2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Trajectory Prediction in Cluttered Voxel Environments

Nikolay Jetchev and Marc Toussaint

Abstract— **Trajectory planning and optimization is a fundamental problem in articulated robotics. It is often viewed as a two phase problem of initial feasible path planning around obstacles and subsequent optimization of a trajectory satisfying dynamical constraints. There are many methods that can generate good movements when given enough time, but planning for high-dimensional robot configuration spaces in realistic environments with many objects in real time remains challenging. This work presents a novel way for faster movement planning in such environments by predicting good path initializations. We build on our previous work on trajectory prediction by adapting it to environments modeled with voxel grids and defining a frame invariant prototype trajectory space. The constructed representations can generalize to a wide range of situations, allowing to predict good movement trajectories and speed up convergence of robot motion planning. An empirical comparison of the effect on planning movements with a combination of different trajectory initializations and local planners is presented and tested on a Schunk arm manipulation platform with laser sensors in simulation and hardware.**

I. Introduction

Movement generation, one of the most basic robotic tasks, is often viewed as an optimization problem that aims to minimize a cost function. There are many different methods for local trajectory optimization which uses the cost gradient information for minimization. Popular approaches use spline-based representation and gradient descent in [1], covariant gradient descent in [2], Differential Dynamic Programming (DDP) [3] [4], also known as iterated Linear Quadratic Gaussian [5], and Bayesian inference [6].

Another approach for finding good movement trajectories is sampling to find obstacle free paths in the configuration and work space of the robot, i.e. finding an appropriate initialization of the movement plan. Popular methods for planning feasible paths without collisions are Rapidly-exploring Random Trees (RRT) [7] and probabilistic road maps [8], where random sampling is used to build networks of feasible configuration nodes. These methods are powerful and can find difficult solutions for motion puzzles, but also have the disadvantage to be too slow for some manipulation problems. Building an RRT takes some time, and a path to the target in such a network often requires additional optimization to derive an optimal robot trajectory.

In our previous work [9] we introduced trajectory prediction as a way to speed up movement generation with initialization using knowledge of successfully optimized movement trajectories. Gathering such data allows to create a mapping between the situation and an initial path likely to quickly lead the local optimizer to converge to a good movement.

N. Jetchev and M. Toussaint are with TU Berlin, Machine Learning and Robotics Group, Franklinstrasse 28/29, 10587 Berlin, Germany.
{ jetchev,mtoussai}@cs.tu-berlin.com

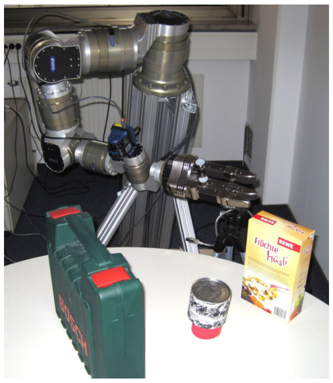


Fig. 1. The Schunk arm with the Hokuyo laser sensor mounted, in a scenario with a table and 3 other block obstacles. The goal is to reach the target, a point behind the cylinder in the middle.

There are also other methods for reusing feasible path databases, e.g. with RRT path repairing a subset of no longer feasible edges [10], reusing paths between situations [11], and biasing RRT search to promising regions [12]. These methods differ from our approach to trajectory prediction in that they do not use machine learning for a mapping from situation to movement, and are too slow to be appropriate for real time movements.

Another interesting way to exploit a database of previous motions is to learn a "capability map", i.e., a representation of a robot's workspace that can be reached easily [13].

Laser range finders are a cheap and reliable way to add sensor capabilities to robots. They are often used for navigation and of mobile manipulation [14]. Voxel representations and the use of laser range information for manipulation are examined in [15], where the sensor model is coupled with a sampled road map describing space connectivity. Our work has a similar sensor model, but we use this information as input to various motion planning algorithms.

The main contribution of this paper is the new trajectory prediction algorithm with novel representations for situations and invariant trajectories in cluttered voxel scenes, as well as the analysis of the effect of initialization on different trajectory optimizers. In the next sections we will overview our previous work on trajectory prediction, then proceed to describe our new sensor model in section III and combine it with a modification of trajectory prediction from [9] in section IV. Finally we will present empirical results from simulations of movement generation with different initializations and local optimizers.

II. Background: Movement Planning

A. Optimizing Trajectories and Movement Generation

Let us describe the robot configuration as $q_t \in \Re^N$, the joint posture vector. We define $\boldsymbol{q} = (q_0, .., q_T)$ as a movement trajectory with time horizon T. In a given situation x, i.e., for a given initial posture q_0 and the positions of obstacles and targets in this problem instance (we will formally define descriptors for x in section IV-B), the motion generation problem is to compute a trajectory which fulfills different constraints, e.g. an energy efficient movement not colliding with obstacles.

We formulate this task as an optimization problem by defining a cost function $F(\boldsymbol{q}; x)$ that characterizes the quality of the joint trajectory in the given situation and task constraints. A local optimizer, like DDP and the other algorithms mentioned in the introduction, will try to find the best movement for a given situation:

$$\boldsymbol{q}^* = \underset{\boldsymbol{q}}{\operatorname{argmin}}\, F(\boldsymbol{q}; x) \tag{1}$$

To arrive at the optimal trajectory $\boldsymbol{q}^*$ (or one with a very low F value), most local optimizers start from an initial trajectory $\tilde{\boldsymbol{q}}$ and then improve it (e.g. by using the cost function gradient). We call L the local optimizer operator and write $\boldsymbol{q}^* = L(\tilde{\boldsymbol{q}})$.

Optimizing F is a challenging high-dimensional nonlinear problem. Many of the movement optimization methods are sensitive to initial conditions and their performance depends crucially on it. For example, initial paths going straight through multiple obstacles are quite difficult to improve on, since the collision gradients provide confusing information and try to jump out of collision in different conflicting directions, as mentioned by [2].

B. Trajectory Prediction

Humans and animals execute complex motions constantly, without stopping for a long time to plan. A person can look at a table cluttered with bottles and glasses and immediately take what he needs, without building mentally a network of all accessible paths like a sampling-based planner. This suggests some kind of goal-oriented "reactive trajectory policy" and such an approach can be useful in robotics to design algorithms to generate movement instantaneously, or at least fast enough to have fluid interaction (a few seconds at most).

The lifetime of an autonomous robot consists of sensing its environment, calculating proper movements and executing them. Environments encountered by robots are often highly structured, and good movements can be reused and transferred between situations. We aim to improve the convergence time of local optimizers by learning a mapping from a situation to a proper initialization of the optimizer in task space:

$$x \mapsto \phi_x^{-1} \tilde{y} = \tilde{\boldsymbol{q}} \tag{2}$$

Here ϕ_x^{-1} is the inverse kinematics (IK) transfer in a situation x, and $\tilde{y} \in \Re^{T \times 3}$ is the predicted task space trajectory of the 3D coordinates of the robot endeffector. As defined in [9], such IK transfer takes a path in the endeffector task space from a previous situation and transforms it into a joint trajectory in the new situation x, roughly following the previous path and correcting for collisions and other constraints. For the mapping ϕ^{-1} we use the IK method of [16] to include multiple task constraints with precisions.

Our goal is to use a descriptor of the situation x to quickly find a good initial solution $\tilde{\boldsymbol{q}}$. We restrict ourselves to choose movements from a predefined prototype set C which are "appropriate" for the given situation and likely to have low cost. We need to predict this cost through a function $f(x, y) \approx F(L(\phi_x^{-1} y); x)$. Trajectory prediction through cost prediction can be defined as the following mapping:

$$x \mapsto y^* = \underset{y \in C}{\operatorname{argmin}}\, f(x, y) \approx \underset{y \in C}{\operatorname{argmin}}\, F(L(\phi_x^{-1} y); x) \tag{3}$$

C is a set of task space movements (actions) over a longer time horizon, the whole T.

By constructing an appropriate representation of x, we can approximate the cost of a movement in a situation, without actually making that movement in the simulator (via IK) and calculating the joint posture, body positions and collisions of the robot. The complexity of IK and local planning scale with the number of joints N and time horizon T. Using a learned approximation f is almost simultaneous regardless of the system parameters. and can be trained to make good movement choices out of a proper set of alternatives C. In the next sections we will explain our sensor model and define the representations of situations and prototypes used for trajectory prediction in cluttered situations.

III. Sensor Model

A. Laser Point Cloud

We need a sensor model to provide an accurate collision gradient for obstacle avoidance, a necessary part for any local optimizer of the cost function F. A 2D laser scanner is often used in robotics to provide data in the form of point cloud measurements. This sensor type makes a sweep in a 2D plane and reports where the rays hit an object. In order to get 3D information for the world and allow range-vision based motion planning, the sensor needs to be moved to cover the 3D space. We use a simple heuristic to gather information for the scene. Before movement planning, the arm-mounted laser is rotated by moving the robot joint of the arm segment carrying it. A full rotation in 20 steps between the joint limits covers practically the whole workspace with rays. This is a good approximation for the obstacles in the cluttered scenes we examine, and it takes less than a second to make such 20 scanner planner sweeps.

To test our methods we need large amounts of data generated offline in simulation, so we implemented an accurate geometric simulator of the laser rays, which is a good approximation of the real laser range finder, see Figure 2.

B. Voxel Occupancy Grid World Model

Given a set of laser cloud points $P = \{p_i\}$, we construct a 3D grid system $V = \{v_i\}$ of voxels. Each voxel is identified

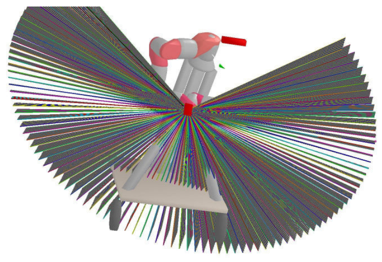
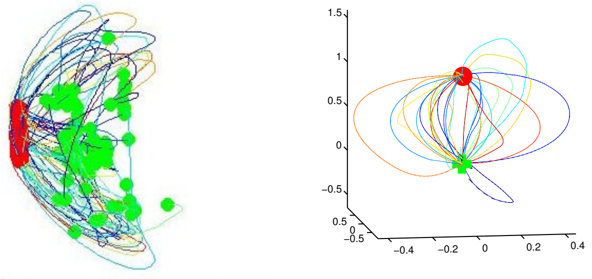


Fig. 2. An illustration of the arm-mounted laser range sensor. A single laser sweep covers 270 degrees in the 2D plane of the sensor module.



(a) World frame trajectories (b) Invariant frame Γ trajectories

Fig. 3. World frame trajectories y_i and invariant frame projections z_i of hand endeffector paths in different situations. The trajectories go from the green start locations to the red target destinations. The graphic shows visually why the invariant frame is so useful for generalization.

with its coordinates and its occupancy probability $p(v_i) \in [0, 1]$. The procedure for calculating $p(v)$ is straightforward:

1) Loop through all available measurements p_i
2) Loop through all voxel v_j
3) If $p_i \subset v_j$ set $p(v_j) = 1 - 0.9 * (1 - p(v_j))$

Intuitively, for every measurement point within some voxel bounds the occupied space probability of the voxel increases. Other papers [15] [17] use sensor models with state distributions for free, unknown and occupied voxel space, but for our tests just the occupied space probability suffices for collision avoidance. We add each occupied voxel where $p(v_i) > 0$ as a solid body to our simulator, and these voxels can be used to estimate potential collisions and plan movements. We modelled each voxel as a cube of size 5cm, allowing accurate collision measurements and avoiding too many voxels that can slow down calculations.

IV. Data Representations for Trajectory Prediction

In this section we will present the modifications of trajectory prediction required to generalize to voxel models.

A. *Invariant Prototype Movement Set*

Trajectory prediction as defined in section II-B requires a set C of task space movements that represent feasible paths to the target in any possible world configuration. Similar to our previous work [9], we generate data of situations and optimal endeffector trajectories (obtained via offline DDP optimization until convergence) and then cluster them with the k-Means algorithm in c clusters. The cluster centroids represent averaged movements and are useful movement prototypes, retaining the variety and characteristics of good movements in previous situations.

In the current work we improve the movement prototype database generalization ability by defining the averaged prototypes in an invariant space, a novel geometric technique to deal with variance in trajectory databases. We define a 3D frame where the initial hand position is always $(0, 0, 0)$, the target and final position is $(0, 0, 1)$. To define this uniquely, we use a frame rotation such that the hand-target line is the $(0, 0, 1)$ axis in the new frame. Finally, we rotate the y axis of this frame to be perpendicular to the original $(0, 0, 1)$ world axis (thus uniquely defined), and we scale so that the hand-target distance is unit distance. We write Γ_x to define the projection in this invariant frame, since it is uniquely defined for each world situation x by the hand and target positions.

The procedure to obtain C is summarized as follows:

1) Get a set of optimized endeffector trajectories $\{x_i, y_i\}_{i=1}^{1000}$ (in world coordinates)
2) Get projections $\Gamma_{x_i} y_i = z_i$ (in invariant frame)
3) Cluster z_i in c clusters
4) The set of prototype movements consists of the cluster centroids $C := \{\bar{z}_i\}_{i=1}^{c}$
5) Given a novel situation x, $\bar{y}_i = \Gamma_x^{-1} \bar{z}_i$ for $\bar{z}_i \in C$ will be a movement from the endeffector to the target (in world coordinates)

We constructed a set of c possible paths from the hand to the target that can adapt to any situation, regardless of the relative positions and orientations of the current situation, see Figure 3. In every situation x we derive from the invariant prototypes $\bar{z}_i \in C$ the world frame prototypes $\bar{y}_i \in \Gamma_x^{-1} C$.

B. *Descriptor for Trajectory Prediction*

We want to define the descriptor of a given world situation x and prototype movement $\bar{y}$ using only information that is available before IK transfer and local optimization, i.e. variables that can be read instantly without calling time expensive routines like collision checks and internal robot control simulation. This information consists of the point cloud data from the laser, the current robot position, the target destination and the task space trajectory $\bar{y}$ itself. We combine this information in the following situation and planned movement descriptor:

$$\xi^{x\bar{y}} = (v^{\bar{y}}, d^{\bar{y}}) \tag{4}$$

The first component $v^{\bar{y}}$ represents information from the laser sensor point cloud describing the scene according to section III-B. We define a set of voxel grids, 9 voxels across each dimension, where each voxel is a cube with side 5cm long. Each such grid is centered on a point from the task space path $\bar{y}$. Since each grid state is of dimension $9^3 = 729$, it is advantageous to reduce its dimension via a simple PCA transform, and retain the 15 dimensions capturing almost the whole data variance. The interpretation of these lower

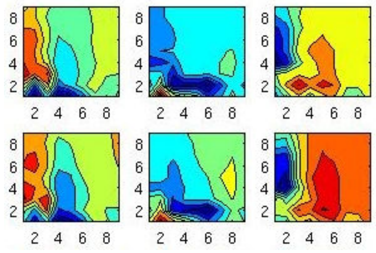


Fig. 4. Several volume contour slices of 9x9 voxels, corresponding to different PCA components. Red areas are more likely to be occupied, and blue areas are probably free.

dimensional components is of typical volume slices in the blocks world we used, see Figure 4. One can find analogies with work on classification of terrain into passable and nonpassable regions, see [18].

The second feature component $d^{\bar{y}}$ consists of pairwise distance measurements between the robot immobile base, the target location, and the samples of the task space movement $\bar{y} \in C$. For each point of $\bar{y}$, we calculated pairwise 3D offset vector and distance to the robot body immobile base, the reach target, and the previous task space point sample, a total of 12 measurements per point sample.

To ease computation, we use a resampled version to follow the movement $\bar{y}$. We chose a fixed coarser resolution for the descriptor with a small number of point samples. Concretely, in our experiments we had 11 samples, each with $15+12 = 27$ features, concatenated in $\xi^{x\bar{y}} \in \Re^{297}$. Such descriptors with voxels are flexible and can adapt to any terrains and number of objects, an extension on our previous work [9] which only had pairwise object distances.

C. Regression for Cost Prediction f

The final component we need for trajectory prediction, as defined in Equation 3, is a regressor $f(\xi^{x\bar{y}})$ for $F(L(\phi_x^{-1}\bar{y}); x)$. It should be trained to predict the potential cost of initializing a local optimizer with $\bar{y}$ transferred in situation x. Given a set of movement prototypes C and situations x, we can choose the action that minimizes the cost $f(\xi^{x\bar{y}})$ and execute it. We gather a training dataset $D := \{\xi^{x_i\bar{y}_j}, L(F(\bar{y}_j; x_i))\}$, i.e. pairs of descriptors and costs after optimizing locally with DDP for different situations x_i and each prototype $\bar{y}_j \in C$.

We will use D to learn the regression f. We use Support Vector Regression (from the SHOGUN package [19]) with linear kernel and train a separate cost regression model for each of the c prototypes.

The policy we use when presented with a new situation x to arrive at an optimized joint trajectory $\boldsymbol{q}^*$ can be summarized like this:

1) $\tilde{y} = \underset{\bar{y}\in\Gamma_x^{-1}C}{\operatorname{argmin}}\ f(\xi^{x\bar{z}})$ (select best prototype)
2) $\tilde{\boldsymbol{q}} = \phi_x^{-1}\tilde{y}$ (IK transfer to joint initialization)
3) $\boldsymbol{q}^* = L(\tilde{\boldsymbol{q}}) = \underset{\boldsymbol{q}}{\operatorname{argmin}}\ F(\boldsymbol{q}; x)$ (optimize)

V. Experiments

A. The Task: Reaching in a Cluttered Table

The scenario we examine is reaching in a cluttered environment, namely a blocks world with a different number of obstacles placed on a large table. We generate different scenarios by randomly changing the reach target locations across the table, the position and sizes of the obstacles. This task has a cost function F that is a sum of terms for collision avoidance, joint limit avoidance, smooth joint transition, and reaching a specified target in the last step of the movement.

We use a Schunk LWA3 arm, a SDH hand and an arm-mounted Hokuyo URG-04LX laser sensor[1]. The robot, shown in Figure 1, has 14 joints. The first 7 joints corresponding to the arm posture are the most critical for this application, the other 7 hand joints have only minor effect on the motions and collisions. We also set in our experiments $T = 200$, a time horizon of 200 slices of 0.01 seconds each. This means that $\boldsymbol{q} \in \Re^{200\times14}$, i.e. we have a challenging trajectory optimization problem of dimension 2800. The simulator and robot control were compiled in C++ and run on a PC running Ubuntu, with 2GB RAM and 2.4 GHz CPU.

B. Method Comparison Setup

We will test three different initialization methods. Linear path initialization (*LI*) is the default option, where the start and goal endeffector positions are connected with a straight line path IK. The more interesting methods are an RRT path planner (*RRT*), and trajectory prediction (*TP*). Both trajectory prediction and straight line initialization require an IK operator from the endeffector path to joint space, which costs around 0.6s. *RRT* requires to construct a tree of sample positions locally accessible from each other, and for a tree of 500 nodes this costs already more than 8 seconds.

The prototype set C used by *TP* is obtained by clustering a set of 1000 movements optimized with DDP in random situations. We chose $c = 20$ clusters as a good tradeoff for a compact set that can react adequately in most situations. If c is too large, gathering train data D would be more expensive. To learn the mapping from situation and prototype to cost f we used a dataset D with $d = 1000$ scenarios, for each of which all 20 prototypes were evaluated. All training data is generated in worlds with 5 random blocks in the world. The evaluation of $f(\xi^{x\bar{y}})$ for all 20 prototypes $\bar{y} \in \Gamma_x^{-1}C$ takes less than 0.01s, which is a great advantage of our chosen compact descriptors and simple linear predictor, and allows to deal with potentially larger and more diverse sets C.

We also test three popular local optimizers - DDP [4], AICO [6] and direct gradient descent in joint space with the RPROP general optimization algorithm [20]. This makes for a total of 9 initialization-optimizer pairs, which are shown in the result tables with a name indicating the initialization and the optimization method.

[1]A video of our robot reaching its target, as well as the datasets used for trajectory prediction, are available at `http://user.cs.tu-berlin.de/~jetchev/TrajectoryVoxel.html`

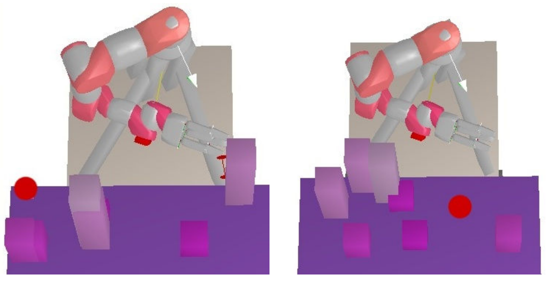

(a) 5 obstacles (b) 8 obstacles

Fig. 5. Examples of different simulated scenes: random positions and sizes of rectangular obstacles. The goal is to reach the red point target.

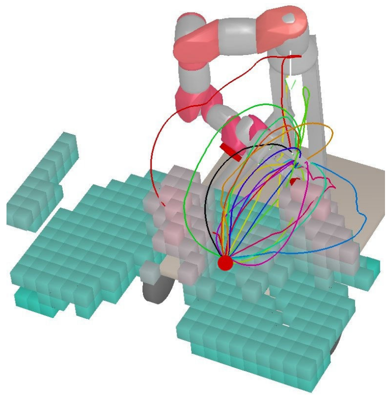


Fig. 6. The cluster set C with 20 possible endeffector trajectories leading to the red target.

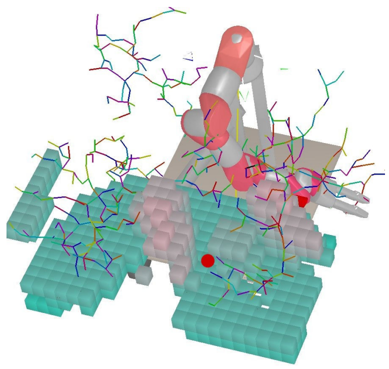


Fig. 7. The jagged trees built typically by RRT trees.

The different scenarios generated by randomizing objects are of greatly varying difficulty; some of them are trivially easy and others are impossible to solve, but the statistics of the average performance still allows to compare the different algorithms. Note that all the results are for test sets of situations not encountered during the train phase, but generated by the same random distribution.

We measure total computation time (initialization and local optimization) on two simulated test sets of 500 situations each, one with 5 and the other with 8 objects. We measure the convergence of the costs after initialization and optimization for 30 iterations. Each optimizer iteration costs 0.5s, with the most expensive operation being collision detection.

A cost margin $\epsilon = 0.5$ implies a feasible solution without collisions, whereas a smaller margin corresponds to solutions which are near the optimum. $\#$ stands for the proportion of the situations where the particular method did not reach level ϵ, i.e. convergence failure. μ is the average time to reach level ϵ, calculated on the situations where *all* of the methods reached the corresponding level. With $\pm$ the Standard Mean Error of our estimate of μ is shown. Small values of μ and $\#$ indicate better performance.Such a setup for μ allows to compare convergence speed for all 9 methods on the same set of situations, but has a bias for situations which can be solved for all methods.

We present results for convergence on the voxel world models. Despite the sensor noise and imprecision from the voxel grid resolution, the costs of trajectories calculated in the voxel model highly correlate with costs in the real world situation.

C. Results

Tables I and II show that trajectory prediction improves over the default initialization *LI* for all optimization methods, both in fast convergence times μ and low failure rate $\#$. The combination *TP*-DDP has very good $\#$ and the fastest μ, requiring less than 2 iterations of DDP usually, which makes it a great choice for repetitive motion in real time.

In a situation like Figure 6, the prototype set C offers a range of paths to the target, and the learned cost approximation f will prefer paths within the reach of the robot avoiding the obstacles in front of it. A human can look at this image and immediately choose an action, and our SVR regression learns such a model. Optimization techniques like RPROP and DDP depend greatly on good initialization. If they start distant from a good minimum, they are more likely to fall in a bad local optimum, and stay there. AICO uses probabilistic inference, which has a different way of incorporating prior information, but a good initialization has positive influence on this algorithm as well, with better $\#$ values, though with more time necessary for convergence.

RRT is good at finding narrow passages and often manages to converge in more situations (lowest $\#$). However, its random sampling and jagged initial paths, see Figure 7, makes it difficult to optimize to a smooth trajectory, and it is always much slower in μ than *TP* or even *LI* in local optimizer convergence time. When one also considers the 8 seconds required to build the tree of accessible configurations for RRT, the other methods like *TP* seem much better suited for real time interaction and manipulation.

Regardless of the initialization, DDP had the best convergence times, closely followed by AICO, and RPROP was worst. This is to be expected, since gradient descent on such a high dimensional problem is at great disadvantage, not using the structure of the problem, unlike techniques which were specifically designed with trajectory and control optimization in mind, like AICO.

The effect of adding 3 more obstacles on Table II is to make all methods slower and less likely to find an optimum, since situations with blocked paths happen more

TABLE I

RESULTS FOR 5 OBSTACLE SCENARIO. μ IS AVERAGE TIME IN SECONDS, # IS PROPORTION OF FAILURES. INITIALIZATION TIME IN SECONDS IN BRACKETS.

Method		$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$	$\epsilon = 0.05$
TP-DDP	μ	**0.70± 0.02**	**0.86 ± 0.04**	**1.06 ± 0.09**	**1.52 ± 0.42**
(0.6 sec)	#	0.020	0.034	0.064	0.112
LI-DDP	μ	0.90 ± 0.02	1.35 ± 0.05	1.77 ± 0.09	2.65 ± 0.35
(0.6 sec)	#	0.036	0.074	0.122	0.252
RR-DDP	μ	3.12 ± 0.06	3.43 ± 0.07	3.62 ± 0.09	3.97 ± 0.20
(8 sec)	#	**0.010**	**0.020**	0.040	0.076
TP-AICO	μ	1.03 ± 0.07	1.35 ± 0.10	1.70 ± 0.19	2.62 ± 0.59
(0.6 sec)	#	0.022	**0.032**	**0.038**	**0.072**
LI-AICO	μ	1.01 ± 0.06	1.12 ± 0.06	1.33 ± 0.11	2.23 ± 0.31
(0.6 sec)	#	0.024	0.078	0.094	0.136
RR-AICO	μ	5.07 ± 0.20	5.34 ± 0.23	6.05 ± 0.30	6.72 ± 0.67
(8 sec)	#	**0.010**	0.028	0.056	0.112
TP-RPROP	μ	1.18 ± 0.10	2.30 ± 0.18	2.85 ± 0.30	2.87 ± 0.55
(0.6 sec)	#	0.070	0.158	0.316	0.582
LI-RPROP	μ	2.31 ± 0.11	4.14 ± 0.15	4.73 ± 0.20	6.17 ± 0.36
(0.6 sec)	#	0.126	0.224	0.368	0.536
RR-RPROP	μ	12.46 ± 0.21	13.62 ± 0.28	14.88 ± 0.40	16.12 ± 0.65
(8 sec)	#	0.360	0.570	0.748	0.904

TABLE II

RESULTS FOR 8 OBSTACLE SCENARIO. μ IS AVERAGE TIME IN SECONDS, # IS PROPORTION OF FAILURES. INITIALIZATION TIME IN SECONDS IN BRACKETS.

Method		$\epsilon = 0.5$	$\epsilon = 0.2$	$\epsilon = 0.1$	$\epsilon = 0.05$
TP-DDP	μ	**0.81 ± 0.02**	**1.07 ± 0.05**	**1.54 ± 0.14**	**2.53 ± 0.73**
(0.6 sec)	#	0.034	0.074	0.094	0.164
LI-DDP	μ	1.10 ± 0.04	1.61 ± 0.06	2.32 ± 0.15	3.55 ± 0.50
(0.6 sec)	#	0.048	0.102	0.168	0.300
RR-DDP	μ	3.31 ± 0.06	3.69 ± 0.08	4.18 ± 0.12	5.03 ± 0.36
(8 sec)	#	**0.026**	**0.042**	**0.066**	**0.114**
TP-AICO	μ	1.19 ± 0.07	1.53 ± 0.11	2.19 ± 0.27	2.92 ± 0.91
(0.6 sec)	#	**0.032**	**0.056**	**0.084**	**0.114**
LI-AICO	μ	1.21 ± 0.08	1.35 ± 0.08	**1.51 ± 0.14**	2.71 ± 0.43
(0.6 sec)	#	0.056	0.102	0.146	0.194
RR-AICO	μ	4.94 ± 0.20	5.66 ± 0.25	6.83 ± 0.39	7.82 ± 0.96
(8 sec)	#	0.040	0.064	0.094	0.160
TP-RPROP	μ	1.58 ± 0.12	2.77 ± 0.24	3.76 ± 0.39	3.59 ± 0.77
(0.6 sec)	#	0.122	0.248	0.420	0.678
LI-RPROP	μ	2.98 ± 0.16	4.84 ± 0.21	5.87 ± 0.43	6.70 ± 0.62
(0.6 sec)	#	0.182	0.316	0.472	0.644
RR-RPROP	μ	11.40 ± 0.21	12.67 ± 0.26	14.01 ± 0.37	15.57 ± 0.76
(8 sec)	#	0.406	0.602	0.758	0.926

often. However, trajectory prediction remains the fastest initialization even with this more cluttered setup, a transfer of useful behavior from the training database setup with 5 blocks. This shows that the descriptors $\xi^{x\bar{y}}$ and the predictor f can transfer knowledge to a more diverse set of scenarios without modification. On the other side, when considering the potential effect of adding even more objects (e.g. more than 20), *RRT* has the best chance to solve such puzzles. The design of the scenario has big effect on performance.

In addition to simulation, we also did hardware tests as in Figure 1, and had robust performance in real scenes with different obstacles on tables.

VI. CONCLUSIONS AND FUTURE WORK

We tested extensively in simulation 9 combinations of diverse path initialization and local optimization methods to plan reaching trajectories in cluttered scenes. The gain in convergence speed when using trajectory prediction, our method for path initialization, makes it a good choice to improve performance in conjunction with various optimization algorithms, at no additional computation cost.

The trajectory prediction framework is simple to implement, but has potential for future research. The movement policy we used can be modified: instead of choosing only once an action from C - a start-to-goal movement - and executing it for the time horizon T, we can follow just the first few time steps of the chosen prototype and then predict a next action with a newly calculated situation descriptor. The prototypes invariant space can easily handle different time resolutions and starting robot configurations. The predicted actions can be used in a parallel CPU framework to explore different solution sequences simultaneously.

VII. ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG), Emmy Noether fellowship TO 409/1-3.

REFERENCES

[1] J. Zhang and A. Knoll, "An enhanced optimization approach for generating smooth robot trajectories in the presence of obstacles," in *Proc. of the European Chinese Automation Conf.*, 1995, pp. 263–268.

[2] J. A. B. Nathan Ratliff, Matthew Zucker and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, May 2009.

[3] P. Dyer and S. R. McReynolds, *The Computation and Theory of Optimal Control*. Elsevier, 1970.

[4] C. G. Atkeson, "Using local trajectory optimizers to speed up global optimization in dynamic programming," in *NIPS*, 1993, pp. 663–670.

[5] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proc. of the American Control Conf.*, vol. 1, 2005, pp. 300–306.

[6] M. Toussaint, "Robot trajectory optimization using approximate inference," in *26th Int. Conf. on Machine Learning (ICML)*, 2009, pp. 1049–1056.

[7] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2006, pp. 1874–1879.

[8] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan, "Randomized query processing in robot path planning," in *Twenty-seventh annual ACM Symposium on Theory of Computing (STOC)*, 1995, pp. 353–362.

[9] N. Jetchev and M. Toussaint, "Trajectory prediction: Learning to map situations to robot trajectories," in *26th Int. Conf. on Machine Learning (ICML)*, 2009, pp. 449–456.

[10] J.-M. Lien and Y. Lu, "Planning motion in environments with similar obstacles," in *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009.

[11] M. Branicky, R. Knepper, and J. Kuffner, "Path and trajectory diversity: Theory and algorithms," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2008, pp. 1359–1364.

[12] S. Martin, S. Wright, and J. Sheppard, "Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in dynamic environments," in *IEEE Int. Conf. on Automation Science and Engineering (CASE).*, 2007, pp. 1131–1136.

[13] F. Zacharias, C. Borst, and G. Hirzinger, "Capturing robot workspace structure: representing robot capabilities," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2007, pp. 3229–3236.

[14] R. Kümmerle, P. Pfaff, R. Triebel, and W. Burgard, "Active monte carlo localization in outdoor terrains using multi-level surface maps," in *AMS*, 2007, pp. 29–35.

[15] A. Nakhaei and F. Lamiraux, "Motion planning for humanoid robots in environments modeled by vision," in *8th IEEE-RAS Int. Conf. on Humanoid Robots*, 2008, pp. 197–204.

[16] M. Toussaint, "Bayesian inference for motion control and planning," Technische Universitaet Berlin, Tech. Rep. 22, 2007.

[17] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[18] D. F. Wolf, G. S. Sukhatme, D. Fox, and W. Burgard, "Autonomous terrain mapping and classification using hidden markov models," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2005, pp. 2026–2031.

[19] C. S.Sonnenburg, G.Raetsch and B.Schoelkopf, "Large scale multiple kernel learning," *Journal of Machine Learning Research*, no. 7, pp. 1531–1565, 2006.

[20] W. W. Christian Igel, Marc Toussaint, "Rprop using the natural gradient," *Trends and Applications in Constructive Approximation. International Series of Numerical Mathematics*, vol. 151, pp. 259–272, 2005.