Proceedings of the 1998 International
Conference on Robotics and Automation
May 1998 Leuven Belgium

Hybrid Control as a Method for Robot Motion Programming

Alfred A. Rizzi

Microdynamic Systems Laboratory
The Robotics Institute
Carnegie Mellon University
arizzi@cs.cmu.edu

Abstract

This paper presents a class of fundamental control policies suitable for use in a novel method for designing and specifying the dynamic motion of robotic systems. Through recourse to formal stability mechanisms a hybrid control strategy (one that relies on switching of underlying continious policies) with desirable performance characteristics is demonstrated to be both stable and expressive for programming such motions. The long term intent is to utilize slightly more general control methods of this form to drastically simplify the process of integrating and programming modular automated assembly systems.

1 Introduction

It has long been a goal of the robotics and automation community to provide tools that will simplify the design, development, and programming of *intelligent* systems and mechanisms. And while much progress has been made on the development of new and more capable mechanisms, there has been only minimal progress at providing new paradigms for programming or *instructing* these mechanisms. The ideas presented here are complimentary to some early ideas on task level programming of dynamic tasks [2, 1], but focus instead on how collections of controllers can be used to simplify the task of programming the behavior of a generic mechanism.

As a component of a long term project (minifactory[1] [5]) which is focused on the development of modular robotic components and tools to support the rapid deployment and programming of high-precision assembly systems, the work presented here targets the most basic levels of a modular control and coordination architecture which is central to the larger project. The overall intent is to provide mechanically, computationally, and algorithmically modular factory *agents* and a collection of tools to support a users interaction with the agents thus facilitating the process of designing, integrating, and programming the factory system. This paper addresses the problem of developing flexible low level programming models and controllers to simplify the often tedious task of designing, programming, and controlling trajectories for robotic factory elements.

The ideas presented here offer a novel model of "robot programming" based on *deploying* a set of controllers, each with an associated domain and goal, to describe the motion of a factory agent. This is in sharp contrast to the traditional approach of designing parameterized trajectories and relying on local control policies to track these trajectories. While widely accepted as an effective means for programming robot motion, this traditional approach is often difficult to "tune" and lacks any inherit ability to gracefully recover from a large classes of potential disturbances — for example a controller that "falls behind" a time parameterized trajectory may well saturate its actuators in an attempt to "catch up," often resulting in significant wear and tear or worse yet instability. Under the model proposed here, which is purely feedback based and thus not explicitly dependent on an arbitrary notion of time, performance comparable to trajectory planning can be ensured for the nominal operating case, while also allowing for the graceful recovery from disturbances.

Rather than generating trajectories through the free configuration space of the robotic agent, the programmer will be responsible for decomposing the free configuration space into overlapping polygonal regions and parameterizing controllers associated with each region.

[1] See http://www.cs.cmu.edu/~msl

A *hybrid control* system is then responsible for switching or sequencing between the control policies associated with this decomposition to achieve the desired overall goal. This provides a programming model that allows low level behavior to be managed and executed by the mechanisms while allowing high-level behavior to be sequenced and guided by the programmer.

2 Sequencing Controllers

The central notion behind this scheme is to describe the behavior of any one agent in terms of a collection of feedback strategies based on the state of both the individual agent and its immediate peers. The result is a hybrid on-line control policy (one that discretely switches between various continuous policies) which makes use of the entire collection of available controllers to systematically make progress toward a goal based on an agent's estimate of both its own and its peers' state. To provide the desired system level flexibility the selection of specific goals and their sequencing is left to the factory programmer.

More formally, given a set of controllers, $\mathcal{U} = \{\Phi_1, ..., \Phi_N\}$, each with an associated goal, $\mathcal{G}(\Phi_i)$, and domain,$\mathcal{D}(\Phi_i)$ — where it is presumed that under the action of Φ_i any state that starts in $\mathcal{D}(\Phi_i)$ will be taken to $\mathcal{G}(\Phi_i)$ without leaving the set $\mathcal{D}(\Phi_i)$. We then say that controller Φ_1 *prepares* controller Φ_2, denoted $\Phi_1 \succeq \Phi_2$, if its goal lies within the domain of the second $\mathcal{G}(\Phi_1) \subset \mathcal{D}(\Phi_2)$ [1]. For an appropriately parameterized set of controllers, $\mathcal{U}$, this relation induces a generally cyclic directed graph. Assuming that the overall goal, $\mathcal{G}$, coincides with the goal of at least one controller, $\mathcal{G}(\Phi_i) = \mathcal{G}$, then by starting with Φ_i and recursively tracing the relation backwards through the corresponding graph, one arrives at $\mathcal{U}_\mathcal{G} \subset \mathcal{U}$ — the set of all controllers from whose domains the overall goal might be eventually reached by switching between control policies. The domain of a properly conceived composite controller, should then be $\bigcup_{\Phi \in \mathcal{U}_\mathcal{G}} \mathcal{D}(\Phi)$, and thus we have an "automatic" method by which to guide the system from any state in this union of domains to the goal.

Consider, for example, the trivial planar configuration space depicted in Figure 1. Here we can see that the user has chosen to decompose the free space into four separate regions with the overall goal located in the upper right corner of the configuration space ($\mathcal{G}_4$). Here, Φ_1 is responsible for for taking all states in the lower convex region to $\mathcal{G}_1$, and thus prepares Φ_2. Similarly the placement of $\mathcal{G}_2$ and $\mathcal{G}_3$ allow both Φ_2 and Φ_3 to prepare Φ_4 which regulates the state to

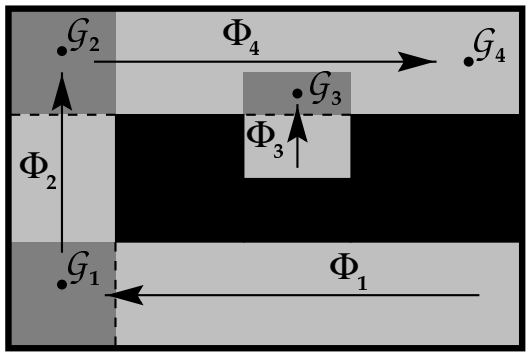

Figure 1: Example decomposition of a trivial planar configuration space.

$\mathcal{G}_4$, the overall goal. It is the responsibility of the hybrid control system to systematically switch between the individual Φ_i in order to achieve the overall task goal. It is this discrete switching of underlying continuous control policies which makes this a "hybrid" control strategy and is discussed in detail in [1]. While this trivial example is illustrative it is important to note that we have only considered the configuration of the system here, while in general the actual domains, $\mathcal{D}(\Phi_i)$, for the constituent controllers are defined over the state space of the system — *i.e.* the positions and velocities.

The remainder of this paper will focus on developing the individual controllers, Φ_i, which are responsible for efficiently taking as large a possible region of the state space to the local goal $\mathcal{G}_i$.

3 Component Controllers

We will consider the second order system governed by

$$\ddot{x} = u, \tag{1}$$

with $u, x \in \mathbb{R}^n$. This model can most easily be thought of as a unit mass in n-dimensions to which arbitrary acceleration can be applied, although for simplicity we will use it to stand in as the dynamics for an arbitrary mechanism under our control. Furthermore we will subject (1) to the constraints

$$\begin{aligned} \|u\| &\leq \mathrm{U}_{\max} && (2)\\ \|\dot{x}\| &\leq \mathrm{V}_{\max} && (3)\\ x(t) &\in \mathcal{P} \quad \forall t \geq 0 && (4)\\ & \mathcal{P} := \{x \in \mathbb{R}^n \mid \beta_i(x) \geq 0\} \quad \forall i \in \{1 \ldots N\}\\ & \beta_i := l_i^T x - c_i, \end{aligned}$$

where

$$\|l_i\| = 1 \quad \forall i \in \{1 \dots N\}$$
$$l_i^T l_j \neq 1 \quad \forall i \neq j \in \{1 \dots N\}$$
$$N \geq n + 1$$

and $\mathcal{P}$ is presumed to be non empty. These constraints specify, respectively, that there is a maximum acceleration that can be applied to the body, a maximum velocity which can not be exceeded, and finally and most importantly that there is a convex polygonal region from which x should not depart. This final constraint corresponds directly to the decomposition introduced earlier.

The problem is to design a family of controllers which takes the largest possible set of initial states to to an arbitrary point in the interior of $\mathcal{P}$ at zero velocity, $(x^*, 0)$. Furthermore if these controllers are to be considered for application in any real system they should afford a reasonable level of performance, meaning that they well utilize the available velocity and acceleration limits to accomplish this task.

3.1 The Savable Set and Stopping (Φ_S)

The first problem is to explicitly decide what initial conditions from $T\mathcal{P} := \{(x, \dot{x}) | x \in \mathcal{P}, \dot{x} \in I\!R^n\}$ can possibly be brought to the destination, $(x^*, 0)$. To begin with, recognize that any state which can be brought to rest within $\mathcal{P}$ can eventually be driven to $(x^*, 0)$ by recourse to any number of conservative and simple control strategies. Thus we reduce this problem to one of finding those initial conditions in $T\mathcal{P}$ which can be brought to rest without violating the constraints of (2)-(4). This set of states will be referred to as the *savable set*, $\mathcal{S}$, and is $\mathcal{D}(\Phi_S)$.

Consider a controller which acts to *push* x away from the nearest (in time) boundary of $\mathcal{P}$. Begin by determining the *time to impact* with boundary component i by

$$\delta_i := -\frac{l_i^T x - c_i}{l_i^T \dot{x}}, \tag{5}$$

this is the time it will take for x to satisfy $l_i^T x - c_i = 0$ and thus depart the interior of $\mathcal{P}$ if no input is applied. Discarding the non-positive and infinite δ_i, then selecting the set of indices corresponding to the boundary components with the minimum time to impact yields a set, $\mathcal{I}$, of no more than n boundary components. The *stopping* controller, Φ_S, is then given by $u = 0$ if $\mathcal{I} = \emptyset$ and otherwise

$$u = -\frac{\sum_{j \in \mathcal{I}} l_j l_j^T \dot{x}}{\| \sum_{j \in \mathcal{I}} l_j l_j^T \dot{x} \|} \mathrm{U}_{\max}. \tag{6}$$

Intuitively this control policy uses the maximum available acceleration to push x away from the boundary component it will strike first. Presuming that $\dot{x}$ is not aligned with the corresponding l_i (*i.e.* traveling straight at the boundary component) there will be a time when two boundary components will have equally small positive δ_i's (*i.e.* $\dot{x}$ will be heading toward a *corner*), at this point $\mathcal{I}$ will contain two elements and u will be directed to apply the maximum acceleration away from the intersection of these two boundary components. This process continues until $\dot{x}$ reaches zero.

Proposition 1 *The controller specified by (6) brings to rest every initial condition in $T\mathcal{P}$ which can be brought to rest without violating (2)-(4).*

Proof: Consider that there existed a state, $(x, \dot{x})$, which can not be brought to rest inside of $\mathcal{P}$ by (6), but can be brought to rest by another controller, **C**. Note that under the action of (6) any violation of the boundaries of $\mathcal{P}$ must involve a violation of the boundary component which has the smallest time to impact for the initial state. Further note that (6) uses all of the available acceleration to maximize the rate of increase of δ_i for the "nearest" boundary component(s). So **C** must not have this property and thus must not increase the time to impact with the initially "nearest" boundary component more quickly than (6). But (6) was unable to prevent the violation of that boundary component and we conclude neither was **C**.

□

3.2 Velocity Regulation (Φ_V)

The second problem is to design a family of controllers capable of taking any point at rest in $\mathcal{P}$ to the specified destination, again respecting the constraints, (2)-(4). While the generation of a solution of this problem is not fundamentally difficult, it is imperative that the trajectories generated by these controllers be efficient — *i.e.* they are near minimum time trajectories and provide a natural parameterization for such things as approach velocity, stiffness, and damping.

Begin by considering a measure of *distance* to the target position of the form

$$\gamma(x) := \frac{\|x - x^*\|^2}{\|x - x^*\| + \alpha} \tag{7}$$

where x^* is the desired rest position for the system. This function is both positive definite about x^* and

has a bounded gradient. Figure 2 graphs the value of γ and its gradient for scalar x and a particular choice of α and demonstrates how the parameter α is used to smooth the transition from -1 to 1 in the vicinity of $\|x - x^*\| = 0$.

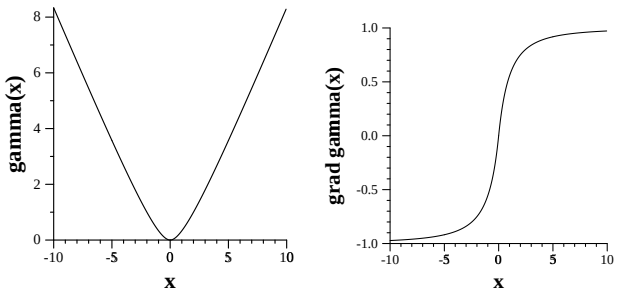

Figure 2: Graph of γ and its gradient with $\alpha = 5.0$ and $x^* = 0$.

Intuitively it would be attractive to use the transposed Jacobian (gradient) of γ, $D_x\gamma^T$, as a position dependent velocity command for (1), for example in a controller of the form

$$u = -\left(V^* D\gamma^T + \dot{x}\right) K - V^* D^2\gamma\, \dot{x}, \tag{8}$$

where $0 < V^* < \mathrm{V_{max}}$ specifies the nominal approach velocity, $K > 0$ serves as a "velocity regulation" gain, and the Hessian of γ, $D_x^2\gamma$, is used to effectively feedforward the necessary acceleration to track the desired position dependent velocity profile. The stability of this controller in the absence of any external constraints can be seen be considering

$$\eta_\gamma = V^* K\gamma(x) + \frac{1}{2}\dot{x}^T\dot{x} \tag{9}$$

as a candidate Lyapunov function for (1) under the influence of (8), about x^*. Taking the time derivative of (9) along the trajectory of the system yields.

$$\dot{\eta_\gamma} = V^* K D\gamma\, \dot{x} + \dot{x}^T u. \tag{10}$$

Finally substituting (8) gives

$$\begin{aligned}\dot{\eta_\gamma} \;=\; & V^*\left(KD\gamma\,\dot{x} - \dot{x}^T D\gamma^T K\right) \\ & -\dot{x}^T K\dot{x} - \dot{x}^T V^* D^2\gamma\,\dot{x}.\end{aligned} \tag{11}$$

The term in parenthesis is identically zero, and the remaining two terms are negative semidefinite in the state (they do not include x), this follows since K was defined to be positive, and the Hessian of γ is positive definite. Thus we can immediately conclude that $\lim_{t\to\infty}\dot{x} = 0$. And finally by recourse to LaSalle's Invariance Principle [3] that in fact $\lim_{t\to\infty} D_\gamma = 0$ and thus γ must converge to 0 while x converges to x^*.

Proposition 2 *Under the influence of (8) the surface* $\mathcal{V} := \left\{(x,\dot{x}) \mid V^* D\gamma^T + \dot{x} = 0\right\}$ *is both attractive and invariant.*

Proof: Consider a candidate Lyapunov of the form

$$\eta_v = \frac{1}{2}\left(V^* D\gamma^T + \dot{x}\right)^T\left(V^* D\gamma^T + \dot{x}\right). \tag{12}$$

Again, taking the derivative along the trajectories of (1) under the control of (8) yields

$$\dot{\eta_v} = \left(V^* D\gamma^T + \dot{x}\right)^T\left(V^* D^2\gamma\,\dot{x} + u\right), \tag{13}$$

Which simplifies to

$$\dot{\eta_v} = -\left(V^* D\gamma^T + \dot{x}\right)^T\left(V^* D\gamma^T + \dot{x}\right) K, \tag{14}$$

and allows us to conclude that $\dot{x}$ does indeed converge to $-D\gamma\,(x)^T$.

□

The problem remains to demonstrate that appropriate parameterizations of this controller, (8), can be chosen to ensure that the constraints (2)-(4) are not violated by this policy over a reasonable domain.

First consider (3). The conditions under which this constraint are maintained can be examined be evaluating the time derivative of $\frac{1}{2}\dot{x}^T\dot{x}$ when $\|\dot{x}\| = \mathrm{V_{max}}$. This evaluates to

$$-\dot{x}^T\dot{x}K - \dot{x}D^2\gamma\,\dot{x} - KD\gamma\,\dot{x}. \tag{15}$$

and allows us to conservatively conclude that this control policy will not violate (3) whenever

$$D\gamma\,\dot{x} \geq 0. \tag{16}$$

Intuitively this implies that, with respect to this constraint, it is safe to utilize this controller whenever the current velocity is roughly in the same direction (has a positive inner product with) the "reference velocity," $D\gamma^T$.

Next consider (2). By inspecting (8) it is clear that at very least we must ensure that $V^*\|D^2\gamma\, D\gamma^T\| \leq \mathrm{V_{max}}\ \forall x \in \mathcal{P}$ to guarantee that (2) is not violated while tracking the surface $\mathcal{V}$. This yields an inequality constraint on α which dictates that α be chosen large enough to allow ample room for deceleration from V^* in the vicinity of x^*. Furthermore there is a tradeoff between the choice of the gain K and the extent of the domain over which (8) can be applied without violating (2). This simply takes the form

$$\|\left(V^* D\gamma^T + \dot{x}\right) K + V^* D^2\gamma\,\dot{x}\| \leq \mathrm{U_{max}}. \tag{17}$$

Finally we must consider the conditions under which (8) can be guaranteed to not violate the boundary of the convex region defined by (4). This is the most difficult problem in verifying the applicability of (8), the details of which are currently under investigation. Roughly, the intent is to make recourse to ideas developed in [4] to develop a "potential function" that will serve as yet another Lyapunov like function for (1) under the influence of (8). For example, consider

$$\phi(x) = \frac{\gamma(x)}{\left(\gamma(x)^k + \beta(x)\right)^{\frac{1}{k}}} \tag{18}$$

where

$$\beta(x) := \prod_{i \in \{1 \ldots N\}} \beta_i(x). \tag{19}$$

This potential function is uniformly unity on the boundary of $\mathcal{P}$, where at least one $\beta_i = 0$, and has a unique minimum at the destination, x^*, where $\gamma(x) = 0$. Augmenting this with a measure of kinetic energy we have

$$\eta_\phi = \phi(x) + \frac{1}{2}\dot{x}^T\dot{x}, \tag{20}$$

and taking the derivative,

$$\dot{\eta_\phi} = D_\gamma\phi \, D\gamma \, \dot{x} + D_\beta\phi \, D\beta \, \dot{x} + \dot{x}^T u. \tag{21}$$

Finally substituting (8) for u results in

$$\begin{aligned} \dot{\eta_\phi} \quad = \quad & \left(D_\gamma\phi \; - K\right) D\gamma \, \dot{x} \\ & + D_\beta\phi \, D\beta \, \dot{x} \\ & - \dot{x}^T K \dot{x} - \dot{x}^T D^2\gamma \, \dot{x}. \end{aligned} \tag{22}$$

Under appropriately conservative assumptions (large k in (18), large K in (8), $\beta(x) > \epsilon$) this expression will be negative over a suitably large domain of the state space, but the formal demonstration of this remains to be shown.

3.3 Joining the Savable Set to the Velocity Regulator (Φ_J)

In order to construct a single "controller" capable of regulating the largest possible domain to x^* it is necessary to devise a suitable scheme to smoothly transition from the "emergency stop" policy of (6) to the "velocity regulator" of (8), and ideally the transition policy should make efficient us of the available force capabilities. Begin by defining

$$\hat{e} = \frac{(x - x^*)}{||x - x^*||} \tag{23}$$

then consider

$$u = \begin{cases} \left(I - \hat{e}\hat{e}^T\right) \frac{\dot{x}}{\|\dot{x}\|} \mathrm{U}_{\max} & \text{if } \| \left(I - \hat{e}\hat{e}^T\right) \dot{x}\| > 0 \\ \hat{e}\mathrm{U}_{\max} & \text{otherwise} \end{cases} \tag{24}$$

which applies the maximum available force perpendicular to $\dot{x}$ in order to align it with $\hat{e}$, which is parallel to $D\gamma$, otherwise it uses all the available force for acceleration. Obviously this control policy does not violate (2), but some care is required to limit its application so as to not violate (3) or (4). First by choosing only to use this policy when

$$\|\dot{x}\| < \|D\gamma\| < \mathrm{V}_{\max} \tag{25}$$

we ensure that this controller will not actively violate (3). To guard against violating (4) we begin by defining a different and slightly abstracted "time to impact" from that used in Section 3.1. Define the time to impact with obstacle i, ρ_i, as the smallest positive root of

$$0 = \beta_i(x) + l_i^T \dot{x}\rho_i + \frac{1}{2} l_i^T \left(I - \hat{e}\hat{e}^T\right) \frac{\dot{x}}{||\dot{x}||} \mathrm{U}_{\max}\rho_i^2 \tag{26}$$

and define $\rho_i = \infty$ if (26) has no positive real roots or either $l_i^T\dot{x} \geq 0$ or $l_i^T\hat{e} < 0$. If $\rho_i = \infty \; \forall \, i \in \{1 \ldots N\}$ then (24) can be guaranteed to stop x from departing from $\mathcal{P}$. This new "time to impact" differs from that of (5) in that it considers the influence of the controller (24) on (1), rather than just the uncontrolled dynamics of (1).

4 The Complete Controller

Section 3 proposed three controllers to be used in concert to regulate a maximal set of initial conditions to an arbitrary interior point of a convex region in a free configuration space. To assembly these controllers we need to demonstrate that they prepare one another, and to prioritize their application. First note that (8) has $(x^*, 0)$ as a stable equilibrium, and that this point can be placed arbitrarily in the interior of $\mathcal{P}$. Thus the controller of Section 3.2 can serve as the final or "highest priority" controller. Thus it is valid to utilize a properly parameterized version of this controller whenever (16) and (17) are satisfied, and the intersection of these two constraints defines its domain.

The controller of Section 3.3, (24), has a domain given by (25) and the slightly more complex conditions associated with (26). Under it's action $\dot{x}$ will tend toward $D\gamma$ and thus the state, $(x, \dot{x})$ will approach $\mathcal{V}$, at which point the state will depart from the domain of the controller, however from Proposition 2 it is clear

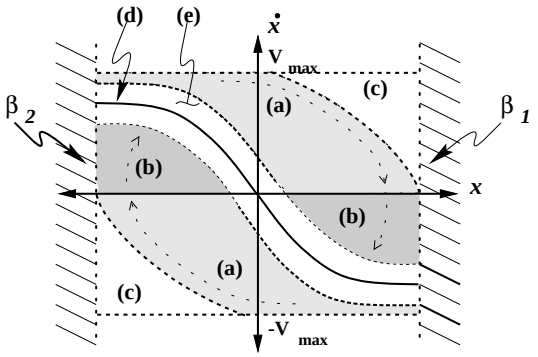

Figure 3: Diagram of the three controllers of Section 3 and their associated domains for the scalar control problem: (a) domain of Φ_S; (b) domain of Φ_J; (c) states outside of the savable set, $\mathcal{S}$; (d) the surface $\mathcal{V}$; (e) the domain of Φ_V.

that $\mathcal{V}$ lies within the domain of Φ_V, and thus these controllers have a *prepares* relationship.

Finally the "stopping" controller of Section 3.1 is capable of bringing to rest all possible states, thus having the goal set $\{(x, \dot{x}) \mid x \in \mathcal{P}\ \dot{x} = 0\}$. This set is clearly within the domain of Φ_J and there is again a prepares relationship between these two controllers.

So the final result is a hybrid control policy, which at every instant in time chooses which constituent controller to apply by sequentially testing whether the current state is in the domain of *i*) the "velocity regulator", Φ_V, *ii*) the "joining controller", Φ_J, or *iii*) the "stopping controller", Φ_S. If the state is outside all three domains then it has either already violated one of the constraints (2)-(4) or will inevitably violate (4) independent of the control action taken.

Figure 3 offers a somewhat stylized view of the relationships between these controllers for a sample scalar problem ($x \in I\!R^1$). Here the boundaries between the domains of the various controllers is clear, and it is relatively easy to envision how a trajectory evolves through the state space as the controllers are sequenced.

5 Conclusions

While the primary focus here has been on the technical issues surrounding the development of a particular control policy capable of operating over a convex region of a systems configuration space, it is important to not lose sight of the longer term utility of this policy. As mentioned in Section 2 it is only when this hybrid policy is itself used as a constituent controller for a yet higher level controller switching policy that we begin to realize the proposed method for motion programming. The flexibility of this approach becomes clear when you consider the behavior taken by such a collection of controllers running under a switching policy such as that in [1], where controllers which are "closer" to the overall goal are opportunistically activated whenever the state of the system enters their domain. Thus even in the trivial example of Figure 1, where the "control points", $\mathcal{G}_i$, are specified in the center of the corners, the controller will naturally "cut" the corners without jeopardizing safety. This results in a much more natural and efficient path with no programmer intervention, and provides a more natural and expressive means for specifying robot motion.

Acknowledgments

This work was supported in part by the NSF under grant CDA-9503992. The author would like to thank Dan Koditschek, and Howie Choset for their inspiration and many helpful discussions.

References

[1] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. submitted to the International Journal of Robotics Reaserch, 1996.

[2] Robert R. Burridge, Alfred A. Rizzi, and Daniel E. Koditschek. Toward a dynamical pick and place. In *IROS*, pages 2:292–297, August 1995.

[3] J.P. LaSalle. "Some extensions of Liapunov's second method". *IRE Transactions on Circuit Theory*, pages 520–527, December 1960.

[4] Elon Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, Oct 1992.

[5] A. A. Rizzi, J. Gowdy, and R. L. Hollis. Agile assembly architecture: An agent-based approach to modular precision assembly systems. In *IEEE Int'l. Conf. on Robotics and Automation*, Albuquerque, April 1997.