

Integrated Planning and Control for Graceful Navigation of Shape-Accelerated Underactuated Balancing Mobile Robots

Umashankar Nagarajan, George Kantor and Ralph Hollis

Abstract—This paper presents controllers called **motion policies** that achieve fast, graceful motions in small, collision-free domains of the position space for balancing mobile robots like the ballbot. The motion policies are designed such that their valid compositions will produce overall graceful motions. An automatic instantiation procedure deploys motion policies on a 2D map of the environment to form a library and the validity of their composition is given by a gracefully prepares graph. Dijkstra's algorithm is used to plan in the space of these motion policies to achieve the desired navigation task. A hybrid controller is used to switch between the motion policies. The results of successful experimental testing of two navigation tasks, namely, point-point and surveillance motions on the ballbot platform are presented.

I. INTRODUCTION

Personal mobile robots will soon be operating in human environments, offering a variety of assistive technologies that will augment our capabilities and enhance our lives. Balancing mobile robots can be effective personal robots as they can be tall enough for eye-level interaction and narrow enough to navigate cluttered environments. They are also dynamically capable of moving with speed and grace comparable to humans. This paper presents an integrated motion planning and control procedure that enables balancing mobile robots like the ballbot [1] to navigate human environments in a graceful manner. The ballbot is an underactuated, human-sized mobile robot that balances on a ball. Unlike its two-wheeled counterparts [2], [3], the ballbot is omnidirectional. Our early successes with the ballbot have encouraged many other groups [4], [5] to explore such designs.

Motion planning and control for mobile robots have been traditionally decoupled. A high-level motion planner plans a collision-free path that achieves the overall navigation goal and a low-level controller attempts to follow this path. Traditionally, the motion planner has no knowledge of either the system dynamics or the controller used to track the planned motion and the controller has no knowledge of either the environment constraints or the overall navigation goal. These decoupled approaches can work well in kinematic wheeled robots but will fail miserably in highly dynamic, balancing mobile robots like the ballbot. Such approaches result in sub-optimal, jerky motions and often drive the system unstable. In order to achieve robust, collision-free graceful motions, the motion planning and control for such systems must be integrated.

The last decade has seen several approaches towards integrating planning and control procedures for robotic systems. Burrige *et al.* [6] introduced *Sequential Composition*, a controller composition technique that connects different control policies and switches between them to generate a globally convergent feedback policy. It was successfully applied to a variety of systems [7], [8]. Conner *et al.* [9] used sequential composition to achieve global navigation tasks for convex-bodied wheeled mobile robots. The control policies were deployed on a map of the environment and their composability relationship was given by a directed graph called *prepares graph*. Graph search algorithms were used to find a sequence of control policies to achieve the navigation task. In our previous work [10], we extended this procedure to balancing mobile robots like the ballbot.

Though the procedure presented in [10] was capable of navigating an environment with obstacles, the resulting robot motion was not graceful. This is primarily due to the discrete switching between the control policies. The control policies were not designed such that their composition results in an overall graceful motion. In the present work, we define graceful motion to be any feasible robot motion in which its configuration variables' velocity and acceleration trajectories are continuous and bounded. Continuous and bounded acceleration trajectories exhibit low jerk, are visually appealing, and result in smoothed actuator loads. Moreover, high jerk trajectories can excite the resonant frequencies of the robot, which can drive a balancing system unstable.

Frazzoli *et al.* [11] presented *Maneuver Automata*, which used open-loop maneuvers and steady-state trim trajectories as motion primitives, which consisted of feasible state and control trajectories. They used RRT-like [12] algorithms for motion planning in maneuver space and demonstrated aggressive maneuvering capabilities of autonomous helicopters in simulations [13]. Since coverage of the free space is not the objective of the algorithm, it replans every time the state exits the defined domain. Tedrake *et al.* [14] introduced *LQR-trees* algorithm, which builds a sparse tree of LQR-stabilized trajectories with verified stability regions that probabilistically cover the controllable subset of the state space. Though this algorithm can be applied to a variety of nonlinear systems, it is computationally expensive and has only been demonstrated on systems of dimension up to five. The verification of the stability regions is the bottleneck in its application to high dimensional systems.

Contributions: In this paper, we present an integrated planning and control procedure that ensures overall graceful navigation of balancing mobile robots like the ballbot. The

U. Nagarajan, G. Kantor and R. Hollis are with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA umashankar@cmu.edu, kantor@ri.cmu.edu and rhollis@cs.cmu.edu

contributions of this paper are: (i) a design procedure for gracefully composable control policies called the *motion policies* that result in collision-free graceful motions in small domains of position space (see Sec. III); (ii) an automatic instantiation procedure that generates a motion policy library from a small collection of motion policies (see Sec. IV-A); (iii) the notion of *gracefully prepares relationship*, which is a restrictive definition on the prepares relationship [6], [9] (see Sec. III-C); and (iv) the successful experimental testing on the ballbot platform to perform two navigation tasks, namely, point-point and surveillance motions along with handling disturbances (see Sec. V).

II. THE BALLBOT

In this work, we model the ballbot (Fig. 1(a)) as a rigid cylinder on top of a rigid sphere with the following assumptions: (i) there is no slip between the ball and the floor, and (ii) there is no yaw/spinning motion for both the ball and the body, *i.e.*, they have 2-DOF each. A planar model with the planar configurations is shown in Fig. 1(b).

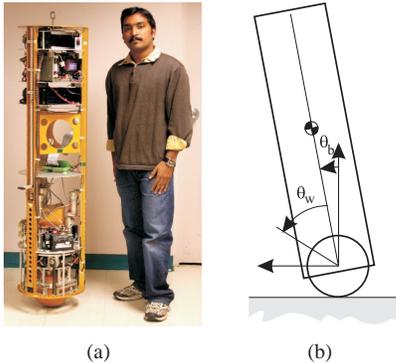


Fig. 1. (a) The ballbot balancing; (b) Planar ballbot model.

The configuration space of any dynamic system can be divided into *position* and *shape* space. The position variables q_x represent the position of the mobile robot in the world, and the robot dynamics is invariant to transformations of its position variables. However, the shape variables q_s affect the inertia matrix of the system and dominate the system dynamics. For the ballbot, the ball angles form the position variables, *i.e.*, $q_x = [\theta_w^x, \theta_w^y]^T \in \mathbb{R}^{2 \times 1}$, while the roll and pitch angles of the body form the shape variables, *i.e.*, $q_s = [\theta_b^x, \theta_b^y]^T \in \mathbb{R}^{2 \times 1}$. The equations of motion of the 3D ballbot model can be written as:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = \begin{bmatrix} \tau \\ \mathbf{0} \end{bmatrix}, \quad (1)$$

where, $q = [q_x, q_s]^T \in \mathbb{R}^{4 \times 1}$, $M(q) \in \mathbb{R}^{4 \times 4}$ is the mass/inertia matrix, $C(q, \dot{q}) \in \mathbb{R}^{4 \times 4}$ is the Coriolis and centrifugal matrix, $G(q) \in \mathbb{R}^{4 \times 1}$ is the vector of gravitational forces and $\tau \in \mathbb{R}^{2 \times 1}$ is the vector of actuator inputs. The last two equations of motion in Eq. 1 form the dynamic constraint equations and are of the form:

$$\Theta(q_s, \dot{q}_s, \ddot{q}_s, \dot{q}_x) = \mathbf{0}. \quad (2)$$

In shape-accelerated balancing mobile robots [15] like the ballbot, any non-zero shape change generally results in

acceleration in position space. Although we are primarily interested in the motions in position space for navigation purposes, planning for appropriate motions in shape space that result in desired motions in position space is the only way to make balancing mobile robots move with speed and grace. In [15], we presented an optimal shape trajectory planner that uses the dynamic constraint equations (Eq. 2) to optimally track desired position trajectories. This planning procedure finds feasible state (position and shape) trajectories that best approximate any desired position space motion.

III. MOTION POLICY DESIGN

This section presents the design of control policies called motion policies for the ballbot using the optimal shape trajectory planner [15] and the control architecture shown in Fig. 2. Each motion policy consists of a reference state trajectory called a motion primitive, a time-varying feedback trajectory tracking controller and a time-varying domain that is verified to be asymptotically convergent [10].

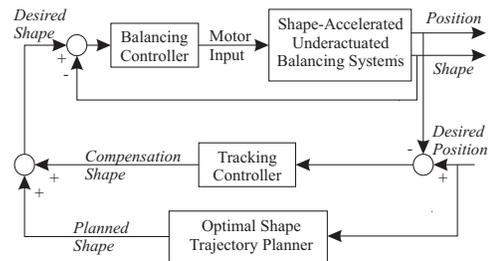


Fig. 2. The control architecture.

A. Motion Primitives

Motion primitives are elementary, feasible state trajectories that produce graceful motions in small domains of the position space and they can be combined sequentially to produce more complicated trajectories. In this work, these trajectories are designed to be graceful, *i.e.*, the position, velocity and acceleration trajectories are continuous and bounded. Moreover, they are feasible state trajectories and hence satisfy the dynamic constraints of the system.

Following [11], we define two classes of motion primitives: (i) Trim primitives and (ii) Maneuvers. Trim primitives are motion primitives that correspond to steady-state conditions and they can be arbitrarily trimmed (cut), *i.e.*, the time duration of the trajectory can be arbitrarily chosen. In this work, we restrict trim primitives to constant position or velocity trajectories in position space with zero shape change. Maneuvers are motion primitives that start and end at steady-state conditions given by the trim primitives. Unlike trim primitives, maneuvers have fixed time duration and non-zero shape change. There is no restriction on how complicated the maneuver motion can be, for example, a maneuver can result in a *S-curve* or *U-turn* motion in position space. In [11], the motion primitives consisted of both state and control trajectories, whereas, in this paper, the motion primitives consist only of feasible state trajectories.

In this work, we define a motion primitive set $\Sigma(d)$ as a collection of motion primitives, each of which produces a

net Δx and Δy motion in position space such that Δx and Δy are integral multiples of the distance parameter d . Figure 3(a) shows position space motions of some example motion primitives from a motion primitive set with the distance parameter $d = 0.5$ m. Here, the desired position trajectories were chosen to be nonic polynomials. The feasible shape and position trajectories that best achieve these desired position space motions were obtained using the optimal shape trajectory planner described in [15], [16].

The motion primitives presented in Fig. 3(a) may strike a strong resemblance to the state lattices [17] used by the motion planners in unmanned ground vehicles. The state lattices represent feasible paths and the lattice planners plan in the space of these paths, whereas, in this paper, the motion planner plans in the space of motion policies, which are controllers designed around motion primitives as will be described in the following sections.

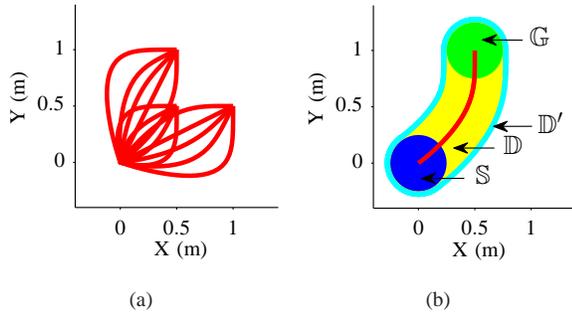


Fig. 3. (a) Position space motions of example motion primitives with $d = 0.5$ m; (b) XY projection of a motion policy domain.

B. Motion Policies

Each motion policy Φ_i consists of a motion primitive $\sigma_i(t)$, a time-varying feedback tracking control law $\phi_i(t)$ and a time-varying domain $D_i(t)$. The motion policy domains $D(t)$ are defined as geometric domains in 4D position state space, *i.e.*, (x, y, \dot{x}, \dot{y}) , similar to the ones in [10]. The motion policies use the control architecture in Fig. 2 that exploits the strong coupling between the shape and position dynamics to achieve the desired position space motions. The effectiveness of this control architecture has been experimentally demonstrated on the ballbot in [18], [16]. Since the motion policy control architecture achieves motions in position space by controlling the shape space motions, we restrict the policy domain definitions to 4D position state space.

In this paper, we define these time-varying domains $D(t)$ as 4D hyper-ellipsoids centered around the time-varying desired position states of the motion primitives. Each time-varying domain $D(t)$ has a start domain $\mathbb{S} = D(0)$ and a goal domain $\mathbb{G} = D(t_f)$. Moreover, each domain $D(t)$ has another domain $D'(t)$ defined such that $D(t) \subset D'(t) \forall t \in [0, t_f]$ and any state trajectory starting in \mathbb{S} will remain in $D'(t)$ until it reaches $\mathbb{G} \forall t \in [0, t_f]$. The overall domains for each motion

policy are given by $\mathbb{D} = \bigcup_{t=0}^{t_f} D(t)$ and $\mathbb{D}' = \bigcup_{t=0}^{t_f} D'(t)$. An XY projection of an example motion policy domain is shown in Fig. 3(b). The geometric domain definitions make it easier to verify the validity of a motion policy for a position state.

The verification of these domains is done using the 3D dynamic model of the ballbot system. Various system identification experiments were conducted [19] on the ballbot to estimate the system parameters such that the dynamics of the model better match the real robot dynamics. The dynamic model under the action of the control architecture in Fig. 2 was simulated from finitely many states on the surface of the start domain \mathbb{S}_i of each motion policy Φ_i and was verified to remain inside the domain $D'_i(t) \forall t \in [0, t_f]$ until it reaches the goal domain \mathbb{G}_i . Moreover, the resulting shape space motions were also verified to be in the domain of the balancing controller, which tracks them.

In this paper, we define a motion policy palette $\Pi(\Sigma)$ to be a collection of unique motion policies whose constituent motion primitives belong to the motion primitive set $\Sigma(d)$ with the distance parameter d .

C. Gracefully Prepares Relationship

In this section, we define the *gracefully prepares relationship* between motion policies that ensures graceful switching between them. A motion policy Φ_1 is said to gracefully prepare Φ_2 , *i.e.*, $\Phi_1 \succeq_G \Phi_2$ if:

- (i) The goal domain of Φ_1 is contained in the start domain of Φ_2 , *i.e.*, $\mathbb{G}_1 \subset \mathbb{S}_2$;
- (ii) The motion primitive $\sigma_1(t)$ of the motion policy Φ_1 is gracefully composable with the motion primitive $\sigma_2(t)$ of the motion policy Φ_2 , *i.e.*, $\sigma_1(t_{f_1}) = \sigma_2(0)$ and $\dot{\sigma}_1(t_{f_1}) = \dot{\sigma}_2(0)$. This ensures that the overall reference position, velocity and acceleration trajectories are continuous; and
- (iii) The time-varying feedback control law $\phi_1(t)$ of the motion policy Φ_1 is gracefully composable with the feedback control law $\phi_2(t)$ of the motion policy Φ_2 , *i.e.*, $\phi_1(t_{f_1}) = \phi_2(0)$. This ensures that the overall closed-loop control trajectory is continuous.

The first condition satisfies the *prepares relationship* [6], while the next two conditions reduce it to a *gracefully prepares relationship*. Hence, any *gracefully prepares relationship* is by definition a *prepares relationship*, *i.e.*, $\Phi_1 \succeq_G \Phi_2 \Rightarrow \Phi_1 \succeq \Phi_2$ but not vice-versa.

Since a sequence of gracefully composable motion policies consists of continuous reference position, velocity, acceleration and control trajectories, the resulting closed-loop motion is graceful. In this work, the motion policy palette $\Pi(\Sigma)$ with a motion primitive set $\Sigma(d)$ is designed such that for every pair of gracefully composable motion primitives (σ_1, σ_2) , their corresponding time-varying feedback control laws $(\phi_1(t), \phi_2(t))$ are designed to be gracefully composable, *i.e.*, if $\sigma_1(t_{f_1}) = \sigma_2(0)$ and $\dot{\sigma}_1(t_{f_1}) = \dot{\sigma}_2(0)$, then $\phi_1(t)$ and $\phi_2(t)$ are designed such that $\phi_1(t_{f_1}) = \phi_2(0)$.

IV. INTEGRATED PLANNING AND CONTROL

Section III presented the offline procedure to design gracefully composable motion policies. Now, this section presents the integrated planning and control procedure that will run real-time on the robot. First, it will present an automatic instantiation procedure that generates a large motion policy

library from the offline generated motion policy palette with a few motion policies. It also presents the procedure to plan in the space of motion policies and the hybrid control architecture used to execute the plans.

A. Automatic Instantiation of Motion Policies

The dynamics of wheeled mobile robots are invariant to transformations of their position variables. This allows us to place these motion policies at any point in the position space in any orientation. The process of setting the initial position and orientation of the motion policies and their constituent motion primitives is called *instantiation*.

While many instantiation approaches are possible, in this paper, we present a simple approach of uniformly distributing the motion policies in position and orientation space to illustrate the concept. Given a motion policy palette $\Pi(\Sigma)$ and a 2D map of the environment \mathbb{M} , the map is uniformly discretized into instantiation points separated by the distance d along X and Y directions. This distance d is given by the distance parameter of the motion primitive set $\Sigma(d)$. A large collection of motion policies can be generated from just a small number of motion policies in the motion policy palette by instantiating them at the instantiation points in different pre-defined orientations. The discretization in orientation space depends on the position space motions produced by the motion policies in the motion policy palette. In this paper, we design motion policies that produce motions in the first quadrant of the position space as shown in Fig. 3(a) and hence the orientation spacing is set to 90° .

An instantiated motion policy is valid only if its domain \mathbb{D}' is obstacle-free. We define a motion policy library $\mathbb{L}(\Pi, \mathbb{M})$ as a collection of valid instantiations of motion policies $\Phi_i \in \Pi$ on the map M . This automatic instantiation procedure calculates the percentage of the bounded position state space covered by the start domains of the motion policies in the library by uniformly sampling the bounded position state space. If the desired coverage (100%) is not achieved then the grid spacing is halved and this process continues until either the desired coverage or the maximum number of such iterations is achieved. In the case of failing to achieve the desired coverage, there will be regions in the map that cannot be achieved using this navigation procedure.

The gracefully prepares relationship between every pair of motion policies (Φ_1, Φ_2) in the motion policy library $\mathbb{L}(\Pi, \mathbb{M})$ can be determined using the conditions presented in Sec. III-C. A directed graph called the *gracefully prepares graph* $\Omega(\mathbb{L})$ is generated where each node represents an instantiated motion policy and each directed edge represents the gracefully prepares relationship.

B. Planning in Motion Policy Space

The gracefully prepares graph $\Omega(\mathbb{L})$ contains all possible graceful motions that the robot can perform using the motion policies in the motion policy library $\mathbb{L}(\Pi, \mathbb{M})$. The problem of navigating the map \mathbb{M} can now be formulated as a graph search problem. Unlike traditional approaches of motion planning in the space of discrete cells or paths, in this paper,

we use the graph search algorithms to plan in the space of motion policies (controllers). The graph search algorithms now provide a sequence of motion policies to achieve the overall navigation task.

In this work, any navigation task is assumed to be a motion between trim motion policies, *i.e.*, motion policies with trim motion primitives. In order to illustrate the validity of this assumption, let's consider two navigation tasks: (i) point-point motion and (ii) surveillance motion. Any point-point motion can be formulated as a motion between trim motion policies that have constant position trajectories as trim primitives. Similarly, any surveillance motion can be formulated as a motion between trim motion policies that have constant velocity trajectories as trim primitives.

Given a goal position state, we use the Euclidean distance metric to find the closest trim motion policy whose goal domain contains it. In this paper, we use Dijkstra's algorithm [20] to solve a single-goal optimal navigation problem for the gracefully prepares graph. Some candidates for the optimality criterion are fastest time and shortest path.

Unlike other graph search algorithms like A^* that find a path between two nodes in the graph, Dijkstra's algorithm generates a single-goal optimal tree $\Gamma(\Omega, G)$ from the gracefully prepares graph $\Omega(\mathbb{L})$ such that the optimal path from all nodes in the graph to the goal node G is obtained. This ensures that all trim conditions in the motion policy library from which the goal can be reached will be reached by switching between the motion policies in the tree. Fig. 4 shows example optimal sequences of motion policies with their domain projections from different initial positions. Each node in the optimal tree $\Gamma(\Omega, G)$ contains a motion policy and a pointer to the next node that is optimal towards reaching the goal node. For a fixed goal navigation task, Dijkstra's algorithm is powerful as it does not require any replanning (unlike A^*) even when the robot's position state jumps to another motion policy domain.

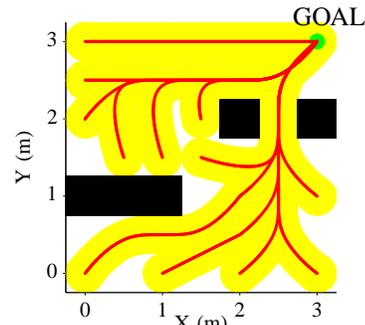


Fig. 4. An example time-optimal single goal motion policy tree (partial) with three obstacles, shown in black.

C. Hybrid Control

Given any start node in the optimal tree $\Gamma(\Omega, G)$, the optimal path to the goal node G is obtained by following the next pointer in the node. A hybrid control architecture is used as a master/supervisory controller to enable successful completion of the navigation task. It is responsible for executing the current motion policy and also switching to the next optimal motion policy.

The hybrid control architecture contains a timer, which is reset at the start of every motion policy execution and it runs out at the end of the motion policy’s time duration t_f . The execution of a motion policy Φ_i is initiated only if the robot’s position state is in the start domain \mathbb{S}_i of the motion policy and it continues until the timer runs out and the robot’s position state is in the goal domain \mathbb{G}_i of the motion policy. During motion policy execution, the hybrid controller checks whether the robot’s position state lies inside the domain $D'(t) \forall t \in [0, t_f]$. The switching to the next motion policy Φ_j happens naturally because $\mathbb{G}_j \subset \mathbb{S}_i$ by construction. Therefore, the presence of the robot’s position state in \mathbb{G}_i implies its presence in \mathbb{S}_j .

The feedback control law $\phi_i(t)$ is capable of handling small disturbances and uncertainties. But in case of large disturbances, the robot’s position state can exit the domain $D'(t)$, for some $t \in [0, t_f]$. In such a case, the hybrid controller stops execution of the current motion policy, finds a motion policy Φ_k whose start domain \mathbb{S}_k contains the robot’s position state and starts its execution. There is no need for replanning as the optimal tree $\Gamma(\Omega, G)$ will have an optimal path from the current motion policy Φ_k to the goal node G . This motion policy switching is discrete and is not graceful as the disturbance added to the system is discontinuous.

If 100% coverage was guaranteed during the automatic instantiation procedure, then there will always exist a motion policy that captures the robot’s exiting position state. But if 100% coverage was not guaranteed and if no such motion policy exists, then the hybrid controller stops the navigation task and switches to the simple balancing mode.

V. EXPERIMENTAL RESULTS

This section presents the results of successful experimental testing of the proposed integrated planning and control procedure on the ballbot platform in our lab. The ballbot uses a particle filter based localization algorithm [21] for localizing itself on a 2D map of the lab. The odometry data is provided by the encoders on the ball motors and the laser readings are provided by a Hokuyo URG-04LX laser range finder with a 180° field of view mounted on the front of the robot. The ballbot uses occupancy grids for detecting obstacles with the laser data.

For all the results presented here, the motion policy palette we used consisted of 39 unique motion policies. The corresponding motion primitives form a motion primitive set with distance parameter $d = 0.5$ m (Fig. 3(a)). The motion policies were automatically instantiated in a $3.5 \text{ m} \times 3.5 \text{ m}$ free area in the lab as described in Sec. IV-A. After instantiation, the motion policy library consisted of 4521 instantiated motion policies. The automatic instantiation of the motion policies and gracefully prepares graph generation happened in 2.5 s on the dual core computer on the robot. Dijkstra’s algorithm implementation only takes 0.05 s to generate the single-goal optimal tree on the same computer. This allows real-time regeneration of the optimal tree, which is needed to account for changing goals and dynamic obstacles. In this work, we choose fastest time as the optimality criterion.

The companion video, *Integrated Planning and Control for Graceful Navigation of Ballbot*, shows the ballbot performing all the navigation tasks presented here.

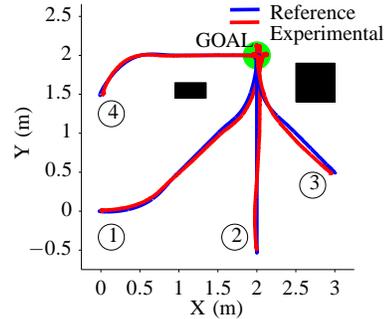


Fig. 5. Point-Point motion with two obstacles, shown in black.

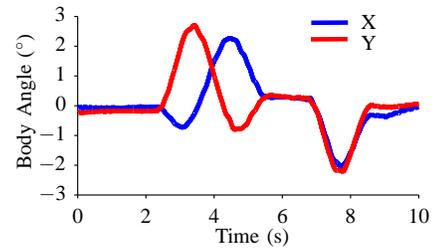


Fig. 6. Body angle trajectories for the point-point motion no. 4.

A. Point-Point Motion

The point-point motion is a motion between static/rest configurations. This navigation task can be formulated as the motion between trim motion policies with constant position motion primitives. Figure 5 shows the ballbot reaching a single goal position state of (2 m, 2 m, 0 m/s, 0 m/s) from four different starting configurations using a single-goal fastest time tree. The resulting body angle trajectories for the fourth motion is shown in Fig. 6.

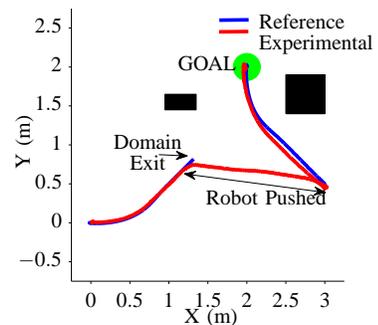


Fig. 7. Handling disturbances during point-point motion.

B. Disturbance Handling

To illustrate the ability of our integrated procedure to handle large disturbances, we physically stopped the ballbot from moving towards the goal while executing a point-point motion. Then, we dragged it to a different point on the map and let go. The hybrid control architecture detected the domain exit of the motion policy it was executing and continued to find the best motion policy whose domain contained the exit state. Since the robot was moved by hand in an orthogonal direction to its desired motion, the position

state kept exiting the domain of any chosen motion policy until it was set free to move on its own. Figure 7 shows the robot successfully reaching the goal once it's set free.

C. Surveillance

The surveillance motion is a motion between moving configurations and the task is specified as a sequence of moving goal configurations that repeat. This navigation task can be formulated as the motion between trim motion policies that have constant velocity motion primitives. Unlike point-point motion, the surveillance motion has changing goals and hence the optimal tree is regenerated while executing the last motion policy to the current goal. This process repeats until the user quits the surveillance task. The successful

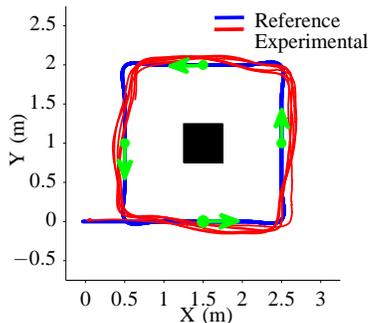


Fig. 8. Surveillance motion with four goal configurations, shown in green and one obstacle, shown in black.

execution of five loops of a surveillance motion with four goal configurations is shown in Fig. 8. The resulting body angle trajectories are shown in Fig. 9.

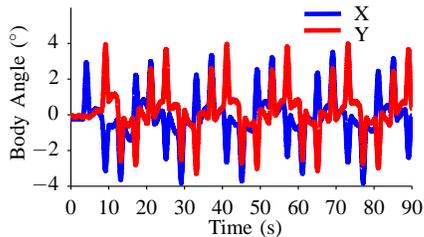


Fig. 9. Body angle trajectories for the surveillance motion.

VI. CONCLUSIONS AND FUTURE WORK

An integrated planning and control procedure that enables graceful navigation of balancing mobile robots in human environments was presented and successfully tested on the ballbot platform. The optimality of the resulting motion is limited by the motion policies available in the library. It is difficult to achieve 100% coverage in environments with narrow paths as the motion policy domain definitions presented in this work are fixed. We will explore ways to scale-down these policy domains during instantiation so that the obstacles can be avoided and the desired coverage can also be guaranteed.

Dijkstra's algorithm may not be fast enough for navigation problems covering larger areas. Therefore, other heuristic based graph search algorithms like D* must be considered. The design of heuristics is a challenge for such applications

and must be explored. Another approach to handle navigation problems in larger areas will be to divide them into smaller regions and piece the locally optimal motions together to achieve the global goal.

VII. ACKNOWLEDGEMENTS

This work was supported in part by NSF grants IIS-0308067 and IIS-0535183. We thank Joydeep Biswas for providing us the localization algorithm implementation.

REFERENCES

- [1] R. Hollis, "Ballbots," *Scientific American*, pp. 72–78, October 2006.
- [2] P. Deegan, B. Thibodeau, and R. Grupen, "Designing a self-stabilizing robot for dynamic mobile manipulation," *Robotics: Science and Systems - Workshop on Manipulation for Human Environments*, 2006.
- [3] M. Stilman, J. Olson, and W. Gloss, "Golem Krang: Dynamically stable humanoid robot for mobile manipulation," in *IEEE Int'l Conf. on Robotics and Automation*, 2010, pp. 3304–3309.
- [4] M. Kumagai and T. Ochiai, "Development of a robot balancing on a ball," *Intl. Conf. on Control, Automation and Systems*, 2008.
- [5] <http://www.rezero.ethz.ch/project.en.html>.
- [6] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.
- [7] A. A. Rizzi, J. Gowdy, and R. L. Hollis, "Distributed coordination in modular precision assembly systems," *The International Journal of Robotics Research*, vol. 20, no. 10, pp. 819–838, 2001.
- [8] G. Kantor and A. A. Rizzi, "Feedback control of underactuated systems via sequential composition: Visually guided control of a unicycle," in *11th International Symposium of Robotics Research*, Siena, Italy, October 2003.
- [9] D. C. Conner, H. Choset, and A. A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback," in *Proc. Robotics: Science and Systems II*, 2006, pp. 57–64.
- [10] U. Nagarajan, G. Kantor, and R. Hollis, "Hybrid control for navigation of shape-accelerated underactuated balancing systems," in *Proc. IEEE Conference on Decision and Control*, 2010, pp. 3566–3571.
- [11] E. Frazzoli, M. A. Dahleh, and E. Feron, "Maneuver-based motion planning for nonlinear systems with symmetries," *IEEE Transactions on Robotics*, vol. 21, no. 6, 2005.
- [12] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," in *TR 98-11, Computer Science Dept., Iowa State University*, Oct. 1998.
- [13] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA J. Guid., Control, Dynam.*, vol. 25, no. 1, 2002.
- [14] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *International Journal of Robotics Research*, vol. 29, no. 8, 2010.
- [15] U. Nagarajan, "Dynamic constraint-based optimal shape trajectory planner for shape-accelerated underactuated balancing systems," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, 2010.
- [16] U. Nagarajan, B. Kim, and R. Hollis, "Planning in high-dimensional shape space for a single-wheeled balancing mobile robot with arms," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, St. Paul, USA, 2012.
- [17] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005.
- [18] U. Nagarajan, G. Kantor, and R. Hollis, "Trajectory planning and control of an underactuated dynamically stable single spherical wheeled mobile robot," *Proc. IEEE Int'l. Conf. on Robotics and Automation*, pp. 3743–3748, 2009.
- [19] U. Nagarajan, A. Mampetta, G. Kantor, and R. Hollis, "State transition, balancing, station keeping, and yaw control for a dynamically stable single spherical wheel mobile robot," *Proc. IEEE Int'l. Conf. on Robotics and Automation*, pp. 998–1003, 2009.
- [20] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
- [21] J. Biswas, B. Coltin, and M. Veloso, "Corrective gradient refinement for mobile robot localization," in *Proc. IEEE Int'l Conf. on Intelligent Robots and Systems*, 2011, pp. 73–78.