Perception-Constrained Robot Manipulator Planning for Satellite Servicing

Tariq Zahroof
Dept. of Mechanical Engineering
Stanford University
Stanford, CA 94305
281-940-9805
tzahroof@stanford.edu

Andrew Bylard
Dept. of Aeronautics and Astronautics
Stanford University
Stanford, CA 94305
925-978-6146
bylard@stanford.edu

Hesham Shageer
Center of Excellence for Aeronautics and Astronautics
King Abdulaziz City for Science and Technology
Riyadh 12354, Saudi Arabia
904-679-1213
hshageer@kacst.edu.sa

Marco Pavone
Dept. of Aeronautics and Astronautics
Stanford University
Stanford, CA 94305
650-723-4432
pavone@stanford.edu

***Abstract*— Satellite servicing is a rapidly developing industry which requires a number advances in semi- and fully-automated space robotics to unlock many key servicing capabilities. One upcoming mission example is the NASA Restore-L Robotic Servicing spacecraft, which is equipped with two 7-joint robotic manipulators used to capture a satellite and perform a complex series of refueling tasks, including swapping between various end-effector tools stored on board. In this scenario, planning of the manipulator motions must account for a number of constraints, such as collision avoidance and the potential need for uninterrupted visual tracking of objects or of the end-effector. Such complex constraints in a cluttered environment, such as the interface between two spacecraft, are time-consuming to incorporate into hand-designed trajectories. Thus, in this work we present a software tool which uses robot motion planning and path refinement algorithms for automated, real-time computation of near-optimal, collision-free trajectories which satisfy the aforementioned perception constraints. The tool is built on the ROS MoveIt! framework, which can simulate and visualize trajectories as well as seamlessly switch between motion planning and refinement algorithms depending on task requirements. Additionally, we performed experimental campaigns to benchmark a number of available algorithms for performance in handling such perception constraints. Although the framework is applied to a mock-up of Restore-L satellite servicer in this paper, the tool can be applied to any fixed-base manipulator planning scenario with a similar class of constraints.**

TABLE OF CONTENTS

978-1-5386-6854-2/19/$31.00

1. INTRODUCTION

In its history and through to the present, the space industry continues to lose billions of dollars as a result of satellite system failures [1] and the decommissioning of functional satellites due to depletion of fuel [2]. This presents an open opportunity for on-orbit satellite servicing, a broad field incorporating rendezvous, docking, and a variety of satellite repair and maintenance tasks. Currently, due to the high cost of crewed missions in space, satellite servicing operations often rely on robotic tools to complete tasks [3]. Thus, a robust and agile workflow between ground operators and servicing robots is necessary to make full use of available robotic capabilities, reduce operator workload, and ultimately reduce costs. In particular, adding more autonomous or semi-autonomous functionality to these robotic tools is crucial to streamlining operations.

To perform tooling and servicing tasks, space servicing technologies have often incorporated manipulator systems for complex object and environment interaction. Early 6-degree-of-freedom (DoF) systems, such as the Canadarm and the European Robotic Arm (ERA) [4], [5], provided basic SE(3) end-effector interaction using the minimum number of manipulator joints. These led to later systems, such as the 7-DoF Canadarm2 and the 15-DoF Special Purpose Dexterous Manipulator (SPDM) for the International Space Station, using the additional DoF to improve avoidance of singularities and collisions [6], [7]. Unfortunately, these more recent manipulator systems still make use only of astronaut tele-operated control or pre-designed trajectories, forcing operators to personally direct task-specific end-effector movements around obstacles [8]. Designing these trajectories is both difficult and tedious, especially when operating in cluttered environments and given redundancy in the manipulator.

In some areas of on-orbit satellite servicing, researchers are continuing to investigate the design and implementation of autonomous capabilities. For example, research in Autonomous Rendezvous and Capture (AR&C) aims to perform the entire process of localization, navigation, capture, and stabilization of target satellites with minimal human intervention [9]. This includes operations such as the motion synchronization and capture of non-cooperative target objects with a chaser spacecraft [10], [11], [12]. However, while AR&C is an ongoing, active field of research, there has been limited work on implementing autonomy on robot manipulators to complete satellite servicing tasks after a target object has been

captured.

Fortunately, within the broader robotics community, such automated algorithms for guiding robotic manipulators are readily available. For example, in early work, Khatib introduced artificial potential fields to model obstacles as repulsive surfaces within the classical manipulator torque control framework [13]. This approach had some success for simple obstacle avoidance, but in more cluttered environments, the method gets stuck in local minima, halting manipulator movement or leading it into dead ends. Later work includes the broad field of trajectory optimization, which makes use of advances in convex and nonconvex solvers to plan new trajectories or refine coarse trajectories. Well-known algorithms of this class include CHOMP [14], STOMP [15], and TrajOpt [16]. These approaches have been quite successful in providing low-cost manipulator trajectories in environments which require collision avoidance, sometimes even when seeded with simple, naive initializations. However, their dependence on initialization is quite severe, and when planning in very cluttered environments or on complex dynamical systems, the task of finding a seed trajectory that leads to a successful or desirable solution is a difficult one on its own [17].

Another available option is sampling-based planning (SBP), which performs a global search for near-optimal, collision-free robot trajectories. SBP algorithms sample collision-free robot configuration states (also referred to as nodes) from the configuration space of the robot and then connect the nodes with subpaths (referred to as edges) validated by a collision-checker. The scheme for choosing nodes and edges is often designed to result in full trajectories that optimize some cost function. In addition, since the collision-checker can be considered as a black-box within these algorithms, SBP is amenable to robot planning problems having a wide variety of complex constraints such as collision-avoidance. SBP has been particularly successful for high-dimensional robots requiring fast computation of collision-free geometric paths [18]. Within space robotics, SBP has historically been used for tasks such as attitude control [19], docking [20], multi-robot control [21], and manipulator control in the pre-capture phase [22], but not for fine robotic satellite servicing tasks. Accordingly, a key objective of this paper is to investigate the use of SBP in the context of manipulator planning for satellite servicing tasks.

Importantly, SBP approaches alone are not always sufficient for generating desirable trajectories, as although they are adept at quickly generating coarse, near-globally-optimal paths, the full refinement of trajectories to optimality via SBP algorithms can be very computationally intensive. As such, various post-processing techniques are often used for path refinement. Such post-processing can be accomplished using the class of previously discussed trajectory optimization algorithms. However, quick quality improvements can also be achieved for a SBP-generated path via simple approaches such as path-pruning and shortcutting by removing unnecessary nodes and performing simple interpolations [23].

When designing trajectories for spacecraft manipulation tasks, there are often special requirements desired by mission designers and operators. A key example that we address in this paper is the ability to enforce desired camera visibility constraints throughout a manipulator trajectory. For example, it may be desired that a supervising camera can always keep the manipulator end-effector in view or that an end-effector-mounted camera keep a particular target in view throughout a maneuver. Though previous work has incorporated visual information into manipulator control (e.g. visual-servo control, using camera data streams within a manipulator position control loop [24], [25], [26], [27]), it has not been used for planning of optimal nominal trajectories. In particular, the consideration of hard perception constraints of this sort within planning for robotic spacecraft systems, alongside other constraints such as collision avoidance, has not yet been addressed. Thus, an additional key objective of this paper is to incorporate such perception constraints naturally into a planning pipeline while maintaining fast planning times.

Statement of Contributions

The goal of this paper is to provide a planning tool to compute manipulator motions for satellite servicing tasks, with a key emphasis on accounting for perception constraints and avoidance of complex obstacles. Specifically, the contribution of this paper is twofold. First, we formulate field-of-view and line-of-sight perception constraints within a SBP framework to generate trajectories which guarantee that specified cameras retain view of desired targets throughout a robot manipulator maneuver. Second, by leveraging such a formulation, we present a SBP and trajectory refinement pipeline and tool to rapidly generate collision-free trajectories for a satellite-servicing robotic manipulator, providing detailed simulation data to evaluate and select the best combination of SBP and trajectory refinement algorithms. The planning tool can be applied to a variety of satellite servicing tasks in the phase after docking with the satellite has been achieved; in this paper, we use a mock-up of the Restore-L satellite servicing mission scenario as an environment for evaluating the tool.

Paper Organization

The paper is organized as follows: In Sec. 2 we review the motion planning problem solved by the tool and detail the SBP and trajectory refinement algorithms evaluated for use in the planning pipeline. Then, in Sec. 3, we discuss the perception constraints and their integration into the planning process. Next, in Sec. 4, we provide a high-level overview of the manipulator trajectory generation tool, which is the main contribution of the paper, as well as further implementation details of the tool. Sec. 5 presents simulations evaluating SBP and refinement algorithm combinations in various environments, and Sec. 6 discusses the experimental results. Finally, Sec. 7 provides the conclusions and directions for future research.

2. PROBLEM FORMULATION AND ALGORITHMS

A. The Motion Planning Problem

SBP algorithms generate plans by connecting feasible states of the robot in the configuration space. The configuration space, or C-space, represents all possible configurations of the manipulator (i.e. combinations of joint angles). The C-space is further divided into two regions: the obstacle-space $\mathcal{C}_{\text{obs}}$, representing infeasible robot states due to collisions, and its complement, the free-space $\mathcal{C}_{\text{free}}$. To clarify, collision states do not only include undesired contact between the robot and its environment, but also violation of other problem constraints (e.g., breaking visual line-of-sight with a perception target).

The motion planning problem is defined as the determination of a robot action trajectory $u(t)$ yielding a feasible C-space path $x(t)$ such that $x(t) \in \mathcal{C}_{\text{free}}$ within a specified time horizon $t \in [0; T]$ and also satisfying $x(0) = x_{\text{start}}$ and

$x(T) \in X_{\text{goal}}$ [28]. In other words, a successful motion planner will provide a feasible trajectory linking the starting configuration x_{start} to the goal region X_{goal} within a specified amount of time T. A motion planner may additionally attempt minimize a cost $J = \int_0^T g(x,u)dt$, where $g(x,u)$ penalizes motions by some criteria (in this paper, we consider path length in the manipulator joint space). Thus, an optimal motion planner will provide a set of controls, $u(t)$ that produces the successful trajectory having a globally minimal total cost.

B. Sampling-Based Motion Planning Algorithms

There exists a wide variety of sampling-based motion planning algorithms, amenable to many different applications. For this work, we select three of the most commonly used classes of SBP algorithms, described at a high level below. Of these, five total variants are chosen for simulation comparisons, as described in Sec. 5.

Rapidly-Exploring Random Trees (RRT)— RRT is an SBP algorithm that performs simultaneous graph construction and search, incrementally building a tree of feasible paths through the space. At each step, RRT randomly samples nodes within the C_{free} and attempts to connect the node to the closest feasible node in the tree [29].

We use two variants of RRT for benchmarking. The first variant is RRTConnect, a bi-directional version of RRT which grows two trees from the start and goal towards one another in order to reduce overall planning time [30]. Second, we use RRT*, an asymptotically optimal version of RRT that converges towards the globally optimal solution as the number of samples increases through a process of rewiring connections between nodes when a new node is connected to the tree [31].

Probabilistic Roadmaps (PRM)—PRM takes a different approach from RRT, first representing the free space by sampling a large batch of collision-free nodes and connecting nodes to all of their k-nearest neighbors with collision-free edges where possible. This creates a graph or "roadmap," which can be queried for optimal paths using standard graph-search algorithms (e.g. Djikstra's) [32]. Furthermore, PRM*, an asymptotically optimal variant, improves path quality and uses fewer collision-checks by using a particular scaling of k for the k-nearest neighborhood construction, chosen as a function of the size of the sample set [31]. In this case, the main computational bottleneck is the full creation of a collision-free roadmap before beginning a path search. This is often wasteful since an exhaustive roadmap is not required for most planning problems, and if obstacles or constraints change, the roadmap must be regenerated.

Fast Marching Tree (FMT)*— Like RRT, FMT* performs graph construction and search simultaneously. However, FMT* makes use of a "lazy" collision-checking technique, reducing the total number of computationally expensive edge collision-checks needed to solve a planning problem compared to RRT and PRM variants. In order to maintain the same guarantees of asymptotic optimality as RRT* and PRM* while using this technique, FMT* operates on a fixed batch of collision-free nodes sampled in advance [33].

We use FMT* and a bi-directional variant of FMT* called BFMT*. Like RRTConnect, BFMT* grows two trees, one from the start and one from the goal, towards each other [34]. Using this approach, the algorithm is able to find solutions using two small trees that connect somewhere in the middle of the search space, rather than using a single large tree that must expand the entire way from the start to the goal. Since the number of actively exploring nodes (referred to as the search frontier) tends to grow with the size of a tree, the dual-tree approach of BFMT* tends to find solutions with a much smaller frontier than FMT*, thereby decreasing computational expense.

The algorithm can use different strategies for alternating between the two trees, such as by taking turns ("Alternating Trees") or by operating on the tree with the lowest-cost frontier node from the tree's root ("Balanced Trees"). Additionally, the algorithm can be either be specified to terminate on the first connection between the two trees returned ("First Path") or when the trees have expanded sufficiently far into one another ("Best Path"). We use the Balanced Trees and Best Path combination as a balance between computation speed and path quality.

C. Shortcutting Refinement Algorithms

Additionally, a variety of shortcutting methods are available for refining coarse paths generated by SBP algorithms. The simplest shortcutting methods are path-pruning techniques, which only remove single nodes in a path and attempt to reconnect the remaining nodes. However, there are a number of more complex variants, of which we chose four for comparison.

Shortcut— The first shortcutting technique, Shortcut, improves upon simple path-pruning techniques by attempting to optimally connect two random nodes along a robot joint trajectory via a straight line. If the path is feasible, the former intermediate nodes are discarded, and the shortcut is cemented with interpolated points [23].

Adaptive Shortcut—The Adaptive Shortcut method builds on Shortcut by including an oracle to assist in reducing clearance between the manipulator joints trajectory and constraints in order to further decrease path length [35]. After completion of a Shortcut routine, the oracle inserts additional nodes along the new path and starts another Shortcut routine. The additional inserted points give the Shortcut routine new vantage points from which to find interpolated paths that go through the free region while cutting closer to constraints. This results in better final path quality at the cost of extra computation due to running the Shortcut algorithm multiple times.

Partial Shortcut—The Partial Shortcut method improves path optimization over Shortcut by employing Shortcut on manipulator joints individually. After selecting two random nodes $x^{(i)}$ and $x^{(j)}$ along a trajectory and selecting a specific joint k to shortcut, Partial Shortcut replaces the values $x_k^{(i)}, x_k^{(i+1)}, \ldots, x_k^{(j)}$ representing joint k's trajectory from nodes $x^{(i)}$ to $x^{(j)}$ with a linear interpolation between $x_k^{(i)}$ and $x_k^{(j)}$. Due to employing more incremental shortcutting along single manipulator joints, Partial Shortcut provides higher quality paths than Shortcut, at the cost of more iterations.

Adaptive Partial Shortcut—The final shortcut method, Adaptive Partial Shortcut, is a variant of Partial Shortcut that prioritizes manipulator joints which contribute the most cost to the full trajectory [36]. Specifically, in each iteration, Adaptive Partial chooses a joint to optimize according to a weighted distribution, where higher probability is assigned to joints which contribute higher additional cost across the current trajectory, relative to the optimal obstacle-free trajectory from

the start state to goal region. Using this approach, Adaptive Partial can often produce high cost reduction in the first few iterations compared to other algorithms. However, in the end, the algorithm may also get stuck prioritizing and repeatedly choosing joints that contribute towards overall trajectory cost but cannot be optimized further, thus wasting iterations.

3. Formulating the Perception Constraints

The objective of the perception constraints is to guarantee that generated manipulator motions maintain visual tracking of desired targets. One such constraint is an "eye-in-hand" perception constraint, which provides continual visibility of a target in the environment from a camera mounted on the manipulator end-effector. This would allow operators to perform an additional visual check of trajectory accuracy and gain better insight of target-manipulator interactions during complex task sequencing. Another perception constraint is an "eye-to-hand" constraint, where the camera is attached somewhere else in the environment, such as on a secondary arm or on the body of the servicing spacecraft. Here, the goal is to restrict end-effector movement to within the field-of-view of the camera. Similarly, this limitation provides increased supervision and understanding for an observing operator. Both constraints require constant field-of-view (Figure 1) and line-of-sight (Figure 2) of the target.

To enforce these constraints within a SBP pipeline, we simply add them to the collision-checking process. As such, samples are considered in collision not only when the manipulator is in undesired contact with an obstacle, but when either of the perception constraints are violated. In particular, for each camera within a scenario, the following two types of constraints are formulated and applied:

A. Field-of-View Constraint

An example diagram of a field-of-view cone constraint can be seen in Figure 1. Let p represent the camera's location and t represent the target location. If $\hat{u}$ represents the unit vector along the direction the camera is facing, then the vector l along the camera's normal vector to the target is

$$l = [(t \quad p) \quad \hat{u}]\hat{u}:$$

The vector d to the target point within a plane normal to the camera's direction is determined by

$$d = t \quad (p + l):$$

Thus, the cone constraint is satisfied if the angle to the target is within the field-of-view cone, defined by the angle :

where $= \tan^{-1} \frac{\|d\|}{\|l\|}$.

B. Line-of-Sight Vision Constraint

To implement the line-of-sight constraint, a thin rectangular prism is added to the end-effector to ensure no obstacles are present between the target and the camera. The following calculations show how to transform an arbitrarily-oriented thin rectangular prism with length $kt \quad pk$ to span the length between the target and the camera, as shown in Figure 2:

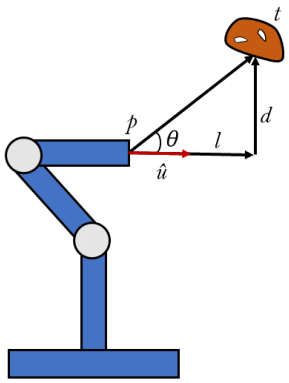


Figure 1: Diagram of a field-of-view cone constraint from an end-effector camera to a target.

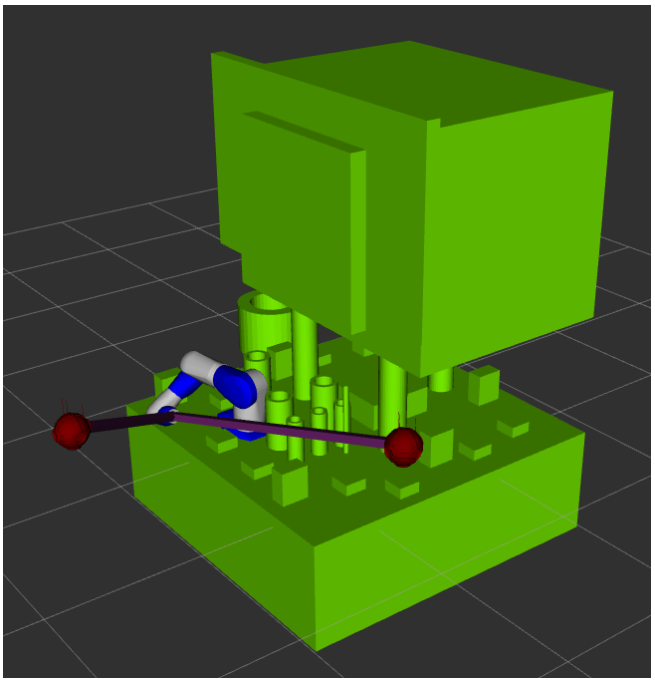


Figure 2: Line-of-sight perception constraints are shown by the purple rectangular prisms stretching from the end-effector to the target (the object the end-effector is facing) and to a camera mounted in the environment.

Let $\hat{z}$ represent the unit direction of the long axis of the prism upon creation and before rotation. The axis of rotation $\hat{a}$ and the angle of rotation to correctly orient the prism are then found by

$$\hat{a} = \hat{z} \quad \frac{t \quad p}{kt \quad pk};$$

$$= \cos^{-1} \quad \hat{z} \quad \frac{t \quad p}{kt \quad pk} \quad :$$

Then, the quaternion q rotating the prism into proper orientation is calculated by

$$q = \begin{matrix} \hat{a}\sin(\frac{}{2}) \\ \hat{a}\cos(\frac{}{2}) \end{matrix} :$$

After the rotation the prism is translated into its position between the target and the camera.

4. The Perception-Constrained Motion Planning Tool

The motion planning pipeline developed in this work is built on the ROS MoveIt! library in order to easily interface the shortcutting post-processing techniques with existing Open Motion Planning Library (OMPL) SBP algorithms. For further specialization to our problem setting, we modified the core MoveIt! collision-detection module to add optional perception constraints for target and/or end-effector tracking during motion execution. The workflow of the tool is depicted in Figure 3, and the code is available at `https://github.com/StanfordASL/PerceptionConstrArmPlanning`.

The tool's modularity allows for the integration of custom motion planners and pairings of SBPs and refinement algorithms. Additionally, although the tool is applied in this paper to a satellite-servicing arm on a spacecraft berthed to a servicing target, the tool can be readily applied to any fixed-base robot planning scenario. Definitions of robots and environments can be imported through URDF files, and by starting robot configurations and goal regions are updated, operators or high-level decision-making modules can repeatedly query the tool for rapid planning, as demonstrated in our experiments. Generated trajectories can also be viewed through RViz for final verification by ground operators.

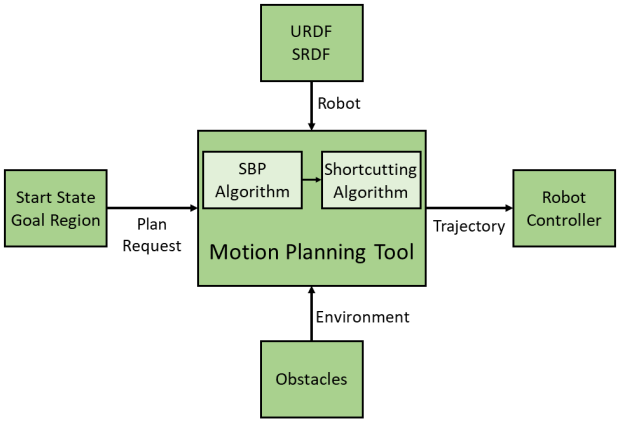


Figure 3: Diagram showing the tool's motion planning workflow.

To quickly generate trajectories, MoveIt! solves a kinodynamic planning problem which is divided into two stages. During the first stage, MoveIt! determines path nodes (manipulator joint configurations) using geometric straight-line planning. Then, the path is turned into a full trajectory using time parameterization, scaling in time in order to satisfy dynamics constraints. To allow finer velocity and acceleration control, the tool interpolates additional waypoints into the trajectory before time parameterization. As there are a number of algorithms implemented and available for use in this tool, the next two sections benchmark a number of SBP and refinement algorithm combinations, in order to provide guidelines on algorithm selection.

Environment	Max Time (s)
Sphere	1.0
Box	1.0
Cluttered	3.0
Maze	11.0
Satellite	22.0

Table 1: Enforced maximum computation time for planning environments.

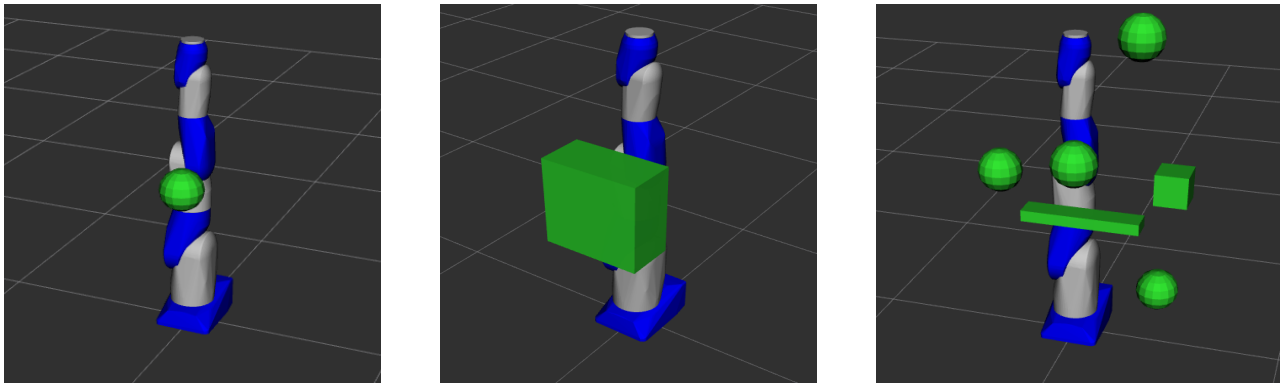
5. Experimental Setup

To test the planning pipeline and evaluate different SBP and refinement algorithm combinations, we used a model of the Motoman SIA20D robot arm, which has the same joint configuration and similar scale to the Restore-L servicer arms and can thus serve as an effective analog in simulation. Specifically, the 7-DoF arm incorporates revolute joints in a standard shoulder-elbow-wrist configuration, with a redundant joint in the elbow. We placed the manipulator in five different planning environments: (1-3) three simple environments without perception constraints applied, (4) a maze of floating rubble with an eye-in-hand perception constraint applied, and (5) a mock-up of a satellite with both an eye-in-hand and eye-to-hand perception constraint applied. The three simple environments (a sphere environment, a box environment, and a cluttered environment, as shown in Figure 4), served as isolated test cases to provide baseline statistics on the planners in simple planning tasks. In particular, the sphere environment offered minimal obstruction to demonstrate small obstacle-induced path perturbations, the box environment forced the manipulator to stretch while moving around the box's extremities, and the cluttered environment offered a variety of pathways through which the manipulator could reach the goal. Conversely, the maze of space rubble, shown in Figure 5, challenged the robotic arm to navigate around obstacles to a new vantage point while maintaining perception of the target. This leads to the final application, the Restore-L mission mock-up, where the manipulator was tasked with maneuvering to a tooling location while maintaining vision of a target in the environment. Additionally, the end-effector was further constrained to operate within the vision of a camera mounted in the environment. As such, the robot needs to perform a cork-screwing motion while moving forward to settle near the tooling location, guaranteeing constant end-effector and target visibility.

As mentioned in Sec. 2, five SBP algorithms were considered for comparison: BFMT*, FMT*, RRTConnect, RRT*, and PRM*. After planning, the SBP solutions were fed into one of four different shortcutting methods for trajectory post-processing. Each algorithm and shortcutting pair were run 100 times and the results were averaged. Additionally, we imposed a maximum time limit to prevent excessive planning times, as shown in Table 1.

6. Results and Analysis

Using the results of our experiments, we evaluated SBP and shortcutting algorithm pairs to determine the best combination for manipulator arm planning for different types of satellite-servicing scenarios. SBP algorithms were evaluated by computation time, path quality, and failure rate, and shortcutting techniques were evaluated by computation time and path quality improvement. As shortcutting techniques performed similarly across all experiments, they are discussed at



(a) Sphere environment. (b) Box environment. (c) Cluttered environment.

Figure 4: Simple environments with no perception constraints.

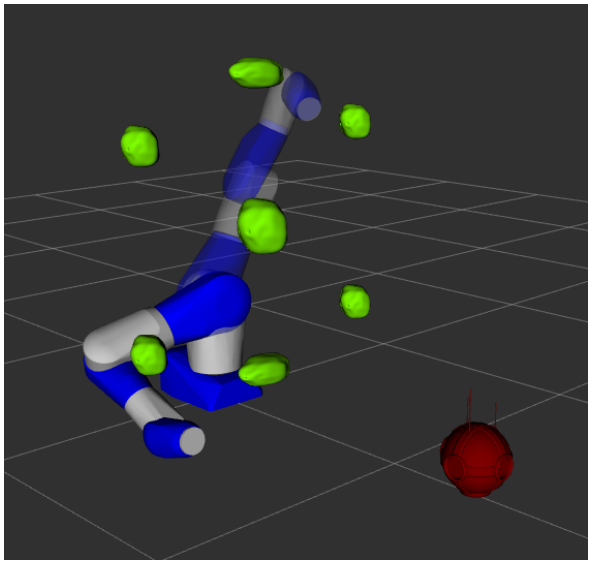


Figure 5: Maze environment with target perception constraint.

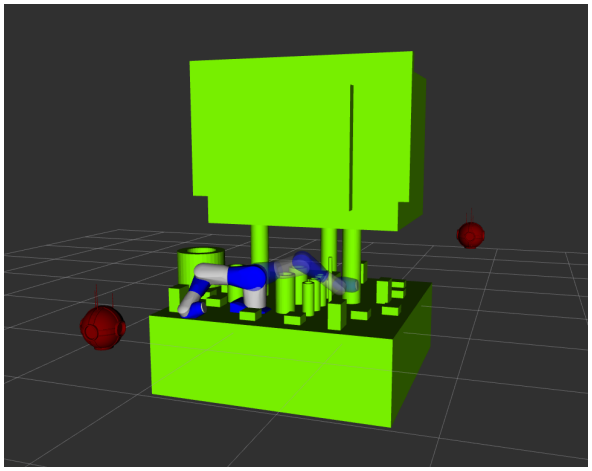


Figure 6: Satellite environment with target and environmental camera constraint.

the end of the section.

A. Three Simple Environments (No Perception Constraints)

The trajectory cost and computation time results for the SBP algorithms in the simple environments are summarized in Figures 7 and 8, respectively. RRTConnect produced the fastest solutions using less than 10% of the computation of its closest competitor, BFMT*. However, RRTConnect paths also had the poorest quality, due to the algorithm's emphasis on quickly finding a feasible path rather than refining its search to seek optimality. On the other end of the spectrum, RRT* produced the highest quality paths but had the longest computation times. These excessive times can be attributed to RRT*'s excessive collision-checking both while searching for initial node connections and during the subsequent rewiring step.

Alternatively, BFMT* and FMT* struck a balance between RRT* and RRTConnect both in terms of path quality and time. In particular, their resulting path quality was only marginally lower than RRT*, but they had significantly faster computation times in the more challenging cluttered environment. However, of the two, BFMT* was the most promising, producing better quality paths up to twice as fast as FMT*.

B. Maze Environment (Eye-in-Hand Constraint)

For the maze environment, the SBP results are provided in Figure 9. It is worth noting that in this and the satellite environments, the added perception constraints and obstacle complexity resulted in some rate of failure for each of the algorithms, due to their inability to find any feasible solution within the maximum allotted planning times, shown in Table 1. Indeed, the excessive collision-checking of RRT* during node connection and rewiring caused it to rarely succeed within the enforced time limits, resulting in over a 70% failure rate. Thus, RRT* was replaced by PRM* as a SBP algorithm for evaluation in these scenarios.

As in the simple environments, RRTConnect produced the lowest quality trajectories by a significant margin, although it produced solutions the fastest and thereby had the lowest failure rate. PRM* improved path quality over RRTConnect, but was the slowest and had the highest failure rate. Indeed, FMT* outperformed PRM* by every metric, though it was in turn outperformed by BFMT*, which required only half the planning time, while trading off with a slightly higher failure rate.

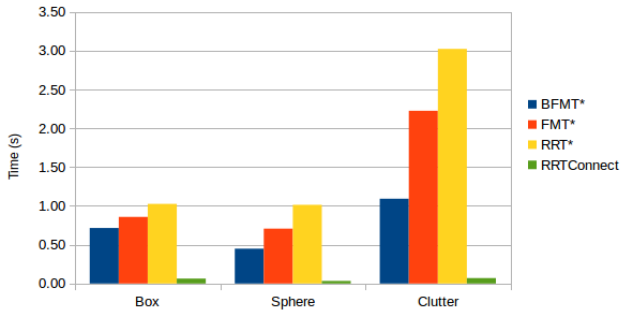


Figure 7: Average computation time of the SBP algorithms for the simple environments.

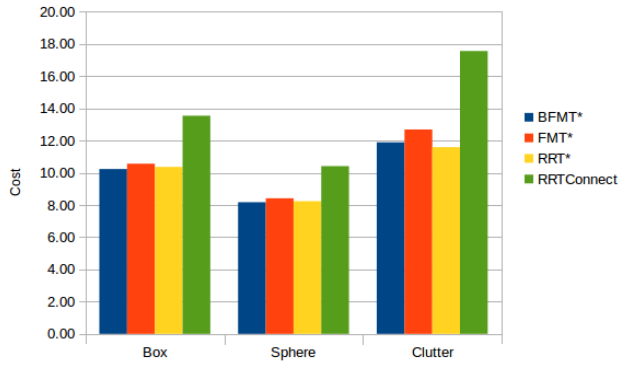


Figure 8: Average trajectory cost of the SBP algorithms in the simple environments.

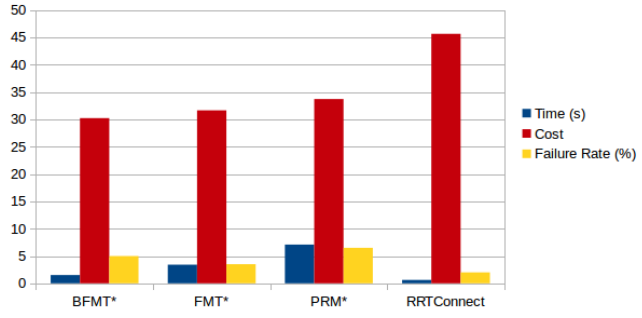


Figure 9: Average SBP computation time, trajectory cost, and failure ras for the maze environment.

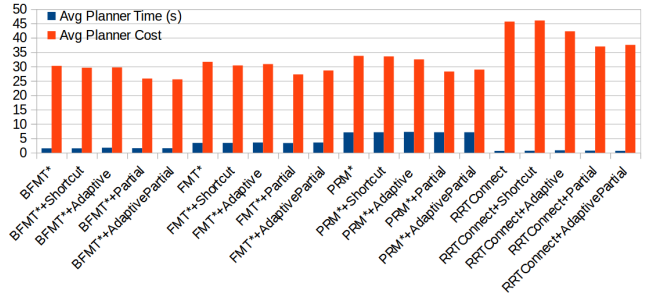


Figure 10: Average computation time and trajectory cost for SBP with shortcutting for the maze environment.

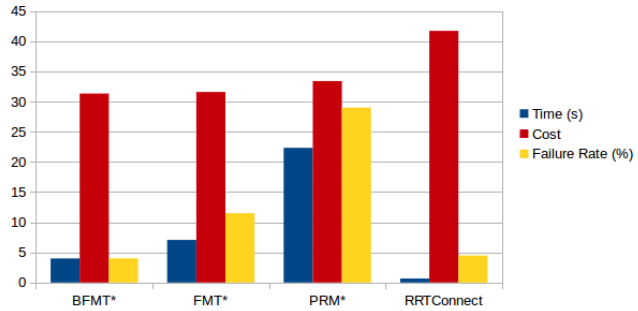


Figure 11: Average SBP computation time, trajectory cost, and failure rate for the satellite environment.

C. Satellite Environment (Eye-to-Hand and Eye-in-Hand Constraints)

The satellite environment induced the heaviest strain on the SBP algorithms, incorporating four perception constraints (two field-of-view and two line-of-sight) alongside a very complex obstacle set. As a result, planning times were longer and failure rates were significantly higher than in the maze environment. In particular, PRM* experienced a high 29% failure due to number of edge collision-checks required in the initial construction of a roadmap. This failure rate and the associated slowdown was avoided by FMT*, which draws samples in the same fashion as PRM*, but has a significant advantage in cluttered and tightly constrained environments such as this due to the algorithm's "lazy" collision-checking technique. However, FMT* was again outperformed by BFMT* in this environment, this time by all metrics.

D. Shortcutting Results

The shortcutting algorithms produced consistent results in comparison to each other across all environments, as shown in Tables 2-4. Shortcut and Adaptive Shortcut provided the lowest path quality improvement as well as the highest computation time. On the other hand, Partial and Adaptive Partial provided much higher path cost reduction, for example up to 23.8% and 22.9%, respectively, in the simple environment. This demonstrated the importance of Partial's and Adaptive Partial's freedom to refine the subpaths of single manipulator joints individually rather than shortcutting over full-state joint subpaths. In contrast, in Shortcut and Adaptive Shortcut, a collision caused by one manipulator joint in a potential shortcutting subpath would immediately invalidate it, negating the remaining potential for refinement along the other joints present in that subpath.

As might be expected, the shortcutting algorithms had reduced effectiveness in scenarios that were more tightly constrained by obstacle complexity and perception constraints. By reducing the number of eligible collision-free states available in the sample space, the perception constraints made straight-line replacements more difficult, thus diminishing the cost reduction locally available to the algorithms. Indeed, the shortcutting algorithms produced half the cost reduction in the satellite environment (two perception constraints) compared to the simple cluttered environment (no perception constraints).

Overall, the computation time of the shortcutting techniques was trivial compared to that of the SBP algorithms, and given their often significant trajectory cost reduction, it is clear that it is worth including shortcutting within the planning pipeline.

Shortcutting Algorithm	Box		Sphere		Clutter	
	Time (s)	Cost Red. (%)	Time (s)	Cost Red. (%)	Time (s)	Cost Red. (%)
Shortcut	0.04	5.0	0.05	4.8	0.06	5.5
Adaptive	0.17	8.2	0.22	7.5	0.30	9.6
Partial	0.02	18.9	0.01	14.8	0.03	23.8
Adaptive Partial	0.01	19.0	0.01	13.2	0.03	22.9

Table 2: Average computation time and cost reduction results from using shortcutting algorithms for SBP solution path refinement in the simple environments.

Shortcutting Algorithm	Time (s)	Cost Red. (%)
Shortcut	0.05	2.5
Adaptive	0.17	5.4
Partial	0.05	14.4
Adaptive Partial	0.04	13.4

Table 3: Average path refinement computation time and cost reduction for the maze environment .

Shortcutting Algorithm	Time (s)	Cost Red. (%)
Shortcut	0.16	2.1
Adaptive	0.65	3.9
Partial	0.14	11.3
Adaptive Partial	0.11	9.8

Table 4: Average path refinement computation time and cost reduction for the satellite environment.

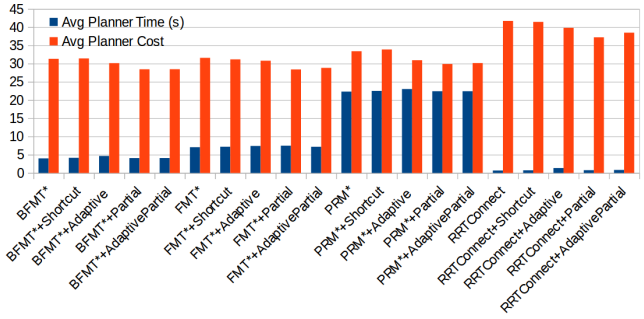


Figure 12: Average computation time and trajectory cost for SBP with shortcutting for the satellite environment.

E. SBP and Shortcut Selection

Based these results, we determined that the BFMT* + Partial Shortcut combination offered the best balance of speed, success rate, and path quality for scenarios similar to what may be expected in a satellite servicing task. The paired SBP and shortcutting results in Figures 10 and 12 demonstrate the efficacy of this combination. In complex environments with the perception constraints included, BFMT* produced the highest quality paths with a low failure rate and a comparatively fast planning time. However, in situations where planning time is more precious, for example if the tool is being used for rapid replanning during a maneuver, RRT-Connect + Partial Shortcut can be used for fast, immediate planning. The downside of this approach is lower path quality, potentially requiring more energy expense and time to complete maneuvering tasks.

Indeed, it is worth noting that although shortcutting is important, the choice of SBP algorithm is the main driver for cost reduction. As such, it is important to choose an asymptotically-optimal planner (e.g. variants of FMT*, RRT*, PRM*, etc.) to ultimately find high-quality paths, as they tend to return paths that are close to the global optimum. As shown in our experiments, shortcutting on a method such as RRTConnect cannot recover the path quality achieved by an asymptotically-optimal planner, as an algorithm like RRTConnect will often return paths that are refined to a bad local optimum.

7. Conclusions

In this paper, we have presented a fast motion planning tool that can be used to generate collision-free trajectories for satellite servicing manipulators, such as the 7-DoF Restore-L manipulator arms, while incorporating perception constraints for target and end-effector visibility. By automating the manipulator planning, the tool eliminates the cumbersome process of hand-designing end-effector paths and/or trajectories during satellite-servicing procedures. Additionally, BFMT* for initial planning and Partial Shortcut for post-processing were found to be the best combination in tightly constrained scenarios, such as those including perception constraints, which are typical of task scenarios that may be encountered during satellite-servicing operations.

There are several potential directions for future work, in addition to demonstration on a hardware test bed or in a satellite servicing mission. One direction is to extend this tool to a larger class of tasks within satellite servicing. For example, in cases where the servicing spacecraft is not rigidly attached to the target satellite, the fixed-based planning used by this tool may not provide sufficient accuracy. Additionally, for missions which include servicing manipulators having a very high number of total DoF, biased sampling [37] or planning using latent representations [38] may be required to plan trajectories in a reasonable amount of time. Also, integrating the tool into the operations of a mission may call for improved interface tools and visualizations of the planning space to let operators better evaluate and request adjustments to planned trajectories, allowing the planning pipeline to seamlessly fill in adjustments which continue to satisfy constraints.

Acknowledgments

This work was supported in part by an Early Career Faculty grant from NASA's Space Technology Research Grants

Program and by the King Abdulaziz City for Science and Technology (KACST). The authors wish to thank the team at the Satellite Servicing Projects Division at NASA Goddard for helpful discussions and insight on satellite servicing operations and special considerations for space manipulators.

REFERENCES

[1] G. A. Landis, S. G. Bailey, and R. Tischler, "Causes of power-related satellite failures," in *IEEE World Conf. on Photovoltaic Energy Conversion*, 2006.

[2] D. H. Martin, P. R. Anderson, and L. Bartamian, *Communication Satellites*, 5th ed. American Institute of Aeronautics and Astronautics, 2007.

[3] T. B. Sheridan, "Space teleoperation through time delay: Review and prognosis," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 5, pp. 592–606, 1993.

[4] B. A. Aikenhead, R. G. Daniell, and F. M. Davis, "Canadarm and the Space Shuttle," *Journal of Vacuum Science & Technology A*, vol. 1, no. 2, pp. 126–132, 1983.

[5] R. Boumans and C. Heemskerk, "The European Robotic Arm for the International Space Station," *Robotics and Autonomous Systems*, vol. 23, no. 1, pp. 17–27, 1998.

[6] R. Mcgregor and L. Oshinowo, "Flight 6A: Deployment and checkout of the Space Station Remote Manipulator System (SSRMS)," in *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2001.

[7] R. Mukherji, D. A. Rey, M. Stieber, and J. Lymer, "Special Purpose Dexterous Manipulator (SPDM) advanced control features and development test results," in *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2018.

[8] D. King, "Space servicing: past, present and future," in *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2001.

[9] D. Zimpfer, P. Kachmar, and S. Tuohy, "Autonomous rendezvous, capture and in-space assembly: Past, present and future," in *AIAA Space Exploration Conference*, 2005.

[10] C. Jewison, B. McCarthy, D. Sternberg, D. Strawser, and C. Fang, "Resource aggregated reconfigurable control and risk-allocative path planning for on-orbit servicing and assembly of satellites," in *AIAA Conf. on Guidance, Navigation and Control*, 2014.

[11] D. C. Sternberg, "Optimal docking to tumbling objects with uncertain properties," Ph.D. dissertation, Massachusetts Inst. of Technology, 2017.

[12] J. Telaar, I. Ahrns, S. Estable, W. Rackl, M. De Stefano, R. Lampariello, N. Santos, P. Serra, M. Canetri, F. Ankersen, and J. Gil-Fernandez, "GNC architecture for the e.Deorbit mission," in *European Conf. for Aeronautics and Space Sciences*, 2017.

[13] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE Conf. on Robotics and Automation*, 1985.

[14] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Conf. on Robotics and Automation*, 2009.

[15] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Conf. on Robotics and Automation*, 2011.

[16] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Robotics: Science and Systems*, 2013.

[17] J. Pan, Z. Chen, and P. Abbeel, "Predicting initialization effectiveness for trajectory optimization," in *Proc. IEEE Conf. on Robotics and Automation*, 2014.

[18] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[19] E. Frazzoli, M. A. Dahleh, E. Feron, and R. Kornfeld, "A randomized attitude slew planning algorithm for autonomous spacecraft," in *AIAA Conf. on Guidance, Navigation and Control*, 2001.

[20] R. Zappulla II, J. Virgili-Llop, and M. Romano, "Near-optimal real-time spacecraft guidance and control using harmonic potential functions and a modified RRT," in *AIAA/AAS Space Flight Mechanics Meeting*, 2017.

[21] J. Cortés and T. Siméon, "Sampling-based motion planning under kinematic loop-closure constraints," in *Algorithmic Foundations of Robotics VI*. Springer, 2005.

[22] F. James, S. Shah, K. Krishna, and A. Misra, "Reactionless maneuvering of a space robot in precapture phase," in *AIAA Journal of Guidance, Control, and Dynamics*, 2016.

[23] R. Geraerts and M. Overmars, "Creating high-quality paths for motion planning," *Int. Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.

[24] N. P. Papanikolopoulos, P. K. Khosla, and T. Kanade, "Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 1, pp. 14–35, 1993.

[25] S. Morikawa, T. Senoo, A. Namiki, and M. Ishikawa, "Realtime collision avoidance using a robot manipulator with light-weight small high-speed vision systems," in *Proc. IEEE Conf. on Robotics and Automation*, 2007.

[26] K. Hosoda, K. Sakamoto, and M. Asada, "Trajectory generation for obstacle avoidance of uncalibrated stereo visual servoing without 3D reconstruction," in *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, 1997.

[27] R. Lampariello, H. Mishra, N. Oumer, P. Schmidt, M. De Stefano, and A. Albu-Schäffer, "Tracking control for the grasping of a tumbling satellite with a free-floating robot," *IEEE Transactions on Robotics and Automation*, vol. 3, no. 4, pp. 3638–3645, 2018.

[28] S. M. LaValle, *Planning Algorithms*. Cambridge Univ. Press, 2006.

[29] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Workshop on Algorithmic Foundations of Robotics*, 2000.

[30] J. J. Kuffner and S. M. LaValle, "RRT-connect: an efficient approach to single-query path planning," in *Proc. IEEE Conf. on Robotics and Automation*, 2000.

[31] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.