March 2016

An Efficient Path Robustness Metric for Compliant Robots

Nathan Harold Hughes
Worcester Polytechnic Institute

An Efficient Path Robustness Metric for Compliant Robots

Nathan Hughes
Autonomous Robotic Collaboration Lab
Worcester Polytechnic Institute
Dmitry Berenson (Advisor)
Autonomous Robotic Collaboration Lab
Worcester Polytechnic Institute

Abstract— **Compliant robots are able to make contact with obstacles safely, but many compliant robots suffer from significant actuation uncertainty. When executing a path that is collision-free contact with a nearby obstacle, though safe, can lead to the robot becoming "stuck" and being unable to continue execution. It is very computationally expensive to compute the probability of being stuck through forward-simulation and impractical to capture the distribution over the robot state in contact with parametric probability distributions, precluding the use of existing methods to evaluate the robustness of the path. This paper presents a robustness metric for compliant robots that captures the probability of the robot completing a given path, i.e. the probability of avoiding stuck configurations. Our metric is intended to be used inside a motion planner to determine the quality more efficiently than forward simulation. Our approach constructs a set of reachable C-space volumes between the way-points of a path that bound the set of configurations the robot could achieve given the actuation noise. We can then identify stuck configurations within these volumes and approximate their joint pre-image, which we then use to compute the probability of successfully reaching a way-point and subsequently the probability of reaching the path's goal. In our experiments we compare our method to forward-simulation on 3DoF and 6DoF free-flying robots in a narrow passage environment. We find that our method computes similar robustness predictions to forward-simulation but does so significantly more efficiently.**

I. Introduction

In recent years researchers have focused on the development of compliant robots for assembly and human-robot collaboration applications. These robots possess a key advantage: that they can make contact safely, even when the contact is unexpected. However, especially if the robot's compliance is implemented mechanically, compliant robots' motion often exhibits a significant amount of actuation noise. Thus, while the robot may make contact safely, the uncertainty in its motion makes it unclear if the robot will successfully execute a given task in the presence of obstacles, even if it is following a path that is collision-free. We seek to formulate an efficient metric to compute the robustness of a path. This metric should approximate the probability of the robot successfully reaching the goal when executing the given path.

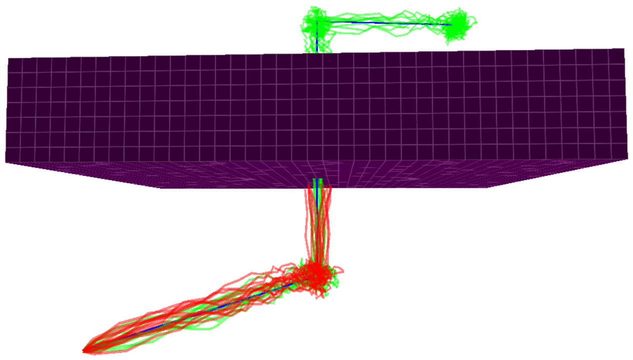
Fig. 1. A depiction of several sample trajectories for a 3DoF robot. The PD velocity controller tracking the path has a chance of getting stuck on the obstacle (trajectories in red) or reaching the goal successfully (trajectories in green)

Understanding what happens when the robot contacts obstacles is central to computing this metric. When a compliant robot contacts an obstacle two outcomes are possible: either the robot slides along the obstacle or it becomes "stuck" (i.e. unable to make further progress toward its goal). In this paper we assume that sliding is permissible for the robot but that being stuck is considered an execution failure. A motion planner that is planning a path for a compliant robot should thus attempt to plan a path that maximizes robustness, minimizing the probability of being stuck. Such a planner would need a function $S : \Pi \to \mathbb{R}$ that efficiently determines the probability of path Π not being stuck. For instance a trajectory optimizer would need to compute such a metric at every iteration.

A method to calculate this probability without making further assumptions about the distribution of the robot's state is readily apparent: Starting at the robot's initial configuration, we can sample many particles and forward simulate the robot's controller for each particle with perturbations drawn from the distribution of the actuation noise [13, 5, 16]. The percentage of particles that reach the goal region would then be an approximation of $S(\Pi)$. However, there are three important problems with this forward-simulation approach: First, forward-simulating particles for the duration of the path is computationally expensive. Consider that each simulation step requires at least a collision-check and possibly a procedure to resolve collisions through sliding and the number of simulation steps would scale with the length of the path. Second, a large number of particles would be needed to ensure that a representative sampling of the noise distribution is obtained (especially in high dimensions), thus compounding the computational expense. Third, the accuracy of this method is very sensitive to obstacle geometry. If the path enters a narrow passage and the actuation noise is high, the number of particles that enter the passage will be much smaller than the total number and this small number of particles may not approximate the probability distribution for the remainder of the path well. This issue could be resolved by re-sampling particles (as in a particle filter), but the sampling distribution for re-sampling is very unclear in the presence of obstacles.

The prupose of this major qualifying project was to investiagate a solution to these above limitations and create an algorithm that efficiently and accurately computes s, and then explore the effectiveness of this solution. Over the course of this project, we created a simulation environment for simple 3DoF and 6Dof robots and used these environments to test and verify the different approaches we created to compute s. After exploring several different approaches, the first of which was using graph centrality metrics to detect regions of configuration space that would contribute to trajectory failure, the second of which was identifying an explicit representation of the configuration of an abitrary controller and noise. Our final approach constructs a set of reachable C-space volumes between the way-points of a path that bound the set of configurations the robot could achieve given the actuation noise. We can sample these reachable volumes and determine which samples lie approximately on the contact manifold of the obstacles. We then evaluate which of the points on the contact manifold are stuck for the current way-point and approximate the pre-images of these points. The probability of being stuck is then the union of these pre-images. This union can be used to compute the probability of successfully reaching a way-point and the probabilities of reaching each way-point can be multiplied to compute the probability of reaching the goal.

The key advantage of our approach is its efficiency. As opposed to the forward-simulation approach, which simulates the robot's controller and collision response at every time step regardless of where the robot is in space, we only simulate configurations on the contact manifold for one time step to determine if they are stuck or sliding. Despite using geometric approximations in the computation of S our experiments in 3DoF and 6DoF virtual environments suggest that we obtain similar probabilities of success to the forward-simulation approach, even when the forward-simulation approach uses large numbers of samples. Thus, while we improve significantly in efficiency, we do not significantly compromise on the accuracy of the metric. This suggests that the metric we have developed will be useful in motion planning algorithms for compliant robots. As a final outcome of this project, this approach (and the majority of this paper) was submitted to the Robotics Science and Systems 2016 Conference.

In the remainder of the paper we present related work, the problem statement, and our approach to computing the robustness metric. We concluded with experiments that demonstrate the efficacy of the metric.

II. Related Work

There are many varieties of compliant robots that can make contact safely. These range from completely flexible robots [11, 19], to rigid-bodied robots with series-elastic actuators [18, 22], to flexible/steerable needles [24, 16], to robots where compliance is controlled in software [25, 14]. Our method is not focused on a particular type of compliant robot but rather requires that a simulator is available for the robot in question. A common strategy for these kinds of robots is to use an Impedance controller or PD controller to track a desired path. Likewise, our method assumes that the robot is endowed with a PD controller that tracks the input path.

Planning motion in the presence of actuation uncertainty dates back to the seminal work of Lozano-Perez et al. [9] on *pre-image backchaining*. A pre-image, i.e. a region of configuration space from which a motion command attains a certain goal recognizably, was used in a planner that produced actions guaranteed to succeed despite pose and action uncertainty. However, it was shown that constructing pre-images was computationally expensive [4, 3]. In our work, we approximate the pre-images of stuck configurations. While this process incurs some inaccuracy, we show that our method compares favorably with simulating particles.

Motion planning for robots under uncertainty has been studied extensively in recent years. Significant progress has been achieved through methods that plan in the belief space of the robot, i.e. the space of probability distributions over the robot's state. Belief-space planning in its general form is formulated as a Partially-Observable Markov Decision Process (POMDP). POMDPs are difficult to solve in high dimensions (though some approximation methods are making advances [6, 15]). Thus researchers have pursued relaxations of the problem which are low-dimensional [23], and/or locally explore the space around an initial trajectory [17, 23, 7]. Regardless of the planning approach, these methods seek to find a path that minimizes the probability that the robot collides with an obstacle. However, in our work the robot is assumed to be compliant, thus collisions are not inherently problematic, and may even be useful to complete a task. Instead, we focus on computing the probability that the robot will become stuck, which results in failure. This paper presents a metric that could be used by a planning algorithm like the ones above to compute the quality of a path.

Many methods compute the quality of a given motion by first determining the belief state of the robot that would result when executing that motion. This is done either by using parametric distributions or forward-simulating particles subject to actuation noise (see [8] for an overview). In terms of parametric distributions, the Gaussian distribution is commonly used to represent the belief state of the robot for motion planning [21, 23, 2, 20, 10]. However when the robot is in contact the distribution of states can be trans-dimensional and disconnected, making it difficult to fit parametric distributions to sets of possible states. Alternatively, forward-simulation of particles has been used to plan paths under actuation uncertainty as part of RRT-based methods [13, 5, 16]. Our approach avoids the need for an explicit representation of the belief state by computing the probability of the robot becoming stuck through geometric approximations.

Finally, our method computes a set of reachable volumes of configuration space for a given waypoint as part of the computation of our robustness metric. Reachable volumes for kinematic linkages, such as robot arms, were introduced by [12]. These reachable volumes allow fast sampling of valid configurations of a linkage given a fixed root and end effector. [12] shows their applicability for very high (70DoF or greater) robots. We extend this concept to *trajectory-linkages*, i.e. sequences of configurations which are used to compute reachable volumes in C-space.

III. Problem Statement

The goal of this project is to provide an efficient metric to evaluate how successful a trajectory controller will be at tracking an arbitrary path Π composed of a sequence of way-points $\pi_0, \pi_1, \ldots, \pi_g$, i.e. a mapping $S : \Pi \rightarrow \mathbb{R}$ that outputs the probability of reaching π_g. We assume that each way-point in Π is collision free, and that each way-point π_i is visible from π_{i-1}. A configuration q' is visible from configuration q if for $\theta \in [0, 1]$, $\theta q + (1-\theta)q'$ is collision-free. The execution of the trajectory is deemed successful if the end configuration of the robot is within the Euclidean ball $\mathbb{B}_\epsilon(\pi_g)$, where $\epsilon > 0$ is the amount of error allowed. We assume execution terminates once the robot reaches a configuration inside $\mathbb{B}_\epsilon(\pi_g)$.

Throughout this paper, we use the notation $X(q, t)$ to denote the random process describing the configuration of the robot, q, at time t. Using this notation, our trajectory robustness metric should approximate

$$S(\Pi) = \lim_{t \to \infty} \mathbb{P}(X(q, t) \in \mathbb{B}_\epsilon(\pi_g)) \tag{1}$$

We assume that independent of the trajectory controller, the robot is displaced by some arbitrary amount every time-step by actuation noise. The displacement at each time-step is denoted as $W(t)$. We assume that $\mathbb{P}(W(t))$ has convex finite support. This paper focuses on path robustness in the presence of actuation noise, so we assume no sensor noise is present during execution.

We assume that the robot is only prevented from making forward progress towards the goal region when in contact with obstacles, i.e. no other constraints on the motion of the robot exist (such as non-holonomic or torque constraints). We also assume that the robot is allowed to make sliding contact with obstacles — one of the major advantages of compliant robots. While sliding contact allows forward progress, it is possible for the robot to reach a configuration where it is no longer possible to slide along the obstacle while making progress towards the next way-point. For the rest of the paper, we refer to such configurations as "stuck" configurations. We assume that we are given a function $f : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ which simulates the execution of the trajectory in the presence of obstacles for a single time-step. For example, to compute the result of moving from a configuration $q_{current}$ toward a way-point π_t for one time-step, we would compute $q_{result} = f(q_{current}, \pi_t)$.

Finally, we assume the trajectory controller is a stable PD velocity controller that outputs velocities to a lower-level controller responsible for the actuation of the robot. This is the standard control structure used on many robots, e.g. the PR2.

IV. Computing Path Robustness

Our approach models the actuation noise of the robot as a random process that is independent of the trajectory controller. When the path is near obstacles, some trajectories that result from the combination of this actuation noise and the trajectory controller reach stuck configurations; the likelihood that these trajectories will occur is the complement to the probability in Equation 1. We compute this likelihood for each waypoint independently and then multiply the likelihoods to arrive at our metric. Below we describe how to model the current configuration of the robot as a random process and then how to bound the volume of the random process. This volume is sampled to find stuck configurations and the pre-images of stuck configurations are approximated to arrive at a probability of being stuck. Below we describe in detail how each step of this computation is performed.

A. Modeling the Trajectory Controller as a Random Process

This section describes how the PD velocity controller and actuation noise of the robot interact. Denoting the proportional constant and derivative constant of the controller as K_p and K_d respectively, and denoting the time-step of the PD velocity controller as Δt, we can write the update equation for the random process $X(q, t)$ as:

$$\begin{aligned} X_i &= [K_p e_i + K_d \dot{e}_i + W_i]\Delta t + X_{i-1} \\ e_i &= \pi_t - X_i \\ \dot{e}_i &= \frac{X_{i-1} - X_i}{\Delta t} \end{aligned} \tag{2}$$

Here we use the notation X_i and W_i to denote the state of the random processes $X(q, t)$ and $W(t)$ at the time step t_i. This update equation is given in terms of the set-point π_t, which is chosen from the way-points that make up the desired path. We use the following policy to select the target way-point:

$$\pi_t = \begin{cases} \pi_j & X_i(q) \in \mathbb{B}_\epsilon(\pi_j) \wedge j > i \\ \pi_i & \text{otherwise} \end{cases} \tag{3}$$

where π_i is the current element of Π chosen as the set point. π_t starts at π_1 and the initial error is $\pi_1 - \pi_0$. The pseudo code describing the implementation of this controller is shown in Algorithm 1.

This way-point selection policy guarantees that the robot is always within some ϵ of the previous way-point when a way-point is selected as a set-point. This means that independent of when a way-point is selected as the set-point, the process will generate the same set of trajectories for the duration of the time that the way-point remains the set-point. This allows us to examine the behavior of the trajectory controller when trying to reach each way-point individually, i.e. as if the starting configuration of the controller is within $\mathbb{B}_\epsilon(\pi_{i-1})$. For each way-point we can calculate how likely the controller is to fail to reach that way-point, i.e. to be stuck, conditional on starting at the previous way-point. Using these probabilities, we can approximate Equation 1 as follows:

$$\lim_{t\to\infty} \mathbb{P}(X(q,t) \in \mathbb{B}_\epsilon(\pi_g)) = \prod_{i=1}^{N} \lim_{t\to\infty} \mathbb{P}(X_t \in \mathbb{B}_\epsilon(\pi_i) | X_0 \in \mathbb{B}_\epsilon(\pi_{i-1})) \tag{4}$$

Algorithm 1: PD controller implementation with way-point selection

```
inputs: $K_p, K_d, \Pi, \Delta t, t_{\max}, \epsilon$
$x_i \leftarrow \Pi_0$;
$i \leftarrow 1$;
$\pi_t \leftarrow \Pi_1$;
$t \leftarrow 0$;
while $t < t_{\max}$ do
    $x_i \leftarrow$ GetCurrentPosition();
    $t \leftarrow t + 1$;
    $j \leftarrow i$;
    while $j <$ Length($\Pi$) do
        if Distance($x_i$, $\Pi_i$) $< \epsilon$ then
            $i \leftarrow i + 1$;
            $\pi_t \leftarrow \Pi_i$;
            Break;
        $j \leftarrow j + 1$;
    $u \leftarrow K_p(\pi_t - x_i)\Delta t + K_d(x_{i-1} - x_i)$;
    SetControlOutput($u$);
    $x_{i-1} \leftarrow x_i$;
    Wait($\Delta t$);
```

B. Reachable Volumes of Random Processes

We now present a method of computing bounds on the configuration of the robot for a given time-step and set-point. This is important to our approach because it allows us to define a subset of the configuration space to check for stuck configurations rather than naively simulating forward. We do this by extending the concept of reachable volumes of linkages to trajectories. As presented in [12], the reachable volume of a joint in a linkage is the set of all possible points in the workspace that the center point of the joint could occupy given every possible workspace position of the previous joints in the linkage. We extend this concept to trajectories by considering each joint in the linkage to be a configuration $X(q, t_i)$; i.e. a linkage is a sequence of configurations. For the rest of this section, we refer to the sequence of configurations $l_n = (q_0, q_1, \ldots, q_n)$ as a trajectory-linkage. The reachable volume of a trajectory-linkage is the set of configurations that the trajectory-linkage could end at, given any sequence of previous configurations in the trajectory-linkage that may have occurred.

The advantage of using reachable volumes is that it is possible to sample configurations of linkages while fixing the end-effector of the linkage to a certain position in the workspace [12]. This same framework allows us to examine which trajectories lead to a stuck configuration. To do this, we must first define explicitly what the reachable volume for a trajectory-linkage is. The reachable volume of a trajectory linkage l_n is recursively defined as (adapted from [12]):

$$RV(l_n) = RV(l_{n-1} : l_n) \oplus RV(l_{n-1}) \tag{5}$$

Here, we denote the reachable volume of the entire trajectory-linkage l_{n-1} as $RV(l_{n-1})$ and the reachable volume of the link (q_{n-1}, q_n) as $RV(l_{n-1} : l_n)$. The reachable volume of a single link (such as the link $l_{n-1} : l_n$) in a trajectory-linkage is the set of configurations that could be reached in a single time-step by the controller, given that the first configuration in the link is grounded at the origin. This is expressed as:

$$RV(l_i : l_{i+1}) = \{a + \omega\} ; a \in \mathbb{R}^d \tag{6}$$

where a represents the possible displacement produced by the trajectory controller and ω represents the finite volume of displacements from which $W(t)$ is draw. The reachable volume of a zero length trajectory-linkage $RV(l_0)$ is $\mathbb{B}_\epsilon(q_0)$.

This reachable volume of a link is not informative as it encompasses all of $\mathcal{C}$ when a is unbounded. Instead, we examine the constrained reachable volume of a trajectory linkage. For a trajectory-linkage of length greater than two, we can express the constrained reachable volume of some trajectory linkage l_i, $CRV(l_i)$ in terms of constraints on the displacement vector a:

$$\begin{aligned} &a \in \{K_p(\pi_t - q_{i-1})\Delta t + K_d(q_{i-2} - q_{i-1})\} \\ &q_{i-1} \in CRV(l_{i-1}) \\ &q_{i-2} \in CRV(l_{i-2}) \end{aligned} \tag{7}$$

Illustrations of the constrained reachable volumes for a path segment are shown in Figure 2.

We show an important property for the reachable volume of an arbitrary trajectory-linkage: that it is possible to calculate the bounds of the constrained volume recursively from the last bounds of a constrained reachable volume. We denote the bounds of the constrained reachable volume of a trajectory-linkage l_i as B_i. We show this for a single dimension of the reachable volume and a set-point π_t in the following proof.

Theorem 1: Let $\alpha = K_p \Delta t \pi_t$, $\beta = 1 - K_p \Delta t - K_d$, $\gamma = K_d$, and $K_p, K_d > 0$. For $|l_i| > 1, \beta < 0$, $\min B_i$ and $\max B_i$ are given as:

$$\begin{aligned} \min B_i &= \alpha + \beta \max B_{i-1} + \gamma \min B_{i-2} + \min \omega \\ \max B_i &= \alpha + \beta \min B_{i-1} + \gamma \min B_{i-2} + \min \omega \end{aligned} \tag{8}$$

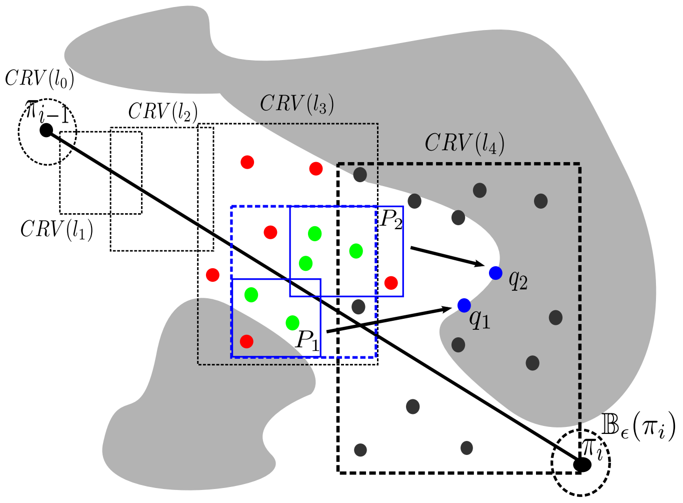

Fig. 2. A depiction of how we compute the probability that the trajectories generated from one way-point to another will fail (in this case, π_{i-1} to π_i).The reachable volume for each trajectory linkage $CRV(l_k)$ is a dashed black box. Stuck samples for the current reachable volume (l_4) are blue and the other samples as black. Each stuck sample (q_1 and q_2) corresponds to their respective pre-images (P_1 and P_2). The bounding box of the pre-images is dashed blue. Samples inside the union of the pre-images are green, other samples from $CRV(l_{k-1})$ are red.

For $|l_i| > 1, \beta >= 0$:

$$\begin{aligned} \min B_i &= \alpha + \beta \min B_{i-1} + \gamma \min B_{i-2} + \min \omega \\ \max B_i &= \alpha + \beta \max B_{i-1} + \gamma \min B_{i-2} + \min \omega \end{aligned} \tag{9}$$

Proof: We can write the constrained reachable volume for any trajectory linkage l_i as:

$$\alpha \oplus (\beta \cdot CRV(l_{i-1})) \oplus (\gamma \cdot CRV(l_{i-2})) \oplus \omega \tag{10}$$

Provided that the previous two constrained reachable volumes are convex (W_i was assumed to have convex finite support, thus ω is convex and finite), the resulting constrained reachable volume is convex as it would be a Minkowski sum of convex sets. The initial reachable volume, $\mathbb{B}_\epsilon(q_0)$ is also convex. The constrained reachable volume of l_1 is also convex as it is the Minkowski sum of the noise and a affine transformation of the initial reachable volume. By induction the constrained reachable volume is always convex.

It can be shown that the Minkowski sum of the convex hulls of convex sets is equal to the convex hull of the Minkowski sum of the sets. The bounds in one dimension can be calculated from the convex hull. Using this property, Equation 10 can be bounded by summing the bounds of the following sets: $\beta \cdot CRV(l_{i-1}), \gamma \cdot CRV(l_{i-2})$ and ω. This yields Theorem 1.

■

It is also important to show that the process $X(q,t)$ is bounded by the constrained reachable volume for the corresponding trajectory-linkage for every time-step.

Theorem 2: $X_i \subseteq CRV(l_i)\ \forall i \in (0, 1, \dots)$

Proof: It is trivial to show that $X_0 \subseteq CRV(l_0)$. Additionally by substitution, it is trivial to show that $X_1 \subseteq CRV(l_1)$. For an arbitrary time-step i, we have that $X_i = K_p(\pi_t - X_{i-1})\Delta t + K_d(X_{i-2} - X_{i-1}) + W_i \Delta t + X_{i-1}$. Assuming that $X_{i-1} \subseteq CRV(l_{i-1})$ and $X_{i-2} \subseteq CRV(l_{i-2})$, it is clear that X_i can never exceed the bounds of $CRV(l_i)$ presented in Theorem 1. Therefore, $X_i \subseteq CRV(l_i)\ \forall i \in (0, 1, \dots)$ by induction. ■

Finally, we show that when neglecting obstacles, the process is guaranteed to reach $\mathbb{B}_\epsilon(\pi_t)$ at some time. This justifies our approach of individually analyzing each way-point as a set-point; if this guarantee was not present we would need to also consider the probability trajectories fail due to other constraints (e.g. torque limits). For this, we assume that the PD controller is stable.

Theorem 3: $\exists t_i : X(q, t_i) \in \mathbb{B}_\epsilon(\pi_n)$

Proof: At t_0, the reachable volume is $\mathbb{B}_\epsilon(\pi_{n-1})$. It is clear from Theorem 1 that the bounds of the reachable volume never decrease, so the reachable volume will always be able to contain a ball $\mathbb{B}_\epsilon$. It is also clear that the reachable volume continues to advance toward π_n at every time-step. Once $\mathbb{B}_\epsilon$ is contained in the reachable volume, it will not advance further since the controller always moves the current configuration toward π_n. We know that $X(q, t_i)$ must be contained in the reachable volume and that its probability distribution has non-zero support everywhere within the bounds. Thus $X(q,t)$ will eventually cover all of the reachable volume, and because $\mathbb{B}_\epsilon$ is contained in the reachable volume it follows that $\exists t_i : X(q, t_i) \in \mathbb{B}_\epsilon(\pi_n)$. ■

C. Computing Stuck Configurations

Stuck configurations are configurations where the robot cannot make any more progress towards the next way-point. Our assumptions in our problem statement allow us to conclude that these stuck configurations only occur on the obstacle manifold. The region of $\mathcal{C}$ where the robot is in collision with obstacles in the workspace is $\mathcal{C}_{\text{obs}} = \{q \in \mathcal{C} : P_r(q) \cap P_o \neq \varnothing\}$ (P_o is the geometry of all the obstacles in the workspace and $P_r(q)$ is the geometry of the robot in the workspace for some configuration q). The contact manifold for all obstacles in this space is given as $\mathcal{C}_{\text{obs}} \backslash int(\mathcal{C}_{\text{obs}})$. This is a zero measure set with respect to the configuration space and therefore impossible to sample via rejection sampling.

To identify stuck configurations, we need an approximate representation of this manifold:

$$\mathcal{O} = \{q \in \mathcal{C}_{\text{obs}} | \exists q' \in \mathcal{C}_{\text{free}} : \text{norm}(q - q') < r_{\max}\} \tag{11}$$

$\mathcal{O}$ represents the region of configuration space where the configuration generated by the PD velocity controller and actuation noise of the robot results in collision with an obstacle, forcing the robot to slide along or be stuck on an obstacle. $r_{\max}$ is a parameter that allows us to trade off between ease of sampling and the accuracy of projection onto the manifold. As $r_{\max}$ increases, it becomes more likely to be able to generate samples inside $\mathcal{O}$, but it becomes more and more unreliable to project these samples onto the contact manifold.

One of the advantages of our approach is that we can identify bounds on where the robot will be in the configuration space for a given time-step and set-point using constrained reachable volumes. This allows us to only examine the portion

Algorithm 2: isStuck

```
input  : A starting configuration $q$ and a tolerance $\epsilon_{\text{stuck}}$
returns: Whether the configuration is stuck or not
$q_{\text{current}} \leftarrow q$;
while inCollision($q_{\text{current}}$) do
    $q_{\text{current}} \leftarrow q_{\text{current}} + \nabla\, q_{\text{current}} \cdot \Delta t$;
$q_{\text{prev}} \leftarrow q_{\text{current}}$;
$q_{\text{current}} \leftarrow$ getTarget($q_{\text{current}}$);
while inCollision($q_{\text{current}}$) do
    $q_{\text{current}} \leftarrow q_{\text{current}} + \nabla\, q_{\text{current}} \cdot \Delta t$;
if isWithin($q_{\text{current}}, q_{\text{prev}}, \epsilon_{\text{stuck}}$) then
    return true;
return false;
```

of $\mathcal{O}$ that is relevant to the current set-point. We do this by uniformly sampling the configuration space inside the bounds of the constrained reachable volume for that time step. Once we have this set of samples, we identify members of $\mathcal{O}$. We do this by identifying which samples inside the set are in collision with an obstacle, and then check to see if any of the free samples is within r_{max} of the sample in collision. For all of the samples in $\mathcal{O}$, we check which of these samples is stuck (which we discuss below), and then use the stuck samples as a discrete representation of the region of configurations inside the reachable volume that are stuck.

In order to reduce the amount of simulation necessary, we determine whether a sample is stuck by projecting the sample to the contact manifold and then simulating forward one time-step using f (described previously). If the particle makes forward progress during this time-step (which we determine by checking if the distance between the original projected sample and the resultant configuration from simulating one time-step forward is greater than ϵ_{stuck}), then the sample is not a stuck configuration. We use this method as long as the resultant configurations are still inside the bounds of the current and next reachable volume. If the resultant configuration is outside these bounds, then the obstacle must have pushed the robot out of the reachable volume. There are two options when this occurs: we could forward simulate until the robot is either stuck or back inside the reachable volume, or we could simply assume that the sample is stuck. The second is a conservative assumption, and reduces the amount of simulation we perform in calculating the metric. An example of a situation is when the robot encounters some sort of wedge close to the path. The wedge would slowly push the robot away from the path, and then once past the wedge, the robot would be able to reach the set-point. In this case forward simulation could be used to more accurately determine whether the sample is stuck or not. In our experiments we make the conservative assumption to reduce the computational cost of simulation.

The simulation function f is robot-specific and our metric does not depend on how the simulation is performed. For our experiments f was implemented as follows: The robot moved with the velocity commanded by the controller for one time step. If the robot ended in collision, we simulated

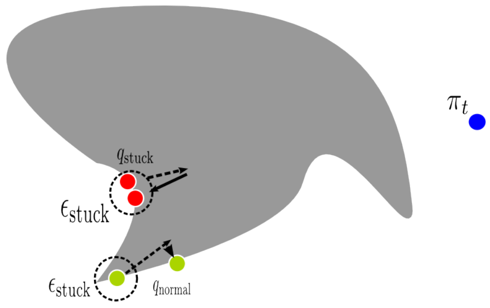

Fig. 3. Two different starting configurations of the robot (are shown as well as the configurations that result from forward simulation.

sliding contact by using gradient descent to project the desired configuration onto the obstacle manifold. The loss function in this case is the signed distance between the robot and the obstacle manifold. We compute this gradient much like [1], which uses a discretization of the workspace to calculate the gradient of the signed distance field in the workspace, and then computes the Jacobian of the robot to transform the cost gradient from workspace to configuration space. Our gradient is given as:

$$\nabla q = \sum_{i=1}^{n} J(q, x_i)^{\intercal} \nabla x_i^{\intercal} \tag{12}$$

The algorithm for computing whether a sample is stuck is given in Algorithm 2. An illustration of how this algorithm determine whether a members of $\mathcal{O}$ is stuck or not is shown in Figure 3. Note that this implementation does not consider the effects of friction, which would require more sophisticated simulation methods.

D. *A metric for path robustness*

We can now approximate the probability given in Equation 4 using the concepts discussed above. We do this for each way-point π_i by starting a trajectory-linkage at the previous way-point and iteratively increasing the length of the trajectory-linkage until it contains $\mathbb{B}_\epsilon(\pi_i)$. For each iteration, we identify stuck samples as described in Section IV-C. For all of these stuck samples we set the last configuration of the trajectory-linkage to each of the stuck configurations and calculate the subset of the previous reachable volume that could possibly result in the stuck configuration (See Figure 2 for a depiction of these subsets), which we call pre-images of the configuration. We can bound a pre-image of a stuck configuration using Equations 8 and 9. For a given configuration q_i, the bounds of the corresponding pre-image P_i for each dimension are

$$\begin{aligned} \min P_i &= \frac{q_i + \min \omega - \alpha}{1 - K_p \Delta t} \\ \max P_i &= \frac{q_i + \max \omega - \alpha}{1 - K_p \Delta t} \end{aligned} \tag{13}$$

We then clip these bounds to the previous reachable volume (i.e. there may exist some control inputs that could result in q_i that are not inside the previous reachable volume), as shown in Figure 2. These bounds for the pre-image P_i approximate the true set of configurations from the previous reachable volume that could result in the stuck configuration q_i. We use these pre-images to determine the likelihood of being stuck in the current reachable volume. Explicitly, we want the probability that a trajectory-linkage could result in a stuck configuration, i.e. the probability over the union of all P_i. This is given (for n pre-images) as:

$$\sum_{i=1}^{n} \int_{P_i} \mathbb{P}(X(q,t)) + \sum_{j=1}^{n} \sum_{1 \le i_1 \ldots < i_j \le n} \int_{P_{i_1} \cdots \cap P_{i_j}} \mathbb{P}(X(q,t))(-1)^j \tag{14}$$

which is intractable to compute except for very small n.

To approximate this union we first construct a bounding box of all the pre-images. Then, for each free sample of the previous reachable volume, we first check to see if the sample is inside the bounding box of all the pre-images. If so, we check to see if the sample is a member of any of the pre-images. The approximation to Equation 14 is then the number of free samples that are inside a pre-image divided by the total number of samples taken.

We then approximate the probability that a trajectory will fail in the current reachable volume by weighting the probability that the trajectories in that volume will reach a stuck configuration (our approximation to Equation 14) by the likelihood of stuck particles occurring in the current reachable volume. The algorithm to compute this likelihood for a given reachable volume is shown in Algorithm 3 and the full algorithm to compute our robustness metric is given in Algorithm 4.

Algorithm 3: Likelihood Computation

```
input   : A set of pre-images $P$ and set of samples $S$
returns: Likelihood of stuck samples occurring

$B \leftarrow$ computeBoundingBox($P$);
$n \leftarrow 0$;
$m \leftarrow 0$;
for $s \in S$ do
    if $\neg$ inCollision($s$) $\wedge s \in B$ then
        for $P_i \in P$ do
            if $s \in P_i$ then
                $n \leftarrow n + 1$;
                break;
    if isStuck($s$) then
        $m \leftarrow m + 1$;
return $\frac{n \cdot m}{|S|^2}$;
```

E. Design of Simulation Environment

To demonstrate the efficieny of our method, we first needed to create a simulation environment to compare the predicted robustness of the metric with the true robustness of the path. This simulation environmnet ideally would accomplish the following:

Algorithm 4: Robustness Metric Computation

```
input   : $K_p, K_d, \Delta t, \Pi, \epsilon$
returns: Probability trajectories will succeed in reaching
         target

$p_{\text{success}} \leftarrow 1$;
for $\pi_i \in \Pi$ do
    while goalRegionOutside($CRV(l_i)$, $\pi_i$, $\epsilon$) do
        $CRV(l_i) \leftarrow$ computeBounds($\pi_i$, $l$);
        $S \leftarrow$ sampleVolume($CRV(l_i)$);
        $p_i \leftarrow 1$;
        for $s \in S \cap \mathcal{O}$ do
            $P \leftarrow \varnothing$;
            if isStuck($s$) then
                $P_i \leftarrow$ computePreimage($s$);
                $P \leftarrow \{P_i\} \cup P$;
            $p_i \leftarrow (1-$ likelihood($P$, $S$) $) \cdot p_i$;
        $p_{\text{success}} \leftarrow p_{\text{success}} \cdot (1- p_i)$;
return $p_{\text{success}}$;
```

1) Allow the simulation of a variety of different robots
2) Allow visualization of a single run or a batch of runs
3) Have a flexible and modular design
4) Allow the user to easily configure simualation parameters

In order to construct this simulation environent, we explored several different options; this happened in parrallel with the exploration of different approaches to calculate our desired metric. Initially, we wrote a simple python script with little structure that allowed us to quickly compare the latest approach to simulation results in two dimensional code. From this approach, we discerned that a good way to pass in simulation parameters was to maintain a configuration file, and that this configuration file needed to have a relatively flexible format, as different approaches required different kinds of parameters. To visualize result, the simulator saved images of each timestep, and then ffmpeg was used to convert the image sequence into a movie. However, this made it hard to easily maintain records of results.

From this simple two dimensional simulation environment, we developed a simulation environment capable of simulating three and six dimensional rigid bodies. There were two components to this simulator: the actual simulator, and the front-end responsible for visualizing the simulation results, configuring the simulator, and saving the results in a persistent manner. For the actual simulator, we explored two options: using FleX (a defomrable object simulator developed by NVidia) and custom written code or using a collision simulation method developed by the ARC lab and custom written code. We chose the latter, as the results FleX produced were too dependent on the parameters for FleX itself.

For the frontend, we explored two options to visualize the simulation results. The first was a custom written OpenGL visualizer. The development time necessary to make the custom written visualizer flexible enough was too great, so instead

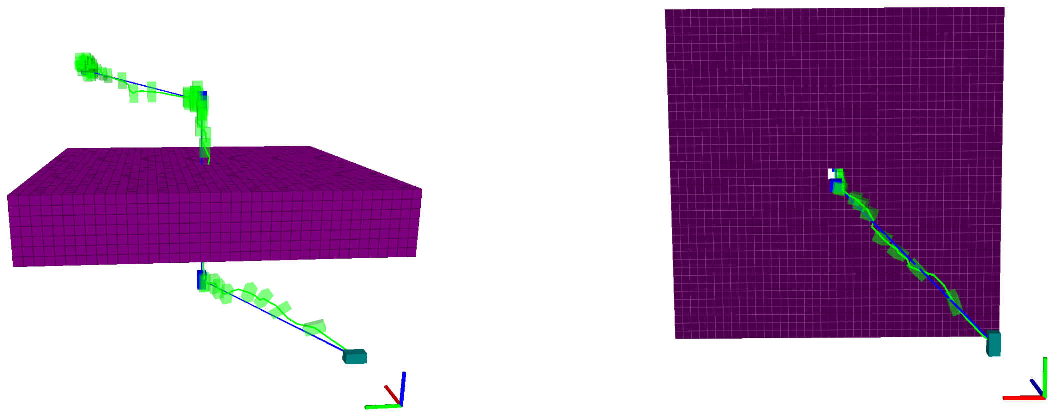

Fig. 4. Two views of our 6DoF simulation environment. The given path is shown in blue (along with the position of the robot at each way-point) and the obstacle is shown in purple. The path of the center of the robot during trajectory execution is shown as the green line, and sample orientations of the robot for different time-steps are also shown in green.

we used RViz and markers to display simualtion results. We explored using MongoDB to keep track of simulation runs and other information, but due to the lack of a good C++ driver for MongoDB, the fact that the simulator was being developed on several different computers, and time constraints, we never incorporated saving results to MongoDB in the final simualtor.

Finally, for the configuration of the simulation environment, we briefly explored using the CMake file responsible for compiling the simulator to pass in parameters. This was not a viable option, because we wanted to maintain sets of parameters for each robot and trial we ran. We developed a configuration file format based on XML, and a configuration file system that leverages the C++ Boost Libraries.

Together, our final simulation environment consists of a back-end simulator that is a C++ library using voxel grid and signed distance field code developed by the ARC lab and custom written code to create a robot given certain parameters and actuate the robot through the modeled environment. Our final simulation environment also contains a front-end system that uses a Boost Library to parse a XML structured configuration file, run various simulations, and display the results to RViZ. As a link between the back-end simulator and the front-end visualizer and configuration file, we used ROS.

V. Experiments

To demonstrate the efficiency of our method, we performed two comparisons of our robustness metric and the probability of success based on forward-simulation for a small 3DoF robot and a larger 6DoF free-flying rigid body. Both experiments were run in the same workspace: a $10 \times 10 \times 10$ unit cube with a wall containing a narrow passage in the center between $z = 4$ and $z = 6$ (see Figure IV-D). We specified the path for the robot manually. The 3DoF robot was a unit cube with a side length of 0.3 units that could only translate. The 6DoF rigid body was a box with width 0.6 units and height and length 0.15 units that could translate and rotate. The path for each robot originates in the lower left of the coordinate space and finishes on the other side of the narrow passage in the upper right portion of the workspace (see IV-D). The planned path for the 6DoF experiment was made so that the robot must re-orient to align itself with the narrow passage before proceeding through.

TABLE I
Comparison of Robustness Metric and Simulation: 3DoF Experiment

Metric (100 Runs)

Samples	Probability $\mu(\sigma)$	Time (s) $\mu(\sigma)$
300	$0.864(0.021)$	$2.74(0.195)$
400	$0.825(0.021)$	$3.884(0.280)$
500	$0.797(0.023)$	$5.045(0.381)$

Simulation

Particles	Probability	Time (s)
1000	0.800	17.67
10000	0.816	179.4
100000	0.817	1735

We ran our experiments on a computer with an Intel i7-3770 processor. All code was written in C++. We used a uniform voxel-grid representation of the workspace with a voxel resolution of 0.25 units. The narrow passage had side length 0.5 units in the x and y dimension. We used a truncated normal distribution to approximate the actuation noise of the robot during simulation and a uniform normal distribution to approximate the probability distribution inside a constrained reachable volume; variances and bounds are reported for each experiment separately. Our trajectory controller had a K_p of 10.0, K_d of 0.05 and each time-step lasted for a duration of 0.01 seconds. Our choice of ϵ for $\mathbb{B}_\epsilon(\pi_t)$ was 0.1 for both experiments and simulations were run for 250 time-steps.

For the 3DoF robot, we computed our path robustness metric for $300, 400$ and 500 samples per reachable volume. We ran forward-simulation with $1000, 10000$ and 100000 particles. We used an ϵ_{stuck} of 0.2 for these experiments. The simulated noise had a variance of 8.0 units and we used a uniform distribution with bounds of ± 15 units in each dimension for the path robustness metric. For our path metric, we ran 100 consecutive trials with different random seeds, and report the

TABLE II
COMPARISON OF ROBUSTNESS METRIC AND SIMULATION: 6DOF EXPERIMENT

Metric (100 Runs)

Samples	Probability $\mu(\sigma)$	Time(s) $\mu(\sigma)$
300	$0.765(0.023)$	$2.11(0.111)$
400	$0.747(0.022)$	$2.91(0.131)$
500	$0.729(0.023)$	$3.752(0.181)$

Simulation

Particles	Probability	Time(s)
1000	0.747	26.75
10000	0.7737	277.3
100000	0.763	2767

mean and variance of these trials in Table I. We also list the results from the simulation. From these results, it is clear that given enough samples, our metric approximates the simulated probability in a significantly smaller amount of time.

For the 6DoF robot, we again ran our path metric for $300, 400$ and 500 samples. We used the same noise parameters for these experiments as the 3Dof experiment, with the exception of the variance and bounds of the rotational degree of freedom of the robot of the robot (1.0 and 2.0 units respectively). However, our initial sampling-based approach to approximating the union of pre-images failed to generate enough samples inside the pre-images of the stuck samples. This is not surprising, as the 6D space will be more sparse than the 3D one in terms of pre-images. Instead, we simply check how many free samples are inside the bounding box of the pre-images, yielding a more conservative approximation that does not require as many samples (this method is also faster). The 6DoF results are reported in Table II. We also ran simulations with 1000, 10000 and 100000 samples per reachable volume using the less conservative method, which generated the probabilities 0.994, 0.953 and 0.833, which shows that even though our original method to compute likelihood is sensitive to dimensionality, increasing the sampling resolution reduces the error. We also show the simulation results for 1000, 10000 and 100000 particles in II. Again, our metric produces comparable results in significantly less time than forward-simulation.

VI. CONCLUSION

We have presented a path robustness metric that approximates the probability of successfully executing a path for a compliant robot. The metric generates comparable predictions to forward-simulation but does so significantly more efficiently. The key to our approach is to reason geometrically about the probability of arriving at stuck configurations, where the robot cannot progress further toward its next way-point. Our approach constructs a set of reachable C-space volumes between the way-points of a path that bound the set of configurations the robot could achieve given the actuation noise. We can then identify stuck configurations within these volumes and approximate their joint pre-image, which we then use to compute the probability of successfully reaching a way-point and subsequently the probability of reaching the path's goal. Our future work will investigate using the presented metric inside a motion planning algorithm for compliant robots, where its efficiency would allow us to evaluate the quality of a path much faster than existing methods.

REFERENCES

[1] Dmitry Berenson, Thierry Siméon, and Siddhartha S Srinivasa. Addressing cost-space chasms in manipulation planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4561–4568. IEEE, 2011.

[2] A. Bry and N. Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[3] J. Canny. On computability of fine motion plans. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 1989.

[4] M. Erdmann. Using Backprojections for Fine Motion Planning with Uncertainty. *The International Journal of Robotics Research*, 5(1):19–45, March 1986. ISSN 0278-3649. doi: 10.1177/027836498600500102.

[5] Yifeng Huang and K. Gupta. Rrt-slam for motion planning with motion and map uncertainty for robot exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2008.

[6] Hanna Kurniawati, D Hsu, and WS Lee. SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In *Robotics: Science and Systems*, 2008.

[7] Alex Lee, Y Duan, S Patil, John Schulman, Zoe McCarthy, Jur Van Den Berg, Ken Goldberg, and Pieter Abbeel. Sigma Hulls for Gaussian Belief Space Planning for Imprecise Articulated Robots amid Obstacles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[8] Z. Littlefield, D. Klimenko, H. Kurniawati, and K. E. Bekris. The importance of a suitable distance function in belief-space planning. In *International Symposium on Robotic Research (ISRR)*, September 2015.

[9] Tomas Lozano-Perez, Matthew Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1), 1984.

[10] Brandon D. Luders, Sertac Karaman, and Jonathan P. How. Robust sampling-based motion planning with asymptotic optimality guarantees. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013.

[11] Andrew D Marchese, Robert K Katzschmann, and Daniela Rus. Whole arm planning for a soft and highly compliant 2d robotic manipulator. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 554–560. IEEE, 2014.

[12] Troy McMahon, Stephan Thomas, and Nancy M Amato. Sampling-based motion planning with reachable volumes: Theoretical foundations. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6514–6521. IEEE, 2014.

[13] N. A. Melchior and R. Simmons. Particle rrt for path planning with uncertainty. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1617–1624, 2007. doi: 10.1109/ROBOT.2007.363555.

[14] Danial Nakhaeinia, Pascal Laferriere, Pierre Payeur, and Robert Laganiere. Safe close-proximity and physical human-robot interaction using industrial robots. In *Computer and Robot Vision (CRV), 2015 12th Conference on*, pages 237–244. IEEE, 2015.

[15] S.C.W. Ong, S.W. Png, D. Hsu, and W.S. Lee. POMDPs for Robotic Tasks with Mixed Observability. In *Robotics: Science and Systems (RSS)*, 2009.

[16] Sachin Patil, J van den Berg, and Ron Alterovitz. Motion planning under uncertainty in highly deformable environments. In *Robotics: Science and Systems*, 2011.

[17] Robert Platt, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-Perez. Belief space planning assuming maximum likelihood observations. In *Robotics Science and Systems (RSS)*, 2010.

[18] G.A. Pratt and M.M. Williamson. Series elastic actuators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 399–406, Aug 1995.

[19] Robert F. Shepherd, Filip Ilievski, Wonjae Choi, Stephen A. Morin, Adam A. Stokes, Aaron D. Mazzeo, Xin Chen, Michael Wang, and George M. Whitesides. Multigait soft robot. *Proceedings of the National Academy of Sciences*, 108(51):20400–20403, 2011.

[20] Wen Sun, S. Patil, and R. Alterovitz. High-frequency replanning under uncertainty using parallel sampling-based motion planning. *IEEE Transactions on Robotics (T-RO)*, 31(1):104–116, Feb 2015.

[21] R. Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[22] N.G. Tsagarakis, S. Morfey, G.M. Cerda, Li Zhibin, and D.G. Caldwell. Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 673–678, May 2013.

[23] Jur van den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, September 2012. ISSN 0278-3649.

[24] Jing Xiong, Xia Li, Yangzhou Gan, and Zeyang Xia. Path planning for flexible needle insertion system based on improved rapidly-exploring random tree algorithm. In *Information and Automation, 2015 IEEE International Conference on*, pages 1545–1550. IEEE, 2015.

[25] Taizo Yoshikawa and Oussama Khatib. Compliant motion control for a humanoid robot in contact with the environment and humans. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 211–218. IEEE, 2008.