A System for Multi-Step Mobile Manipulation: Architecture, Algorithms, and Experiments

Siddhartha S. Srinivasa[1], Aaron M. Johnson[1], Gilwoo Lee[1], Michael C. Koval[1], Shushman Choudhury[1], Jennifer E. King[1], Christopher M. Dellin[1], Matthew Harding[1], David T. Butterworth[1], Prasanna Velagapudi[1], and Allison Thackston[2⋆]

[1] Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA
[2] Toyota Motor Engineering & Manufacturing North America, San Jose, CA, USA
{ss5,amj1,gilwool,mkoval,shushmac,jeking,
cdellin,mharding,dbutterw,pkv}@andrew.cmu.edu,
allison.thackston@toyota.com
https://personalrobotics.ri.cmu.edu/

Abstract. Household manipulation presents a challenge to robots because it requires perceiving a variety of objects, planning multi-step motions, and recovering from failure. This paper presents practical techniques that improve performance in these areas by considering the complete system in the context of this specific domain. We validate these techniques on a table-clearing task that involves loading objects into a tray and transporting it. The results show that these techniques improve success rate and task completion time by incorporating expected real-world performance into the system design.

Keywords: Task and Motion Planning, Mobile Manipulation

1 Introduction

Our goal is to make robots that are capable of helping out around the house. We have previously demonstrated our robot HERB's *capability* to perform complex manipulation tasks, like loading a dishwasher [1], fetching drinks [2], and stocking a bookshelf [3]. However, a key criticism of our own work has been its lack of *reliability* in planning and execution. This is especially exacerbated in multi-step tasks, where the interplay between various steps, each of which are somewhat reliable, can prevent the robot from even finding a complete feasible path let alone executing it.

In order to enable the robot to more reliably plan these sorts of tasks, our first contribution is a multi-step planning system that ensures that the robot does not make poor decisions early on that prevent future steps from succeeding. This system uses simple heuristics to backtrack and work around problems. Furthermore, we present a simple extension to patch around errors detected at execution time.

⋆ This work was sponsored in part by the Toyota Motor Corporation.

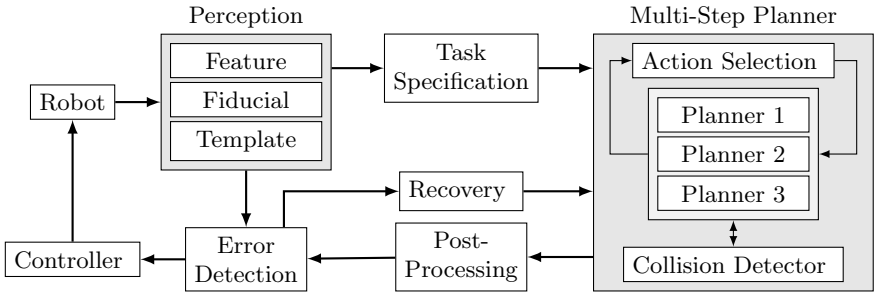


Fig. 1: Summary of the major components of the system.

While adding multi-step and recovery capabilities increases reliability, the system's extensive introspection produces excruciatingly slow plans. Several actions may have to be explored to produce a plan. In this way, reliability and speed are inexorably linked. In order to increase speed, and thereby enable more complicated tasks, our second contribution is a set of optimizations of the system. These include decomposing multi-step planning, motion planning, and post-processing, and sequencing multiple motion planners.

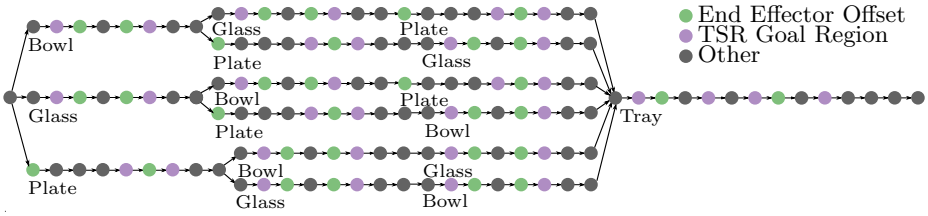
Inspired by recent work in benchmarking randomized motion planners [4], we ground our design decisions in the empirical performance of HERB doing a specific task: clearing a table. By doing so, we can expose the interaction of different components for a complete system in addition to testing each part individually. We show several examples of the importance of this system-level design, including the surprising result that in some cases the planner that is fastest on a given planning call may not lead to the fastest overall task completion.

Each component of our open-source[1] software system unifies contributions from existing work. The planning system employs multi-step planning algorithms [5,6] to decompose a task into individual components quickly and reliably solved by sequential calls to multiple planners [7]. We separate post-processing from motion planning [8,9] to speed planning time by only post-processing plans that will be executed. The perception system combines several existing algorithms [10–13], and decomposes the world into modeled objects and un-modeled clutter [14] while using physical constraints to refine object pose estimates [15]. We use perception to detect failure [16] and recover by locally modifying an existing solution [17].

2 Technical Approach

The software system summarized in Fig. 1 allows HERB to solve multi-step manipulation tasks. Prior experience with the general task and motion planning problem [5] reveals it is often difficult to specify the pre- and post-conditions necessary to elicit the desired behavior from a task planner. Instead, we require a user to specify a high-level task strategy and the system focuses on finding a

[1] https://www.personalrobotics.ri.cmu.edu/software



(a) Action Graph for the Table-Clearing Task

(b) Initial Setup (c) Object Transfer (d) Tray Lift

Fig. 2: The (a) table-clearing task requires HERB to load a (b) plate, bowl and glass, (c) into a tray, and (d) lift the tray for transport. This task requires different types of motion planners, and the order of transporting the objects is encoded as branches in the graph that may be explored by the multi-step planner.

geometrically feasible instantiation of the task. We use a table-clearing task as a running example throughout this paper, where the robot must load objects from a table onto a tray, then pick up and carry the tray away (Fig. 2).

The user specifies the task as a directed acyclic graph of actions. Each *action* has *parameters* that determine its feasibility. We encode high-level decisions, such as which object to move first, as special *parallel actions* that choose between a set of possibilities. The table-clearing task with three objects has 140 individual actions in the action graph (of which any complete solution path will consist of 40 actions) and, at its widest point, six parallel branches (Fig. 2a). A multi-step planner [18] uses a repertoire of geometric motion planners to search for a feasible sequence of action parameters. The resulting solution is then post-processed to improve its quality before execution. Perception is used both to setup the initial scene and also monitor execution in order to recover from errors by locally altering the solution.

Collision Detection. A key operation for motion planning or post-processing is to check if a given configuration is in collision either with itself or the environment. Testing configurations for collision can comprise a significant fraction of the planning time for complex manipulation tasks. Identifying and optimizing an efficient collision detector is therefore crucial for solving the task quickly.

There are many algorithm implementations that can solve the collision detection problem. Here, we consider three: *Open Dynamics Engine (ODE)*, a rigid body dynamics libray with integrated collision detection [19]; *Proximity Query Package (PQP)*, a triangle mesh distance computation and collision detection library [20]; and *Flexible Collision Library (FCL)*, a library for fast and accurate collision detection based on hierarchical representations [21].

In addition to the algorithm chosen, it is important to carefully consider how the algorithms are invoked. We implement two main optimizations. First, an *active-only* optimization restricts collision checks to only links of the robot affected by active degrees of freedom. All links of the robot that remain stationary (e.g. the left arm in Fig. 2c), may be checked just once prior to the action.

The second optimization performs a *pre-allocation* step to speed up collision checking of the repeated queries over identical link pairs characteristic of common motion planners. Consider the set of checks needed at each configuration as an undirected graph, with each vertex a link, and each edge a collision check. This optimization partitions the vertices (links) into sets with the same neighbors, and then pre-allocates the data structures for each pair of adjacent sets. This automatically coalesces rigidly attached objects in the scene (e.g. objects grabbed by the hand, or the table and robot base), and may thereby allow the underlying collision library to make better use of internal hierarchical scene decompositions.

Motion Planning. The most expensive operation in multi-step planning is solving the motion planning problems necessary to evaluate geometric feasibility of actions. Heuristic algorithms, such as connecting a start and goal with a straight line in configuration space or simulating the execution of a Jacobian psuedo-inverse controller, provide fast and efficient methods of finding solutions. However, they are prone to failure in cluttered environments, as is often the case in manipulation tasks. Trajectory optimizers [22, 23] solve more problems than these heuristic methods but can get stuck in local minima and fail in complex environments. Randomized motions planners [24, 25] most reliably solve difficult motion planning problems, but are often slower and produce poor-quality trajectories that are time-consuming to execute.

The key insight is that running all of these motion planners in a sequence allows us to quickly solve the easy problem instances without sacrificing the ability to solve difficult ones. The composition of this sequence is designed based on the overall system performance and not just the performance of the individual planners. A further simplification to the motion planning problems is to exploit kinematic redundancy and symmetries by specifying a continuous goal set represented as a *Task Space Region* (TSR), instead of a single goal configuration [26]. In particular, this is useful when planning grasps for objects, as often several valid grasps exist. The abundance of goals means that trivial solutions often exist [27] and can be found by faster planning methods early in the sequence.

Multi-Step Planning. The multi-step planner recursively searches the graph attempting to instantiate each action. The key challenge of multi-step planning is coping with dependencies between actions in the task, e.g. HERB cannot grasp a tray if he previously placed a glass near its handle. When this occurs, the planner must *backtrack* and choose a different parameter for the node associated with its placement of the glass. This is in contrast to a *hardcoded* strategy that fixes the task sequence and does not try to backtrack.

We introduce four heuristics for backtracking. *DFS* performs a depth-first search on the action graph, trying each action up to n times with different pa-

rameters before backtracking to the action's parent in the task's graph. *Restart* backtracks to the root of the graph upon encountering any failure. *Parallel-Restart* backtracks to the previous parallel-node instead of the overall root. *Frustration* lies somewhere in between these approaches by performing some local search, unlike *Restart*, while leveraging state to restrict the number of evaluations. It does this by recording the number of failures for a given node and backtracking faster when this value is large. For both *DFS* and *Frustration*, an action is only retried if its previous solution is deemed *non-deterministic*, i.e. re-planning the action will explore different parameters.

Post-Processing and Execution. Geometric motion planners typically return piecewise linear paths in configuration space. These paths are geometrically feasible, but lack timing information and may be of low quality. Before execution, a path is converted to a timed trajectory that satisfies HERB's dynamical constraints [8] and is locally optimized to reduce its duration [28]. To preserve geometric feasibility throughout this step, each modification is collision checked against the robot and the environment before accepting it. Consequently, post-processing is a computationally expensive operation.

However, there are two facts that can help mitigate this cost: many paths planned by the multi-step planner will never be executed and there is no interdependence between the post-processing of separate paths. As such, we consider three post-processing strategies. The first, *immediate*, runs post-processing just after planning each action. The second, *deferred*, delays post processing until after the full task plan has been finalized. By deferring post-processing, this strategy avoids post-processing plans generated for actions that are not included in the final task plan. The third strategy, *pipelined*, delegates post-processing and execution to separate threads. Here, once post-processing on a path is completed, it is sent to a parallel thread to be executed.

Finally, we introduce a fourth post-processing and execution strategy, *fully parallel*, that is like pipelined but also runs in parallel to the planning. To do this the action graph must be modified to identify a set of actions in the graph as *checkpoints*. These are actions that prevent the multi-step planner from backtracking further up the action graph. As soon as the multi-step planner finds geometrically feasible parameters for a checkpoint, the system can begin post-processing and executing all actions leading up to this checkpoint. This allows the robot to begin executing before planning is complete. Note that these checkpoints are incompatible with the *Restart* multi-step strategy, as once the execution has started the planner cannot change the first step.

Perception and Error Recovery. Planners only guarantee feasibility if perception provides them with an accurate model of the environment. HERB uses a repertoire of techniques to detect relevant objects in the environment and estimate their poses. We tailor the strategy to the specific properties of each object: feature matching for richly textured objects [11], template matching for textureless objects [12], and visual fiducials [10] for objects with challenging shapes or material properties. To avoid collision with objects not detected by these systems,

collision is checked against a mesh constructed from point cloud data [13]. Finally, the perception system creates a coherent model of the environment by enforcing non-interpenetration and stability constraints.

We also use perception to monitor execution. We annotate critical actions with validators that check for critical changes in the environment, i.e. an object to be grasped has been knocked over. When an action is deemed infeasible, the system will attempt to *recover* by locally altering the solution. For example, the system can replace the current plan with a new trajectory to reach a goal configuration if the previously planned trajectory is no longer collision free. If all attempts at local alteration fail, there is no choice but to invoke the multi-step planner again to generate a new solution.

3 Experiments and Insights

Our goal is to ground the design decisions in their ability to improve success rate and completion time for the table-clearing task described in Section 2. The experiments in this section use data recorded from execution of this task (Fig. 2a) to evaluate the efficacy of each component shown in Fig. 1.

Collision Detection. We collected a dataset of 259,135 collision checks from five executions of the task. Each collision check can be divided into two components: a *self* check that verifies that the robot has no links colliding with each other and an *environment* check that verifies that no links of the robot collides with another object. Fig. 3 shows the mean time of a collision check—as well as the breakdown of a check into these two components—for ODE, PQP, and FCL.

Our results show that FCL outperforms ODE and PQP by a factor of 6×. The active-only optimization, which culls a large number of redundant collision checks for the predominantly single-arm queries found in this task, yields an aggregate speedup of 17–77% across the algorithms (from FCL-PA to PQP, respectively). The pre-allocation optimization (FCL-PA) achieves a further speedup of 4.5×.

Fig. 3 highlights that the choice of collision checker has a large impact on computation time: the difference between ODE and FCL-PA is 27×. This performance is possible because the active DOFs and pre-allocation optimizations leverage the structure of collision checks performed during manipulation planning.

Motion Planning. As with collision detection, we record all the motion planning queries encountered across several runs of the table-clearing task and use those queries to benchmark a repertoire of motion planners. We extract the two most common queries, highlighted in Fig. 2a, for analysis.

End-effector offset queries plan a straight-line motion of the end-effector in the workspace. We consider the following constrained motion planners:

1. *Vector Field* (VF) numerically integrates a Jacobian pseudo-inverse controller that follows a vector field defined by the end-effector pose constraint.
2. *Greedy IK* (GIK) generates a workspace path, computes dense inverse-kinematics solutions, and interpolates between them in configuration space.

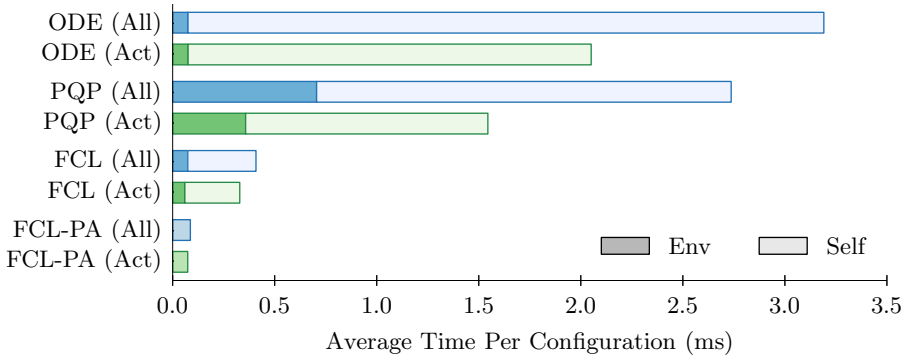


Fig. 3: The timing performance for the ODE, PQP, FCL, and FCL-PA on 259,135 collision checks recorded from the table-clearing task. Each pair of bars shows the effect of checking all links and only those moved by the active DOFs. Each stacked bar shows the time required for the environment and self-collision check. No such distinction exists for FCL-PA.

3. *CBiRRT* is a constrained bi-directional rapidly-exploring random tree [29].

TSR goal region queries plan an unconstrained trajectory to a continuous goal region specified by a TSR [26]. We consider the following planners and, when necessary, sample inverse-kinematic solutions from the TSR for initialization:

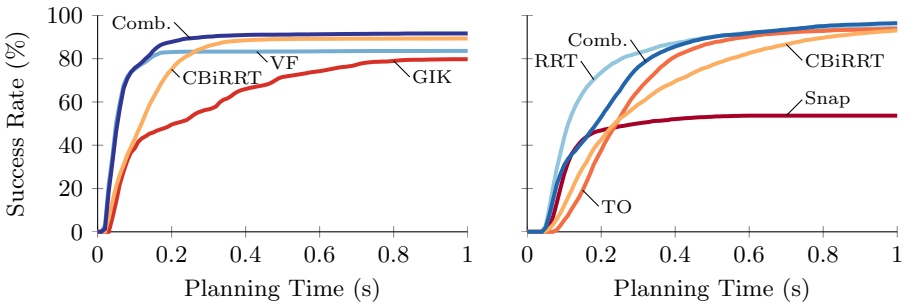
4. *Snap* returns the straight line in configuration space to a goal configuration.
5. *Trajopt (TO)* is a trajectory optimizer intended for motion planning [23].
6. *RRT-Connect (RRT)* is an unconstrained bi-RRT [30] in OMPL [31].

Fig. 4 shows the performance and Table 1 shows the relative success rate of these motion planners. As mentioned in Section 2, our insight is that running multiple planners in sequence will allow us to solve more problems than committing to any one algorithm. For end-effector offset queries, Vector Field is the fastest but does not achieve the highest success rate. Combining Vector Field and CBiRRT in a sequence (labeled as *Comb.*) is faster and more likely to succeed than any individual planner.

In contrast, TSR goal region queries show that RRT-Connect is faster and succeeds more often than all other planners[2]. However, trajectories returned by RRT-Connect require additional post-processing, are slower to execute, and introduce non-determinism into the action graph. We quantify the impact of these factors in the post-processing section by comparing RRT-Connect against a sequence that calls Snap, then Trajopt, then RRT-Connect (labeled as *Comb.*).

Multi-Step Planning. The multi-step planning results of running the table clearing task with the backtracking strategies introduced in Section 2 are shown in Fig. 5a. All of the backtracking strategies show significant improvement in success rate over the hardcoded task sequence. Surprisingly, the fastest backtracking

[2] Note, however, that these order critically depend on collision checking speed.



(a) End Effector Offset (b) TSR Goal Region

Fig. 4: Success rate as a function of planning time for (a) end-effector offset, where the — combined planner is both fast and highly successful, and (b) TSR goal region, where — RRT-Connect does best. However, see Fig. 5b for total task time results.

	VF	GIK	CBiRRT	Comb.
VF	83.8	2.7	8.2	8.2
GIK	6.5	79.9	9.6	12.0
CBiRRT	2.5	0.0	89.5	2.5
–	–	–	–	–
Comb.	0.0	0.0	0.0	91.9

(a) End Effector Offset

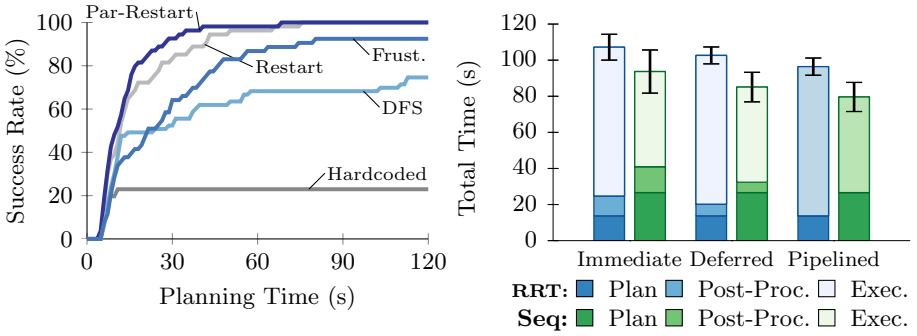
	Snap	TO	RRT	CBiRRT	Comb.
Snap	53.7	43.6	44.5	46.1	46.0
TO	2.2	95.1	4.6	4.7	4.6
RRT	1.5	3.1	96.6	3.2	3.1
CBiRRT	0.0	0.0	0.1	99.7	0.1
Comb.	0.1	0.2	0.2	0.2	99.6

(b) TSR Goal Region

Table 1: Success rate, in percent, of (a) end-effector offset and (b) TSR goal region planning queries using the planners described in the text. The main diagonal shows the success percent of each planner. An off-diagonal entry (i, j) shows the probability of planner j succeeding given that planner i has failed.

strategies are *Restart* and *Parallel Restart*. These strategies can suffer from a much more significant waste of planning time compared to the others because all progress in the search is discarded on each restart. However, in this task, the good performance of the *Restart* strategies can be attributed to two factors. First, the significant improvements in speed and success rate in collision checking and motion planning reduce the penalty of throwing away many planning calls. Second, many of the individual motion planning failures occur on difficult or impossible problems, e.g. an early choice for bowl placement in the tray prevents grasping the handle near the end of the task. The other strategies must backup many steps and replan large portions of the task to find a feasible solution. The *DFS* and *Frustration* strategies assume backtracking a small number of steps and selecting different parameters will lead to success, making them less suitable for the types of failures experienced in this task.

Post-Processing. Fig. 5b shows that deferring post-processing until multi-step planning is complete reduces total task completion time by 7 seconds by only



(a) Multi-Step Planning Strategies (b) Post-Processing

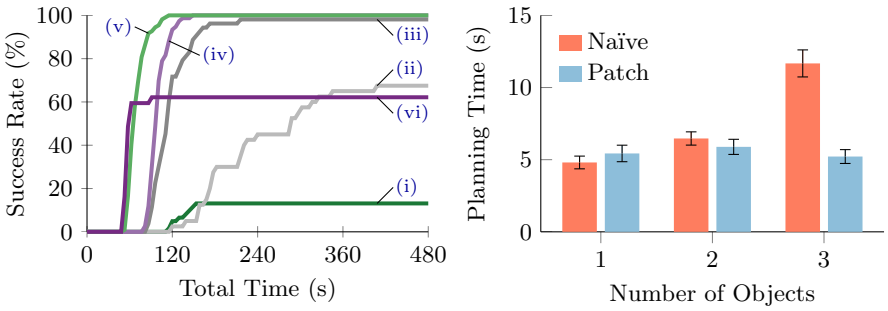
Fig. 5: (a) Success rate as a function of time for multi-step planning. All backtracking strategies outperform a — Hardcoded solution, with — Restart performing best. (b) Amount of time spent during ■■ planning, ■■ post-processing, and □□ execution. Deferred and Pipelined both reduce total time over Immediate. These results also show that ■ RRT-Connect (left bars) is faster at planning, but ■ Sequence (right bars) achieves lower total time by returning higher-quality solutions.

post-processing trajectories that are to be executed. Pipelining post-processing with execution leads to a further reduction of 6 seconds.

Additionally, Fig. 5b shows that the reduced planning time of RRT-Connect does not outweigh the cost of its lower-quality trajectories: the sequence including Snap and Trajopt achieves a faster task completion time. We performed this comparison after optimizing separate post-processing parameters for the sequence and RRT-Connect on a training dataset.

Overall Results. We incorporate the results from the collision checking, planning, and post-processing evaluations to examine the effects of each of these improvements on overall task success and planning time. We consider the following conditions:

(i) The first condition is a baseline that uses ODE on all links, a single planner (CBiRRT, as the only planner capable of solving all types of planning calls), a hardcoded task sequence, and immediate post-processing.
(ii) The next condition is the same but adds in the Restart multi-step planning.
(iii) Next the ODE collision checker is replaced with the best performing collision checking strategy, FCL-PA with *active-only*.
(iv) The next condition uses the fastest planning sequence for each type (Vector-Field and CBiRRT for End Effector Offset and RRT-Connect for TSR Goal Region) as well as pipelined post-processing.
(v) In this condition we consider changing the previous to use a sequence of Snap, Trajopt, and RRT-Connect for TSR Goal Region.
(vi) Finally, we modify the previous system to use the fully parallel post-processing and execution (i.e. starting execution in parallel with planning), and change to the Parallel-Restart multi-step strategy.



(a) Overall System Performance

(b) Error Recovery Strategies

Fig. 6: (a) Overall impact of the optimizations on planning, post-processing, and execution time. Conditions range from (i) — baseline to (v) — fully-optimized system. See the text for a detailed description of the conditions. (b) Recovery planning time with one, two, and three objects to transport. Unlike the naïve re-planning, patching is a local operation that does not take longer as the task becomes longer.

The distribution of complete trial times for these settings is shown in Fig. 6a. As expected from the individual component experiments in this section, the use of a multi-step planner improves success rate, as can be seen by comparing conditions (i) and (ii). The faster collision detection produces a dramatic speedup between (ii) and (iii). Using the optimal planning and post-processing strategy provides a further improvement to (v).

While the individual planner tests indicated that RRT-Connect is the fastest for TSR Goal Region plans, condition (iv) is actually slower than (v) due mostly to the increase in execution time that results from this randomized planner.

Finally, parallelization of planning and execution does speed up the total trial time from (v) to (vi), however the final success rate is reduced because the multi-step planner cannot backtrack once execution of an action has begun.

Error Detection and Recovery. We introduce failures into the experiments to measure the efficacy of the error recovery strategy. In order to patch to an existing solution the system must be able to plan to preconditions of a solution, which requires integrated task-and-motion planning. Here, we have selected simple preconditions that the system can plan to, e.g. matching the pose of the existing solution after moving an object. We compare this with a *naïve* recovery planner which re-plans the entire remaining task. Results shown in Fig. 6b suggest that a patch strategy is faster than a naïve strategy, whose planning time increases as the remaining number of objects increases. Since the patch strategy makes local solutions, it does not take longer even on more complex tasks.

4 Conclusion

The system design presented here is an empirically motivated software system for mobile manipulators performing complex tasks. This is a stepping stone between hard-coded sequences of actions and fully autonomous task and motion planning.

One key to the design of this system was taking an integrated view of all of the components. For example, the best performing motion planner for TSR Goal Regions (RRT-Connect) actually resulted in slower overall performance as the plans it produced required more time to post-process and execute. By iteratively improving and testing each component, we are able to identify the algorithms that best work *together* to solve the full task.

The strategies identified as best for table-clearing may not generalize across all problems. For many components, we could generate problems that reach very different conclusions. It is challenging to identify a universally optimal design, as the field lacks an operational notion of the space of all problems. Therefore, it is important to ground the design in a real example or a set of real examples. Actual performance on a problem is often just as important as theoretical bounds.

Our system requires the user to design an action graph that provides the multi-step planner the flexibility to explore multiple different solutions. This places the burden on the user to design a detailed and effective graph. In the future we plan to gradually reduce the granularity of the input action graph, instead allowing the user to specify higher-level goals and empowering the multi-step planner to expand these to sufficient action graphs.

References

1. Srinivasa, S., Ferguson, D., et al.: The robotic busboy: Steps towards developing a mobile robotic home assistant. In: Int. Conf. Intelligent Autonomous Systems. (2008)
2. Cakmak, M., Srinivasa, S., et al.: Human preferences for robot-human hand-over configurations. In: IEEE/RSJ Int. Conf. Intelligent Robots and Systems. (2011)
3. King, J., Klingensmith, M., et al.: Pregrasp manipulation as trajectory optimization. In: Robotics: Science and Systems. (2013)
4. Moll, M., Şucan, I.A., Kavraki, L.E.: An extensible benchmarking infrastructure for motion planning algorithms. CoRR **abs/1412.6673** (2014)
5. Kaelbling, L.P., Lozano-Pérez, T.: Integrated task and motion planning in belief space. Int. J. Robotics Research **32**(9–10) (2013) 1194–1227
6. Hebert, P., Bajracharya, M., et al.: Mobile manipulation and mobility as manipulation—design and algorithms of RoboSimian. J. Field Robotics **32**(2) (2015) 255–274
7. Bagnell, J.A., Cavalcanti, F., et al.: An integrated system for autonomous robotics manipulation. In: IEEE/RSJ Int. Conf. Intelligent Robots and Systems. (2012)
8. Bobrow, J.E., Dubowsky, S., Gibson, J.: Time-optimal control of robotic manipulators along specified paths. Int. J. Robotics Research **4**(3) (1985) 3–17
9. Geraerts, R., Overmars, M.H.: Creating high-quality paths for motion planning. Int. J. Robotics Research **26**(8) (2007) 845–863

10. Olson, E.: Apriltag: A robust and flexible visual fiducial system. In: IEEE Int. Conf. Robotics and Automation. (2011) 3400–3407
11. Pauwels, K., Kragic, D.: Simtrack: A simulation-based framework for scalable real-time object pose detection and tracking. In: IEEE/RSJ Int. Conf. Intelligent Robots and Systems. (2015) 1300–1307
12. Cao, Z., Sheikh, Y., Banerjee, N.K.: Real-time scalable 6-DOF pose estimation for textureless objects. In: IEEE Int. Conf. Robotics and Automation. (2016) 2441–2448
13. Klingensmith, M., Dryanovski, I., et al.: Chisel: Real time large scale 3D reconstruction onboard a mobile device. In: Robotics: Science and Systems. (2015)
14. Chitta, S., Jones, E.G., et al.: Mobile manipulation in unstructured environments: Perception, planning, and execution. IEEE Robot. Autom. Mag. **19**(2) (2012) 58–71
15. Correll, N., Bekris, K.E., et al.: Lessons from the Amazon picking challenge. CoRR **abs/1601.05484** (2016)
16. Gini, M., Smith, R.: Monitoring robot actions for error detection and recovery. In: Workshop on Space Telerobotics. (1987)
17. Edsinger, A.L.: Robot Manipulation in Human Environments. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (2007)
18. Shapiro, E.: A Hierarchical Framework for Configuration Space Task Planning. Masters Thesis, Carnegie Mellon University, Computer Science Department (2015)
19. Smith, R.: Open Dynamics Engine (ODE). http://www.ode.org/
20. Larsen, E., Gottschalk, S.: Proximity Query Package (PQP) (1999) University of North Carolina at Chapel Hill. http://gamma.cs.unc.edu/SSV/.
21. Pan, J., Chitta, S., Manocha, D.: FCL: A general purpose library for collision and proximity queries. In: IEEE Int. Conf. Robotics and Automation. (May 2012) 3859–3866
22. Zucker, M., Ratliff, R., et al.: CHOMP: Covariant Hamiltonian optimization for motion planning. Int. J. Robotics Research **32**(9–10) (2013) 1164–1193
23. Schulman, J., Ho, J., et al.: Finding locally optimal, collision-free trajectories with sequential convex optimization. In: Robotics: Science and Systems. (2013) 1–10
24. Kuffner, J.J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: IEEE Int. Conf. Robotics and Automation. (2000)
25. Berenson, D., Srinivasa, S.S., et al.: Manipulation planning on constraint manifolds. In: IEEE Int. Conf. Robotics and Automation. (2009)
26. Berenson, D., Srinivasa, S., Kuffner, J.: Task Space Regions: A framework for pose-constrained manipulation planning. Int. J. Robotics Research **30**(12) (2011) 1435–1460
27. Dragan, A., Ratliff, N., Srinivasa, S.: Manipulation planning with goal sets using constrained trajectory optimization. In: IEEE Int. Conf. Robotics and Automation. (2011)
28. Hauser, K., Ng-Thow-Hing, V.: Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In: IEEE Int. Conf. Robotics and Automation. (2010) 2493–2498
29. Berenson, D.: Constrained Manipulation Planning. PhD thesis, Carnegie Mellon University, Robotics Institute (May 2011)
30. Kuffner, J.J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: IEEE Int. Conf. Robotics and Automation. Volume 2. (2000) 995–1001 vol.2
31. Şucan, I.A., Moll, M., Kavraki, L.E.: The Open Motion Planning Library. IEEE Robot. Autom. Mag. **19**(4) (December 2012) 72–82 http://ompl.kavrakilab.org.