International Journal of Robotics Research (IJRR), Special Issue: Motion Planning for Physical Robots
Volume 32, Issue 9 – 10, pp. 1005 – 1029, September 2013

Integrated Motion Planning and Control for Graceful Balancing Mobile Robots

Umashankar Nagarajan, George Kantor, and Ralph Hollis

***Abstract*—This paper presents an integrated motion planning and control framework that enables balancing mobile robots to gracefully navigate human environments. A palette of controllers called *motion policies* is designed such that balancing mobile robots can achieve fast, graceful motions in small, collision-free domains of the position space. The domains determine the validity of a motion policy at any point in the robot's position state space. An automatic instantiation procedure that generates a motion policy library by deploying motion policies from a palette on a map of the environment is presented. A *gracefully prepares relationship* that guarantees valid compositions of motion policies to produce overall graceful motion is introduced. A directed graph called the *gracefully prepares graph* is used to represent all valid compositions of motion policies in the motion policy library. The navigation tasks are achieved by planning in the space of these gracefully composable motion policies. In this work, Dijsktra's algorithm is used to generate a single-goal optimal motion policy tree, and its variant is used to rapidly replan the optimal motion policy tree in the presence of dynamic obstacles. A hybrid controller is used as a supervisory controller to ensure successful execution of motion policies and also successful switching between them.**

The integrated motion planning and control framework presented in this paper was experimentally tested on the ballbot, a human-sized dynamically stable mobile robot that balances on a single ball. The results of successful experimental testing of two navigation tasks, namely, point-point and surveillance motions are presented. Additional experimental results that validate the framework's capability to handle disturbances and rapidly replan in the presence of dynamic obstacles are also presented.

I. Introduction

Personal mobile robots will soon be roaming freely in human environments, operating in our proximity and interacting with us. They will provide an array of assistive technologies that will augment our capabilities and enhance the quality of our lives. Personal robots operating in human environments and interacting with humans do not have to necessarily look like humans but must move, act and interact like humans. They should be tall enough for eye-level interaction, narrow enough to navigate cluttered spaces, and should be able to move with speed and grace comparable to that of humans.

However, traditional robotic locomotion platforms are three or four-wheeled platforms that move slowly, and have wide bases and low centers of gravity. They are statically stable robots, $i.e.$, they stand still when powered down. A human-sized statically stable mobile robot needs a wide base to have a large polygon of support, and a lot of dead weight in the base to keep its center of gravity as low as possible. A high center of gravity and/or a small polygon of support can cause the robot to tip over easily, which is undesirable. The chances of tipping over drastically increase when lifting heavy objects, and while moving up or down steep slopes because the net center of gravity can shift outside the polygon of support (Fig. 1). The wide bases, however, make statically stable robots unsuitable for operation in human environments that are often narrow and cluttered.

A. The Need for Balancing Robots

The drawbacks of statically stable mobile robots can be avoided by building mobile robots that actively balance, just like humans do. Balancing mobile robots are dynamically stable, and can be tall and skinny with high centers of gravity. They can have small footprints as they are continually balancing, and can accelerate or decelerate quickly (Hollis 2006). Balancing mobile robots can also avoid tipping by actively compensating for the shift in center of gravity as shown in Fig. 1.

Moreover, balancing mobile robots are physically interactive. People have the need to physically interact with machines in their environments, especially when they are their personal robotic assistants. Statically stable robots need a multiplicity of sensors to detect mechanical forces imparted to their bodies, whereas balancing mobile robots are naturally reactive because they inherently respond to mechanical forces as disturbances

U. Nagarajan is with Disney Research, Pittsburgh, PA 15213, USA. This work was done when he was with The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA. email: umashankar@disneyresearch.com

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

International Journal of Robotics Research (IJRR), Special Issue: Motion Planning for Physical Robots
Volume 32, Issue 9 – 10, pp. 1005 – 1029, September 2013

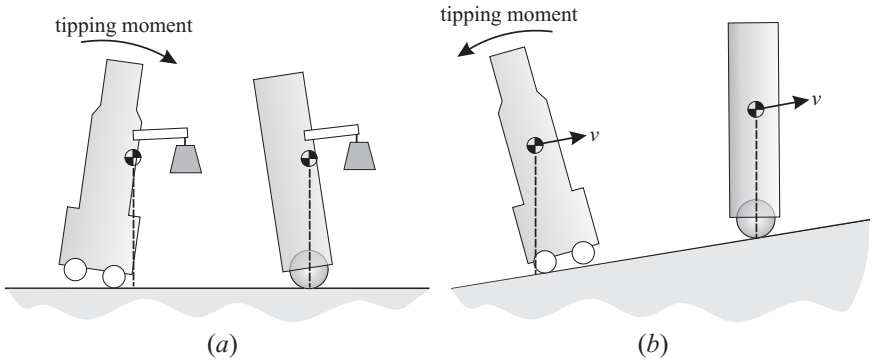


Fig. 1. (*a*) A statically stable robot can tip over when attempting to lift a heavy weight, whereas a balancing robot can lean to keep the net center of gravity over the point of ground support. (*b*) A statically stable robot could tip when going up or down slopes, whereas a balancing robot can stay balanced on slopes.

to their balancing behavior.This makes balancing robots responsive to human touch (Nagarajan et al. 2009b). All these characteristics make balancing mobile robots ideal candidates for personal robotic assistants that move, act and interact like humans.

B. Balancing Wheeled Robots

There has been a significant growth of interest in developing balancing mobile robots in the last decade. Two-wheeled balancing mobile robots became popular after the introduction of the *Segway Robotic Mobility Platform* (Nguyen et al. 2004). Rod Grupen and his group introduced a two-wheeled balancing mobile robot called *uBot* (Deegan et al. 2006), which is used as a mobile manipulation research platform. They demonstrated that balancing mobile robots can be effective mobile manipulators with the ability to maintain postural stability, generate forces on external objects, and withstand greater impact forces (Deegan et al. 2007). Dean Kamen introduced *iBot* (iBOT 2003), a balancing wheelchair, and demonstrated its advantages over its statically stable counterparts. Anybots (Anybots 2010) introduced a tele-presence robot that balanced on two wheels. Mike Stilman and his group introduced *Golem Krang* (Stilman et al. 2010), a two-wheeled balancing mobile manipulator platform that has the capability to autonomously stand and sit.

Our group introduced the *ballbot* (Hollis 2006; Lauwers et al. 2005, 2006), the first successful dynamically stable mobile robot that balances on a single ball. As it moves on a ball, it is omnidirectional, and hence, circumvents the limitations associated with the kinematic constraints of two-wheeled mobile robots. We have demonstrated the robustness, dynamic motion, and physical interaction capabilities of the ballbot in our previous work (Nagarajan et al. 2009a,b,c). Recently, several other groups have developed single-wheeled balancing mobile robots. Masaaki Kumagai developed the *BallIP* (Kumagai and Ochiai 2008), and demonstrated that ball balancing robots can be used for co-operative transportation of wooden frames (Kumagai and Ochiai 2009). A group of mechanical engineering students at ETH Zurich developed the *Rezero* (Rezero 2010), and re-emphasized the dynamic capabilities of ball balancing mobile robots.

C. The Need for Graceful Motion

Balancing mobile robots are capable of moving with speed and grace comparable to that of humans. The objective of the work presented in this paper is to enable balancing mobile robots like the ballbot to achieve graceful navigation in human environments. In this work, graceful motion is defined as *any feasible robot motion in which its configuration variables' position, velocity and acceleration trajectories are continuous and bounded with low jerk*.

Apart from being visually appealing, a graceful robot motion has a variety of advantages. Graceful, low jerk motions result in smoothed actuator loads (Kyriakopoulos and Saridis 1988) because jerk is directly proportional to the torque rate of actuators. High jerk motions, on the other hand, can excite resonant frequencies of the robot, which can drive a balancing system unstable.

It has been shown in biomechanics literature that humans tend to move such that their movements minimize jerk (Hogan 1984). Minimum-jerk models have been proposed to model arm movements (Flash and Hogan 1985), and are used in various rehabilitation and haptic applications (Amirabdollahian et al. 2002). Smoothness is a characteristic of unimpaired human movements,

and humans generally associate "high jerk" motions to "panic" motions (Amirabdollahian et al. 2002; Rohrer et al. 2002). Therefore, humans are unlikely to feel comfortable around robots that have high jerk, non-smooth (ungraceful) motions.

Moreover, personal mobile robots operating in human environments are likely to engage in physical interactions with humans, wherein high jerk motions can be undesirable as humans interacting with the robots will also experience the same. Therefore, in order to build successful personal robots that operate and interact in human environments, it is important to ensure that they have graceful motions.

D. The Need for Integrated Planning and Control

The objective of the work presented in this paper is to enable balancing mobile robots to achieve graceful navigation in human environments. Traditionally, motion planning and control for mobile robots have been decoupled. A high-level motion planner plans a collision-free path, and a low-level controller tracks it. The motion planner does not understand the capabilities and limitations of the controller, while the controller has no knowledge of the environment and the obstacles in it. This decoupled approach works well in achieving navigation tasks for kinematic mobile robots whose dynamics can be safely ignored, but for balancing mobile robots with significant dynamics, such a decoupled approach generally results in sub-optimal and ungraceful motions. Moreover, when subjected to disturbances, the decoupled approach often results in collisions with obstacles or drives the balancing system unstable.

Therefore, in order to make balancing mobile robots to navigate human environments in a graceful and collision-free manner, it is essential to integrate motion planning and control. The motion planner must understand and respect the constraints of both the system dynamics and its controllers. The controller, on the other hand, must be aware of the obstacles and navigation objectives.

E. Approach towards Graceful Navigation

This paper presents an integrated motion planning and control framework based on sequential composition (Burridge et al. 1999; Conner et al. 2006; Nagarajan et al. 2010) that enables balancing mobile robots like the ballbot to achieve the desired navigation tasks while moving gracefully.

The approach presented in this paper has two phases: (i) an offline controller design phase, and (ii) an online planning phase. In the offline controller design phase, controllers called *motion policies* that track feasible state trajectories called *motion primitives* are designed. A palette of motion policies is designed such that the individual motion policies result in graceful motion, and there exist combinations of motion policies that are gracefully composable. When two motion policies are gracefully composable, they guarantee graceful switching between them, thereby resulting in an overall graceful motion. In the online planning phase, a motion policy library is generated by automatically instantiating the motion policies from the palette to fill a map of the environment. A motion planner plans in the space of these gracefully composable collision-free motion policies to achieve desired navigation tasks. This paper is an extended and significantly improved version of the paper presented in (Nagarajan et al. 2012a).

F. Contributions

The contributions of this paper are as follows:

- (i) the notion of *gracefully prepares relationship* as a restrictive definition on the prepares relationship (Burridge et al. 1999; Conner et al. 2006), which ensures graceful switching between motion policies (see Sec. IV-C);
- (ii) an offline procedure to design a palette of motion policies, wherein the motion policies produce collision-free graceful motions in small domains of position space, and also gracefully prepare each other (see Sec. IV);
- (iii) an automatic instantiation procedure that fills a map of the environment with motion policies from a palette and generates a motion policy library (see Sec. V-A);
- (iv) a motion planner that plans in the space of motion policies that gracefully prepare each other to achieve desired navigation tasks, and also replans to avoid dynamic obstacles (see Sec. V-B and Sec. V-D); and
- (v) the experimental testing on the ballbot to successfully achieve two navigation tasks, namely, point-point and surveillance motions, while handling disturbances and dynamic obstacles (see Sec. VI).

II. Related Work

The last decade has seen a large body of work on using hybrid control techniques to integrate motion planning and control. This section discusses some of the existing work most relevant to the work presented in this paper.

A. Sequential Composition

Burridge et al. introduced *Sequential Composition* (Burridge et al. 1999), a controller composition

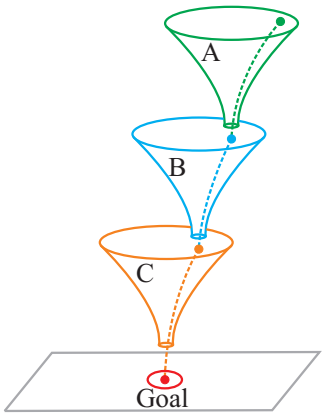


Fig. 2. Prepares relationship represented using funnels (Burridge et al. 1999).

technique that connects a sequence of control policies, and automatically switches between them to generate a globally convergent feedback control policy. Given a set of control policies $\mathbb{U} = \{\Phi_1, \cdots \Phi_n\}$, each with an invariant domain $\mathbb{D}(\Phi_i)$ and a goal set $\mathbb{G}(\Phi_i)$, a control policy Φ_1 is said to *prepare* another control policy Φ_2, denoted by $\Phi_1 \succeq \Phi_2$, if the goal of the control policy Φ_1 lies inside the domain of the control policy Φ_2, *i.e.*, $\mathbb{G}(\Phi_1) \subset \mathbb{D}(\Phi_2)$. The *prepares relationship* between control policies can be represented using cascading funnels, where one control policy leads to the other as shown in Fig. 2. The stability and convergence of the individual control policies guarantee the stability and convergence of any valid sequence of control policies given by the prepares relationship. Sequential composition was successfully applied to a variety of systems (Kantor and Rizzi 2003; Klavins and Koditschek 2000; Rizzi et al. 2001).

Conner et al. presented an integrated motion planning and control procedure based on sequential composition to achieve navigation tasks for kinematic wheeled robots (Conner et al. 2006). A map of the environment was randomly filled with instantiations of a pre-defined set of control policies using a partially automated procedure. The invariant domains for these control policies were restricted to the configuration space of the system, and the system dynamics were ignored.

In our previous work (Nagarajan et al. 2010), the integrated motion planning and control procedure presented in (Conner et al. 2006) was extended to balancing mobile robots like the ballbot. The system dynamics were not ignored, and in fact, the control policies exploited the natural dynamics of the system to achieve desired motions. All these approaches (Burridge et al. 1999; Conner et al. 2006; Nagarajan et al. 2010) ensured stability and convergence of a sequential composition of control policies, but did not guarantee graceful motion. Although these approaches resulted in a robust system that can navigate a map with obstacles under disturbances, the robot did not achieve graceful motion. This paper presents a sequential composition based approach that ensures overall graceful motion while still switching between different control policies to achieve desired navigation tasks.

B. Other Hybrid Control Approaches

Belta et al. presented a hybrid control policy with piecewise affine control policies defined over simplices (Belta et al. 2005). The motion planning was performed in the space of simplices, and control policies were designed over each simplex that induced the desired closed-loop motion. This approach was presented only for kinematic mobile robots, and it did not produce overall graceful motion.

Frazzoli et al. presented *Maneuver Automata* (Frazzoli et al. 2005) that used open-loop maneuvers and steady-state trim trajectories as motion primitives, which consisted of feasible state and control trajectories. They used algorithms based on Rapidly-exploring Random Trees (RRT) (LaValle and Kuffner 2001) for motion planning in maneuver space, and demonstrated aggressive maneuvering capabilities of autonomous helicopters in simulations (Frazzoli et al. 2002). The algorithm presented did not deal with coverage but rather, stopped when a sequence of motion primitives to the goal was found. Therefore, every time the state exited the defined domain, the algorithm had to replan. They also presented *Robust Hybrid Automata* (Frazzoli et al. 2000) that used closed-loop control for its maneuvers. Although this approach ensured overall stability of the closed-loop system while switching between motion primitives, it did not ensure closed-loop graceful motion.

Russ Tedrake introduced *LQR-trees*, a feedback motion planning algorithm that combines locally valid linear quadratic regulator (LQR) controllers into a nonlinear feedback policy that globally stabilizes a goal state (Tedrake 2009; Tedrake et al. 2010; Tobenkin et al. 2011). The controllable subset of the state space was probabilistically covered by verified stability regions of a sparse set of LQR-stabilized trajectories. The estimation and verification of the stability regions are computationally expensive, and hence do not allow real-time planning. However, these approaches to estimate invariant domains for control policies can be used in the design of motion policies presented in this paper.

C. Hybrid Motion Planning Approaches

Hauser et al. demonstrated successful humanoid walking on uneven terrains in simulation (Hauser et al. 2008). Unlike approaches that limit the derived motions to the motion primitives, they used motion primitives to derive sampling strategies for a probabilistic sampling-based planner that generated new motions. Plaku et al. developed *SyCLoP* (Plaku et al. 2010), a motion planning algorithm for systems with dynamics, which synergistically combines high-level discrete planning and low-level motion planning. They demonstrated fast motion planning for ground and flying vehicles, with computation times up to two orders of magnitude faster than other kinodynamic motion planners like RRT (LaValle and Kuffner 2001) and Expansive Space Trees (EST) (Hsu et al. 2001). Phillips et al. introduced *Experience Graphs* (E-Graphs), a fast motion planning algorithm that uses motion plans from its previous planning experience to rapidly generate motions for mundane, similar tasks (Phillips et al. 2012). They successfully demonstrated navigation and manipulation tasks with Willow Garage's PR2 robot (PR2 2009). None of these hybrid motion planning approaches integrate the design of feedback controllers with motion planning for dynamic systems, which is the focus of this paper.

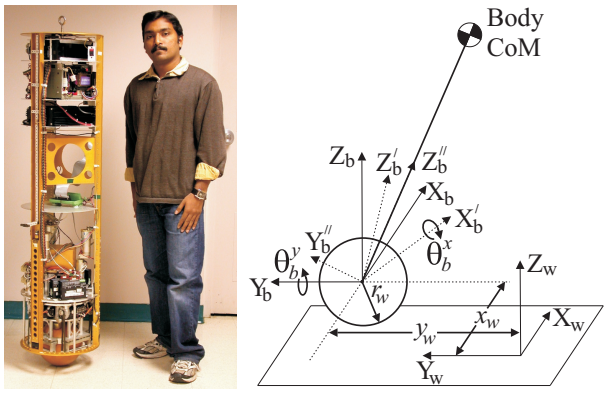
III. Background

This section introduces the ballbot, its 3D dynamic model, its shape variables, and shape-accelerated balancing systems. It also briefly describes the shape trajectory planner and the control architecture that play a key role in the design of motion policies.

A. The Ballbot

The ballbot, shown in Fig. 3(a), is a human-sized dynamically stable mobile robot that balances on a single ball. The ball is actuated using an inverse mouse-ball drive with four active rollers. A pair of opposing rollers drive the ball in each of the two orthogonal motion directions on the floor, and the encoders on the ball motors provide odometry information for the ball.An inertial measurement unit (IMU) provides the body lean angles w.r.t. gravity. A more detailed description of the ballbot's hardware and its control architecture can be found in (Nagarajan et al. 2009c).

For the work presented in this paper, the 3D ballbot is modeled 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, (ii) the floor is flat and level, and (iii) the body does not yaw, $i.e.$, it does not rotate about its vertical axis.



(a) (b)

Fig. 3. (a) The ballbot balancing; (b) 3D ballbot model with its configurations. The ball position w.r.t. the world frame is given by (x_w, y_w) and the body angles w.r.t. the body frame are given by pitch angle θ_b^y (rotation about Y_b) and roll angle θ_b^x (rotation about X_b').

The kinematics of a ball rolling on a plane has been extensively studied using the plate-ball system for manipulation (Agrachev and Sachkov 1999; Bicchi and Sorrentino 1995; Oriolo et al. 2003). The plate-ball system consists of a ball rolling between two flat and level planes, wherein one of the planes is fixed, while the other is used to move the ball. A ball rolling on a plane has five configurations, two configurations for the ball position and three configurations for the ball orientation, and it has been proved to be controllable (Li and Canny 1990; Marigo and Bicchi 2000).

However, in this paper, we are interested only in the position of the ball and not in its orientation. For each orthogonal motion direction on the floor, the kinematic mapping between the plate velocity and linear velocity of the center of the ball is linear, and it is decoupled from the plate velocity in the other orthogonal motion direction (Brockett and Dai 1993).

Figure 3(b) shows the configurations of the 3D ballbot model used in this paper. The origin of the world frame is fixed to the initial position of the center of the ball. Since we have assumed a flat and level floor, the position (x_w, y_w) of the center of the ball matches the position of the ball's contact point on the floor. The ball position (x_w, y_w) can be obtained by integrating the velocity of the center of the ball. The body axis of the robot is given by the line connecting the center of mass (CoM) of the body to the center of the ball. The orientation of the body axis w.r.t. the vertical is given by pitch angle θ_b^y and roll angle θ_b^x as shown in Fig. 3(b). These body angles are directly measured by the IMU.

The ball position (x_w, y_w) w.r.t. the world frame is given by configurations (θ_w^x, θ_w^y) such that $x_w =$

$r_w(\theta_w^x + \theta_b^y)$ and $y_w = r_w(\theta_w^y - \theta_b^x)$, where r_w is the radius of the ball. It is important to note that the configurations (θ_w^x, θ_w^y) are angular configurations that represent the ball position, and they do not represent the orientation of the ball. Therefore, $\theta_w^x, \theta_w^y \in (-\infty, \infty)$. There are two advantages in choosing these coordinates: one is that the ball position configurations (θ_w^x, θ_w^y) directly correspond to the encoder readings on the ball motors, and the other is that this coordinate choice allows one to remove the input coupling between the ball and the body from the equations of motion.

The work presented in this paper deals only with the translation of the ballbot on the floor and does not deal with yaw motions, $i.e.$, rotations about the body axis. However, the ballbot has an independent yaw mechanism that enables rotation about its body axis. The yaw controller presented in (Nagarajan et al. 2009c) can be independently used to achieve desired robot headings. In fact, for all the experimental results presented in Sec. VI, the yaw controller was used to ensure that the body did not yaw while the ballbot was in motion.

B. Position and Shape Variables

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 robot in the world, and the robot dynamics are 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 3D ballbot model, the ball position configurations form its position variables, $i.e.$, $q_x = [\theta_w^x, \theta_w^y]^T \in \mathbb{R}^{2\times1}$, while the body angles form its shape variables, $i.e.$, $q_s = [\theta_b^x, \theta_b^y]^T \in \mathbb{R}^{2\times1}$.

The ballbot is an underactuated system (Spong 1998), $i.e.$, the number of independent control inputs is less than the number of degrees of freedom. The ball is actuated while the body is not. Hence, the ballbot's position variables q_x are actuated, while its shape variables q_s are unactuated.

C. Dynamic Constraints

The Euler-Lagrange equations of motion of the 3D ballbot model can be written in matrix form as follows:

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

where, $q = [q_x, q_s]^T \in \mathbb{R}^{4\times1}$, $M(q) \in \mathbb{R}^{4\times4}$ is the mass/inertia matrix, $C(q,\dot{q}) \in \mathbb{R}^{4\times4}$ is the Coriolis and centrifugal matrix, $G(q) \in \mathbb{R}^{4\times1}$ is the vector of gravitational forces and $\tau \in \mathbb{R}^{2\times1}$ is the vector of generalized forces.

The last two equations of motion in Eq. 1 corresponding to the unactuated shape variables, $i.e.$, the body lean angles, are of the form:

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

These second-order differential equations are not integrable, and hence form second-order nonholonomic constraints (or) dynamic constraints (Oriolo and Nakamura 1991). However, they are numerically integrable. It can be seen from Eq. 2 that the dynamic constraint equations are independent of the position and velocity of the position variables (q_x, $\dot{q}_x$). They map the position, velocity and acceleration in shape space (q_s, $\dot{q}_s$, $\ddot{q}_s$) to the acceleration in position space $\ddot{q}_x$, and vice-versa. The dynamic constraint equations for balancing mobile robots like the ballbot have a special structure wherein any non-zero shape configuration results in acceleration in position space as shown in our previous work (Nagarajan 2010; Nagarajan et al. 2012b). Hence, these systems are called *shape-accelerated balancing systems*.

D. Planning in Shape Space

This section briefly describes a shape trajectory planner that exploits the natural dynamics of shape-accelerated balancing systems like the ballbot to achieve desired motions in position space. Navigation tasks are generally posed as desired motions in position space, without any specifications on shape space motions. However, shape space motions cannot be ignored for highly dynamic systems like the ballbot because there is a strong coupling between their shape and position dynamics. Since the shape dynamics dominates the system dynamics, any desired motion in position space can be successfully achieved only if an appropriate motion in shape space is planned and tracked.

In our previous work (Nagarajan 2010; Nagarajan et al. 2012b), a trajectory planner that uses only the dynamic constraint equations (Eq. 2) to plan shape trajectories, which when tracked will result in approximate tracking of desired position trajectories was presented. This shape trajectory planner exploits the properties of shape-accelerated balancing systems, and plans motions in shape space that are proportional to desired accelerations in position space. This section presents a brief description of the shape trajectory planner, while a more detailed presentation can be found in (Nagarajan 2010; Nagarajan et al. 2012b).

From the dynamic constraint equations in Eq. 2, the acceleration in position space can be written as a nonlinear function of shape variables and their derivatives given by

$$\ddot{q}_x = f(q_s, \dot{q}_s, \ddot{q}_s), \qquad (3)$$

subject to certain invertibility conditions, which hold for shape-accelerated balancing systems as shown in (Nagarajan 2010). When a shape-accelerated system sticks to a constant shape configuration, $i.e.$, $\dot{q}_s = 0$ and $\ddot{q}_s = 0$, it can be shown that the system achieves constant acceleration in position space given by

$$\ddot{q}_x = f'(q_s), \tag{4}$$

where $f'(q_s) = f(q_s, \dot{q}_s = \mathbf{0}, \ddot{q}_s = \mathbf{0})$. The Jacobian linearization of the nonlinear map $f'(q_s)$ around the origin shows that an invertible linear map K^0 exists such that the constant shape configuration that produces a constant acceleration in position space can be given by

$$q_s = K^0 \ddot{q}_x, \tag{5}$$

where $K^0 \in \mathbb{R}^{2\times 2}$ for the 3D ballbot model presented in Sec. III-A. But such an invertible map does not exist for any arbitrary acceleration trajectory in position space. However, the constant shape configuration case shows that the shape configuration and the acceleration in position space are linearly proportional to each other in the neighborhood of the origin, and this insight is used to design a trajectory planner that plans shape trajectories, which when tracked will result in approximate tracking of desired position trajectories. More details can be found in (Nagarajan 2010).

Given a desired position space acceleration trajectory $\ddot{q}_x^d(t)$, the planned shape trajectory $q_s^p(t)$ is chosen to be

$$q_s^p(t) = K \ddot{q}_x^d(t), \tag{6}$$

where K is a constant linear matrix, and $K \in \mathbb{R}^{2\times 2}$ for the 3D ballbot model presented in Sec. III-A. The acceleration trajectory in position space $\ddot{q}_x^p(t)$ that results from tracking the planned shape trajectory $q_s^p(t)$ is obtained from Eq. 3 as follows:

$$\ddot{q}_x^p(t) = f(K\ddot{q}_x^d(t), K\dddot{q}_x^d(t), K\ddddot{q}_x^d(t)). \tag{7}$$

The shape trajectory planning can be formulated as an optimization problem with the objective of finding elements of the constant linear matrix K such that the resulting acceleration trajectory in position space $\ddot{q}_x^p(t)$ best approximates the desired acceleration trajectory in position space $\ddot{q}_x^d(t)$, $i.e.$, $\ddot{q}_x^p(t) \approx \ddot{q}_x^d(t)$. This optimization can be solved using nonlinear least-squares solvers like Nelder-Mead simplex (Nelder and Mead 1964) and Levenberg-Marquardt (Levenberg 1944) algorithms. For tracking constant desired accelerations in position space, the matrix K^0 (Eq. 5) will be the optimal solution for the matrix K, whereas for any arbitrary desired acceleration trajectory, the K^0 forms a good initial guess for K.

A navigation task will generally involve tracking a desired position trajectory and not an acceleration trajectory. In order to use the shape trajectory planner, the desired position trajectory $q_x^d(t)$ must be at least of class C^2, $i.e.$, $\dot{q}_x^d(t)$ and $\ddot{q}_x^d(t)$ exist and are continuous, so that the planned shape trajectory $q_s^p(t) = K\ddot{q}_x^d(t)$ exists and is continuous. However, the shape trajectory planner prefers the desired position trajectory $q_x^d(t)$ to be of class C^4, $i.e.$, $\dddot{q}_x^d(t)$ and $\ddddot{q}_x^d(t)$ exist and are continuous, so that the planned velocity and acceleration trajectories in shape space $(\dot{q}_s^p(t), \ddot{q}_s^p(t))$ that depend on them exist and are continuous.

Moreover, tracking a desired position trajectory $q_x^d(t)$ is the same as tracking its corresponding acceleration trajectory $\ddot{q}_x^d(t)$ only if the initial conditions are met. The initial conditions are rarely met on real robots, and in addition, there are unmodeled dynamics, nonlinear friction effects and noise. In order to solve these issues, a feedback position tracking controller that feeds back position variables is introduced. The feedback position tracking controller outputs a compensation shape trajectory that compensates for the deviation of the position trajectory from the desired trajectory. The planned and compensation shape trajectories are summed to form the desired shape trajectory, which is tracked by the balancing controller as shown in Fig. 4. The successful tracking of a variety of different fast, dynamic motions in position space using the shape trajectory planner and the control architecture in Fig. 4 have been experimentally demonstrated on the ballbot in (Nagarajan et al. 2012b).

Recently, a pair of 2-DOF arms were added to the ballbot (Nagarajan et al. 2012b). The arm angles also form shape variables, and a variant of the shape trajectory planner presented here has been developed to plan motions in high-dimensional shape space in order to achieve desired motions in low-dimensional position space. A detailed presentation of the shape trajectory

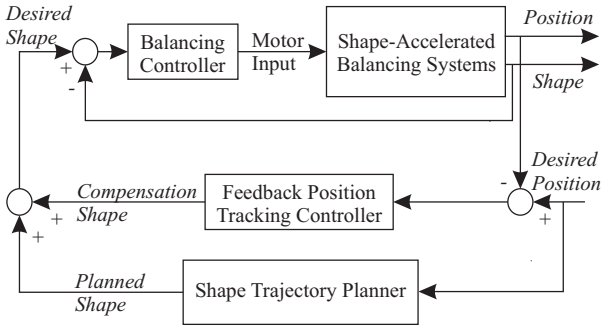


Fig. 4. The control architecture.

planner that handles motions in high-dimensional shape space is currently under review (Nagarajan and Hollis 2013). However, the work presented in this paper deals only with the ballbot without arms. The shape trajectory planner and the feedback position tracking controller shown in Fig. 4 form the control architecture for motion policies that will be presented in Sec. IV.

IV. Motion Policy Design

This section describes an offline procedure of designing a palette of control policies called *motion policies* for shape-accelerated balancing mobile robots like the ballbot using the shape trajectory planner and the control architecture shown in Fig. 4. A motion policy Φ_i consists of a reference state trajectory called a *motion primitive* $\sigma_i(t)$, a time-varying feedback trajectory tracking control law $\phi_i(t)$, and a time-varying domain $D_i(t)$ that is verified to be asymptotically convergent. All these components of a motion policy are described below.

A. Motion Primitives

Motion primitives $\sigma(t)$ are elementary, feasible state trajectories that produce motions in small domains of the position space, and they can be combined sequentially to produce more complicated trajectories. Motion primitives are feasible state trajectories, and hence by definition satisfy the constraints on the dynamics of the system. In this work, the motion primitives are defined such that they result in graceful motion, $i.e.$, their position, velocity and acceleration trajectories are continuous and bounded. Moreover, any valid sequential composition of motion primitives must also result in an overall graceful motion.

This paper follows (Frazzoli et al. 2005) to define two classes of motion primitives:

(i) *Trim primitives*: 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, the trim primitives are restricted to constant position or velocity trajectories in position space, which implies that they have zero acceleration in position space and also zero shape configurations.

(ii) *Maneuvers*: 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 acceleration in position space, which implies that they achieve non-zero shape configurations. However, maneuvers start and end at trim conditions, which correspond to zero shape configurations. Maneuvers can be any arbitrary feasible state trajectories as long as they satisfy the trim conditions.

Here, the zero shape configurations correspond to any set of shape configurations that produce zero acceleration in position space. The motion primitives in (Frazzoli et al. 2005) consisted of both feasible state and control trajectories, whereas the motion primitives in this work include only feasible state trajectories. Motion primitives can represent feasible state space motions that produce different motions in position space like straight line, turning, circular, S-curve or figure-8 motions.

A collection of motion primitives with a distance parameter d is defined as a *motion primitive set* $\Sigma(d)$, wherein each motion primitive produces a net Δx and Δy motion in position space such that Δx and Δy are integral multiples of the distance parameter d. Figure 5 presents the position space motions of a sample of motion primitives for the 3D ballbot model from a motion primitive set $\Sigma(d)$ with $d = 0.5$ m. It is important to note that the motion primitives include both position and shape trajectories, and are not restricted to just position trajectories. Therefore, for the 3D ballbot model, the motion primitives represent feasible motions in 8D state space, while Fig. 5 shows only their corresponding 2D position space motions.

The number of motion primitives in a motion primitive set $\Sigma(d)$ is dependent on the "richness" of motions one desires to achieve with the robot. For example, the simplest and minimalist motion primitive set will consist of just two motion primitives: a stationary (constant position) trim primitive and a rest-to-rest maneuver that

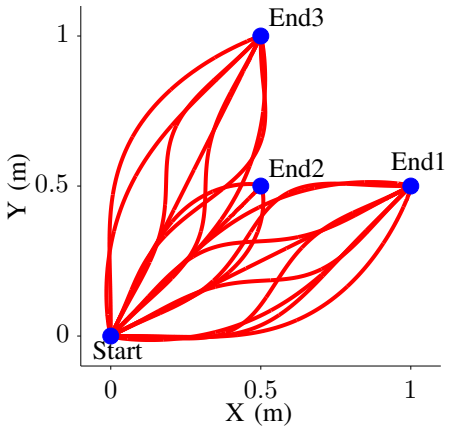


Fig. 5. Position space motions of a sample of motion primitives for the 3D ballbot model from a motion primitive set with $d = 0.5$ m.

moves the robot a distance d. A balancing mobile robot can indeed use these two motion primitives to stay in place and move around. But any motion longer than distance d will consist of a concatenation of several rest-to-rest motions of distance d each, $i.e.$, the robot stops multiple times before coming to rest at the goal. This minimalist motion primitive set is sufficient to achieve a stationary goal, but it is not necessarily desirable. Therefore, the number and type of motion primitives in a motion primitive set $\Sigma(d)$ depends on the motions desired from the robot, and the larger and more varied the motion primitive set is, "richer" are the motions achieved using the motion primitive set.

The position space motions of the motion primitives shown in Fig. 5 may strike a strong resemblance to state lattices (Pivtoraiko and Kelly 2005; Pivtoraiko et al. 2009) and path sets (Knepper and Mason 2008, 2009) used by the motion planners in unmanned ground vehicles. State lattices and path sets are defined as reference motions in only the position space, and the motion planners plan in the space of these reference position space motions to achieve desired navigation tasks. However, the motion primitives presented in this paper are defined as feasible state space motions that include both position and shape space motions, and the motion planner plans in the space of motion policies, which are controllers designed around these motion primitives as will be described in Sec. IV-B. State lattices (Pivtoraiko and Kelly 2005; Pivtoraiko et al. 2009) and path sets (Knepper and Mason 2008, 2009), however, can be used to determine the reference position space motions for generating the motion primitive sets presented here.

The step-by-step procedure used in this work to design a motion primitive set for the 3D ballbot model, including the one shown in Fig. 5 is as follows:

(i) The trim (or) steady-state conditions were defined. The trim conditions used in this work were limited to constant position and constant velocity conditions in position space.

(ii) A number of unique, desired position space motions with a distance parameter d were chosen in the first quadrant of the XY$-$plane. These desired motions included constant position and constant velocity trajectories given by the trim conditions in step (i), and several other trajectories with varying position and velocity in position space whose start and end conditions satisfied the trim conditions with zero acceleration in position space. The desired trajectories in position space were defined as nonic polynomials, $i.e.$, polynomials with degree nine, satisfying the desired boundary trim conditions. The desired trajectories were chosen such that they satisfy all characteristics of desired position trajectories presented in Sec. III-D, and also satisfy all requirements of a graceful motion, $i.e.$, the position, velocity and acceleration trajectories are continuous and bounded with low jerk.

(iii) The shape trajectory planner presented in Sec. III-D was used to obtain the feasible position and shape trajectories that approximately achieve the desired position space motions in step (ii). The feasible state trajectories that produce constant position and constant velocity motions in position space are the trim primitives, while the rest that satisfy the boundary trim conditions are the maneuvers. Together, they form the motion primitives in the motion primitive set $\Sigma(d)$. It is important to note that the trim conditions for the trim primitives and the maneuvers match, and hence these elementary state trajectories can be composed to produce more complicated motions.

As described above, the motion primitives are designed to have matching trim conditions, which enable them to be composable. The necessary condition for composability of motion primitives in a motion primitive set $\Sigma(d)$ is that for every trim condition in $\Sigma(d)$, there must exist a corresponding trim primitive, at least one maneuver that starts with that trim condition and at least one maneuver that ends with that trim condition. The sufficient condition for strong composability of motion primitives is that in addition to the necessary condition for composability, for every ordered pair of trim conditions, there must exist at least one maneuver that transitions between them.

Each motion primitive in a motion primitive set can be rotated and translated in the position space to achieve a variety of different motions, and this process of setting the position and orientation (heading) of a motion primitive is called *instantiation*. This is possible because the dynamics of mobile robots are invariant to transformations of their position variables. It is important to note that the entire motion primitive along with its start and end trim conditions are invariant to rotations and translations of the position variables.

In this work, the motion primitives in the motion primitive set $\Sigma(d)$ are designed to be strongly composable. Hence, for each motion primitive $\sigma_1(t) \in \Sigma(d)$, there exists at least one motion primitive $\sigma_2(t) \in \Sigma(d)$ such that its instantiation is gracefully composable with $\sigma_1(t)$. Moreover, for every ordered pair of motion primitives $\big(\sigma_1(t), \sigma_2(t)\big)$, there exists at least one motion primitive $\sigma_3(t) \in \Sigma(d)$ such that an instantiation of $\sigma_3(t)$ is

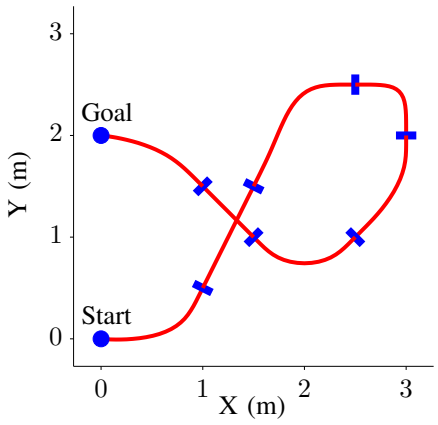


Fig. 6. Position space motion of an example motion plan using instantiated motion primitives from Fig. 5. The shaded circles represent constant position trim conditions, while the bars represent constant velocity trim conditions.

gracefully composable with $\sigma_2(t)$ and an instantiation of $\sigma_1(t)$ is gracefully composable with $\sigma_3(t)$, *i.e.*, there exists instantiations such that $\sigma_1(t) \rightarrow \sigma_3(t) \rightarrow \sigma_2(t)$ form a valid sequence of gracefully composable motion primitives.

A motion primitive $\sigma_1(t)$ is gracefully composable with another motion primitive $\sigma_2(t)$ if and only if $\sigma_1(t_{f_1}) = \sigma_2(0)$ and $\dot{\sigma}_1(t_{f_1}) = \dot{\sigma}_2(0)$, *i.e.*, the final position, velocity and acceleration of $\sigma_1(t)$ matches the initial position, velocity and acceleration of $\sigma_2(t)$. The motion primitives in the motion primitive set $\Sigma(d)$ with matching trim conditions are designed such that they are gracefully composable. Figure 6 shows an example motion in position space resulting from a graceful composition of appropriately instantiated motion primitives from a motion primitive set $\Sigma(d)$ with $d = 0.5$ m.

It is important to note that although the motion primitives can be rotated and translated in the position space, they cannot be dilated, *i.e.*, scaled to achieve shorter or longer motions in position space. This is because the dilation of feasible position and shape trajectories does not necessarily result in feasible state trajectories, *i.e.*, the resulting state trajectories do not necessarily satisfy the dynamic constraint equations. However, one can dilate the desired position space motions in step (*ii*) of the motion primitive design process and find the feasible shape and position trajectories that achieve them using the shape trajectory planner as described in step (*iii*).

B. Motion Policies

The motion primitives presented in Sec. IV-A are feasible state trajectories that satisfy the dynamic constraints of the system, and also produce graceful motion by construction. Any dynamic system requires control effort to track these feasible state trajectories. In (Frazzoli et al. 2005), open-loop control trajectories were used as part of the motion primitives. But in a real world, especially for robots operating in human environments, where there are environment uncertainties and perturbations, one needs to use closed-loop control. This section presents *motion policies* that contain motion primitives and feedback controllers that stabilize them.

A 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 $D_i(t)$, all defined for time $t \in [0, t_{f_i}]$. Since the entire motion policy execution is time parameterized, each motion policy Φ_i also contains a timer T_i that starts at zero and ticks till the duration t_{f_i} of the motion primitive $\sigma_i(t)$. A motion policy that consists of a trim primitive is called a *trim policy*, while a motion policy that consists of a maneuver is called a *maneuver policy*. Since all motion primitives in the motion primitive set $\Sigma(d)$ are feasible state trajectories, they can be stabilized using locally linear feedback controllers, and hence a motion policy Φ_i exists for each motion primitive $\sigma_i(t) \in \Sigma(d)$. Given a motion primitive set $\Sigma(d)$, its corresponding motion policy palette $\Pi(\Sigma)$ is constructed by defining a time-varying linear feedback control law $\phi_i(t)$ and its time-varying domain $D_i(t)$ for every motion primitive $\sigma_i(t) \in \Sigma(d)$. Therefore, there is a one-to-one correspondence between the motion primitive $\sigma_i(t) \in \Sigma(d)$ and the motion policy $\Phi_i \in \Pi(\Sigma)$. The approach used in this work to design the feedback control law and its domain are discussed below.

For every motion policy $\Phi_i \in \Pi(\Sigma)$, the feedback control law $\phi_i(t)$ stabilizes its constituent motion primitive $\sigma_i(t) \in \Sigma(d)$. In general, standard approaches like linear quadratic regulators (LQR) can be used to stabilize the motion primitives. However, in this work, the motion policies defined for shape-accelerated balancing mobile robots like the ballbot use the control architecture shown in Fig. 4, which exploits the strong coupling between the dynamics of position and shape variables to achieve desired motions in position space. This control architecture of using an inner-loop balancing controller that stabilizes the shape dynamics and an outer-loop position tracking controller that achieves desired position space motions has been experimentally verified to be robust as shown in (Nagarajan et al. 2009a,b,c). The ability of this control architecture to successfully track desired motions in position space has also been experimentally verified on the ballbot as shown in (Nagarajan et al. 2012a,b)

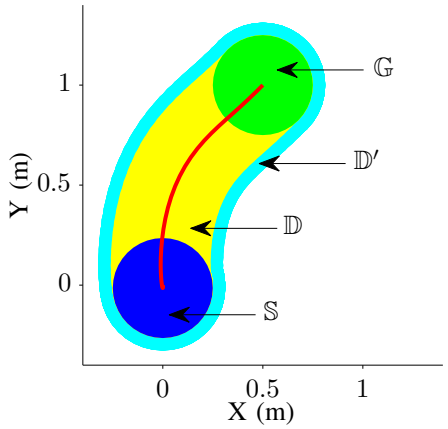


Fig. 7. XY projection of the domain of a sample motion policy designed for the 3D ballbot model.

and in Sec. VI of this paper.

In general, the time-varying domain $D_i(t)$ of the feedback control law $\phi_i(t)$ is defined to be the largest invariant domain along the feasible state trajectory $\sigma_i(t)$ in the state space of the system. Lyapunov functions can be used to determine these invariant domains, and recent advancements in algebraic verification tools like sums-of-squares (SOS) tools (Tedrake et al. 2010; Tobenkin et al. 2011) make such computations a possibility. However, in this work, the motion policy domain definitions are restricted to 4D position state space, *i.e.*, $(x, y, \dot{x}, \dot{y})$, since the control architecture of the motion policy shown in Fig. 4 achieves motions in position space by controlling shape space motions.

The time-varying domains $D(t)$ are defined as 4D hyper-ellipsoids centered around the time-varying desired position states of the motion primitives $\sigma(t)$. Each time-varying domain $D(t)$ has a start domain $\mathbb{S} = D(0)$ and a goal domain $\mathbb{G} = D(t_f)$. Each domain $D(t)$ is verified to be asymptotically convergent, similar to the ones defined in (Nagarajan et al. 2010). This implies that 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 position 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 a motion policy are given by $\mathbb{D} = \bigcup_{t=0}^{t_f} D(t)$ and $\mathbb{D}' = \bigcup_{t=0}^{t_f} D'(t)$. The domain $\mathbb{D}'$ is used for checking collisions with obstacles in the environment and hence, their geometric definitions make it easier to verify the validity of their motion policies. Figure 7 shows the XY projection of a sample motion policy design for the ballbot.

In this work, the design and verification of the asymptotically convergent motion policy domains were done using simulation. For a motion policy Φ_i, the radii of the 4D hyper-ellipsoids in the domain $D_i(t)$ were chosen to be constant along the entire motion primitive $\sigma_i(t)$, and the outer domain $D'_i(t)$ was designed as $D'_i(t) = kD_i(t)$, where $k > 1$ is the scale factor. For a choice of radii values, the 3D ballbot model along with the control architecture in Fig. 4 was simulated from finitely many states in the start domain $\mathbb{S}_i$. The largest radii values for which the closed-loop position state trajectory from each of these start states remained inside the outer domain $D'_i(t)\ \forall t \in [0, t_f]$ and also reached the goal domain $\mathbb{G}_i$ at $t = t_f$ were determined by trial and error. These radii values defined the motion policy domain $D_i(t)$. It is important to note that the motion policy domain obtained using the above described procedure is a conservative estimate of the largest invariant domain of the motion policy. However, in this work, it is not essential to have an accurate estimate of the largest invariant domain, and a conservative estimate that is verified to be asymptotically convergent satisfies the requirements of the integrated motion planning and control framework presented in this paper.

The above design procedure also verifies by simulation that all position states in the start domain $\mathbb{S}_i$ reach the goal domain $\mathbb{G}_i$ and remain within $D'_i(t)\ \forall t \in [0, t_f]$ under the action of the motion policy Φ_i. Additionally, the resulting closed-loop shape trajectory was also verified to remain within the domain of the balancing controller that tracks it. Several system identification experiments were conducted on the ballbot to estimate the system parameters such that the dynamics of the model better match the real robot dynamics (Nagarajan et al. 2009c). When a motion policy Φ_i is deployed on a map of the environment, the verification guarantees that the resulting closed-loop motion of the system in position state space will remain within its domain $\mathbb{D}'_i$. Hence, if the domain $\mathbb{D}'_i$ is collision-free, then the motion policy Φ_i is guaranteed to produce a collision-free closed-loop motion of the system.

The process of deploying a motion policy on a map is called instantiation, just like in the case of a motion primitive. A motion policy instantiation involves the instantiation of its motion primitive and its feedback control law. The condition for two motion primitives to be gracefully composable was presented in Sec. IV-A. Here, a time-varying feedback control law $\phi_1(t)$ is defined to be gracefully composable with another time-varying feedback control law $\phi_2(t)$ if $\phi_1(t_{f_1}) = \phi_2(0)$, *i.e.*, the final control law of ϕ_1 matches the initial control law of ϕ_2. These conditions will be used in Sec. IV-C to define the graceful composition of motion policies.

International Journal of Robotics Research (IJRR), Special Issue: Motion Planning for Physical Robots
Volume 32, Issue 9 – 10, pp. 1005 – 1029, September 2013

C. Gracefully Prepares Relationship

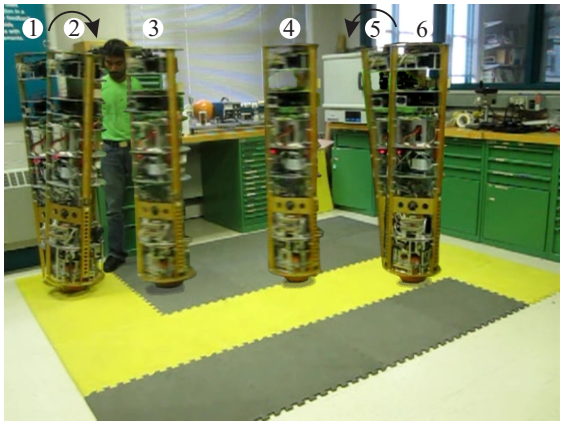
This section introduces the notion of *gracefully prepares relationship* that guarantees graceful sequential composition of two motion policies. A motion policy Φ_1 is said to *gracefully prepare* Φ_2, denoted by $\Phi_1 \succeq_G \Phi_2$, if and only if all the conditions listed below are satisfied.

- (i) The goal domain of the motion policy Φ_1 is contained in the start domain of the motion policy 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 $\dot{\sigma}_1(t_{f_1}) = \dot{\sigma}_2(0)$, which ensures that the overall reference position, velocity and acceleration trajectories are continuous. A motion primitive $\sigma(t)$ is a state trajectory, which includes position and velocity trajectories. Therefore, its derivative $\dot{\sigma}(t)$ includes velocity and acceleration trajectories.
- (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.*, $\phi_1(t_{f_1}) = \phi_2(0)$, which ensures that the overall control trajectory is continuous.

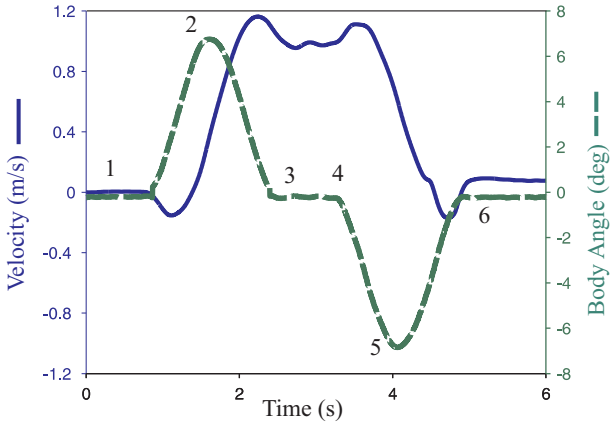
The first condition satisfies the prepares relationship (Burridge et al. 1999), while the next two conditions reduce it to a gracefully prepares relationship. Hence, a gracefully prepares relationship is by definition a prepares relationship, *i.e.*, $\Phi_1 \succeq_G \Phi_2 \Rightarrow \Phi_1 \succeq \Phi_2$, but not vice-versa. Since the reference position, velocity and acceleration trajectories along with the control trajectory are continuous, the resulting closed-loop motion of the system under the action of a sequence of gracefully composable motion policies is graceful. Moreover, the stability and asymptotic convergence of the individual motion policies in the motion policy palette guarantee the stability and asymptotic convergence of any valid sequence of instantiated motion policies given by the gracefully prepares relationship. It is important to note that the motion primitives in the motion primitive set $\Sigma(d)$ are designed to be strongly composable, and for each pair of motion primitives $\sigma_1(t), \sigma_2(t) \in \Sigma(d)$ where $\sigma_1(t)$ is gracefully composable with $\sigma_2(t)$, their corresponding motion policies $\Phi_1, \Phi_2 \in \Pi(\Sigma)$ are designed such that Φ_1 gracefully prepares Φ_2.

Figures 8 and 9 show the experimental results of the ballbot achieving fast, graceful motions while switching between gracefully composable motion policies. The ballbot achieves a fast, graceful straight line motion in Fig. 8 that is composed of three different motions. The ballbot achieved a peak velocity of 1.16 m/s, a peak acceleration of 1.1 m/s^2, and a maximum lean of 6.75° in the plane of motion. In Fig. 9, the ballbot makes three sharp left turns by gracefully switching between nine gracefully composable motion policies. The videos of the ballbot performing both these motions can be found in Extension 1.

The sequence of gracefully composable motion policies used for the experimental results shown in Fig. 8 and Fig. 9 were manually chosen, and the switching operation was also performed manually. Section V will present automated approaches towards autonomous planning in the space of gracefully composable motion policies to achieve desired navigation tasks. It will also present a hybrid control architecture that successfully executes the motion plan.



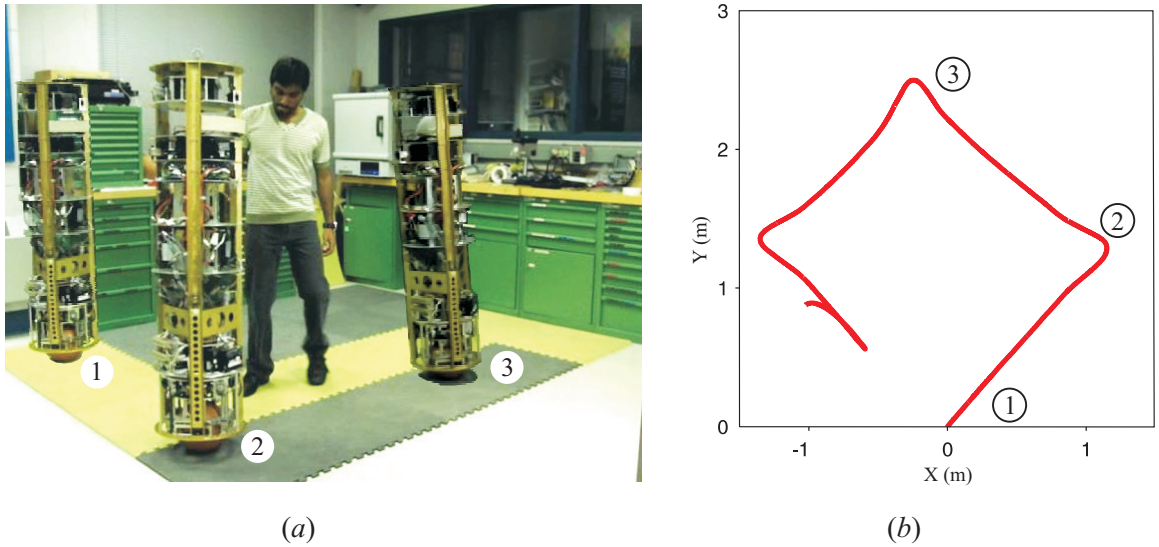
(*a*)



(*b*)

Fig. 8. Fast straight line motion: (*a*) composite frames from a video, (*b*) plot of body angle and velocity $vs.$ time in the plane of motion.

International Journal of Robotics Research (IJRR), Special Issue: Motion Planning for Physical Robots
Volume 32, Issue 9 – 10, pp. 1005 – 1029, September 2013



(*a*) (*b*)

Fig. 9. Sharp turning motion: (*a*) composite frames from a video, (*b*) plot of the motion tracked on the floor.

V. Integrated Motion Planning and Control

This section presents the online procedures that run on-board the robot to achieve graceful navigation. An automatic motion policy instantiation procedure is presented, which uses the motion policy palette to generate a library of instantiated motion policies whose domains fill an obstacle-free map of the environment. A motion planner that plans in the space of these gracefully composable motion policies, and a hybrid control architecture that executes a generated motion plan are presented. A dynamic replanning algorithm that actively replans in the space of gracefully composable motion policies to avoid dynamic obstacles is also presented.

A. Automatic Instantiation of Motion Policies

Here, an automatic instantiation procedure that uniformly distributes motion policies both in position and in orientation (heading) on a 2D map of the environment is presented. Given a motion policy palette $\Pi(\Sigma)$, the 2D map $\mathbb{M}$ is defined as a grid that uniformly discretizes the environment with instantiation points (x, y) that are separated by a distance d along both X and Y directions, where d is the distance parameter of the motion primitive set $\Sigma(d)$. Every motion policy $\Phi_i \in \Pi(\Sigma)$ is instantiated at these instantiation points in different orientations (headings).

The uniform discretization in orientation (heading) space depends on the position space motions produced by motion policies in the motion policy palette. The motion policies presented here produce motions in the first quadrant of the position space as shown in Fig. 5, and hence the orientation spacing is chosen to be 90° so that their instantiations can cover all four quadrants in position space. It is important to note that the map $\mathbb{M}$ is used only for the instantiation of motion policies and is not used for localization. This is because the instantiation map $\mathbb{M}$ has coarser grids, while an occupancy grid map for localization needs to be finer. Finer occupancy grids are used for localization of the ballbot in all experimental results presented in this paper.

As described in Sec. IV-A, all motion primitives $\sigma_i(t) \in \Sigma(d)$ are feasible state trajectories that produce position space motions wherein their Δx and Δy displacements are integral multiples of the distance parameter d. Therefore, the motion primitive set $\Sigma(d)$ is a collection of feasible state trajectories that move between the grid points on the map $\mathbb{M}$ that is discretized by d. Moreover, since the motion primitives in the motion primitive set $\Sigma(d)$ are designed to be strongly composable and their corresponding motion policies in $\Pi(\Sigma)$ are designed to be gracefully composable, instantiated motion policies with matching trim conditions can be sequentially composed to gracefully navigate between the grid points on the map.

The automatic instantiation procedure generates a motion policy library $\mathbb{L}(\Pi, \mathbb{M})$, which is a collection of valid instantiations of motion policies from the motion policy palette $\Pi(\Sigma)$ on a map $\mathbb{M}$. A subset of valid instantiated motion policies from a motion policy library on a map with static obstacles is shown in Fig. 10. An instantiated motion policy Φ_i is considered valid if and

only if the following conditions are satisfied:

(i) The motion policy domains $\mathbb{D}_i$ and $\mathbb{D}'_i$ must be obstacle-free in order to guarantee that the closed-loop motion resulting from the execution of the motion policy will remain collision-free.

(ii) For motion policies that start at non-stationary trim conditions, the adjacent cell in the direction opposite to the motion must be obstacle-free. Such motion policies, if included in the motion policy library, will become orphan motion policies as other valid motion policies cannot prepare them.

(iii) For motion policies that end at non-stationary trim conditions, the adjacent cell in the direction of the motion must be obstacle-free. Such motion policies, if included in the motion policy library, will result in collisions as they cannot prepare valid motion policies.

The percentage of the bounded position state space covered by the start domains of the motion policies in the motion policy library represents the coverage of the motion policy library. The coverage percentage is calculated by uniformly sampling the bounded position state space and verifying its existence in the union of the start domains of the motion policies in the motion policy library. If the desired coverage is not achieved then the grid spacing for the instantiation points is halved, and the automatic instantiation procedure is repeated until either the desired coverage or the maximum number of such iterations is achieved. If 100% coverage is not achieved, then it indicates that there are certain position states that cannot be handled by the motion policies in the motion policy library.

It is important to note that the motion policy domains are verified in simulation and hence, under the action of the motion policies in the motion policy library, the robot's state is verified to remain within these domains. Therefore, the position states not covered by the union of the domains of the motion policies in the motion policy library are not reached under normal circumstances. However, they can be reached when the robot is subjected to large disturbances. A backup emergency controller is used to handle such cases. In this work, the ballbot switches to a simple balancing mode when such a situation is encountered. While in balancing mode, the robot slows down and comes to rest, and it can potentially enter the domain of one of the motion policies in the motion policy library and can still get to the goal. An experimental demonstration of this behavior is presented in Sec. VI-D. Currently, our group is also exploring real-time trajectory planning algorithms (Shomin and Hollis 2013) that will enable the ballbot to recover from such cases and enter nearby motion policy domains in the motion policy library.

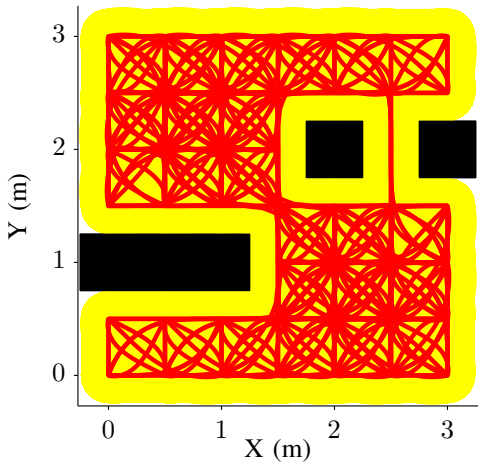


Fig. 10. A subset of motion policies from a motion policy library $\mathbb{L}(\Pi, \mathbb{M})$, instantiated from the motion policy palette $\Pi(\Sigma)$ with the motion primitive set $\Sigma(d)$ shown in Fig. 5. The lines represent position space motions of the motion primitives, while the shaded regions show 2D projections of the 4D motion policy domains, including their outer domains. The three obstacles are shown in black.

In this work, the grid spacing of the map $\mathbb{M}$ was set to match the distance parameter d of the motion primitive set $\Sigma(d)$, which was chosen such that the desired coverage was achieved. The position space radii of the motion policy domains play the primary role in determining the coverage, and hence they are used to determine bounds on the choice of d. The number and variety of motion primitives in the motion primitive set $\Sigma(d)$ also affects the coverage, but is relatively insignificant. However, a larger and more varied motion primitive set allows a richer collection of graceful motions to be achieved.

The automatic instantiation procedure presented in this section uniformly instantiates the motion policies on a given map. This uniformly dense instantiation provides "rich" motions for the robot at all points on the map. However, this is not a necessity. One can also use variable density instantiation procedures that sparsely instantiates motion policies in free, open spaces and densely instantiates motion policies in cluttered, narrow spaces. However, good coverage of the map is essential in order to successfully handle large disturbances.

B. Planning in Motion Policy Space

Given a map $\mathbb{M}$ and a motion policy palette $\Pi(\Sigma)$, the automatic instantiation procedure presented in Sec. V-A generates a motion policy library $\mathbb{L}(\Pi, \mathbb{M})$. The gracefully prepares relationship between every pair of motion

International Journal of Robotics Research (IJRR), Special Issue: Motion Planning for Physical Robots
Volume 32, Issue 9 – 10, pp. 1005 – 1029, September 2013

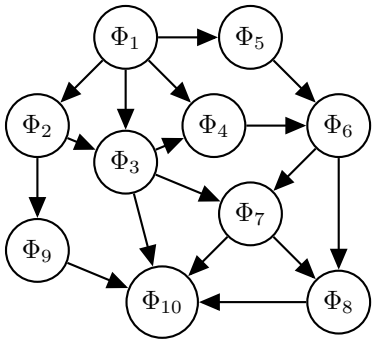


Fig. 11. An example gracefully prepares graph.

policies (Φ_i,Φ_j) in the motion policy library $\mathbb{L}(\Pi, \mathbb{M})$ is verified using the conditions presented in Sec. IV-C, and a directed graph called the *gracefully prepares graph* $\Omega(\mathbb{L})$ is generated, an example of which is shown in Fig. 11. Each node in $\Omega(\mathbb{L})$ represents a valid instantiated motion policy $\Phi_i \in \mathbb{L}$, and each directed edge from Φ_i to Φ_j represents the gracefully prepares relationship, $i.e.$, $\Phi_i \succeq_G \Phi_j$. The gracefully prepares graph is similar to the prepares graph presented in (Conner et al. 2006), but differs from the fact that the edges represent the gracefully prepares relationship and not just the prepares relationship as explained in Sec. IV-C. Unlike in the prepares graph, the switching between motion policies in the gracefully prepares graph is guaranteed to result in overall graceful motion. Therefore, the gracefully prepares graph $\Omega(\mathbb{L})$ contains all possible graceful motions that the robot can achieve on the map $\mathbb{M}$ using motion policies in the motion policy library $\mathbb{L}(\Pi, \mathbb{M})$.

If the gracefully prepares graph is strongly connected, $i.e.$, every motion policy node has a path to every other motion policy node in the directed graph (Cormen et al. 2001), then its controllable subspace is given by the union of the motion primitives of the motion policies in the library. In general, given a gracefully prepares graph, its largest controllable subspace is given by the union of the motion primitives of the motion policies in its largest strongly connected subgraph.

This work assumes that any navigation task can be formulated as a motion between trim motion policies, $i.e.$, motion policies with trim motion primitives, and hence any navigation task can be formulated as a graph search problem on the gracefully prepares graph. This assumption is valid since any navigation task can be formulated as either a point-point motion or a surveillance motion or a combination of the two. A point-point motion can be formulated as a motion between trim motion policies that have constant position trajectories as trim primitives, while a surveillance motion can be formulated as a motion between trim motion policies that have constant velocity trajectories as trim primitives.

Algorithm 1: Single-Goal Optimal Motion Policy Tree using Dijkstra's Algorithm

input : Gracefully Prepares Graph Ω
Goal Motion Policy Node G
output: Optimal Motion Policy Tree Γ

begin
  $\Gamma \leftarrow \emptyset$
  foreach $i \in$ Node(Ω) **do**
    Cost2Goal(i) $\leftarrow \infty$
    Next2Goal(i) $\leftarrow \emptyset$
    $\Gamma \leftarrow \Gamma \cup \big(i$, Cost2Goal($i$), Next2Goal($i$)$\big)$
  end
  Cost2Goal(G) $\leftarrow 0$
  $Q \leftarrow \Gamma$
  while $Q \neq \emptyset$ **do**
    $j \leftarrow$ MinCostNode(Q)
    if Cost2Goal(j) = ∞ **then**
      break
    end
    $Q \leftarrow Q - \{j\}$
    foreach $k \in$ Parent(j) **do**
      c $\leftarrow$ Cost2Goal(j) + EdgeCost(k, j)
      if c $<$ Cost2Goal(k) **then**
        Cost2Goal(k) $\leftarrow$ c
        Next2Goal(k) $\leftarrow$ j
      end
    end
  end
end

Traditionally, graph search algorithms have been used to plan in the space of discrete cells or paths. But, in this work, graph search algorithms are used to plan in the space of gracefully composable motion policies, $i.e.$, controllers. The graph search algorithms now provide a motion plan that consists of a sequence of gracefully composable motion policies that achieve the overall navigation task. In this work, the Dijsktra's algorithm (Dijkstra 1959) shown in Algorithm 1 is used to solve the single-goal optimal navigation problem. The candidates for the optimality criterion include fastest time and shortest path. Unlike other heuristic-based graph search algorithms like A^* (Hart et al. 1968), Dijsktra's algorithm uses a dynamic programming approach and optimizes over the actual cost funtion without the use of any heuristics.

Given a goal position state, the Euclidean distance metric is used to find the closest trim motion policy

whose goal domain contains it, and its corresponding node in the gracefully prepares graph Ω forms the goal motion policy node G. Algorithm 1 generates a single-goal optimal motion policy tree $\Gamma(\Omega, G)$, which contains optimal paths from all motion policy nodes in the gracefully prepares graph Ω to the goal motion policy node G. Each motion policy node i in the optimal motion policy tree $\Gamma(\Omega, G)$ contains a motion policy Φ_i, its cost to reach the goal motion policy node given by *Cost2Goal*(i), and a pointer to the next optimal motion policy node given by *Next2Goal*(i). The algorithm begins with resetting the *Cost2Goal* values and the *Next2Goal* pointers for all the motion policy nodes in the optimal policy tree Γ except for the goal motion policy node G whose *Cost2Goal* value is set to zero (*lines* $3-7$). A list of unoptimized nodes Q is created and iterated over (*lines* $10-23$). At each iteration, the motion policy node with the minimum *Cost2Goal* given by *MinCostNode*(Q) is removed from Q, and the *Cost2Goal* values and the *Next2Goal* pointers of its parents are updated (*lines* $16-22$). The function *EdgeCost*(k, j) returns the cost of switching from motion policy Φ_k to Φ_j. This iteration continues until all unoptimized nodes are optimized or all remaining motion policy nodes in Q do not have a path to the goal motion policy node G, which is indicated by infinite *Cost2Goal* (*lines* $12-14$).

The optimal motion policy tree $\Gamma(\Omega, G)$ represents all optimal graceful motions that the robot can achieve in order to reach the goal motion policy node G. However, the optimality is limited by motion policies in the motion policy library. Any motion policy node that has a path to the goal motion policy node will reach the goal motion policy node by switching between an optimal sequence of motion policies that guarantee overall graceful motion by construction. A subtree of a single-goal time-optimal motion policy tree obtained using Algorithm 1 is shown in Fig. 12. The optimal motion policy sequences with XY projections of their 4D domains from a number of different initial positions on a map with static obstacles are shown in Fig. 12. Since the optimal motion policy tree contains the optimal sequence of gracefully composable motion policies from all trim conditions to the goal motion policy node, replanning is unnecessary when the robot's start position changes. However, the optimal motion policy tree has to be regenerated when the goal motion policy node changes.

This section presented a motion planner that plans in the space of gracefully composable motion policies (controllers) and explicitly accounts for both the dynamics of the system and the domains of the controllers. Each motion policy knows the exact motion it achieves in the environment and also knows that it is collision-free. Therefore, in this framework, the motion planner has knowledge of the system dynamics, and capabilities and limitations of the underlying controllers used to achieve the motion plans. The controllers, on the other hand, have knowledge of the environment constraints and also the motions that they produce, thereby, forming a truly integrated motion planning and control framework.

C. Hybrid Control

The optimal path to the goal motion policy node from any start motion policy node in the optimal motion policy tree is obtained by following its *Next2Goal* pointer until the goal motion policy node is reached. A hybrid controller is used as a master/supervisory controller to ensure successful execution of the optimal sequence of motion policies. The hybrid controller starts executing a motion policy Φ_i and resets its timer only if the robot's position state is inside its start domain $\mathbb{S}_i$. It continues executing the motion policy Φ_i only if the robot's position state lies inside its domain $D_i{}'(t)\ \forall t \in [0, t_f]$, and the motion policy execution is stopped when its timer runs out and the robot's position state reaches its goal domain $\mathbb{G}_i$. The switch to the next motion policy Φ_j happens naturally as the goal domain of Φ_i lies inside the start domain of Φ_j, *i.e.*, $\mathbb{G}_i \subset \mathbb{S}_j$ by construction of the gracefully prepares graph.

The feedback control law $\phi_i(t)$ of a motion policy Φ_i is capable of handling small disturbances and uncertainties. But when subjected to large disturbances, the

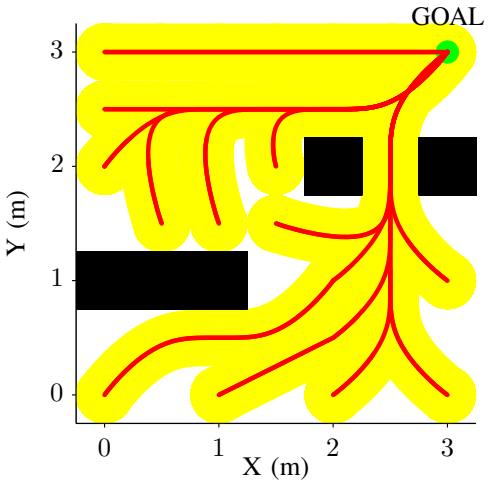


Fig. 12. A subtree of a single-goal time-optimal motion policy tree. The lines represent position space motions of the motion primitives, while the shaded regions show 2D projections of the 4D motion policy domains, including their outer domains.

robot's position state can exit the domain $D'(t)$, for some $t \in [0, t_f]$. Since the hybrid controller checks for the validity of the motion policy $\forall t \in [0, t_f]$, it detects the domain exit, stops the execution of the current motion policy, finds another motion policy Φ_k whose start domain $\mathbb{S}_k$ contains this exiting position state, and switches to the new motion policy Φ_k. If there exists a path from the new motion policy Φ_k to the goal, then the optimal motion policy tree $\Gamma(\Omega, G)$ already contains the sequence of gracefully composable motion policies that lead the motion policy Φ_k to the goal motion policy node G. If the automatic instantiation procedure guaranteed 100% coverage, then there will always exist a motion policy Φ_k in the motion policy library whose start domain will capture the robot's exiting position state. But if 100% coverage was not guaranteed and if there is no motion policy in the motion policy library whose start domain captures the robot's exiting position state, then the hybrid controller executes the backup emergency controller. In this work, the hybrid controller switches to a simple balancing mode when such a scenario is encountered. It is to be noted that the switching from the motion policy Φ_i to the new motion policy Φ_k or the backup emergency controller is discrete and is not graceful as the disturbance added to the robot that caused the domain exit is discontinuous.

D. Dynamic Replanning

While navigating human environments, it is inevitable that new static obstacles or dynamic obstacles are encountered. In such cases, a dynamic replanning procedure is necessary that will avoid motion policies that result in collision with the new obstacles.

Figure 13(a) shows an example optimal motion policy tree with a goal motion policy Φ_1. The optimal motion policy tree accounts for the known static obstacles in the environment, and hence each motion policy Φ_i represents a collision-free motion policy in the known environment. However, the introduction of new static and dynamic obstacles can invalidate some of these motion policies, for example, the motion policy Φ_3 in Fig. 13(b) is invalidated by a new obstacle. In such cases, the optimal motion policy tree must be updated to avoid these invalid motion policies as shown in Fig. 13(c).

The simplest approach to account for the new obstacles is to regenerate the entire optimal motion policy tree while ignoring the invalid motion policy nodes as shown in Algorithm 2. Unlike Algorithm 1, Algorithm 2 optimizes only the valid motion policy nodes given by the *Valid*() function (*lines* $17 - 23$). In this approach, every time there is a change in the validity of motion

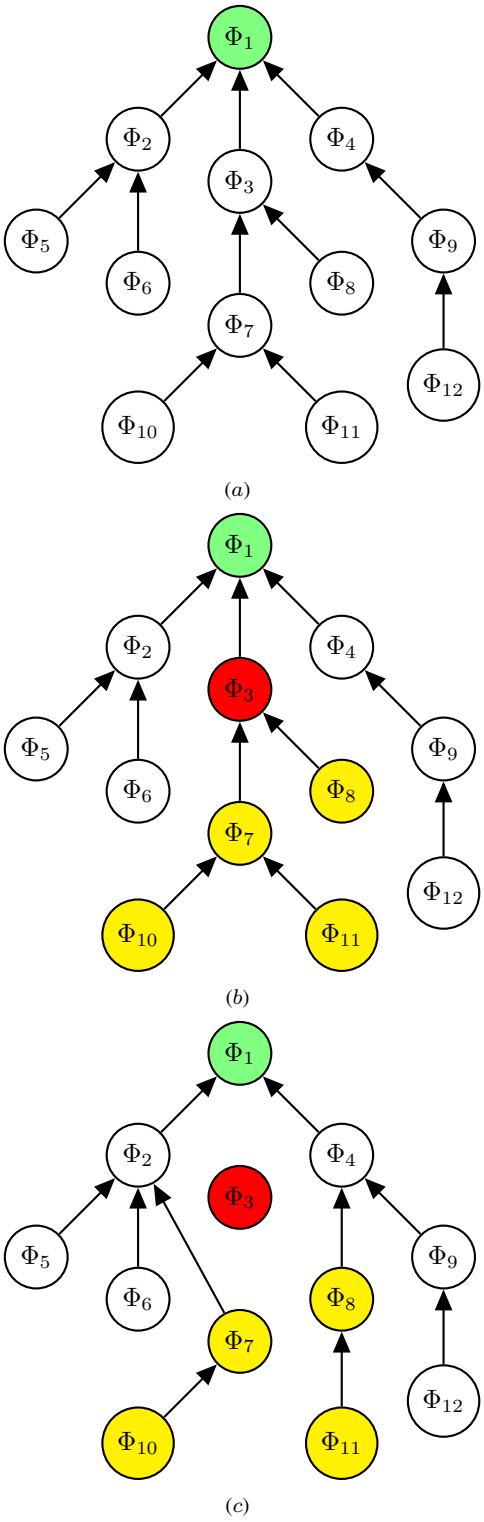


Fig. 13. (a) Single-goal optimal motion policy tree with the goal motion policy Φ_1; (b) The motion policy Φ_3 is invalidated by a new obstacle, and hence the motion policies $\Phi_7, \Phi_8, \Phi_{10}$ and Φ_{11} need to be updated; and (c) Updated optimal motion policy tree.

Algorithm 2: Dijkstra's Algorithm with Invalid Nodes

input : Gracefully Prepares Graph Ω
Goal Motion Policy Node G
output: Optimal Motion Policy Tree Γ

```
1  begin
2      Γ ← ∅
3      foreach i ∈ Node(Ω) do
4          Cost2Goal(i) ← ∞
5          Next2Goal(i) ← ∅
6          Γ ← Γ ∪ (i, Cost2Goal(i), Next2Goal(i))
7      end
8      Cost2Goal(G) ← 0
9      Q ← Γ
10     while Q ≠ ∅ do
11         j ← MinCostNode(Q)
12         if Cost2Goal(j) = ∞ then
13             break
14         end
15         Q ← Q − {j}
16         foreach k ∈ Parent(j) do
17             if Valid(k) then
18                 c ← Cost2Goal(j) + EdgeCost(k, j)
19                 if c < Cost2Goal(k) then
20                     Cost2Goal(k) ← c
21                     Next2Goal(k) ← j
22                 end
23             end
24         end
25     end
26 end
```

Algorithm 3: Find Nodes to be Updated

input : Base Optimal Motion Policy Tree Γ_0
Goal Motion Policy Node G
Set of Invalid Nodes I
output: Set of Nodes to be Updated U

```
1  begin
2      U ← ∅
3      foreach i ∈ I do
4          U ← U ∪ i
5      end
6      Q ← G
7      while Q ≠ ∅ do
8          i ← Pop(Q)
9          foreach j ∈ TreeParent(Γ_0, i) do
10             if i ∈ U then
11                 U ← U ∪ j
12             end
13             Q ← Q ∪ j
14         end
15     end
16 end
```

policy nodes, the optimal motion policy tree has to be regenerated. For example, the motion policy nodes $\Phi_7, \Phi_8, \Phi_{10}, \Phi_{11}$ shown in Fig. 13(b) are the only nodes that had to be updated, whereas the other nodes did not require any updates. It is computationally inefficient to regenerate the entire motion policy tree when only a subset of the motion policy nodes need to be updated. Therefore, this section presents an efficient dynamic replanning algorithm that updates only the motion policy nodes that need to be updated. This algorithm consists of the following three steps: (i) finding invalid motion policies, (ii) finding motion policy nodes to be updated, and (iii) updating the motion policy nodes.

1) Finding Invalid Motion Policies: The motion planner uses occupancy grids (Elfes 1989) to find the invalid cells in a finely discretized 2D map of the environment. The mapping from an occupancy cell to a motion policy in the motion policy library is obtained during the automatic motion policy instantiation procedure described in Sec. V-A. Each occupancy cell has a list of motion policies whose domains cover it, and this list is updated for every valid motion policy instantiation. Therefore given a list of occupied cells, a simple look-up operation provides the list of invalid motion policy nodes I.

2) Finding Motion Policy Nodes to be Updated: Figure 13(b) shows that the motion policy nodes that need to be updated belong to the subtree with the invalid motion policy node as its head. Any valid motion policy that reaches the goal motion policy via an invalid motion policy must be updated. This subtree of motion policy nodes can be found by traversing down the optimal motion policy tree of the base case, $i.e.$, the map with all known static obstacles and no new obstacles. The algorithm used to find the motion policy nodes to be updated is shown in Algorithm 3. The base optimal motion policy tree Γ_0 is the best possible motion policy tree that can be generated, and is generated using Algorithm 1. Each motion policy node i has a list of motion policy nodes that point to it (parents) in the base optimal motion policy tree Γ_0 given by *TreeParent*(Γ_0, i). All invalid motion policy nodes are added to the list of nodes to be updated U (*lines* $3-5$), and the search for nodes to be updated begins with the goal motion policy node G and traverses down the entire base optimal motion policy tree Γ_0 in a breadth-first search manner (*lines* $6-15$). If a motion policy node i is marked to be

Algorithm 4: Update Optimal Motion Policy Tree

input : Base Optimal Motion Policy Tree Γ_0
Goal Motion Policy Node G
Set of Nodes to be Updated U
output: Updated Optimal Motion Policy Tree Γ

1 **begin**
2 $\Gamma \leftarrow \Gamma_0$
3 **foreach** $i \in U$ **do**
4 Cost2Goal(i) $\leftarrow \infty$
5 Next2Goal(i) $\leftarrow \emptyset$
6 **end**
7 **foreach** $i \in U$ **do**
8 **if** Valid(i) **then**
9 **foreach** $j \in$ Child(i) **do**
10 **if** Valid(j) **then**
11 c $\leftarrow$ Cost2Goal(j) + EdgeCost(i, j)
12 **if** c $<$ Cost2Goal(i) **then**
13 Cost2Goal(i) $\leftarrow$ c
14 Next2Goal(i) $\leftarrow j$
15 **end**
16 **end**
17 **end**
18 **else**
19 $U \leftarrow U - \{i\}$
20 **end**
21 **end**
22 $Q \leftarrow U$
23 **while** $Q \neq \emptyset$ **do**
24 $j \leftarrow$ MinCostNode(Q)
25 **if** Cost2Goal(j) = ∞ **then**
26 **break**
27 **end**
28 $Q \leftarrow Q - \{j\}$
29 **foreach** $k \in$ Parent(j) **do**
30 **if** Valid(k) **then**
31 c $\leftarrow$ Cost2Goal(j) + EdgeCost(k, j)
32 **if** c $<$ Cost2Goal(k) **then**
33 Cost2Goal(k) $\leftarrow$ c
34 Next2Goal(k) $\leftarrow$ j
35 **end**
36 **end**
37 **end**
38 **end**
39 **end**

updated, then all its parents in the base optimal tree Γ_0 given by *TreeParent*(Γ_0, i) are added to the list of nodes to be updated U (*lines* $10 - 12$).

3) Update the Motion Policy Nodes: The procedure to update the optimal motion policy tree Γ is presented in Algorithm 4. Every time the optimal motion policy tree Γ is to be updated, it is reset to the base optimal motion policy tree Γ_0, and the *Cost2Goal* values and *Next2Goal* pointers for all motion policy nodes in the list of nodes to be updated U are reset (*lines* $3 - 6$). For each valid motion policy node i in the list U given by *Valid*(i), its *Next2Goal* pointer is initialized to its valid child motion policy node with the minimum *Cost2Goal* (*lines* $9 - 17$). The children of each motion policy node are obtained from the gracefully prepares graph Ω. The invalid motion policy nodes are removed from the list U, while the valid motion policy nodes with the initialized *Cost2Goal* values and *Next2Goal* pointers are added to the list of unoptimized nodes Q. The procedure to optimize the *Cost2Goal* values for just these motion policy nodes (*lines* $25 - 40$) is same as that in Algorithm 1 (*lines* $10 - 23$) and in Algorithm 2 (*lines* $10 - 25$).

The dynamic replanning algorithm presented above is used only when the validity of a motion policy in the motion policy library changes. When such a change is detected, the current set of invalid motion policy nodes I is determined, and Algorithm 3 is used to find the motion policy nodes in the base optimal motion policy tree Γ_0 that need to be updated. The optimal motion policy tree Γ is then reset to Γ_0 and updated using Algorithm 4.

VI. Experimental Results with The Ballbot

This section presents experimental results of the ballbot successfully achieving different navigation tasks in a graceful manner using the integrated motion planning and control framework presented in Sec. V.

A. Experimental Setup

The ballbot localizes itself on a 2D map of the environment using a particle filter based localization algorithm presented in (Biswas et al. 2011). The encoders on the ball motors of the inverse mouse-ball drive provide the odometry data for the prediction step, while a Hokuyo URG-04LX laser range finder provides the laser scan readings for the correction step.

The laser range finder is also used to detect obstacles using an occupancy grid map (Elfes 1989) with a grid spacing of 0.1 m. The laser range finder is mounted on the front of the robot with only a 180° field of view, and hence obstacles behind the laser cannot be detected. However, the ballbot can remember a previously encountered obstacle using its occupancy grid map. The occupancy grid map is primarily used to detect new static and dynamic obstacles, whereas permanent static obstacles are included in the map.

B. Motion Policy Library

In order to illustrate the integrated motion planning and control framework's capability to handle different motion policy palettes, this section presents experimental results that used two motion policy palettes, one with 39 unique motion policies and another with 43 unique motion policies. They used different motion primitive sets with the same distance parameter d = 0.5 m like the ones shown in Fig. 5. The palette with 43 motion policies had more non-stationary trim conditions than the palette with 39 motion policies. The problem of determining the minimal set of motion policies required to achieve a navigation task is an open research question. It is not addressed in this paper and will be explored in the future.

For all the results presented in this section, the motion policies were automatically instantiated in a 3.5 m × 3.5 m obstacle-free area of a map of our lab as described in Sec. V-A. The motion policy palette with 39 motion policies produces a motion policy library of 4521 instantiated motion policies, while the motion policy palette with 43 motion policies produces a motion policy library of 4569 instantiated motion policies. For both the motion policy palettes, the total time taken to generate the motion policy library and the gracefully prepares graph is approximately 2.5 s on the dual core computer on-board the robot. Similarly, the same computer takes about 0.05 s to generate the optimal motion policy tree for both the motion policy palettes. This allows the optimal motion policy tree to be regenerated and updated in real-time. All the results presented in this section use fastest time as the optimality criterion.

C. Point-Point Motion

The point-point navigation task can be formulated as a motion between trim motion policies with constant position reference trajectories as motion primitives. The goal motion policy in the motion policy library is given by the trim motion policy whose goal position state is the closest to the desired goal position by Euclidean distance metric. Figure 14 shows the ballbot successfully reaching a single goal position state of (2 m, 2 m, 0 m/s, 0 m/s) from four different starting configurations in the presence of two static obstacles. The experimental position space motions shown in Fig. 14 were obtained from the localization algorithm (Biswas et al. 2011).

This navigation task used the motion policy palette with 39 motion policies to generate the motion policy library. A single-goal time-optimal motion policy tree was generated online using Algorithm 2 that takes into account the known static obstacles in the map. Each of the motions shown in Fig. 14 were obtained by tracking

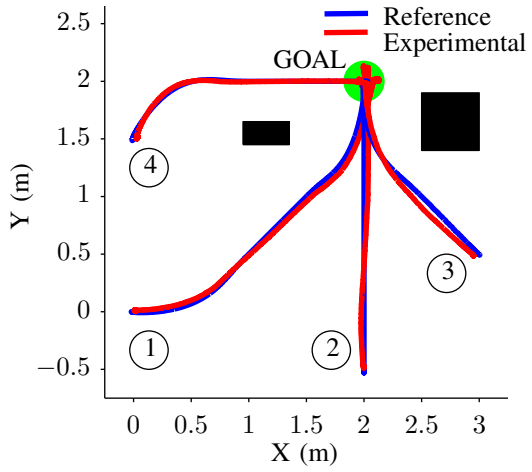


Fig. 14. Point-Point motion with two obstacles, shown in black.

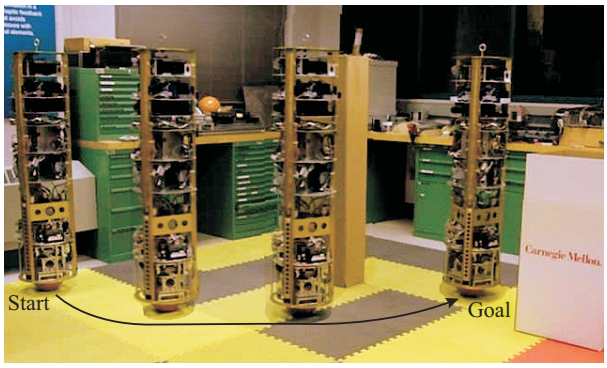


Fig. 15. Composite frames from a video of the ballbot achieving the goal from start position no. 1.

motions from the single-goal time-optimal motion policy tree, and were not regenerated for each run.

Figure 15 shows the composite frames from a video of the ballbot achieving the goal from the first starting position, and the resulting body angle trajectories are

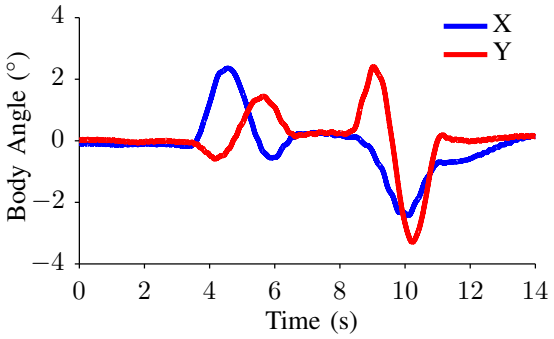


Fig. 16. Point-Point motion: Body angle trajectories to achieve the goal from start position no. 1.

shown in Fig. 16. The video of the ballbot achieving all four motions can be found in Extension 2.

D. Disturbance Handling

Figure 17 presents the experimental results that illustrate the capability of the integrated motion planning and control framework to handle large disturbances. While the ballbot was executing a point-point motion, it was physically held and stopped from moving towards the goal. Then, the ballbot was physically moved to a different point on the map and let go.

When the ballbot was physically stopped from moving towards the goal, its position state exited the domain of the motion policy it was executing, and the hybrid control architecture detected this domain exit. It then searched the motion policy library to find motion policies whose start domain contained this exiting position state. Since a continued disturbance was applied to the robot, its position state continued to exit the domains that were found until the ballbot was set free to move on its own. After it was set free, the ballbot successfully reached the goal as shown in Fig. 17. Here again, the single-goal time-optimal motion policy tree was not regenerated. The composite frames from a video of the ballbot achieving this motion are shown in Fig. 18, and the video can be found in Extension 2.

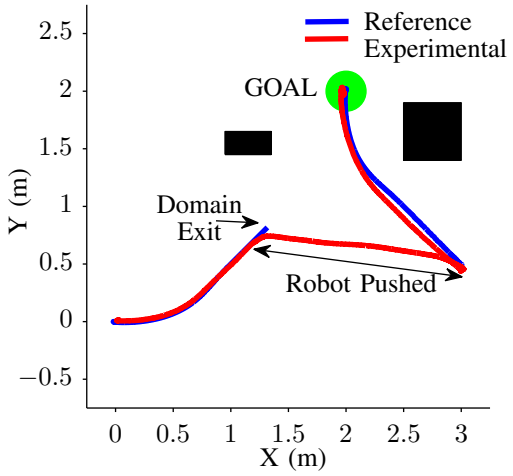


Fig. 17. Disturbance Handling

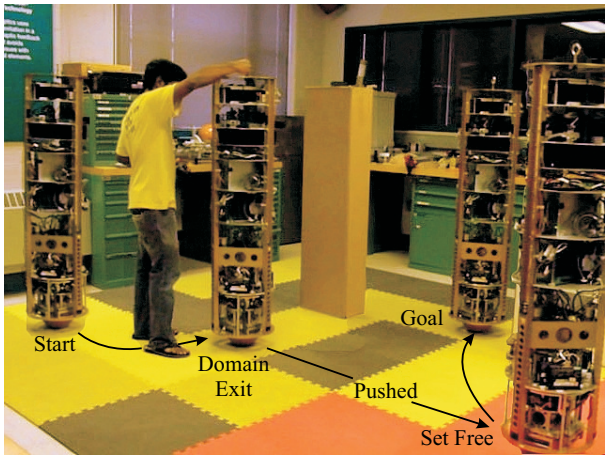


Fig. 18. Composite frames from a video of the ballbot reaching the goal while handling a disturbance.

E. Surveillance

The navigation task of surveillance can be formulated as a motion between trim motion policies that have constant velocity reference trajectories as motion primitives. The task is specified as a sequence of moving goal position states that are repeated in a cyclic fashion. Unlike point-point motion, the surveillance motion has multiple cyclic goals. A new goal motion policy is found every time the current goal motion policy is reached, and hence the time-optimal motion policy tree has to be regenerated. The motion planner takes only 0.05 s to regenerate the motion policy library used here, and all of its motion policies have time durations greater than or equal to 1 s. Hence, the optimal motion policy tree can be regenerated in real-time while executing the last motion policy to the current goal. This process repeats until the user quits the surveillance task. Figure 19 shows the ballbot successfully performing a surveillance task with four goal configurations using the motion policy palette with 39 motion policies. In this run, the surveillance

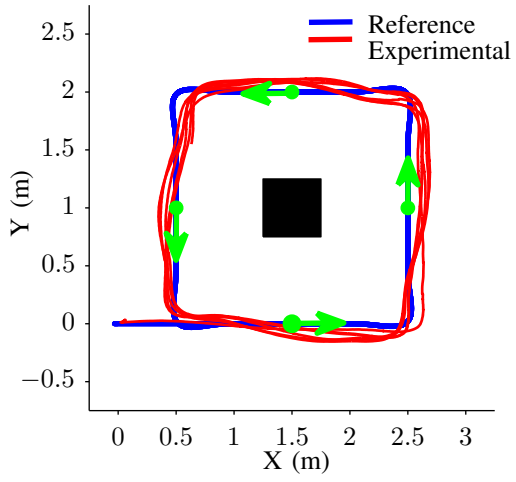


Fig. 19. Surveillance motion with four goal configurations represented by arrows, and one obstacle, shown in black.

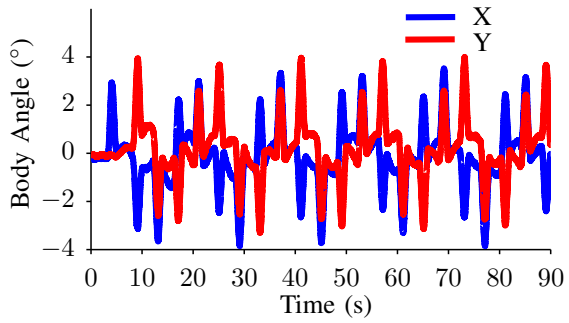


Fig. 20. Body angle trajectories for the surveillance motion with four goal configurations.

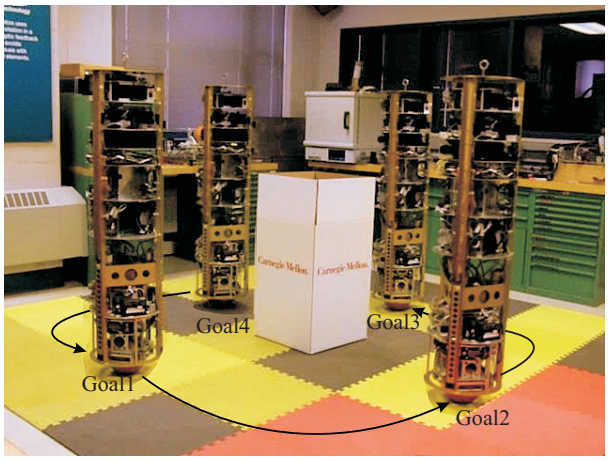


Fig. 21. Composite frames from a video of the ballbot achieving the surveillance motion with four goal configurations.

task was quit after five successful loops. The body angle trajectories for one complete surveillance loop are shown in Fig. 20. The composite frames from a video of the ballbot achieving this task is shown in Fig. 21, and the video can be found in Extension 2.

Two successful loops of another surveillance task with ten goal configurations are shown in Fig. 22. This surveillance task was achieved using the motion policy palette with 43 motion policies, and the optimal motion policy tree was successfully regenerated real-time on the robot for every single goal configuration. The resulting body angle trajectories for one complete surveillance cycle are shown in Fig. 23.

F. Dynamic Replanning

Figure 24 shows the ballbot using the dynamic replanning algorithm presented in Sec. V-D to avoid new static and dynamic obstacles, and successfully reach the goal. The composite frames from a video of the ballbot performing this motion is shown in Fig. 25, and the video can be found in Extension 2.

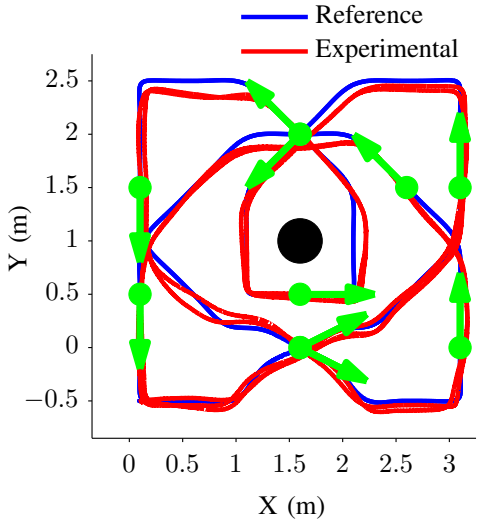


Fig. 22. Surveillance motion with ten goal configurations represented by arrows, and one obstacle, shown in black.

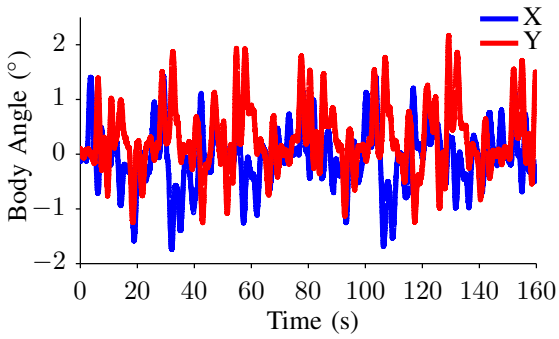


Fig. 23. Body angle trajectories for the surveillance motion with ten goal configurations.

The base optimal reference motion to the goal without any new static or dynamic obstacles is shown in Fig. 26(a). The optimal reference motion to the goal with the new static obstacle and the dynamic obstacle in its first position is shown in Fig. 26(b). The optimal reference motion to the goal with the dynamic obstacle moving to its second location is shown in Fig. 26(c). The final optimal reference motion to the goal along with the resulting ballbot motion are shown in Fig. 24. It is important to note that the optimal motions achieved using this approach are limited by the motion policies in the motion policy library. For example, one can see that the optimal reference motion shown in Fig. 26(b) is not optimal in a true sense but is optimal w.r.t. the motion policies available in the motion policy library.

The motion policy palette with 43 motion policies

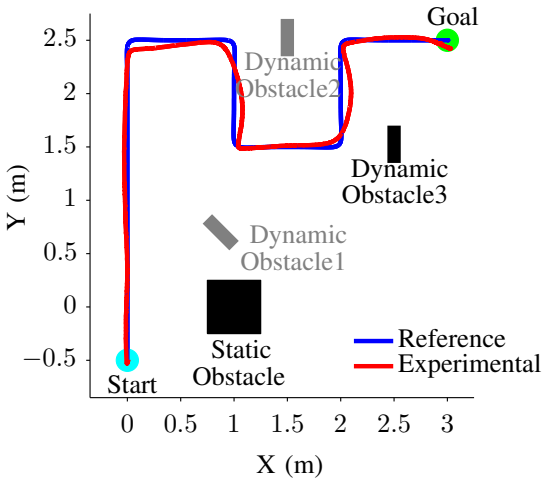


Fig. 24. Dynamic replanning to reach the goal

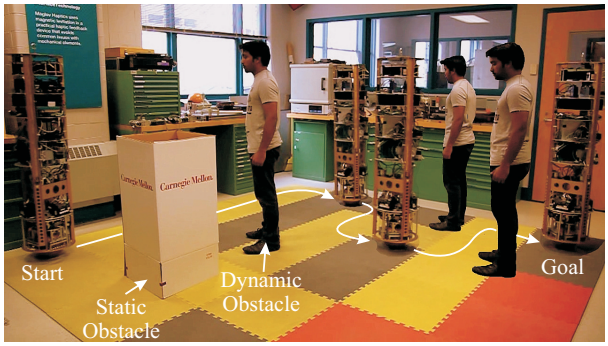


Fig. 25. Composite frames from a video of the ballbot dynamically replanning to avoid static and dynamic obstacles.

was used for this run. The dual-core computer on-board the ballbot takes 0.01 s to 0.05 s to update the optimal motion policy tree depending on the number of motion policy nodes to be updated. Each motion policy is of time duration greater than or equal to 1 s, and hence the optimal motion policy tree can be updated fast enough to account for the dynamic obstacles, which are detected by the laser range finder at 10 Hz, *i.e.*, every 0.1 s.

VII. Conclusions

This paper presented an integrated motion planning and control framework that enables balancing mobile robots like the ballbot to gracefully navigate human environments. A notion of *gracefully prepares relationship* that ensured graceful switching between motion policies was introduced. A palette of gracefully composable motion policies were designed offline, and an online automatic instantiation procedure that deploys these motion policies to fill a map of the environment was presented.

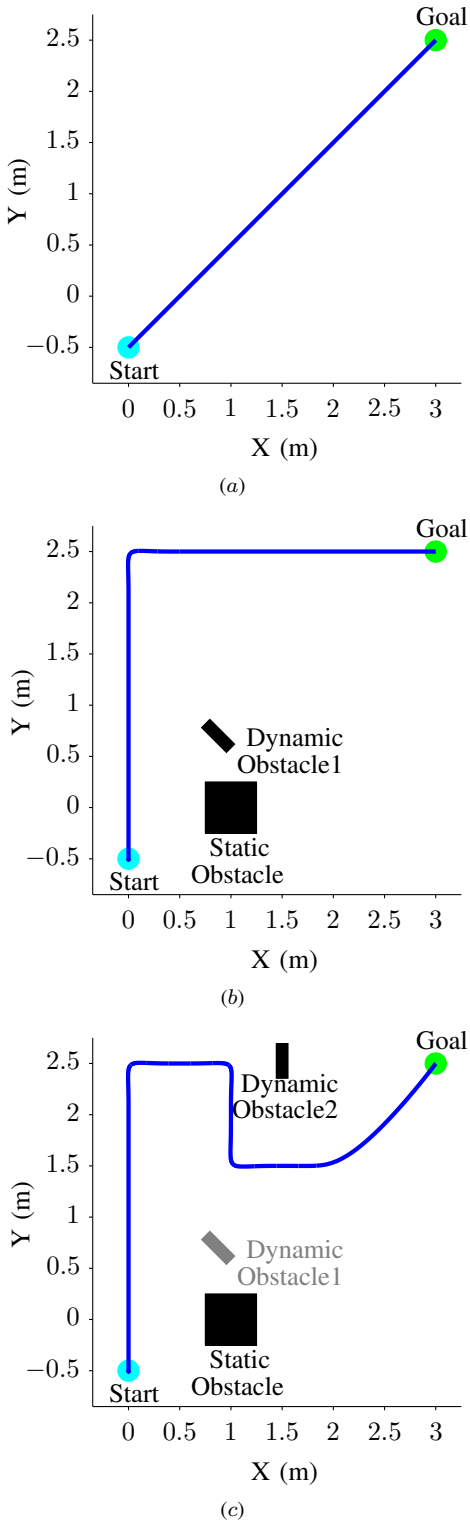


Fig. 26. (*a*) The base optimal reference motion to the goal; (*b*) The optimal reference motion with the static obstacle, and the dynamic obstacle at its first location; (*c*) The optimal reference motion when the dynamic obstacle has moved to its second location.

A *gracefully prepares graph* that represented all possible graceful motions that the robot can achieve in a map was generated online, and the navigation tasks were formulated as graph-search problems.

Dijkstra's algorithm was used to generate the single-goal optimal motion policy tree, and the optimality was limited to the motion policies available in the motion policy library. A variety of different point-point and surveillance navigation tasks were successfully tested on the ballbot using the integrated motion planning and control framework. Experiments that demonstrate the capability of the framework to successfully handle disturbances were also presented. Finally, a dynamic replanning algorithm that replans the single-optimal motion policy tree in the presence of dynamic obstacles was presented, and successfully tested on the ballbot.

VIII. Future Work

One of the improvements that can be done to the work presented in this paper is in the design and verification of the motion policy domains. The current advancements in algebraic verification of controllers using sums-of-squares (SOS) tools (Tedrake et al. 2010; Tobenkin et al. 2011) can be used to design invariant domains for the motion policies. It is difficult to achieve the desired coverage in environments with narrow paths as the motion policy domain definitions presented in this paper are fixed. But the invariant motion policy domain definitions will allow one to explore ways to scale-down these motion policy domains during instantiation so that the obstacles can be avoided and the desired coverage can also be guaranteed.

The work presented in this paper used Dijkstra's algorithm for graph search, which may not be fast enough on bigger graphs that cover larger areas. In order to handle large maps, two different approaches can be explored. One approach involves the use of heuristic based graph search algorithms like D^* (Stentz 1995) and D^* *Lite* (Koenig and Likachev 2002), while the other approach involves dividing the large map into smaller regions and piecing together locally optimal motions to achieve the global goal. The design of admissible heuristics in the space of motion policies is a challenging problem and must be explored. The automatic instantiation procedure presented in this paper uniformly distributed motion policies on a map. However, variable density instantiation procedures that sparsely fill open spaces and densely populate narrow spaces can also be explored. The problem of determining the minimal set of motion policies that can optimally achieve a navigation task is an open research question, and it needs to be addressed. Moreover, metrics that can compare and evaluate two motion policy palettes for a navigation task must also be explored.

IX. Acknowledgements

This work was supported in part by NSF grants IIS-0308067 and IIS-0535183. We thank Joydeep Biswas for providing the localization algorithm implementation, and Byungjun Kim and Michael Shomin for their vital help in running the ballbot experiments. We also thank the reviewers for their invaluable comments and suggestions that helped us significantly improve the paper.

References

A. A. Agrachev and Y. L. Sachkov. An intrinsic approach to the control of rolling bodies. In *Proc. IEEE Conf. on Decision and Control*, pages 431–435, 1999.

F. Amirabdollahian, R. Loureiro, and W. Harwin. Minimum jerk trajectory control for rehabilitation and haptic applications. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 3380–3385, 2002.

Anybots. http://anybots.com, 2010.

C. Belta, V. Iser, and G. J. Pappas. Discrete abstractions for robot planning and control in polygonal environments. *IEEE Trans. on Robotics*, 21(5):864–874, 2005.

A. Bicchi and R. Sorrentino. Dextrous manipulation through rolling. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 452–457, 1995.

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*, pages 73–78, 2011.

R. W. Brockett and L. Dai. Nonholonomic kinematics and the role of elliptic functions in constructive controllability. *Nonholonomic Motion Planning*, pages 1–20, 1993.

R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18(6):534–555, 1999.

D. C. Conner, Howie Choset, and A. A. Rizzi. Integrated planning and control for convex-bodied nonholonomic systems using local feedback. In *Proc. Robotics: Science and Systems II*, pages 57–64, 2006.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

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.

P. Deegan, R. Grupen, A. Hanson, E. Horrell, S. Ou, E. Riseman, S. Sen, B.Thibodeau, A. Williams, and D. Xie. Mobile manipulators for assisted living in residential settings. *Autonomous Robots*, 2007.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

T. Flash and N. Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of Neuroscience*, 5:1688–1703, 1985.

E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *Proc. IEEE Conf. on Decision and Control*, pages 821–826, 2000.

E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA J. Guid., Control, Dynam.*, 25(1):116–129, 2002.

E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Trans. on Robotics*, 21(6):1077–1091, 2005.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics*, 4 (2):100–107, 1968.

K. Hauser, T. Bretl, K. Harada, and J. C. Latombe. Using motion primitives in probabilistic sample-based planning for humanoid robots. *Algorithmic Foundations of Robotics VII, Springer Tracts in Advanced Robotics*, 47:507–522, 2008.

N. Hogan. An organizing principle for a class of voluntary movements. *Journal of Neuroscience*, 4: 2745–2754, 1984.

R. Hollis. Ballbots. *Scientific American*, pages 72–78, Oct 2006.

D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21(3):233–255, 2001.

iBOT. http://www.ibotnow.com, 2003.

G. Kantor and A. A. Rizzi. Feedback control of underactuated systems via sequential composition: Visually guided control of a unicycle. In *11th Int'l Symp. of Robotics Research*, Oct. 2003.

E. Klavins and D. E. Koditschek. A formalism for the composition of concurrent robot behaviors. In *Proc. IEEE Int'l. Conf. on Robotics and Automation*, volume 4, pages 3395–3402, 2000.

R. A. Knepper and M. T. Mason. Empirical sampling of path sets for local area motion planning. In *Int'l Symp. on Experimental Robotics*, 2008.

R. A. Knepper and M. T. Mason. Path diversity is only part of the problem. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 3260–3265, 2009.

S. Koenig and M. Likachev. D* lite. In *Proc. 18th National Conf. on Artificial Intelligence*, pages 476–483, 2002.

M. Kumagai and T. Ochiai. Development of a robot balancing on a ball. *Intl. Conf. on Control, Automation and Systems*, 2008.

M. Kumagai and T. Ochiai. Development of a robot balancing on a ball - application of passive motion to transport. In *Proc. IEEE Int'l. Conf. on Robotics and Automation*, pages 4106–4111, 2009.

K. J. Kyriakopoulos and G. N. Saridis. Minimum jerk path generation. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 364–369, 1988.

T. Lauwers, G. Kantor, and R. Hollis. One is enough! In *Proc. Int'l. Symp. for Robotics Research*, Oct. 2005.

T. B. Lauwers, G. A. Kantor, and R. L. Hollis. A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive. In *Proc. Int'l. Conf. on Robotics and Automation*, pages 2884–2889, 2006.

S. M. LaValle and J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

K. Levenberg. A method for the solution of certain non-linear problems in least squares. *The Quarterly of Applied Mathematics*, 2:164–168, 1944.

Z. Li and J. Canny. Motion of two rigid bodies with rolling constraint. *IEEE Trans. on Robotics and Automation*, 6(1):62–72, 1990.

A. Marigo and A. Bicchi. Rolling bodies with regular surface: Controllability theory and applications. *IEEE Trans. on Automatic Control*, 45(9):1586–1599, 2000.

U. Nagarajan. Dynamic constraint-based optimal shape trajectory planner for shape-accelerated underactuated balancing systems. In *Proc. Robotics: Science and Systems*, 2010.

U. Nagarajan and R. Hollis. Shape space planner for shape-accelerated balancing mobile robots. *Int'l Journal of Robotics Research*, 2013. (Under Review).

U. Nagarajan, G. Kantor, and R. Hollis. Trajectory planning and control of an underactuated dynamically stable single spherical wheeled mobile robot. In *IEEE Int'l. Conf. on Robotics and Automation*, pages 3743–3748, 2009a.

U. Nagarajan, G. Kantor, and R. Hollis. Human-robot

physical interaction with dynamically stable mobile robots. *4th ACM/IEEE Int'l. Conf. on Human-Robot Interaction*, 2009b. (Short paper and video).

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. In *IEEE Int'l. Conf. on Robotics and Automation*, pages 998–1003, 2009c.

U. Nagarajan, G. Kantor, and R. Hollis. Hybrid control for navigation of shape-accelerated underactuated balancing systems. In *Proc. IEEE Conf. on Decision and Control*, pages 3566–3571, 2010.

U. Nagarajan, G. Kantor, and R. Hollis. Integrated planning and control for graceful navigation of shape-accelerated underactuated balancing mobile robots. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 136–141, 2012a.

U. Nagarajan, B. Kim, and R. Hollis. Planning in high-dimensional shape space for a single-wheeled balancing mobile robot with arms. In *IEEE Int'l Conf. on Robotics and Automation*, pages 130–135, 2012b.

J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1964.

H. G. Nguyen, J. Morrell, K. Mullens, A. Burmeister, S. Miles, N. Farrington, K. Thomas, and D. Gage. Segway robotic mobility platform. In *SPIE Proc. 5609: Mobile Robots XVII*, Philadelphia, PA, 2004.

G. Oriolo and Y. Nakamura. Control of mechanical systems with second-order nonholonomic constraints: Underactuated manipulators. In *Proc. IEEE Conf. on Decision and Control*, pages 2398–2403, 1991.

G. Oriolo, M. Vendittelli, A. Marigo, and A. Bicchi. From nominal to robust planning: The plate-ball manipulation system. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 3175–3180, 2003.

M. Phillips, B. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. In *Proc. Robotics: Science and Systems*, 2012.

M. Pivtoraiko and A. Kelly. Efficient constrained path planning via search in state lattices. In *8th Int'l Symp. on Artificial Intelligence, Robotics and Automation in Space*, 2005.

M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

E. Plaku, L. E. Kavraki, and M. Y. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *IEEE Transactions on Robotics*, 26(3):469–482, 2010.

PR2. http://www.willowgarage.com/pr2, 2009.

Rezero. http://www.rezero.ethz.ch, 2010.

A. A. Rizzi, J. Gowdy, and R. L. Hollis. Distributed coordination in modular precision assembly systems. *The International Journal of Robotics Research*, 20 (10):819–838, 2001.

B. Rohrer, S. Fasoli, H. I. Krebs, R. Hughes, B. Volpe, W. Frontera, J. Stein, and N. Hogan. Movement smoothness changes during stroke recovery. *Journal of Neuroscience*, 22(18):8297–8304, 2002.

M. Shomin and R. Hollis. Differentially flat trajectory generation for a dynamically stable mobile robot. In *Proc. Int'l. Conf. on Robotics and Automation*, 2013.

M. W. Spong. Underactuated mechanical systems. In *Control Problems in Robotics and Automation*. Springer-Verlag, 1998.

A. Stentz. The focussed D* algorithm for real-time replanning. In *Int'l Joint Conf. on Artificial Intelligence*, Aug. 1995.

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*, pages 3304–3309, 2010.

R. Tedrake. LQR-trees: Feedback motion planning on sparse randomized trees. In *Proc. Robotics: Science and Systems*, 2009.

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*, 29(8):1038–1052, 2010.

M. M. Tobenkin, I. R. Manchester, and R. Tedrake. Invariant funnels around trajectories using sum-of-squares programming. In *IFAC World Congress*, 2011.

APPENDIX A: INDEX TO MULTIMEDIA EXTENSIONS

The multimedia extensions to this article are at: www.ijrr.org.

Extension	Type	Description
1	Video	It demonstrates the ballbot achieving fast and graceful motions by manually switching between gracefully composable motion policies
2	Video	It demonstrates the ballbot autonomously navigating an obstacle-ridden environment to successfully achieve point-point motions while handling disturbances, and also to achieve a surveillance motion with four goals. It also shows the ballbot achieving a point-point motion while replanning in real-time to successfully avoid dynamic obstacles