Integrating Classical Planning and Real Robots in Industrial and Service Robotics Domains

Oscar Lima, Rodrigo Ventura
Institute for Systems and Robotics
Instituto Superior Tecnico
Av. Rovisco Pais 1, 1049-001 Lisbon, Portugal
Email: {olima, rodrigo.ventura}@isr.tecnico.ulisboa.pt

Iman Awaad
Bonn-Rhein-Sieg University
Grantham-Allee 20, 53575
Sankt Augustin, Germany
Email: iman.awaad@h-brs.de

Abstract

In this paper we propose an architecture to integrate classical planning and real autonomous mobile robots. We start by providing with a high level description of all necessary components to set the goals, generate plans and execute them on real robots and monitor the outcome of their actions. At the core of our method and to deal with execution issues we code the agent actions with automatas. We prove the flexibility of the system by testing on two different domains: industrial (Basic Transportation Test) and domestic (General Purpose Service Robot) in the context of the international RoboCup competition. Additionally we benchmark the scalability of the planning system in two domains on a set of planning problems with increasing complexity. The proposed framework is open source[1] and can be easily extended.

1 Introduction

Bridging robotics and classical AI planning poses several challenges to both areas. In robotics one faces continuous time, temporal actions, concurrency, in the presence of partial observability, time constraints, stochastic events, to name a few. Classical planning is often formulated as discrete time with instantaneous actions, plans are sequential, and the world is assumed to be both fully observable and deterministic. However, classical planning research is now a mature field, providing a broad range planning methods that the community can benefit from. This paper addresses the problem of integrating such planners into real robots, in particular when such domains are highly unstructured and stochastic. In this paper we present an approach to use a classical planner by integrating it with an execution layer based on finite automata. We have tested the approach in two example realistic scenarios: industrial and service robotics domains.

1.1 Motivation

Artificial intelligence and robotics are two research areas that have benefited significantly from cross-fertilization, however, each of them tend to have their own research agenda with few researchers working on the intersection between them[2]. The International Planning Competition (IPC) and RoboCup are major tournaments for AI planning and robotics both of them which offer a common ground for benchmarking and valuable knowledge sharing. In RoboCup most teams in industrial and domestic domains use finite automata to coordinate the execution of particular skills, such as navigation and manipulation. However, big state machines are hard to maintain, unintuitive for humans to program and hard to reuse. The use of classical planners allows the specification of a goal and domain in an high-level planning domain. However, classical planning algorithms tend to not scale well with the complexity of the domain: it is not reasonable for a robotics domain for a planner to take more than a few seconds to obtain a feasible plan.

1.2 Problem Description

The problem is described as the integration of classical planning and real robots and how to overcome the limitations of classical planning when dealing with unstructured, stochastic, real world domains. We assume the availability of a set of robot skills, namely autonomous navigation, mobile manipulation, perception, and natural language interaction. Given a task, requiring a subset of these skills, we aim at integrating a classical planning into an execution layer of a robot. We focus particularly on the problem of near real-time planning time and robustness to unexpected action effects.

1.3 Structure of this work

In section 2 we start highlighting some of the features that are needed in robotics from the planning community, then we present existing approaches that have merged classical planning and robotics, then we talk about ROSPlan (Cashmore et al. 2015) as one of our dependencies and a close related work, then we briefly describe our selected solver Mercury (Katz and Hoffmann 2014) a well-known planner from IPC 2014, that scored in second place in the deterministic track. In section 4 we briefly describe each of the required components for planning in robotics and how they interact with each other by using a planning coordinator automata. In section 5 we talk about the robots used, their hardware, a description of the industrial and domestic domains that were modeled and the experiments that we designed to investigate some of the Mercury (Katz and Hoffmann 2014) features. Section 6 comments on the scalability and cost assignment

[1] www.github.com/oscar-lima/isr_planning

[2] AI summer school 2017 www.lucia.isr.tecnico.ulisboa.pt

results and sections 7 and 8 talk about the conclusions of this work.

2 Related Work

2.1 Planning under time constraints

In the international planning competition typically planners are provided with a 30 min timeout, however in service robotics (or @Home) for instance you can't afford having a robot thinking for more than e.g. 10 seconds, otherwise you deeply affect the human robot interaction process.

Fast approaches like real time (Korf 1990), deadline aware (Burns, Ruml, and Do 2013) or anytime search (Richter and Westphal 2010) are critical for service robotics.

2.2 Classical Planning in Robotics

One of the oldest examples of integration between AI and robotics is the Stanford Research Institute Problem Solver (STRIPS) (Fikes and Nilsson 1971). The famous automated planner was implemented on a real robot "Shakey" (Nilsson 1984). Without a doubt, a milestone in AI planning and robotics.

2.3 RoboCup Logistics League

In RoboCup (Kitano et al. 1997) Logistics league, the system used by the Carologistics (Niemueller et al. 2015) team (winner of 2014-2016) is based on the Fawkes Robot Software Framework (Niemueller, Reuter, and Ferrein 2015). The software stack contains components for localization, navigation, perception and basic behaviors using a Lua-base behavior engine and complete task-level executive based on the C Language Integrated Production System (CLIPS) (Wygant 1989). CLIPS is public domain software built for expert systems. It was initially developed in 1985 at the NASA Jhonson Space Center and presents an object-oriented language for writing expert systems. Like other languages CLIPS deals with rules and fact to operate. One of the main problems with ruled based systems is the amount of rules they require to work efficiently, making them hard to extend and maintain.

2.4 RoboCup at Work League

While most teams in the league use automatas for decision making, the team LUHbots @Work[3] from Hannover, uses a graph-based search algorithm (greedy) and a minimization cost function.

2.5 ROSPlan

Developed at King's College London University KCL ROSPlan (Cashmore et al. 2015) introduces a framework with a generic method for the integration of PDDL planning and the famous Robot Operating System (ROS) which is the standard middleware in robotics. It exposes a Knowledge base (KB), an automatic PDDL problem generator and plan dispatcher to interact with the dispatcher.

Re-planning in ROSPlan is done based upon 3 different criteria: because the dispatcher reports failure on an action, the KB changed in such a way that invalidates the current plan, the action has consumed to much resources (e.g. time or energy).

[3] http://luhbots.de/wordpress/

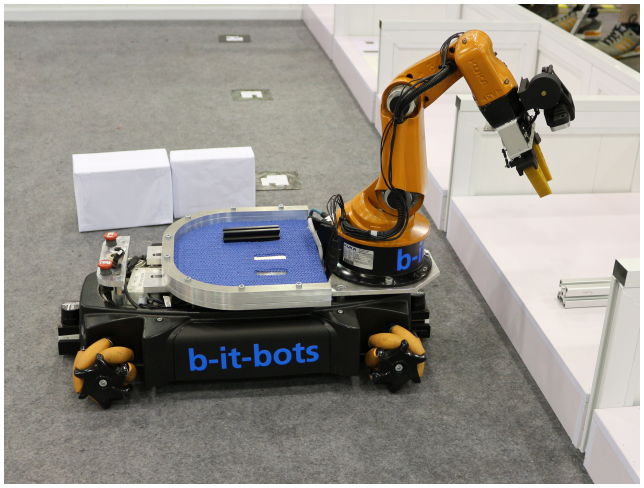

Figure 1: KUKA™YouBot industrial robot at RoboCup world championship 2015 Hefei, China.

2.6 Mercury Planner

Mercury (Katz and Hoffmann 2014) is a sequential, classical, satisficing planner (no optimal solution is guaranteed) which won 2nd place at IPC 2014. The planner starts with an initial greedy best first search and once a solution has being found it performs multiple iterations of heuristic search with a weighted A* algorithm. Mercury uses a partial relaxation delete list heuristic called Red-Black. Red variables take the relaxed semantics (ignore the delete list) while black variables take the regular semantics (will not ignore the delete list) (Katz and Hoffmann 2014). The method used to find black variables is called the paint strategy and is domain dependent. The rationale behind having selected this planner was because it outperformed all other planners in the transport domain (in IPC 2014), which is similar to our industrial domain : basic robot transportation tasks.

3 Background

The domains were modeled based on the tasks proposed by the international RoboCup competition @Work and @Home leagues and in particular, the Basic Transportation Test (BTT) and General Purpose Service Robot (GPSR).

3.1 Industrial Domain

In the BTT domain, the industrial KUKA YouBot robot (Bischoff, Huggenberger, and Prassler 2011) (see Figure 1), the standard platform of the league needs to transport objects between locations. The robot has a rear metal platform where a maximum of three objects can be stored and a robotic manipulator with a gripper that can fetch small objects. Additionally a RGBD camera was placed near the end effector to perceive the objects in the environment.

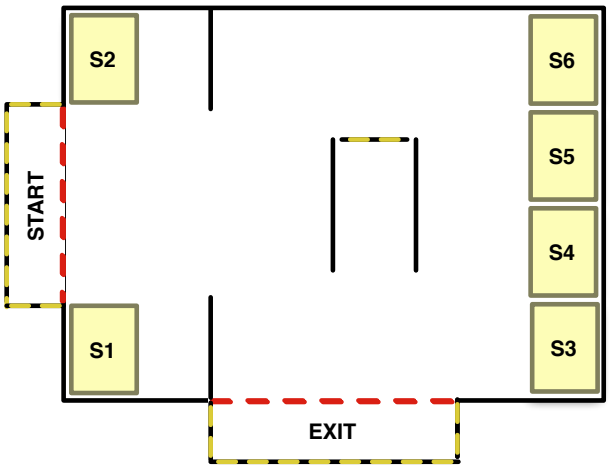

Figure 2: Basic transportation test domain simplified diagram.

A diagram of the simplified environment is depicted in Figure 2.

From the planning perspective the domain is composed of the following actions: move to location, perceive, pick, place, stage and unstage object. The full domain PDDL definition can be consulted online[4].

3.2 Domestic Domain

The General Purpose Service Robot (GPSR) domain tests the ability of the robots to respond to various commands given from human beings. The robot is provided with a list of locations, items, persons and objects and is expected to execute flexible commands such as: guiding people from source to destination, answer questions, introduce himself, grasp and place objects (transportation) or telling something to someone.

The robot that we use for this purpose is the MOnarCH robot (see Figure 3) which is equipped with various sensors and actuators that allow him to interact in the home scenario.

4 System Architecture

Our integration strategy is to use commercial off-the-shelf software while developing custom components we are interested from a research perspective. We briefly describe them in section 4.1.

4.1 Component Description

Speech recognition, natural language understanding and intention to knowledge - In the domestic domain a human needs to interact with the robot, we do this via a 3 step pipeline (see Figure 4). The speech recognition module inputs audio stream from the robot's microphone into the computer and converts it into a sentence. The sentence is input to the natural language understanding component that

[4] www.github.com/oscar-lima/mercury_planner_experiments

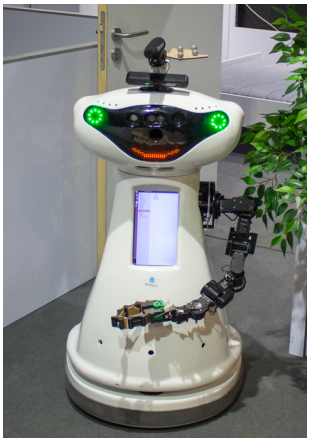

Figure 3: MOnarCH service robot.

first divides the sentence into multiple phrases and recognizes the intention and arguments, e.g. sentence: "*go to the kitchen and pick the water bottle*", would get divided into 2 phrases: "*go to the kitchen*" and "*pick the water bottle*", from each phrase we extract the intentions: "*go, grasp*" with the arguments: "*kitchen, water bottle*". Finally we map the intention to knowledge, e.g. (at robot destination), (holding object robot), where destination is kitchen and object is water_bottle.

Knowledge Base (KB) - We reuse this components from ROSPlan (Cashmore et al. 2015) to store the instances, facts and goals that are required for the planning process. We define three possible ways to interact with the knowledge base.

1. At startup fixed initial conditions can be uploaded to KB, e.g. A robot is at the entrance and its gripper is empty.
2. A human can provide facts or goals through voice.
3. The dispatcher (based upon success or action failure) can update the KB with the world state.

PDDL problem generator - We partially reuse this components from ROSPlan (Cashmore et al. 2015), fetch instances, facts and goals from KB to construct a PDDL problem definition. Only one thing is missing: cost. We have extended ROSPlan to accept and produce PDDL problems with cost information. We compute the cost by calling a motion planning algorithm based upon a particular environment, calculating distance between locations and generating a distance matrix that increases the cost function as en effect of the navigation operator.

Knowledge base analyzer - This component answers the following questions: 1. are there unfinished goals in the KB? 2. Is there new knowledge in the KB? (w.r.t. the last query).

Planner - This component currently calls Mercury (Katz and Hoffmann 2014), however other PDDL planners could be used due to the flexible architecture that we developed.

Plan Validation (VAL) - The plan validation tool (Howey, Long, and Fox 2004) inputs the generated plan, the domain

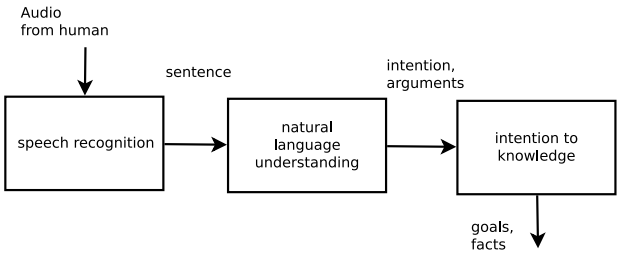

Figure 4: Human voice to knowledge pipeline.

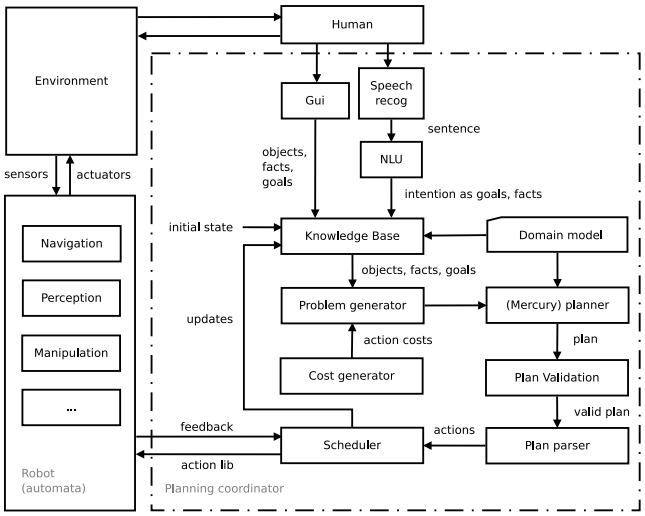

Figure 5: System architecture showing the specifics of the planning framework.

model and the problem definition and outputs a boolean response with information whether if the plan solves the imposed goals.

Plan parser - Currently structured to parse IPC formated plans, can be easily modified to adapt to other planners output. However most recent planners available in the community will produce the plan in the correct format.

Scheduler and execution layer - Receives the plan as a sequence of actions to be executed, iterates over each of them and updates the world state based on the action outcome (success or failure). The execution layer is based on high level actions logically structured by the domain model but on the inside each action is coded as an automata. This approach keeps each state machine well factored while maintaining the planner search space in control of the user (depends on how the domain is modeled).

4.2 Planning coordinator

In Figure 5 we present the planning architecture, it shows the interaction between components described in section 4.1.

The planning coordinator implementation itself is also an automata, depicted in Figure 6, notice that the framework

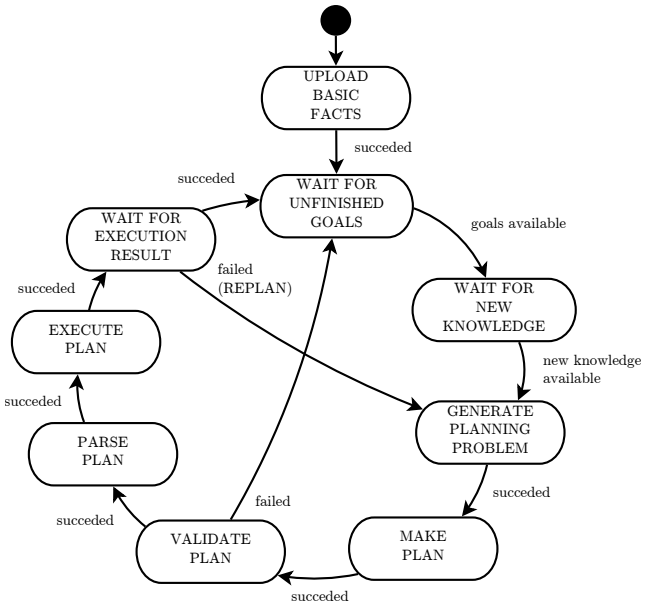

Figure 6: Planning coordinator automata.

allows you to implement your own re-planning strategy by creating your own automata.

The planning coordinator loop starts by uploading intrinsic or basic facts to the KB, then waits until unfinished goals are available in the KB, afterwards it checks if new knowledge is available. This step avoids loops when the planner fails to make a solution, preventing it from continuous failure and waiting until new knowledge arrives, before attempting to solve the problem again.

Next step is to automatically generate a PDDL problem instance from the Knowledge stored in the KB, then the produced plan is validated with the plan validation tool (VAL) (Howey, Long, and Fox 2004) to ensure that the solution solves the goal. Then we parse the planner output and convert it into a vector of actions that is sent for execution to individual action based automatas. Every time an actions is completed the dispatcher updates the KB. If one of the actions reports failure then we trigger re-planning.

4.3 Execution layer : State Machines

Planners are well known to suffer from curse of dimensionality, this usually leads to the advice: keep the domain as simple as possible. In our approach the domain expert has to balance this situation. There is always a trade off between flexibility and planning time.

Behind each high level planning operator there is an automata that deals partially with the complexity of the stochastic domain. In this approach there is a mutual benefit: the planning domain helps to refactor and logically organize each individual automata and they in return help the planner to keep its actions simple enough to be able to find a plan in real time.

The state machine refactoring was guided by the operators within the planning domain. This is one more example of how planning theory can help guide roboticists. What

existed before the integration of planning technology were monolithic state machines that would carry out the tasks seen in the domains. The burden of providing a state machine responsible for the whole plan fell to the developers. The use of automated planning technology enabled us to simply model the domains and create a state machine for each operator leaving the planner to decide which sequence of actions would constitute the plan. This refactoring was thoroughly explained in previous work (Lima 2016).

5 Experiment Setup

We performed three different experiments, in the first one we have participated in two RoboCup international scientific competition (China 2015, Leipzig 2016) to evaluate against other teams which are mostly using automatas (only two @Work teams used planning). Since this work presents results in planning, execution and monitoring on real robots we benchmark our robot with this real life experiment.

In the second and third experiment we focused on the planner. The first experiment aims to examine the scalability of the planner. We start giving a small problem to solve and we increase the amount of goals gradually. The experiments were performed on computers with 5 cores and 2-8 GB of RAM (2GB for @Work scenario and 8GB for @Home).

5.1 Mercury planner parameters

The planner has mainly the following search parameters to configure: timeout, cost type, Landmark (lm) cost type and heuristic weight. Since the planner is not anytime the timeout in this case will just basically interrupt the planning process without a solution. The cost type can be NORMAL, ONE and PLUSONE and it refers to the operator cost adjustment type. The lm cost type can be NORMAL, ONE, PLUSONE and it refers to the landmark action cost adjustment. 10 different parameter sets were selected for our experiments based on what the original authors have used in their planning scripts. Table 1 shows the selected parameters that from now on will be refereed as parameter set n.

	cost type	lm cost type	w
parameter set 1	1	1	1
parameter set 2	1	1	2
parameter set 3	1	1	3
parameter set 4	1	1	4
parameter set 5	1	1	5
parameter set 6	2	2	1
parameter set 7	2	2	2
parameter set 8	2	2	3
parameter set 9	2	2	4
parameter set 10	2	2	5

Table 1: 10 different search parameters were used for the experiments.

5.2 Scalability experiment

The problem instance for the @Work scenario requires the robot to transport objects between locations (see Figure 2). We start with one object to be transported and we gradually increase the amount of objects one at a time until 25. A timeout of 1 minute was given to the planner to produce solution for all experiments.

For the @Home scenario we have generated an example problem instance where the robot has to guide a certain amount of people (1-25). Notice that the domain can do many other things (including the transportation of objects) and only one particular operator is being tested in this experiment (guide).

The idea behind the scalability experiments is to investigate how the planning time and plan quality (cost) behave when the problem size grows.

The navigation cost information for the @Work domain obeys a distance matrix generated from an example scenario (see Figure 2) and was calculated by using a motion planning algorithm between all locations. All other actions (perceive, pick, place, stage, unstage) are having unit cost.

The cost information used for the @Home experiment is as follows: navigation action (2), guide action (500), all others (1).

6 Experiment Results

Automated scripts were used to create different solutions to the proposed PDDL problems in a 26 hour experiment run. The planner was asked to create over 40,000 plans.

6.1 Scalability Tests

In Figure 7 we present the results for the scalability experiment, we can see that while planning time grows exponentially the plan length grows linearly. Additionally each planner parameters produces different plan quality and time, e.g. parameter set 1 ”saturates” fast and is unable to handle problems which are bigger than 5 objects to transport for 1 min timeout.

In Figure 8 we present the results from the scalability test in the @Home scenario. We can observe a similar behavior compared to @Work, but this particular domain is more complex, therefore “saturates” earlier.

7 Discussion

We have seen that one possibility to reduce the search space of the planner is to interleave high level actions with the use of automatas. The domain model provides with a logical structure on how such automatas should be factored. Modeling real robot domains can be quite challenging as it requires experience on the domain. Typically roboticists are focused on specific areas, e.g. navigation, manipulation, perception, human-robot interaction, but rarely on all of them.

Modeling such domains is usually an iterating process, you start with a subset of actions and you try to scale up the domain by adding more actions on each iteration. It is important to notice that task planning software is not mature and is quite experimental, usually you don’t get much feedback about syntax mistakes as you would get in programming languages such as c++ or python.

Without the first extension to the problem generator, we would not have been able to use the Mercury planner. The

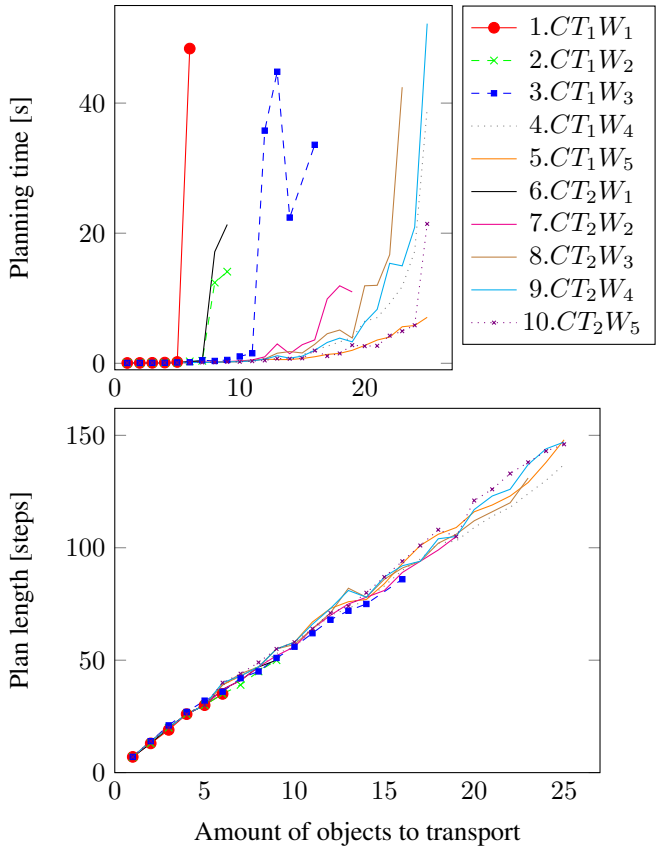

Figure 7: As problem increases in complexity, so does the planning time, 10 different parameters have been used.

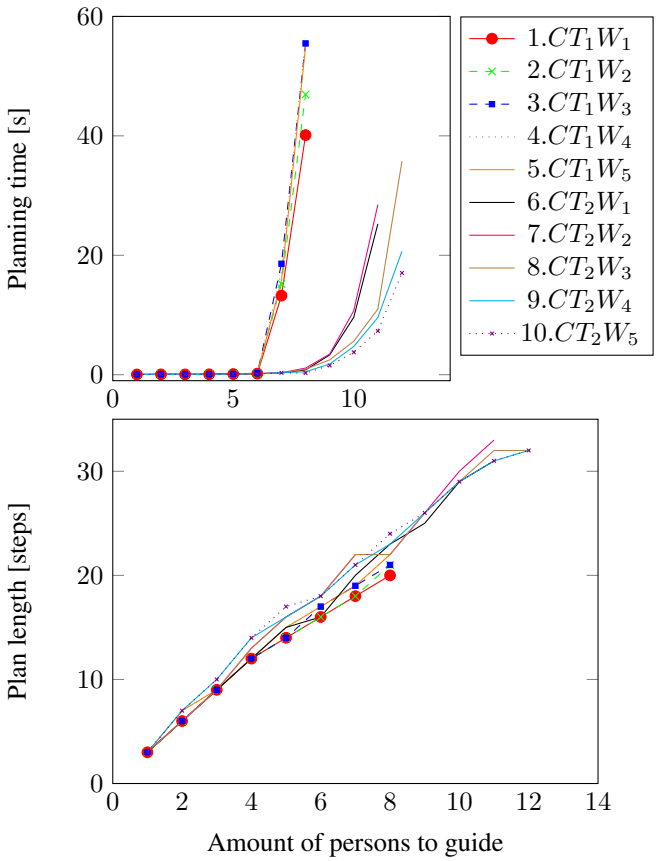

Figure 8: As problem increases in complexity, so does the planning time, 10 different parameters have been used.

automata-based execution layer allows for complex execution of a plan that was generated with various assumptions to simplify the planning part. The use of refactored action-based automatas in the execution layer helps to balance the various robot skills by offloading the complexity either to the planner or to the automata (design choice).

Although any PDDL planner can be easily integrated in our architecture, we provide with an interface for the IPC 2014 Mercury planner (Katz and Hoffmann 2014). The reason behind it, being it's top performance in the IPC 2014 transport domain which was similar to our industrial @Work domain.

8 Conclusions

In this work which we consider a success story in planning we have shown the modularity advantage of the individual contributions that were published in the ICAPS domain, that can be used to build the overall system out of diverse but compatible components.

The system has proven to be a working solution and led the team to win 3rd place in 2015 and 2nd place in 2016 in the RoboCup @Work international scientific competition (Kitano et al. 1997). With regard to @Home the system is still in experimental phase and has only being tested in local demonstrations in our lab. A video showing the global robot performance can be seen here[5].

The complexity analysis that we have done is helpful to know the scalability of the planner and to balance the complexity between domain operators and the automata. Additionally our benchmark experiments provide an intuition of the problem size that a classical planner can handle.

Both competitions domains include non-deterministic outcomes and incomplete information. The execution layer performed as expected and yielded impressive results[6] for the overall systems performance.

The representation of the execution layer as a state machine is an improvement over other implementations, e.g. ROSPlan[7], where it is difficult to analyze or change the replanning behavior. It is important to highlight that while this work uses some tools from ROSPlan (Knowledge Base and part of the PDDL problem generator) the framework is entirely our own work. The changes we have made to the ROSPlan problem generator were necessary to enable it to cope with cost information however the system we have created can handle any PDDL planner with or without cost informa-

[5] www.youtube.com/watch?v=7fvAQVNoKjo

[6] Our team obtained 3rd place in 2015 and 2nd place in 2016 in the RoboCup @Work international scientific competition.

[7] www.github.com/KCL-Planning/ROSPlan

tion.

One of the central execution issues is dealt with by coding actions as automatas.

Our experience shows that SMACH state machines meet our requirements and allows us to represent what is necessary to achieve the robustness that we have during the acting phase. This includes the action execution monitoring, handling failures that do not require replanning, updating the Knowledge Base (based on whether actions were perceived by the robots sensors to have succeeded or failed during the monitoring phase) thereby providing up-to-date information for the replanning process when it is needed.

Contributions of this work are: The integration of various different planning components into real robot systems with two use cases: Industrial and Domestic service robots. Domain models that work in real scenarios. Experiments on a state of the art planner on scalability and cost assignment that provide with valuable guidelines on what problem size can the planner solve (and in which time), and cost assignment strategy, regarding the numerical values that can be set.

This work is relevant for roboticists that want to add planning capabilities to their robot systems.

Acknowledgment

Bonn-Rhein-Sieg University, as I did part of this work during my master thesis work in there.

This work was supported by the FCT project [UID/EEA/50009/2013].

This work was supported by project [PTDC/EEI-SII/4698/2014].

References

Bischoff, R.; Huggenberger, U.; and Prassler, E. 2011. Kuka youbot-a mobile manipulator for research and education. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1–4. IEEE.

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *Journal of Artificial Intelligence Research* 47:697–740.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *ICAPS*, 333–341.

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

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, 294–301. IEEE.

Katz, M., and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; Osawa, E.; and Matsubara, H. 1997. Robocup: A challenge problem for ai. *AI magazine* 18(1):73.

Korf, R. E. 1990. Real-time heuristic search. *Artificial intelligence* 42(2-3):189–211.

Lima, O. 2016. Task planning, execution and monitoring for mobile manipulators in industrial domains, (master dissertation). retrieved from: https://github.com/oscar-lima/isr_planning/blob/kinetic/oscar_lima_master_thesis.pdf.

Niemueller, T.; Reuter, S.; Ewert, D.; Ferrein, A.; Jeschke, S.; and Lakemeyer, G. 2015. The carologistics approach to cope with the increased complexity and new challenges of the robocup logistics league 2015. In *Robot Soccer World Cup*, 47–59. Springer.

Niemueller, T.; Reuter, S.; and Ferrein, A. 2015. Fawkes for the robocup logistics league. In *Robot Soccer World Cup*, 365–373. Springer.

Nilsson, N. J. 1984. Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA.

Richter, S., and Westphal, M. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Wygant, R. M. 1989. Clipsa powerful development and delivery expert system tool. *Computers & Industrial Engineering* 17(1-4):546–549.