Using a Robot Workspace Description for Planning On-Orbit-Servicing Tasks on Modular Satellites

Steffen W. Ruehl, Jan Oberlaender, Arne Roennau Ruediger Dillmann

FZI Research Center for Information Technology
Haid-und-Neu-Str. 10–14, 76131 Karlsruhe, Germany
e-mail: {ruehl,oberlaen,roennau,dillmann}@fzi.de

Abstract

Interest in modular satellites has increased recently. In this paper, we present our approach for planing an On-Orbit-Servicing (OOS) task for a robotic servicer. One key challenge is the gap between the symbolic planing of action sequences and their execution in the continuous space. To bridge this gap, we propose a sampling based workspace description. It is queried by an Hierarchical Task Network Planer (HTN) which generates a symbolic plan for the servicing task. We present heuristics and decompositions for the HTN planer for the task of planning OOS tasks. An experimental evaluation is presented.

1 Introduction

Interest in modular satellites has increased recently due to three important advantages modular satellites provide . First, modularization enables On-Orbit-Servicing (OOS). A satellite consisting of a system of modules can be repaired by replacing defective modules on orbit, leading to longer satellite lifetimes. Second, standard modules can be mass-produced, leading to lower component prices. Third, self-contained modules can simplify the process of designing a satellite. Software assistance for the designer can simplify the process further.

The first point in this list, longer satellite lifetime, is the key benefit of modular satellites. It enables savings in launch costs and reduces space debris. In order to exploit this advantage, on-orbit robotic servicing is a key challenge. Sending humans to space is too expensive for this task.

A robotic servicer requires some degree of autonomy since communication is not feasible on every position in the orbit. The servicer has to adapt its action to a given task. Adaption includes the selection of a suitable action sequence, manipulator trajectories and servicer positions. Solutions for kinematic and path planning problems exist in the state of the art in AI and robotics. Anyhow, addressing those problems independently does not add up to a consistent plan. Instead, they need to be integrated closely.

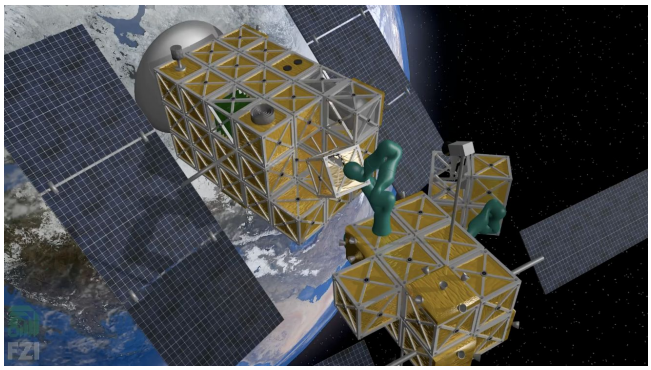

Figure 1. Scenario for on orbit servicing

Constraints of the manipulation tasks require planning of new actions beyond adapting the motion of given robot actions. For example, we cannot tolerate free floating satellite modules. Such problems can be solved by symbolic planning.

We use a Hierarchical Task Network (HTN) planner to generate a symbolic plan for the servicing task. Fig. 1 shows a modular satellite in such a task. Although symbolic task planning algorithms are well understood in simple domains such as the block world, real-world manipulation applications are still rare. One key reason for this is the fact that the discrete symbolic model does not map easily to the continuous would in which we live. In this paper we will present a workspace model for a manipulator to bridge this gap between pure symbolic task planning domains and continuous world models with a focus on planning on-orbit-servicing tasks for a manipulator.

The paper is structured as follows: Section 2 provides an overview of related work with respect to the integration of symbolic planing and motion planing robotics. It also deals with related work in the field of workspace descriptions. In Section 3, the problem domain of On-Orbit-Servicing is described. Section 4 presents on overview of the proposed system. Section 5 presents the workspace description. In Section 6 we describe our approach for symbolic planing using a HTN planer and the integration of the workspace description. First results are presented in section 7.

2 Related Work

Approaches exists, which overcome the limitations of pure symbolic planing and integrate methods from the domains of robot manipulation planing, like path or grasp planing. Those methods are able to exploit geometric models within a symbolic planner.

Choi and Amir [2009] use a motion planner to sample pairs of robot configurations, which are connected by a collision free path. Such pairs are used as action symbols within a symbolic planner. The symbols are interpreted as executable pick-and-place actions. A similar approach is taken by Cambon et al. [2004] for the aSyMov planner.

While the previously described methods are based on geometric planners which are extended by a symbolic reasoning method, the following methods approach the problem from the opposite direction. Dornhege et al. [2010] extends a symbolic planner by "semantic attachments". They enable the symbolic planner to send requests to a motion planner. Based on its result, an action is feasible in a situation or cannot be executed. Karlsson et al. [2012] present the integration of a rapidly exploring random tree (RRT) planner into a hierarchical task networks (HTN) planner.

An alternative approach is presented in Kaelbling and Lozano-Perez [2010]. Their system uses an HTN planner. It is modified by an idea, the authors call "planning in the now": Actions are executed, once a leave is inserted into the HTN. The system has to actually undo actions for backtracking. Stilman et al. [2007] present the planing of manipulation actions in variable scenes. A trajectory for a task is planed in an obstacle free space by RRT planing. The required free space is calculated. Objects inside the free-space are obstacles and must be removed, using the approach recursively. Geometric heuristics are applied to identify objects which may be moved.

A method for the description of a robot's workspace is described by Zacharias et al. [2007]. They use inverse kinematics to sample he workspace in a backward approach. This leads to the *reachability map* (previously and especially in the work cited referred to as *capability map*). The Cartesian workspace is sampled into equally sized cubes. A sphere is fitted into each cube. On its surface N equally distributed points are generated. For each point a set of frames with the z axis pointing towards the center of the sphere is created. The remaining degree of freedom rotating around that vector is sampled equally distributed. Inverse kinematics are used to find configurations resulting in those Tool Center Point (TCP) frames. If solutions are found for a frame, it is reachable. A point on the sphere is reachable if there is a frame which was created from that point and which is reachable. The resulting reachability spheres are visualized in Fig. 2. The reachability index is the ratio of the number of points on the sphere and the number of points on the sphere which are reachable in percent. In Zacharias et al. [2009], the reachability map is be used to find kinematically reachable approach position for object grasping.

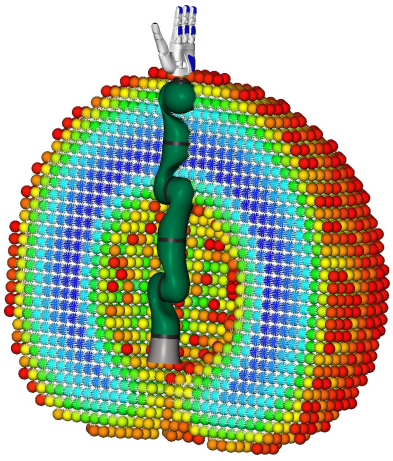

Figure 2. Visualization of the reachability of a Kuka Light Weight Arm (LWR). Blue spheres indicate poses with a very high reachability index, while red spheres have a low rating.

Rühl et al. [2011] extend the approach to consider objects and applicable grasps. Therefore the placement of objects is restricted to a plane. Their orientation is defined by stable planes, on which they have contact with a carrying object. Different orientations around the z axis are sampled. An example of a Graspability Map is shown in Fig. 3.

3 On-Orbit-Servicing for modular satellites

As outlined in the introduction, the goal of On-Orbit Servicing is to repair or modify satellites. In order to enable this, a modular building block concept has been presented in Weise et al. [2012]. A rendering of satellite made up from modules of the concept is depicted in Fig. 4. It consists of equally sized cubes. The cubes are arranged in a 3D grid. A cube has an interface on each face, which allows coupling with its neighbors.

The interfaces consists of the following components:

- *Docking Mechanism* An androgynous docking mechanism is present. It allows to rigidly connect the interface to any other interface. Rotation around the connecting axis is possible in 90° steps; the cubes have to be aligned with the grid when connected. The docking device is actuated by an electrical motor.

- *Electrical interface* Electrical connectors enable power transmission and distribution.

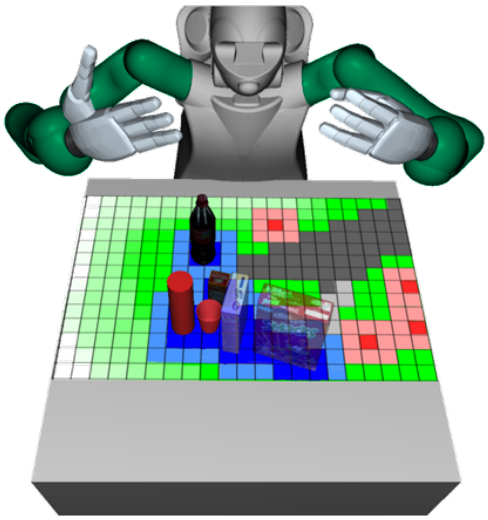

Figure 3. Visualisation of the Graspability of a quader shaped object projected onto a table. Green shades indicate the graspability, different colors show blocked poses for a planning task.

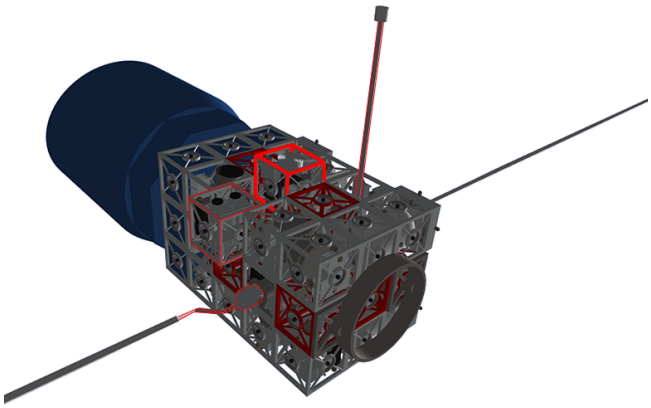

Figure 4. Visualization of a servicing task for a modular satellite. The bold red marked module is to be exchanged. In order to grasp that module, the thin red marked module has to be removed temporarily.

- *Data interface* The data interface enables a data bus between the satellite modules. It is used for control and user data.

- *Thermal interface* The thermal interface is required for the thermal management between the modules.

Payload such as cameras or antennas may occupy faces of the cube and create restrictions on its positioning. A module contains control electronics and room for payload. More details on the modular structure of the client can be found in Weise et al. [2012].

A servicer for on-oribit-servicing consists of a satellite body equipped with at least two manipulators. On manipulator is used to provide a rigid connection to the client during the servicing operation. In this paper, we do not consider the initial berthing process, but assume an initial connection has been made.

The manipulator is equipped with an gripper which is identical to the module interface. Thus it can be used to move, power and control satellite modules. The modular structure of the client reduces a repair task to the task of removing a broken module and replacing it with a new one. Several constraints must not be violated: The module has to be inside the workspace of the manipulator, it has to be accessible and there may not be free floating modules. In order to move a module, it may not be locked to the client.

4 System Overview

A symbolic planer is used to generate action sequences which transfers satellites into desired goal configurations with given constraints. The challenge of this approach are the different abstraction and model levels that occur during planing and execution.

Satellite servicing takes place in the continuous three dimensional world while the symbolic planer operates on a finite set of symbols. Due to the regular structure of the grid defined by the cubes of the modular satellite, the representation of the client as well as effects and preconditions can be modeled in a symbolic representation of client configurations.

On the other hand, this model lacks information, when generated actions have to be mapped to manipulator trajectories. A „Pick" operation has to be aware of the workspace of the manipulator. When planing the trajectory, obstacles like antennas have to be considered as well as the dynamics of the free floating system of client and servicer. Modeling those in the domain of a symbolic planer is at least difficult. Thus, our approach integrates classic robot manipulation planer and their non symbolic models. The complete system is visualized in Fig. 5.

The *Perception* component creates a model of the current state of the client, based on static knowledge and sensors such as cameras or sensors integrated in the interfaces of the modules. The *Symbolic Planning* component generates a symbolic plan which transfers the satellite into a desired configuration. Different means are used to ensure the feasibility of the planed actions: 1) Symbolic precon-

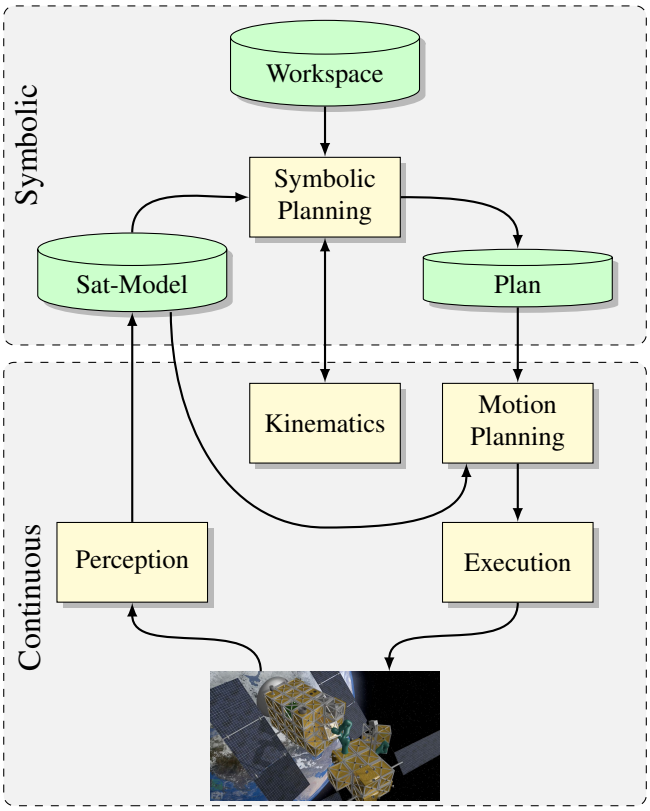

Figure 5. Overview over the symbolization process.

ditions are modeled in the planing domain. 2) The manipulators workspace is discretize and saved in the *Workspace Model*. The symbolic planer uses it to determine reachable modules or in heuristics to calculate servicer positions, from which a module is reachable. 3) Inverse kinematics and collision checkers are queried to check if feasible configurations exist.

The resulting plan is stored into the *Plan* database. *Motion Planing* algorithms are used to map the (known feasible) arm configurations of the planed states and actions to collision free trajectories. Those algorithms have full access to the continuous models and can also account for the system dynamics. Finally, controllers of the manipulator map the trajectory to their *Execution*.

5 Workspace Description

The workspace description is based on the work of Zacharias et al. [2009]. In the original approach, the goal is to describe the whole possible workspace of an manipulator. Since inverse kinematics are used, a discrete model of the workspace has to be used to approximate the description. In the case of servicing of modular satellites, the discretization is implied by the structure of the satellite modules. Anyhow, this restriction does not apply to the position of the servicer itself, which can still be arbitrary.

We therefor use a different discretization for the servicer position, which uses the grid of half the size. The same applies for orientations. Additionally to considering orientations aligned with the grid, we also consider rotations in 45 degree steps.

Another observation is, that manipulators usually (and especially the used one) have a rotational degree of freedom at the beginning and at the end, which is aligned along its main axis (the local z-axis). In the case of the gripper, this is the axis perpendicular to the surface of the satellite module. Rotations around this axis are used for rating the quality of the pose in the original approach. Due to the kinematics, we know, that if one orientation of this axis is reachable, all other orientations around this axis must be reachable too. Sampling is not necessary around this axis in the grid of the client. Only the six faces of the module have to be considered. A similar fact can be determined for the base of the manipulator. Due to the rotational degree of freedom around the local z-axis of the manipulator, the reachability of the workspace is rotational symmetric around the z-axis, the sampling of the base position can be omitted.

Those conclusions lead to the following discretisations:

- *Client space:* The client space is dicretized into a cubic grid where a cell has exactly the size of one satellite module. Six orientations, one for each side are considered.

- *Servicer space* The servicer may be placed in the center of a cube of the client grid, or on the corner of a cube with half the edge length and one corner in the center of the original cube. The orientation can be aligned in the grid, or change in 45 degree steps around the axis. Considering symmetry, 4 orientations have to be considered as well as 8 positions relative to the client grid. The remaining positions and orientations are provided by the client space discretization.

The workspace description in client space is generated for all $8 * 4 = 32$ possibilities of the servicer space. This model describes, which modules can be manipulated from a given position of the manipulator. It is called the forward model. An inversion of the model, describing where the manipulator has to be placed in order to manipulate a module can be generated form that model easily.

Those models are calculated offline and stored into a database. For the access by the heuristics, an efficiently accessible grid based representation in the 3D space is generated. For each position, feasible orientations are stored in the grid data structure. This representation al-

lows to calculate the intersection of two workspace descriptions in O(n), where n is the number of cells in the grid.

6 HTN Planing

Classical symbolic planner perform a search in the graph of possible states, connected by actions. The performance of this approach can be problematic for complex tasks. To avoid this, we use Hierarchical Task Network (HTN) Planning [Nau et al., 2003]. HTN allow the decomposition of planning problems on different abstraction layers. Solutions for the lower layers are only generated if solutions for higher layers have been found. HTN are graphs where the inner nodes are described as *Methods* and the leafs are called *Operators*. Operators are identical to the actions of classical symbolic planner and can be executed by the servicer. Methods specify a decomposition of a sub-task into a solution consisting of methods and operators.

The decomposition allows the introduction of task specific knowledge. Similar sub problems occur frequently and a solution for them may be reused without spending time for the search. Different methods may be specified for a sub problem to provide different solutions, if one solution is not always applicable.

6.1 PyHop

PyHop[1] is a Python implementation of a HTN Planing system by D. Nau. It uses the python language to dynamically specify the decompositions of methods. Thus, efficient algorithms can be used to generate heuristic decompositions based on the current state. Heuristics can further be used to dynamically create symbols from continuous properties. The Python integration allows also to query external libraries. For example a valid solution of the inverse kinematics for a module's pose can be required as a precondition for an action. In PyHop, the world state is modeled as a python class. Operators modify the world state, while methods are limited to create decompositions.

6.2 State model

The modeled world state consists of three object types. Methods are implemented which map the internal state of the object to queries which may be used as preconditions. The object types are:

- *Grid* It models the client and its surrounding. It consists of a three dimensional array, where every element of the array represents an position in the grid. It can either be empty or occupied by a satellite module. In the last case, a reference to the satellite module is stored. Further, for the manipulator, a reference to a potentially grasped object is stored. Based on the grid, the predicates "Occupied", "At" and "Taken" are provided. "Occupied" provides access to a module at a position in the grid. "At" maps an object to its current position. "Taken" returns the object, which is currently graped.

- *Satellite Module* The satellite module object contains a list of interfaces of the module and its orientation. The orientation is described as roll, pitch and yaw angles. Its values are limited to multiples of 90 degrees.

- *Interface* The interface object contains a list of other interfaces it is connected with, as well as references to their satellite modules. It is used by the global Locked(Module, Interface, Module, Interface) method, which is true, if the two interfaces are connected by their locking mechanism.

6.3 Operators

The On-Orbit-Servicing scenario defines five operators:

- Pick(Manipulator, Module, Interface): Take a module. Move the arm towards the specified interface and close the lock between arm and interface. As a precondition, at least three sides of the module have to be accessible, otherwise the module cannot be moved.

- Place(Manipulator, Position, Orientation): Moves the grasped module to the specified location. Precondition: A module must be grasped and locked to the manipulator.

- Lock(Module, Interface, Module, Interface), Unlock (Module, Interface, Module, Interface): Opens and closes the interfaces between two modules. As a precondition, the modules have to be located next to each other. Is also used to open the connection between manipulator and the grasped interface.

- Position Servicer(Position, Orientation): Moves the servicer to a desired position. Currently implemented as an operator. Will be extended to a method in the future in order to allow motion sequences to reach locations farther away from the current location.

6.4 Decompositions: Operators

We use the following decompositions:

- *Move Method* The "Move Method" ensures two properties of the client, before the specified module is taken: The module itself has to be accessible and the place position has to be free. Therefor, a set of obstacles is generated consisting of the module at the target position and (at most) three neighbors of the

[1] https://bitbucket.org/dananau/pyhop

module to be moved. For all obstacles, "Move Somewhere" Methods are scheduled, followed by Pick and Place Methods. The analogous defined "Move Somewhere" Method leads to a recursive clean up around the module.

- *Move Somewhere Method* Like the Move Method, but with "Place Somewhere" instead of "Place" and without a target position in the obstacle set.
- *Place Method* Decomposes to the "Positon Servicer" Methode, the "Place" operator, a sequence of the "Lock" Method for the neighbor interfaces and the "UnLock" Method for the manipulator.
- *Pick Method* Analogous to the "Place" Method
- *Place Somewhere Method* Like "Place", but the position is not taken from the goal, but from the "Get Place Position" heuristic. Generating the place position on this level enables accounting for the results of the lower level decomposition of the "Move Somewhere Method"
- *Lock Method* Can be decomposed to $\emptyset$, if the participating interfaces are not locked, otherwise
- *Unlock Method* Analogous to Lock Method
- *Position Servicer* Moves the servicer to a position, where a desired module is graspable (as defined by the workspace description and the kinematics). It is decomposed to $\emptyset$, if the module is graspable from the current position, otherwise a position generated by the "Get Servicer Position" is generated.

Another method "Process Goal" is required to map the specified goals to a sequence of "Move" Methods.

6.5 Heuristics

- *Get Place Position* Return a reachable position in the work space. The position has to be feasible for placing a module; there have to be one to three neighbors. Positions, from which a transfer to the goal position of that module is possible, are preferred, as well as the target position itself.
- *Get Servicer Position* Returns a position, from which a module at a given position can be grasped. Possible positions are determined by the workspace decomposition. Further, a list of future goals from the symbolic planer is used. Positions in the future are considered. The selected position of the servicer should be feasible for a many consecutive position after the current action as possible. Therefore, the set of possible positions is generated and intersected with the remaining set of the previous position. Positions in the last non empty set are candidates for the result. The highest rated position is selected. In case of equal ratings any position can be returned.

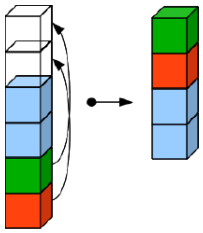

Figure 6. Evaluation scenario.

Number of blocks	Number of blocks to move			
	1	3	6	9
10	2.34	6.73	14.01	21.96
12	2.19	6.79	14.01	21.96
14	2.21	6.75	14.08	21.87
16	2.23	6.79	14.16	22.05
20	2.29	7.01	14.57	22.61

Table 1. Resulting planning times from the simple experiment (in ms).

7 Evaluation

In a first experiment, we analyze feasibility an performance of our approach. A simplified client in the form of a stack of modules is assumed. The task is to move objects from one end of the stack to another. It is sketched in Fig. 6. The experiment is repeated with different sizes of the client and numbers of modules to move. Feasible plans could be found for all configurations. The times required for symbolic planing are shown in Tab. 1. For comparison: An approach with the Metric FF Planer in the same domain took 1 to 60 seconds for comparable problem sizes.

We conducted a second experiment with a more challenging problem for the planer. In the first setup, the order, in which the modules had to be placed were the same in which the were taken. This leads to a sequence in the form of alternating pick and place operations for each module. In this experiment, we invert the order, in which the modules have to be placed. This forces the planer to use temporary place locations. Again, solutions where found for all problem instances. Results are printed in Tab. 2. The times are longer, but still faster then Mertic FF in the simple domain. A remarkable feature which can be seen in the numbers is that the time for planing depends only on the number of modules to move, not on the number of modules in the satellite.

Number of blocks	Number of blocks to move			
	1	3	6	9
10	2.14	14.78	75.67	21425.14
12	2.01	14.48	89.27	1411.93
14	2.04	14.31	84.41	1398.97
16	2.09	13.58	77.76	1409.27
20	2.14	14.82	83.93	1428.91

Table 2. Resulting planning times from the harder experiment (in ms).

8 Conclusions

We have presented a system for the symbolic planing of actions in an On-Orbit-Satellite servicing domain. A workspace description has been developed to connect the discrete domain of the symbolic planer to the continuous domain of manipulation tasks. A Hierarchical Task Network Decomposition has been presented, together with the integration of the workspace description which enables the generation of arbitrary position in the symbolic model base on continuous models and manipulation algorithms. The system was evaluated in experiments in simulation.

9 Acknowledgment

The study has been co-funded by the German Aerospace Centre, DLR under national registration no. 50RA1202.

References

S Cambon, Fabien Gravot, and Rachid Alami. A robot task planner that merges symbolic and geometric reasoning. In *ECAI*, pages 895–899, Valencia, 2004.

Jaesik Choi and Eyal Amir. Combining planning and motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 238–244, Piscataway, NJ, USA, 2009. IEEE. ISBN 978-1-4244-2788-8.

Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *Safety, Security & Rescue Robotics (SSRR), 2009 IEEE International Workshop on*, pages 1–6. IEEE, November 2010.

LP Kaelbling and Thomas Lozano-Perez. Hierarchical task and motion planning in the now. *IEEE Intl. Conf. on Robotics and Automation*, 2010.

Lars Karlsson, Julien Bidot, and Fabien Lagriffoul. Combining task and path planning for a humanoid two-arm robotic system. In *TAMPRA workshop, ICAPS 2012*, 2012.

D S Nau, T C Au, O Ilghami, U Kuter, J W Murdock, D Wu, and F Yaman. SHOP2: An HTN Planing System. *Journal of Artificial Intelligence Research*, 20: 379–404, 2003.

Dana Nau. Pyhop HTN Planner, 2013. URL `https://bitbucket.org/dananau/pyhop`.

Steffen W Rühl, Andreas Hermann, Zhixing Xue, Thilo Kerscher, and Ruediger Dillmann. Graspability: A Description of Work Surfaces For Planning of Robot Manipulation Sequences. In *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011. IEEE.

Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation Planning Among Movable Obstacles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3327–3332. IEEE, April 2007. ISBN 1-4244-0602-1. doi: 10.1109/ROBOT.2007.363986.

J. Weise, K. Brieß, A. Adomeit, H.-G. Reimerdes, M. Göller, and R. Dillmann. An intelligent building blocks concept for on-orbit-satellite servcing. In *i-SAIRAS: International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2012.

F Zacharias, C Borst, and G Hirzinger. Object-Specific Grasp Maps for Use in Planning Manipulation Actions. In *Advances in Robotics Research*, volume 7, pages 203–213, Braunschweig, Germany, 2009. Springer-Verlag.

Franziska Zacharias, Christoph Borst, and Gerd Hirzinger. Capturing robot workspace structure: representing robot capabilities. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, number 7, pages 3229–3236, San Diego, California, USA, 2007.