The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

On the Performance of Random Linear Projections for Sampling-Based Motion Planning

Ioan Alexandru Şucan and Lydia E. Kavraki

Abstract— **Sampling-based motion planners are often used to solve very high-dimensional planning problems. Many recent algorithms use projections of the state space to estimate properties such as coverage, as it is impractical to compute and store this information in the original space. Such estimates help motion planners determine the regions of space that merit further exploration. In general, the employed projections are user-defined, and to the authors' knowledge, automatically computing them has not yet been investigated. In this work, the feasibility of offline-computed random linear projections is evaluated within the context of a state-of-the art sampling-based motion planning algorithm. For systems with moderate dimension, random linear projections seem to outperform human intuition. For more complex systems it is likely that non-linear projections would be better suited.**

I. INTRODUCTION

Sampling-based motion planners [1]–[5] are a class of algorithms capable of quickly solving the motion planning problem – finding a continuous path between a starting state and a goal state for a robotic system under certain constraints – even for high-dimensional systems [6]–[10]. Sampling-based algorithms can be only probabilistically complete [11], [12], which means if a solution exists, it will eventually be found, but if no solution exists, termination is not guaranteed. These algorithms implement a search in the continuous state space $\mathcal{X}$ of the robotic system. $\mathcal{X}$ is assumed to be a differentiable manifold consisting of all states $\mathbf{x}$ the robotic system could be in. In a state space of dimension m, a single element $\mathbf{x} \in \mathcal{X}$, $\mathbf{x} = (x_1, ..., x_m)$ completely describes the state of a robotic system. This means that each component x_i, $i \in \{1, ..., m\}$ defines a parameter of the system; such parameters can be joint angles, positions in space, velocities, accelerations, etc. It is often the case that systems of practical interest have high-dimensional state spaces. For instance, a typical manipulator arm has 6 or 7 degrees of freedom [13]. Considering only the position and velocity at each joint, a 12- or 14-dimensional state space is generated. In multi-robot coordination [14] or modular robots [15], as more robots/modules are added, the dimensionality grows even higher.

When considering dynamic constraints of the robotic system, such as bounds in accelerations, it is typical that sampling-based planers perform the search by growing a tree of motions [7]–[10], [16]–[24] in $\mathcal{X}$, towards the goal region. The search progresses iteratively by building a tree with the root at the initial state of the robotic system, as shown in Figure 1. At each iteration, a state in the tree is selected and expanded from, towards some direction. Under certain conditions, this process can cover the state space $\mathcal{X}$ and find a state in the goal region, if one exists. An important issue is guiding the growth of this tree. This in fact has been one of the important subjects of recent research in sampling-based motion planning.

This work supported in part by NSF IIS 0713623 and Rice University Funds. Equipment used for the development of this work includes equipment purchased by CNS-0421109 ADA in partnership with Rice University, AMD and Cray.

I. A. Şucan and L. E. Kavraki are with the Department of Computer Science, Rice University, USA {isucan, kavraki}@rice.edu

A. Guiding Tree Expansion

When searching high-dimensional state spaces using a tree of motions, deciding which part of the tree merits further exploration is not trivial. Various approaches have been tried to address this issue. To name a few, Rapidly-exploring Random Trees (RRTs) expand towards randomly produced states [7], [17], Expansive Space Trees (ESTs) attempt to detect less explored regions and expand starting from them [8], [16], Utility Trees attempt to evaluate the utility of expanding from a specific state [21].

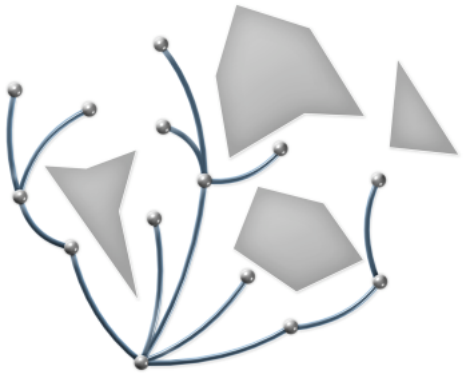


Fig. 1. Example tree of motions grown in the state space $\mathcal{X}$.

This work concentrates on recent algorithms [6], [9], [10], [22] which employ a projection of the state space $\mathcal{X}$ to make the decision of where to continue the tree expansion from. We denote this projection space by $\mathcal{E}(\mathcal{X})$. Using $\mathcal{E}(\mathcal{X})$ instead of $\mathcal{X}$ avoids the issue of exponential blow-up when evaluating space properties such as coverage. This is an approximation based on the assumption that if the tree of motions covers $\mathcal{E}(\mathcal{X})$ well, it also covers $\mathcal{X}$ well. It has not been proven when or whether this assumption holds for projections that have been used in the literature. Previous work has empirically shown that the tree exploration can be

guided towards the goal region for some user-defined projections [6], [10], [22]. However, automatically computing these projections remains an open question.

The advantage of having automatically computed projections is the potential of finding projections that are better than what the user provides – in the sense that planners will run faster, the possibility of using different projections for different environments and simplifying the input to motion planning algorithms. Moreover, as the complexity of robotic systems increases, human intuition may fail to produce any useful projections. For the purpose of this work, we will make the simplifying assumption that a projection is specific to the robotic system only and does not depend on the environment.

B. Contribution

In [10] we showed that simple intuitive projections are suitable for approximating state space coverage for the purpose of motion planning. Using the robot's workspace as a projection has also been shown to be beneficial [22]. In this work, we evaluate the possibility of using random linear projections for the same purpose and we present an offline method to automatically generate such projections. The feasibility of these random linear projections is computed based on comparisons with user-defined projections. In particular, we consider systems of varying complexity to determine how a sampling-based motion planner performs when using random linear projections as opposed to user-defined projections. The planner we use for this purpose is Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (`KPIECE`) [10], an effective state-of-the-art algorithm that was designed to make use of such projection spaces. The results presented in this work are generalizable to other algorithms that employ such projections [6], [9], [22].

II. METHODOLOGY

A. Using a Projection to Estimate State Space Coverage

When guiding the tree expansion based on a low-dimensional projected space $\mathcal{E}(\mathcal{X})$, the question that arises is how the coverage in this lower dimensional space is computed. To the authors' best knowledge, this is in general done by discretizing $\mathcal{E}(\mathcal{X})$. Many options for discretization exist: hierarchy of cells [6], grid [9], [10], [22], triangulation [25]. We call an element of a discretization a cell. Once the projection space $\mathcal{E}(\mathcal{X})$ is discretized, the tree of motions is projected onto it, as it is being grown. The coverage of a cell is typically a function of the cell's size and the number of states on the projected tree that are contained in that cell.

B. Generating a Random Linear Projection

The projections we are attempting to use are random linear projections. The inspiration to use such projections came from a theorem by Johnson and Lindenstrauss [26] which states:

For any $\epsilon > 0$, any n points from a l_2 metric can be embedded in a l_2 metric of dimension $O(\log n/\epsilon^2)$, with $(1+\epsilon)$ distortion.

The proof of this theorem uses random linear projections to achieve this bound. Intrigued by this result, we decided to investigate the performance of random linear projections in sampling-based motion planning.

We will assume a tractable dimension for our projection space; low-dimensional projection spaces are preferred: 2- or 3-dimensional. Using 4- or 5-dimensional projection spaces is possible, but computationally expensive by today's standards. For the purposes of this work, dimensions of k = 2 and k = 3 were used. Next, k vectors in $\mathcal{R}^n$, where $n > k$ is the dimension of the space we are projecting from, are randomly sampled according to a normal distribution (with mean 0 and variance 1). Other methods of sampling, such as uniform sampling, would work as well. The k vectors already constitute a projection, but to avoid representing the same information in multiple dimensions, the Gramm-Schmidt process is ran to make the k vectors orthonormal. For a state $\mathbf{x} \in \mathcal{X}$, a random linear projection $\mathbf{V}$,

$$\mathbf{V} = (\mathbf{v}_1, ..., \mathbf{v}_k),\ \mathbf{v}_i \in \mathcal{R}^n,$$

the projection of $\mathbf{x}$ is $\mathbf{p} \in \mathcal{R}^k$, with

$$\mathbf{p} = \mathbf{V}^T\mathbf{x},$$

assuming all vectors are column vectors.

C. Evaluating a Projection's Score

Since the number of projections we could attempt to evaluate is large, a fast method of approximating a projection's feasibility is needed. As mentioned in Section I-B, we use `KPIECE` for evaluating projections. `KPIECE` is ran twice on a *trial* environment (defined in Section III-A) with a relatively low time limit (shown in Table IV), using the projection to be evaluated as input. If both times a solution was obtained, this projection is considered *valid* and its *score* is set to be the average runtime of the two runs. Otherwise, its score is considered to be infinity. This is not an accurate method of comparing projections, it is only intended to give a rough idea of which projections are potentially good. As it will be shown later, this is adequate for the purposes of this paper.

D. Searching for a Good Random Linear Projection

Finding a good random linear projection proceeds as described in Algorithm 1.

We generate $N_{attempts}$ random linear projections, evaluate them and sort them according to their score. We analyze the 3 projections (referred to as $R1$, $R2$ and $R3$) with lowest score and the projection with median score (referred to as M) in more detail. We do so by running the planner in different environments, with the same random linear projection, and averaging the runtime over N_{runs} runs, where N_{runs} is large enough to obtain statistical significance. These are in fact the values that are compared against running the planner on

the same environments with user-defined projections. Other than the projection, no other inputs to the motion planner are changed. This requires that the same projection works on different environments, so we can keep the projection computation offline. Running the motion planner multiple times on multiple environments with all projections would be computationally expensive and the authors do not believe this is needed. This is why we look at the top 3 projections. The median is also analyzed to evaluate whether or how much the quality of the projections decreases.

Algorithm 1 FINDPROJECTION(k, dim)

k *// the dimension of the random linear projection*
dim *// the dimension of the space projecting from*
for i = 1 **to** $N_{attempts}$ **do**
  // generate projection as in Section II-B
  for j = 1 **to** k **do**
    $\mathbf{v}_j$ = RANDOMVECTOR() *// with normal components*
  end for
  $\mathbf{V} \leftarrow (\mathbf{v}_1, ..., \mathbf{v}_k)$ *// as in Section II-B*
  RUNGRAMMSCHMIDT($\mathbf{V}$)
  $score[i] \leftarrow$ EVALUATEPROJECTION($\mathbf{V}$) *// as in Section II-C*
end for
($R1$, $R2$, $R3$) $\leftarrow$ the 3 projections with lowest $score$
$M \leftarrow$ the projection with median $score$
return ($R1$, $R2$, $R3$, M)

III. EXPERIMENTS

KPIECE uses physics models for the robots it plans the motion for. For this purpose, the Open Dynamics Engine (ODE) [27] (version 0.10) physics simulation library is used. A set of ODE models of increasing complexity are defined in the following section. We consider a robot more complex if it has a higher dimensional state space.

A. Employed Models

Car Robot. A model of a car was created. The model is fairly simple and consists of five parts: the car chassis and four wheels. Since ODE does not allow for direct control of accelerations, desired velocities are given as controls for the forward velocity and steering velocity (as recommended by the developers of the library). These desired velocities go together with a maximum allowed force. The end result is that the car will not be able to achieve the desired velocities instantly, due to the limited force. In effect, this makes the system a second order one. The state space for this model is $\mathcal{X} = \{\mathbf{x} \mid \mathbf{x} = (x, y, \theta, v, \dot{\theta})\}$, where (x, y) denote the center of the car chassis, θ is the car's orientation and v is the velocity along the orientation. The user-defined projection $U1$ is the (x, y) coordinates of the center of the car chassis. The environments the system was tested in are shown in Figure 2.

Blimp Robot. The second robot that was tested was a blimp robot [28]. The motion in this case is executed in a 3D environment. This robot is particularly constrained in its motion: the blimp must always apply a positive force to move forward (slowing down is caused by friction), it must always apply an upward force to lift itself vertically (descending is caused by gravity) and it can turn left or right along the direction of forward motion. Since ODE does not include air friction, a Stokes model of drag was implemented for the blimp. The state space for this model is $\mathcal{X} = \{\mathbf{x} \mid \mathbf{x} = (x, y, z, \theta, v, \dot{z}, \dot{\theta})\}$, where (x, y, z) denote the center of the blimp, θ is the blimp's orientation and v is the forward velocity along the orientation. The user-defined projection $U1$ is the (x, y, z) coordinates of the center of the blimp. The environments the system was tested in are shown in Figure 3.

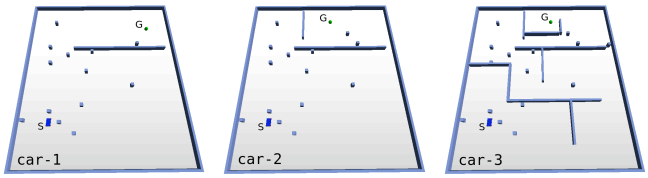


Fig. 2. Environments used for the car robot (car-1, car-2, car-3). Start and goal configurations are marked by "S" and "G". The *trial* environment was car-3.

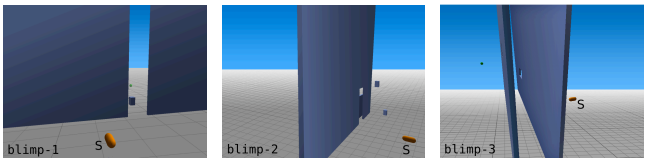


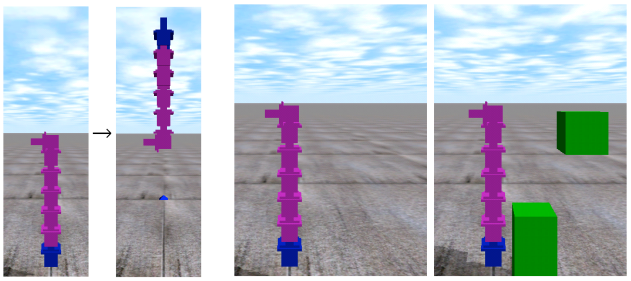
Fig. 3. Environments used for the blimp robot (blimp-1, blimp-2, blimp-3). Start configurations are marked by "S". The blimp has to pass between the walls and through the hole(s) at the center of the wall(s), respectively. The *trial* environment was blimp-3.

Modular Robot. The model for this robot was implemented in collaboration with Dr. Mark Yim[1], and characterizes the CKBot modules [15]. Each CKBot module contains one motor. An ODE model for serially linked CKBot modules has been created [23]. The task is to compute the controls for lifting the robot from a vertical down position to a vertical up position for varying number of modules, as shown in Figure 4. Each module adds one degree of freedom. The controls represent torques that are applied by the motors inside the modules. The difficulty of the problem lies in the high dimensionality of the control and state spaces as the number of modules increases, and in the fact that the maximum torques of the motors in the modules are only able to statically lift approximately 5 modules. This is why the planner has to find swinging motions to solve the problem. The state space for a chain modular robot with m modules is $\mathcal{X} = \{\mathbf{x} \mid \mathbf{x} = ((x_1, \dot{x}_1), ..., (x_m, \dot{x}_m))\}$, where x_i is the angle position of module i, $i \in \{1, .., m\}$. Because this

[1]Dr. Mark Yim is with the Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania yim@grasp.upenn.edu

system proved to be a more difficult one [23], we define multiple user projections.

The projection $U1$ is a 3-dimensional one, the first two dimensions being the (x, z) coordinates of the last module (x, z is the plane observed in Figure 4) and the third dimension, the square root of the sum of squares of the rotational velocities of all the modules. This projection is a difficult one to find, and was first suggested in [29]. A simpler (more intuitive) projection is $U2$: a 2-dimensional projection that takes the first two angles of the chain (the first two components of a state). The intuition behind why this projection could work is that the first two angles that it considers cause the most variance in the positions of the tip of the chain. The environments the system was tested in are shown in Figure 4.



(a) (b)

Fig. 4. (a) Start and goal configurations. (b) Environments used for the chain robot (example shown with 7 modules). In the case without obstacles, the environments are named chain1-*x* where *x* stands for the number of modules used in the chain. In the case with obstacles, the environments are named chain2-*x*. The *trial* environments were chain1-*x*.

B. The ODE State Space

Physics simulation libraries usually define models in terms of bodies and joints. This is the case for ODE as well. In order for motion planning to be performed, the state of the robot (in its ODE world) needs to be retrieved or set. For this purpose, we define the ODE state of a robot to be the set of parameters that completely describe each robot body in the ODE world. For each body, we need the following 13 parameters:

- (p_x, p_y, p_z), position of the body
- $(\dot{p}_x, \dot{p}_y, \dot{p}_z)$, linear velocity of the body
- (r_x, r_y, r_z, r_w), quaternion describing the orientation of the body
- (a_x, a_y, a_z), angular velocity of the body.

The set of ODE states makes up the ODE state space, $\mathcal{X}_{\texttt{ODE}}$. Although $\mathcal{X}_{\texttt{ODE}}$ is usually of higher dimension than $\mathcal{X}$, there is a one-to-one correspondence between states in $\mathcal{X}_{\texttt{ODE}}$ and states in $\mathcal{X}$. This is so because $\mathcal{X}_{\texttt{ODE}}$ may contain redundant information. For instance, in an articulated manipulator, joints constrain certain degrees of freedom. However, specifying all the parameters for each body part of the robot implies a state space where each rigid body is free-flying ($\mathcal{X}_{\texttt{ODE}}$). Of course, the actual states of the robot exist in a usually lower dimensional manifold $\mathcal{X}$, where the manipulator links are connected by joints (i.e., joints are not broken).

The ODE state spaces for the models defined in Section III-A are of dimensionality $13 \cdot 5 = 65$ (we have 5 ODE bodies) for the car, $13 \cdot 1 = 13$ (we have one ODE body) for the blimp and $13 \cdot m$ (we have m ODE bodies) for the modular robot.

Since we are aiming to use random projections, having redundant information is a plus. This is why we in fact generate random linear projections from $\mathcal{X}_{\texttt{ODE}}$ to a low-dimensional space instead of $\mathcal{X}$ to a low-dimensional space. This increases the chances that a random projection will account for pertinent components in the state definition.

C. Results and Discussion

All runtimes reported in this section are the result of running KPIECE with different projections using a shared-memory parallel implementation on an 8-core machine, with 16 GB RAM and a 10 minute time limit. For each value, KPIECE was run $N_{runs} = 50$ times; the best 2 and worse 2 results (in terms of runtime) were dropped; the runtime of the remaining 46 runs was averaged to produce the reported value. All values are reported in seconds. The value of $N_{attempts}$ in Algorithm 1 was 150.

TABLE I

User-defined & 8 random linear projections ($\mathcal{E}$) for the car robot. For each environment, *runtime* (s) / *succsess rate* are reported.

$\mathcal{E}$	k	car-1	car-2	car-3
U1	2	7.15 / 1.00	8.84 / 1.00	15.90 / 1.00
R1	**2**	**5.77 / 1.00**	**7.93 / 1.00**	**14.62 / 1.00**
$R2$	2	6.02 / 1.00	11.13 / 1.00	37.13 / 1.00
$R3$	2	5.58 / 1.00	8.82 / 1.00	24.03 / 1.00
M	2	6.30 / 1.00	8.82 / 1.00	17.72 / 1.00
$R1$	3	8.04 / 1.00	10.51 / 1.00	31.99 / 1.00
$R2$	3	9.27 / 1.00	12.84 / 1.00	37.06 / 1.00
$R3$	3	6.22 / 1.00	7.76 / 1.00	31.12 / 1.00
M	3	6.25 / 1.00	9.43 / 1.00	28.82 / 1.00

TABLE II

User-defined & 8 random linear projections ($\mathcal{E}$) for the blimp robot. For each environment, *runtime* (s) / *succsess rate* are reported.

$\mathcal{E}$	k	blimp-1	blimp-2	blimp-3
U1	3	4.04 / 1.00	7.86 / 1.00	49.24 / 1.00
$R1$	2	6.69 / 1.00	132.76 / 0.78	307.61 / 0.13
$R2$	2	5.42 / 1.00	15.92 / 1.00	273.59 / 0.43
$R3$	2	4.12 / 1.00	12.10 / 1.00	136.74 / 0.59
M	2	10.67 / 1.00	125.47 / 0.93	371.79 / 0.26
$R1$	3	3.50 / 1.00	6.78 / 1.00	74.68 / 0.98
R2	**3**	**3.43 / 1.00**	**7.10 / 1.00**	**38.50 / 1.00**
$R3$	3	3.52 / 1.00	30.36 / 1.00	181.11 / 0.65
M	3	3.56 / 1.00	6.79 / 1.00	64.55 / 1.00

Tables I, II and III show the averaged runtimes of KPIECE using different projections. The user projections were defined by the authors. We tried our best to define projections that work well. We tried different combinations of using the velocity of the car and blimp in their projections but the best results we obtained were with the simplest projections: the workspace. For the modular robot, defining a more

TABLE III

User-defined & 8 random linear projections ($\mathcal{E}$) for each modular robot. For each environment, $runtime$ (s) / $succsess\ rate$ are reported.

N	$\mathcal{E}$	k	chain1-N	chain2-N
5	U1	3	3.11 / 1.00	3.14 / 1.00
	U2	2	3.24 / 1.00	20.71 / 0.76
	$R1$	2	3.63 / 1.00	29.67 / 0.87
	$R2$	2	3.72 / 1.00	51.22 / 0.46
	$R3$	2	3.33 / 1.00	24.66 / 1.00
	M	2	3.64 / 1.00	86.40 / 0.61
	$R1$	3	5.80 / 1.00	26.68 / 1.00
	R2	**3**	**5.10 / 1.00**	**9.20 / 1.00**
	$R3$	3	5.90 / 1.00	21.68 / 1.00
	M	3	6.63 / 1.00	17.40 / 1.00
6	U1	3	3.26 / 1.00	3.34 / 1.00
	U2	2	24.35 / 0.96	85.41 / 0.33
	$R1$	2	150.18 / 0.78	41.48 / 0.13
	$R2$	2	108.01 / 0.65	N/A / 0.00
	$R3$	2	3.79 / 1.00	19.42 / 1.00
	M	2	25.18 / 1.00	109.74 / 0.65
	R1	**3**	**7.94 / 1.00**	**7.68 / 1.00**
	$R2$	3	8.48 / 1.00	8.59 / 1.00
	$R3$	3	10.10 / 1.00	60.09 / 1.00
	M	3	38.65 / 1.00	131.63 / 0.39
7	U1	3	3.92 / 1.00	4.55 / 1.00
	U2	2	120.40 / 0.04	N/A / 0.00
	$R1$	2	10.06 / 1.00	74.95 / 0.67
	$R2$	2	108.20 / 0.20	N/A / 0.00
	$R3$	2	15.33 / 1.00	146.62 / 0.35
	M	2	74.60 / 0.74	409.50 / 0.02
	$R1$	3	19.88 / 1.00	30.32 / 1.00
	R2	**3**	**18.62 / 1.00**	**23.27 / 1.00**
	$R3$	3	34.67 / 1.00	41.67 / 1.00
	M	3	72.12 / 1.00	138.81 / 0.11
8	U1	3	6.04 / 1.00	30.35 / 1.00
	U2	2	N/A / 0.00	N/A / 0.00
	$R1$	2	N/A / 0.00	N/A / 0.00
	$R2$	2	31.14 / 0.17	N/A / 0.00
	$R3$	2	62.67 / 0.13	N/A / 0.00
	M	2	155.81 / 0.28	N/A / 0.00
	$R1$	3	197.12 / 0.09	N/A / 0.00
	R2	**3**	**39.49 / 1.00**	**52.70 / 0.96**
	$R3$	3	162.81 / 0.76	N/A / 0.00
	M	3	212.03 / 0.26	182.23 / 0.26
9	U1	3	37.24 / 1.00	133.84 / 0.48
	U2	2	N/A / 0.00	N/A / 0.00
	$R1$	2	N/A / 0.00	N/A / 0.00
	$R2$	2	N/A / 0.0	N/A / 0.0
	$R3$	2	N/A / 0.0	N/A / 0.0
	M	2	N/A / 0.0	N/A / 0.0
	R1	**3**	**185.90 / 0.67**	**144.51 / 0.93**
	$R2$	3	139.60 / 0.41	228.04 / 0.13
	$R3$	3	201.33 / 0.43	257.78 / 0.76
	M	3	N/A / 0.0	N/A / 0.0
10	U1	3	214.85 / 0.41	656.44 / 0.02
	U2	2	N/A / 0.00	N/A / 0.00
	$R1$	2	N/A / 0.0	N/A / 0.0
	$R2$	2	N/A / 0.0	N/A / 0.0
	$R3$	2	N/A / 0.0	N/A / 0.0
	M	2	N/A / 0.0	N/A / 0.0
	$R1$	3	N/A / 0.0	N/A / 0.0
	$R2$	3	N/A / 0.0	N/A / 0.0
	$R3$	3	N/A / 0.0	N/A / 0.0
	M	3	N/A / 0.0	N/A / 0.0

TABLE IV

The percentage of the projections that were considered valid by the evaluation procedure in Section II-C. Maximum allowed time per trial environment is presented as well.

	2 dimensions		3 dimensions	
Trial environment	valid	time (s)	valid	time (s)
car-3	52.7%	90.0	78.0%	90.0
blimp-3	83.3%	90.0	64.7%	90.0
chain1-5	100.0%	90.0	100.0%	90.0
chain1-6	84.0%	90.0	86.7%	90.0
chain1-7	42.7%	90.0	47.3%	90.0
chain1-8	10.0%	90.0	15.3%	90.0
chain1-9	0.7%	200.0	3.3%	200.0
chain1-10	0.0%	600.0	0.0%	600.0

complicated projection ($U1$) seems to help more [29]; for comparison purposes, we also define a simple projection ($U2$). The $R1$, $R2$, $R3$ and M projections were obtained as discussed in Section II-D, by projecting from the ODE state space $\mathcal{X}_{\texttt{ODE}}$, with $k = 2$ and $k = 3$ (2- and 3-dimensional projection spaces).

We mark in bold-face the random linear projection we believe was best among the 8 tried for each robot model. We observe that in the case of the car and the blimp, the random projections actually do a little better than our user-defined projections. This in itself represents an impressive result, considering the simplicity of the process through which the random projections were found. In addition, looking at Table IV, we notice that the percentage of random linear projections that produce some results, as defined in Section II-C, is very high (above 50%). This means that for systems of moderate dimension, finding a good random linear projection should be an easy task. Further evidence supporting this observation is the fact that the M projections also perform well. Of course, there may be other potentially non-linear projections that could do better.

Looking at Table III, where we test systems with higher-dimensional state spaces, random linear projections do not perform as well as the non-linear user-defined projection $U1$, which took us a long time to find. For 5, 6 and 7 modules we do however get results that are no worse than 5 times slower, with 100% success rate. For 8 modules, we get similar results in terms of runtime but the success rate drops under 100%. At 9 modules an interesting result is observed. With the hard to find $U1$ projection, the success rate is 0.48 (48%) for the chain2-9 environment while with the best found random linear projection, the average runtime is almost the same but the success rate is much higher: 0.93 (93%). At 10 modules, the process in Section II-D did not find any projections, even though we increased the allowed runtime for the trial environment (as shown in Table IV). However, even with the $U1$ projection, we obtain poor results (low success rate). The fact that the runtime is limited to 10 minutes is likely the primary cause for this low success rate. It is possible that using a higher-dimensional projection would also improve results. Comparing with the user-defined projection $U2$, random linear projections do significantly better.

Given that the results for the modular robot could be improved, if comparing to $U1$, we also looked at the possibility that taking our projection from $\mathcal{X}$ instead of $\mathcal{X}_{\texttt{ODE}}$ could be better. We used the algorithm in Section II-D projecting from $\mathcal{X}$, but the results did not improve. In fact, we found no valid random linear projections for more than 7 modules with this setup. This leads us to believe that indeed representing parameters for all the robot's body parts is a better approach.

Overall, for systems with moderate dimension it is likely that easy to find random linear projections will perform well. As the dimension increases, this is no longer the case, but we still get reasonable results.

IV. CONCLUSIONS AND FUTURE WORK

We present an offline method to automatically compute usable, low-dimensional projections for sampling-based motion planners. To the authors' knowledge, this work is a first step in acknowledging the need to carefully look at such projections. A number of existing planners rely on such projections in one way another, but these projections are in general defined by hand. An exception to this is [30], where `PCA` [31] is used locally, in narrow passages, to attempt to find a good direction of growth for a `RRT`. This is different from what we propose in the sense that our projection is global and can be used at every step of the planning algorithm. This has the advantage that it allows faster computation, but may be less accurate for systems where the dimensionality is high. In such cases, a local approach can provide better results.

Last but not least, we do not claim linear projections are sufficient. As our experiments show, a used-defined non-linear projection ($U1$ for modular robot, Section III-A) performs better than randomly produced linear projections. This is a clear indication that more complex, non-linear projections should be considered as well.

ACKNOWLEDGEMENTS

The authors would like to thank Mark Moll for his helpful suggestions and Marius Şucan for help with the pictures in this document.

REFERENCES

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, June 2005.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[3] S. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research: The Eleventh International Symposium*. Berlin: Springer-Verlag, 2005, pp. 36–54.

[4] S. Carpin, "Randomized motion planning - a tutorial," *International Journal of Robotics and Automation*, vol. 21, no. 3, pp. 184–196, 2006.

[5] K. I. Tsianos, I. A. Şucan, and L. E. Kavraki, "Sampling-based robot motion planning: Towards realistic applications." *Computer Science Review*, vol. 1, no. 1, pp. 2–11, August 2007.

[6] A. M. Ladd and L. E. Kavraki, "Motion planning in the presence of drift, underactuation and discrete system changes," in *Robotics: Science and Systems*, Boston, MA, June 2005, pp. 233–241.

[7] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[8] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, March 2002.

[9] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *International Journal of Robotics Research*, vol. 6, pp. 403–417, 2003.

[10] I. A. Şucan and L. E. Kavraki, "Kinodynamic motion planning by interior-exterior cell exploration," in *International Workshop on the Algorithmic Foundations of Robotics*, Guanajuato, Mexico, December 2008.

[11] L. E. Kavraki, J.-C. Latombe, R. Motwani, and P. Raghavan, "Randomized query processing in robot path planning," *Journal of Computer and System Sciences*, vol. 57, no. 1, pp. 50–60, 1998.

[12] A. Ladd and L. Kavraki, "Measure theoretic analysis of probabilistic path planning," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 229–242, 2004.

[13] "Barrett technology, inc." http://www.barrett.com.

[14] J. Bruce and M. Veloso, "Real-time multi-robot motion planning with safe dynamics," *Multi-Robot Systems: From Swarms to Intelligent Automata Volume III*, pp. 159–170, 2005.

[15] J. Sastra, S. Chitta, and M. Yim, "Dynamic rolling for a modular loop robot," *International Journal of Robotics Research*, vol. 39, pp. 421–430, January 2008.

[16] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE International Conference on Robotics and Automation*, vol. 3, April 1997, pp. 2719–2726.

[17] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 995–1001.

[18] E. Plaku, K. Bekris, B. Chen, A. Ladd, and L. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Transactions on Robotics and Automation*, vol. 21, no. 4, pp. 597–608, Aug. 2005.

[19] L. Jaillet, A. Yershova, S. M. LaValle, and T. Siméon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *International Conference on Intelligent Robots and Systems*, 2005, pp. 2851–2856.

[20] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato, "Resampl: A region-sensitive adaptive motion planner," in *International Workshop on the Algorithmic Foundations of Robotics*, New York City, July 2006.

[21] B. Burns and O. Brock, "Single-query motion planning with utility-guided random trees," in *IEEE International Conference on Robotics and Automation*, Rome, Italy, April 2007, pp. 3307–3312.

[22] E. Plaku, M. Y. Vardi, and L. E. Kavraki, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Robotics: Science and Systems*, W. Burgard, O. Brock, and C. Stachniss, Eds. Atlanta, Georgia: MIT Press, June 2007, pp. 326–333.

[23] I. A. Şucan, J. F. Kruse, M. Yim, and L. E. Kavraki, "Kinodynamic motion planning with hardware demonstrations," in *International Conference on Intelligent Robots and Systems*, September 2008, pp. 1661–1666.

[24] ——, "Reconfiguration for modular robots using kinodynamic motion planning," in *ASME Dynamic Systems and Control Conference*, Michigan, Ann Arbor, October 2008.

[25] E. Plaku and L. E. Kavraki, "Impact of workspace decompositions on discrete search leading continuous exploration (dslx) motion planning," in *IEEE International Conference on Robotics and Automation*, Pasadena, CA, May 2008, pp. 3751–3756.

[26] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," *Contemporaty Mathematics*, vol. 26, pp. 189–206, 1984.

[27] http://www.ode.org.

[28] A. M. Ladd and L. E. Kavraki, "Fast tree-based exploration of state space for robots with dynamics," in *Algorithmic Foundations of Robotics VI*. Springer, STAR 17, 2005, pp. 297–312.

[29] A. M. Ladd, "Direct motion planning over simulation of rigid body dynamics with contact," Ph.D. dissertation, Rice University, Houston, Texas, December 2006.

[30] S. Dalibard and J.-P. Laumond, "Control of probabilistic diffusion in motion planning," in *International Workshop on the Algorithmic Foundations of Robotics*, Guanajuato, Mexico, 2008.

[31] I. Jolliffe, *Principal Component Analysis*, 2nd ed. New York, USA: Springer, 2002.