Toward a Dynamical Pick and Place

Robert R. Burridge*, Alfred A. Rizzi†, and Daniel E. Koditschek‡

Artificial Intelligence Laboratory
University of Michigan
Department of Electrical Engineering and Computer Science

Abstract

This paper presents an initial overview of our attempts to expand our understanding of autonomous dynamic manipulation. Specifically, it proposes a sample "Dynamical Pick and Place" task and a hierarchical feedback based control strategy for its execution. Included are a sequence of experimental verifications of the underlying behaviors required.

*Supported in part by the National Science Foundation under grant IRI-9123266 and in part by ARPA under grant B457.

†Supported in part by the National Science Foundation under grant IRI-9123266 and in part by the University of Michigan Center for Display Technology and Manufacturing.

‡Supported in part by the National Science Foundation under grant IRI-9123266.

1 Introduction

We consider in this paper the problem of sensor driven Dynamical Pick and Place manipulation. We are interested in constructing a system capable of using a flat paddle to acquire balls thrown into the workspace; balance or bat them as required through a clutter; loft them into an out of reach destination receptacle; and resist large unanticipated perturbations along the way. The controllers required to carry out these tasks *autonomously* presently exceed the capability of any robot or programmable mechanism we are aware of (including, of course, our own) and thus present a suitable challenge to the design principles underlying dynamical dexterity that we have developed to date [20, 2].

Our past work has resulted in an expanding family of working machines [24, 3] that exhibit superlative dexterity (arguably superior to that of a human) in a narrow domain, as well as an immature but growing body of theory to explain how [1, 18]. The problem proposed here focuses both on enlarging the domain of robot dexterity and on attempting to extract from the algorithms that confer it a primitive but very robust sort of computational intelligence.

A flexibly deployable robot will undoubtedly be equipped with a general purpose manipulator capable of handling many differently shaped and sized objects in unstructured environments — it cannot be optimized for any one. To be efficient such a robot must handle objects in a dynamically dextrous fashion: most objects are grabbed, batted or thrown, whereas guarded moves and quasi-static manipulations are reserved only for fragile objects.

1.1 Problem Statement

Our three degree of freedom (nearly spherical kinematics) Bühgler robot (see Figure 1 and [24, 21]) is equipped with a flat paddle and a field rate (60 Hz) stereo camera system [20]. Its workspace will be cluttered with fixed obstacles — some suspended from the ceiling creating low "doors" and some protruding from the floor creating high "windows." The suspended obstacles hang low enough that the paddle can only just pass through the doors when level with the ground. The protruding obstacles are high enough that the robot must raise its paddle to pass through.

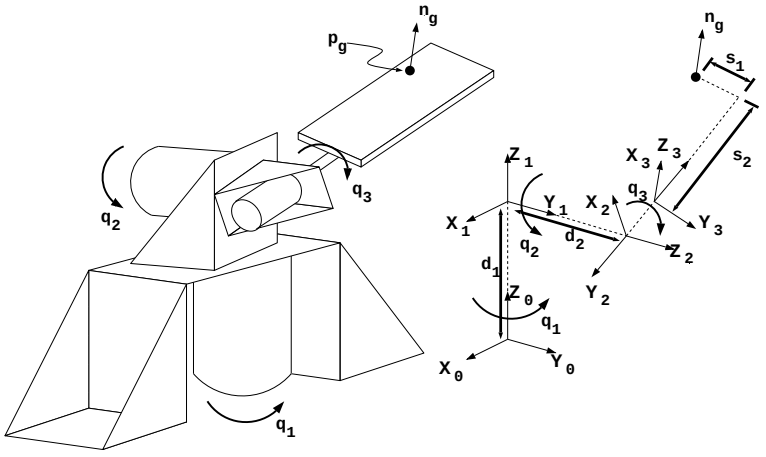

Figure 1: The Bühgler robot (taken from [20]).

The robot will be thrown a ball without warning into one of the free cells defined by the obstacles. The robot's task will then be to deposit the ball into a bowl in one of the other cells. The bowl will be located out of the robot's "reach," so that balls must be tossed in. By design the ball can not be tossed or batted through the "doors" because of the suspended obstacles: thus the robot must perform a "carry." Similarly, the ball must be lofted to get it through a "window" since the protruding obstacles are so high that it will roll off the paddle if an attempt is made to "carry" it across: the robot must perform a "toss and catch." When a ball is initially presented to the machine it may have unacceptably high horizontal velocity or be heading for an obstacle and require a fast "rescue." The robot must be able to distinguish situations where a ball can be saved from those where it would be too likely to damage itself in the effort to be worth trying. Finally, during the course of the manipulation the work cell may be "invaded" by disturbances (we will poke at the ball with a stick as we presently do in the juggling work [23]) that take the ball far off its course and off the track of the sequence of maneuvers previously planned.

1.2 Background

Let the state of an environment (in our case, the positions and velocities of the unactuated ball)be represented by $b \in B$, and let u denote the means by which a robot can change the state of the environment according to the rule

$$b' = f(b, u). \tag{1}$$

Much work in robotics is concerned with developing plans,

$$u = \Pi(t; b_I), \tag{2}$$

to bring b from a specified initial condition, b_I to a desired final condition with time (t) an explicit parameter. Unfortunately, Π often is very sensitive to b_I (an open loop move-box-to-pallet procedure will fail badly if the box is not initially located as assumed) and relies very strongly upon a predictive model for the world (removing a bottom box from the pallet during the stacking operation may result in catastrophe). Most implemented robot systems therefore surround (2) with periodic sensor derived "sanity checks" and include "exception handling." But no human programmer can anticipate all the varied ways in which the real world will depart from the model.

We are concerned instead with constructing purely feedback driven time-independent autonomous systems where

$$u = \Phi(b). \tag{3}$$

1.2.1 Robot Task Planning Literature

In recent years there have been a number of promising advances in the robotics literature in a variety of problem areas of concern to us. We have drawn motivation and inspiration from much of this work. Yet the literature continues to study problems in isolation that we wish to study in concert, albeit in greatly simplified form.

Pick and Place in Cluttered Environments Our focus on the dynamical pick and place problem is inspired in part by the HANDEY system developed by Lozano-Perez and colleagues [9] who emphasized the importance of developing task planning capabilities suitable for situations where regrasping is necessitated by environmental clutter (however our setup presents dynamical rather than quasi-static regrasping problems). Indeed, our emphasis on robustness and error recovery as the driving consideration in robot task planning derives from their original insights on fine motion planning [12] and the subsequent literature in this tradition bears a close relation to our concerns as will be touched upon below.

Comprehensive work by Kak and colleagues [26] has yielded a contemporary quasi-static assembly environment in the best traditions of Handey. The system builds a plan, (2), based upon sensed initial conditions, spawns an execution process that senses exceptions to the planned evolution of states and produces actions to bring the world's state back to the intermediate situation presumed by the plan. It is understanding the ways in which this higher level "feedback" (local replanning *is* a form of feedback) can be reasoned about and guaranteed to succeed that we wish to examine, albeit in the drastically simplified setting of the dynamic pick and place problem outlined above.

Discrete Events and Real-Time Computation The difficult question of how to reason about the interplay between sensing and recovery at the event level in robotics has been considerably stimulated by advent of the Ramadge-Wonham DES control paradigm [15]. In some of the most careful and convincing of such DES inspired robotics papers, Lyons proposes a formalism for encoding and reasoning about the construction of action plans and their sensor-based execution [14, 13] but explicitly avoids the consideration of problems wherein geometric and force sensor reports must be used to estimate progress and thereby stimulate the appropriate event transitions.

Error Detection and Recovery Despite the many dissimilarities in problem domain, models and representation, the work proposed here seems to bear the closest correspondence to the "fine-motion planning with uncertainty" literature in robotics (for example, as expounded in the excellent text of Latombe [11]) originated by Lozano-Perez and colleagues [12]. Traditionally this literature focuses on quasi-static problems (generalized damper dynamics [28] with coulomb reaction forces [8] on the contact set). Furthermore, the possible control actions typically are restricted to piecewise constant velocity vectors, each constant piece corrupted by a constant disturbance vector of known (bounded) magnitude and uncertain direction arising from a uniform distribution over the unit ball.

In contrast, we are interested in Newtonian dynamical models, and our control primitive is not a constant action but the entire range of actions consequent upon the closed loop policy, Φ, to be specified below. Moreover, we never "bother" to develop an explicit disturbance model even though we are specifically interested in operating in the empirical world with all its uncertainties and unmodeled phenomena. On the one hand, the theory and experience of building working controllers [10, 27] teaches us that the inevitable small but persistent disturbances arising from modeling errors, sensor inaccuracy and slight miscalibration are countered by the local structural stability properties of stable dynamical systems. On the other hand, large, arbitrary and unanticipated perturbations should be recoverable as long as they are relatively rare and do not violate the domain of attraction of every feedback policy at our disposal. In the end, of course, the only acceptable test of one or another model is to place the finished system in the physical setting for which it was designed.

Differences in models notwithstanding, we have derived considerable motivation from the "LMT" framework which attempts to match systematically the low level controllers' goals and capabilities with the more abstract requirements that characterize a task. In the work of Lozano-Perez and colleagues [5, 6], high level progress is made through a sequence of controller actions whose successful termination is ensured via careful choice of compliant motions in the presence of rigid objects. The sequence itself has been designed via a back chaining of motion pre-images. In our work, the funneling [6] action of sensorless compliance to rigid objects is replaced by the stability of a general dynamical system, but we borrow heavily from the notion of pre-image back-chaining, as will be seen in Section 4.

Unquestionably, our theoretical understanding of such questions in the proposed novel domain is rudimentary in comparison to the rich body of results that ten years of effort have brought the fine motion planning literature. For example, we require and are attempting to develop a notion of *progress* analogous to that introduced by Erdmann for piecewise constant control policies in the quasi-static regime [7], specifically we expect our behaviors to ensure some level of certainty at their termination in much the same way that certainty can be derived by making use of an inaccurate compliant motion to "push" an object into a rigid corner. Unfortunately we do not see any "corners" to push against in the dynamic setting beyond those which we can "create" via control policies. Finally there is clearly a need to carefully explore the design of termination predicates that are recognizable as discussed by Donald [5] but fit our richer sensor and actuator models.

Empirical and Conceptual Background The problems explored here address higher level issues of task execution, but there is still a strong relation to previous work in dextrous manipulation, in particular "robot juggling" performed in our laboratory. Indeed, it is exactly our contention that these ideas can be extended to build a variety of useful dextrous machines that are similarly single-minded in their pursuit of the user's goal behavior and ability to surmount unanticipated obstacles along the way. [3, 24, 2, 20].

The computational architecture used for the simple demonstrations of concept presented in this paper (and that we expect to use for the final implementation) is the direct descendant of that presented in [24].

2 Setup

In the problem with which we are concerned, there is a robot, which we can control directly, and an environment, which can only be manipulated through contact with the robot. The task will be to devise a strategy for the robot that drives the environment to a goal state, or set of states. In our experiments, the environment takes the form of a ball subject to gravity, and the robot manipulates the ball by batting it with a paddle.

In this section we will first develop some definitions and notation to be used in the sequel, then describe the experimental apparatus.

2.1 Definitions and Notation

Let $b = (b^1, b^2) \in \mathcal{B} = \mathbb{R}^6$ be the full state of the ball in Cartesian coordinates. Let $r = (r^1, r^2) \in \mathcal{R} = S^3 \times \mathbb{R}^3$ be the state of the robot in joint space. We use b^1 and b^2 to refer to the position and velocity of the ball respectively, and similarly r^1 and r^2 will represent the position and velocity of the robot.

2.1.1 Ball Flight

The ball in flight will be modeled by the Newtonian dynamics:

$$\dot{b} = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} b + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -g \end{bmatrix}. \tag{4}$$

Due to the simplicity of the ball flight dynamics, we can derive a closed form expression for the ball state at time t in the future as a function of present state:

$$b(t) = F^t(b) = \begin{bmatrix} I_{3\times3} & tI_{3\times3} \\ 0_{3\times3} & I_{3\times3} \end{bmatrix} b - g \begin{bmatrix} 0 \\ 0 \\ t^2/2 \\ 0 \\ 0 \\ t \end{bmatrix}. \tag{5}$$

Of course, we want the robot and ball to interact. We will use the standard restitution model of collisions between the paddle and the ball. At impact, we will assume that the robot's velocity is unchanged, and that the robot imparts an impulse to the ball, instantaneously redirecting the velocity of the ball. Only the component of the ball's velocity normal to the paddle is affected[1]. If we break the velocity of the ball into the normal part, $(b^2)_n$, and the tangential part, $(b^2)_t$, then the post-impact velocity is given by:

$$(b^2)' = (b^2)_t - \alpha(b^2)_n + (1+\alpha)(r^2)_n, \tag{6}$$

where $\alpha \in (0,1)$ is the coefficient of restitution, and $(r^2)_n$ is the normal velocity of the robot at the point of contact. We denote the full state after impact as a function of the ball and robot states before impact by $b' = C(b,r)$, as specified in (6).

Unless the ball and robot are in continuous contact, it is natural to divide the trajectory of the ball into epochs of time punctuated by collisions. The k^{th} epoch starts with the ball in state b_k, and ends immediately after the next impact, in state b_{k+1}. The trajectory of the robot during the k^{th} epoch will be denoted $r_k(b)$, emphasizing the dependence on b, but not t. The duration of the k^{th} epoch is τ_k.

[1] See Synge and Griffith [25] for a discussion of restitution models.

2.1.2 Interaction with the Environment

Define the contact set, $\mathcal{C}$, as the set of all ball and robot positions such that the ball position is precisely on the surface of the paddle (here we ignore the diameter of the ball). Define:

$$\begin{aligned} \rho : \mathcal{B} \times R \rightarrow \{0,1\} \quad &= 0 \quad if \quad (b^1, r^1) \in \mathcal{C} \\ &= 1 \quad otherwise, \end{aligned} \tag{7}$$

$$\tau_k = \min t : \rho(b(t), r_k(b(t))) = 0. \tag{8}$$

Assuming that our observer and robot controllers are tracking correctly (see below), we may now express τ_k as an implicit function of b_k:

$$\tau_k = \min t : \rho(F^t(b_k), r_k(F^t(b_k))) = 0. \tag{9}$$

2.1.3 Controllers and Their Induced Return Maps

Since the robot has no effect on the ball except at contact, we would like to ignore from now on the trajectory of the actuator system, and only consider the state of the robot at the next impact. We will denote the state of the robot immediately prior to the next impact (9) by the shorthand $u_k = \Phi(b_k)$, and use the collision law (6) to determine the induced effect on the ball.

Given the time to impact, (9), and the flight model, (5), the state of the ball at the start of the next epoch can now be expressed as a function of the state at the beginning of the previous epoch, yielding the "return map",

$$b_{k+1} = f(b_k, u_k) = C(F^{\tau_k}(b_k), r_k(F^{\tau_k}(b_k))) := f_\Phi(b_k). \tag{10}$$

Suppose there is an attracting set, $\mathcal{G}$[2], arising from iteration of f_Φ. In our methodology, the specific goal of a controller, Φ, is encoded by $\mathcal{G}$, and it follows that the domain of attraction of Φ to $\mathcal{G}$ is given by

$$\mathcal{D}_\Phi(\mathcal{G}) = \{b \in \mathcal{B} | {f_\Phi}^\infty(b) \in \mathcal{G}\}. \tag{11}$$

2.2 The Experimental Apparatus

2.2.1 Mechanism

All of the experimental work described in this paper has been implemented on the Bühgler robot [20, 24] described above. For sensing, this machine is equipped with 2 CCD cameras located approximately 3.5m away from the base of the robot, and 2m above the zero plane. A microphone is attached to the paddle to serve as an impact detector, and provides more precise time measurements of impact events than is possible with the 60Hz frame rate of the cameras [16].

All parameters for the camera positions and orientations, as well as the kinematic offsets for the robot itself, are derived by using off-line parameter estimation during a calibration process undertaken every several months during the operation of the machine [19].

[2] A closed, invariant set $\mathcal{G}$ is attracting if it has the property that there exists an open neighborhood, $\mathcal{N}(\mathcal{G}) \supseteq \mathcal{G}$, such that ${f_\Phi}^\infty(\mathcal{N}) \subseteq \mathcal{G}$.

2.2.2 Sensor System

The basic sensing system is identical to that described in [20] and [17]. However a number of modifications have been incorporated in support of the less structured tasks we wish to explore.

For the current project, the window managing scheme (which is responsible for choosing where in an image to look for balls), has been altered from the original to allow a ball to be thrown into the workspace, rather than "fed" to the machine at a predetermined location. This has led to greater ease in robot operation, and a more unstructured starting environment.

Furthermore, the system has been altered to "prepare" for the next ball shortly after losing sight of the last one. This gives the system the ability to "process" balls at some rate, assuming it is successfully placing them somewhere else at the same rate, or has been programmed to handle more than one.

Finally, note that the observer (used to produce position and velocity estimates from the incoming stream of position measurements), has been modified as well. Whenever the ball reappears after having been absent, the observer uses a "fast and sloppy" method of arriving at the velocity of the ball quickly, then switches into a slower, more accurate mode. This prevents large velocity transients associated with initialization from propagating throughout the reminder of the system. As a result of this change, coupled with the change in window management, it is now possible to throw balls into the workspace (within a limited range), and have the robot successfully handle them, as will be described below.

2.2.3 Actuators

The sensing system provides continuous (500Hz to 1kHz) state estimates, which are fed through a nonlinear transformation to arrive at a desired reference trajectory for the robot. This reference is in turn passed through a smoothing "follow-through" generator to produce a trajectory that has continuous velocity (but not necessarily acceleration), and is in some sense "kind" to the robot [16]. Finally, the output of that is passed to a robot controller. The controllers used here are chosen from the class of inverse dynamics controllers constructed by Whitcomb [27].

For the purposes of this work, we will assume that at the times of impact the observer is successfully tracking the ball and the robot controller is successfully tracking the reference trajectory. Our experiments show that this is very nearly the case.

3 The Constituent Controllers

Following the traditions of Buehler [4], we are interested in programming the robot via "mirror" laws. A mirror law, when properly tracked, guarantees that the state of the robot is strictly a function of the current state of the ball. This section presents our present ideas about generalizing mirror laws to perform additional (other than juggling) "behaviors" as well as a detailed documentation of our empirical success to date in their implementation.

3.1 Mirror Laws as Fundamental Modes

3.1.1 Initial Design

Originally, the system was only programmed with a direct extension of Buehler's mirror law [16, 21]. This took the following form:

$$r = \begin{bmatrix} \phi_b \\ -\frac{\pi}{2} - (\kappa_0 + \kappa_1(\eta - \bar{\eta}))\left(\theta_b + \frac{\pi}{2}\right) + \kappa_{00}(\rho_b - \bar{\rho}_b) + \kappa_{01}\dot{\rho}_b \\ \kappa_{10}(\phi_b - \bar{\phi}_b) + \kappa_{11}\dot{\phi}_b \end{bmatrix}, \tag{12}$$

where ϕ_b, θ_b, and r_b are the cylindrical coordinates of the ball, and η is the ball's *vertical energy* [21]:

$$\eta \stackrel{\triangle}{=} \gamma b_z + \frac{1}{2}\dot{b}_z^2. \tag{13}$$

3.1.2 Modifications

For the purposes of this work, we have added several new mirror-style reference laws in addition to (12) (which we will call *juggling* – Φ_J). These result in *catching* – Φ_C, *palming* – Φ_P, *tossing* – Φ_T, and *placing* – Φ_K.

Juggling, Φ_J; The underlying control policy used to construct our juggling behavior is exactly that of (12).

Catching, Φ_C; We have constructed a preliminary catching behavior based directly on (12) by choosing a set-point which represents an extremely low juggle (on the order of $10cm$). This results in the ball's energy being quickly dissipated while its lateral position is still well regulated[3].

Palming, Φ_P; The machine has been endowed with this capability since its inception [22]. Implementation is accomplished by again re-working the template (12). In this case gains are adjusted such that the hitting portion of the pitch law has been removed. In consequence the machine does not react to the height of the ball, but only its lateral velocity and displacement.

Tossing, Φ_T; This behavior is based on the palming behavior, but follows a predefined trajectory in pitch (and is therefore not a true mirror law) to loft the ball.

Placing, Φ_K; This behavior takes a post-impact vector as a parameter and causes the robot to hit a falling ball in such a manner as to achieve it.

The possibility of switching between mirror laws has led to a second sort of follow-through: the smoothing of the reference trajectory when switching from one to another. This is added after the other, and we will simply consider the two to be parts of the same law.

[3] This in contrast to the work of Buehler in [2], where a "catch" was constructed by causing a zero relative velocity impact between the robot and ball. The difficulty with his approach is that while it does bring the ball to rest on the paddle, it does not bring both to rest at a desired location – both the ball and paddle continue to move at a non-zero velocity after contact is established.

3.2 Implementation

The following data should give the reader a feel for the nature of the regulation and accuracy of the juggling and palming behaviors at various set points around the domain. Since these behaviors will play an important role in the experiments we perform, it is important to demonstrate (at least empirically) that they possess a usefully large domain, and that there is a reasonable region within which we can choose an attracting set point. Because they are still rough and need further development, we have not presented statistical evidence for catching, tossing, or placing behaviors. The catch presently works well enough to build the composite behavior of section 4.

There is little yet that can be said analytically about these different controllers and their domains of attraction or ranges of acceptable "set-points." The one degree of freedom juggling case has been analyzed successfully by Buehler and Koditschek [1], and Rizzi and Koditschek [18] have made progress toward the two and three dimensional cases. We hope to use those results for characterizations of the juggling behavior and, with some work, to apply them to the other behaviors based on similar mirror laws. Lacking useful formal results for either the domains of attraction or ranges of set-points, we are nonetheless encouraged by the empirical results of the following sections and believe in general that conservative estimates of these characteristics can be derived either computationally or experimentally, using experiments similar to those described here.

3.2.1 Juggling at a Set Point

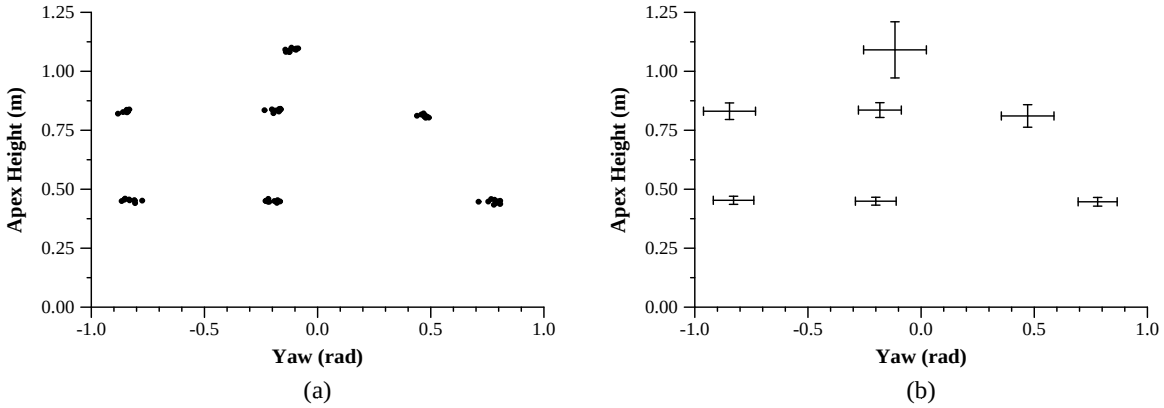

Figure 2: Juggling around a set point. (a) Average apex positions. (b) Error bars showing twice the standard deviation of apex positions.

Several set points varying in yaw and e (vertical energy) were chosen from around the workspace, and ten runs of 30 seconds each (roughly 30 - 60 impacts) were recorded for each point. The radial set point was 0.7m in every run. In figure 2a, we plot the average apex position for each run in yaw and z coordinates, where z is the height of the apex above the paddle[4]. Figure 2b, in the same coordinates, shows the averages of the collections of points in figure 2a with error bars representing twice the average standard deviations[5] of the positions of *all* the apex points of the runs. Thus, for each dimension, 96% of all apex points lie within the error bar shown.

[4] z has a linear relationship with e.

[5] Actually, twice the squareroot of the average variance, which is a slightly larger and more meaningful statistic.

Because the cameras are pointing down at the workspace, the variance grows larger as the energy of the set point increases and the ball starts to leave the visual field near its apex. The trajectories of the ball almost never left the visual field during the runs with the lowest energy, those in the middle were in the field of view approximately 75% of the time, and the highest energy runs were visible only 45% of the time. To handle these occurrences, the observer predicts the trajectory of the ball, and allows the rest of the system to receive uninterrupted ball data. Unfortunately, the ball model in the observer isn't perfect, and the characteristics of the juggle suffer somewhat as a result of not seeing the ball.

Another reason for the errors to increase with energy is that the paddle is not entirely flat. Although every effort was made to design a flat paddle, there are small irregularities which are magnified at higher velocities (the ball is moving at around 5 $\frac{m}{s}$ at contact in the highest energy runs).

3.2.2 Palming at a Set Point

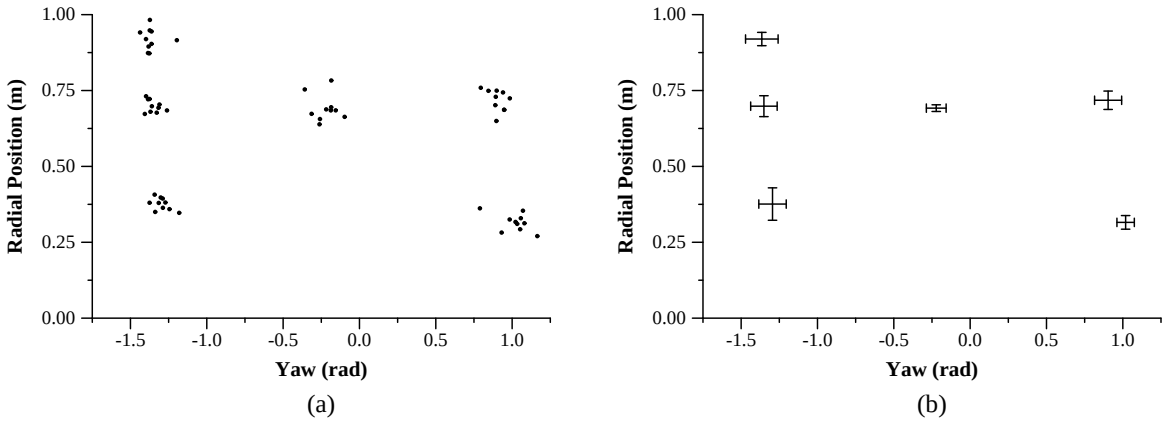

Figure 3: Palming around a set point. (a) Average position of the ball. (b) Error bars showing twice the standard deviation of ball position.

Again, several representative set points were chosen, this time varying in *yaw* and r. For each set point, ten runs of 30 seconds were recorded.

Figure 3a shows the average locations of the individual runs, plotted in *yaw* and r coordinates this time, while figure 3b has error bars representing twice the average standard deviation[6] from the overall averages. This time, there is no particular event, such as an apex, at which to choose a reading, so these statistics are for the entire traces of each run (which are sampled at 60Hz).

The scatter plots show a much larger dispersion for palming than juggling, which we feel is attributable to small irregularities on the paddle, magnified by the very small velocities of the ball near the set point. It is possible for the ball to get caught on a minute bump near the set point and stay there, or be deflected. This is demonstrated particularly by the run at (-0.3, 0.7), where the average positions vary considerably from run to run (See figure 3a, in the middle), but the variance for each run is quite small (See figure 3b,).

[6] see previous footnote

3.2.3 Other Controllers

There are three controllers still "under construction". These are the catch, toss, and place.

Several varients of a catching controller, Φ_C have been developed, the most successful of which uses a combination of open and closed loop references. Since we are moving toward using mirror-style controllers for all of our behaviors, the catch demonstrated in section 4 does not yet behave as well as our earlier one, but is more in line with our philosophy.

Currently, the tossing controller, Φ_T is completely open loop. This is largely due to the inaccuracy of the observer, as we do not model the continuous contact mode between the ball and paddle, and rely on a linear observer instead. In practice, however, this controller adequately performs its duty of getting the ball off the paddle and projecting it roughly straight up to a height of approximately half a meter.

The place, Φ_K, is the hardest of these to implement, as it relies on the accuracy of the robot to achieve a specific post-impact velocity. We presently have a working algorithm, but inaccuracies in the calibration of the robot and noise in the observer signal are making it difficult to tune.

All three of these controllers act roughly in the correct way, and we believe that we will shortly be able to present statistical data documenting them in a fashion that is comparable with the data for the juggling and palming controllers.

4 A Manipulation Sequence

This section presents initial work on autonomous "sequencing" or "chaining" of the behaviors of Section 3 in order to accomplish higher level goals. We believe that the general concepts presented here are sound, but also suspect that certain details may require revision before the logical bugs are completely worked out. These concepts have been implemented in preliminary form for a very simple case documented in section 4.1.2, giving us further reason to believe that we're on the right track.

4.1 Overview

Presuming that we have a "task goal" and a finite set of behaviors, we wish to develop a directed graph based on the idea of *preparation* introduced in 4.1.1. Next, we find a behavior whose goal set coincides with the task goal (for now, we assume that this is possible). Using a recursive algorithm similar to preimage backchaining [12], we then set up a partition of the ball's state space governing the behavior selection by moving away from the goal node of the graph in a breadth-first manner and choosing the appropriate subset of the domain of attraction of each controller encountered.

4.1.1 Preparation Graphs and the Partial Ordering of Behaviors

Say that behavior Φ_1 *prepares* behavior Φ_2 if the goal set of Φ_1 lies within the domain of attraction of behavior 2:

$$\Phi_1 \succeq \Phi_2 \iff \mathcal{G}_{\Phi_1} \subseteq \mathcal{D}_{\Phi_2} \tag{14}$$

Note that this property may be symmetric or transitive, but need not be. Thus, for any set of behaviors, $\mathcal{S} = \{\Phi_1, \Phi_2, \ldots, \Phi_N\}$, there is a directed (possibly cyclic) graph, $\mathsf{G}_\mathcal{S}$ induced by the *prepares* relation. The nodes of $\mathsf{G}_\mathcal{S}$ represent the elements of $\mathcal{S}$, while the links represent the *prepares* relation. In general, the *prepares* relation should not be an easy computation. However, a fair bit of analysis and experience has already been accumulated for the $\mathcal{G}$ and $\mathcal{D}$ pertinent to the classes of controllers we're working with [2, 18], and the conservative estimates that result have proven sufficient to develop the experiments of Section 4.1.2.

If the overall task goal set ($\mathcal{G}$) coincides with the goal set of Φ_i, for some $i < N$, then by starting with Φ_i and recursively tracing the *prepares* relation backwards through the graph, we can find $\mathcal{S}_G \subseteq \mathcal{S}$, the set of all behaviors from whose domains the goal is achievable were the appropriate controllers applied at the correct times.

After finding Φ_i, next choose a partial ordering on $\mathsf{G}_\mathcal{S}$ which transforms it into an acyclic graph with all paths leading to Φ_i. For example, consider a breadth-first search back from Φ_i, as outlined in the following steps:

1. Let the OPEN-LIST contain Φ_i. Let $\bar{\mathcal{D}}_{\Phi_i} = \mathcal{D}_{\Phi_i}$, and $\mathcal{D}_{\mathcal{S}_G}(1) = \mathcal{D}_{\Phi_i}$.
2. Append to the back of the OPEN-LIST the list of all behaviors which *prepare* the first element, and have not previously been placed on the list.
3. Remove the first element of the OPEN-LIST.
4. For Φ_j, the new first element of OPEN-LIST, let $\bar{\mathcal{D}}_{\Phi_j} = \mathcal{D}_{\Phi_j} - \mathcal{D}_{\mathcal{S}_G}$, and let $\mathcal{D}_{\mathcal{S}_G}(N+1) = \mathcal{D}_{\mathcal{S}_G}(N) \cup \mathcal{D}_{\Phi_j}$.
5. Repeat 2,3 and 4 until OPEN-LIST is empty.

At the end of this process, the regions $\bar{\mathcal{D}}_{\Phi_l}$ will be cells in a partition of $\mathcal{D}_{\mathcal{S}_G}(M)$, where M is the number of cells in the partition. The automaton will choose behavior Φ_j exactly when the ball state lies within $\bar{\mathcal{D}}_{\Phi_j}$.

The technique proposed here is a variant of the preimage backchaining idea introduced by Lozano-Perez, Mason, and Taylor [12], although their algorithm was used for compliant motion in the face of velocity uncertainty with respect to a generalized damper model of dynamics. We choose to substitute sensory events for the physical transitions that characterized their control sequences. The domain of each behavior in $\mathcal{D}_{\mathcal{S}_G}$ is partitioned in such a manner that the automaton will only switch to a new behavior if the ball moves into the domain of a behavior which was processed earlier from OPEN-LIST. Since the attracting goal set of each behavior lies in the domain of another, it is inevitable that $\bar{\mathcal{D}}_{\Phi_i}$ finally be reached, and thus the goal set will be approached.

The domain of attraction for the goal set can now be considered to be not just $\mathcal{D}_{\Phi_i}$, but $\mathcal{D}_{\mathcal{S}_G} = \bigcup_{\Phi \in \mathcal{S}_G} \mathcal{D}_\Phi$.

4.1.2 Sensor Based Implementation

Using $\mathsf{G}_\mathcal{S}$ and proceeding in the manner described above, divide $\mathcal{S}$ into $\mathcal{S}_\mathcal{G}$, the set of behaviors that can lead to the goal, and $\bar{\mathcal{S}}_\mathcal{G}$, the set of behaviors which can not. Say that all behaviors in $\bar{\mathcal{S}}_\mathcal{G}$, are OFF, and those in $\mathcal{S}_\mathcal{G}$ are ON. Further divide the ON behaviors into the one active behavior for the current epoch, which is ACTIVE, and the rest, which are INACTIVE.

Using the partition constructed in the previous section and the output from the sensor, one may now decide at the beginning of each epoch upon a behavior (from ON) to be followed for the duration of that epoch.

$$\Phi_{k+1} = \Phi_i \qquad if \quad b_k \in \bar{\mathcal{D}}_{\Phi_i}. \tag{15}$$

When the robot and ball are in continuous contact, we get a new epoch every update from the sensors.

4.2 Mode Switching: Tying Behaviors Together

Having shown that our behaviors are reasonably robust in Section 3.2, we now move on to tying them together to achieve a more abstract goal according to the method described above. The task is simply to bring the ball to rest on the paddle at a desired location, $(-0.3, 0.7, 0.0)$. There are no obstacles, and we have a controller, Φ_P, whose goal set is exactly the task goal point. In addition, we choose our catching behavior to have a goal at the goal point, and our juggle set point is 0.8m above the goal point. This small set of possible behaviors was obviously chosen to be advantageous to the task. At this stage we merely seek to test the feasibility of our abstract algorithm.

The choice of behavioral mode at any given moment is entirely based on the state of the ball according to the partition algorithm described above. In this simple case it is obvious how to "tune" the individual controllers such that the ball's phase space is divided into three cells, each of which will be associated with a particular controller from the state Φ_J, Φ_C, and Φ_P. The implementation of this partition is still fairly *ad hoc*, based upon our crude approximations for the domains of attraction, $\mathcal{D}$, for the various behaviors. Palming is only acceptable when the ball has little vertical motion, and is low. Thus we define $\eta = gz + \frac{1}{2}\dot{z}^2$, and the domain for palming is given by $\eta < K_1$.

Catching, on the other hand, can handle much larger vertical energies (but not every magnitude), but is sensitive to lateral errors in both position and velocity. Thus we define $\phi(b) = \sqrt{(x - x_g)^2 + (y - y_g)^2}$, and $\kappa(b) = \sqrt{(\dot{x} - \dot{x}_g)^2 + (\dot{y} - \dot{y}_g)^2}$, where x_g, y_g, etc. are understood to be the Cartesian coordinates of the goal point. The domain for catching mode is given by $\eta < K_2$, $\phi(b) < K_3$, and $\kappa(b) < K_4$.

For convenience, we will let the domain of juggling be the entire state space of the ball that is reachable by the robot.

All of the cut-off constants, K_i, are chosen by educated guess, as described above. Following the algorithm outlined in the previous section, we start with Φ_P and backchain through our small set of behaviors. In the language of section 3, we can now say:

- $\bar{\mathcal{D}}_P = \mathcal{D}_P = \{b \in \mathcal{B} | \eta(b) < K_1\}$
- $\mathcal{G}_P = (-0.3, 0.7, 0.0) = \mathcal{G}$
- $\bar{\mathcal{D}}_C = \{b \in \mathcal{B} | K_1 < \eta(b) < K_2, \phi(b) < K_3, \kappa(b) < K_4\}$
- $\mathcal{G}_C = (-0.3, 0.7, 0.0) = \mathcal{G}_P \in \mathcal{D}_P$
- $\bar{\mathcal{D}}_J = \{b \in \mathcal{B} | K_1 < \eta(b), \phi(b) > K_3 \quad or \quad \kappa(b) > K_4\}$

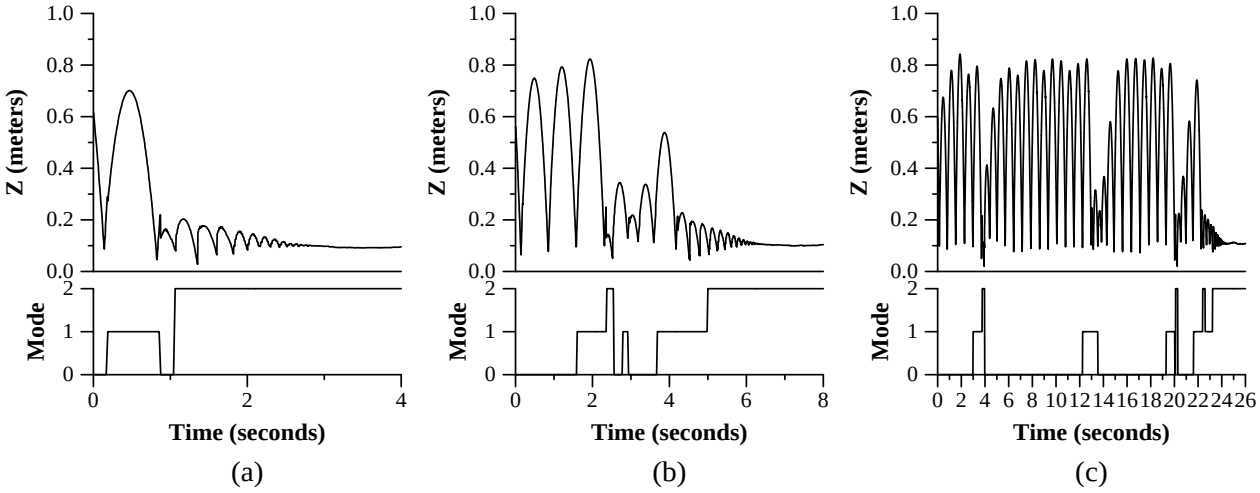

Figure 4: Behavior switching. (a) A quick example. (b) Several mode switches. (c) four attempts before success.

- $\mathcal{G}_J = (-0.3, 0.7, 0.8) \in \bar{\mathcal{D}}_C$

In figure 4, we show three traces of the vertical position of the ball as a function of time, coupled with the behavioral mode of the system. In the mode traces, 0 represents juggling, 1 catching, and 2 is the palming mode.

In figure 4a, the ball is dropped into the workspace from above the goal point. As a result, the only criterion keeping the system in juggling mode is the energy maximum for the catch. Immediately after the first collision, the energy is low enough for the system to try to catch the ball. At the second collision, however, the observer predicts the post-impact conditions incorrectly, which sends the system back to juggling (due to lateral error, not shown). The observer quickly corrects to the actual ball trajectory, which has low enough η to send the system straight to palming, where the ball quickly loses its remaining energy.

In figure 4b, the ball is not introduced so nicely, and the robot juggles it a few times before the lateral errors are small enough for an attempted catch. This time, the observer takes too long to notice the second bounce, predicting that the ball has fallen further than it did in reality. Thus, η falls low enough for palming to be switched on, but the revised ball data quickly send the system back to juggling. The lateral errors briefly dip low enough for the system to switch to catching, then back to juggling, but finally catching turns on, and then palming.

In figure 4c, the system tries to catch three times and fails before finally bringing the ball down to the paddle on the fourth attempt. Once the system is in palming mode and the ball has settled on the paddle, it will remain there.

The third example shown here demonstrates the robustness of this approach to planning. The system has no memory, and no "plan" in the traditional sense. It will jump from any node to any other based on the sensed ball state, regardless of how or whether those nodes are connected in $\mathsf{G}_{\mathcal{S}}$. A traditional planner might need pre-programmed exception handling to go from palming mode back to catching or juggling, as such jumps violate the world model. The physical stability of each of our behaviors guarantees that the system will continue to be inexorably drawn toward the goal state regardless of any unexpected changes in state.

5 Controller Deployment

5.1 Automatic Tuning

For each behavior, Φ_i, there are parameters which must be adjusted: set points, gains, etc. We will refer to each vector of parameters as $\xi_i \in \Xi_i$, and denote the dependence of Φ_i upon ξ_i by $\Phi_i(\xi_i)$. Call a particular $\xi \in \Xi = \Xi_1 \times \Xi_2 \times \ldots \times \Xi_M$ a *deployment*.

It may be the case that some set of initial states important to us is not within $\mathcal{D}_{\mathcal{S}_G}$. For example, we might desire global attraction to $\mathcal{G}$. In such cases our only option is to *tune* the parameters of our behaviors. Such adjustments tend to lead to different domains for the behaviors (for instance, changing the control gains might collapse the domain to a single point!), and may likely lead to different topologies for the graph $\mathsf{G}_{\mathcal{S}}$.

We would like to develop a method for moving the deployment within Ξ in such a manner as to alter the connectivity of $\mathsf{G}_{\mathcal{S}}$ in our favor. It seems likely that G is locally constant in Ξ, but that significant perturbation will change its topology –perhaps favorably, perhaps not. In the long run, we hope to develop some insight into an offline deployment algorithm, and ultimately an online algorithm for doing so as well.

The question arises as to how many different copies of (essentially) the same behavior are desirable. The set point for palming, for example, may be placed almost anywhere in the workspace. Each set point technically gives rise to a new behavior, yet it seems natural to consider the entire set as a single behavior with a varying set point. We would like to be able to reason about driving the set point for a single behavior around the workspace, an activity we demonstrate in the following section. For example, if the workspace is divided by obstacles into many similar cells (such as a maze), it seems much more natural to have a few behaviors that are tuned depending on which obstacle cell the system is in, rather than a new behavior from each class in each cell.

It may also be the case that our task goal is not contained within the goal set of any particular behavior, or that it is described in terms of a limit cycle of parallel behaviors (such as two-juggling). In such cases the first problem may be finding a representation for the goal set.

We find the fact that different regions in Ξ give rise to distinct classes of behavior (such as juggling or palming) to be intriguing and exciting. Although we don't understand in what manner, we believe that there is a connection between navigation within Ξ and rudimentary artificial intelligence.

5.2 Palming Around a Moving Set Point

Here we demonstrate the ability of the palming behavior to follow a moving set point. We present this meta-behavior to demonstrate that we are moving toward the ability of online deployment, as discussed above. Although we have no formal analysis, it seems clear that the palming behavior is robust to this sort of activity.

Having shown that the palming behavior has a healthy steady-state about a fixed point, we now start moving the set point and require the palming algorithm to track it. It should be emphasized that the algorithm is still simply trying to regulate about a set point, and has no "idea" that the set point might be in motion. We could have given anecdotal evidence similar to these figures for the juggling behavior, but chose to show only the palming traces due to space constraints.

The trajectory of the set point is a rectangular loop in joint space, with two sides at constant yaw (at -1.3 and 0.4), and the other at constant r (at 0.4 and 0.8). Because this behavior is difficult to capture with averages or error bars, we demonstrate it with two anecdotal traces of four minutes each. Note that these figures clearly show the annular nature of the actual robot's workspace.

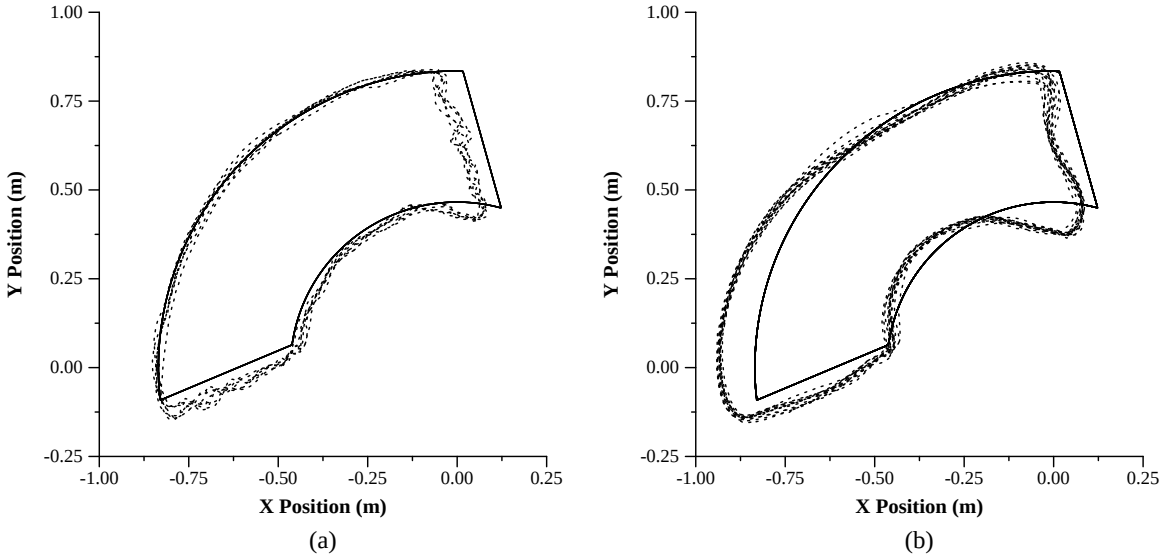

Figure 5: Palming around a moving set-point: (a) a slow reference trajectory, (b) a fast reference.

In figure 5a, the robot is asked to regulate the ball to a set point moving clockwise around the loop drawn in solid lines. The constant-r sides take 10 seconds each, and the constant-yaw sides take 7 seconds each. The actual trajectory of the ball is shown by the dotted lines. In figure 5b, the set point is following the same pattern, but this time the constant-r sides take only 5 seconds, and the other sides are 3 seconds.

At the slower rate, the robot is able to keep the ball close to the desired track, but allows the ball to wander a little. On the other hand, because the ball is moving faster in figure 5b, there is considerable overshoot in the radial direction, but the ball trajectories are more uniform. The radial overshoot is exacerbated by coriolis forces at the higher velocities, and the robot has weaker control over the ball in that direction than in the angular direction. These plots also demonstrate a systematic error in the yaw position of the ball relative to the set point. This error is also evident in the other plots (e.g., the column in the middle of figure 2a should all have $yaw = -0.3$), and can be attributed to a zeroing error in the third (roll) degree of freedom.

6 Conclusion: Looking Ahead to Dynamical Obstacle Avoidance

Although we have alluded to the presence of obstacles in this paper, the reader will have observed that there has been as yet no mention of how to cause a ball to avoid them. We are indeed developing versions of each of the controllers mentioned in this paper that will perform a similar manipulation primitive yet insure that the ball does not penetrate an obstacle at the same time.

Since these controllers are not yet complete at the time of this writing, we have no empirical results to report. For the time being, we simply report on the notion of *safety* that provides the key ingredient in their design.

6.1 Obstacles and Safety

Let the set of all ball states which are unacceptable be denoted by $\mathcal{O}$. This set includes the positions where the ball is in a physical obstacle (or close enough that the robot can not manipulate it without penetrating the obstacle), as well as all states where the ball is beyond the cylinder of the workspace, or too fast to be viewed by the current vision system.

We define the time of flight until the ball hits the obstacle by

$$\tau_{\mathcal{O}}(b) = \min t : F^t(b) \in \mathcal{O}. \tag{16}$$

The time to impact for a particular behavior, Φ, is given by [7]

$$\tau_{\Phi}(b) = \tau_k - (\pi_{\dot{z}} b_k - \pi_{\dot{z}} b)/g. \tag{17}$$

We call a behavior Φ *safe* if there can be found an open set $\mathcal{D}_{\Phi S} \subseteq \mathcal{D}_{\Phi}$ satisfying the following criteria:

$$\mathcal{D}_{\Phi S} \subseteq \{b \in \mathcal{D}_{\Phi} | \tau_{\Phi}(b) < \tau_{\mathcal{O}}(b)\} \tag{18}$$

$$f_{\Phi}(\mathcal{D}_{\Phi S}) \subseteq \mathcal{D}_{\Phi S} \tag{19}$$

Assuming that the robot is acting according to Φ, then the first criterion requires that the domain not contain any ball states from which the trajectory of the ball will lead to the obstacle before the next contact. The second criterion – invariance to impacts – implies that once in the safe domain the robot will never knock the ball out (again assuming that the robot continues in Φ). Together, these criteria insure that once a ball is in $\mathcal{D}_{\Phi S}$, it will neither leave nor hit an obstacle as long as the robot remains in Φ.

6.2 Future Work

The previous definition is not utterly vacuous, as it is easy to provide examples of safe domains for certain controllers faced with certain types of obstacles. Consider, for example, the juggler, Φ_J, and the domain, V of all those ball states that are positioned directly at the correct horizontal position of the specified juggle apex with no horizontal velocity. We have demonstrated in previous work [18] that V is an invariant submanifold, $f_{\Phi_J} V \subset V$. Moreover, it is clear that no ball in V will ever hit an obstacle that does not intersect the vertical line through the specified apex point. Thus, V is a safe domain for our existing juggler. Obviously, V is a disappointingly small set and we would be hard pressed to achieve any safe progress toward horizontally displaced goals were that our only safe behavior. What could be done with a larger safe domain and more safe behaviors?

[7] We use $\pi_{\dot{z}}(b)$ to refer to the vertical velocity component of b.

If one re-reads this paper and substitutes for each Φ the safe version, Φ_S and substitutes for D_Φ the induced restriction dynamics $f_\Phi|_{\mathcal{D}_{\Phi S}}$, then all the same definitions and algorithmic procedures are applicable. Clearly, however, the domains, $D_{\Phi S}$, can be expected to be much smaller, and the corresponding problem of deployment much more difficult.

In the near future we plan to consider the "obstacle" of all those ball states that can never again be batted because they are out of reach, or unalterably becoming so. We will attempt to build a "containing" version of the sequenced manipulation of section 4. In this scenario, a ball would be thrown into the workspace and the robot would first insure that it is brought "under control" so as not to fly out of reach before settling into its descent toward the palming goal. Moreover, the robot would be capable of containing balls that we try to poke out of its workspace.

More realistic obstacles will raise even more interesting versions of the deployment problem. We hope to work on doors and windows in the near future.

References

[1] M. Bühler and D. E. Koditschek. From stable to chaotic juggling. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1976–1981, Cincinnati, OH, May 1990.

[2] M. Bühler, D. E. Koditschek, and P. J. Kindlmann. Planning and control of a juggling robot. *International Journal of Robotics Research*, 13(2):101–118, 1994.

[3] M. Bühler, D. E. Koditschek, and P.J. Kindlmann. A family of robot control strategies for intermittent dynamical environments. *IEEE Control Systems Magazine*, 10:16–22, Feb 1990.

[4] Martin Bühler. *Robotic Tasks with Intermittent Dynamics*. PhD thesis, Yale University, New Haven, CT, May 1990.

[5] Bruce R. Donald. *Error Detection and Recovery for Robot Motion Planning with Uncertainty*. PhD thesis, MIT, 1987.

[6] M. A. Erdmann and M. T. Mason. An exploration of sensorless manipulation. *IEEE J. Rob. and Aut.*, 4(4):635–642, Aug 1988.

[7] Michael Erdmann. Towards task-level planning: Action based sensor design. Technical Report CMU-CS-92-116, Carnegie Mellon University, Pittsburgh, PS, Feb. 1992.

[8] Michael A. Erdmann. On motion planning with uncertainty. Technical Report AI-TR-810 (MS Thesis), MIT AI Lab, Aug 1984.

[9] Tomas Lozano-Pérez et al. Handey: A robot systems that recognizes, plans, and manipulates. In *Proceedings IEEE Int. Conf. Robotics and Aut.*, pages 843–849, 1987.

[10] Daniel E. Koditschek. Robot control systems. In Stuart Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 902–923. John Wiley and Sons, Inc., 1987.

[11] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[12] Tomás Lozano-Perez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, 3(1):3–23, 1984.

[13] D. M. Lyons and A. J. Hendriks. Planning by adaption: Experimental results. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 855–860, 1994.

[14] Damian M. Lyons. Representing and analyzing action plans as networks of concurrent processes. *IEEE Transactions on Robotics and Automation*, 9(3):241–256, June 1993.

[15] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, Jan 1987.

[16] A. A. Rizzi and D. E. Koditschek. Further progress in robot juggling: The spatial two-juggle. In *IEEE Int. Conf. Robt. Aut.*, pages 3:919–924, May 1993.

[17] A. A. Rizzi and D. E. Koditschek. An active visual estimator for dexterous manipulation. *IEEE Transactions on Robotics and Automation*, (submitted), 1994.

[18] A. A. Rizzi and D. E. Koditschek. Further progress in robot juggling: Solvable mirror laws. In *Int. Conf. Rob. and Aut.*, pages 2935–2940, 1994.

[19] Alfred Rizzi and Daniel E. Koditschek. Preliminary experiments in robot juggling. In *Proc. Int. Symp. on Experimental Robotics*, Toulouse, France, June 1991. MIT Press.

[20] Alfred A. Rizzi. *Dexterous Robot Manipulation.* PhD thesis, Yale University, 1994.

[21] Alfred A. Rizzi and D. E. Koditschek. Progress in spatial robot juggling. In *IEEE Int. Conf. Robt. Aut.*, pages 775–780, Nice, France, May 1992.

[22] Alfred A. Rizzi and Daniel E. Koditschek. Preliminary experiments in robot juggling. In *IEEE International Conference on Robotics and Automation Video Proceedings*, Sacramento, CA, USA, 1991.

[23] Alfred A. Rizzi and Daniel E. Koditschek. Dynamically dexterous robotics: The two-juggle. In *IEEE International Conference on Robotics and Automation Video Proceedings*, Atlanta, GA, USA, 1993.

[24] Alfred A. Rizzi, Louis L. Whitcomb, and D. E. Koditschek. Distributed real-time control of a spatial robot juggler. *IEEE Computer*, 25(5), May 1992.

[25] J. L. Synge and B. A. Griffith. *Principles of Mechanics.* McGraw Hill, London, 1959.

[26] C. P. Tung and A. C. Kak. Integrating sensing, task planning and execution. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 2030–2037, 1994.

[27] Louis L. Whitcomb, Alfred Rizzi, and Daniel E. Koditschek. Comparative experiments with a new adaptive controller for robot arms. *IEEE Transactions on Robotics and Automation*, 9(1):59–70, 1993.

[28] D. E. Whitney. Force feedback control of manipulator fine motions. *ASME J. Dyn. Syst.*, 98:91–97, June 1977.