Learning Controller Success Rate for an $SE(2)$ Robot in Contact-Rich Environments

Ruikun Luo and Dmitry Berenson

***Abstract*—Traditionally a motion planner outputs a trajectory, which is then tracked by a controller, but tracking the trajectory becomes non-trivial under actuation uncertainty and the possibility of contact. In order to design a planner that can take a controller into account, we need to estimate the success rate of the controller completing a trajectory produced by the planner. Instead of using simulation to estimate the success rate, which is too expensive, we propose a contact-point-based feature extraction method that can be used to learn the controller success rate for an $SE(2)$ robot in contact-rich environments. Our results suggest that our method can predict the success rate in cluttered environments much better than random predictions. Moreover, the computation time is much less than using simulation to estimate the success rate.**

I. Introduction

Real world robot performance is limited by the uncertainties of the sensors, robot actuators, and environment dynamics. The problem is especially acute for manipulation tasks in which the robot is moving in a contact-rich environment. Seminal work dating back to the 1980s Lozano-Perez et al. [7] proposed the idea of using contact to mitigate robot actuation uncertainty. However, it planning for motion in contact and accounting for actuation uncertainty is still a difficult problem.

We would like to create a motion planner that can plan contact motions in order to reduce the actuation uncertainties and increase the success rate of achieving assembly tasks. In order to create such motion planner, we need to estimate the success rate of the controller on each edge the planner considers. However, running simulations to estimate the success rate is computational expensive. As this estimation is required for every edge considered by the planner, we need a faster way to estimate the controller success rate. Thus in this work we focus on quickly estimating the success rate of moving a robot between two waypoints when obstacles are nearby.

We use machine learning to learn the controller success rate in contact-rich environments in order to avoid simulation in the planner loop. In this preliminary work, we focus on an $SE(2)$ robot example. We propose an approach to extract potential contact-point-based features from the environment to generate useful features that can capture both workspace and C-space information. Our results suggest that the our proposed method can predict the success rate of the controller much better than random predictions and is much faster than using simulation to estimate the success rate.

Robotics Institute, University of Michigan, Ann Arbor, Email: ruikunl@umich.edu, berenson@eecs.umich.edu. This work is supported in part by NSF grant IIS-1658635.

II. Related Work

Motion planning under actuation uncertainty has been studied for many years. Lozano-Perez et al. [7] introduced pre-image backchaining which can generate motions guaranteed to succeed under actuation uncertainty. However, computing the pre-image is computationally expensive[2, 3].

Partially-Observable Markov Decision Processes (POMDPs) have been used widely to formulate motion planning in belief-space[5, 4]. Some sampling-based planners have been proposed in [10, 1]. These planners require an estimate of the transition model between two states. Our work tries to learn the success rate of the controller which can serve as the transition model in POMDP problems.

Levine et al. [6] used guided policy search to learn a controller for robot manipulation tasks in contact-rich environments. However, our work focuses on learning the success rate of the controller moving between arbitrary waypoints instead of learning a specific controller for specific tasks. Our work is highly motivated by [9] and [8]. Melchior and Simmons [8] proposed a particle RRT where the extension to the search tree is treated as a stochastic process and use simulation to estimate the probability of moving from one state to another. Phillips-Grafflin and Berenson [9] used a kinematic simulator to estimate the transition probability between two states and used information gathered in execution to update the model. Since online kinematic simulation is time-consuming, we seek to replace it with a learned model to predict the success rate of the controller to transit from one state to the other.

III. Problem Statement

We focus on the problem of motion planning under uncertainty, specifically, we are interested in the problem of manipulating objects in contact-rich environments. We consider the object that we try to manipulate as a robot. The uncertainty we consider is the robot actuation uncertainty, where for each joint (i.e. each DOF of $SE(2)$), the control command is u and the executed command is $\hat{u} = u + \max(\omega_1, \omega_2|u|)v$. v is a random number generated from a truncated normal distribution. ω_1 is the minimum uncertainty parameter. ω_2 is the proportional uncertainty parameter. When the absolute value of the control command is larger than ω_1/ω_2, the uncertainty is proportional to the absolute value of the control command. If the absolute value of the control command is smaller than the threshold, the uncertainty magnitude is ω_1.

During the execution, the robot may become stuck. To be stuck means that given a finite time horizon, the robot

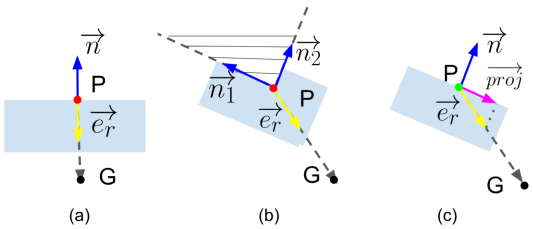

Fig. 1. Examples of C-space kinematic constraint. Green: unstuck point. Red: stuck point. Blue arrow $\overrightarrow{n}$: surface normal. Yellow arrow $\overrightarrow{e_r}$: normalized pose error. Magenta arrow $\overrightarrow{proj}$: normalized pose error projected onto the surface.

pose doesn't change significantly, however, the robot hasn't reached the goal. Both friction and kinematic constraints (e.g. an obstacle) may cause the robot to be stuck. For friction, when the robot contacts obstacles, if the force at the contact is within the friction cone the robot will be stuck. However, if the force is outside the friction cone the robot will either slide along the surface or break contact. To execute motion in contact we use the contact motion controller from [9], which is a PD velocity controller that adapts parameters to overcome the effect of static friction.

Kinematic constraints can also cause the robot to be stuck when the current robot pose, environment, and goal have a certain relationship. Fig 1 illustrates the kinematic constraints that cause the robot to be stuck in C-space. The light blue region represents a C-space obstacle. Let $\overrightarrow{n}$ represent the C-space surface normal, $\overrightarrow{n_P}$ is the surface normal at point P. $\overrightarrow{e_r}$ represents the normalized pose error vector to the goal $\overrightarrow{PG}$, where G represents the target pose. There are two different kinds of stuck point condition. Fig 1(a) shows the case that the surface normal $\overrightarrow{n}$ is well-defined at point P. In this case, if $\langle \overrightarrow{n}, \overrightarrow{e_r} \rangle = -1$, then the point P is a stuck point as shown in Fig 1(a). The second stuck condition is shown in Fig 1(b) where the point P lies on the intersection of two manifolds. In this case, we will check all the surface normal $\{\overrightarrow{n_1}, \overrightarrow{n_2}, ...\overrightarrow{n_n}\}$ at this point P on each manifold. If $-\overrightarrow{e_r}$ is in the positive linear span of $\{\overrightarrow{n_1}, \overrightarrow{n_2}, ...\overrightarrow{n_n}\}$, then the point is a stuck point. For example, in Fig 1(b), $\overrightarrow{n_1}, \overrightarrow{n_2}$ are the surface normal at P on the two manifolds. The shaded area is the positive linear span of $\overrightarrow{n_1}, \overrightarrow{n_2}$. As $-\overrightarrow{e_r}$ is lies in the positive linear span, this point P is a stuck point.

Fig 1(c) shows the case that the robot will slide on the surface. In this case, we have $-1 < \langle \overrightarrow{n}, \overrightarrow{e_r} \rangle < 0$. Thus the robot is going to slide along the magenta vector $\overrightarrow{proj}$, where $\overrightarrow{proj} = \overrightarrow{e_r} - \langle \overrightarrow{e_r}, \overrightarrow{n} \rangle \overrightarrow{n}$.

In order to plan a trajectory $\tau_1, \tau_2, ..., \tau_n$ that has high probability to be successfully executed by the given controller under actuation uncertainty, we want to estimate the probability of success for each pair of adjacent waypoints τ_i, τ_{i+1}. As τ_i, τ_{i+1} are generated from motion planning algorithms such as RRT, there is a collision free path p between the two waypoints. To succeed in moving the robot from τ_i to τ_{i+1} means that the robot will reach to the ϵ-ball around τ_{i+1} within a certain time limit and the resulting path between τ_i and τ_{i+1} should be in the region of interest around p. We would like to estimate the probability that the robot successfully reaches the ϵ-ball around τ_{i+1}.

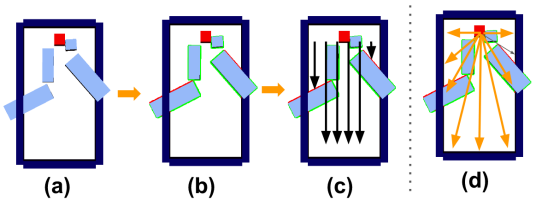

Fig. 2. Process to extract features from the environment. (a) Set a region of interest. The red box is the robot. The light blue boxes are the random obstacles in the environment. The dark blue boxes is the boundary of the region of interest. (b) Generate potential contact points on the obstacles and label them as "good" (green) or "bad" (red) points. If the robot contacts the obstacles at these "good" points, it is likely to continue toward the goal. (c) Generate feature vector from the environment using these potential contact points. The black arrows represent the parallel rays. (d) The orange arrows represent the "lidar" rays.

One can run many simulations to estimate the probability of success for moving between waypoints. However, this will require a lot of simulation time and would be prohibitive for use within a planner. Instead of running simulations, we try to use a learning method to predict the probability of success given a robot, two waypoints, and the environment.

IV. Approach

In this preliminary work, we consider the task of moving an $SE(2)$ robot from a fixed start to a fixed goal. The obstacles in the environment are randomly generated rectangles and we assume that the straight line in C-space between start and goal is collision free (as would be output by a planner). The key question to answer for learning to predict the success rate is how do we represent the important features of the environment? We propose several features to represent the environment and compare the performance of these features using Logistic Regression and Support Vector Regression. We will discuss how we design the features in this section.

As shown in Fig 2, the feature extraction process can be separated into two steps. The first step is trying to sample potential contact points around each obstacle in the workspace and label the contact points as good or bad points. The bad points are either stuck points or points from which the object is likely to be moved to a stuck point. The good points are all other surface points. The second step is to build a feature vector from the environment using these potential contact points.

A. Labeling potential contact points

As described in Section III, we can find the stuck point in the C-space and use these information to build feature vectors. However, it is hard and expensive to sample contact points on the surface of C-space obstacles. In stead of labeling the contact points in C-space, we try to label the potential contact points in the workspace.

We first interpolate a set of points on each edge of the obstacles with a fixed step size. These points can be treated as the potential contact points on the obstacles. Then we need to label these potential contact points as good or bad points as shown in Fig 2(b). The bad points are the points that are detected as stuck points described in Section III and the points will slide to the stuck points. Other points are the good points.

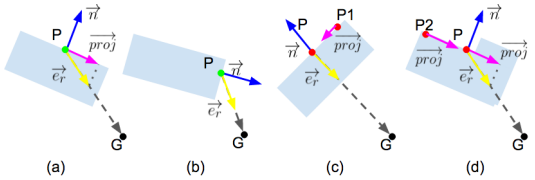

Fig. 3. Examples of "good"(green) and "bad"(red) points. Blue arrow $\overrightarrow{n}$: surface normal. Yellow arrow $\overrightarrow{e_r}$: normalized pose error. Magenta arrow $\overrightarrow{proj}$: normalized pose error projected onto the surface.

The basic assumption is if the robot has multiple contact points with the obstacles, if one of them is bad point, then the robot gets stuck. If all contact points are good points, then the current pose of the robot is likely to be a good point in C-space. However the second part of the assumption is not always be true, but we can still use this assumption in order to save computation time (we would have to check many sets of contacts otherwise).

In the workspace, labeling the potential contact points has two steps: 1) Find stuck points; 2) Find the points from which the object is likely to slide to a stuck point.

Fig 3 shows the example of labeling potential contact points in 2D space. Fig 3(a) shows an example where $\overrightarrow{e_r}$ points into an obstacle, where $\langle \overrightarrow{n}, \overrightarrow{e_r} \rangle < 0$. In this case, the point P will slide on the surface following the magenta vector $\overrightarrow{proj}$. Fig 3(b) shows the case that $\overrightarrow{e_r}$ points out of the obstacle, where $\langle \overrightarrow{n}, \overrightarrow{e_r} \rangle > 0$. Fig 3(c) and (d) are the case of stuck points. In the first step, we will find these stuck points.

Note that in Fig 3(c) and (d), point $P1$ and $P2$ are locally good points as they are similar to the example in Fig 3(a). However, if the object made contact at $P1$ or $P2$ and slid toward the goal, it would arrive at the stuck point P. Thus, $P1$ and $P2$ should also be labeled as bad points. In the second step, we will find these kind of points.

As we first discretize each edge of the rectangle obstacles, we can not directly use the stuck point condition to check whether each point is a stuck point or not because it's unlikely we will generate a point whose surface normal directly opposes the error vector. Algorithm 1 shows the process to find the bad points along one edge of the obstacle in $SE(2)$. From line 6 to line 16, we find the stuck points defined above. Lines 8-11 checks if the $\overrightarrow{proj}$ vectors of the adjacent points are pointing to each other. This is an approximate way to find the stuck point such that $\langle \overrightarrow{n}, \overrightarrow{e_r} \rangle = -1$. Lines 12-14 check if the point is a stuck point at the intersection of two surfaces. Lines 17 explore other points that are sliding to the stuck points found before. For each bad point, we also label the points from which sliding the object toward the goal will result in being stuck as "bad".

B. Build feature vector from labeled potential contact points

After labeling potential contact points as discussed above, we can get a map like Fig 2(b). In order to extract a feature vector from this map, we propose several variants of ray-shooting.

We first set up a set of rays in either workspace or C-space. We tried two different distribution of rays. The first one is shooting the rays from the origin similar to a lidar sensor as

Algorithm 1: Find Bad Points On One Edge

Input:
ps: an ordered array of points on the given edge.
p $\in$ ps: an element stored in the array.
p.pos: the position of this point in the world frame.
p.norm: the surface normal at this point.
p.isGood: indicate the point is a good point or not.
target: target pose of the robot.
δ: small step size.

Initialize:
1 **for** p *in* ps **do**
2 p.isGood $\leftarrow$ True
3 $e_r \leftarrow$ (target $-$ p.pos)/ $\|$target $-$ p.pos$\|$
4 p.proj $\leftarrow e_r - \langle$p.norm, $e_r\rangle$ p.norm
5 **end**

Find Stuck Point:
6 **for** p *in* ps **do**
7 **if** $\langle$p.norm, $e_r\rangle < 0$ **then**
8 **if** $\langle$p.proj, p.next.pos $-$ p.pos$\rangle > 0$ *and* $\langle$p.proj, p.next.proj$\rangle < 0$ **then**
9 p.isGood $\leftarrow$ False
10 p.next.isGood $\leftarrow$ False
11 **end**
12 **if** *CheckCollision(Ray(*p.pos, δp.proj/ $\|$p.proj$\|$*))* **then**
13 p.isGood $\leftarrow$ False
14 **end**
15 **end**
16 **end**

Explore Bad Point:
17 For each "bad" point, check its neighbor points. If the neighbor point's $\overrightarrow{proj}$ points to this "bad" point, label the neighbor point as "bad". Loop until all "bad" points have been checked.

the orange arrows indicate in Fig 2(d). The second approach is to use a set of parallel rays as shown in Fig 2(c).

For each ray, there is an expected length of the ray. Suppose there is no obstacles but only the dark blue boxes shown in Fig 2(a) which is the bounding box of the region of interest. The ray will hit the bounding box and stop. The length of this line segment is the expected length of the ray when the environment is clear. When given an environment with obstacles we will shoot the ray until it either hits a bad point or it hits the bounding box. The length of this line segment is called measured length of the ray. For each ray, we have two types of features: binary and continuous. If the ray is labeled as a bad ray (hitting a bad point), then the binary feature will label it as 0 and the continuous feature will label it as the ratio between the true length and the expected length. Note that if the ray hits a good point, it will keep going forward.

We also evaluate whether to use the rays in workspace or C-space. In the C-space, we move the robot along the C-space ray and when robot contacts an obstacle, if all the workspace contact points are good points, we treat this C-space contact point as a good point, otherwise it is a bad point. Thus, by combining these different choices of parallel/lidar, binary/continuous rays, C-space/workspace, we have 8 types of features: Parallel Binary C-space, Parallel Continuous C-space, Parallel Binary Workspace, Parallel Continuous Workspace, Lidar Binary C-space, Lidar Continuous C-space, Lidar Binary

Workspace, Lidar Continuous Workspace. Each combination produces a feature vector of size N, where each element is either binary or in $[0, 1]$.

C. Collecting Ground-Truth Data

In order to learn the probability of success for each environment, we need to collect a dataset of different environments and ground truth labels of probability of success. We first randomly generate a set of different environments. For each environment, the feature vector proposed in the previous section can be treated as the data point for the learning algorithm. The ground truth label is generated by simulation results. For each environment, we run simulation in Gazebo for 20 times and compute the success rate during this 20 trials. This success rate is treated as the ground truth label. We test three different regression algorithm: Logistic Regression(LR), Support Vector Machine Regression with RBF kernel and Support Vector Machine Regression with linear kernel.

V. Experiment

A. Data Collection

In this preliminary work, we set up a simple experiment for an $SE(2)$ robot as shown in Fig 2(a). The red box represents the robot which is a $0.3m \times 0.3m$ square robot. The region of interest is bounded by the dark blue boxes. The size of the region is $2m \times 3m$. The start pose of the robot is at the origin in the world frame and the target pose is $(0, 3, 0)$. The light blue boxes are randomly sampled obstacles such that there is a collision-free path between the start and target for the robot. In this experiment, we sample two rectangles on each side of the space. In order to manually introduce some difficult cases; we rotate the obstacles until one of their corners is close to the boundary of the robot path. We sampled 800 normal environments and 800 difficult cases. For each environment, we run simulations in Gazebo 20 times to control the robot moving from the start to the target. For each run, we set a 120s simulation time limit. The controller terminates if the distance between the robot and the target is smaller than $\epsilon = 0.1m$. We use the noise model described in Section III, $\hat{u} = u + \max(\omega_1, \omega_2|u|)v$, where v is generated from a truncated normal distribution. In the experiment, we use $\omega_1 = 0.02$ and $\omega_2 = 1.0$. The bound of the truncated normal distribution is $[-1, 1]$ and $\mu = 0$, $\sigma^2 = 0.5$.

We use a contact motion controller as in [9]. The noise added to the actuator of the robot is discussed in Section III. A run is successful if the resulting trajectory ends in the target region and this trajectory is inside the region of interest.

B. Results

To compare the different types of features we proposed, we tested three benchmark regression algorithms: Logistic Regression (LR), Support Vector Regression with RBF kernel (SVR with RBF), Support Vector Regression with Linear kernel (SVR with Linear). For each algorithm and each feature, we run a 10-fold cross validation to test the performance of the algorithm on this type of feature. We use the mean absolute value of errors to evaluate the performance. This measurement computes the mean absolute value of the errors between the predicted success probability and the ground truth success probability.

Table I shows the results. For the 4 Lidar versions of the feature, we use the same resolution of the rays. For the workspace feature, we have 37 rays. For the C-space feature, we have $37 \times 6 = 222$ rays, where we add 6 different orientation directions to the workspace feature. For the 4 parallel versions of the feature, we also use the same resolution of the rays. For the workspace feature, we have 31 rays. However, for the C-space feature, we have 31×6 rays, where we also add 6 different orientation directions to the workspace feature. There is no significant performance difference between the different features.

TABLE I
Mean Absolute Value of Errors and Feature Computation Time

	LR(%)	SVR with RBF (%)	SVR with Linear(%)	Time(s)
Parallel Binary C-space	12.5 ± 1.0	13.6 ± 0.9	12.1 ± 0.8	2.31
Parallel Continuous C-space	12.7 ± 0.9	13.5 ± 0.8	12.3 ± 0.7	2.31
Parallel Binary Workspace	13.3 ± 0.8	12.9 ± 0.8	13.1 ± 1.0	0.42
Parallel Continuous Workspace	13.5 ± 0.6	12.5 ± 0.9	13.2 ± 0.8	0.42
Lidar Binary C-space	13.1 ± 1.0	13.7 ± 0.9	12.1 ± 0.9	1.72
Lidar Continuous C-space	13.1 ± 0.9	12.9 ± 0.8	12.0 ± 0.9	1.72
Lidar Binary Workspace	13.8 ± 0.9	13.9 ± 1.1	13.2 ± 0.8	0.32
Lidar Continuous Workspace	13.4 ± 1.0	12.4 ± 0.9	13.5 ± 1.0	0.32
Random		37.83 ± 2.43		

During the data collection process, we run simulation 20 times to estimate the success rate for a single environment. The average computation time to estimate the success rate of a given environment using simulation method is $604.4s$. The last column in Table I shows the computation time of feature extraction a single environment. The computation time is much smaller than the simulation method.

While there is no significant difference in performance between our feature representations, the results are clearly better than randomly assigning a success rate ($37.83 \pm 2.43\%$). The small variances also suggest that our methods are not very sensitive to changes in the environment geometry, so the predictions will likely be similar for similar environments, as would be expected.

VI. Conclusion

In this paper, we proposed a contact-point-based feature extraction method to generate useful features for determining controller success probability. We tested Logistic Regression, Support Vector Machine Regression with RBF and linear kernels to learn the success rate of the controller for an $SE(2)$ robot in a contact-rich environment. The preliminary experiment shows that our features can be used to learn the success rate of the controller better than random prediction and can be done much faster than simulation. In future work, we will try to extend the method to $SE(3)$ robots.

References

[1] Ron Alterovitz, Thierry Siméon, and Kenneth Y Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *RSS*, 2007.

[2] J. Canny. On computability of fine motion plans. In *ICRA*, 1989.

[3] Michael Erdmann. Using backprojections for fine motion planning with uncertainty. *IJRR*, 1986.
[4] Michael C Koval, Nancy S Pollard, and Siddhartha S Srinivasa. Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty. *IJRR*, 2016.
[5] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *RSS*, 2008.
[6] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. In *ICRA*, 2015.
[7] Tomas Lozano-Perez, Matthew T Mason, and Russell H Taylor. Automatic synthesis of fine-motion strategies for robots. *IJRR*, 1984.
[8] Nik A Melchior and Reid Simmons. Particle rrt for path planning with uncertainty. In *ICRA*, 2007.
[9] Calder Phillips-Grafflin and Dmitry Berenson. Planning and resilient execution of policies for manipulation in contact with actuation uncertainty. *WAFR*, 2016.
[10] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *IJRR*, 2009.