Kinodynamic Motion Planning with Hardware Demonstrations

Ioan A. Şucan Jonathan F. Kruse Mark Yim Lydia E. Kavraki

Abstract— **This paper provides proof-of-concept that state-of-the-art sampling-based motion planners that are tightly integrated with a physics-based simulator can compute paths that can be executed by a physical robotic system. Such a goal has been the subject of intensive research during the last few years and reflects the desire of the motion planning community to produce paths that are directly relevant to realistic mechanical systems and do not need a huge post-processing step in order to be executed on a robotic platform. To evaluate this approach, a recently developed motion planner is used to compute paths for a modular robot constructed from seven modules. These paths are then executed on hardware and compared with the paths predicted by the planner. For the system considered, the planner prediction and the paths achieved by the physical robot match, up to small errors. This work reveals the potential of modern motion planning research and its implications in the design and operation of complex robotic platforms.**

I. INTRODUCTION

During the last two decades, motion planning [1], [2] has grown from a field that considered basic geometric problems, such as the piano movers' problem [3], to a field that tackles planning for complex robots with kinematic and dynamic constraints and has implications to areas such as computational biology and computer graphics [4]. Much of the recent progress in motion planning is attributed to the development of sampling-based algorithms [1], [2], [5], [6], [7]. With the development of this class of methods, it is possible to produce motions that take into account high order dynamic constraints of the robot [8], friction and gravity [9], dynamically changing environments [10] and others. Despite recent successes, it is typically assumed that the motions produced by a planner will undergo a post-processing step, which might alter and tune them significantly, before these are applied to a physical robot.

Only very recent efforts have tried to bridge this gap in the general case [9] by developing powerful planners that are tightly coupled with physical simulators. The latter are used to encode a realistic model of the robot and its potentially complex physical behavior. This paper provides strong evidence that the above approach is a very promising one by demonstrating how the motion plans produced by a variant of the Path-Directed Subdivision Tree Planner (PDST) [11], [9] are applied to a modular robot composed of CKBot modules [12]. Modular robots are composed of multiple small modules (Fig. 2) that can change their connectivity depending on the application. In many cases, the modules are all identical and can be connected in a very large number of topologies, typically exponential in the number of modules [13]. Producing general motion for robots with many degrees of freedom and arbitrary topologies such as these is a challenging problem. This paper discusses motion planning for specific topologies and does not consider reconfiguration.

A model of the robot is built using a physics-based simulator that incorporates the dynamics (inertial components) of the robot. It is shown that the motion produced by the planner described in this work can serve as input to drive the actual robot from an initial to a final state such that the motion executed by the physical robot closely matches the motion produced by the planner. Although preliminary, the results indicate the presented approach is viable for planning motions for real modular robots.

1) Earlier Work - Motion Planning: Sampling-based motion planning became popular with the development of Probabilistic Road-Maps (PRM) [14]. While the basic principles of these methods are simple, many new problems could be solved. For this kind of methods, samples are usually random states of the robot. Other sampling-based motion planning methods were quickly developed to handle more and more complex problems [15], [16], [17], [18], [19], [20], [21]. For sampling-based motion planners, computational improvements come at the cost of completeness. A sampling-based motion planing algorithm can only be probabilistically complete [22], which means it will eventually find a solution if one exists.

The PDST planner, used in this paper, falls under the broader category of sampling-based planners. PDST has been shown to solve kinodynamic problems that are difficult to tackle [11] with popular planners such as Rapidly-exploring Random Trees (RRT) [16], [18] or Expansive Space Trees (EST) [15], [19] by a combination of sampling path segments and keeping a subdivision of the search space to guide the exploration. PDST is probabilistically complete and has been designed to be tightly coupled with a physics-based simulator so as to take realistic models of robots into account [9].

2) Earlier Work - Planning for Modular Robots: While there exists significant work for generating sequences of open loop gaits [23], reconfiguration planning without considering dynamic constraints [24], [25], [26], [27], quasi-static motion planners [28], hand-tuned closed-loop dynamic gaits [12] and control theoretic approaches [29], an algorithm for automatically solving the motion planning problem for modular

This work was supported in part by NSF IIS 0713623 and Rice University funds. The computational experiments were run on equipment obtained by NSF CNS 0454333 and NSF CNS 0421109 in partnership with Rice University, AMD and Cray. I.A. Şucan and L.E. Kavraki are with the Department of Computer Science, Rice University {isucan, kavraki}@rice.edu

J. Kruse and M. Yim are with the Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania {jonathaf, yim}@seas.upenn.edu,@grasp.upenn.edu

robots with dynamics is not available. The work in this paper does not try to optimize a particular path or solve a single problem. Instead, it tries to develop a general algorithm that may not give optimal solutions, but can be used as a black box to produce paths that can be executed by a real system.

3) Contribution: In this work, a step is taken towards the application of a sampling-based motion planner coupled with a physics-based simulator for computing the dynamic motion of a modular robot among obstacles. To the authors' knowledge, this has not been done before. The basic approach is to simulate the robot in a virtual world using a physics-based simulator and compute a continuous path satisfying a set of requirements (such as collision avoidance and robot dynamics) using a sampling-based algorithm. The physics-based simulator used is Open Dynamics Engine (ODE) [30]. The sampling-based algorithm is an improved version of PDST. This algorithm is applied on test problems and the solution path is executed on physical hardware (Section III-.1). Two such test problems are analyzed in this paper. The path the robot actually performs is compared to the path projected by the planner. The end result is that with the described method, the real robot is able to closely follow the projected path. This paper discusses relevant decisions that had to be made in order to obtain this result. Despite the fact that certain aspects of the hardware were not modeled, the average difference at the joint angles between the executed and the projected path is only a few degrees.

This paper does not deal with all the issues that apply to modular robots. While the presented approach will be extended in the future for use with self-reconfiguration, (i.e., the connectivity and topology changes in the middle of a plan) only a single topology of the robot (that does not reconfigure) is used in the described applications.

The organization of the paper is as follows. Section II presents the algorithm used in this work. A particular application and experimental validation follows in Section III. Future work and conclusions are presented in Section IV.

II. MOTION PLANNING APPROACH

For the type of problems addressed in this work, a kinodynamic motion planner is needed. The most popular sampling-based planners for kinodynamic problems generate a tree of collision free motions that obey the dynamic constraints of the robot. There are various ways to guide the tree expansion. RRT expands towards randomly produced states [18], EST attempts to detect less explored regions and expand towards them [19].

As it will be shown later, for the examples in this paper, better results than those of RRT/EST can be achieved with a recent sampling-based motion planner: PDST [9], [11]. The basic principles of PDST are that it generates a tree of motions where samples are path segments instead of states and the coverage of the space is guaranteed through the use of a space subdivision scheme. The space to be subdivided is application specific and usually equal to the state space itself or a projection of the state space.

Algorithm 1 $\texttt{PDST}_{gb}(q_{start}, N_{iterations})$

1: Let p_0 be the path of duration 0 containing q_{start}.
2: $p_0.priority \leftarrow 0$.
3: Initialize priority queue P with p_0.
4: Initialize the subdivision with $\{Q\}$.
5: Initialize priority queue P_b with $\{\}$.
6: **for** $i \leftarrow 1..N_{iterations}$ **do**
7: Let $bias$ be **true iff** $rand() < \mathcal{P}$ **and** $P_b \neq \{\}$.
8: **if** $bias$ **then**
9: Let s be a random sample from P_b.
10: Let $(c_{new}, t_{new}) \leftarrow$ HILLCLIMB(s).
11: **else**
12: Let $s \in P$ such that $score(s)$ is minimized.
13: Let (c_{new}, t_{new}) be random control and duration.
14: **end if**
15: $new =$ SIMULATE(s, c_{new}, t_{new}).
16: **if** simulation was successful **then**
17: If new is a solution, **return** path to new.
18: $new.priority \leftarrow i$.
19: Add new to P and P_b.
20: Drop last element of P_b if $P_b.size() > \mathcal{M}$
21: **end if**
22: **if not** $bias$ **then**
23: $s.priority \leftarrow 2 \cdot s.priority + 1$.
24: Update subdivision by splitting cell containing s.
25: **end if**
26: Update P, P_b so each sample lies in a unique cell.
27: **end for**

The PDST algorithm proceeds as follows. The root of the generated tree is a path of duration 0 consisting solely of the starting state. The tree is built iteratively as described in Algorithm 1. The purpose of the subdivision space Q is to help in deciding in which direction exploration should continue, and guarantee probabilistic completeness [9]. A sample cell subdivision is shown in Fig. 1. The samples are assigned a score and kept in a priority queue. For a sample s, $score(s) = s.priority/volume(s.cell)$, $s.priority$ is initialized to the number of the iteration that produced s and $s.cell$ identifies the cell containing s. At every iteration, the sample with minimal score is selected (line 12). A new sample is formed by branching from the selected sample at a random time in a random direction (line 15). The cell containing the selected sample is subdivided (line 24). At the end of each iteration, the invariant that a sample cannot intersect multiple cells must be satisfied. This means that some samples from the subdivided cell and the newly inserted sample may need to be split (line 26).

The results obtained by PDST can be improved if some goal biasing is added (Section III-.4). The main change to the basic PDST implementation this paper proposes is the use of *ranking functions*. These are application specific functions that assign a rank to every state – a heuristically obtained positive value indicating how close a state is to the goal region. A ranking function can be a metric, but does not need to be. The rank of a sample is defined to be the rank of the

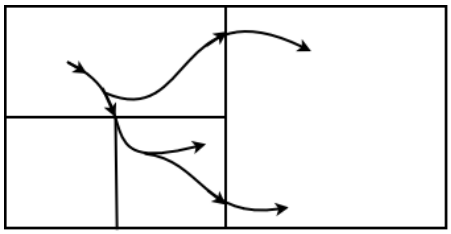

Fig. 1. 2-dimensional subdivision in a run of PDST (3 iterations).

last state of the sample. In the process of adding samples to the tree of motions (line 19), a small (maximum size is $\mathcal{M}$) secondary priority queue is maintained such that it contains the samples with highest rank (line 20). Samples from this priority queue are used for goal biasing: with a small probability ($\mathcal{P} \in [0, 1)$, line 7), a sample from the secondary priority queue is selected instead of the sample with minimal score (line 9). In this case, cell subdivision is not performed. Extra time is spent however selecting the control to be applied (line 13). Since the planner attempts to branch from a state known to be among the closest ones to the goal, time could be saved by further progressing towards the goal. A hill climbing algorithm is used to produce a control that maximizes the rank of the generated sample. This algorithm works by successively increasing and decreasing the value of each component in the applied control and keeping the changes that increase the rank of the generated sample. With these changes, the motion planner is referred to as PDST with goal biasing, or $\texttt{PDST}_{gb}$. It is important to note that since $\mathcal{P} < 1$, $\texttt{PDST}_{gb}$ retains PDST's probabilistic completeness property.

III. APPLICATION AND EXPERIMENTS

This section presents an application of $\texttt{PDST}_{gb}$ to planning motion for CKBot modules. First, a model of the robot described in Section III-.1 is constructed. This model is placed in two test workspaces: one without any obstacles and the other with obstacles (see Fig. 4). For each workspace, $\texttt{PDST}_{gb}$ is used to compute a solution to the problem described in Fig. 3. Once a solution is found, the controls that lead to that solution are extracted. The feasibility of the produced solution is then tested by sending the controls to the real robot and checking if it performs the same motion the planner projected.

1) Robots: The hardware test-bed of this work consists of seven identical CKBot modules [12]. The modules are serially connected, forming a chain. The first module of the chain is rigidly attached to a cantilever such that the cantilever itself does not interfere with the motion of any of the other modules. One module can be viewed as a cube with connectors on top, bottom, left, and right faces as shown in Fig. 2. The top, left and right faces are rigidly mounted together, the bottom face is actuated to rotate up to form the front or rear face of a perfect cube. These modules each have one rotational degree of freedom controlled with a position feedback hobby servo, a micro controller, CANbus [31] communications, and a structure made of laser-cut ABS plastic. Colored acrylic is attached to each module, along their centerlines, for use in position tracking (see Fig. 2).

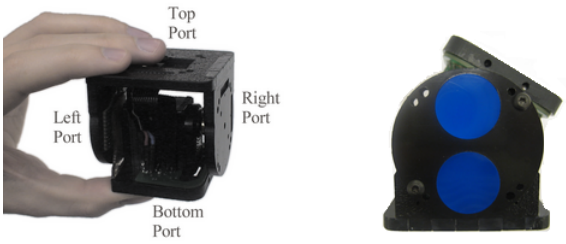

Fig. 2. Different views of a CKBot module.

The modules can either have their on-board controller to command the servo to move to a sequence of positions, or position commands can be communicated using the Robotics bus protocol [31] over a CANbus. The hobby servos have a highly tuned position feedback control built in and accept commands from the micro controller at 60Hz. Each module is attached to another on one of 4 faces by screwing them together. For this work, the modules were attached end-to-end in a serial chain with power and high level position commands issued off-board.

2) Simulated Model for the Hardware: A model of the hardware module described in Section III-.1 was created for ODE [30]. Based on this model, a method for automatically creating ODE models of arbitrarily connected modules was designed. A model of the chain robot previously described was then constructed. The model includes dimensions, masses of the rigid bodies and joint limits of the structure. Two attributes of the CKBot servo motors were difficult to determine and were not included in the model.

The first is the highly tuned position control code (typically PID). ODE easily models torque commands to rigid bodies and represents realistic motion from these commands. However, the servos in CKBot cannot be commanded by torque, they only receive desired angle commands. Attempts were made to have $\texttt{PDST}_{gb}$ apply desired angles as controls and let an ODE function act as the servo motors' on-board controller, but modeling the controller turned out to be inaccurate and achieving positions close to the goal was not possible.

The second unmodeled attribute is the inertia and friction from the gear train within the servo. The reflected inertia of the motor and transmission in the servo was approximated by adding a "friction" force at every time-step which was a torque equal to a constant times the joint angle velocity, $-k\dot{\theta}$, applied to the hinge joint.

The motion planner itself is not aware of any of the parameters describing the model. The only information the planner has is the number of joints it can control, which is the number of modules - 1, the first module being fixed to the space. The fact that the first module is anchored has no bearing in the planning process, it only decreases the dimension of the state space by 1. There are no conceptual difficulties with considering problems where the first module is not anchored, but such examples were outside the scope

of this paper.

3) The Problem: A simple task to test the feasibility of the proposed method is computing the controls for lifting the robot from a vertical down position to a vertical up position, as shown in Fig. 3.

The difficulty of the problem lies in the fact that the maximum torques of the motors in the modules are only able to statically lift approximately 5 modules. One strategy to overcome this, is to exploit momentum to reduce the maximum torque seen by the motors. Creating a sequence of motions manually would be difficult as the timing of the issued commands is critical. In [29], an example of control theoretic methods for finding optimal controllers for underactuated serial chains in actions such as swinging up is presented. However, those methods cannot automatically handle arbitrary configurations nor arbitrary obstacles in the workspace of the robot.

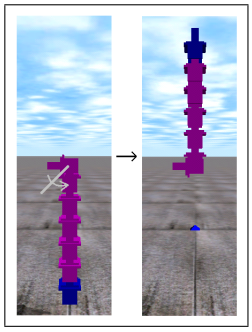

Fig. 3. The start and goal states (7 modules).

The experiment was tried in two virtual worlds (see Fig. 4), both of which allow a solution. The first world (`Test 1`) has no obstacles and only challenges the planner to find feasible controls. The second world (`Test 2`) includes two obstacles, posing the additional difficulty of avoiding the obstacle. This forces the planner to find a path that uses momentum while keeping the lower part of the robot rotated up to avoid collisions.

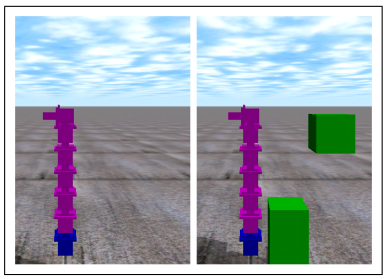

Fig. 4. Used environments: one with (`Test 1`) and the other without (`Test 2`) obstacles. The obstacles are in the plane of the robot and limit the motion. The top module is attached to the space (7 modules).

$\texttt{PDST}_{gb}$ only relies on the subdivision scheme and the ranking functions for directing the search. The reason for this is that the developed algorithm should be reusable with different starting and ending robot states. The space where subdivision occurs for this problem 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 the images) and the third dimension, the square root of the sum of squares of the rotational velocities of all the modules. Choosing these dimensions for the subdivision allows the planner to explore the positions in the workspace the robot can possibly reach with its endpoint (the first two dimensions) and the space of angular velocities (last dimension). Exploring the third dimension allows the planner to more easily find paths that make use of momentum. The ranking function takes into account the height reached (the z coordinate of the last module) and the sum of squares of the angles between modules (to reflect how close to vertical the robot is).

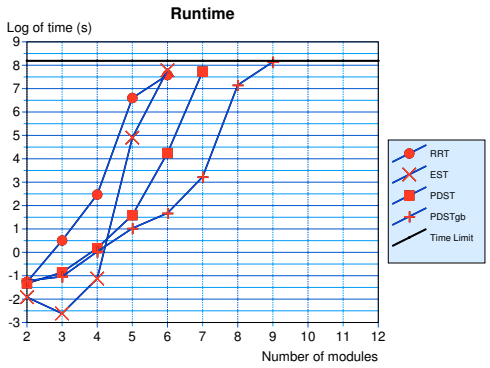

Fig. 6. Logarithmic execution times in `Test 1` for `RRT`, `EST`, `PDST`, $\texttt{PDST}_{gb}$ with varying number of modules. Time limit set to one hour.

TABLE I

RUNTIME (SECONDS) FOR THE DIFFERENT ALGORITHMS

	Test 1		Test 2	
	6 modules	7 modules	6 modules	7 modules
RRT	1959.46	N/A	N/A	N/A
EST	2402.67	N/A	3504.19	N/A
PDST	69.77	2269	60.68	1656
$\texttt{PDST}_{gb}$	5.34	25	18.57	143

4) Computation of Paths: $\texttt{PDST}_{gb}$ was executed on the environments shown in Fig. 4. The parameters used were $\mathcal{P} = 0.05, \mathcal{M} = 20$. The runtime was limited to one hour. For comparison purposes, `RRT`, `EST` and `PDST` were also ran to solve the same problem. All implementations are in C++. For `EST` and `RRT`, the implementation from the `OOPSMP` framework [32] was used. The metric that was found best for `RRT` was the Euclidean distance between positions of the last module. The results are averaged over 30 runs (after removing the slowest and fastest two runs). A run is considered to have found a solution if it reaches a height of at least 95% of the maximally achievable height. When close enough to the goal state, the hardware controller can take over and achieve the desired final state. Experiments were run on the Rice Cray XD1 cluster where each processor is at 2.2 Ghz and has up to 8GB RAM. Fig. 6 shows a logarithmic plot of the runtime of these algorithms for different number of modules. For 6 and 7 modules, runtimes are presented in Table I. The results clearly show the need for algorithms like `PDST`, as well as the benefits of adding biasing. Lifting

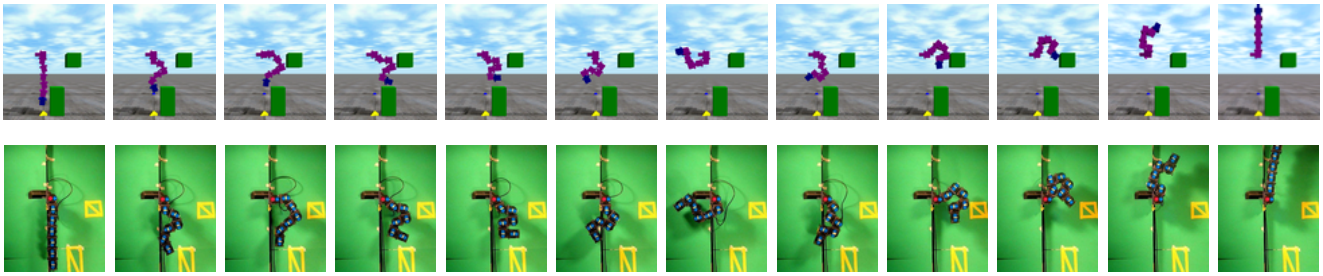

Fig. 5. Experiment results. The top row includes frames from the simulation. The bottom row includes frames from the implementation in hardware.

the robot can be solved with up to 9 modules when using $\texttt{PDST}_{gb}$ as opposed to 6 modules when using EST or RRT. In Test 2, the placement of obstacles is such that it poses difficulty especially for biasing. As expected, Table I shows the runtime for $\texttt{PDST}_{gb}$ increases from Test 1 to Test 2. An interesting effect of this particular workspace is that PDST is able to ignore part of the space it should explore, thus having a slightly shorter execution time on Test 2. Even so, without biasing, PDST takes significantly more time to find a solution. Significant effort was put into generating realistic paths, to avoid jerky motions and not exceed the capabilities of the hardware. These constraints slow down the planning process, hence the need for faster planners.

The output of the algorithm is a succession of states, 0.05 seconds apart. Each state describes the exact position of all the modules in the robot and the torques to be applied for achieving the next state. However, the hardware only needed the joint angles of the modules. A set of screen-shots from both the simulation and the hardware execution with seven modules are shown in Fig. 5.

5) Hardware Validation: In order to validate that the path produced by $\texttt{PDST}_{gb}$ is feasible, the output needs to be tested on the hardware. The angles generated by $\texttt{PDST}_{gb}$ algorithm are converted into a favorable format for the hardware's controller. However, the differences between the ODE model and the servo control need to be addressed. The servo uses position control applying torques to move the joint to a commanded angle as quickly as possible, and once there, to have an angular velocity of zero. $\texttt{PDST}_{gb}$ uses torques as controls for its path generation, and essentially takes snapshots of the angles of the servos without velocity being constrained to zero at each time step.

In order to successfully execute a path, adjustments are made to the rate at which instructions are sent to the hardware in lines per second (LPS) where each line is one position command to all the modules and are nominally executed at a fixed rate. The tuning in the vertical lifting experiment adjusts the LPS to minimize the tendency for the servos to reach zero angular velocity at each step. By adjusting the speed, so many of the servos are near maximum force, the servo controls are saturated and thus more predictable. The LPS within each run would vary between 25 and 60 LPS for the run without obstacles and 6 and 60 for the run with obstacles.

For example, if a swinging path is attempted at a slower instruction rate, the servos would essentially try to stop at each of the desired positions, producing a jerky motion that does not take advantage of momentum or gravity. With too high of an instruction rate, the servos cannot keep pace with the desired trajectory resulting in large errors. If the proper instruction rate is selected (and changed properly throughout the execution of the path), the servos are able to utilize the momentum of the modules to complete the desired path. By tuning the rate at which these instructions are given to the modules, the servo motors are able to better match the $\texttt{PDST}_{gb}$ generated path.

6) Evaluation of the results: In order to quantitatively measure the differences between the instruction data generated by $\texttt{PDST}_{gb}$ and the real path execution, a vision system and a program using the Image Processing Toolbox from MATLAB tracked the positions of the colored acrylic circles on each module throughout the execution of the path.

The program finds the center points of each circle and creates vectors describing each module's position and direction. From these vectors, the angles between each adjacent modules are calculated. These results are then compared to the instruction input from $\texttt{PDST}_{gb}$, after being scaled for differences in time. These values are then computed to give the average error between hardware output and instruction input per module for each instruction given (see Table II). With seven modules, one module serves as the base, so the six angles in the table represent the average joint error between the hardware output and the instructions given for each module. These differences most likely stem from the non-modeled position control of the servo motors in the simulation.

TABLE II

AVERAGE JOINT ANGLE ERROR (IN DEGREES) BETWEEN ODE AND HARDWARE IMPLEMENTATION (7 MODULES)

Without Obstacles					
1	2	3	4	5	6
4.064	8.5044	5.2193	5.2519	3.6285	1.1467
With Obstacles					
1	2	3	4	5	6
7.2188	8.5283	6.767	5.0843	5.3291	4.3001

It is significant that the average error is not too large even though the low-level hardware control is not modeled. It may be that a position based low-level control is more robust to errors in system identification than a pure torque control method. A position feedback servo with some model errors will try to track the desired trajectory to a first order where a torque controller with the same model errors may lead

to integrated position errors. However, a system with little model error will likely perform best with low-level torque control.

As the work continues, the aim will be to decrease these error values through a better simulation that can more accurately take into account many more physical attributes of the modules and their motors. These results raise some interesting questions about what ranges of error are acceptable in automated path generation. If the robot reaches the final desired position, or reaches the desired position while avoiding all obstacles, how much error in the overall performance is allowable?

IV. CONCLUSIONS AND FUTURE WORK

This paper explores the application of a motion planner to a hardware platform. The motion planner ($\texttt{PDST}_{gb}$) is tightly integrated with a physics-based simulator (ODE) which uses a model of a real robot constructed from CKBot modules. $\texttt{PDST}_{gb}$ is run on two test problems and the produced paths are executed on a real robot. The path the real robot executes closely follows the path projected by the planner. This represents strong evidence that the method is viable and can lead to solving more complex problems.

The authors plan to continue improving the motion planner, reduce its runtime and tackle more complex problems.

At the same time, a new module that has a direct-drive high-torque brushless motor as the main drive instead of the servo is almost complete. This new module will likely alleviate many of the errors and inconsistencies in the ODE model which should enable much more interesting dynamic experiments, with higher degrees of freedom, more complex motions and environments, creating motions that would otherwise be impossible for a human to create.

V. ACKNOWLEDGMENTS

The authors would like to thank Mark Moll, Erion Plaku and Konstantinos Tsianos for reading this work and providing valuable comments.

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] J. Schwartz and M. Sharir, "On the piano movers' problem: General techniques for computing topological properties of real algebraic manifolds," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345–398, 1983.

[4] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *International Journal of Robotics Research*, vol. 18, no. 11, pp. 1119–1128, November 1999.

[5] 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.

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

[7] 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.

[8] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *IEEE International Conference on Robotics and Automation*, 2007.

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

[10] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *IEEE International Conference on Robotics and Automation*, 2006.

[11] 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.

[12] 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.

[13] M. Park, S. Chitta, A. Teichman, and M. Yim, "Automatic configuration recognition methods in modular robots," *International Journal of Robotics Research*, vol. 27, pp. 403–421, March 2008.

[14] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.

[15] 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.

[16] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 11, 1998.

[17] R. Bohlin and L. Kavraki, "Path planning using lazy prm," in *IEEE International Conference on Robotics and Automation*, vol. 1, 24-28 April 2000, pp. 521–528.

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

[19] 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.

[20] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *International Journal of Robotics Research*, pp. 403–407, 2003.

[21] E. Plaku, M. Y. Vardi, and L. E. Kavraki, "Discrete search leading continuous exploration for kinodynamic motion planning," in *Robotics: Science and Systems*, Atlanta, Georgia, 2007.

[22] 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.

[23] M. H. Yim, C. Eldershaw, Y. Zhang, and D. G. Duff, "Limbless conforming gaits with modular robots," in *International Symposium on Experimental Robotics*. Singapore: Springer-Verlag, June 2004.

[24] J. E. Walter, J. L. Welch, and N. M. Amato, "Distributed reconfiguration of metamorphic robot chains," in *Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, 2000, pp. 171–180.

[25] Z. Butler, R. Fitch, and D. Rus, "Distributed control for unit-compressible robots: goal-recognition, locomotion, and splitting," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 4, pp. 418–430, December 2002.

[26] C. C. Ünsal, H. Kiliççöte, and P. K. Khosla, "A modular self-reconfigurable bipartite robotic system: Implementation and motion planning," *Autonomous Robots*, vol. 10, pp. 67–82, 2001.

[27] K. C. Prevas, C. Ünsal, M. O. Efe, and P. K. Khosla, "A hierarchical motion planning strategy for a uniform self-reconfigurable modular robotic system," in *IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002, pp. 787–792.

[28] C. Eldershaw and M. Yim, "Motion planning of legged vehicles in an unstructured environment," in *IEEE International Conference on Robotics and Automation*, 2001, pp. 3383–3389.

[29] G. Sohl and J. Bobrow, "Optimal motions for underactuated manipulators," in *Proc. of Design Engineering Technical Conference*, 1999.

[30] R. Smith, "Open dynamics engine," http://www.ode.org.

[31] D. Gomez-Ibanez, E. Stump, B. Grocholsky, V. Kumar, and C. Taylor, "The robotics bus: a local communications bus for robots," in *Mobile Robots XVII*, D. W. Gage, Ed., vol. 5609, 2004, pp. 155–163.

[32] E. Plaku, K. E. Bekris, and L. E. Kavraki, "OOPS for Motion Planning: An Online Open-source Programming System," in *IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007, pp. 3711–3716.