MOPL: A Multi-Modal Path Planner for Generic Manipulation Tasks

Sören Jentzsch, Andre Gaschler, Oussama Khatib and Alois Knoll

***Abstract*— For intelligent robots to solve real-world tasks, they need to manipulate multiple objects, and perform diverse manipulation actions apart from rigid transfers, such as pushing and sliding. Planning these tasks requires discrete changes between actions, and continuous, collision-free paths that fulfill action-specific constraints. In this work, we propose a multi-modal path planner, named MOPL, which accepts generic definitions of primitive actions with different types of contact manifolds, and randomly spans its search trees through these subspaces. Our evaluation shows that this generic search technique allows MOPL to solve several challenging scenarios over different types of kinematics and tools with reasonable performance. Furthermore, we demonstrate MOPL by solving and executing plans in two real-world experimental setups.**

I. Introduction

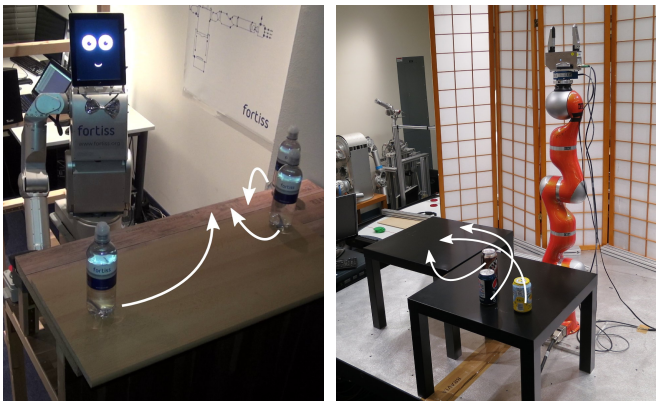
Realistic manipulation problems with multiple objects cannot be solved in a hierarchical search: Motion paths that push an object are sparse in the robot's configuration space, must fulfill geometric constraints, and cannot be undone. While planning a sequence of manipulation actions with movable objects can be seen as a generalization of classical, collision-free path planning, it is a computationally harder problem [1]: Movable objects are obstacles themselves, they cannot move on their own, but must rather be transported through paths that are sparse and lower in dimensionality than the configuration space. Objects that are not transported must be placed on support surfaces, which are again lower-dimensional manifolds. In the early days of the Handey robot [2], it was already realized that manipulation may require a sequence of primitive actions with re-grasping.

Our proposed algorithm for **m**ulti-m**o**dal **p**ath p**l**anning (abbreviated, MOPL) allows *generic definitions of manipulation primitives* for serial kinematics, including but not limited to rigid transfers, stable pushes, and sliding motions. MOPL respects the subspaces defined by these actions, and is built upon a bidirectional rapidly-exploring random tree (RRT), which spans a search tree from both the initial configuration and the goal. In this work, the goal can also be defined as a goal region or a whole subspace, allowing for arbitrary configuration of certain components of the search space. The edges of the search trees represent discrete modes, which are instances of manipulation actions, such as pushing a certain object along a certain direction.

S. Jentzsch and A. Gaschler are with fortiss GmbH, An-Institut Technische Universität München, Munich, Germany. Correspondence should be addressed to jentzsch@in.tum.de and gaschler@fortiss.org

O. Khatib is with the Department of Computer Science, Stanford University, Stanford, CA.

A. Knoll is with the Department of Informatics, Technische Universität München, Munich, Germany.



(a) Meka Clean-up scenario (b) Kuka Table scenario

Fig. 1: MOPL can solve complex manipulation tasks by finding a path consisting of collision-free sequences of manipulation primitives. The planner itself is domain-independent, thus allows primitives for various types of transfer motions to be defined for a wide range of kinematics and scenarios. A video is available at http://youtu.be/1QRvjBw58bU.

The main contribution of our approach with respect to earlier multi-modal planners [3], [4], [5], [6] is that instead of applying projection functions to project samples onto the constraints defined by each primitive, we rather sample appropriate state vectors that fulfill these constraints implicitly. These samples may be incomplete containing free coefficients, but allow determining a single well-defined nearest neighbor and enable a flexible and adaptive sampling procedure. Furthermore, our implementation keeps the generic algorithm separate from problem definitions, which are given in a markup language, and allows diverse problems from redundant kinematics to mobile robots to be solved without tuning parameters of the planner.

II. Related Work

Our notion of the manipulation planning problem was first defined by Alami, Siméon and Laumond in 1989 [7], and is most commonly approached by random sampling in a combined search space of robot and object configurations [7], [8, p. 332ff.]. Siméon et al. [9] generate a probabilistic roadmap in a composite configuration space of robots and movable objects to capture the connectivity of sub-dimensional manifolds of manipulation actions. Apart from pure pick-and-place tasks, Barry et al. [10], [4], [3] describe a multi-modal planner based on rapidly exploring random trees (RRT) allowing for more general actions, such as pushing or sliding. Stilman and Kuffner [11] studied

the related problem of navigation among movable obstacles, where a robot can move certain objects, but the goal is to reach a certain robot position, irrespective of the locations of the objects. Their approach is to extract the navigational structure from the high-dimensional state space, and achieves resolution completeness under reasonable restrictions to the problem. Berg et al. [12] later extended this work to a probabilistically complete algorithm. Certainly, manipulation planning is also a hybrid problem, with continuous paths and discrete transitions between modes. Hauser and Ng-Thow-Hing [6] search in a hybrid search space of modes, such as contact states, and lower-level transitions between these states. Using several heuristics, they can demonstrate a bipedal humanoid robot pushing an object.

While manipulation is closely related to grasp planning and dynamics with its questions of stability, friction, and physics [13], [8], [14], we constrain ourselves to pre-defined manifolds of manipulation poses, which in the following will be introduced as *transfer primitives*.

III. Approach

Our definition of the multi-modal manipulation problem closely follows Barry's notion of the diverse action manipulation (DAMA) problem [10], [4], [3]. We first define the manipulation task and the types of primitive motions for this task, after which we propose an RRT-based sampling search, the MOPL algorithm, to approach this class of problem.

A. Problem Definition

Assuming a robot with configuration space (C-space) R and n movable objects and their corresponding C-spaces $O_1, ..., O_n$, our approach searches in the composed C-space $X = R \times O_1 \times ... \times O_n$. A *trajectory* from a configuration x_S to x_G is then defined as a continuous function $\tau : [0, 1] \to X$ that fulfills $\tau(0) = x_S$ and $\tau(1) = x_G$.

Next, we define a set of *manipulation primitives*, which describe the actions the robot can perform in X. Manipulation primitives return a trajectory, which most commonly traverses only a subspace of X. A manipulation primitive can only begin at or reach a certain set of configurations, also referred to as its *domain*.

Definition III.1 (Manipulation Primitive). Given an initial configuration x_S and a goal configuration x_G, a *manipulation primitive* $p(x_S, x_G)$ returns a trajectory from x_S to x_G. It is *applicable* only to pairs of configurations within its domain $X(p) = \{(x_S, x_G) \in X \times X \mid (x_S, x_G) \text{ in domain of } p\}$. The set of valid initial configurations $X_S(p)$ and the set of reachable configurations $X_G(p)$ for p are then given as $X_S(p) = (X \times \{x_G\}) \cap X(p)$ and $X_G(p) = (\{x_S\} \times X) \cap X(p)$. Finally, $X_G(p|x_S) = \{x_G \in X \mid (x_S, x_G) \in X(p)\}$ gives us the set of reachable configurations from a certain initial $x_S \in X$ using primitive p.

Manipulation primitives can be separated into the two classes of transit and transfer primitives. For a primitive p, let $(x_S, x_G) \in X(p)$ and $\tau = p(x_S, x_G)$. The primitive p is a *transit* primitive if and only if for all $\alpha \in [0, 1]$, the configuration of every object in $\tau(\alpha)$ remains unchanged with respect to $\tau(0)$. Otherwise, p is called a *transfer* primitive.

For example, our implementation `Transit` of the transit primitive returns a trajectory from initial to goal configuration following a straight line in the joint space of the robot, assuming a domain of configuration pairs $(x_S, x_G) \in X$ in which all objects rest unmoved on a support surface in both x_S and x_G.

In contrast, the transfer primitive `Push` describes the robot pushing an object in a certain direction. The domain of `Push` consists of configuration pairs $(x_S, x_G) \in X$, in which all objects rest on a support surface in x_S, only one object moves on its support surface between x_S and x_G, and the robot is in pushing contact with this object along the pushing path to x_G. In our implementation, `Push` utilizes primitive nesting and calls `Transit` to move the robot to the initial pushing position, in which the gripper is in contact with the object and their respective center points are aligned along the pushing path. `Push` then returns a trajectory sequence, in which the robot first moves to the pushing pose via `Transit`, and then traverses along the pushing path.

Similar to the DAMA problem in [3], the problem can be defined as follows:

Definition III.2 (Multi-modal Manipulation Problem). The multi-modal manipulation problem $\mathcal{P}$ is a tuple $\langle R, \{O_1, ..., O_n\}, \{B_0, ..., B_q\}, \{p_0, ..., p_m\}, x_0, X_G \rangle$, in which R is the configuration space for a robot, $\{O_1, ..., O_n\}$ are the configuration spaces for the movable objects, $\{B_0, ..., B_q\}$ is a set of fixed obstacles, $\{p_0, ..., p_m\}$ is a set of manipulation primitives, $x_0 \in X$ is an initial configuration, and X_G is a set of goal configurations.

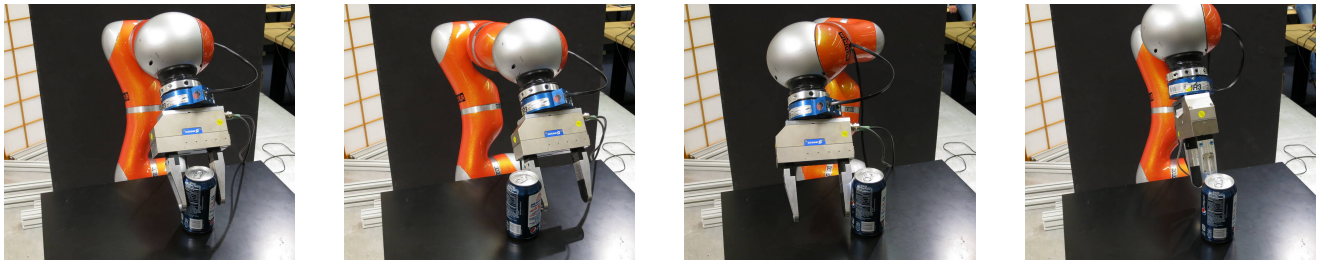
Note that X_G often contains an infinite number of possible configurations, in which only the goal configuration of (individual) objects is of interest.

Let $X_{free}(\mathcal{P})$ be the *free space* for multi-modal manipulation problem $\mathcal{P}$ containing all configurations in which there is no contact between the robot bodies, objects, or obstacles. A trajectory τ generated by primitive p is *collision-free* in $\mathcal{P}$ if and only if for all $\alpha \in [0, 1]$: $\tau(\alpha) \in X_{free}(\mathcal{P})$. Note that according to [3], transfer primitives can permit certain collisions for the actual path planning, as `Push` may permit collisions between gripper and pushed object. A *solution* is a collision-free trajectory sequence generated by the given primitives from x_0 to any configuration in X_G.

B. MOPL Planner

To solve multi-modal manipulation problems, we modify and extend the sampling-based RRT algorithm in multiple ways, concerning the Extend step, sampling and distance metrics.

1) Empty Space Planner: As our C-space X is only partially controllable [7], we apply the concept of an *empty space planner* (ESP) to guide the non-holonomic way of extending configurations. By non-holonomic extension, we mean extension within constrained subspaces of motion



(a) `Transfer-Rigid` (b) `Push-Exterior` (c) `Push-Interior` (d) `Push-Frontal`

Fig. 2: Available transfer primitives in both the KUKA TABLE and KUKA CABINET scenario. The design of MOPL is domain-independent and allows geometric primitives for a particular robot to be defined in a markup language separate from the search algorithm. Whereas pushing an object to a certain position precisely defines the robot pose, `Transfer-Rigid` induces one degree-of-freedom (the rotational axis of the object), which MOPL uses to avoid joint limits in the robot pose.

primitives, for example pushing along a certain direction. Given a predefined set of primitives, the ESP returns a trajectory sequence from one towards another selected configuration in X, assuming the absence of any obstacles. The EXTEND function then checks this sequence for collisions and returns a collision-free (possibly truncated) trajectory sequence.

In order for the ESP to plan through those low-dimensional subspaces of X, [3] proposes to implement two functions ISUSEFUL and PROPAGATE for each primitive: Whereas ISUSEFUL first checks whether p is applicable to (x_S, x_G) with $(x_S, x_G) \in X(p)$, PROPAGATE returns a trajectory sequence from x_S to x_G by applying the primitive's manipulation. The overall ESP repeatedly chains propagate steps of randomly selected useful primitives, ultimately returning a trajectory sequence from x_S towards x_G.

In contrast to [3], our implementation of ISUSEFUL does not evaluate computationally costly inverse kinematics (IK), but rather assumes all poses to be feasible. Later, PROPAGATE calculates the IK of contact points. When IK fails, it propagates only partially towards x_G, and tries again with a different useful primitive.

As an example, the primitive `Push` is found useful if all objects are placed on a support surface in x_S and at least one object moves on its support surface between x_S and x_G. PROPAGATE then returns a trajectory sequence from x_S towards x_G by pushing the object nearest to the robot, computing the initial pushing position, propagating via `Transit`, then calculating the final pushing position with the object positioned as in x_G and appending the trajectory for this pushing path.

2) Sampling: For our non-holonomic RRT planner to expand through and respect the lower-dimensional subspaces of X, we cannot sample uniformly at random. Barry [3] proposes primitive projection functions $f_i(x_I, x_S)$, which project x_S to a constrained subspace defined by primitive p_i and configuration x_I, with the resulting configuration being reachable for p_i within the ESP. Barry emphasizes that the choice of projection functions must ensure that there is a non-zero probability of the ESP being able to apply every possible sequence of primitives.

We take a different approach, allowing undefined coefficients in sampling, which will be resolved later by nearest neighbor selection. First, we randomly select a subspace of X and a respective sample. If that subspace is an object, it will either be sampled on a support surface, or sampled in a predefined region in free space, entailing the robot being set to an appropriate `Transfer-Rigid` pose. For all other subspaces, we choose between three options: setting it to the start configuration, to the goal configuration, or leaving it undefined. Undefined subspaces will later be resolved in nearest neighbor comparison to yield a minimum distance.

Compared to Barry's approach, our sampling allows for a more meaningful nearest neighbor search, as our sample does not require projection anymore, and thus remains in a well-defined fixed state. Furthermore, being independent of primitive projections, we can guide the search in a more fine-grained fashion through our C-space for each subspace by adaptively sampling from the goal or undefined subspaces. On the downside, though, our approach requires some expertise and initial adjustment of probability parameters, but we managed to find a fixed setting which generalizes to all scenarios presented in this work. Ultimately, it turned out to yield at least similar results compared to [3], but allowed us to solve more complex puzzles and manipulations of multiple objects.

3) Distance Metrics: Since X is not fully controllable, our distance metric ρ cannot be a simple Euclidean distance, but should rather reflect the resulting robot path, and thus the probability of colliding when propagating from one to another configuration. The empty space planner (ESP) could, in theory, return an accurate measure, but would require immense computational costs for the nearest neighbor search. Instead, we devise a heuristic distance metric to model the basic characteristics of the ESP, circumventing the computation of inverse kinematics.

Our distance metric is composed of distance metrics ρ_i for each subspace. For object subspaces O_i we can directly measure Euclidean distances. For a robot with revolute joints, the distance metric considers the overall joint, Euclidean and angular distances between end-effector poses with a weighting parameter. While [3] could prove that their algorithm is

exponentially convergent when taking the maximum distance of all subspaces to define ρ, however, this would not reflect well the non-holonomic motion in our C-space. An important example is a short displacement of object positions, which has minimal impact on this maximum metric, but may require a robot to travel from and to all objects, resulting in a very long path.

Our proposed metric sums up all subspace distances ρ_i. For each object to be moved, it adds a constant penalty, and the distance for the robot to move to and from the object, calculated by forward kinematics. This metric is reasonably efficient and reflects well the basic characteristics of our non-holonomic space by introducing additional costs for transitions and moving objects that cannot move by themselves.

C. MOPL Algorithm

In the following, we extend the concepts of the RRT algorithm to multi-modal manipulation. Our implementation, named MOPL (Algorithm 1), is derived from the bidirectional RRT algorithm, which grows trees from both the initial configuration and the goal set. A more detailed description of an earlier version of the algorithm can be found in the main author's master thesis [15].

We iteratively select one of both trees, represented by V_a, sample from the C-space, compute the nearest neighbor, and extend the tree in a collision-free fashion yielding new vertices and edges. If the extension step induces at least one additional vertex in V_a, we extend the opposing tree V_b towards the last configuration added to V_a. In other words, while V_a explores in a random fashion, V_b will be extended towards V_a, trying to find a solution path by connecting both. In this work, we use an unbalanced version of bidirectional search by swapping both trees after each iteration, so they are not guaranteed to be of equal size.

For planning in both directions in our non-holonomic system, a crucial adaptation is required. Since our goal set will most likely be a subspace of X, the backwards tree should ideally cover this subspace. While [3] proposes to simply sample and add a goal configuration in every iteration, we can rather take advantage of our previously discussed undefined subspaces. When searching for the nearest neighbor in our backwards tree, we also consider the configuration given by PROJSAMP, which projects the sample x_S on the undefined subspaces of X_G and generates random samples for subspaces which are undefined in both x_S and X_G. This process always creates a fully-defined configuration in X_G closest to x_S.

Algorithm 1
MOPL$(X, B, P, x_0, X_G, \{\rho_0, ..., \rho_n\})$, derived from [3, p. 68]
Input: C-space $X = R \times O_1 \times ... \times O_n$, Fixed obstacles $B = \{B_0, ..., B_q\}$, Primitives $P = \{p_0, ..., p_m\}$, Initial configuration x_0, Goal set X_G, Distance metrics $\{\rho_0, ..., \rho_n\}$
Output: Trajectory sequence from x_0 into X_G

$V_a \leftarrow \{x_0\}$, $V_b \leftarrow \{\}$
$F \leftarrow$ **true** // *true extends forwards, false backwards*
while true do
  $x_S \leftarrow$ SAMPLE(X)
  $x_T \leftarrow \arg\min_{v \in V_a \cup \text{PROJSAMP}(\neg F, x_S, X_G)} \rho(v, x_S)$
  $\{\tau_0, ..., \tau_l\} \leftarrow$ EXTEND(x_T, x_S, X, B, P, F)
  $V_a \leftarrow V_a \cup \bigcup_{\tau \in \{\tau_0, ..., \tau_l\}} \bigcup_{\alpha \in [0,1]} \tau(\alpha)$
  if $l > 0$ or there are new configurations in τ_0 **then**
    // *extend V_b towards V_a*
    $x_T \leftarrow \arg\min_{v \in V_b \cup \text{PROJSAMP}(F, x_S, X_G)} \rho(v, \tau_l(1))$
    $\{\sigma_0, ..., \sigma_k\} \leftarrow$ EXTEND$(x_T, \tau_l(1), X, B, P, \neg F)$
    $V_b \leftarrow V_b \cup \bigcup_{\sigma \in \{\sigma_0, ..., \sigma_k\}} \bigcup_{\alpha \in [0,1]} \sigma(\alpha)$
    if $\sigma_k(1) = \tau_l(1)$ **then**
      return EXTRACTTRAJECTORY(V_a, V_b)
  SWAP(V_a, V_b), $F \leftarrow \neg F$

IV. EVALUATION

Our implementation of the MOPL algorithm keeps the search distinct from domain-specific problem descriptions. For collision detection, inverse kinematics, and graph algorithms, it uses functions of the Robotics Library [1] by Rickert [16]. Scenarios are specified in a markup language, defining robot and workspace models, sampling spaces, problem description, planner settings, manipulation primitives, sampling and metric parameters, but also post processing routines like path smoothing.

To show the generic capabilities of MOPL, we evaluate four non-trivial scenarios that cover diverse robot kinematics, multiple objects to plan for, and a variety of manipulation primitives, involving pushing and grasping objects. Table I shows the benchmark results on all scenarios averaged over 200 runs.

A. Scenario MEKA CLEAN-UP

Our first scenario uses a humanoid robot with a torso, an arm, and a tendon-driven hand, built by Meka Robotics. Altogether it features 15 degrees-of-freedom, but we will only plan for the torso and the arm, resulting in a 10-dimensional configuration space for the robot. While we show a detailed CAD model for visualization purposes, our collision detection routine utilizes a bounding convex decomposition for efficiency [17], [18]. In this scenario, the robot acts as a bartender and is supposed to clean up the workspace by aligning all three bottles on the lower support surface (Figure 1a). Besides `Transit`, the motion primitives include grasping and transferring an object (`Transfer-Rigid`), pushing with the palm or interior surface of the hand (`Push-Interior`), and pushing with the exterior surface of the hand (`Push-Exterior`). Whereas pushing an object in a certain direction fully defines the contact configuration, transferring a rigidly attached object has one degree-of-freedom and allows the robot to grasp from any direction. For calculating the respective hand pose in the PROPAGATE step, we implement an iterative approach to inverse kinematics that avoids joint limits, which results

[1] `http://www.roboticslibrary.org`

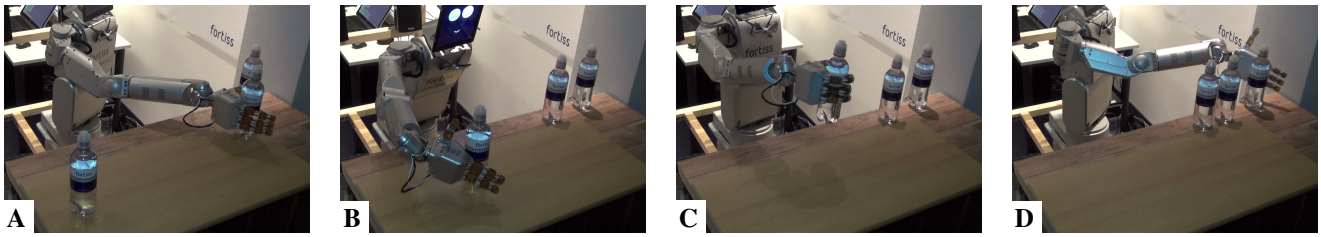


Fig. 4: Example real-world solution of the MEKA CLEAN-UP scenario featuring a 10-degrees-of-freedom manipulator. While the bottle in the center can be easily picked up and transferred directly (**A**), the leftmost bottle is not immediately accessible for grasping, but can be pushed towards a position where it can then be transferred (**B**, **C**). Finally, in order to align all three bottles, the rightmost bottle is being pushed with the exterior surface of the hand (**D**).

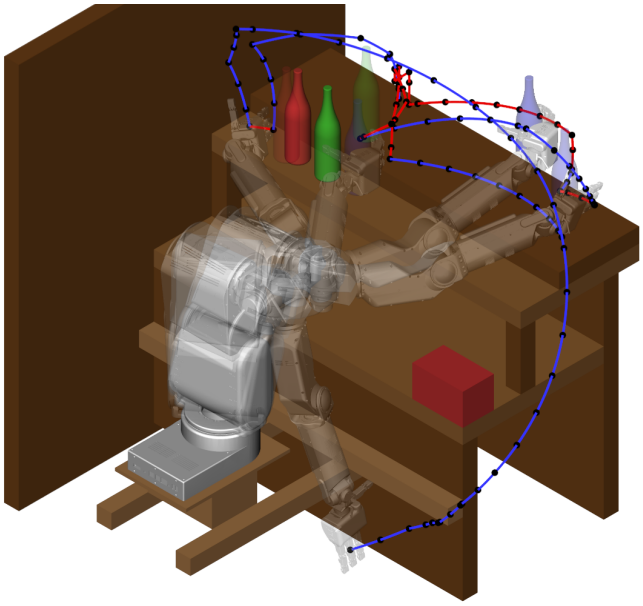


Fig. 3: Example solution path to the MEKA CLEAN-UP scenario in simulation. Edges drawn in red correspond to transfer primitives being applied, blue edges to transit primitives. The illustrated solution path has 101 vertices (black dots) and a Cartesian length of 6.47 meters.

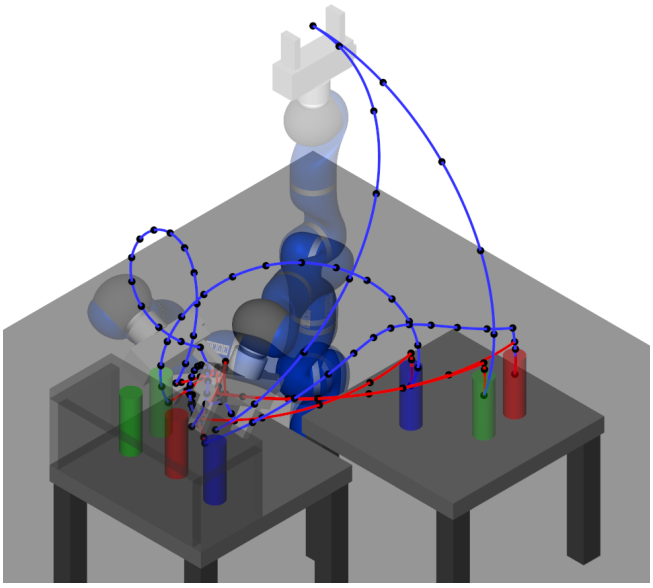


Fig. 5: Example solution path to the KUKA CABINET scenario in simulation. Edges drawn in red correspond to transfer primitives being applied, blue edges to transit primitives. The illustrated solution path has 118 vertices (black dots) and a Cartesian length of 10.12 meters.

in more natural robot poses and avoids near-singular robot configurations.

In this scenario, the robot plans in a 19-dimensional composed C-space with two separate support surfaces on top of the table and obstacles, such as the table itself or the wall (Figure 3). All three manipulation primitives are useful to the solution, as transferring between support surfaces requires `Transfer-Rigid`, the rightmost blue bottle must first be pushed towards a position where more space for grasping is available (`Push-Interior`), and the leftmost red bottle can easily be transferred by `Push-Exterior`. Figure 3 shows a solution path in simulation, including the respective hand poses induced by the applied primitives. Figure 4 shows a real-world demonstration of a solution.

When benchmarking this scenario, all 200 instances could be solved within a given time limit of 10 minutes each, resulting in 152 seconds average search time. As this scenario features a high-dimensional C-space with 19 dimensions and a complex robot with well-defined limits for each joint, the computation of the inverse kinematics becomes more complex (38% of total search time), but also the nearest neighbor search takes increasingly more time (49% of total search time) with each newly added vertex. Although the random nature of this RRT-based algorithm yields large standard deviation values, after applying a simple but common path smoothing technique by shortcutting `Transit` edges, the final solution paths vary only by about 7% in Cartesian length.

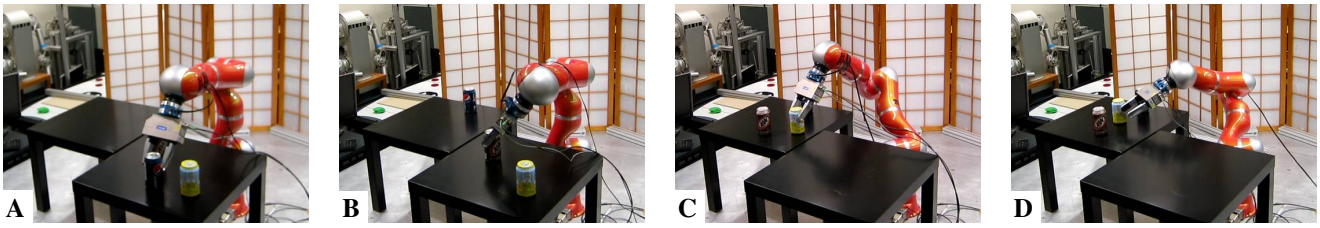


Fig. 6: Example real-world solution of the KUKA TABLE scenario. A 7-degrees-of-freedom manipulator is supposed to transport three objects from one support surface (**A**) and line them up on another (**D**). While the push primitives cannot move objects to a different support surface, they are often needed to move objects that are close together (**C**), where `Transfer-Rigid` would collide.

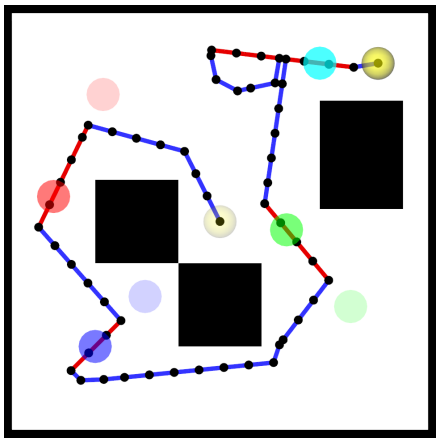


Fig. 7: Example solution path to the MOBILE BLOCKED scenario in simulation. The illustrated solution path has 58 vertices (black dots) and a Cartesian length of 16.00 meters.

B. *Scenarios* KUKA TABLE *and* KUKA CABINET

Our next two scenarios feature manipulation tasks with a 7-degrees-of-freedom Kuka robot. In the KUKA TABLE scenario, the robot has to transport three objects to a nearby table and line them up (Figure 1b). Additionally, in the KUKA CABINET scenario, the objects have to be placed within a cabinet. Besides `Transfer-Rigid`, the robot can apply several push primitives (Figure 2). These are crucial to solve the latter scenario, because all rigid transfers to the goal would collide with the cabinet. Figure 6 shows a solution for KUKA TABLE in the real world, while Figure 5 illustrates a solution path for KUKA CABINET in simulation.

In contrast to the Meka robot, Kuka's larger joint limits allow for a larger variety of poses for individual actions, which leads to these scenarios being solved much faster. Whereas KUKA TABLE can be solved in under five seconds on average, KUKA CABINET requires several primitives to be applied for each of the three objects and can be solved in under one minute on average.

C. *Scenario* MOBILE BLOCKED

Our fourth and last scenario MOBILE BLOCKED features a simple holonomic mobile robot operating in a five-by-five meters 2D world. This scenario is mainly to show that MOPL is not limited to joint-based robots and that the free subspace coefficients in our algorithm allow us to introduce movable objects without fixed goal positions. In this scenario, the robot has to transfer the lower left object before heading to the center, but is blocked by three other movable objects (Figure 7). Here, pushing is the only transfer primitive. This scenario is solved in about 11 seconds on average.

D. *Limitations*

The current version of MOPL is a flat planner and does not subdivide the problem through hierarchical planning as proposed by Barry [3], or applies other kinds of heuristics to guide the search in higher dimensional search spaces. Thus, MOPL does not scale well with larger numbers of movable objects.

Manipulation in MOPL is quasi-static. Dynamic motion primitives, such as throwing objects, can in principle be integrated with physics simulation, but would limit search to a forward search with a single tree. Also, contacts with unrelated objects are treated as collisions and are not allowed, which limits solving highly cluttered scenarios.

V. CONCLUSION AND FUTURE WORK

Our approach to multi-modal path planning contributes a new sampling technique without explicit projection and a nearest neighbor computation that completes the state vector. It has been shown to be effective for a wide range of kinematics, tools and motion primitives, from highly redundant humanoid manipulators to mobile robots. MOPL does not require fine-tuning of planner parameters to successfully plan and solve challenging scenarios for different robot platforms including user-defined motion primitives. All software is published in an open-source repository[2].

For future work, we plan to integrate additional RRT variants, which can potentially increase the performance of the search.

ACKNOWLEDGEMENTS

The authors would like to thank Torsten Kröger for his help with the Kuka experiments. This research was supported by the European Union's Seventh Framework Programme

[2]`https://github.com/fortiss/mopl`

TABLE I: Benchmark and evaluation of search space and solution quality of four different types of scenarios, which cover all types of manipulation primitives and robot kinematics described in the scenario discussion. All benchmark tests were executed non-parallelized on a 3.10 GHz desktop computer and repeated 200 times with different random seeds, measuring mean and standard deviations.

Scenario	MEKA CLEAN-UP Place three objects on a lower support surface using diverse transfer primitives.		KUKA TABLE Transfer three objects from one table to another.		KUKA CABINET Place three objects from one table in the cabinet on the other table.		MOBILE BLOCKED Get multiple movable objects out of the way to rearrange a certain object.	
Timeout	10 minutes		1 minute		3 minutes		1 minute	
Success rate [%]	100%		100%		96.5%		97.5%	
Benchmark Results	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Total search time [s], thereof	152.367	81.054	4.955	2.329	59.766	39.969	10.966	9.676
a) Sample Generation [s]	4.303	1.646	0.159	0.087	1.209	0.588	0.031	0.015
b) Nearest Neighbor Search [s]	75.270	56.188	0.617	0.584	28.208	25.624	9.052	8.232
c) PROPAGATE step [s]	53.958	18.190	2.985	1.472	25.747	12.486	0.031	0.015
d) CONNECT step [s]	14.253	5.759	0.417	0.202	3.038	1.642	0.273	0.168
Inverse Kinematics [s]	57.828	19.635	3.125	1.538	26.759	12.949	0	0
Iterations of the algorithm	3 505	1 295	225	117	1 693	823	4 132	1 959
Edges of the search trees	7 893	3 095	1 387	639	9 065	4 369	4 820	2 205
Vertices of the search trees	7 896	3 095	1 391	640	9 070	4 369	5 304	2 409
Collision checking queries, thereof	188 770	56 460	20 234	4 709	71 377	24 258	55 445	20 570
Collision checks in free space	184 466	54 912	19 906	4 616	69 245	23 313	44 127	15 416
Solution path								
Vertices of the path	104.330	10.315	74.720	16.400	142.860	24.571	63.703	8.214
Cartesian length of the path [m]	6.171	0.447	8.112	1.535	11.071	1.928	17.389	2.202
Primitives applied in the path, thereof								
Number of `Transit`s	5.910	0.816	4.515	0.885	12.150	2.067	6.462	1.374
Number of `Push`es and variants	1.710	0.747	0.290	0.692	7.187	1.881	5.462	1.374
Number of `Transfer`s	4.165	1.115	3.460	0.912	4.415	1.340	0	0

through the projects JAMES under grant agreement no. 270435[3], SMErobotics under grant agreement no. 287787[4], and HBP under grant agreement no. 604102[5].

REFERENCES

[1] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[2] T. Lozano-Pérez, J. Jones, E. Mazer, P. O'Donnell, W. Grimson, P. Tournassoud, and A. Lanusse, "Handey: A robot system that recognizes, plans, and manipulates," in *Intl Conf on Robotics and Automation (ICRA)*, vol. 4, 1987, pp. 843–849.

[3] J. L. Barry, "Manipulation with diverse actions," Ph.D. dissertation, Massachusetts Institute of Technology, June 2013.

[4] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez, "A hierarchical approach to manipulation with diverse actions," in *IEEE Conference on Robotics and Automation (ICRA)*, 2013.

[5] K. Hauser, "Motion planning for legged and humanoid robots," Ph.D. dissertation, Stanford University, December 2008.

[6] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[7] R. Alami, T. Siméon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps," in *International Symposium on Robotics Research*, 1989.

[8] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[9] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 729–746, 2004.

[10] J. Barry, K. Hsiao, L. P. Kaelbling, and T. Lozano-Pérez, "Manipulation with multiple action types," in *Experimental Robotics*, ser. Springer Tracts in Advanced Robotics. Springer International Publishing, June 2012, vol. 88, pp. 531–545.

[11] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics*, vol. 2, no. 04, pp. 479–503, 2005.

[12] J. Berg, M. Stilman, J. Kuffner, M. Lin, and D. Manocha, "Path planning among movable obstacles: A probabilistically complete approach," in *Algorithmic Foundation of Robotics VIII*, ser. Springer Tracts in Advanced Robotics, G. Chirikjian, H. Choset, M. Morales, and T. Murphey, Eds., 2009, vol. 57, pp. 599–614.

[13] M. R. Dogar and S. S. Srinivasa, "Push-grasping with dexterous hands: Mechanics and a method," in *Intl Conf on Intelligent Robots and Systems (IROS)*, 2010, pp. 2123–2130.

[14] J. King, J. Haustein, S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *Intl Conf on Robotics and Automation (ICRA)*, May 2015, pp. 2508–2515.

[15] S. Jentzsch, "Multi-modal path planning for solving abstract robot tasks," Master's thesis, Technische Universität München, Germany, January 2014.

[16] M. Rickert, "Efficient motion planning for intuitive task execution in modular manipulation systems," Dissertation, Technische Universität München, 2011.

[17] A. Gaschler, R. P. A. Petrick, M. Giuliani, M. Rickert, and A. Knoll, "KVP: A Knowledge of Volumes Approach to Robot Task Planning," in *Intl Conf on Intelligent Robots and Systems (IROS)*, November 2013, pp. 202–208.

[18] A. Gaschler, I. Kessler, R. P. A. Petrick, and A. Knoll, "Extending the knowledge of volumes approach to robot task planning with efficient geometric predicates," in *Intl Conf on Robotics and Automation (ICRA)*, May 2015, pp. 3061–3066.

[3] `http://www.james-project.eu/`

[4] `http://www.smerobotics.org/`

[5] `https://www.humanbrainproject.eu/`