}(t) = P(W_{\tau_i} > t) = \exp(-(\frac{t}{\theta})^\beta)$ at t; namely, the probability that the task will continue without failures for time $t < W_{\tau_i} \leq t_{end}$, needed to wrap up. For the reliability of the task sequence $\langle \tau_i, \tau_j \rangle$, we consider the reliability function $R_{\langle W_{\tau_i}, W_{\tau_j} \rangle}(t)$:

$$\begin{array}{rcl} R_{\langle W_{\tau_i}, W_{\tau_j} \rangle}(t) & = & P(W_{\tau_j} > t) P(W_{\tau_i} > t) = (1 - F_{W_{\tau_i}}(t; \beta_1, \theta_1))(1 - F_{W_{\tau_j}}(t; \beta_2, \theta_2)) \\ & = & \exp\left(-\sum_{i \in \{1,2\}} \left(\frac{t}{\theta_i}\right)^{\beta_i}\right). \end{array} \tag{31}$$

This expression can be extended to three tasks in sequence.

[3]Here, α indicates the failure rate and differs from the denotation of confidence level given in Section 3.

[4]Here, β denotes the rate parameter of the Weibull distribution and differs from the parameters used in Section 3. Note that if $\beta = 1$, then the Weibull reduces to the exponential distribution.

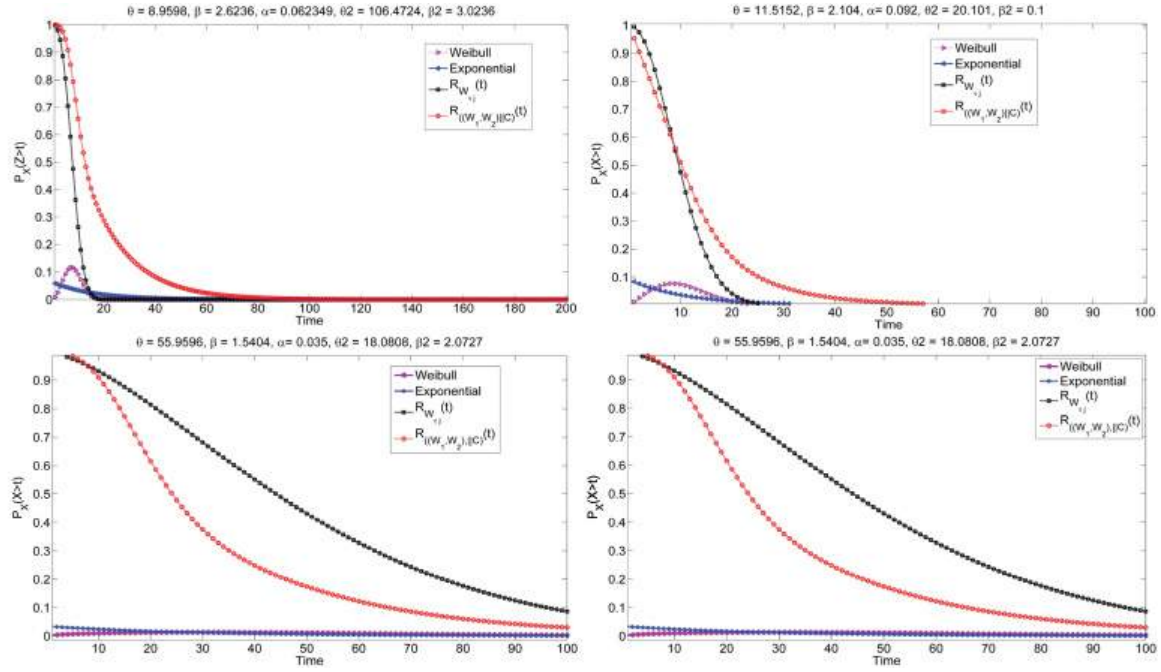


Fig. 5. Examples of reliability of the current task (the dashed black line) against the reliability of the parallel composition of the switching cost reliability with the sequence formed by the current task τ_i and the chosen task τ_j.

Finally, the reliability of the combination of the two tasks, taken in sequence, and the switching cost estimation, taken in parallel with the two tasks, is defined as follows:

$$\begin{aligned} R_{(\langle W_{\tau_i}, W_{\tau_j}\rangle || U_{sc})}(t) &= P(W_{\tau_j}>t)P(W_{\tau_i}>t) + P(U_{sc} > t) - P(W_{\tau_j}>t)P(W_{\tau_i}>t)P(U_{sc}>t) \\ &= \exp(-t\alpha) + \exp\left(-\sum_{i\in\{1,2\}}\left(\frac{t}{\theta_i}\right)^{\beta_i}\right) - \exp\left(-t\alpha - \sum_{i\in\{1,2\}}\left(\frac{t}{\theta_i}\right)^{\beta_i}\right). \end{aligned} \tag{32}$$

At this point, we can see that $R_{(\langle W_{\tau_i}, W_{\tau_j}\rangle || U_{sc})}(t)$ and $R_{W_{\tau_i}}(t)$ are the two sought for functions R_S and R_I introduced earlier, and, therefore, the payoff for switching is:

$$\text{payoff} = R_{(\langle W_{\tau_i}, W_{\tau_j}\rangle || U_{sc})}(t) - R_{W_{\tau_i}}(t). \tag{33}$$

Clearly, if at the given time t, $R_{W_{\tau_i}}(t) < R_{(\langle W_{\tau_i}, W_{\tau_j}\rangle || U_{sc})}(t)$, then the payoff is positive and switching is convenient; otherwise it is negative or null. Some examples showing reliability are illustrated in Figure 5. The simulation-based computation for task time to failures is reported in the Appendix A.7, and experiments computing the payoff for switching are reported in Section 7.3.

7. EXPERIMENTS

In this section, we present the results of experiments undertaken to evaluate the proposed framework's performance. In Section 7.1 , the performance of the stimulus model is tested. In Section 7.2, we evaluate the generalizability of the score matrix **A**, described in Section 4. In Section 7.3, we test the switching decision described in Section 6. The whole framework is tested in Section 7.4. Finally, in Section 7.5, we compare the performance of the stimulus-response framework with two different approaches to robot replanning [Beetz et al. 2010; Talamadupula et al. 2010; Göbelbecker et al. 2011]. For all experiments, we recruited 20 operators between 26 and 32 years of age, in good

(a) (b) (c)

Fig. 6. (a) Simulated disaster area. (b) Obstacles located to the disaster area. (c) Operator interacting with the robot through the task switching interface.

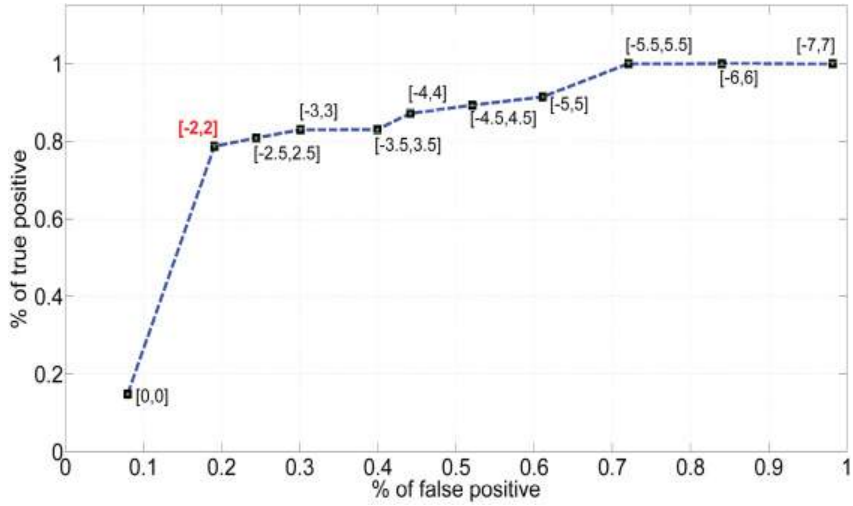


Fig. 7. Performance of the stimulus model: ROC curve of the true and false positives as the time range, indicated in brackets, changes.

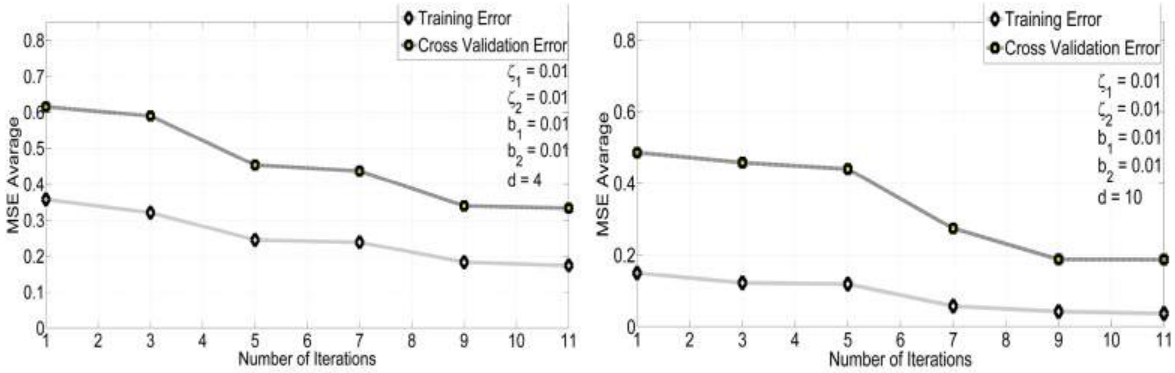
physical conditions. Each operator performed up to 10 trials using the task switching interface (see Figure 6(c)) and choosing a task from the task library described in Section A.3, in Appendix A.

7.1. Performance of the Stimulus Model

The performance of the stimulus model is evaluated by the ROC curve of both the true and false stimuli over a varying time range, with the stimulus occurring in the range. The performance of the model is measured on a testset, collected as follows. Once a task is chosen, the operators manually label the yields of the running processes upon the occurrence of a stimulus. The yields, labeled by the operators, were compared with the results of the stimuli classification provided by the model (Section 3). Both correct classifications and mismatches are noted, considering the reference time interval $[init, end]$ centered at the time the operator notified occurrences of stimuli. Figure 7 shows the ROC curve of the stimulus model with respect to the size of Δt. The trend reports the percentage of true positives versus false positives for every considered time range. Then, $[-2, 2]$ is taken as the reference time interval to evaluate the correctness of the stimulus model, due to the tradeoff between the false and true positives.

7.2. Generalizability of the Score Matrix A

The generalizability of score matrix $\mathbf{A}$ is measured according to the convergence of the Mean Square Error (MSE) with respect to two different values of rank d of the matrices $\mathbf{Q}$ and $\mathbf{P}$, factorizing $\mathbf{A}$ (Section 4). The dataset for both stimuli and responses



(a) $d = 4$ (b) $d = 10$

Fig. 8. Learning curve for the score matrix **A** given two different rank values of the matrices **Q** and **P**, factorizing **A**: (a) $d = 4$ and (b) $d = 10$. The curve is derived with a k-fold cross-validation technique.

to them is gathered by operators in the field as they execute tasks chosen from the task library via the task switching interface set to active mode (see Section A.5 in Appendix A). The operator responds to a stimulus by either switching or continuing–inhibiting the stimulus. A k-fold cross-validation technique on the gathered dataset is used to assess the generalizability of the model. Figures 8(a) and 8(b) show the average of both the MSE training and cross-validation error under two different values ($d = 4$ and $d = 10$, respectively) specifying the rank of the matrices **Q** and **P**. Indeed, $d = 10$ is the reference value of the rank. The completed score matrix, resulting from the cross-validation process, is given in Tables XI and XII in Appendix A. The MSE error with respect to the training set is shown in Figure 2.

7.3. Reliability Tests and Payoff

This experiment computes the payoff of switching, taking into account both the time to failure for each task and the time to failure for the switching cost, as described in Section 6 (see Table XIII in Appendix A). Table III lists both the values of the reliability $R_{W_{\tau_i}}(t)$ of the tasks that the robot was executing before the occurrence of the stimulus and the reliability $R_{\langle\langle W_{\tau_j}, W_{\tau_i}\rangle||U_{sc}\rangle}(t)$ of the combination of the current tasks with the tasks to switch to. The indices of the tasks in Table III are made explicit in Table VII in Appendix A. The choice of switching, given the payoff, is indicated in the last column. These data are used in Section 7.4 to evaluate the whole stimulus-response framework.

7.4. Experimental Evaluation for Task Switching

In this experiment, we compare system decisions with operator decisions. The set of robot tasks are variations of task τ_{10}; namely, approaching a number of objects located in the experiment area and already reported in the 3D map. A number of stimuli are purposefully triggered: battery exhaustion, WiFi disconnection, write access to full memory, and read access to missing data. During execution, robot choices along with payoff values and the log for reliability computation are recorded (see Section 6). In parallel, the responses selected by the operator via the task-switching interface are recorded. Note that the operator does not have to choose the preferred task from the score matrix **A**, as the robot will do. Furthermore, the operator does not see the robot's choices, to ensure unbiased decisions. The gathered data were used to build the sequences of task choices for both the robot and the human operators. In Figure 9, we report one run, among all those collected, showing operator and robot choices in parallel. Comparing these sequences, we note that in more than 90% of the cases the model of task switching correctly induces the robot to shift to the same task as that

Table III. Reliability Examples Used in Experiment 7.4

Time	Current Task	Candidate Task	$R_{W_{\tau_i}}(t)$	$R_{\langle\langle W_{\tau_i}, W_{\tau_j}\rangle \| U_{sc}\rangle}(t)$	Task Switching
11	τ_{10}	τ_4	.3065	.4150	Yes
22	τ_{10}	τ_6	.3375	.4387	Yes
34	τ_{10}	τ_{12}	.4571	.4705	Yes
45	τ_{10}	τ_5	.3917	.4955	Yes
70	τ_{10}	τ_4	.2736	.3272	Yes
76	τ_4	τ_{12}	.3018	.6043	Yes
89	τ_{10}	τ_6	.4181	.5361	Yes
103	τ_{10}	τ_{12}	.3894	.2840	Yes
124	τ_{10}	τ_5	.5083	.6735	Yes
134	τ_5	τ_4	.5310	.5649	Yes
158	τ_{10}	τ_{12}	.2739	.3770	Yes
164	τ_{12}	τ_4	.3366	.5213	Yes
179	τ_6	τ_4	.3861	.6015	Yes
202	τ_{10}	τ_5	.2991	.3127	Yes
217	τ_5	τ_4	.3305	.3820	Yes
246	τ_{10}	τ_{12}	.5937	.6436	Yes
273	τ_{10}	τ_4	.4031	.4036	Yes
302	τ_{10}	τ_6	.3593	.5912	Yes
321	τ_{10}	τ_5	.2859	.3683	Yes
344	τ_{10}	τ_{12}	.2039	.3091	Yes
366	τ_{10}	τ_4	.3709	.5905	Yes
395	τ_{10}	τ_5	.1901	.2197	Yes

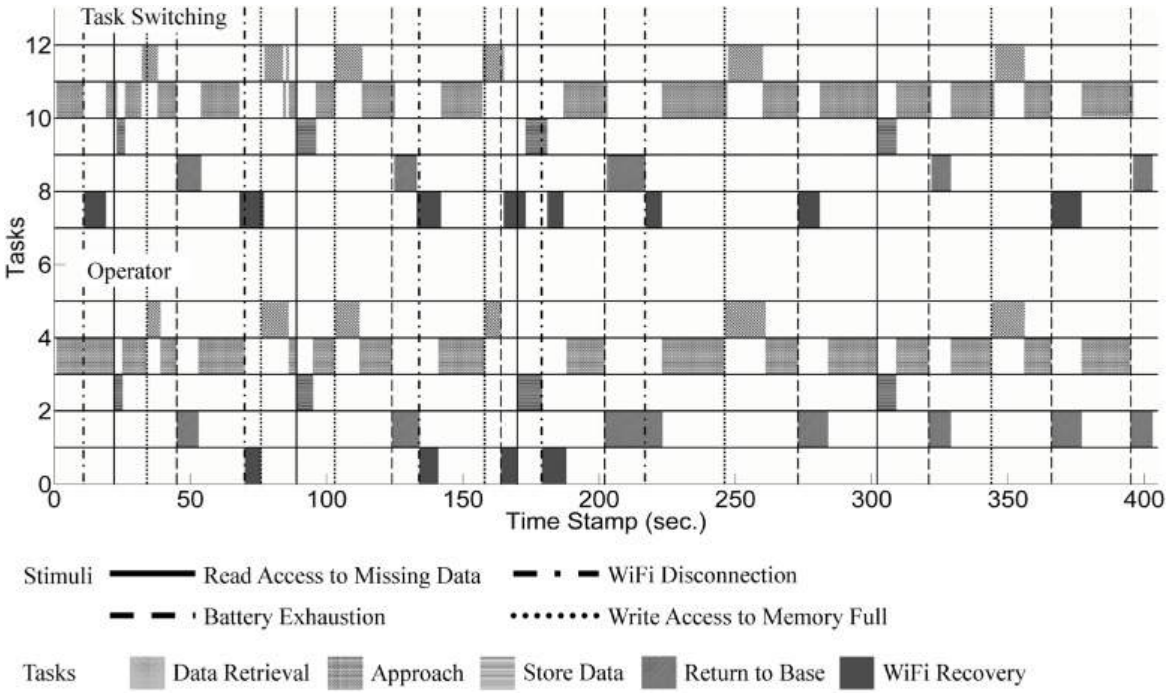


Fig. 9. Sequences of task choices according to both the robot and the human operator, performed during the mission of approaching selected objects positioned in known places in the experiment area. Vertical lines denote the occurrences of the stimuli at different times. In more than 90% of the cases, task switching correctly induces the robot to shift to the same task as that selected by the operator. In more than 41.1% of cases, the robot anticipates the operator in shifting. At time $t = 11$ *sec*., as well as at time $t = 217$ *sec*., the operator inhibits the stimuli while the robot switches to a new task.

selected by the operator. Among the percentage of correct robot switches, 41.1% happen in advance. At time $t = 11$ *sec*., as well as at time $t = 217$ *sec*., the operator inhibits the WiFi disconnection stimulus, while the model induces the robot to switch to the signal task recovery.

7.5. Comparison with Robot Replanning

For this experiment, we considered two approaches to robot replanning. The first [Beetz et al. 2010] is based on the implementation of execution-monitoring (CRAM-EM) in the CRAM plan language. In this implementation, stimuli are explicitly modeled either as *intrinsicStateChangeEvents* or as *physicalEvents*. Each of these events is associated with a recovery procedure. The execution-monitoring determines whether the occurrence of an event can affect the successful outcome of the execution of the current plan. In such a case, the monitor generates a new plan on the basis of the set of predefined failure recovery procedures whose execution is intended to restore the state of the task at hand. The second case [Talamadupula et al. 2010; Göbelbecker et al. 2011] is based on the implementation of a deterministic action goal-oriented POMDP (DAGO-POMDP) in Decision Theoretic PDDL, in which all actions have no-zero cost, and an optimal policy can be formulated as a finite horizon contingent plan. Stimuli are modeled as perceptual propositions (i.e., *percepts)*. Tasks are associated with stimuli by suitably assigning either high-value *rewards* or low-value *costs* to the actions needed to execute the tasks. In this implementation, plans are possibly rebuilt online to react to changes in the underlying domain (e.g., when goals are modified or when stimuli occur). Details about the implementation of both approaches are out of the scope of this work. The experiments were performed in a real partially destroyed area (see Figure 6(a)), where robot can overcome some of the obstacles (see Figure 6(b)).

The performance of the model of task switching with respect to both CRAM-EM and DAGO-POMDP replanning approaches was evaluated by measuring the following indexes: (1) the time average of reaction to stimuli (MSRT); (2) the time average of response to stimuli (MRST); the average time of mission accomplishment (MMA); (4) the ratio between the number of switches performed by the task switching model and the number of replanning instances, where the response to stimuli provided by all the models is the same (SWRPC); (5) the ratio between the number of switches and the number of replanning instances, where responses are different from each other (SWRPD); and, finally, (6) the ratio between the number of times the task switching model inhibited the stimulus and the number of times both CRAM-EM and DAGO-POMDP decided to replan (IRP).

MSRT measures the average time elapsed between the occurrence of a stimulus and its identification. For CRAM-EM, this time corresponds to the average time elapsed between the occurrence of the stimulus and the detection of the corresponding event. For DAGO-POMDP, this is the average time elapsed between the occurrence of the stimulus and the decision to rebuild the plan in reaction to it. MRST represents, on average, the time needed to select the best response to the stimulus (e.g., shifting or continuing). For both replanning approaches, this index denotes the average time spent for replanning. To measure these indexes, the experiments were set up as follows. The robot is instructed to autonomously approach an object whose position is known on the given 3D map. The time estimated to accomplish the task is about 20 *min* (this is a large area). The task ends when either the estimated time expires or the robot reaches all the targets. The occurrence of a stimulus on WiFi signal quality and on battery power level is simulated after 6 *min.* and after 12 *min.* from the beginning of the task, respectively. The signal associated with a stimulus is kept up for 2 *min.* The pulse of the signal has a trapezoidal profile. The choice of a reduced number of stimuli is motivated by the fact that we want to bound the domain of both different approaches

Table IV. Time Comparison with Robot Replanning

Model	MSRT (sec.)	MRST (sec.)	MMA (sec.)
Stimulus-Response	51.49	88.17	837.86
CRAM-EM	70.94	159.03	1093.72
DAGO-POMDP	69.22	131.07	961.91

Table V. Performance Comparison with Robot Replanning

Model	SWRPC	SWRPD	IRP
CRAM-EM	.45	.30	.25
DAGO-POMDP	.60	.15	.25

to replanning. This guarantees the applicability of the two approaches to replanning. Both the duration and the profile of the pulse associated with a stimulus have been chosen to test the degree of proactivity of the stimulus-response, but also to allow both CRAM-EM and DAGO-POMDP to infer plans that avoid stuck conditions for the robot. On the other hand, the static setting of the environment allows us to make repeatable trials. In this setting, 20 trials were performed.

Table IV reports the results obtained from measuring the time indexes MSRT, MRST, and MMA. The MSRT index of the task switching model is lower than the indexes of the other models. The main reason is that the stimulus model anticipates the identification of the stimuli (see Section 3). Instead, both CRAM-EM and DAGO-POMDP require that the amplitude of the signal associated with a stimulus reaches a certain threshold in order to detect the stimulus. Both CRAM-EM and DAGO-POMDP perform a state space search to generate new plans in the replanning phase, whereas the task switching model use the payoff model to select the best choice for a new task (see Section 6), so no state space search is required. Therefore, the MRST time index of the proposed model is the lowest.

The task switching model demonstrated better performance in terms of time needed to accomplish the mission. In fact, the model can compromise between brave and conservative behavior, which is not allowed in a purely reactive framework. The proposed model is able to inhibit the stimulus, and thus balance continuous changes by preserving focus on the task at hand if switching is too demanding. This behavior outperforms the purely reactive behavior of both CRAM-EM and DAGO-POMDP. The SWRPC, SWRPD, and IRP indexes reported in Table V validate these considerations, also showing that the robot not only behaves more rationally but also more rapidly in accomplishing a mission.

In addition, the indexes MSRT, MRST, and MMA highlight that the system is more flexible than both CRAM-EM and DAGO-POMDP. The first column of Table V shows that the robot chooses to switch only in the 45% and 60% of cases in which the other two models decided to replan. On the other hand, the values of the indexes in the second column of Table V show that the task selected, when switching is the choice, are different; this is due to the lack of a stimulus-response matrix learned from humans in the other two models. Therefore, replanning in both CRAM-EM and DAGO-POMDP is wired into the behavior and not learned. Finally, the values of the IRP index in the third column of Table V show the ability of the task switching model to be conservative with respect to the task at hand.

These considerations prove that the task switching model, by exploiting both high-level reasoning and learning from operators, effectively harmonizes robot choices with context requests by conjugating a rational behavior with a flexible one.

8. CONCLUSION

In this work, we proposed a model for robot stimuli and stimulus-response that can significantly improve robot control in general, and particularly in the robot's interaction with the environment. The proposed approach indicates a new way to address robot cognitive control by showing that interaction with the environment can be proactive if stimuli are taken into account not individually but as a whole system whose specific aspects can be learned by interacting with human users. This gained flexibility in robot control can increase the usability of the robot, especially for operators who have to deal with it in extreme environments like first-responder training areas or even post-disaster environments.

Our approach is completely novel, and, to our knowledge, no other approach to robot cognitive control has developed these ideas. The proposed method is supported by a full implementation in an advanced robot platform endowed with multiple abilities to perform several tasks, in which the interplay with operators plays a crucial role for both learning and evaluating performance. Our approach is inspired by the wide psychophysical and psychological literature on human patterns in stimulus-response and task-switching, although we have provided an integrated model that is tailored for a robot system and is replicable for other robotic platforms.

Only recently has the need to model the stimulus-response pattern leading to task switching and the cognitive functions of shifting and inhibition become a hot topic in cognitive robotics [Medina Ayala et al. 2012; Nakano et al. 2011; Ganesh and Burdet 2013; Ajoudani et al. 2013; Ding and Fang 2013; Tao et al. 2012; Li et al. 2013; Sung et al. 2013; Jamone et al. 2013; Saglam and Byl 2013; and Li and Cheah 2012]. The earliest studies were carried out within brain-actuated interaction [Milln et al. 2004], mechatronics [Capi 2007], behavior learning [Ito et al. 2006], navigation [Althaus and Christensen 2003], and planning [Finzi and Pirri 2005]. Recently, several studies posited the need to model task switching to cope with adaptivity and ecological behaviors in a dynamic environment [Capi et al. 2008; Suzuki et al. 2009; Wawerla and Vaughan 2009; Durkee et al. 2011; D'Ambrosio et al. 2011].

In building the stimulus model, the main difficulties we encountered were due to the experimental setting and data collection from several operators. This led us to provide an outliers rejection method specifically for the stimuli; this method supports the selection of the training set on the basis of the features manifold eliciting the stimuli. This turned out to be a good solution to transfer training data acquired by human operators to the robot training set. To learn the stimulus-response selection, we still used several operators, but to infer unknown data we use low-rank matrix factorization methods [Zhang et al. 2012; Chen 2008; Buchanan and Fitzgibbon 2005; Canny 2002] widely employed in recommendation systems and collaborative filtering [Ko and Lee 2002; L. Herlocker et al. 2004]). This model has proved suitable to estimate the latent parameters underlying stimulus-response mapping, as well as to predict the weights given by each stimulus to each response, which are missing in the observed data.

The problem of shifting and inhibition induced by stimuli was modeled by a switching cost, obtained by conjugating a cost to reconfigure the robot state for the new goal and the cost to resolve interference with the current activities set. Finally, the decision problem culminates in the definition of the payoff for switching, which requires us to consider the risk and effort in computing the cost. We used time to failure to combine the whole effort and predict the reliability of all the decision components to calculate the payoff for switching.

We also introduced a connection between stimuli and reasoning level by defining a boundary shared by the two levels via the mapping $\mathcal{E}$ that plays the role of transduction, lifting the terms from the sample space of features, where yields are sampled, to the

terms of the language. Thus, logical inference provides the grounds for the causal and constraint relations among processes carrying the information driven by stimuli. The logical structure is also the framework within which both processes and the robot's mental states can be defined, and it is where the reconfiguration and interference costs can be fully interpreted.

The proposed method improves both autonomy and flexibility in robot control. Furthermore, according to the experiments done to verify the coherence and rationality of robot behaviors with respect to operator behavior while teleoperating the robot, the approach proves to be very promising for improving the proactive interactions with the environment required by a robot relating to people and their needs.

Comparing the proposed framework with two other approaches to robot replanning, both robot flexibility and performance, in terms of time and tasks choices, are improved. It follows that robot usability also is improved since user stress can be reduced by limiting the workload that continuous monitoring of robot activities demands of the operator.

A. APPENDIX: IMPLEMENTATION

A.1. Hardware Description

The robotic platform is designed for harsh, unmanned environments. These are large-scale environments where robot missions last for long time periods and several events and stimuli burden the robot's exploration and reporting activities.

The robot platform ©BlueBotics, named Absolem, has a central body where the electronics is located and has two bogies on the sides. Each bogie is made of a central track for locomotion and two active flippers on both ends to extend the climbing capabilities. A breakable passive differential system allows the rotation of the bogies around the body. Several sensors are installed on the platform, among them a rotating 2D laser scanner, an omnidirectional camera with a 360-degree field of view, an IMU/GPS, and a panoramic microphone.

A.2. Low-Level Robot Functionalities

The robot's Simultaneous Localization and Mapping (SLAM) is based on a real-time 3D ICP [Pomerleau et al. 2013]. A Dead Reckoning Navigation System (DRNS), based on a Complementary Filter (CF), estimates the Euler angles of the robot body from the fusion of both odometry and the IMU inertial data [Cao et al. 2009; Kubelka and Reinstein 2012]. Both ICP-based SLAM and DRNS provide, at real-time, the pose of the robot as a feedback to the Trajectory Tracking Controller (TTC) [Gianni et al. 2013b]. Static traversability cost assessment of the ground, from point cloud data, is performed by both physics-based terrain [Stoyanov et al. 2010; Papadakis and Pirri 2012] and physical constraints analysis [Cafaro et al. 2013]. Visual perception of the environment is limited to object (e.g., signs, cars, and persons) detection accomplished with an omnidirectional camera [Hurych et al. 2011].

A.3. Robot Model

The robot model provides a representation of both the hardware and software components and of the processes activated by each component. For example, the SLAM software module is modeled as a component, whereas the mapping and localization functionalities are represented as processes managed by the SLAM component. Table VI lists the main components and processes of the robot system model. Here, by

Table VI. Components and Processes of the Robot Model

Component Id	Component Description	Process Id	Process Description
C_1	Battery	π_{1,C_1}	Power Level Monitoring
C_2	WiFi	π_{1,C_2}	Connection Management
C_3	3D Laser Scan	π_{1,C_3}	ROS Drivers Running
		π_{2,C_3}	Laser Rotating
		π_{3,C_3}	Scanning
C_4	SLAM	π_{1,C_4}	ROS Packages Running
		π_{2,C_4}	2D Mapping
		π_{3,C_4}	3D Mapping
		π_{4,C_4}	Localizing
C_5	Vision System	π_{1,C_5}	ROS Drivers Running
		π_{2,C_5}	Calibrating
		π_{3,C_5}	Streaming
C_6	Visual Perception	π_{1,C_6}	ROS Packages Running
		π_{2,C_6}	Detecting Object/Person
		π_{3,C_6}	Recognizing Object/Person
C_7	Robot Engine	π_{1,C_7}	ROS Drivers Running
		π_{2,C_7}	Locomotion System Monitoring
		π_{3,C_7}	Temperature Monitoring
		π_{4,C_7}	Locking Differential Drive
C_8	Trajectory Control	π_{1,C_8}	ROS Packages Running
		π_{2,C_8}	Planning
		π_{3,C_8}	Tracking
C_9	Dialogue	π_{1,C_9}	Human to Robot Exchanging msg
		π_{2,C_9}	Robot to Human Exchanging msg
C_{10}	Microphone	$\pi_{1,C_{10}}$	ROS Drivers Running
		$\pi_{2,C_{10}}$	Hearing
		$\pi_{3,C_{10}}$	Speech Recognizing
C_{11}	Memory	$\pi_{1,C_{11}}$	Data Reading
		$\pi_{2,C_{11}}$	Data Writing
		$\pi_{3,C_{11}}$	Memory Checking
C_{12}	IMU	$\pi_{1,C_{12}}$	ROS Drivers Running
		$\pi_{2,C_{12}}$	Robot Pose Estimating
C_{13}	ROS Master Core	$\pi_{1,C_{13}}$	ROS Core Running
C_{14}	Central Processing Unit	$\pi_{1,C_{14}}$	Payload Monitoring
		$\pi_{2,C_{14}}$	Temperature Monitoring
C_{15}	Terrain Analysis	$\pi_{1,C_{15}}$	ROS Packages Running
		$\pi_{2,C_{15}}$	Traversability Analysis
		$\pi_{3,C_{15}}$	Ground Subtracting & Analysis
		$\pi_{4,C_{15}}$	Obstacle Detecting
		$\pi_{5,C_{15}}$	Adapting Robot Morphology
C_{16}	Diagnostic	$\pi_{1,C_{16}}$	System Check
		$\pi_{2,C_{16}}$	Repair

C_k we denote the k-th robot component and by π_{j,C_k} the j-th process managed by the k-th component.

A set of robot processes defines a task. This set of tasks forms the robot task library. For example, the task to overcome an obstacle requires that all the processes managed by the 3D Laser Scan component are active, that the 3D mapping process is running, and that the process adapting robot morphology to the terrain provides the correct

positions for the flippers. Table VII lists the set of tasks designed in agreement with the available robot functionalities. Here, by τ_i we denote the i-th task in the task library. The right column in Table VII lists all the processes that need to be activated for the robot to perform the corresponding task.

Each process yields a quantum of information with characteristic features. This quantum of information is called the *yield* of the process. For example, the process monitoring the temperature of the robot engine outputs, over time, the servo-motors temperature. The information carried by the processes regulates the robot's behaviors. For example, the engine temperature could reach a *critical* level and damage the robot servo-motors. The critical engine temperature level, as well as the loss of the WiFi connection between the robot and the remote command post, are stimuli, and they are identified from the process yields. Table VIII lists a description for each robot stimuli. Note that each process can trigger a single stimulus; therefore, the number and types of stimuli are determined by the number and types of processes and these, in turn, are determined by the number and types of robot components. In the table, the term $y_{\pi_{j,C_k}}$ denotes the stimulus triggered by the yield of the j-th process managed by the k-th component C_k.

A.4. Robot Control

Robot control is defined by a declarative temporal model of its activities and a planning engine. The declarative temporal model is specified in the Temporal Flexible Situation Calculus (TFSC) [Finzi and Pirri 2005; Carbone et al. 2008] so that the main components and processes of the robot system reported in Table VI are represented by cause-effect relationships as well as by the temporal and spatial constraints among the processes [Pirri and Reiter 2000; Finzi and Pirri 2005], as discussed in Section 5. Planning is composed of two main logical modules: the plan generator and the execution monitoring. The plan generator relies on a library of Prolog scripts that designates the set of tasks (see Table VII) that the robot can perform according to the specified processes, their temporal constraints (compatibilities), and their preconditions. Execution-monitoring is a continuous process ensuring that both 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. The execution monitoring embeds the model of task switching to regulate the behavior of the robot in responses to the incoming stimuli. Both the TFSC model and the planning engine are implemented in Eclipse Prolog [Apt 2006], which optimally combines the power of a constraint solver (for time and compatibility constraints) with inference in order to generate the set of action sequences and also to enable continuous updates from incoming new knowledge by using finite regression [Pirri and Reiter 2000].

The distributed working memory structure interconnects the TFSC model and the planning engine with the task switching engine and with the robot functionalities that are integrated in the ROS Operating System (ROS) [Quigley et al. 2009] (see Figure 10). The structure is distributed over a collection of ROS services, organized as a set of addressable memory registers. In addition to the usual reading and writing operations, the structure provides additional operations like memory dumping and update acknowledgment operations, suitable to check both consistency and persistence of the state of the memory. The working memory system also takes care of mapping the internal state of the robot into a representation suitable for planning and reasoning about tasks.

The overall robot control provides bidirectional communication interfaces with the human user. It further allows the human operator to switch between several

Table VII. Task States

Task Id	Task Description	Processes
τ_1	Robot System Recovery	$\{\pi_{1,C_16}, \pi_{2,C_16}\}$
τ_2	Request for Path to the Operator	$\{\pi_{1,C_2}, \pi_{1,C_4}, \pi_{3,C_4}, \pi_{1,C_7}, \pi_{1,C_9}, \pi_{1,C_{13}}, \pi_{1,C_{15}}, \pi_{3,C_{15}}, \pi_{4,C_{15}}\}$
τ_3	Track Trajectory	$\left\{ \begin{array}{l} \pi_{1,C_1}, \pi_{1,C_2}, \pi_{1,C_3}, \pi_{2,C_3}, \pi_{3,C_3}, \pi_{1,C_4}, \pi_{3,C_4}, \pi_{4,C_4}, \pi_{1,C_5}, \pi_{3,C_5}, \\ \pi_{1,C_6}, \pi_{2,C_6}, \pi_{1,C_7}, \pi_{2,C_7}, \pi_{3,C_7}, \pi_{1,C_8}, \pi_{2,C_8}, \pi_{3,C_8}, \pi_{1,C_{10}}, \pi_{2,C_{10}}, \\ \pi_{3,C_{10}}, \pi_{3,C_{11}}, \pi_{1,C_{12}}, \pi_{2,C_{12}}, \pi_{1,C_{13}}, \pi_{1,C_{14}}, \pi_{2,C_{14}}, \pi_{1,C_{15}}, \pi_{2,C_{15}} \end{array} \right\}$
τ_4	Recovery WiFi Signal	$\left\{ \begin{array}{l} \pi_{1,C_1}, \pi_{1,C_4}, \pi_{4,C_4}, \pi_{1,C_7}, \pi_{2,C_7}, \pi_{3,C_7}, \pi_{1,C_8}, \\ \pi_{2,C_8}, \pi_{3,C_8}, \pi_{1,C_{11}}, \pi_{1,C_{12}}, \pi_{2,C_{12}}, \pi_{1,C_{13}}, \pi_{1,C_{14}}, \pi_{2,C_{14}} \end{array} \right\}$
τ_5	Return to Base	$\{\pi_{1,C_1}, \pi_{1,C_4}, \pi_{4,C_4}, \pi_{1,C_7}, \pi_{2,C_7}, \pi_{1,C_8}, \pi_{2,C_8}, \pi_{3,C_8}, \pi_{1,C_{11}}, \pi_{1,C_{12}}, \pi_{2,C_{12}}, \pi_{1,C_{13}}\}$
τ_6	Store Data	$\{\pi_{2,C_{11}}, \pi_{1,C_{13}}\}$
τ_7	Cool Central Processing Unit	$\{\pi_{1,C_1}, \pi_{1,C_2}, \pi_{1,C_3}, \pi_{1,C_4}, \pi_{1,C_5}, \pi_{1,C_7}, \pi_{1,C_{10}}, \pi_{1,C_{12}}, \pi_{1,C_{13}}, \pi_{1,C_{14}}, \}$
τ_8	Label Map	$\{\pi_{1,C_2}, \pi_{1,C_5}, \pi_{3,C_5}, \pi_{1,C_6}, \pi_{3,C_6}, \pi_{2,C_9}, \pi_{2,C_{11}}, \pi_{1,C_{13}}, \pi_{1,C_{15}}\}$
τ_9	Serve Operator Request	$\left\{ \begin{array}{l} \pi_{1,C_2}, \pi_{1,C_3}, \pi_{1,C_4}, \pi_{1,C_5}, \pi_{1,C_7}, \pi_{2,C_7}, \pi_{3,C_7}, \pi_{2,C_9}, \pi_{1,C_{10}}, \\ \pi_{2,C_{10}}, \pi_{1,C_{10}}, \pi_{2,C_{11}}, \pi_{1,C_{12}}, \pi_{1,C_{13}}, \pi_{1,C_{14}}, \pi_{2,C_{14}}, \pi_{1,C_{15}} \end{array} \right\}$
τ_{10}	Approach Object/Person/Sound Source	$\left\{ \begin{array}{l} \pi_{1,C_1}, \pi_{1,C_2}, \pi_{1,C_3}, \pi_{2,C_3}, \pi_{3,C_3}, \pi_{1,C_4}, \pi_{2,C_4}, \pi_{3,C_4}, \pi_{4,C_4}, \pi_{1,C_5}, \\ \pi_{3,C_5}, \pi_{1,C_6}, \pi_{1,C_7}, \pi_{2,C_7}, \pi_{3,C_7}, \pi_{1,C_8}, \pi_{2,C_8}, \pi_{3,C_8}, \pi_{2,C_{11}}, \pi_{1,C_{12}}, \\ \pi_{2,C_{12}}, \pi_{1,C_{13}}, \pi_{1,C_{14}}, \pi_{2,C_{14}}, \pi_{1,C_{15}}, \pi_{2,C_{15}}, \pi_{4,C_{15}}, \pi_{5,C_{15}} \end{array} \right\}$
τ_{11}	Overcome Obstacle	$\{\pi_{1,C_3}, \pi_{2,C_3}, \pi_{3,C_3}, \pi_{1,C_4}, \pi_{3,C_4}, \pi_{1,C_7}, \pi_{2,C_7}, \pi_{3,C_7}, \pi_{4,C_7}, \pi_{1,C_{13}}.\pi_{1,C_{15}}, \pi_{5,C_{15}}\}$
τ_{12}	Data Retrieval	$\{\pi_{1,C_2}, \pi_{1,C_{11}}, \pi_{1,C_{13}}\}$
τ_{13}	Robot System Reconfiguration	$\{\pi_{1,C_3}, \pi_{1,C_5}, \pi_{2,C_5}, \pi_{1,C_7}, \pi_{1,C_{10}}, \pi_{1,C_{12}}, \pi_{1,C_{13}}, \pi_{2,C_{16}}\}$

Table VIII. Stimuli Associated with the Yields of the Robot Processes

(a)

Process Id	Stimulus Id	Stimulus Description
π_{1,C_1}	$y_{\pi_{1,C_1}}$	Battery Exhaustion
π_{1,C_2}	$y_{\pi_{1,C_2}}$	WiFi Disconnection
π_{1,C_3}	$y_{\pi_{1,C_3}}$	3D Laser Device Error
π_{2,C_3}	$y_{\pi_{2,C_3}}$	3D Laser Rotation Stuck
π_{3,C_3}	$y_{\pi_{3,C_3}}$	Scan Data Not Available
π_{1,C_4}	$y_{\pi_{1,C_4}}$	SLAM ROS Nodes Died
π_{2,C_4}	$y_{\pi_{2,C_4}}$	2D Map Not Available
π_{3,C_4}	$y_{\pi_{3,C_4}}$	3D Map Not Available
π_{4,C_4}	$y_{\pi_{4,C_4}}$	Coordinate Frames Transformation Error
π_{1,C_5}	$y_{\pi_{1,C_5}}$	Vision System Device Error
π_{2,C_5}	$y_{\pi_{2,C_5}}$	Calibration Error
π_{3,C_5}	$y_{\pi_{3,C_5}}$	Camera Images Not Available
π_{1,C_6}	$y_{\pi_{1,C_6}}$	Visual Perception ROS Nodes Died
π_{2,C_6}	$y_{\pi_{2,C_6}}$	Detected Object/Person
π_{3,C_6}	$y_{\pi_{3,C_6}}$	Recognized Object/Person
π_{1,C_7}	$y_{\pi_{1,C_7}}$	Engine Device Error
π_{2,C_7}	$y_{\pi_{2,C_7}}$	Locomotion System Stuck
π_{3,C_7}	$y_{\pi_{3,C_7}}$	Temperature Level
π_{4,C_7}	$y_{\pi_{4,C_7}}$	Differential Drive Stuck
π_{1,C_8}	$y_{\pi_{1,C_8}}$	Trajectory Control Nodes Died
π_{2,C_8}	$y_{\pi_{2,C_8}}$	Planning Error

(b)

Process Id	Stimulus Id	Stimulus Description
π_{3,C_8}	$y_{\pi_{3,C_8}}$	Tracking Error
π_{1,C_9}	$y_{\pi_{1,C_9}}$	Human to Robot Message
π_{2,C_9}	$y_{\pi_{2,C_9}}$	Robot to Human Message
$\pi_{1,C_{10}}$	$y_{\pi_{1,C_{10}}}$	Microphone Device Error
$\pi_{2,C_{10}}$	$y_{\pi_{2,C_{10}}}$	Signal Noise Ratio
$\pi_{3,C_{10}}$	$y_{\pi_{3,C_{10}}}$	Recognized Speech
$\pi_{1,C_{11}}$	$y_{\pi_{1,C_{11}}}$	Read Access to Missing Data
$\pi_{2,C_{11}}$	$y_{\pi_{2,C_{11}}}$	Write Access to Memory Full
$\pi_{3,C_{11}}$	$y_{\pi_{3,C_{11}}}$	Memory State
$\pi_{1,C_{12}}$	$y_{\pi_{1,C_{12}}}$	IMU Device Error
$\pi_{2,C_{12}}$	$y_{\pi_{2,C_{12}}}$	Localization Accuracy
$\pi_{1,C_{13}}$	$y_{\pi_{1,C_{13}}}$	ROS Master Connection
$\pi_{1,C_{14}}$	$y_{\pi_{1,C_{14}}}$	Central Processing Unit Overloaded
$\pi_{2,C_{14}}$	$y_{\pi_{2,C_{14}}}$	Temperature Level
$\pi_{1,C_{15}}$	$y_{\pi_{1,C_{15}}}$	Terrain Analysis Nodes Died
$\pi_{2,C_{15}}$	$y_{\pi_{2,C_{15}}}$	Stability Index
$\pi_{3,C_{15}}$	$y_{\pi_{3,C_{15}}}$	Terrain Roughness
$\pi_{4,C_{15}}$	$y_{\pi_{4,C_{15}}}$	Detected Obstacle
$\pi_{5,C_{15}}$	$y_{\pi_{5,C_{15}}}$	Flippers Configuration Error
$\pi_{1,C_{16}}$	$y_{\pi_{1,C_{16}}}$	Critical Error
$\pi_{2,C_{16}}$	$y_{\pi_{2,C_{16}}}$	Recovery Status

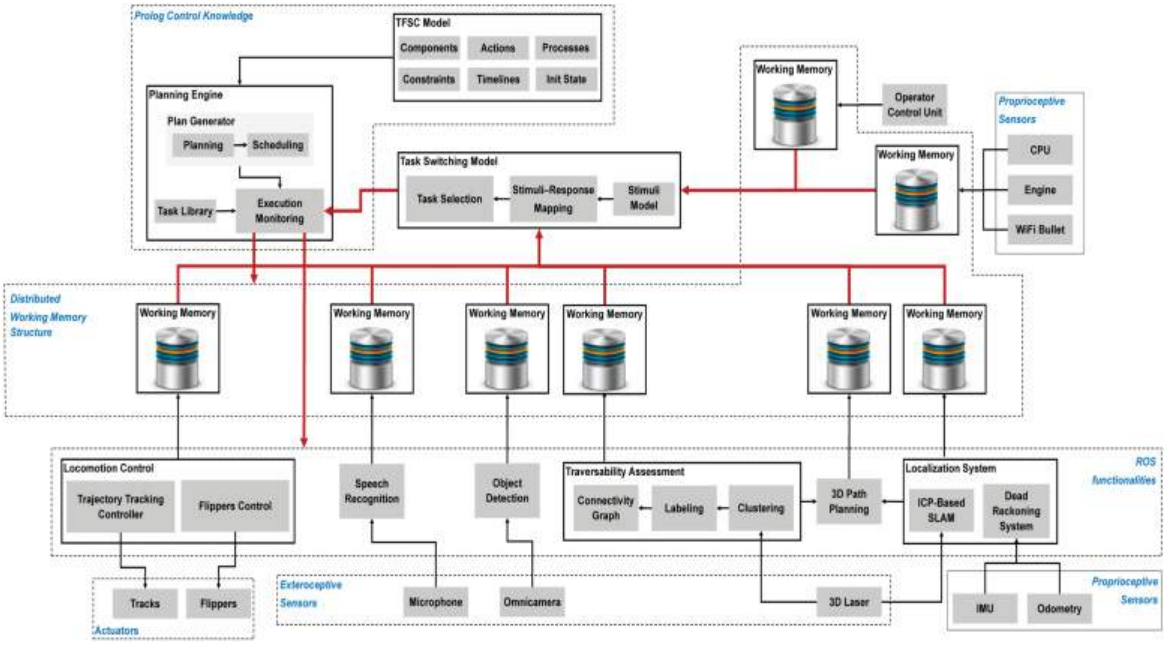


Fig. 10. Distributed working memory structure.

operational modalities that lie between autonomous and teleoperated modes during the execution of a task.

A.5. Task Switching Learning Interface

An interactive interface serves in dataset gathering (see Figure 11). The layout has four principal panels: (1) the stimulus panel, (2) the robot state panel, (3) the task state panel, and (4) the control panel. The stimulus panel includes vertical tabs listing robot components. Each tab is composed of a set of frames, each related to a component process and displaying the trend of the process and its yields. An additional button allows the operator to manually notify the robot of the occurrence of a stimulus. The operator can disable a frame via a check-box button. By selecting a tab in the stimulus panel, the operator can observe the state of each robot component, as well as the yields of the processes of that component. The robot state panel is composed of both a text box reporting the task the robot is currently accomplishing and a table listing all the running processes required to perform that task. The operator can indicate which task the robot has to perform via the tasks buttons displayed in the task panel. The control panel has two horizontal tabs: (1) the robot steering tab and (2) the flipper configuration tab. In the first tab, a number of control commands are listed for steering the robot. The latter serves to change flipper configuration and lock/unlock the differential drive to manually control the robot locomotion system. In addition, the main window reports both reconfiguration and interference costs. Three radio buttons set the interface for three different user interaction modalities: (1) passive mode, (2) active mode, and (3) acquisition mode. The passive mode allows the operator to manually notify the occurrence of the stimuli by observing the yields of the processes. The active mode enables a mixed robot-human initiative. In this modality, the interface displays the data yielded by the active processes and notifies the operator of the occurrence of a stimulus in real-time. Accordingly, the operator can select the task the robot has to perform or allow the robot to continue its current task. The acquisition mode gathers the active processes and shows the robot choice if a stimulus occurs. In this mode, the

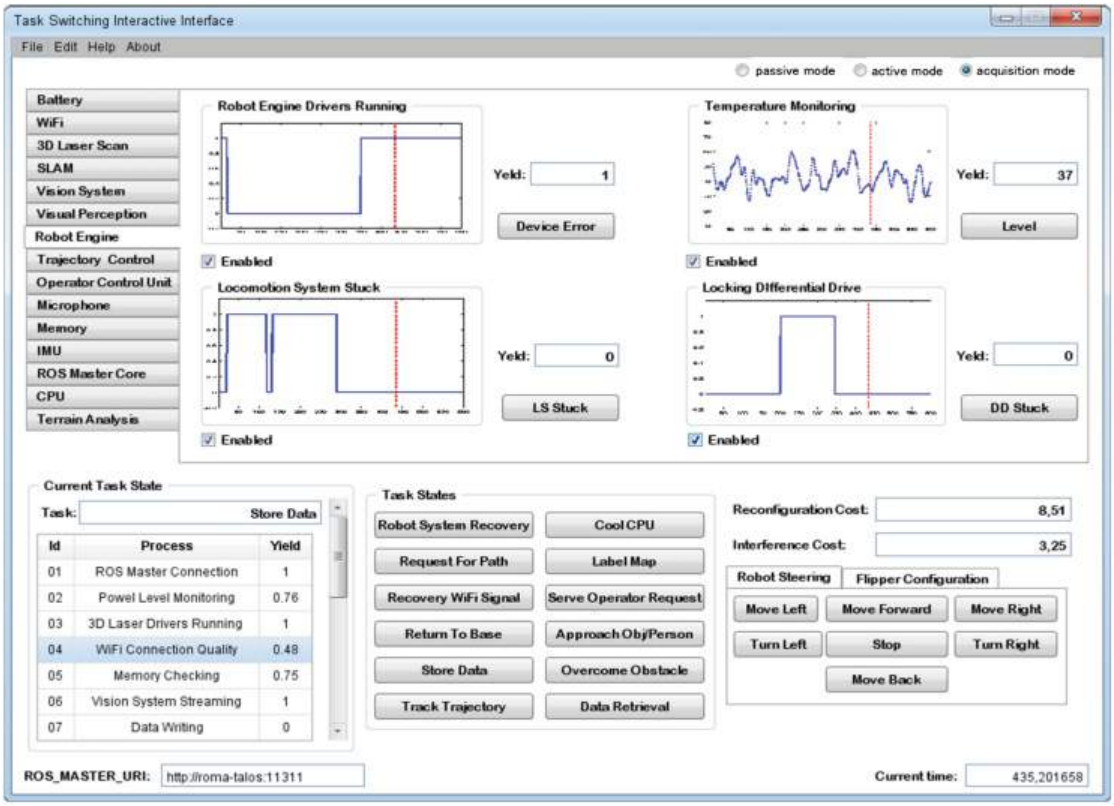


Fig. 11. Task switching interactive interface.

interface notifies the human operator of the occurrence of a stimulus, the choices taken by the robot to respond, and the task chosen; in this mode, the operator can store his or her choice for further analysis.

A.6. Stimulus-Response Matrix: An Example

Tables IX and X, read side by side, provide an example of the score matrix $\mathbf{A}$ introduced in Section 4. Each entry a_{ij} represents the score given by a stimulus $y_{\pi_{j,C_k}}$, triggered by the yield of the j-th process, managed by the k-th component, to a task τ_i. This matrix is filled with data gathered from the experiments described in Section 4 using the graphical user interface described in Section A.5. Dashes denote those stimulus-task pairs that the operators never selected during the experiments. Tables XI and XII, read side by side, form the estimated score matrix $\mathbf{A}$.

A.7. Task Failure Rates

For each task listed in Table VII , failure rates have been computed within the simulation environment ARE [Gianni et al. 2013a]. Each task is assigned a time for termination, and, during the task execution, no stimuli occur. For each task, we repeat up to 10 trials. In the ARE environment, it is possible to generate several events populating the simulated environment; events blocking one of the running processes determine the failure of the task. In Table XIII, we report the time to failure for each task in the simulated environment: In the first column, we list the task identifier, in the second the execution time to end the task, in the third column the number of failures, in the fourth the trial at which the failure occurred, and, in the last, the time at which failure took place.

Table IX. Entries of the Score Matrix (A)

	Stimuli																			
Task State	$y_{\pi_{1,C_1}}$	$y_{\pi_{1,C_2}}$	$y_{\pi_{1,C_3}}$	$y_{\pi_{2,C_3}}$	$y_{\pi_{3,C_3}}$	$y_{\pi_{1,C_4}}$	$y_{\pi_{2,C_4}}$	$y_{\pi_{3,C_4}}$	$y_{\pi_{4,C_4}}$	$y_{\pi_{1,C_5}}$	$y_{\pi_{2,C_5}}$	$y_{\pi_{3,C_5}}$	$y_{\pi_{1,C_6}}$	$y_{\pi_{2,C_6}}$	$y_{\pi_{3,C_6}}$	$y_{\pi_{1,C_7}}$	$y_{\pi_{2,C_7}}$	$y_{\pi_{3,C_7}}$	$y_{\pi_{4,C_7}}$	$y_{\pi_{1,C_8}}$	$y_{\pi_{2,C_8}}$
τ_1	-	.06	.40	.50	.40	.45	.13	.35	.60	.20	.80	.20	.60	-	-	.85	.70	-	.45	.71	.30
τ_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	.12	-	-	.05	.2	-	.62
τ_3	.08	.07	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	.05	.20	-	-
τ_4	-	.46	-	-	-	-	.10	.10	-	-	-	.27	.10	-	-	-	-	-	-	-	-
τ_5	.41	.20	.25	-	.20	-	.10	-	-	-	-	.25	.16	-	-	-	-	.20	-	-	-
τ_6	.16	-	.08	.15	-	.18	.12	-	-	-	-	-	-	.17	.18	-	.14	.10	-	.21	-
τ_7	.08	-	-	-	-	-	-	-	-	-	-	-	-	-	-	.13	.14	.10	-	-	-
τ_8	-	-	.07	-	-	-	-	-	-	-	-	-	-	.43	.18	-	-	.05	-	-	-
τ_9	-	-	-	-	-	-	-	.11	.11	-	-	-	-	-	.11	-	-	-	-	-	-
τ_{10}	-	-	-	-	-	-	-	-	-	-	-	-	-	.13	.30	-	-	-	-	-	-
τ_{11}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
τ_{12}	.10	-	-	.10	.20	.27	.50	.21	.16	.18	-	-	-	-	-	-	-	-	-	-	-
τ_{13}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	.30	-	-	-

Table X. Entries of the Score Matrix (B)

	Stimuli																
Task State	$y_{\pi_{3,C_8}}$	$y_{\pi_{1,C_9}}$	$y_{\pi_{2,C_9}}$	$y_{\pi_{1,C_{10}}}$	$y_{\pi_{2,C_{10}}}$	$y_{\pi_{3,C_{10}}}$	$y_{\pi_{1,C_{11}}}$	$y_{\pi_{2,C_{11}}}$	$y_{\pi_{3,C_{11}}}$	$y_{\pi_{1,C_{12}}}$	$y_{\pi_{2,C_{12}}}$	$y_{\pi_{1,C_{13}}}$	$y_{\pi_{1,C_{14}}}$	$y_{\pi_{2,C_{14}}}$	$y_{\pi_{1,C_{15}}}$	$y_{\pi_{2,C_{15}}}$	$y_{\pi_{3,C_{15}}}$	$y_{\pi_{4,C_{15}}}$	$y_{\pi_{5,C_{15}}}$	$y_{\pi_{1,C_{16}}}$	$y_{\pi_{2,C_{16}}}$
τ_1	-	-	-	.25	.22	-	.10	.20	-	.63	.14	.75	.38	.23	.33	-	-	-	.37	-	-
τ_2	.61	-	.66	.12	-	.10	-	-	.09	-	-	-	-	-	.17	.13	-	.25	.12	-	-
τ_3	.10	.30	-	.10	-	.10	-	-	.08	-	.42	-	-	-	.16	.20	.43	-	.12	-	-
τ_4	-	-	-	-	-	-	.10	-	-	-	-	-	-	-	-	-	-	-	-	-	-
τ_5	.10	-	-	-	-	-	.20	.20	.11	-	.14	-	.16	-	.16	.12	.28	.08	.12	-	.78
τ_6	-	.07	-	-	-	.12	.55	-	.62	-	-	-	.08	.23	.15	-	-	-	-	-	-
τ_7	-	-	-	-	-	-	-	-	-	-	-	-	.08	.38	-	-	-	-	-	-	-
τ_8	-	-	.22	.11	-	.12	-	-	-	-	-	-	-	-	-	-	-	.25	-	-	-
τ_9	-	.39	.10	.12	.19	.22	-	-	-	-	-	-	.08	-	-	-	-	-	-	-	-
τ_{10}	.12	.16	-	.12	.45	.30	-	-	-	-	.14	-	-	-	-	.37	.14	-	.12	-	-
τ_{11}	-	-	-	-	-	-	-	-	-	-	.15	-	-	-	-	.10	.14	.25	-	-	-
τ_{12}	-	-	-	.12	-	-	-	.55	-	.09	-	-	-	-	-	-	-	-	.13	-	-
τ_{13}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	.74	-

Table XI. Completed Score Matrix (A)

	Stimuli															
Task State	y_{π_1,C_1}	y_{π_1,C_2}	y_{π_1,C_3}	y_{π_2,C_3}	y_{π_3,C_3}	y_{π_1,C_4}	y_{π_2,C_4}	y_{π_3,C_4}	y_{π_4,C_4}	y_{π_1,C_5}	y_{π_2,C_5}	y_{π_3,C_5}	y_{π_1,C_6}	y_{π_2,C_6}	y_{π_3,C_6}	y_{π_1,C_7}	y_{π_2,C_7}	y_{π_3,C_7}	y_{π_4,C_7}	y_{π_1,C_8}	y_{π_2,C_8}
τ_1	.04	.15	.39	.49	.41	.44	.13	.35	.60	.20	.79	.21	.61	.01	.02	.84	.72	.04	.45	.71	.30
τ_2	.02	.02	.03	.02	.01	.01	.01	.03	.01	.05	.02	.01	.01	.01	.12	.01	.01	.03	.20	.01	.62
τ_3	.08	.07	.01	.03	.01	.01	.01	.01	.01	.05	.02	.02	.02	.01	.01	.01	.02	.05	.20	.01	.01
τ_4	.02	.45	.02	.04	.02	.01	.10	.10	.01	.05	.02	.26	.10	.02	.01	.01	.00	.03	.02	.01	.01
τ_5	.42	.20	.24	.02	.20	.01	.10	.03	.01	.05	.02	.24	.13	.02	.01	.01	.00	.19	.01	.01	.01
τ_6	.17	.01	.09	.15	.02	.19	.10	.01	.01	.06	.02	.04	.03	.17	.17	.01	.13	.11	.01	.21	.01
τ_7	.08	.02	.01	.02	.01	.01	.01	.03	.01	.06	.02	.01	.01	.03	.01	.13	.15	.10	.01	.01	.01
τ_8	.02	.01	.07	.02	.02	.01	.01	.01	.01	.06	.02	.02	.01	.43	.17	.00	.01	.05	.01	.01	.01
τ_9	.02	.01	.02	.02	.02	.01	.01	.11	.11	.05	.02	.01	.01	.02	.12	.01	.01	.02	.02	.01	.01
τ_{10}	.02	.01	.02	.02	.02	.01	.01	.02	.03	.05	.02	.02	.02	.13	.29	.01	.00	.03	.02	.01	.01
τ_{11}	.02	.01	.02	.01	.01	.02	.01	.03	.01	.05	.02	.03	.02	.02	.01	.01	.01	.02	.04	.01	.01
τ_{12}	.10	.01	.02	.11	.20	.26	.48	.20	.14	.18	.02	.01	.01	.02	.06	.00	.02	.01	.01	.01	.01
τ_{13}	.01	.03	.06	.05	.05	.04	.02	.07	.04	.08	.02	.11	.02	.11	.02	.01	.01	.30	.03	.01	.01

Table XII. Completed Score Matrix (B)

	Stimuli																
Task State	$y_{\pi_{3,C_8}}$	$y_{\pi_{1,C_9}}$	$y_{\pi_{2,C_9}}$	$y_{\pi_{1,C_{10}}}$	$y_{\pi_{2,C_{10}}}$	$y_{\pi_{3,C_{10}}}$	$y_{\pi_{1,C_{11}}}$	$y_{\pi_{2,C_{11}}}$	$y_{\pi_{3,C_{11}}}$	$y_{\pi_{1,C_{12}}}$	$y_{\pi_{2,C_{12}}}$	$y_{\pi_{1,C_{13}}}$	$y_{\pi_{1,C_{14}}}$	$y_{\pi_{2,C_{14}}}$	$y_{\pi_{1,C_{15}}}$	$y_{\pi_{2,C_{15}}}$	$y_{\pi_{3,C_{15}}}$	$y_{\pi_{4,C_{15}}}$	$y_{\pi_{5,C_{15}}}$	$y_{\pi_{1,C_{16}}}$	$y_{\pi_{2,C_{16}}}$
τ_1	.008	.01	.004	.240	.210	.005	.100	.200	.012	.610	.140	.750	.370	.220	.330	.009	.001	.018	.370	.020	.016
τ_2	.600	.005	.670	.110	.010	.110	.007	.005	.090	.044	.001	.020	.014	.015	.170	.130	.001	.250	.120	.024	.020
τ_3	.110	.290	.002	.120	.022	.090	.003	.009	.080	.026	.390	.030	.002	.017	.160	.200	.430	.017	.120	.021	.015
τ_4	.006	.020	.001	.005	.008	.006	.100	.005	.010	.021	.001	.050	.004	.013	.004	.011	.001	.019	.003	.020	.049
τ_5	.120	.007	.002	.015	.010	.004	.190	.200	.120	.029	.140	.010	.160	.019	.170	.120	.280	.080	.120	.033	.750
τ_6	.007	.070	.001	.013	.005	.110	.560	.001	.610	.030	.001	.020	.080	.230	.150	.010	.001	.015	.001	.021	.019
τ_7	.005	.007	.002	.007	.015	.015	.010	.007	.011	.020	.003	.030	.080	.380	.004	.010	.001	.021	.003	.021	.021
τ_8	.007	.009	.210	.110	.010	.110	.004	.003	.012	.025	.001	.020	.002	.015	.003	.013	.001	.250	.004	.022	.018
τ_9	.009	.380	.100	.130	.200	.230	.005	.004	.010	.030	.001	.020	.080	.016	.003	.007	.001	.026	.002	.021	.019
τ_{10}	.100	.170	.002	.110	.440	.290	.006	.006	.012	.027	.140	.030	.003	.017	.005	.370	.140	.017	.120	.024	.016
τ_{11}	.008	.009	.003	.008	.010	.010	.003	.007	.011	.023	.150	.010	.002	.016	.004	.100	.140	.250	.004	.021	.020
τ_{12}	.006	.007	.002	.120	.010	.017	.005	.550	.009	.090	.001	.020	.002	.014	.003	.015	.001	.019	.130	.023	.018
τ_{13}	.014	.016	.001	.012	.050	.003	.007	.003	.013	.025	.001	.020	.003	.018	.004	.005	.001	.018	.003	.710	.019

Table XIII. Task Time to Failure

Task Id	Execution Time (min.)	# of Failures	Failure time (min.)	
τ_1	18	0	-	-
τ_2	16	1	Trial 3	15.23
τ_3	46	4	Trial 2	12.51
			Trial 3	28.09
			Trial 6	31.11
			Trial 9	5.06
τ_4	31	3	Trial 4	19.01
			Trial 7	18.75
			Trial 6	19.23
τ_5	82	4	Trial 1	10.91
			Trial 2	40.05
			Trial 4	19.32
			Trial 10	47.09
τ_6	25	1	Trial 8	7.51
τ_7	27	0	-	-
τ_8	55	2	Trial 1	39.41
			Trial 8	38.02
τ_9	46	3	Trial 2	28.91
			Trial 3	29.78
			Trial 9	20.03
τ_{10}	100	4	Trial 1	12.01
			Trial 3	27.09
			Trial 8	81.53
			Trial 9	82.09
τ_{11}	68	3	Trial 5	51.79
			Trial 9	28.06
			Trial 10	31.55
τ_{12}	39	2	Trial 1	11.77
			Trial 7	12.33
τ_{13}	48	3	Trial 5	39.60
			Trial 6	40.79
			Trial 10	26.56

ACKNOWLEDGMENTS

We also thank reviewers for their thorough advices in improving this article.

REFERENCES

A. Ajoudani, M. Gabiccini, N. G. Tsagarakis, and A. Bicchi. 2013. Human-like impedance and minimum effort control for natural and efficient manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 4499–4505.

J. F. Allen. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26, 11 (1983), 832–843.

P. Althaus and H. I. Christensen. 2003. Smooth task switching through behaviour competition. *Robotics and Autonomous Systems* 44, 3–4 (2003), 241–249.

K. Apt and M. Wallace. 2006. *Constraint Logic Programming using Eclipse*. Cambridge University Press, New York.

A. R. Aron. 2007. The neural basis of inhibition in cognitive control. *The Neuroscientist* 13 (2007), 214–228.

T. Baar, B. Beckert, and P. H. Schmitt. 2001. An extension of dynamic logic for modelling ocls @pre operator. In *Perspectives of System Informatics*, D. Bjrner, M. Broy, and A.V. Zamulin (Eds.). Lecture Notes in Computer Science, Vol. 2244. Springer, Berlin, 47–54.

M. Beetz, L. Mosenlechner, and M. Tenorth. 2010. CRAM: A cognitive robot abstract machine for everyday manipulation in human environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 1012–1017.

S. A. Block, A. F. Wehowsky, and B. C. Williams. 2006. Robust execution on contingent, temporally flexible plans. In *Proceedings of the AAAI Workshop on Cognitive Robotics*. AAAI Press, Palo Alto, California, 802–808.

A. Blum and J. C. Langford. 1999. Probabilistic planning in the graphplan framework. In *Proceedings of the 5th European Conference on Planning*. Springer-Verlag, New York, NY, 8–12.

A. M. Buchanan and A. W. Fitzgibbon. 2005. Damped newton algorithms for matrix factorization with missing data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 316–322.

B. Cafaro, M. Gianni, F. Pirri, M. Ruiz, and A. Sinha. 2013. Terrain traversability in rescue environments. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics*. 1–8.

J. F. Canny. 2002. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. 238–245.

D. Cao, Q. Qu, C. Li, and C. He. 2009. Research of attitude estimation of UAV based on information fusion of complementary filter. In *Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology*. IEEE, 1290–1293.

G. Capi. 2007. Robot task switching in complex environments. In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*. IEEE, 1–6.

G. Capi, G. G. Pojani, and S. I. Kaneko. 2008. Evolution of task switching behaviors in real mobile robots. In *Proceedings of the 3rd International Conference on Innovative Computing Information and Control*. IEEE, 495–501.

A. Carbone, A. Finzi, A. Orlandini, and F. Pirri. 2008. Model-based control architecture for attentive robots in rescue scenarios. *Autonomous Robots* 24, 1 (2008), 87–120.

P. Chen. 2008. Optimization algorithms on subspaces: Revisiting missing data problem in low-rank matrix. *International Journal of Computer Vision* 80, 1 (2008), 125–142.

M. M. Cialdea and F. Pirri. 1993. First order abduction via tableau and sequent calculi. *Logic Journal of the IGPL* 1, 1 (1993), 99–117.

D. B. D'Ambrosio, J. Lehman, S. S. Risi, and K. O. Stanley. 2011. Task switching in multirobot learning through indirect encoding. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2802–2809.

R. Dechter, I. Meiri, and J. Pearl. 1991. Temporal constraint networks. *Artificial Intelligence* 49, 1–3 (1991), 61–95.

X. Ding and C. Fang. 2013. A novel method of motion planning for an anthropomorphic arm based on movement primitives. *IEEE/ASME Transactions on Mechatronics* 18, 2 (April 2013), 624–636.

K. T. Durkee, C. Shabarekh, C. Jackson, and G. Ganberg. 2011. Flexible autonomous support to aid context and task switching. In *Proceedings of the IEEE First International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support*. IEEE, 204–207.

R. Fikes and N. J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2, 3/4 (1971), 189–208.

M. Finger and D. Gabbay. 1996. Combining temporal logic systems. *Notre Dame Journal of Formal Logic* 37, 2 (1996), 204–232.

A. Finzi and F. Pirri. 2005. Representing flexible temporal behaviours in the situation calculus. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. AAAI Press, 436–441.

A. Finzi and F. Pirri. 2010. *Switching Tasks and Flexible Reasoning in the Situation Calculus*. Technical Report 7. DIAG.

M. Fox and D. Long. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20, 1 (2003), 61–124.

G. Ganesh and E. Burdet. 2013. Motor planning explains human behaviour in tasks with multiple solutions. *Robotics and Autonomous Systems* 61 (2013), 362–368. Issue 4.

M. Gianni, F. Ferri, and F. Pirri. 2013a. ARE: Augmented reality environment for mobile robots. In *Proceedings of the 14th Towards Autonomous Systems (TAROS)*. 470–483.

M. Gianni, G. Gonnelli, A. Sinha, M. Menna, and F. Pirri. 2013b. An augmented reality approach for trajectory planning and control of tracked vehicles in rescue environments. In *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics*. 1–6.

J. J. Gibson. 1960. The concept of the stimulus in psychology. *American Psychologist* 15, 11 (1960), 694–703.

M. Göbelbecker, C. Gretton, and R. Dearden. 2011. A switching planner for combined task and observation planning. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*.

N. Hawes, M. Hanheide, K. Sjöö, A. Aydemir, P. Jensfelt, M. Göbelbecker, M. Brenner, H. Zender, P. Lison, I. Kruijff-Korbayová, G. J. M Kruijff, and M. Zillich. 2010. Dora the explorer: A motivated robot. In *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems*, Vol. 1. International Foundation for Autonomous Agents and Multiagent Systems, Canada, 1617–1618.

D. Hurych, K. Zimmermann, and T. Svoboda. 2011. Fast learnable object tracking and detection in high-resolution omnidirectional images. In *Proceedings of the 9th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS, 521–530.

M. Ito, K. Noda, Y. Hoshino, and J. Tani. 2006. Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model. *Neural Networks* 19 (2006), 323–337.

L. Jamone, B. Damas, J. Santos-Victor, and A. Takanishi. 2013. Online learning of humanoid robot kinematics under switching tools contexts. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 4811–4817.

A. T. Jersild. 1927. Mental set and shift. *Archives of Psychology* 14, 89 (1927), 5–82.

S. Ko and J. Lee. 2002. User preference mining through collaborative filtering and content based filtering in recommender system. In *Proceedings of the 3rd International Conference on E-Commerce and Web Technologies*. 244–253.

V. Kubelka and M. Reinstein. 2012. Complementary filtering approach to orientation estimation using inertial sensors only. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 599–605.

J. L. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. T. Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems* 22, 1 (2004), 5–53.

Tao Li, K. Nakajima, and R. Pfeifer. 2013. Online learning for behavior switching in a soft robotic arm. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 1296–1302.

X. Li and C. C. Cheah. 2012. Multiple task-space robot control: Sense locally, act globally. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 265–270.

Pierre Marquis. 1991. Extending abduction from propositional to first-order logic. In *Fundamentals of Artificial Intelligence Research*. 141–155.

D. V. McDermott. 2003. PDDL2.1 - The art of the possible? Commentary on Fox and Long. *Journal of Artificial Intelligence Research* 20 (2003), 145–148.

A. I. Medina Ayala, S. B. Andersson, and C. Belta. 2012. Probabilistic control from time-bounded temporal logic specifications in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 4705–4710.

E. K. Miller and J. D. Cohen. 2007. An integrative theory of prefrontal cortex function. *Annual Review of Neuroscience* 24 (2007), 167–202.

J. Milln, F. Renkens, J. Mourino, and W. Gerstner. 2004. Brain-actuated interaction. *Artificial Intelligence* 159 (2004), 241–259.

S. Monsell. 2003. Task switching. *Trends in Cognitive Sciences* 7, 3 (2003), 134–140.

P. H. Morris, N. Muscettola, and T. Vidal. 2001. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*. 494–502.

M. Nakano, Y. Hasegawa, K. Funakoshi, J. Takeuchi, T. Torii, K. Nakadai, N. Kanda, K. Komatani, H. G Okuno, and H. Tsujino. 2011. A multi-expert model for dialogue and behavior control of conversational robots and agents. *Knowledge-Based Systems Journal* 24 (2011), 248–256. Issue 2.

A. Newell. 1990. *Unified Theories of Cognition*. Harvard University Press.

D. A. Norman and T. Shallice. 1986. *Consciousness and Self-Regulation: Advances in Research and Theory*, Vol. 4. Attention to action: Willed and automatic control of behaviour. Plenum Press.

P. Papadakis and F. Pirri. 2012. 3D mobility learning and regression of articulated, tracked robotic vehicles by physics-based optimization. In *Proceedings of the 9th Workshop on Virtual Reality Interactions and Physical Simulations*. 147–156.

F. Pirri. 2011. The well-designed logical robot: Learning and experience from observations to the Situation Calculus. *Artificial Intelligence* 175, 1 (2011), 378–415.

F. Pirri and R. Reiter. 1999. Some contributions to the metatheory of the situation calculus. *Journal of the ACM* 46, 3 (1999), 325–361.

F. Pirri and R. Reiter. 2000. Planning with natural actions in the situation calculus. In *Logic-Based Artificial Intelligence*. Kluwer Press, 213–231.

F. Pomerleau, F. Colas, S. Siegwart, and S. Magnenat. 2013. Comparing ICP variants on real-world data sets. *Autonomous Robots* 34, 3 (April 2013), 133–148.

M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. 2009. ROS: An open-source Robot Operating System. In *Proceedings of the Open-Source Workshop of the International Conference on Robotics and Automation*.

D. Randell, Z. Cui, and A. Cohn. 1992. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*. 165–176.

J. Rubinstein, D. E. Meyer, and J. E. Evans. 2001. Executive control of cognitive processes in task switching. *Journal of Experimental Psychology: Human Perception and Performance* 27, 4 (2001), 763–797.

C. O. Saglam and K. Byl. 2013. Stability and gait transition of the five-link biped on stochastically rough terrain using a discrete set of sliding mode controllers. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 5675–5682.

T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal. 2010. Path planning in 3D environments using the Normal Distributions Transform. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 3263–3268.

C. Sung, N. Ayanian, and D. Rus. 2013. Improving the performance of multi-robot systems by task switching. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 2999–3006.

S. Suzuki, T. Sasaki, and F. Harashima. 2009. Visible classification of task-switching strategies in vehicle operation. In *Proceedings of the 18th IEEE International Symposium on Robot and Human Interactive Communication*. 1161–1166.

T. Takayuki Okatani and K. Deguchi. 2006. On the Wiberg algorithm for matrix factorization in the presence of missing components. *International Journal of Computer Vision* 72, 3 (2006), 329–337.

K. Talamadupula, J. Benton, S. Kambhampati, P. Schermerhorn, and M. Scheutz. 2010. Planning for human-robot teaming in open worlds. *ACM Transactions on Intelligent Systems and Technology* 1, 2 (2010), 14:1–14:24.

L. Tao, K. Nakajima, M. Cianchetti, C. Laschi, and R. Pfeifer. 2012. Behavior switching using reservoir computing for a soft robotic arm. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 4918–4924.

S. P. Tipper. 2001. Does negative priming reflect inhibitory mechanisms? A review and integration of conflicting views. *The Quarterly Journal of Experimental Psychology* 54 (2001), 321–343.

M. Vilain, H. Kautz, and P. Beek. 1986. Constraint propagation algorithms for temporal reasoning. In *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, 377–382.

J. Wawerla and R. T. Vaughan. 2009. Robot task switching under diminishing returns. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 5033–5038.

E. H. Weber. 1834. *De pulsu, resorptione, audita et tactu. Annotationes anatomicae et physiologicae*. Koehler, New York, NY.

J. Wittocx, M. Denecker, and M. Bruynooghe. 2010. Constraint propagation for extended first-order logic. *Computing Research Repository* abs/1008.2121 (2010).

F. Wolter and M. Zakharyaschev. 2000. Spatio-temporal representation and reasoning based on RCC-8. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning*. 3–14.

J. L. Wyatt, A. Aydemir, M. Brenner, M. Hanheide, N. Hawes, P. Jensfelt, M. Kristan, G. J. M. Kruijff, P. Lison, A. Pronobis, K. Sjöö, A. Vrecko, H. Zender, M. Zillich, and D. Skocaj. 2010. Self-understanding and self-extension: A systems and representational approach. *IEEE Transactions on Autonomous Mental Development* 2, 4 (2010), 282–303.

Z. Zhang, K. Zhao, and H. Zha. 2012. Inducible regularization for low-rank matrix factorizations for collaborative filtering. *Neurocomputing* 97 (2012), 52–62.

Y. Zhou, D. M. Wilkinson, R. Schreiber, and R. Pan. 2008. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*. 337–348.

Received February 2013; revised August 2014; accepted October 2014