Integrating Mission and Task Planning in an Industrial Robotics Framework

Matthew Crosby,
Ronald P. A. Petrick
Department of Computer Science
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
{M.Crosby,R.Petrick}@hw.ac.uk

Francesco Rovida,
Volker Krüger
Robotics, Vision, and Machine Intelligence Lab
Aalborg University
Copenhagen 2450, Denmark
{francesco,vok}@m-tech.aau.dk

Abstract

This paper presents a framework developed for an industrial robotics system that utilises two different planning components. At a high level, a multi-robot mission planner interfaces with a fleet and environment manager and uses multiagent planning techniques to build mission assignments to be distributed to a robot fleet. On each robot, a task planner automatically converts the robot's world model and skill definitions into a planning problem which is then solved to find a sequence of actions that the robot should perform to complete its mission. This framework is demonstrated on an industrial kitting task in a real-world factory environment.

Introduction

Modern industrial robotics is characterised by a need for increased autonomy in factory environments. In particular, the current generation of factory robots often possesses low degrees of autonomous operation in non-static environments. The particular application for the work in this paper is the industrial *kitting* problem, where collections of parts are gathered from a factory 'supermarket' area and delivered to a production line. This paper introduces a robotics framework for solving the kitting problem that utilises planning at two different levels: at a multi-robot mission planning level, and at a single-robot task planning level.

One advantage of industrial robotics tasks is that the efficiency of a solution can often be improved by increasing the number of robots. However, planning domains often scales exponentially with the number of actors. This work therefore presents a multiagent mission planning algorithm that produces low makespan plans in problems traditional planning algorithms cannot solve, and that prior multiagent planning algorithms can only solve by reduction to a single agent.

At the robot level, task-level programming provides a way of simplifying the job of controlling a robot. In this paradigm, a human programmer specifies what the robot should do in terms of the high-level 'skills' and objects involved in the task. Skills are identified as the re-occurring actions that are needed to execute standard operating procedures (e.g., operations like *pick 'object'* or *place at 'location'*). Embedded within skill definitions are the sensing and motor operations that accomplish the goals of the skill as well as a set of condition checks that are made before and after execution. This paper shows how the condition checks can be automatically converted into planning actions, and how a planning problem can be built from the robot's world model and then solved to generate an ordered sequence of skills that will achieve the robot's mission.

The paper is organised as follows. First, the related work is considered and the complete system architecture of our solution is introduced. Next, the mission planning system is discussed and a multiagent planning algorithm is introduced for finding reduced makespan plans for the multirobot problem. The task planning system is then introduced, and it is shown how planning domains and problems can be automatically generated. The paper concludes with a set of experiments, some in simulation and some from a real factory environment, that demonstrate the complete system.

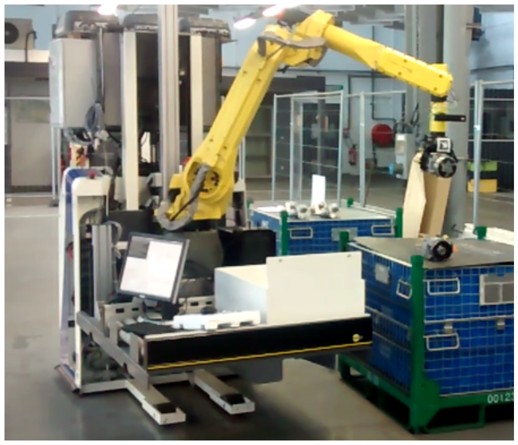


Figure 1: A robot operating in a factory environment using the integrated system. The robot is executing a six-step plan to place two parts in the white kitting box it is carrying.

Related Work

Planning has a long association with robotics, stretching all the way back to Shakey (Nilsson 1984) and Handey (Lozano-Pérez et al. 1989). Most modern autonomous robots follow a hybrid approach (Gat 1998; Ferrein and

Lakemeyer 2008; Bensalem and Gallien 2009; Magnenat 2010), with researchers focused on finding appropriate interfaces between the declarative descriptions needed for high-level reasoning and the procedural ones needed for low-level control. For example, in architectures like (Gat 1998), a deliberative layer is responsible for generating plans. The intermediate "sequencing" layer is responsible for choosing the plan that is appropriate for the current situation and the reactive layer executes the actions specified in the plan.

Robot task planning has also become an active research area recently, with approaches taken from areas such as sampling-based motion planning (Plaku and Hager 2010; Barry 2013), symbolic planning invoking motion-planning functions (Dornhege et al. 2009; Gravot, Cambon, and Alami 2005), and probabilistic pre-image back-chaining (Kaelbling and Lozano-Pérez 2013). ROSPlan (Cashmore et al. 2015), a Robot Operating System (ROS)-based tool for task planning provides a general framework for planning in robotics and has been successfully deployed on autonomous underwater vehicles. Our work adds the capability of automatically generating planning domains and problems from existing robot-level *skill* definitions: reusable robot-level control modules, which are a common technique for representing a robot's actions.

Knowledge representation also plays a fundamental role in the integration of planning in robotics. A prominent example is the KnowRob system (Tenorth and Beetz 2013; 2012), which combines knowledge representation and reasoning methods for acquiring and grounding knowledge in physical systems. KnowRob uses a semantic library which facilitates loading and accessing ontologies represented in the Web Ontology Language (OWL). A similar approach is presented in (Björkelund et al. 2012; Stenmark and Malec 2013; Bjørkelund and Edstrom 2011) as part of the Rosetta project, which focuses on how skills should be modelled for industrial assembly tasks. Another study can be found in (Huckaby 2014), who defined a precise taxonomy of skills. Our approach also uses an OWL ontology at the task planning level for the robot's world model and skill definitions.

Mission planning also plays an important role in our work, for assigning goals to individual robots in a robot fleet, by focusing on multiagent techniques that attempt to exploit the underlying structure inherent in multiagent domains (Brafman and Domshlak 2008). ADP (Crosby, Rovatsos, and Petrick 2013) and MAPR (Borrajo 2013) are existing multiagent planners that could solve the problems, but only with outputs where one robot performs all actions. For MAPR, the *load-balance* option could give results with lowered makespan, but this could not scale up to the full-sized problems. *Load-balance* is a goal assignment strategy that has similarities to the methods employed in this paper, designed to keep a good work-balance among the agents. Other multiagent planners such as MA-FD (Nissim and Brafman 2012) did not scale to the problems sizes used in our domain.

System Architecture

Figure 2 shows the system architecture. Arrows represent generic ROS services, except for communication between the logistics manager and the manufacturing execution systems (MES), which conforms to the factory's system interfaces. At the centre of the upper level is the logistics manager which is responsible for handling the complete environmental (world) model, including the status of each robot, and also for interfacing with the factory's manufacturing execution system (MES) in order to retrieve information about the current state of the factory environment and the orders to be filled. The logistics manager also interacts with the mission planner. When orders are received they are sent to the mission planner which returns mission assignments to be sent to individual robots in the fleet.

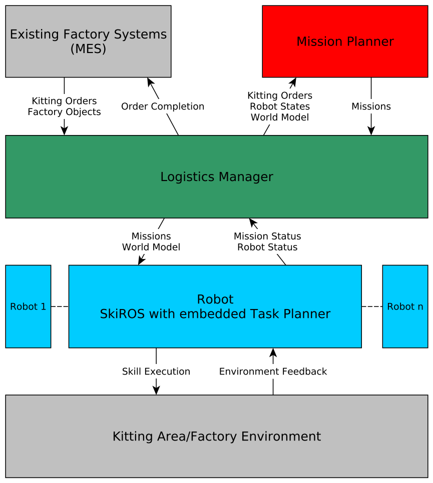

Existing Factory Systems (MES)
Mission Planner
Kitting Orders Factory Objects
Order Completion
Kitting Orders Robot States World Model
Missions
Logistics Manager
Missions World Model
Mission Status Robot Status
Robot 1
Robot SkiROS with embedded Task Planner
Robot n
Skill Execution
Environment Feedback
Kitting Area/Factory Environment

Figure 2: Overview of the system architecture.

At the centre of the lower level is the SkiROS system (Skills-ROS) (Rovida and Krüger 2015), with an instance running on each robot. Once a mission is sent to a robot, SkiROS invokes the task planner to find a sequence of skills whose execution should lead to the successful completion of the mission's goals. Skills contain robot execution code, sensing actions, and world model updates. The world model updates are accessible to the task planner and used to automatically create planning actions. During skill execution, the robot's internal world model is constantly updated and replanning is invoked in the case of execution failure.

Both the logistics manager and SkiROS have GUIs designed for easy access to the current system state, and for factory workers to program and update the system. The logistics manager's user interface can be used to configure the warehouse environment, trigger the mission planner to generate missions, send and receive missions and tasks, and to monitor the system. The SkiROS user interface can be used to add custom skill sequences, monitor and update the robot's world model, and initiate the execution of skills.

Mission Planning

The mission planner has the task of assigning missions (goal sets) to each robot in the fleet by communicating with the logistics planner which models world state information.

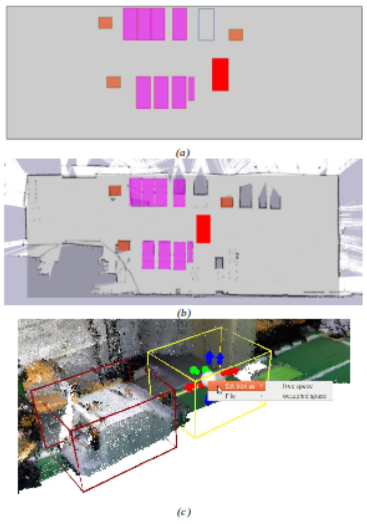


Figure 3: Logistics manager world model modes: (a) 2D CAD image, (b) 2D SLAM image, and (c) 3D point cloud.

Logistics Manager

The logistics manager forms the bridge between the robots, the mission planner, and the factory's MES elements (Crosby et al. 2016). It stores data about the working environment, including identifiers for all physical objects and their three dimensional models. Information is initially compiled from the MES, manual input by a kitting technician, and the states and capabilities of the robots as defined in SkiROS (see below). The information communicated by the logistics manager mainly concerns the physical objects that are located in the environment. In our application, this equates to shelves, small and large boxes, conveyors, kits, parts, and packaging elements.

The logistics manager supports three methods for inputting and storing information about the environment (Figure 3). First, a 2D image is created by a CAD application to visually specify the location and orientation of each object in the logistics supermarket (Figure 3a). The red rectangle represents the object being placed whilst the remaining rectangles represent objects already in place. Second, a 2D image created by the robot through its Simultaneous Localization and Mapping (SLAM) functionality can be used to visually specify an object's location and orientation (Figure 3b). Finally, a 3D image of the supermarket is also created to visually specify object locations and orientations (Figure 3c).

Mission Planner

The mission planner takes goal and world state information from the logistics manager and returns goal assignments for each robot in the fleet. To achieve this, it solves a multiagent planning problem and uses the goal decomposition of the returned plan as the mission assignment. The planning domain here is created manually allowing it to focus on only the salient information for multirobot goal assignment contained in the logistics manager's detailed world model.

Algorithm 1: Heuristic Value Calculation of ADBP.

```
Input  : State S, Goals G
Output: $h\_max$, $h\_square$, or $h\_total$
1  Relaxed Planning Graph Generation (full)
2  if $Max\ layer > 0$ then
3      Calculate Subgoals
       $G \leftarrow G \cup subgoals$
4  foreach $agent\ i$ do
5      foreach $Goal\ g \in G$ do
6          if $h\_add(g, i) > 0$ then
7              ExtractRelaxedPlan(g, i)
8      $hff(i) \leftarrow RelaxedPlan(i).cost$
9  $h\_max = max_i$ hff($i$)
10 $h\_square = \sum_i$ hff($i$)$^2$
11 $h\_total = \sum_i$ hff($i$)
```

The CAD models and exact locations and orientations of objects are ignored. Using the hand-crafted planning domain, a planning problem instance is generated automatically from the logistics manager's world model (which also contains the skill sets for each robot). The details of this generation are fairly trivial once a suitable extracted planning domain is decided upon and will not be covered here.

Table 1 shows the results of testing several planners on problem instances of different sizes. The single-agent planners could not scale to the larger problems and, while the multiagent planner ADP could, it could only return plans with a single robot performing all the actions. The only reason it could cope with the larger problems was that it effectively ignored the extra complexity introduced by multiple robots. Its decomposition algorithm splits the problem into one for each robot and then, because of the way the heuristic is designed, finds that the first robot can complete the entire problem and never needs to consider any of the other robots again. MAPR set to rest-achievable is a similar algorithm and has the same behaviour, while on other settings it cannot scale past the single-agent planners. It is therefore not useful for our purpose as not only do we need to find a solution in a reasonable timeframe, but the solution must also be fairly evenly balanced amongst the robots.

Our solution is a new multiagent planning algorithm called ADBP that is implemented as a derived class of the Heuristic class in Fast Downward (Helmert 2006). ADBP modifies the heuristic function of ADP to return results that respect each agent's possible contribution for each state in the search space. While slowing down planning times, this leads to the planner returning lower makespan plans in a reasonable time frame for our application, and is the only solution we know of that works for our problem with larger numbers of robots. The makespan of the returned plans is calculated as the maximum number of actions that a single agent must perform.

ADBP uses the agent decomposition calculated by ADP (Crosby 2014). The heuristic value calculation is shown in Algorithm 1. Relaxed Planning Graph (RPG) Generation is based on that introduced by FF (Hoffmann and Nebel 2001),

Problem Number	No. of Robots	No. of Goals	POPF2 time (s)	POPF2 cost	FF time (s)	FF cost	LAMA time (s)	LAMA cost	ADP time (s)	ADP cost
1	2	4	0.52	29	0.01	30	0	24	0.01	24
2	2	4	–	–	0.34	54	0.01	54	0	54
3	2	4	–	–	0.01	74	0.01	74	0.01	74
4	2	6	–	–	7.11	132	0.04	111	0.02	132
5	4	6	–	–	–	–	–	–	0.02	111
6	4	8	–	–	–	–	–	–	0.03	148
7	6	8	–	–	–	–	–	–	0.05	148
8	6	10	–	–	–	–	–	–	0.06	185
9	8	10	–	–	–	–	–	–	0.08	185
10	10	10	–	–	–	–	–	–	0.11	185

Table 1: Table showing the performance of planners on mission planning domains when the makespan of the output plans was ignored. A '–' means that the planner did not return a plan within 300 seconds.

but modified for the multiagent case. Each agent generates its own RPG based on the problem reduced to only their variables and environment variables. After this, the union of the final state of each agent is then used to create another layer of RPGs. This process repeats until no further propositions can be added by any agent and guarantees that the full relaxed plan space is explored.

If more than one layer of RPGs have been generated, then subgoals are calculated. Any time a goal proposition appears in a layer above the first, this means that it cannot be reached by an agent on its own. Therefore, plan extraction (Hoffmann and Nebel 2001) is used to find which proposition is utilised from the previous layer. All necessary propositions from the previous layers are added to the set of subgoals.

In the final step of the heuristic calculation, each agent creates its own relaxed plan to every goal in the goal set (including subgoals) that it can achieve. This is simply the goals that appear in the first layer of the multiagent RPG for each agent. The value $hff(i)$ is the cost of agent i's relaxed plan. This only counts each action once (even if it is used to reach multiple goal propositions). Any action that appears in a relaxed plan (of any agent) that is also applicable in the current state is set as a preferred operator. With this information, there are three different versions of ADBP based on the way the individual heuristic values calculated by each agent at each state are combined:

ADBP-max uses the value h_max which is the maximum hff value of all the agents. This discards a lot of the information that has been calculated, but is as close as possible to the estimated makespan of the plan from the current state. As ADBP is primarily concerned with minimising makespan, this is a natural heuristic to investigate. The downsides are that it discards a lot of potentially useful information and has little concern for how close the goal state actually is. For example, 5 agents with $hff = 9$ return a worse heuristic value than if four have $hff = 0$ and one has $hff = 10$.

ADBP-total takes the other extreme and uses the sum of each agent's hff values. This is much further from the makespan heuristic estimate but encodes more information about the distance from the state to the goal. It should be noted that this is different than the single-agent FF value for the state because goals are repeated by all agents that can achieve them and subgoals are included in the calculation. The downside of this calculation is that it does not take makespan into account at all (beyond the fact the value is calculated over multiple agents).

ADBP-square attempts to mediate between the two extremes and uses the sum of squares of the hff values of the agents. The idea is to use all the information available whilst taking the variance between the agent values into account.

The difference between ADBP and ADP's heuristic calculation is massive. Once ADP chooses an agent to perform the next goal set, it restricts the planning problem to that agent and then never considers any of the others until it either gets to a dead end or completes all the current goals. In ADBP, each agent in involved in the heuristic value calculation for every state in the search space.

Task Planning

Task planning is performed by each robot using a domain and problem automatically generated by the robot's onboard world model and skills definitions, with the goals sent to it from the mission planner (mediated by the logistics manager). The world model and skills information are stored and managed by SkiROS (Skills-ROS).

SkiROS

A SkiROS component is located on each robot and controls robot-level execution and ontology management. It also communicates status updates and receives mission assignments from the logistics manager. SkiROS is organised into four layers, each of which is represented by a manager. The lowest layer is the *device manager*, which loads proxies (drivers which conform to a standard interface) and presents standard interfaces for similar devices (e.g., gripper, arm, camera, etc.). The second layer contains the *primitive manager* which contains motion primitives and services. The third layer, the *skill manager*, loads skills (see below) and provides interfaces to the layer above. It also registers the robot subsystem with the world model, specifying the hardware, available modules, and available skills. Finally, the fourth layer of the architecture is the *task manager* which monitors the presence of subsystems via the world model and acts as a general coordinator. The task manager contains

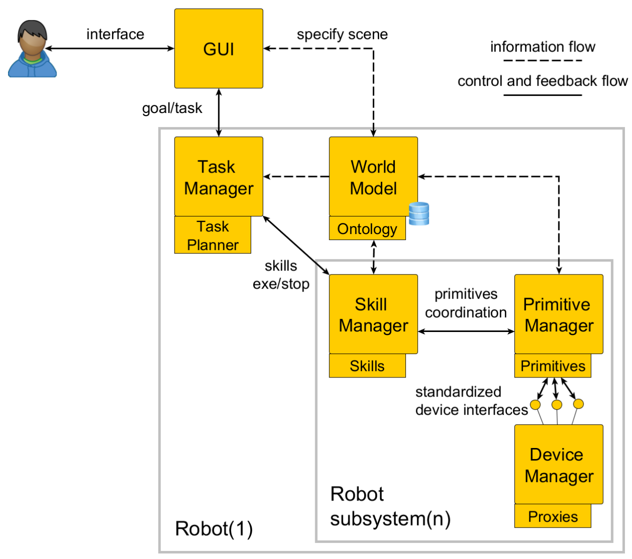


Figure 4: An overview of the SkiROS architecture of which the task planner is a part.

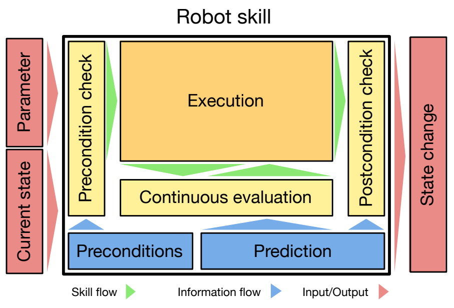


Figure 5: The conceptual model of a SkiROS skill.

the task planner and is the interface for external systems, designed to be connected to a GUI or the factory's MES.

A central concept in SkiROS is the idea of *skills*, which can be thought of as modelling self-contained, re-occurring operations that a robot might perform. For example, a system might include calibration skills, manipulation skills for operations like picking and placing, as well as driving skills for mobile robots. The conceptual model of a robot skill is shown in Figure 5. A skill takes as input a set of parameters and the current world state and outputs a set of state changes. A skill contains interactions with the world model in the form of either pre-execution or post-execution checks and updates. For example, a pre-execution check for a pick skill might be that the item to be picked must be visible to a camera, and a post-execution check might be that the picked item is in the gripper. These become preconditions and postconditions when translated to a planning domain.

SkiROS's *world model* acts as a knowledge integration framework and uses the web ontology language OWL. The world state is partially defined by a human operator in the ontology, partially abstracted by the robot using perception,

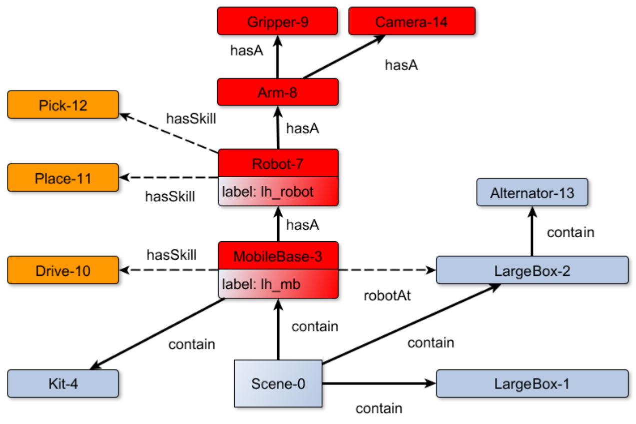


Figure 6: A simplified world model instance, presenting some physical objects (blue) and abstract objects (orange). All physical objects are connected by one spatial relation ("hasA" or "contain") in a scene graph structure.

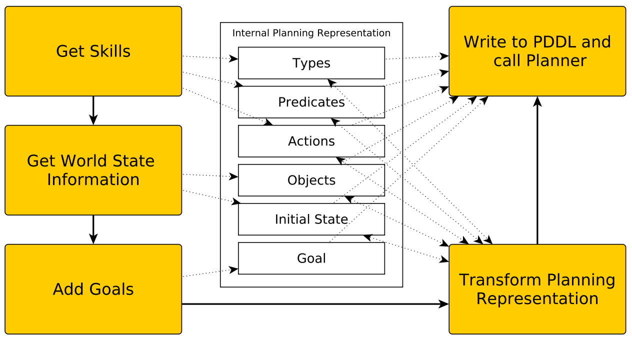
Drive(MobileBase, Container) :
  add: RobotAt(Container, MobileBase)

Pick(Gripper, Object, Container) :
  pre: empty(Gripper)
  pre: robotAt(Container, Robot)
  pre: objectAt(Container, Object)
  del: empty(Gripper)
  add: contains(Gripper, Manipulatable)

Figure 7: Part of the skill definitions in the SkiROS world model accessible to the task planner.

and completed with the procedural knowledge embedded in the skills and primitives. It is originally populated with the robot skills knowledge, robot-specific knowledge such as grasp poses or parameter settings, and with the world model provided by the logistics manager. Each skill manager in the system is responsible for keeping the world model updated with its subsystem information (e.g., hardware, available primitives, skill state, etc.). Figure 6 shows an example instance of a world model for our application domain.

Skills are modelled to be object centric, and are therefore parameterised solely using elements that are instantiated in the scene. If a new skill is added to the system, then it must interact with the same (perhaps updated) world model and same object representations. Figure 7 shows the part of the definition of the *Drive* and *Pick* skills accessible to the task planner. The Properties component shows the different types of parameters that the skill can be instantiated with. The preconditions and postconditions can be chosen from a library of predefined classes derived from properties that can be set or verified in the ontology. The skill also contains an executable part that embeds the procedural knowledge about the action in the real world, and which maps to a corresponding modification in the world model.


Get Skills
Get World State Information
Add Goals
Internal Planning Representation
Types
Predicates
Actions
Objects
Initial State
Goal
Write to PDDL and call Planner
Transform Planning Representation
Figure 8: Overview of the task planning process and the creation of its internal planning representation. Dotted arrows towards the representation represent data modification while dotted arrows in the other direction represent read access.

Task Planner

The task planner has three main functions: it creates a PDDL representation of the skills, current state and goals; it calls an external planner to attempt to find a plan for the current goals; and, if a plan is found, it returns a sequence of skills to the task manager. The task planner creates a planning domain and problem written in PDDL 1.2 with only the `types` requirement. This means that the output is suitable for use with almost all modern planning systems. In what follows we use the standard definition of STRIPS-like planning actions (Fikes and Nilsson 1971) with *pre*, *add*, and *del* representing the preconditions, add effects and delete effects of an action, respectively.

Task Planner Operation

The task planner uses skill information about the parameters, the current state of the world model required for the skill to be execute, and the expected state change caused by the successful execution of the skill. The skill properties become the parameters for the action and the preconditions and postconditions become the members of pre and $del \cup add$, respectively. An overview of the task planner's algorithm for generating the planning problem is shown in Figure 8. This process is invoked (with the internal planning representation reset) every time the task manager requires a plan. The central part of the figure shows the task planning library which contains all the necessary structures to create a PDDL planning problem out of the current world state, skills and goals.

The code blocks on the left hand side of the figure deal with naïve translations of world model information into the structures used in planning. This process is relatively straightforward given the design of the skills. The right hand side involves transforming the skills and world model information to ensure a consistent planning problem.

Initial Planning Representation The process of creating the initial planning representation is given in Algorithm 2 and involves three main steps. The first step is to parse the skills that exist in the world model. Only types or predicates parsed from the skills will be required to solve the planning

Algorithm 2: Planning Domain Creation

```
Input  : SkiROS World Model ($wm$), Goals ($goal$)
Output: Initial Planning Representation
// Parse Skills
1  foreach Skill s : wm do
2      types.addAllNewTypes(s)
3      predicates.addAllNewPredicates(s)
4      actions.addNewAction(s)
// Add Goal State
5  foreach Goal g : goal do
6      goals.add(g);
// Parse World Model State
7  foreach Predicate p: predicates do
8      initState.addAllTrueGroundings(p, wm)
9      objects.addAllNewObjects(p,wm)
```

Algorithm 3: Planning Domain Refinement

```
Input  : Initial Planning Representation
Output: Final Planning Representation
// Add Capabilities
1  foreach Action a : actions do
2      predicates.add(can_a ?robot)
3      a.pre.add(can_a ?robot)
4      foreach Robot r : hasSkill(a, r) do
5          initState.add(can_a r)
// Spatial Relation Constraints
6  foreach Action $a$ do
7      foreach Spatial Relation $S(o, s) \in a.add$ do
8          if $\nexists S \in a.pre$ AND $\nexists S \in a.del$ then
9              $s.params.add(x, s.type)$
10             $s.pre.add(S(o, x))$
11             $s.del.add(S(o, x))$
12         else if $\nexists S \in a.pre$ then
13             $s.pre.add(S(o, s))$
14         else if $\nexists S \in a.del$ then
15             $s.del.add(S(o, s))$
```

problem (it would be impossible to change the truth value of any predicate that does not appear in the effect of an action, or require the truth of any predicate that does not appear in the precondition of an action). Parsing a skill involves an almost direct translation from *Properties* to parameters, *hasPreCondition* to preconditions and *hasPostCondition* to the *add* and *del* effects. Skill parsing adds all required types to the planning representation.

The second step instantiates the goal returning an error if it contains a predicate that has not yet appeared. The final step obtains initial state of the planning problem from the world model. This process creates both the initial state, and the list of objects in the domain. The process iterates over all the predicates added when parsing the skills and queries the world model (through the interface available in SkiROS) to find all ground instances of the predicates that are true.

Transforming the Planning Problem The right hand side of Figure 8 encodes implicit properties of the world model and system. The first part makes sure that skills are only usable by the correct elements, by querying the **hasSkill** relation from the world model. For each action, a new invariant

```
(:action drive
   :param (?R - Agent ?T - Location
     * ?preT - Location)
   :pre (and
     * (can_drive ?R)
     * (RobotAtLocation ?R ?preT))
   :eff (and
     * (not (RobotAtLocation ?R ?preT))
       (RobotAtLocation ?R ?T)))
```

Figure 9: The Drive skill after translation to PDDL. The asterisked (*) lines are added by the translation.

predicate `(can_a ?robot)` is added to the planning description. An extra precondition `(can_a ?robot)` is also added to each action so that it can only be instantiated by the correct robots. If the `Robot` parameter is not in the skill definition then it is added to the action parameters (and removed before returning a plan).

The second part adds any preconditions and delete effects necessary to maintain the tree structure of the spatial relations in the world model. SkiROS contains methods for internally updating its world model that need to be modelled explicitly in the planning domain. For instance (see Figure 7) the Drive skill only contains a single predicate which specifies the new location of the robot. The `drive` action must be modified so that `robotAt` is true of only one grounding for the robot performing the Drive skill, so the old location must be added as a precondition and delete effect of the action.

Figure 9 shows the skills from Figure 7 after translation to PDDL. In terms of implementation, the parameter added to the Drive skill is removed when returning the parameterised skill to the task manager. The translation adds three new preconditions and two new delete effects over the two actions. Performing an action created by the task planner on a problem whose spatial relations form a tree will result in a state in which the spatial relations still form a tree. This is because any deletion of a spatial relation property inserts it elsewhere with the same object (therefore moving the whole subtree), and every addition has a corresponding deletion.

Once the translation is complete, the planning problem is written to domain and problem files in PDDL for use with an external planner. The planner's output (a sequence of instantiated actions) is parsed and converted back to parameterised skills to be sent to the task manager for robot execution.

Experiments

Three separate experiments were performed to test the system. The first experiment involved offline testing of the mission planner to determine which planning algorithm worked best for this domain. As no existing algorithm could solve the largest problem and produce a multiagent solution, a new algorithm was developed for this task. The second set of experiments involved offline testing of the task planner in simulation, and its integration with the SkiROS system. These tests showed successful automatic generation and solution finding for the skills used in the application domain. Finally, on-site testing with a physical robot in a real factory environment explored how the planning system performed as part of the complete system.

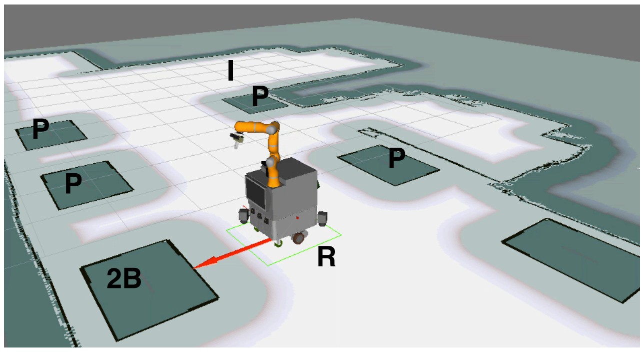


Figure 10: A visualisation in rviz of the simulated environment used for the offline task planner experiments.

Offline Mission Planner Testing Table 2 shows the results for the different versions of ADP. While ADBP takes considerably longer than ADP, it can find properly multiagent plans with the output split between multiple robots. These plans are found within a reasonable time frame for the system. Of the other multiagent planning algorithms tested that can return lower makespan plans, MA-FD solves only problem 1, and MAPR on load-balance setting can solve only up to problem 4. The results for ADBP-max are clearly also not good enough for it to be usable in our application. However, they have an interesting property that the returned plans contain actions that are almost always interleaved between the different robots. ADBP-squ and total on the other hand return plans with agents performing many actions in a row. This does not affect the mission assignments performed in our application, but may of interest for other work.

Offline Task Planner Testing Figure 10 depicts the simulated environment. The experiments used a simulated version of a mobile manipulator containing an articulated robot arm mounted on a mobile platform. For the simulation, the hardware drivers are substituted with simulated versions using the same ROS interfaces. An inherent part of the skills is that they perform the necessary sensing operations to complete the skill, these are set to automatically return success in the simulation.

For the experiments, the task planner used the Fast-Downward planner (Helmert 2006), with A* search and the landmark-cut heuristic. Since the planning problems created by the translation process do not test the limits of the external planner there was no benefit in comparing different planning approaches. Instead, any state-of-the-art planner that supports the required features could be used in its place.

The first experiment did not include the skill manager of the mobile base. The robot was placed in front of a pallet with two smaller boxes containing thermal shields and engine supports. The pick and place skills were executed correctly when the goal contained only those two parts. If other parts (in different locations) are added to the goal, then the task planner returns (correctly) that no plan can be found.

Problem Number	No. of Robots	No. of Goals	Time (seconds)				Plan Length				Makespan			
			orig	max	squ	total	orig	max	squ	total	orig	max	squ	total
1	2	4	0.01	0.01	0.01	0.01	24	24	24	24	24	12	12	12
2	2	4	0	0.01	0.01	0.01	54	54	54	54	54	27	27	27
3	2	4	0.01	0.03	0.02	0.02	74	74	74	74	74	37	37	37
4	2	6	0.02	0.46	13.11	13.07	132	132	153	154	132	66	95	95
5	4	6	0.02	3.91	0.19	0.19	111	174	111	111	111	58	37	37
6	4	8	0.03	–	0.35	0.35	148	–	147	147	148	–	37	37
7	6	8	0.05	–	0.96	0.95	148	–	148	148	148	–	37	37
8	6	10	0.06	–	1.59	1.6	185	–	185	185	185	–	37	37
9	8	10	0.08	–	3.42	3.41	185	–	185	185	185	–	37	37
10	10	10	0.11	–	6.33	6.31	185	–	185	185	185	–	74	74

Table 2: Table showing the performance of the different versions of ADP on the testing domains. Key: orig is the original ADP algorithm. max is ADBP-max, squ is ADBP-square, and total is ADBP-total.

```
drive mobbase-2 loc-1 lbox-10
pick lbox-10 gripper-6 t_shield robot-3
drive mobbase-2 lbox-10 lbox-9
place grip-6 t_shield celld-19 kit-15 robot-3
pick lbox-9 gripper-6 starter robot-3
place grip-6 starter cellb-17 kit-15 robot-3
```

Figure 11: An example plan from testing in simulation.

Order	Attempts	Planning Success	Exec Time (s)
3 parts	15	15	310
4 parts	22	22	380
5 parts	2	2	-

Table 3: Results from testing the complete system in a factory environment.

The extraction of the planning domain, and the planning itself, is completed in 0.6*s*, resulting in a plan with four skills that is executed in 78*s*.[1]

In the second experiment, the skill manager of the mobile base is added, which adds the drive skill, allowing the robot to navigate to different containers. Domain extraction correctly registers the additional drive skill from the extra skill manager, and includes this in the planning domain. The goal can now include any part, and a plan is found that will drive the robot to the correct pallet before picking the part and placing it in the kit. Figure 11 shows the plan found for the goal of filling a kit with two parts: a thermal shield and a starter. If a goal for the robot to finish at a particular location is included, then this is achieved as well. When the goal request is for a complete kit with six parts, the PDDL extraction and planning takes 0.9*s*. The resulting plan with 18 skills is executed correctly in 371*s*.

Factory Testing The complete system was deployed in a real-world environment at a PSA Peugeot Citroën factory (as shown in Figure 1). The results of the experimental testing are presented in Table 3. The tests were carried out using simulated kitting orders as input. The planner used in this system was Temporal FastDownward (Eyerich, Mattmller, and Röger 2009). As we had access to only one robot in the testing factory, this could not test the multi-agent planning component of the system.

[1]Planning, motion planning, simulation, SkiROS and visualisation ran on a 2011 laptop with an i7@2.7GHz processor.

The planning success rate shows the success of the system up to the point of creation of the robot's task-level plan. In all cases, the mission planner successfully found mission assignments, and the task planner found correct skill sequences to complete the missions. The main failure points in the system, leading to no complete missions in the 5 parts case, came from robot execution code for individual skills, which is still under development. In some failure cases, replanning was able to recover from a failure in execution. However, execution failure often led to the robot being manually stopped, or to inconsistent world states where replanning was not effective.

Overall, the results show that the system architecture is sound, but the execution code for the skills needs to be improved. Due to the modular nature of the system design, the skill execution code can be updated without affecting the automated planning in the system.

Conclusions

This paper presented a fully implemented software framework for deploying autonomous robot systems in an industrial setting, which used two different planning systems. A task planner used a robot skills model to automatically generate a planning problem to be solved from the robots' world model. The system also included a high-level mission planner and a new multiagent planning algorithm that could find mission assignments for a larger robot fleet. The resulting system has been shown to operate successfully in a simulated factory environment, and the planning systems have been successfully deployed in a real factory setting.

Acknowledgements

The research leading to these results has received funding from the European Union's 7th Framework Programme under grant no. 610917 (STAMINA, `stamina-robot.eu`).

References

Barry, J. L. 2013. *Manipulation with Diverse Actions*. Ph.D. Dissertation, Massachusetts Institute of Technology, USA.

Bensalem, S., and Gallien, M. 2009. Toward a more dependable software architecture for autonomous robots. *IEEE Robotics and Automation Magazine* 1–11.

Bjørkelund, A., and Edstrom, L. 2011. On the integration of skilled robot motions for productivity in manufacturing. In *IEEE International Symposium on Assembly in Manufacturing*.

Björkelund, A.; Malec, J.; Nilsson, K.; Nugues, P.; and Bruyninckx, H. 2012. Knowledge for Intelligent Industrial Robots. In *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*.

Borrajo, D. 2013. Plan sharing for multi-agent planning. In *Proceedings of the ICAPS Workshop on Distributed and Multi-Agent Planning*, 57–65.

Brafman, R. I., and Domshlak, C. 2008. From One to Many: Planning for Loosely Coupled Multi-Agent Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 28–35.

Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtòs, N.; and Carreras, M. 2015. Rosplan: Planning in the robot operating system. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS)*, 333–341.

Crosby, M.; Petrick, R. P. A.; Toscano, C.; Dias, R. C.; Rovida, F.; and Krüger, V. 2016. Integrating mission, logistics, and task planning for skills-based robot control in industrial kitting applications. In *Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2016)*.

Crosby, M.; Rovatsos, M.; and Petrick, R. P. A. 2013. Automated Agent Decomposition for Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 46–54.

Crosby, M. 2014. *Multiagent Classical Planning*. Ph.d. thesis, University of Edinburgh.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE Intl. Workshop on Safety, Security & Rescue Robotics*, 1–6.

Eyerich, P.; Mattmller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *International Conference on Automated Planning and Scheduling*.

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems* 56(11).

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.

Gat, E. 1998. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press.

Gravot, F.; Cambon, S.; and Alami, R. 2005. aSyMov: a planner that deals with intricate symbolic and geometric problems. *Robotics Research* 100–110.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Huckaby, J. 2014. *Knowledge Transfer in Robot Manipulation Tasks*. PhD thesis, Georgia Institute of Technology, USA.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research* 32(9–10):1194–1227.

Lozano-Pérez, T.; Jones, J.; Mazer, E.; and O'Donnell, P. A. 1989. Task-level planning of pick-and-place robot motions. *IEEE Computer* 22:21–29.

Magnenat, S. 2010. *Software integration in mobile robotics, a science to scale up machine intelligence*. PhD thesis, École polytechnique fédérale de Lausanne, Switzerland.

Nilsson, N. J. 1984. Shakey the robot. Technical Report 323, AI Center, SRI International.

Nissim, R., and Brafman, R. I. 2012. Multi-agent A* for Parallel and Distributed Systems. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1265–1266.

Plaku, E., and Hager, G. D. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *Int. Conference on Robotics and Automation*, 5002–5008.

Rovida, F., and Krüger, V. 2015. Design and development of a software architecture for autonomous mobile manipulators in industrial environments. In *2015 IEEE International Conference on Industrial Technology (ICIT)*.

Stenmark, M., and Malec, J. 2013. Knowledge-based industrial robotics. *Scandinavian Conference on Artificial Intelligence*.

Tenorth, M., and Beetz, M. A. 2012. Knowledge Processing for Autonomous Robot Control. *Proceedings of the AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*.

Tenorth, M., and Beetz, M. 2013. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research* 32(5):566–590.