

Computationally Efficient Motion Planning Algorithms for Agile Autonomous Vehicles in Cluttered Environments

Saptarshi Bandyopadhyay¹, Francesca Baldini², Rebecca Foust³, Amir Rahmani¹, Jean-Pierre de la Croix¹, Soon-Jo Chung², and Fred Y. Hadaegh¹

Abstract—Fast, real-time motion planning of an agile, autonomous vehicle in a cluttered environment, with many geometrically-fixed obstacles, is a very complex problem, especially because of the vehicle dynamics constraints and resource-constrained computational capabilities onboard the vehicle. In this paper, we present computationally-efficient versions of our novel motion planning algorithm called the Spherical Expansion and Sequential Convex Programming (SE-SCP) algorithm. The SE-SCP algorithm first uses a spherical-expansion-based randomized sampling algorithm to explore the workspace. Once a path is found from the start position to the goal position, the algorithm computes a locally optimal trajectory, within its homotopy class for a desired cost function, by solving a sequence of convex optimization problems. Thus, the SE-SCP algorithm is anytime locally optimal and the trajectory is globally optimal if the number of samples tends to infinity. In this paper, we further enhance the computational efficiency of the SE-SCP algorithm using uni-directional and bi-directional rewiring techniques. We also present a detailed proof of the local optimality characteristics of the new SE-SCP algorithms for a special case of vehicle dynamics. Simulation examples involving quadrotor and spacecraft help demonstrate the effectiveness of our new algorithms.

I. INTRODUCTION

Motion planning of an agile, autonomous vehicle, like quadrotor, micro aerial vehicle, or small satellite, has received considerable attention during the last decade due to their large range of applications [1], [2]. Quadrotor and micro aerial vehicle are suitable for deploying mobile sensor platforms for intelligence, surveillance, and reconnaissance missions. Recently, various space agencies have proposed missions to explore small bodies in space using small satellite or spacecraft. A common problem in these applications is that of generating an optimal trajectory for the autonomous

vehicle, which satisfies vehicle dynamics and collision avoidance constraints, in an environment cluttered with multiple geometrically fixed obstacles. In our prior work, we developed the Spherical Expansion and Sequential Convex Programming (SE-SCP) algorithm to address this problem [3]. In this paper, we further enhance the computational efficiency of the SE-SCP algorithm using uni-directional and bi-directional rewiring techniques.

A. Literature Survey

Motion planning algorithms in the robotics and artificial intelligence communities, which do not incorporate vehicle dynamics, can be broadly classified into cell decomposition methods [4], potential field-based methods [5], [6], and sampling-based methods [7], [8], [9], [10]. Cell decomposition methods are not suitable for incorporating vehicle dynamics, because they partition the state space into cells, and suffer from the curse of dimensionality. Similarly, potential field-based methods can get stuck in a local minima [11]. On the other hand, sampling-based methods, like probabilistic road maps (PRM) [8], rapidly exploring random trees (RRT) [9], expansive space trees (EST) [12], sampling-based roadmap of trees (SRT) [13], rapidly-exploring roadmap (RRM) [14]; have enjoyed significant practical success because of their ease of implementation, ability to handle higher-dimensional spaces, and *probabilistic completeness* (i.e., the probability that the algorithm fails to return a solution, if one exists, decays to zero as the number of samples tends to infinity). Moreover, some of these sampling-based methods, like PRM* and RRT* [7], RRT# [15], fast marching trees (FMT*) [10]; are *asymptotically optimal* (i.e., as the number of samples tends to infinity, the algorithm's solution converges almost surely to the globally optimal solution, with respect to a distance-based cost function). Note that there are no optimality guarantees for a finite number of samples and distance-based cost functions do not correlate well with the cost of fuel or control effort that the autonomous vehicle would like to minimize. Therefore, these sampling-based methods are not suitable for practical implementation on board agile autonomous vehicles.

Another key issue with directly applying the above sampling-based methods for motion planning of an agile vehicle is that the algorithm's solution might not satisfy the vehicle's dynamics constraints. A popular approach for vehicles dynamics that are *differentially flat* (e.g., quadrotors [16], [17]) is to project the sampling-based algorithm's solution onto the space of flat output variables, and then generate

*This work was supported by the Jet Propulsion Laboratory's Research and Technology Development (R&TD) program. Part of the research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. ©2017 California Institute of Technology. All rights reserved.

¹Saptarshi Bandyopadhyay, Amir Rahmani, Jean-Pierre de la Croix, and Fred Y. Hadaegh are with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, 91109, USA Saptarshi.Bandyopadhyay@jpl.nasa.gov, Amir.Rahmani@jpl.nasa.gov, Jean-Pierre.de.la.Croix@jpl.nasa.gov, Fred.Y.Hadaegh@jpl.nasa.gov

²Francesca Baldini and Soon-Jo Chung are with the Graduate Aerospace Laboratories, California Institute of Technology, Pasadena, California, 91125, USA fbaldini@caltech.edu, sjchung@caltech.edu

³Rebecca Foust is with the Department of Aerospace Engineering, University of Illinois at Urbana-Champaign, Urbana Illinois, 61801, USA foust3@illinois.edu

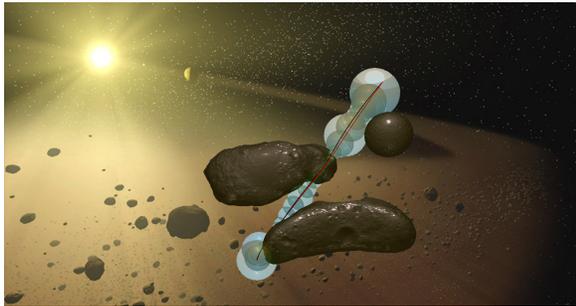


Fig. 1: The SE–SCP algorithm can be used for spacecraft trajectory planning in asteroid fields or other debris-rich environments.

a feasible trajectory. But this differential flatness-based approach cannot incorporate practical dynamics and kinematics constraints like actuator saturation, bounds on some states, etc. Furthermore, a strategy for using the sampling-based algorithm’s solution for vehicles, whose dynamics are not differentially flat, does not exist. Therefore, the objective of this paper is to present a sampling-based algorithm that directly incorporates any vehicle dynamics, like quadrotor, micro aerial vehicle, or small satellite.

On the other hand, spacecraft trajectory planning algorithms have to incorporate the nonlinear spacecraft dynamics into their optimization problems for computing minimum-fuel dynamically-feasible trajectories. Some of the optimization techniques that have been used for solving this complicated nonlinear optimization problem include mixed-integer linear program [18], pseudospectral method ([19]), and convex optimization [20]. In this paper, we use the sequential convex programming (SCP) method, which was first used for decentralized spacecraft swarm guidance in [21]. The SCP method solves multiple convex optimization problems that are approximations of the original nonlinear optimization problem, so that the SCP solution converges to the solution of the nonlinear optimization problem. Moreover, these convex optimization problems can be solved efficiently using freely available software [22] and on resource constrained hardware. Furthermore, in [21], hyperplanes are used for convexification collision avoidance constraints with other spacecraft for the SCP method. But there is no guarantee that a feasible (non-empty) convex set would remain after collision avoidance constraints in cluttered environments (with debris, other spacecraft, or near asteroids) have been convexified using hyperplanes. Therefore, we adopt a spherical-expansion based method to generate convex feasible sets for our SCP method.

B. Spherical Expansion and Sequential Convex Programming (SE–SCP) Algorithm

In our prior work, we introduced the SE–SCP algorithm, which is probabilistically complete, asymptotically optimal, *anytime locally optimal* (i.e., after the first path is found, for any finite number of samples, the algorithm’s solution converges almost surely to the locally optimal solution, with

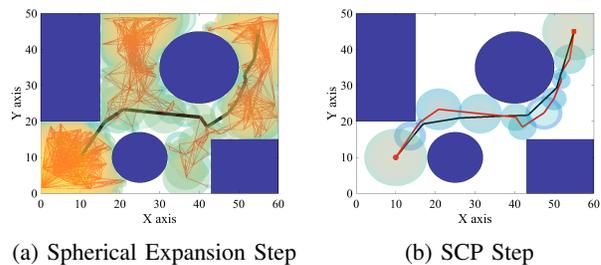


Fig. 2: The two steps in the SE–SCP algorithm, where the obstacles are marked in blue.

respect to the given cost function), and computationally efficient for real-time implementation on resource constrained systems [3]. The key steps in the SE–SCP algorithm are:

- **Spherical Expansion Step:** The algorithm first explores the workspace using a probabilistic or deterministic sampling-based method. In addition to building a graph, each node also encodes the information about the largest sphere centered at that node which does not intersect with any obstacle. As shown in Fig. 2a, the objective of this step is to find paths from the initial to the goal position.
- **SCP Step:** Once a geometric path is found, a dynamically-feasible trajectory is computed by solving the nonlinear optimization problem using the SCP method as shown in Fig 2b. The solution of SCP is locally optimal with respect to the given convex cost function.
- On further sampling, if a better geometric path is found, then the algorithm computes a new optimal trajectory using SCP step. The algorithm always stores the best trajectory found until the current time instant. Hence, this algorithm is anytime locally optimal, asymptotically optimal, and probabilistically complete.

C. Main Contributions

The main shortfall of the SE–SCP algorithm arises from the dense graph generated during the spherical expansion step (see Fig. 2a). It is seen that for each new node, the number of new edges generated during this step grows exponentially. Therefore, the computational load for storing this graph grows very quickly, and is not suitable for resource-constrained hardware. The first main contribution of this paper are two new algorithms that solve this problem. Leveraging on the well-known rewiring technique [7], we first present a new SE–SCP algorithm with uni-directional rewiring (i.e., the rewiring starts from the start position and extends throughout the network). Since the goal position is also known, we also develop a novel SE–SCP algorithm with bi-directional rewiring (where the rewiring happens simultaneously from the start and goal positions).

Another important contribution of this paper is a detailed optimality analysis for the SCP step, for a special case of vehicle dynamics. We show that the optimal trajectory calculated using SE–SCP algorithm indeed converges to the

local optimal trajectory for the given cost function. To our knowledge, this paper presents the first optimality proof of the SCP optimization technique, where the underlying optimization problem also changes during every iteration.

This paper is organized as follows. We present the problem statement in Section II. The SE-SCP algorithm with unidirectional rewiring and its optimality analysis are presented in Sections ?? and IV. Computationally efficient variants of the SE-SCP algorithm are presented in Section ?. Numerical simulation results and comparison with other existing algorithms are shown in Section V. This paper is concluded in Section VI.

II. PROBLEM STATEMENT

In this section, the nonlinear optimal motion planning problem is first presented in continuous-time in Section II-A. This problem is then transformed into a discrete-time nonlinear optimal motion planning problem in Section II-B.

A. Continuous-time Nonlinear Optimal Motion Planning Problem

Let $\mathcal{X} \subset \mathbb{R}^3$ represent the 3D workspace. Let $\mathcal{X}_{\text{obs}} \subset \mathcal{X}$ represent the obstacles in the workspace. Let δ denote the clearance that the vehicle must maintain from any obstacle. Let $\mathcal{X}_{\text{unsafe}} \subset \mathcal{X}$ denote the unsafe regions around the obstacles that the vehicle cannot enter. Therefore, the region where the vehicle can maneuver freely is given by $\mathcal{X}_{\text{free}} = \mathcal{X} / (\mathcal{X}_{\text{obs}} \cup \mathcal{X}_{\text{unsafe}})$. Let $X_{\text{init}} \in \mathcal{X}_{\text{free}}$ and $X_{\text{goal}} \in \mathcal{X}_{\text{free}}$ represent the starting position and the goal position of the vehicle at times t_0 and t_f respectively.

Let $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ and $\mathbf{u}(t) \in \mathbb{R}^{n_u}$ represent the vehicle's state vector and control input at time t . The vehicle's state vector can be further decomposed into $\mathbf{x}(t) = \left(\mathbf{p}(t)^T, \dot{\mathbf{p}}(t)^T, \boldsymbol{\theta}(t)^T, \dot{\boldsymbol{\theta}}(t)^T \right)^T$, where $\mathbf{p}(t) \in \mathbb{R}^3$ and $\boldsymbol{\theta}(t)$ represent the vehicle's position and attitude vectors. The objective of the optimal motion planning problem is to find a feasible trajectory for the vehicle that starts at X_{init} at time t_0 , reaches X_{goal} at time t_f , and minimizes the given convex cost function of the control inputs $c(\mathbf{u}(t))$.

Problem 1: *Continuous-time Nonlinear Optimal Motion Planning*

$$\underset{\mathbf{x}(t), \mathbf{u}(t)}{\text{minimize}} \int_{t_0}^{t_f} c(\mathbf{u}(t)) dt, \quad (1)$$

$$\text{subject to } \mathbf{p}(t_0) = X_{\text{init}}, \quad (2)$$

$$\mathbf{p}(t_f) = X_{\text{goal}}, \quad (3)$$

$$\mathbf{p}(t) \in \mathcal{X}_{\text{free}}, \quad \forall t \in [t_0, t_f], \quad (4)$$

$$\mathbf{u}(t) \in \mathcal{U}, \quad \forall t \in [t_0, t_f], \quad (5)$$

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \forall t \in [t_0, t_f], \quad (6)$$

where (2)–(3) represent the initial and terminal constraints, the constraint (4) ensures that the trajectory is within the safe region, and $\mathcal{U} \subset \mathbb{R}^{n_u}$ in (5) represents the feasible range of control inputs. The vehicle's dynamics is given by (6), where $\mathbf{f}(\cdot)$ is a smooth nonlinear function.

B. Discrete-time Nonlinear Optimal Motion Planning Problem

Here we transform the continuous-time **Problem 1**(1)–(6) into a discrete-time problem so that it can be efficiently solved using SCP. In order to do this, the variables in **Problem 1** are discretized using a zero-order hold approach such that:

$$\mathbf{u}(t) = \mathbf{u}[k], \quad t \in [t_k, t_{k+1}), \quad k \in \{0, \dots, T-1\}, \quad (7)$$

$$\mathbf{x}(t) = \mathbf{x}[k], \quad t \in [t_k, t_{k+1}), \quad k \in \{0, \dots, T-1\}, \quad (8)$$

$$\therefore \mathbf{p}(t) = \mathbf{p}[k], \quad \boldsymbol{\theta}(t) = \boldsymbol{\theta}[k],$$

where Δ is the time step size, $t_k = t_0 + k\Delta$, and T is the number of discrete time steps (i.e., $t_T = t_f$). In **Problem 1**, the cost function (1) and the constraints (2)–(5) are easily discretized using this zero-order hold approach.

In order to write the vehicle's dynamics equations (6) in a discrete-time form, this equation is first linearized around $(\mathbf{x}^*, \mathbf{u}^*)$ as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) + A(\mathbf{x}(t) - \mathbf{x}^*) + B(\mathbf{u}(t) - \mathbf{u}^*), \quad (9)$$

$$\text{where } A = \left. \frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{x}(t)} \right|_{(\mathbf{x}^*, \mathbf{u}^*)}, \quad (10)$$

$$B = \left. \frac{\partial \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))}{\partial \mathbf{u}(t)} \right|_{(\mathbf{x}^*, \mathbf{u}^*)}. \quad (11)$$

Equation (9) is then discretized using a zero-order hold approach as follows:

$$\mathbf{x}[k+1] = F[k] \mathbf{x}[k] + G[k] \mathbf{u}[k] + H[k], \quad (12)$$

$$\text{where } F[k] = e^{A\Delta}, \quad (13)$$

$$G[k] = \int_0^\Delta e^{A(\Delta-\tau)} B d\tau, \quad (14)$$

$$H[k] = \int_0^\Delta e^{A(\Delta-\tau)} (\mathbf{f}(\mathbf{x}^*, \mathbf{u}^*) - A\mathbf{x}^* - B\mathbf{u}^*) d\tau. \quad (15)$$

Therefore, the transformed discrete-time problem is given in **Problem 2**(16)–(21).

Problem 2: *Discrete-time Nonlinear Optimal Motion Planning*

$$\underset{\mathbf{x}[k], \mathbf{u}[k]}{\text{minimize}} \sum_{k=0}^{T-1} c(\mathbf{u}[k]) \Delta, \quad (16)$$

$$\text{subject to } \mathbf{p}[0] = X_{\text{init}}, \quad (17)$$

$$\mathbf{p}[T] = X_{\text{goal}}, \quad (18)$$

$$\mathbf{p}[k] \in \mathcal{X}_{\text{free}}, \quad \forall k \in \{0, \dots, T\}, \quad (19)$$

$$\mathbf{u}[k] \in \mathcal{U}, \quad \forall k \in \{0, \dots, T-1\}, \quad (20)$$

$$\mathbf{x}[k+1] = F[k] \mathbf{x}[k] + G[k] \mathbf{u}[k] + H[k], \quad (21)$$

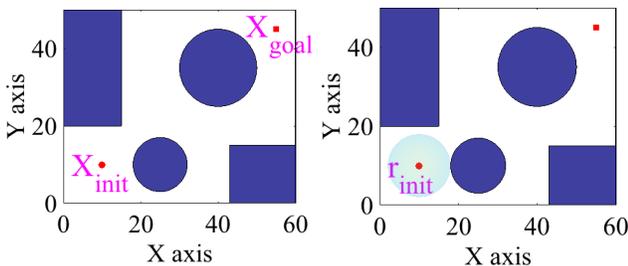
$$\forall k \in \{0, \dots, T-1\}.$$

Note that the constraints (17), (18), and (21) are linear constraints. For most vehicle dynamics, (20) is a convex constraint because \mathcal{U} is a convex set. Therefore, the nonlinearity in **Problem 2** arises from the safety constraint (19), which is usually not convex.

The objective of this paper is to find an efficient technique to solve **Problem 2**(16)–(21). Our SE–SCP algorithms using uni-directional and bi-directional rewiring first explore the workspace to approximate the set $\mathcal{X}_{\text{free}}$ using a combination of convex sets. Once a feasible path is found, then it solves a convex approximation of **Problem 2** using SCP.

III. SPHERICAL EXPANSION AND SEQUENTIAL CONVEX PROGRAMMING (SE–SCP) ALGORITHM WITH UNI-DIRECTIONAL REWIRING

The pseudocode of the SE–SCP algorithm with Uni-directional Rewiring is presented in Algorithm 1. During the *Initialization step* (Section III-A), the necessary data structures are created and initialized. After initialization, the *Spherical Expansion step with Uni-directional Rewiring* (Section III-B) and the *Sequential Convex Programming step* (Section III-C) are executed iteratively until the algorithm is stopped.



(a) X_{init} and X_{goal} are shown (b) r_{init} and r_{goal} are shown

Fig. 3: Initialization step. The obstacles are marked in blue.

A. Initialization Step

The SE–SCP algorithm with uni-directional rewiring intends to generate a tree in the safe region $\mathcal{X}_{\text{free}}$, which is rooted at X_{init} . Each node in the set of vertices \mathcal{V} stores the position of the vertex, the minimum distance of that vertex from any obstacle, the parent node of that vertex, and the cost of the edge from the parent node respectively. For the vertex X_{init} , the minimum distance from the obstacle r_{init} is obtained using the function $\text{FindMinDistObs}(X_{\text{init}}, \mathcal{A}_{\text{obs}})$, which takes in the position of the vertex and the obstacles in the workspace and returns the radius of the largest sphere centered on that vertex which does not intersect with any obstacle (line 1). Then the set of vertices \mathcal{V} is initialized with the vertex $X_{\text{init}}[r_{\text{init}}, 0, 0]$, where the parent node and the cost of the edge from the parent node is set to 0 because it is the root node (line 3). Furthermore, the cost for the path $c_{P_{\text{old}}}$ and the trajectory $c_{x_{\text{old}}}$ are also set to infinity (line 4). The number of samples N is also initialized to an appropriate value.

B. Spherical Expansion Step with Uni-directional Rewiring

During this step, the workspace is explored using the sampling technique shown in lines 6–?? in Algorithm 1. The objective of this step is to populate the tree so that paths from X_{init} to X_{goal} can be found.

1) *Generate Sample*: The SE–SCP algorithm can use both random or quasi-random (deterministic) sampling since the randomness of the samples is not crucial for motion planning applications [23]. For a given sampling choice, the function GenerateSample returns a random sample $X_{\text{rand}} \in \mathcal{X}$ (line 6). If random sampling is chosen, then X_{rand} is drawn from a uniform distribution in \mathcal{X} such that the sequence of random points are independent and identically distributed. If quasi-random sampling is chosen, then X_{rand} is generated using the Halton sequence, which deterministically produces samples with low discrepancy and provide good coverage over the workspace with a limited number of samples [24].

2) *Generate New Point*: Next, the function $\text{NearestVertex}(\mathcal{V}, X_{\text{rand}})$ takes in the current set of vertices \mathcal{V} and the given sample X_{rand} and returns the vertex X_{nearest} that is nearest to X_{rand} (line 7), i.e.,:

$$X_{\text{nearest}} := \underset{X \in \mathcal{V}}{\operatorname{argmin}} \|X - X_{\text{rand}}\|_2. \quad (22)$$

Note that the minimum distance from the obstacles r_{nearest} for the vertex X_{nearest} is already stored in \mathcal{V} .

The function $\text{Steer}(X_{\text{rand}}, X_{\text{nearest}}, r_{\text{nearest}})$ generates the new point X_{new} using either of the two following steps (line 8):

- If X_{rand} is serendipitously inside X_{nearest} 's sphere (i.e., $\|X_{\text{nearest}} - X_{\text{rand}}\|_2 \leq r_{\text{nearest}}$), then new point $X_{\text{new}} = X_{\text{rand}}$.
- Otherwise, X_{rand} is outside X_{nearest} 's sphere. Then the new point X_{new} is on the surface of X_{nearest} 's sphere and closest to the sample X_{rand} , i.e.,:

$$\begin{aligned} X_{\text{new}} &:= \underset{\|X - X_{\text{nearest}}\|_2 = r_{\text{nearest}}}{\operatorname{argmin}} \|X - X_{\text{rand}}\|_2 \\ &= X_{\text{nearest}} + r_{\text{nearest}} \frac{(X_{\text{rand}} - X_{\text{nearest}})}{\|X_{\text{rand}} - X_{\text{nearest}}\|_2}. \end{aligned} \quad (23)$$

The minimum distance from obstacles r_{new} for the point X_{new} is computed using the function FindMinDistObs (line 9). In contrast with the classical steering function in RRT* [7], where the radius of the sphere is fixed and represents the step-size of the RRT* algorithm, the radius of the sphere in the SE–SCP algorithm is variable and the average step-size adapts with the density of obstacles. Therefore, the SE–SCP algorithm generally finds a feasible path faster than other sampling-based algorithms.

We then initialize the new vertex X_{new} 's parent with X_{nearest} (line 10). The function $\text{EdgeCost}(X_{\text{nearest}}, r_{\text{nearest}}, X_{\text{new}}, r_{\text{new}})$ takes in the two vertices and outputs the cost of traversing the directed edge $c_{\text{nearest}, \text{new}}$ from X_{nearest} to X_{new} , which depends on the given convex cost function, such that the trajectory always remains inside the union of the two spheres (line 11). The variable representing the cost from the parent node c_{parent} is initialized to $c_{\text{nearest}, \text{new}}$ (line 11). The variable c_{min} , which represents the minimum cost of reaching the vertex X_{new} , is initialized as the sum of c_{parent} and the cost of reaching the vertex X_{nearest} from the root (line 12). For any vertex X , the function $\text{Cost}(X)$, which represents the

Algorithm 1 SE-SCP Algorithm with Uni-directional Rewiring

```
1:  $r_{\text{init}} \leftarrow \text{FindMinDistObs}(X_{\text{init}}, \mathcal{X}_{\text{obs}})$ 
2:  $r_{\text{goal}} \leftarrow \text{FindMinDistObs}(X_{\text{goal}}, \mathcal{X}_{\text{obs}})$ 
3:  $\mathcal{V} \leftarrow \{X_{\text{init}}[r_{\text{init}}, 0, 0]\}$ 
4:  $c_{P_{\text{old}}} \leftarrow \infty, c_{\mathbf{x}_{\text{old}}} \leftarrow \infty, P_{\text{old}} \leftarrow \emptyset$ 
5: for  $i = 1, \dots, N$  do
6:    $X_{\text{rand}} \leftarrow \text{GenerateSample}$ 
7:    $X_{\text{nearest}} \leftarrow \text{NearestVertex}(\mathcal{V}, X_{\text{rand}})$ 
8:    $X_{\text{new}} \leftarrow \text{Steer}(X_{\text{rand}}, X_{\text{nearest}}, r_{\text{nearest}})$ 
9:    $r_{\text{new}} \leftarrow \text{FindMinDistObs}(X_{\text{new}}, \mathcal{X}_{\text{obs}})$ 
10:   $X_{\text{parent}} \leftarrow X_{\text{nearest}}$ 
11:   $c_{\text{parent}} \leftarrow c_{\text{nearest, new}} \leftarrow \text{EdgeCost}(X_{\text{nearest}}, r_{\text{nearest}}, X_{\text{new}}, r_{\text{new}})$ 
12:   $c_{\text{min}} \leftarrow c_{\text{parent}} + \text{Cost}(X_{\text{nearest}})$ 
13:   $\mathbf{X}_{\text{near}} \leftarrow \text{NearVertices}(\mathcal{V}, X_{\text{new}}, r_{\text{new}})$ 
14:  for all  $X_n \in \mathbf{X}_{\text{near}}$  do
15:    if  $c_{\text{min}} > \text{EdgeCost}(X_n, r_n, X_{\text{new}}, r_{\text{new}}) + \text{Cost}(X_n)$  then
16:       $X_{\text{parent}} \leftarrow X_n$ 
17:       $c_{\text{parent}} \leftarrow \text{EdgeCost}(X_n, r_n, X_{\text{new}}, r_{\text{new}})$ 
18:       $c_{\text{min}} \leftarrow c_{\text{parent}} + \text{Cost}(X_n)$ 
19:    end if
20:  end for
21:   $\mathcal{V} \leftarrow \mathcal{V} \cup \{X_{\text{new}}[r_{\text{new}}, X_{\text{parent}}, c_{\text{parent}}]\}$ 
22:  if  $X_{\text{goal}} \notin \mathcal{V}$  and  $\|X_{\text{goal}} - X_{\text{new}}\|_2 \leq r_{\text{goal}} + r_{\text{new}}$  then
23:     $X_{\text{parent}} \leftarrow X_{\text{new}}$ 
24:     $c_{\text{parent}} \leftarrow \text{EdgeCost}(X_{\text{new}}, r_{\text{new}}, X_{\text{goal}}, r_{\text{goal}})$ 
25:     $\mathcal{V} \leftarrow \mathcal{V} \cup \{X_{\text{goal}}[r_{\text{goal}}, X_{\text{parent}}, c_{\text{parent}}]\}$ 
26:  end if
27:   $\mathbf{X}_{\text{rewired}} \leftarrow \{X_{\text{new}}\}$ 
28:  while  $\mathbf{X}_{\text{rewired}} \neq \emptyset$  do
29:     $X_f \leftarrow$  first element in  $\mathbf{X}_{\text{rewired}}$ 
30:    Remove first element in  $\mathbf{X}_{\text{rewired}}$ 
31:     $\mathbf{X}_{\text{near}} \leftarrow \text{NearVertices}(\mathcal{V}, X_f, r_f)$ 
32:    for all  $X_n \in \mathbf{X}_{\text{near}}$  do
33:      if  $\text{Cost}(X_n) > \text{Cost}(X_f) + \text{EdgeCost}(X_f, r_f, X_n, r_n)$  then
34:         $X_n$ 's parent  $\leftarrow X_f$ 
35:         $X_n$ 's cost from parent  $\leftarrow \text{EdgeCost}(X_f, r_f, X_n, r_n)$ 
36:         $\mathbf{X}_{\text{rewired}} \leftarrow \mathbf{X}_{\text{rewired}} \cup \{X_n\}$ 
37:      end if
38:    end for
39:  end while
40:  if  $X_{\text{goal}} \in \mathcal{V}$  then
41:     $P_{\text{new}} \leftarrow \text{FindPath}(\mathcal{V})$ 
42:     $c_{P_{\text{new}}} \leftarrow \text{Cost}(X_{\text{goal}})$ 
43:    if  $c_{P_{\text{new}}} < c_{P_{\text{old}}}$  and  $P_{\text{new}} \neq P_{\text{old}}$  then
44:       $(\mathbf{x}_{\text{new}}, \mathbf{u}_{\text{new}}, c_{\mathbf{x}_{\text{new}}}) \leftarrow \text{OptimalTrajectory}(P_{\text{new}})$ 
45:      for  $j = 1, \dots, N_{\text{SCP}}$  do
46:         $P_{\text{newer}} \leftarrow \text{GeneratePath}(\mathbf{x}_{\text{new}})$ 
47:         $(\mathbf{x}_{\text{new}}, \mathbf{u}_{\text{new}}, c_{\mathbf{x}_{\text{new}}}) \leftarrow \text{OptimalTrajectory}(P_{\text{newer}}, \mathbf{x}_{\text{new}}, \mathbf{u}_{\text{new}})$ 
48:      end for
49:      if  $c_{\mathbf{x}_{\text{new}}} < c_{\mathbf{x}_{\text{old}}}$  then
50:         $\mathbf{x}_{\text{old}} \leftarrow \mathbf{x}_{\text{new}}, \mathbf{u}_{\text{old}} \leftarrow \mathbf{u}_{\text{new}}, c_{\mathbf{x}_{\text{old}}} \leftarrow c_{\mathbf{x}_{\text{new}}}, P_{\text{old}} \leftarrow P_{\text{new}}, c_{P_{\text{old}}} \leftarrow c_{P_{\text{new}}}$ 
51:      else  $P_{\text{old}} \leftarrow P_{\text{new}}$ 
52:      end if
53:    end if
54:  end if
55: end for
```

cost of reaching the vertex X from the root vertex X_{init} , is recursively evaluated as:

$$\text{Cost}(X) = \begin{cases} 0 & \text{if } X = X_{\text{init}} \\ \text{Cost}(X\text{'s parent}) + \text{EdgeCost}(X\text{'s parent}, X) & \text{otherwise} \end{cases} \quad (24)$$

Here we assume that the cost function is additive.

3) *Add New Point to Tree*: The function $\text{NearVertices}(\mathcal{V}, X_{\text{new}}[r_{\text{new}}])$ takes in the current set of vertices \mathcal{V} and the new vertex $X_{\text{new}}[r_{\text{new}}]$ and generates a list \mathbf{X}_{near} of all the vertices whose spheres intersect with X_{new} 's sphere (line 13), i.e.,:

$$\mathbf{X}_{\text{near}} := \{X_n \in \mathcal{V} : \|X_n - X_{\text{new}}\|_2 \leq r_n + r_{\text{new}}\}. \quad (25)$$

It follows from our construction that the set \mathbf{X}_{near} is not empty because $X_{\text{nearest}} \in \mathbf{X}_{\text{near}}$. There exists a feasible collision-free path exists between each vertex in \mathbf{X}_{near} and X_{new} because their spheres intersect.

We now find the best parent for the new vertex X_{new} (lines 14–20). For each vertex $X_n \in \mathbf{X}_{\text{near}}$, if the sum of the cost of reaching X_n from the root and the cost of traversing the edge from X_n to X_{new} is less than c_{min} , then the vertex X_n is set as the parent of X_{new} . Finally, the new vertex $X_{\text{new}}[r_{\text{new}}, X_{\text{parent}}, c_{\text{parent}}]$ is added to the set of vertices \mathcal{V} (line 21).

4) *Add Goal to Tree*: If the vertex X_{goal} is not yet in the set of vertices \mathcal{V} , but X_{new} 's sphere intersects with X_{goal} 's sphere, then the vertex X_{goal} is added to \mathcal{V} with the vertex X_{new} as its parent (lines 22–26).

5) *Rewire the Tree*: We first initialize the list $\mathbf{X}_{\text{rewired}}$ with the new vertex X_{new} (line 27). Then the tree is recursively rewired as follows (lines 28–39). We take the first vertex X_f in the list $\mathbf{X}_{\text{rewired}}$ and remove it from the list. We find the vertex X_f 's neighbors \mathbf{X}_{near} , and check if any of these neighbors can have a lower cost by traveling through X_f . If so, then the parent of that vertex is updated and neighbor is added to the list $\mathbf{X}_{\text{rewired}}$. The process continues till the list $\mathbf{X}_{\text{rewired}}$ is empty. This process ensures that the rewiring step is used for every vertex whose cost is reduced during the current time instant. This is a significant improvement over the rewiring step in [7], where only the neighbors of X_{new} are rewired.

In Fig. 4, multiple iterations of the spherical expansion step are shown. At each step, a new vertex and a new edge are added to the tree, while some existing edges are rewired. Note that the tree is rooted at the start position.

C. Sequential Convex Programming Step

During this step in 40–54 in Algorithm 1, the path from the initial position to the goal position is used to find the locally optimal trajectory. This step is only executed if the goal position has been found, i.e., the goal vertex X_{goal} has been added to the set of vertices \mathcal{V} (line 40).

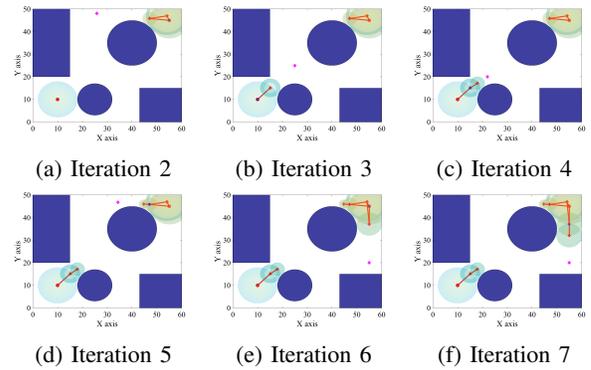


Fig. 4: Multiple iterations of the spherical expansion step with uni-directional rewiring

1) *Find Path*: The function $\text{FindPath}(\mathcal{V})$ takes in the set of vertices \mathcal{V} and returns the new path P_{new} , which is a sequence of vertices starting with X_{init} and ending at X_{goal} (line 41). This path is found using the parent information stored in each vertex. The cost of this new path $c_{P_{\text{new}}}$ is easily calculated using $\text{Cost}(X_{\text{goal}})$ (line 42).

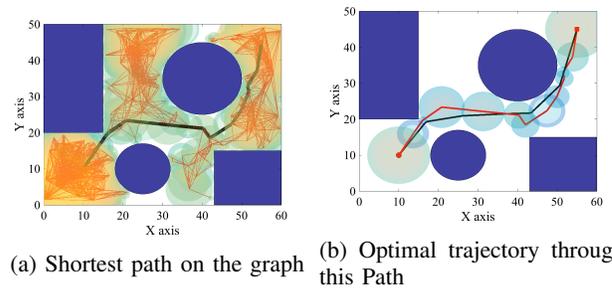


Fig. 5: Generating Shortest Path and Trajectory Optimization over this Path

2) *Generate Optimal Trajectory*: If the cost of the new path $c_{P_{\text{new}}}$ is less than that of the old path $c_{P_{\text{old}}}$ and the new path is different from the old path, then the optimal trajectory is found along this new path. The function $\text{OptimalTrajectory}(P_{\text{new}})$ takes in this new path P_{new} and returns the optimal trajectory \mathbf{x}_{new} , \mathbf{u}_{new} and the cost of traversing this trajectory $c_{\mathbf{x}_{\text{new}}}$ by solving an approximate version of **Problem 2**(16)–(21) (line 44). If no additional trajectory is provided (as is the case in line 47), then a nominal trajectory is used for linearization.

The path $P_{\text{new}} = \{X_1[r_1], X_2[r_2], \dots, X_n[r_n]\}$ is a sequence of n vertices with corresponding radii, where $X_1 = X_{\text{init}}$ and $X_n = X_{\text{goal}}$. An example path is shown in Fig. 6. Our key innovation is to use this sequence of spheres as an approximation of $\mathcal{X}_{\text{free}}$, in order to convexify the constraint (19) in **Problem 2**.

We first set the time step size Δ such that $T = (2n - 1)$. In order to convexify the constraint (19), we enforce that all points with even index are at the intersection of two spheres (29,30), and all points with odd indices are inside their corresponding sphere (31). Thus, the convex problem

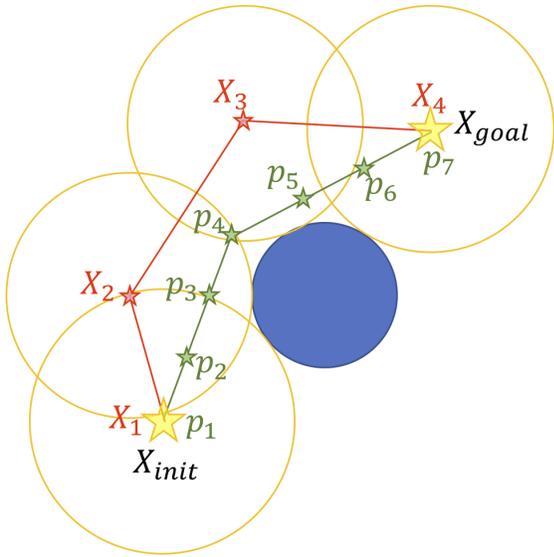


Fig. 6: Visualization of optimal trajectory generation using SCP

is given in **Problem 3**(26)–(33).

Problem 3: Discrete-time Convex Optimal Motion Planning

$$\underset{\mathbf{x}[k], \mathbf{u}[k]}{\text{minimize}} \sum_{k=0}^{T-1} c(\mathbf{u}[k]) \Delta, \quad (26)$$

$$\text{subject to } \mathbf{p}[0] = X_{\text{init}}, \quad (27)$$

$$\mathbf{p}[T] = X_{\text{goal}}, \quad (28)$$

$$\|\mathbf{p}[2\ell] - X_{\ell}\|_2 \leq r_{\ell}, \forall \ell \in \{1, \dots, n-1\}, \quad (29)$$

$$\|\mathbf{p}[2\ell] - X_{\ell+1}\|_2 \leq r_{\ell+1}, \forall \ell \in \{1, \dots, n-1\}, \quad (30)$$

$$\|\mathbf{p}[2\ell+1] - X_{\ell+1}\|_2 \leq r_{\ell+1}, \forall \ell \in \{1, \dots, n-2\}, \quad (31)$$

$$\mathbf{u}[k] \in \mathcal{U}, \quad \forall k \in \{0, \dots, T-1\}, \quad (32)$$

$$\mathbf{x}[k+1] = F[k] \mathbf{x}[k] + G[k] \mathbf{u}[k] + H[k], \quad (33)$$

$$\forall k \in \{0, \dots, T-1\}.$$

Note that **Problem 3**(26)–(33) is a convex optimization problem. Hence it can be solved efficiently using freely available software [22] and on resource constrained hardware.

The optimal solution of **Problem 3**, namely $\mathbf{x}_{\text{new}} = \{\mathbf{x}[1], \mathbf{x}[2], \dots, \mathbf{x}[T]\}$ and $\mathbf{u}_{\text{new}} = \{\mathbf{u}[1], \mathbf{u}[2], \dots, \mathbf{u}[T-1]\}$, can be used to setup a new optimization problem. The function `GeneratePath`(\mathbf{x}_{new}) takes in this optimal solution \mathbf{x}_{new} and returns a new path P_{newer} , which is a sequence of points based on \mathbf{x}_{new} and the corresponding radius (line 46). If the spheres of 2 vertices in \mathbf{x}_{new} do not intersect, then the spheres from the previous step is previous step is used to fill in the gap. Once again, the optimal trajectory along this path can be found using **Problem 3**(26)–(33), where \mathbf{x}_{new} and \mathbf{u}_{new} are used for linearization (line 47). This optimization process using SCP continues N_{SCP} times (lines 45–48). After the optimization process has terminated, the current trajectory cost is compared with the stored cost, and the best trajectory is stored (lines 49–52).

At the end of N such loops, the SE-SCP algorithm using uni-directional rewiring returns the optimal trajectory \mathbf{x}_{old} and \mathbf{u}_{old} .

IV. ANALYSIS OF THE SE-SCP ALGORITHM

V. NUMERICAL SIMULATIONS

In this section, we first show that the SE-SCP algorithm can be used by a spacecraft to navigate a debris field. We then present comparisons with existing algorithms like RRT* and PRM*.

A. Spacecraft in Debris Field

In this subsection, we show that the spacecraft can use the SE-SCP algorithm to move through a cluttered environment like a debris field. The debris environment, the start position, and the goal position are shown in Fig. 7. The spacecraft dynamics are discussed in Section ???. The objective is to find a trajectory from X_{init} to X_{goal} so that the fuel consumed is minimized.

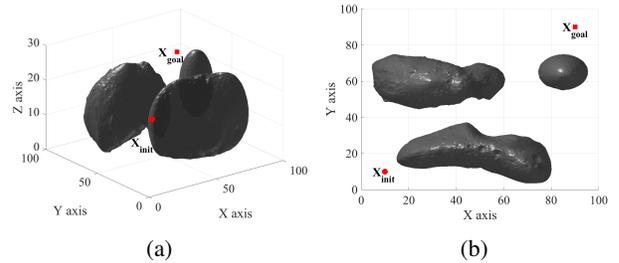


Fig. 7: Asteroid environment

Fig. 8 shows two trajectories in two different homotopy classes that are generated by the spherical expansion step. Each of these trajectories are further refined using the SCP step, as shown in Fig. 9. The solution of the SE-SCP algorithm after a finite number of samples is the best locally optimal trajectory shown in Fig. 9. thus the SE-SCP algorithm is suitable for spacecraft applications.

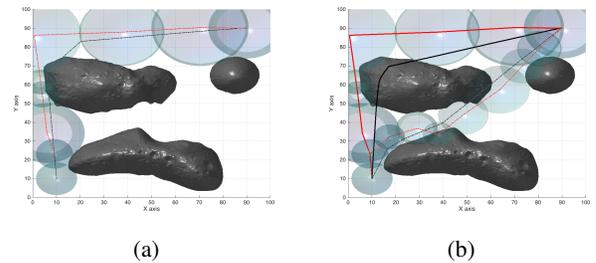


Fig. 8: Paths in different homotopy classes found by the spherical expansion step

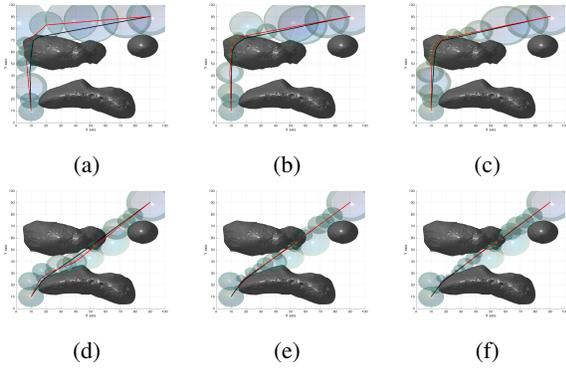


Fig. 9: Trajectories are further optimized in the SCP step

B. Crazyflie Quadrotor in Cluttered Environment

In this subsection, the *Crazyflie quadrotor* [?] is required to move from an initial configuration to a final one, through several cluttered environments, as shown in Fig. 10, Fig. 11 and Fig. 12. The quadrotor dynamics are discussed in Section ?? and, as already mentioned above, it is highly non-linear. However, at each iteration, the non-linear dynamics is linearized around each feasible way-point returned by the algorithm at the previous iteration. The objective is to find the optimal trajectory such that the control input is minimized, where the control input is the voltage which is applied to the motors, which in turn is controlled using a pulse width modulated (PWM) signal.

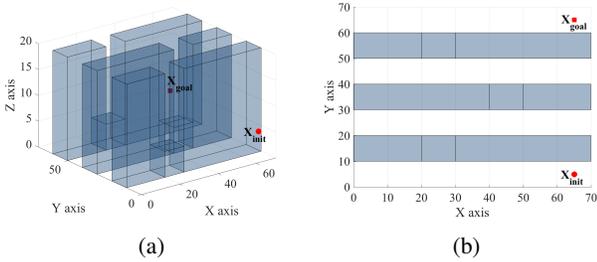


Fig. 10: Walls

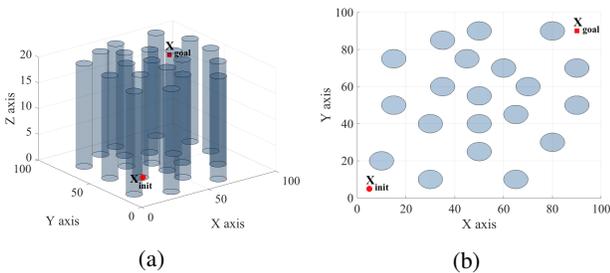


Fig. 11: Forest

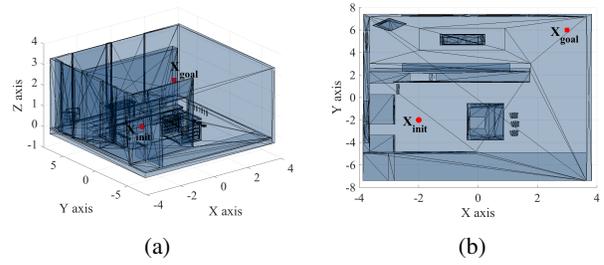


Fig. 12: ROS Kitchen

To assure that the quadrotor trajectory generated by the SE-SCP algorithm is feasible, boundary conditions inherent to the achievement of the desired task are imposed on the quadrotor such that:

- $X(0) = X_{init}$
- $X(T) = X_{goal}$

The inequality constraints of the form $A_{ineq}x0b_{ineq}$ consists to place constraints on velocity v , acceleration a , jerk \dot{a} , and snap \ddot{a} to ensure continuity in the trajectory as well as to restrict the trajectory to the dynamic limits of the quadrotor .

Acceleration constraints are used to ensure the quadrotor is near a desired attitude at a certain step along the trajectory. To ensure continuity of the jerk between the trajectories between consecutive waypoints and since jerk has to be continuous and bounded, jerk and snap respectively have to be bounded.[?]. Angular velocity and acceleration vectors have to be also sufficiently bounded in order to impose the control authority constraints. However, is not guaranteed that the attitude quaternion remain close to a unit vector at each time step. For the remark about this last statement refers to [?],[?],[?]. For this reason , at each time step, a new desired quaternion vector is calculated such that the condition $\|q\| = 1$ is respected.

Results for quadrotor simulation are shown in Fig. 13, Fig.14 and Fig.15.

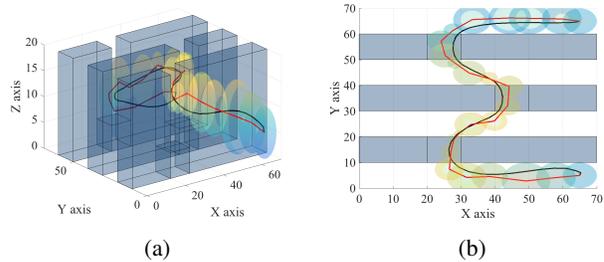


Fig. 13: Walls

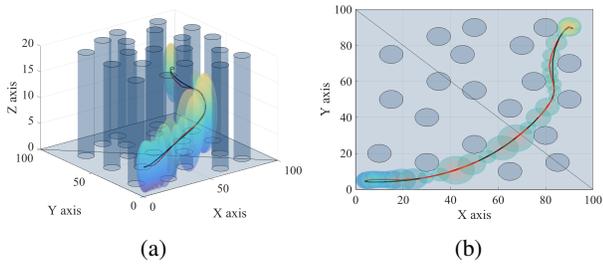


Fig. 14: Forest

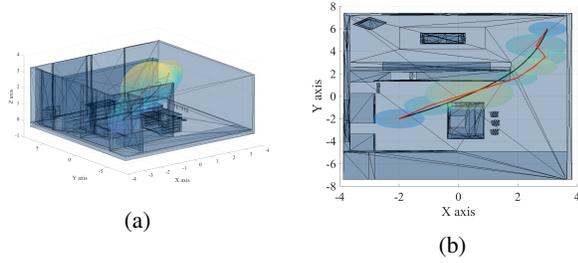


Fig. 15: ROS Kitchen

C. Comparison with RRT* and PRM*

In this subsection, we compare the SE-SCP algorithm with the RRT* and PRM* algorithms [7]. These algorithms cannot directly incorporate the spacecraft dynamics. Therefore, we use the single integrator dynamics which is given by:

$$\dot{\mathbf{x}} = \mathbf{u}. \quad (34)$$

The 3D benchmark environments are shown in Figs. 16–20. Each of the algorithms are executed multiple times and the histogram plots show the average computational runtime and trajectory cost. The SE-SCP's results with both random and quasi-random sampling are shown.

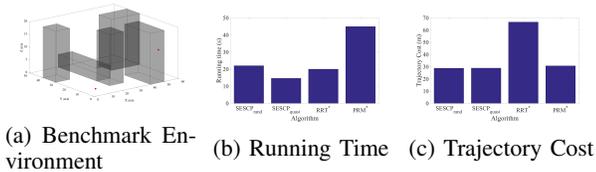


Fig. 16: Benchmark Comparison Room 1. The simulation is run with 500 samples.

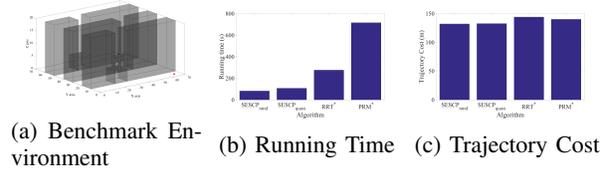
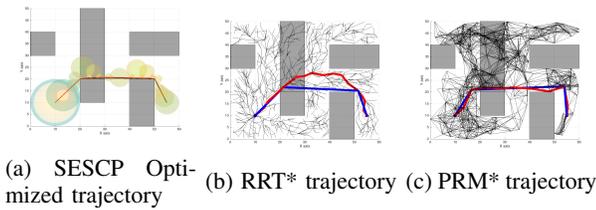


Fig. 18: Benchmark Comparison Room 2. The simulation is run with 2000 samples.

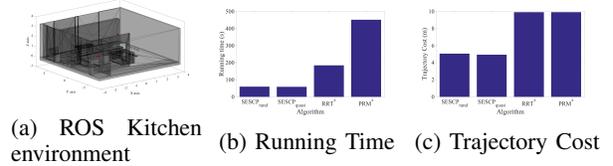
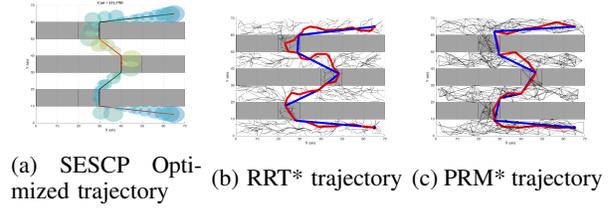
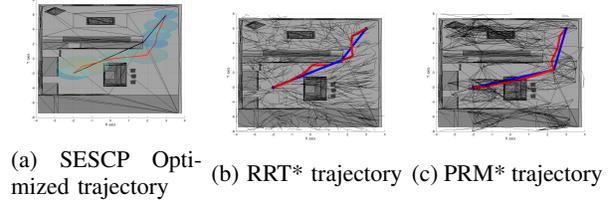


Fig. 20: Benchmark Comparison in the ROS Kitchen environment. The simulation is run with 1000 samples.



We conclude from these comparisons that the SE-SCP algorithm outperforms the the RRT* and PRM* algorithms. The main reasons for the superior performance of the SE-SCP algorithm are as follows:

- The spherical expansion step in the SE-SCP algorithm adapts with the density of the obstacles in the environment. Since there is no pre-defined step-length in the SE-SCP algorithm, the algorithm can build larger spheres when there are no nearby obstacles. This is extremely helpful in quickly covering the workspace and possibly generating a simple path from the start to the goal position.
- The SCP step generates the locally optimal trajectory within the homotopy class, even if the original path is not close to the local optimal. This is useful in reducing the number of samples and computational time necessary to find the optimal path, while reducing the trajectory cost.

Therefore, the SE-SCP algorithm is suitable for generating optimal paths through cluttered environments while obeying the spacecraft dynamics.

VI. CONCLUSIONS

In this paper we presented a novel spacecraft trajectory planning algorithm, through uncooperative cluttered environments, that uses the spacecraft dynamics and minimizes the fuel consumed by the spacecraft. Our SE-SCP algorithm mainly has two steps. During the spherical expansion step, the algorithm explores the workspace using a random or quasi-random sampling technique. This step generates coarse paths from the start position to the goal position. During the SCP step, the locally optimal trajectories are generated along these paths using SCP optimization. Therefore, as newer paths are discovered by the spherical expansion step, better trajectories are generated by the SCP step. Hence the algorithm guarantees locally optimal trajectories using a finite number of samples and a globally optimal trajectory as the number of samples tends to infinity.

REFERENCES

- [1] S. M. LaValle, *Planning Algorithms*. Cambridge Univ. Press, 2006.
- [2] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston, MA: MIT Press, 2005.
- [3] F. Baldini, S. Bandyopadhyay, R. Foust, S.-J. Chung, A. Rahmani, J.-P. de la Croix, A. Bacula, C. M. Chilan, and F. Y. Hadaegh, "Fast motion planning for agile space systems with multiple obstacles," in *AIAA/AAS Astrodynamics Specialist Conference*, (Long Beach, CA), 2016.
- [4] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 10, pp. 681–698, 1981.
- [5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.
- [6] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Transactions on robotics and automation*, vol. 8, no. 5, pp. 501–518, 1992.
- [7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.
- [8] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [9] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
- [10] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *Int. J. Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.
- [11] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *IEEE International Conference on Robotics and Automation*, (Sacramento, CA), pp. 1398–1404, IEEE, Apr. 1991.
- [12] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2719–2726, IEEE, 1997.
- [13] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 597–608, 2005.
- [14] R. Alterovitz, S. Patil, and A. Derbakova, "Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning," in *IEEE International Conference on Robotics and Automation*, pp. 3706–3712, IEEE, 2011.
- [15] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *IEEE International Conference on Robotics and Automation*, pp. 2421–2428, IEEE, 2013.
- [16] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 2520–2525, IEEE, 2011.
- [17] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 969–1002, 2015.
- [18] A. Richards, T. Schouwenaars, J. How, and E. Feron, "Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming," *J. Guid. Control Dyn.*, vol. 25, no. 4, pp. 755–764, 2002.
- [19] F. Fahroo and M. Ross, "Advances in pseudospectral methods for optimal control," in *AIAA Guidance, Navigation and Control Conference*, pp. 18–21, 2008.
- [20] B. Acikmese, D. Scharf, F. Hadaegh, and E. Murray, "A convex guidance algorithm for formation reconfiguration," in *AIAA Guidance, Navigation, and Control Conference*, (Keystone, CO), p. 6070, Aug. 2006.
- [21] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *J. Guid. Control Dyn.*, vol. 37, no. 6, pp. 1725–1740, 2014.
- [22] M. Grant and S. Boyd, "CVX: matlab software for disciplined convex programming," 2008.
- [23] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *arXiv preprint arXiv:1505.00023*, 2015.
- [24] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *Int. J. Robotics Research*, vol. 23, no. 7-8, pp. 673–692, 2004.