J. Baillieul J.C. Willems
Editors

Mathematical Control Theory

With a Foreword by Sanjoy K. Mitter

With 16 Illustrations

Springer

6

Languages, Behaviors, Hybrid Architectures, and Motion Control

Vikram Manikonda
P.S. Krishnaprasad
James Hendler

ABSTRACT In this paper we put forward a framework that integrates features of reactive planning models with modern control-theory-based approaches to motion control of robots. We introduce a motion description language, MDLe, inspired by Roger Brockett's MDL, that provides a formal basis for robot programming using behaviors, and at the same time permits incorporation of kinematic and dynamic models of robots given in the form of differential equations. In particular, behaviors for robots are formalized in terms of kinetic state machines, a motion description language, and the interaction of the kinetic state machine with real-time information from (limited range) sensors. This formalization allows us to create a mathematical basis for the study of such systems, including techniques for integrating sets of behaviors. In addition we suggest cost functions for comparing both atomic and compound behaviors in various environments. We demonstrate the use of MDLe in the area of motion planning for nonholonomic robots. Such models impose limitations on stabilizability via smooth feedback; piecing together open-loop and closed-loop trajectories becomes essential in these circumstances, and MDLe enables one to describe such piecing together in a systematic manner. A reactive planner using the formalism of this discussion is described. We demonstrate obstacle avoidance with limited range sensors as a test of this planner.

6.1 Introduction

In two plenary lectures in June 1983 to the Symposium on the Mathematical Theory of Networks and Systems in Beer Sheva, Israel [10], Brockett drew attention to the emerging opportunities for mathematical formulation of the fundamental problems of robotic manipulation, specifically, the problems of kinematic programming and compliance programming in grasping. He spoke on the *Product of Exponentials Formula*, a favorite device of his, and its role in understanding which kinematic programming problems are easy, and in classifying manipulator types. This formula of course is familiar to the reader, in the guise of canonical coordinates of the second kind on a Lie group and in the representation of curves on Lie groups attributed to Wei and Norman. It is a testament to Brockett's insight

that ten years later this point of view has become thoroughly integrated into the textbook literature on robotics (cf. [32]).

In the time since this influential paper, Brockett initiated and brought to success a highly ambitious program of experimental research in robotics, leading to a steady stream of innovations in robotic hands, compliant fingers, tactile sensing, new types of motors for robotics applications, integration of vision in the loop at different scales (as in the Harvard Hand Eye Machine), and a variety of fundamental investigations motivated by the settings of his experimental program. As an example, we note that Brockett's interests in nonholonomic mechanics had much to do with his work on a motor based on mechanical rectification and his work with David Montana on the kinematics of rolling contact.

One item missing from his Beer Sheva lectures was any systematic effort to take into account how we communicate with robotic machines. The use of *specific* languages tailored for interaction with machines has after all been part and parcel of robotics technology from the days of the Unimate. Brockett apparently saw that, to make progress towards truly "universal" robotics one needs to have a language for motion description that is *device independent*—somewhat akin to the page description languages that we now take utterly for granted every time we command a laser printer over a network to produce a complex document with text and graphics. Brockett came to this point of view in his 1988 paper "On the Computer Control of Movement" [12]. In this, and also in his 1990 paper [13], he articulated the concept of a motion description language MDL. It is here that Brockett comes to grips with the hybrid character of robotic movement. We need *symbolic* descriptions of movement to activate the continuous *signal* generators that constitute all robotic hardware. Further, the recognition that this view of robotic machines is indeed part of a larger world of *hybrid systems*, led him to his later formulation in [14], a model that has been highly influential in the ongoing scientific programs to understand and design Intelligent Systems.

The central ideas of control theory such as feedback and transformations into canonical forms play a role in much of the above mentioned work. One notes further that when computer scientists speak of reactive planning as a stage in robot (motion) control, they are after all speaking of feedback in some high-level symbolic sense. Increasingly one hears of behavioral styles of programming robots (see references to the work of Brooks and others below), as well as references to behavioral approaches to control system modeling and design (see [37] and the contribution of Willems in this volume). These seeming commonalities do not manage to hide the substantial gulf that exists between the perspectives of computer science towards robotics and those arising from control science. The present paper grew out of an effort at bridging this gulf and we argue that Brockett's concept of a motion description language MDL (and as exemplified in our extension MDLe) is indeed a suitable bridge to this end.

At its highest level of abstraction motion control can be viewed as the generation of symbolic inputs to a control system based on sensory information about its current state, desired state and the state of the environment it is operating in. In the case of mobile robots, these symbolic inputs are often commands such as "move,"

"turn," "stop," etc., which, along with sensor information, can then be used to generate more complex behaviors such as "avoid obstacle," "trace wall," etc.

While in many cases these symbolic inputs can be mapped into appropriate control laws that can be accepted by the system, often with more complex systems this requires a deep and a complete understanding of the underlying dynamics. In fact, in many cases the nonlinear dynamics, kinematic (nonholonomic) constraints and limited control authority make the generation of explicit control laws for precise motion control (trajectory tracking, point-to-point locomotion) exceedingly difficult. This leaves us with imprecise behaviors which need to be altered to meet the desired requirements.

We argue that motion control under such situations is reduced to generating strings of accepted symbols which can be pieced together prior to initiation of motion and then altered "on-the-fly" based on real-time input from sensors. An important factor for a motion control strategy of this nature is a hybrid architecture that serves as an abstraction between continuous and discrete (symbolic) control. In addition it is important that this framework integrate real-time sensor information into primitive behaviors in such a way as to incorporate intelligent switching between behaviors, to facilitate planning and learning. Inputs to such a hybrid control system are symbol strings and continuous and discrete inputs from sensors. Outputs are continuous signals to actuators. The input strings can be thought of as being a part of a structured language [13, 31], which is rich enough to encode sensor information and the model (essentially differential equations), and at the same time provide a set of rules for concatenation, switching etc.

Earlier work that discusses some of these aspects of motion control as applied to robotics can be found in [1, 4, 11]. Brooks [11] uses task-achieving behaviors as the primary level of task decomposition. He introduces the concept of a subsumption architecture which is essentially a structured and layered set of behaviors with increasing levels of competence. These "reactive" systems typically exploit domain constraints, using clever algorithms to allow fast processing of complex sensor information (cf. [20]). Arbib and Arkin (cf. [1, 4] and references therein) have applied *schema theory* to the robotics domain. However as discussed in [1] there is no consensus view as to what constitutes schema theory.

Although these approaches have significant advantages from the point of view of architectural design and programming flexibility, they have resisted mathematical formalization[1] and are not amenable to tests for optimality. Comparing two sets of behaviors, even within the same task, is complex and the domain-dependent nature of the solutions can cause these systems to be basically incommensurate—one may fail some times, one may fail at other times and comparison is difficult.

On the other hand, control-theoretic approaches to motion control have traditionally required detailed mathematical models of the system, its environment, and state, to design control laws/algorithms to steer the system. In addition, mobile robots are often approximated as points or disks and dynamic models assume

[1] However, see [23] for Robot Schema Language (RS).

perfect sensors and state information, making implementation of these algorithms in the real world difficult. In practice, however, autonomous systems have little *a priori* information about their environment, have limited range sensors and, in addition, dynamics can get complicated (see the discussion on nonholonomic robots in Section 6.4 making the design of explicit control laws to steer the system along a desired trajectory increasingly difficult.

The inability to effectively integrate robot model dynamics with real-time sensor information stems from the lack of a powerful enough framework to integrate the two approaches (control theoretic vs. behavior-based). This paper is a step in the direction of providing such a framework, integrating features of reactive planning with modern control-theory-based approaches to motion planning. First we introduce a motion description language, MDLe, that provides a formal basis for robot programming using behaviors, and at the same time permits incorporation of kinematic models of robots given in the form of differential equations. The structure of the language MDLe (based on Brockett's MDL [13]) allows descriptions of triggers (generated by sensors) in the language. Feedback and feedforward control laws are selected and executed by the triggering events. Secondly we present a hierarchical and distributed hybrid architecture for generation and execution of behaviors and planning algorithms developed under the formalism of MDLe.

While MDLe and the hybrid architecture provide a formalism to capture and express behavioral and control-theoretic aspects of a large class of systems, including some biological aspects, we find that MDLe is particularly well suited to the demands of nonholonomic path planning with limited range sensors. As an example of the strength of this language, we show that it can be used to support a reactive planner for nonholonomic motion planning in the presence of obstacles, using limited range sensors for obstacle detection. Some background on nonholonomic constraints and a discussion on earlier approaches to path planning with nonholonomic robots are also presented.

6.2 MDLe: A Language for Motion Control

We treat an autonomous robot as a kinetic state machine (following Brockett [13]) which can be thought of as a continuous analog of a finite automaton. In the framework of **MDLe** these kinetic state machines are governed by differential equations of the form

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i; \quad y = h(x) \in \mathbb{R}^p \tag{1}$$

where

$$x(\cdot) : \mathbb{R}^+ = [0, \infty) \to \mathbb{R}^N,$$

$$\begin{aligned} u_i: \quad \mathbb{R}^+ \times \mathbb{R}^p &\rightarrow \mathbb{R}, \\ (t, y(t)) &\mapsto u_i(t, y(t)). \end{aligned}$$

Further, each g_i is a vector field in $\mathbb{R}^N$.

We now define the *atoms* of the motion language, denoted by σ, as triples of the form $\sigma_i = (U_i, \xi_i^a, T_i^a)$ where

$$U_i = (u_1, \cdots u_m)'$$

where each u_j is as defined earlier

$$\begin{aligned} \xi_i^a: \quad \mathbb{R}^k &\rightarrow \{0, 1\}, \\ s(t) &\mapsto \xi^a(s(t)), \end{aligned}$$

is a boolean function, $T_i^a \in \mathbb{R}^+$ and $s(\cdot) : [0, T] \rightarrow \mathbb{R}^k$ is a k-dimensional signal that represents the output of k sensors. The value ξ_i^a can be interpreted as an interrupt or trigger to the system which is activated in a case of emergency or change in the environment, e.g., the robot gets too close to an obstacle.

Let us denote by $\widehat{T}^a$ (measured with respect to the initiation of the atom) the time at which an interrupt was received, i.e., ξ^a changes state from 1 to 0. [2]

If the kinetic state machine receives an input string

$$\sigma_1 \cdots \sigma_n = (U_1, \xi_1^a, T_1^a) \cdots (U_n, \xi_n^a, T_n^a)$$

then the state x will evolve according to

$$\begin{aligned} \dot{x} &= f(x) + G(x)U_1, \quad t_0 \leq t < t_0 + \min[\widehat{T}_1^a, T_1^a]. \\ &\vdots \\ \dot{x} &= f(x) + G(x)U_n, \quad t_0 + \cdots + \min[\widehat{T}_{n-1}^a, T_{n-1}^a] \\ &\qquad\qquad \leq t < t_0 + \cdots + \min[\widehat{T}_n^a, T_n^a] \end{aligned} \tag{2}$$

where $G = (g_1(x) \cdots g_m(x))$.

Hence each atom in the input string is executed in sequential order, execution of a particular atom being inhibited either via interrupts or a "time-out" via the timer T^a.

---

[2]The definition of an atom here can be compared with that in **MDL** where Brockett treats time-outs in T, instead of giving explicit status to triggers.

We denote a kinetic state machine as a six-tuple $(\mathcal{U}, \mathcal{X}, \mathcal{Y}, \mathcal{S}, h, \xi)$, where

$\mathcal{U} = C^\infty(\mathbb{R}^+ \times \mathbb{R}^p; \mathbb{R}^m)$ is a space of control laws,

$\mathcal{X} = \mathbb{R}^N$ is the state space,

$\mathcal{Y} = \mathbb{R}^p$ is an output space,

$\mathcal{S} \subset \mathbb{R}^k$ is the sensor signal space,

$h : \mathcal{X} \to \mathcal{Y}$ maps the state space to the output space

and

$\xi : \mathcal{S} \to \{0, 1\}$ is an interrupt.

As another point of departure from **MDL**, we find it useful to bring input scaling into the picture. This provides considerable flexibility as will be seen in later sections.

Definition 6.2.1 Given an atom, (U, ξ^a, T^a), define

$$(\alpha U, \xi^a, \beta T^a), \quad \alpha = (\alpha^1, \ldots, \alpha^m) \in \mathbb{R}^m, \quad \beta \in \mathbb{R}^+$$

as the corresponding *scaled atom* and denote it as $(\alpha, \beta)(U, \xi^a, T^a)$.

Hence α scaling is used to scale each input and β scaling is used to scale the time for which an atom is to be executed.

Definition 6.2.2 An *alphabet* Σ is a finite set of independent atoms, i.e, (U, ξ^a, T^a) triples. Thus $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ for some finite n where σ_i denotes the triple (U_i, ξ_i^a, T_i^a), such that $\sigma_i \neq (\alpha, \beta)(\sigma_j)$, for some $\alpha \in \mathbb{R}^m$, $\beta \in \mathbb{R}^+$ and $i = 1 \cdots n$, $j = 1 \cdots n$. Hence an alphabet is a set of atoms none of which can be derived from other atoms in the alphabet via scaling.

To simplify notation in the rest of the discussion we denote the scaled atom $(1, 1)\sigma_i$ simply by σ_i.

Definition 6.2.3 An *extended alphabet* Σ_e is the infinite set of scaled atoms, i.e., triples $(\alpha_i U_i, \xi_i^a, \beta_i T_i^a)$ derived from the alphabet Σ.

Definition 6.2.4 A *language* Σ^* (respectively, Σ_e^*) is defined as the set of all strings over the fixed alphabet Σ (respectively, extended alphabet Σ_e).

Definition 6.2.5 A *behavior*, denoted by π, is an element (i.e., word) of the extended language Σ_e^*, with an associated timer T^b and interrupt ξ^b. For example, given an alphabet $\Sigma = \{\sigma_1, \sigma_2\}$, a behavior π_i could be $\pi_i = ((\alpha_{i_1}, \beta_{i_1})\sigma_{i_1}\ (\alpha_{i_2}, \beta_{i_2})\sigma_{i_2}\ (\alpha_{i_3}, \beta_{i_3})\sigma_{i_1}), T^b, \xi^b)$.

The notation σ_{i_j} is used to denote the jth atom in the ith behavior. Similarly $\alpha_{i_j}, \beta_{i_j}$ correspond to scaling factors of the jth atom in the ith behavior.

Often we will have to work with *atomic* behaviors (behaviors with a single atom) with $\xi^b = \xi^a$ and $T^b = T^a$. In such situation to simplify notation we will denote atomic behaviors simply by $\pi_i = ((\alpha_{i_1}, \beta_{i_1})\sigma_i)$, dropping explicit reference to ξ^b and T^b.

Interrupts associated with atoms (ξ^a) are called *level*-0 interrupts. Interrupts associated with behaviors (ξ^b) are called *level*-1 interrupts. If a level-0 interrupt is received while an atom of a behavior is being executed, the execution of that particular atom is inhibited and the next atom in that behavior is executed. If a level-1 interrupt is received while a behavior is being executed, the execution of the entire behavior is inhibited and the next behavior (if there exists one) is executed. The interrupts ξ may be results of a complicated processing of sensory information. In the simplest case, however, they may involve thresholding of sensory information.

Since each atom, when executed by a kinetic state machine, combines in general both open loop and output feedback controls, one could argue that our definition of behavior captures some aspects of the essence of locomotion behavior [5], as well as the sense in which the term is used in [11]. Further, the passage from atoms to behaviors to plans suggests (as we shall see in Section 6.3) a layered architecture. We also argue that MDLe captures the salient features of various architectures/approaches to model behaviors. Comparing it with the schema approach of [1]–[4], one observes that an atom incorporates both "motor" (controls) and "perceptual" (interrupt functions) schemas into one unit.

The introduction of the timer $T^a(T^b)$ server two purposes: (i) It serves as the clock for the evolution of the differential equations, i.e., if an open loop control is an input to the kinetic state machine then the timer interrupt can be used to turn off the control at the desired time T^a. (ii) It guarantees that no behavior will be executed forever. For example if the desired behavior was "move toward a wall" and either the sensors (used to detect the wall) were defective, or the wall did not exist, then the timer guarantees that the atom (behavior) is only executed for a maximum period of time $T^a(T^b)$. With the introduction of scaling factors and a hierarchy of interrupts we provide for "directed control" (the term as used in [7]), optimization and learning. These aspects will be discussed later.

Before proceeding any further on the structure of the language we discuss an example from *Rana computatrix* (also discussed in [1, 2]) to model the visuomotor coordination in frogs and toads with the purpose of pointing out a biological motivation/application of MDLe. Applications to autonomous robots are discussed in detail in later sections.

Example 6.2.1 It has been observed that frogs and toads approach small moving objects (assuming they are prey) and move away from large ones (assuming that they might be predators). It was hypothesized that the tectum (visual region in the animal's mid-brain) was responsible for recognizing small objects and the pretectum processed visual information and determined which objects were large. If one assumes a model in which the prey-seeking behavior is activated by an input from the tectum and the predator-avoiding behavior is triggered by a signal from the pretectum, then a lesion in the pretectum should leave the frog or toad unresponsive toward large objects. It was observed, however, that a frog with a lesioned pretectum approaches both large and small objects while not exhibiting an avoidance behavior.

We model this in MDLe as follows: Define two boolean functions S_{tectum}, $S_{\text{pretectum}} : \mathcal{W} \to \{0, 1\}$ that process visual information from the world (environment) $\mathcal{W}$. We assume that visual information is passed both to the tectum and the pretectum, where each evaluates S_{tectum} and $S_{\text{pretectum}}$, respectively. These boolean functions are defined a follows:

$$S_{\text{tectum}} = \begin{cases} 1 & \text{if object is small,} \\ 0 & \text{if object is large,} \end{cases}$$

$$S_{\text{pretectum}} = \begin{cases} 0 & \text{if object is small,} \\ 1 & \text{if object is large.} \end{cases}$$

Further, let's assume that we have modeled the motion of the frog, and U_{approach} and U_{retreat} are controls that result in a motion toward and away from the prey and predator, respectively. Let us define two atoms as follows:

"APPROACH ATOM":

$$\sigma_1 = (U_{\text{approach}}, \xi_1^a, T_1^a) \quad \text{where } \xi_1^a = (S_{\text{tectum}} \vee \bar{S}_{\text{pretectum}}) \quad \text{and}$$

"RETREAT ATOM":

$$\sigma_2 = (U_{\text{retreat}}, \xi_2^a, T_2^a) \quad \text{where } \xi_2^a = S_{\text{pretectum}}.$$

Here "$\vee$" denotes the logical OR and "$\bar{}$" denotes the logical NOT. Define a behavior π as follows:

$$\pi = ((\sigma_1\ \sigma_2)^*, \xi^b, T^b)$$

where $(\sigma_1\sigma_2)^*$ defines the infinite string $\sigma_1\sigma_2\sigma_1\sigma_2 \cdots \sigma_1\sigma_2 \cdots$. The behavior interrupt and timer are chosen to interrupt this behavior after some prescribed time T^b or via ξ^b in case of undesirable changes in the environment. To understand the working of the above behavior assume that the moving object was small. Hence σ_1 will be executed (as $\xi_1 = 1 : S_{\text{tectum}} = 1$ and $\bar{S}_{\text{pretectum}} = 1$), for a period T_1^a after which execution of σ_2 will begin. But since ξ_2^a is 0 ($S_{\text{pretectum}}$ returns a 0 for small objects), σ_2 is not executed and σ_1 is executed again. This process will continue for a maximum period of T^b unless interrupted by ξ^b. Now assume that the moving object was large. In that case execution of σ_1 will fail ($S_{\text{tectum}} = 0$, $\bar{S}_{\text{pretectum}} = 0$ and hence $\xi_1 = 0$) and only σ_2 will be repeatedly executed.

We observe that the behavior π models the response of a frog to moving objects. It also fits with the observation that a frog with a lesioned pretectum will move toward both small and large objects: Without loss of generality assume that a lesion in the pretectum results in $S_{\text{pretectum}} = 0$, $\forall t$. Hence we observe that σ_2 is always inhibited but σ_1 will always be active for all moving objects. One observes that implicit in this model is that the "approach" atom processes information from both the tectum and the pretectum. (Compare with the schema model suggested by Arbib).

On a separate note one should observe that the sequential execution of atoms in a behavior does not imply that behaviors cannot be executed in parallel. It just

ensures that the same kinetic state machine (essentially the differential equations) does not receive two sets of conflicting inputs at the same time. A large complex system (which would obviously be difficult to model) could be modeled as several (physically interacting) kinetic state machines each one of them evolving in parallel and interacting through the behavior interrupts ξ^b.

Definition 6.2.6 The *length* of a behavior denoted by $|\pi|$ is the number of atoms (or scaled atoms) in the behavior.

Definition 6.2.7 The *duration* $T(\pi)$ of a behavior

$$\pi_i = (\alpha_{i_1}, \beta_{i_1})(U_{i_1}, \xi_{i_1}, T_{i_1}) \cdots (\alpha_{i_l}, \beta_{i_l})(U_{i_l}, \xi_{i_l}, T_{i_l})$$

executed beginning at time t_0 is the minimum of the sum of the time intervals for which each of the atoms in the behavior was executed and the prescribed time for which the behavior was expected to be executed. That is,

$$T(\pi_i) = \min[\ (\min[\widehat{T}_1^a, \beta_{i_1} T_{i_1}^a] + \cdots + \min[\widehat{T}_n^a, \beta_{i_l} T_{i_n}^a])\ , T_i^b] \tag{3}$$

Definition 6.2.8 Given a kinetic state machine and a world-model, a *plan* Γ is defined as an ordered sequence of behaviors which, when executed, achieves the given goal. For example a plan $\Gamma = \{\pi_3 \pi_1 \pi_n \cdots\}$ could be generated from a given language where each behavior is executed in the order in which they appear in the plan. The length of a plan Γ is given by $|\Gamma| = \sum_i |\pi_i|$ and the duration of the plan is given by $T(\Gamma) = \sum_i T(\pi_i)$. In a particular context there may be more than one plan that achieves a given goal.

Example 6.2.2 Consider the problem of path planning for a robot unicycle, with a single sensor, that wanders around in a given environment without colliding into obstacles (analogous to the idea of the first level of competence in Brooks [11]). Let us assume the task of the robot (unicycle) in this case is to wander until it senses an obstacle. If it senses an obstacle it avoids the obstacle and continues to wander around. We now formulate this problem treating the unicycle with its sensor as a kinetic state machine, and find a plan that solves the problem. The differential equations governing the kinetic state machine are

$$\begin{aligned} \dot{x} &= v_1 \cos\theta, \\ \dot{y} &= v_1 \sin\theta, \\ \dot{\theta} &= v_2, \end{aligned} \tag{4}$$

where $(x, y) \in \mathbb{R}^2$ denotes the position of the unicycle w.r.t some inertial frame, θ denotes the orientation of the unicycle relative to the horizontal axis and v_1 and v_2, the velocity of the unicycle and the angular velocity, respectively, are the inputs to the kinetic state machine. With reference to the standard notation (1), we identify $u_1 = v_1, u_2 = v_2, g_1 = (\cos\theta, \sin\theta, 0)'$ and $g_2 = (0, 0, 1)'$.

To generate the "wander behavior" (wander in a given environment without colliding into obstacles) let us consider the following atoms:

$\sigma_1 = (U_1, \xi_1^a, T_1^a)$ where

$$U_1 = (1, 0)',$$
$$\xi_1^a = \begin{cases} 1 & \text{if } \rho > 10, \\ 0 & \text{if } \rho \leq 10, \end{cases}$$
$$T_1^a \in (0, \infty),$$

and where ρ is the distance between the robot and the obstacle that is returned by the sensor;

$\sigma_2 = (U_2, \xi_2^a, T_2^a)$ where

$$U_2 = (0, 1)',$$
$$\xi_2^a = \begin{cases} 0 & \text{if } \rho > 10, \\ 1 & \text{if } \rho \leq 10, \end{cases}$$
$$T_2^a \in (0, \infty), \quad \text{and}$$

$\sigma_3 = (U_3, \xi_3^a, T_3^a)$ where

$$U_3 = (0, 1)',$$
$$\xi_3^a \equiv 1,$$
$$T_3^a \in (0, \infty).$$

Let $\alpha_i = (\alpha_i^1, \alpha_i^2)$ with each $\alpha_i^j \in [k_1, k_2]$, $j = 1, 2$, and $k_1, k_2, \in \mathbb{R}$ and $\beta \in [0, \infty)$. Now consider the following atomic behaviors [3]

$$\pi_1 = (\alpha_1, \beta_1)(U_1, \xi_1^a, 1),$$

$$\pi_2 = (\alpha_2, \beta_2)(U_2, \xi_2^a, 1),$$

$$\pi_3 = (\alpha_3, \beta_3)(U_3, \xi_3^a, 1).$$

Based on the equations of this robot, the behavior π_1 is interpreted as "move forward" with a velocity of α_1^1 units/sec for β_1 seconds, and behaviors π_2 (or π_3) can be interpreted as "turn" with a velocity of α_2^2 (or α_3^2) deg/sec for maximum of β_2 (or β_3) seconds unless interrupted. As explained earlier the atoms of each behavior will only execute as long as their respective ξ functions are 1 and the time of execution is less than T. Since $\xi_3 = 1$ in the entire interval, $[0, \beta_3]$, once π_3 begins executing it continues until $t = \beta_3$.

Consider the plan

$$\Gamma_1 = \{((\alpha_1, \beta_1)\pi_1(\alpha_2, \beta_2)\pi_3))^*, \xi^p, T^p\}$$

with $\alpha_1 = (5, 0)$, $\beta_1 = 100$, $\alpha_2 = (0, -1)$, $\beta_2 = 90$, $\xi^p = 1$, and T^p very large. Observe that, if this plan is executed in the environment (with walls W1 and W2)

[3] See remark on notation for atomic behavior immediatlely following the definition of behavior

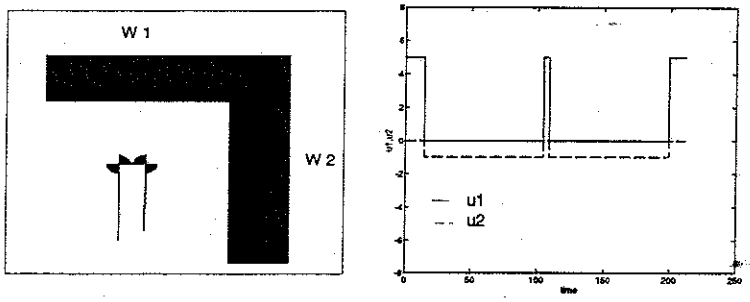

FIGURE 6.1. Trajectory and inputs generated by the plan Γ_1.

as shown in Fig. 6.1, the robot will move forward for time t, $t_0 \leq t < t_0 + 100$, where t_0 is the time at which the behavior was started, when ξ_1 will interrupt[4] it (too close to W1). Let us assume that the interrupt was received at $t_0 + \widehat{T}_1$. The execution of behavior π_1 is then inhibited, behavior π_3 is picked up from the queue and is executed. As $\xi_3 = 1$ in the entire interval $t \in [t_0 + \widehat{T}_1, t_0 + \widehat{T}_1 + 90)$ the robot will then turn clockwise by 90 degrees and then it will again execute π_1, i.e., move forward. But again after some finite time wall W2 (see Fig. 6.1) will cause $\xi_1 = 0$ and hence interrupt the move forward behavior. Behavior π_3 is executed as earlier, i.e., the robot turns clockwise by 90 degrees, and now continues to move forward. If it does not detect an obstacle at the end of 100 seconds since it began moving forward, it will stop, turn clockwise 90 degrees, and continue to repeat the sequence of actions.

Now consider the plan $\Gamma_2 = \{((\alpha_1, \beta_1)\pi_1(\alpha_2, \beta_2)\pi_2)^*, \xi^P, T^P\}$ with $\alpha_1 = (50, 0)$, $\beta_1 = 2$, $\alpha_2 = (0, -20)$, and $\beta_2 = 5$. Observe that, if this plan is executed in the same environment (see Fig. 6.2), then while executing "move forward," i.e., π_1, in the time interval $t_0 \leq t < t_0 + 2$ the robot realizes that the obstacle is at a distance less than 10 units from it and hence ξ_1 interrupts "move forward" and the robot begins to execute "turn right". Due to the choice of the interrupt function ξ_2 the robot will now switch between "turn right" and "move forward" (a condition referred to as chattering) and trace a trajectory as shown in the figure. Hence depending on the choice of the motion alphabet one can generate different plans to achieve the same task.

The question of how to generate a plan given an alphabet and a kinetic state machine, is an open one and it largely depends on the task and the planner. In Section 6.4 we describe a path planner for nonholonomic robots in which we attempt to answer related questions regarding existence and choice of alphabets. Before we discuss the features of the planner we introduce some more definitions that help formalize measures to evaluate the performance of a plan.

[4]We drop the superscript on ξ and T.

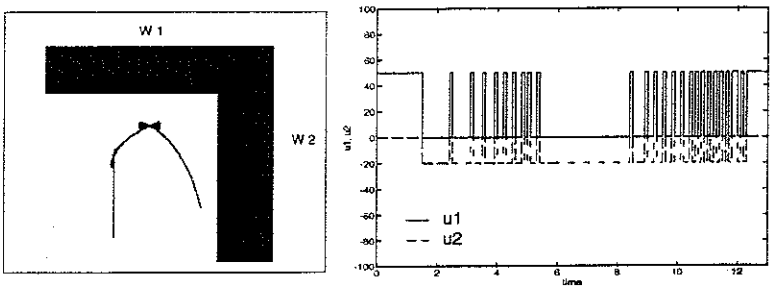

FIGURE 6.2. Trajectory and inputs generated by the plan Γ_2.

6.2.1 *Performance Measure of a Plan*

At first it appears that generating a plan to steer a system from a given initial state x_0 to a final state x_f requires complete *a priori* information of the world, which is not available in many instances of path planning. In the absence of such complete *a priori* information about the world $\mathcal{W}$, the planning system has to generate a sequence of plans based on the limited information about $\mathcal{W}$ that it has. Each such plan will only achieve an intermediate goal. Concatenation of these plans will achieve the desired goal. In MDLe each of these plans is called a *partial plan* and is denoted by $\Gamma_i^p = (\pi_1^i \cdots \pi_n^i, \xi_i^p, T_i^p)$, where ξ_i^p is an interrupt, which when set to 0, inhibits the execution of the partial plan and T_i^p is the prescribed time for which the partial plan is to be executed. Here the notation π_j^i is used to denote the jth behavior in the ith partial plan. Interrupts associated with partial plans are referred to as *level-2* interrupts.

Remark 6.2.1 As a partial plan is generated with limited information of the world, not all the behaviors and not every atom in a behavior generated by the partial plan may be executed at run time for the following reasons:

(i) Let us consider a behavior $\pi_i = (\sigma_3\sigma_1\sigma_4 \cdots \sigma_n, \xi_i^p, T_i^p)$. Let us assume that the atom σ_3 is interrupted by ξ_3^a at $\widehat{T_3^a}$. Now as explained earlier σ_1 will begin to execute. But if $\xi_3^a = \xi_1^a$, σ_1 will not be executed and again depending on ξ_4^a, σ_4 will begin to execute.

(ii) While executing an atom in a particular behavior, a level-1 or a level-2 interrupt might be received, and hence the remainder of the atoms in that particular behavior will not get executed.

Given an algorithm that generates a plan Γ we define a candidate measure of performance $\Theta(\Gamma)$ of the plan as

$$\Theta(\Gamma) = T(\Gamma) + \tau|\Gamma| \tag{5}$$

where τ is a normalizing factor having the units of time. (One need not limit oneself to such additive combinations although this is the only case used here.)

Defining a performance measure for a path planner is a difficult task as it is largely dependent on the goal the robot seeks to achieve. Some path planners use

the total time to achieve the goal as a measure of performance. In many situations one might be interested in not only the time but also on the smoothness of the path traversed or the number of times switching between different controls was necessary. For example, consider the task of parallel parking of a car. One might be able to achieve the goal by using only open-loop controls but switching between them at regular intervals, hence possibly reducing the time to achieve the goal but compromising on the smoothness of the path. On the other hand if one uses a time dependent feedback law, the same task could be achieved, possibly by moving along a smooth trajectory at the risk of taking longer time to achieve the goal. This indicates a trade-off between two competing requirements which is captured by the performance measure (5).

We now define the optimal performance of a plan as

$$\Theta(\Gamma)_{\text{optimal}} = \min\{T(\Gamma) + \tau|\Gamma|\}. \tag{6}$$

Here the minimization is performed over the subset of plans generated by the subset B of admissible behaviors. Depending on the kinetic state machine and the choice of the planner one can now place bounds on the optimal performance and hence compare the performance of different planners given the same language or that of the planner given a new language. This is illustrated in the example given below.

Example 6.2.3 Consider the problem of steering the unicycle from a given initial location z_o to z_f. The equations of the unicycle are given in Example 6.2.2. Let us assume that the language is based on the following atoms: $\sigma_1 = (U_1, \xi_1^a, 1)$, $\sigma_2 = (U_2, \xi_2^a, 1)$ where $U_1, \xi_1^a, U_2, \xi_2^a$ are as defined in example 1. Let $\alpha = (\alpha^1, \alpha^2)$ with each $\alpha^i \in [-5, +5]$ and $\beta \in (0, \infty)$.

Let us also assume that the planner did not have complete information about the world and had to generate n partial plans to achieve the goal. Each partial plan consists of steering the unicycle from z_i to z_j (see Fig. 6.3) such that there are no obstacles in some small neighborhood of the line segment joining these two locations. Let us make a further assumption that the planner uses $\alpha^i \in \{1, -1\}$ as the scaling factor while generating partial plans.

From the kinematic equations of the unicycle we know that a simple partial plan to steer a unicycle from $z_i = (x_i, y_i, \theta_i)$ to $z_j = (x_j, y_j, \theta_j)$ would be:

(i) turn by $(\theta_{z_i z_j} - \theta_i)$,

(ii) move by a distance d_i, and

(iii) finally turn by $(\theta_j - \theta_{z_i z_j})$,

where $z_i z_j$ is the vector in $\mathbb{R}^2$ joining (x_i, y_i) and (x_j, y_j), $d_i = \|z_i z_j\|_2$ and $\theta_{z_i z_j}$ is the orientation of the vector w.r.t. to the x-axis.

We can rewrite this simple algorithm as a partial plan derived from the language using the atomic behaviors $\pi_1 = \sigma_1$ and $\pi_2 = \sigma_2$,

$$\Gamma^p{}_i = \{((\theta_{i1}/|\theta_{i1}|, |\theta_{i1}|)\sigma_2\ (1, d_i)\sigma_1\ (\theta_{i2}/|\theta_{i2}|, |\theta_{i2}|)\sigma_2), \xi_i^p, T_i^p\}$$

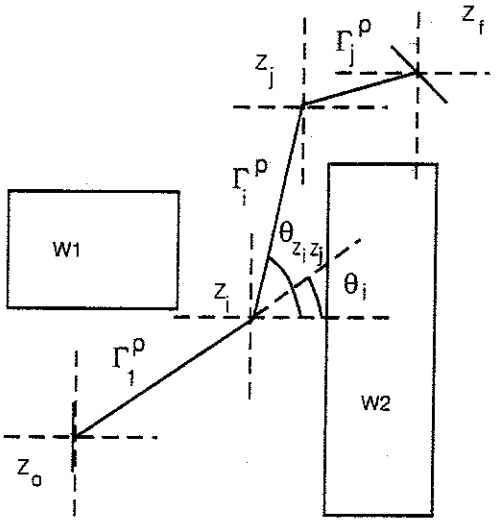

FIGURE 6.3. Partial plan generation.

where θ_{i1} and θ_{i2} are the angles of the two turns as described above of the ith partial plan. Hence the plan to steer the system from z_o to z_f is given by

$$\Gamma = \{\Gamma^\rho{}_1 \Gamma^\rho{}_2 \cdots \Gamma^\rho{}_n\}.$$

Given a plan we now illustrate how bounds can be placed on the optimal performance based on the knowledge of the kinetic state machine and the language. Let $d_{\max} = \max \| z_i z_j \|$.

$$T_{\max}(\Gamma^\rho{}_i) \leq 2\pi + d_{\max} + 2\pi \leq 4\pi + d_{\max}$$

and $|\Gamma^\rho{}_i|_{max} \leq 3$. Hence,

$$0 \leq \Theta(\Gamma) \leq 3n + n(4\pi + d_{\max}).$$

However, as we are using only open-loop controls, we know from the kinematics of the system that given an initial state x_o and a final state x_f both the behaviors $(\alpha_i, \beta_i)\sigma_i$ and $(k\alpha_i, \beta_i/k)\sigma_i$ would steer the kinetic state machine from the initial state to the final state. Hence we could replace $(\alpha_i, \beta_i)\sigma_i$ by $(k\alpha_i, \beta_i/k)\sigma_i$.

Observe that in the generation of the above partial plan and in the calculation of the performance measure we restricted α^i to $\{-1, 1\}$ (in some sense placed bounds on the speed of the unicycle) because the planner did not have complete information about the world. But since the language permits $\alpha^i \in [-5, 5]$, we have

$$0 \leq \Theta(\Gamma)_{\text{optimal}} \leq \frac{n(4\pi + d_{\max})}{5} + 3n.$$

Having placed bounds on a plan generated by one set of behaviors we can now compare the performance of another set of behaviors (for example, one using periodic functions to steer the robot) against these bounds.

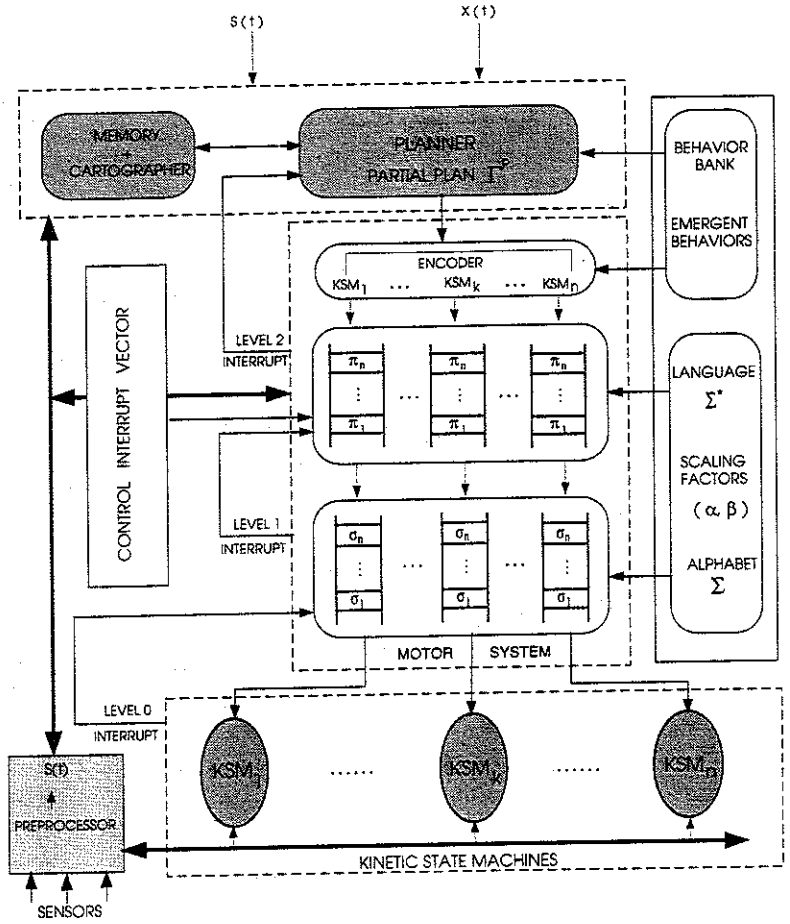

FIGURE 6.4. Hybrid control architecture.

In the above examples we have used very simple controls in our alphabet. But one should note that depending on the application, a wide variety of controls (open loop and closed loop) could be included in the alphabet and some examples of such controls can be found in [15, 16, 25, 33, 34, 36].

6.3 Hybrid Architecture

As we seek to attain higher levels of autonomy in robots, the need for hierarchical and distributed control schemes becomes apparent. Motivated in part by the hierarchical structure of neuromuscular control systems [5] we present a control architecture (see Fig. 6.4), to generate and execute plans to achieve a given task. The lowest level is the **kinetic state machine** where the sensors are used in a low-level feedback loop. Kinetic state machines serve as abstractions between

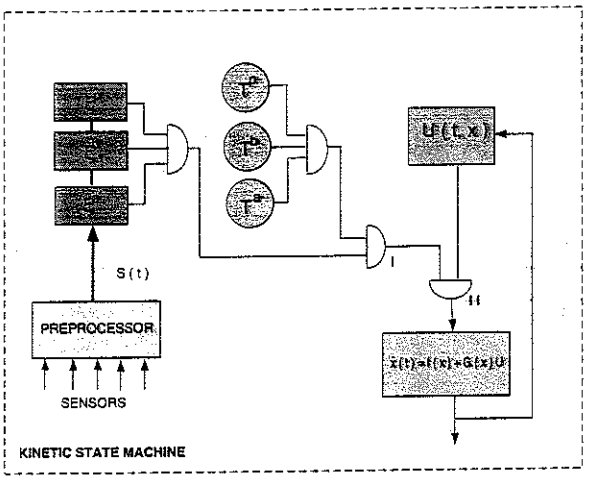

FIGURE 6.5. Kinetic state machine.

discrete (atoms) inputs and continuous time control. The working of the kinetic state machine (see Fig. 6.5) is as follows: Let us assume that an atom (U_i, ξ_i^a, T_i^a) is executed at time $t = t_0$. T is a timer whose output is 1 (active high) while $t_0 \leq t < T_i$ and is 0 (active low) if $t \geq t_0 + T_i$. $\xi_i(s(t))$ returns an interrupt (active low) to the system when conditions defined by either $\xi_i^a(s(t))$, $\xi_i^b(s(t)$ or $\xi_i^p(s(t))$ are satisfied. Observe here that the interrupt could be of level 2,1, or 0.The functioning of the AND gates in the kinetic state machine can be interpreted as follows: If either the KSM receives an interrupt or $t \geq t_0 + T_i$ (either T_i^a, T_i^b, T_i^p) the input to gate II is an active low and hence the input to the kinetic state machine is inhibited, i.e., the current atom/behavior/partial plan (depending on the interrupt level) is stopped and the next atom/behavior/partial plan in the respective queue is executed.

The **planner** is the highest end of the architecture where sensory information is processed to generate goal-related trajectory information. It uses information from its memory and the "behavior bank" to generate a partial plan to achieve the desired goal based on its current information about the world. This partial plan is in the form of actions or symbols. The **motor system** serves as an abstraction between these symbols and behaviors encoded in MDLe. Once the behaviors have been encoded, atoms in a behavior are executed as explained above. In case of sequential dependence of a set of kinetic state machines on one another, the motor system introduces "dummy atoms" (atoms with zero control input) into the respective behaviors. The interrupt functions of these behaviors are activated by one another.

External control or "directed control" [7] is permitted at two levels. The user can inhibit the input from the planner and introduce a valid behavior from the existing behavior bank, or inhibit the execution of a current behavior via the "**control interrupt vector**"(CIV). The control interrupt vector is an interrupt function ξ_{civ}^b appended to every behavior, i.e., $\xi_i^b = (\xi_i^b)_{\text{desired}} \wedge \xi_{civ}^b$. In its default state the

CIV for the particular behavior is at a logical one, but can be set to zero by the user when it is desired to inhibit the execution of a particular behavior.

As explained earlier, often the planner generates plans based on local sensor information. This information may be incorrect and might result in some of the atoms in a behavior not being executed or executed for a time less that the estimated one. If a behavior successfully completes its tasks receiving only zero level interrupts, then this behavior in its "cleaned up" version (removing unnecessary atoms, and scaling T_i^a appropriately) is loaded into the behavior bank. Alternatively a partial plan could successfully execute with interrupts of level zero or level 1. Then such a successful partial plan (in its cleaned up state), which is in essence a concatenation of behaviors, is introduced into the behavior bank. Hence we see that existing atoms and behaviors can give rise to new emergent behaviors.

The layered and distributed nature of the control becomes apparent when one observes that once a plan has been generated, each level and even various modules at the same level continue to execute independently.

6.4 Application of MDLe to Path Planning with Nonholonomic Robots

The problems of obstacle avoidance and path planning with autonomous mobile robots have been studied in various settings [21, 22, 26, 27, 35]. These approaches either assumed that the planner had to have substantive *a priori* information about the location, shapes, and sizes of obstacles, or assumed that the constraints on the robot (geometric and kinematic) were holonomic or integrable. In practice, however, most real-world robotic systems have little *a priori* information about the shapes and sizes of the obstacles and in addition include kinematic constraints that are nonholonomic. A few examples of nonholonomic systems are a front wheel drive car, dextrous manipulation or assembly with robotic hands, and attitude control of a satellite. As traditional path planners assume arbitrary motion they cannot be applied to nonholonomic robots as they result in nonfeasible trajectories, i.e., trajectories that do not satisfy the constraints on the configuration variables.

More recently, researchers have been examining nonholonomic path planning in the presence of obstacles [8, 18, 24, 29]. However, while most of these planners provide some excellent results they are quite rigid in the choice of control laws used to steer the robots and often do not exploit the control laws available in control literature, for example, [15, 16, 33, 36]. They also assume near complete *a priori* information about the world and only account for small changes in the environment.

MDLe is particularly well suited to the demands of nonholonomic motion-planning with limited range sensors. As nonholonomic robot models impose limitations on stabilizability via smooth feedback [9], the ability to piece together open-loop and closed-loop trajectories becomes essential. MDLe enables one to describe such piecing together in a systematic manner. As an example of the

strength of this language, we show that it can be used to support a reactive planner for nonholonomic motion planning in the presence of obstacles, using limited range sensors for obstacle detection. In addition, the system assumes no *a priori* information on the location and shapes of the obstacles.

In the following section we reinterpret existing results in the literature on nonholonomic robots, and answer questions related to the existence and choice of an alphabet Σ (respectively Σ_e) which can be used to generate behaviors and hence plans to achieve a required goal. We also describe (Section 6.5.3) how we can update world models and provide examples of the system's performance. For further details on nonholonomic motion planning we refer the reader to [17] and references therein.

6.4.1 *Nonholonomic Constraints*

In addition to being subject to geometric constraints many robotic systems are subject to velocity constraints. These velocity constraints are represented by relations between generalized coordinates and their velocities, and are written in matrix form as

$$A(q)\dot{q} = 0 \tag{7}$$

where $q \in \mathbb{R}^n$ determines the generalized coordinates, $\dot{q}$ are the generalized velocities, and $A(q) \in \mathbb{R}^{k\times n}$ represents a set of k velocity constraints. We also assume that $A(q)$ has full row rank. Since a kinematic constraint restricts the allowable velocities and not necessarily the configuration, it is not always possible to represent it as an algebraic constraint on the configuration space. A kinematic constraint is said to be integrable if there exists a vector-valued function $h : Q \to \mathbb{R}^k$ such that

$$A(q)\dot{q} = 0 \;\Rightarrow\; \frac{\partial h}{\partial q}\dot{q} = 0 \tag{8}$$

An integrable kinematic constraint is hence equivalent to a holonomic constraint.

Kinematic constraints that are not integrable are said to be *nonholonomic*. The constraint (7) defines a $(2n - k)$-dimensional smooth manifold $\mathcal{M} = \{(q, \dot{q}) | A(q)\dot{q} = 0\}$. These kinematic constraints generate a set of constraint forces so as to ensure that the system does not move in the direction of the rows of the constraint matrix (see Fig. 6.6). In mechanical systems such constraints are often those of rolling without slipping, conservation of angular momentum, etc.

If the controls $u(t) \in \mathbb{R}^m$ satisfy $n - k \leq m < n$ then the kinematics are sufficient to model the system and (7) can be written in the form of a drift-free control system

$$\dot{x} = \sum_{i=1}^{m} b_i(x)u_i; \tag{9}$$

with state $x(t)$ and control $u(t)$, and each b_i is a vector field. Often such drift-free nonholonomic systems are controllable (cf. [32]).

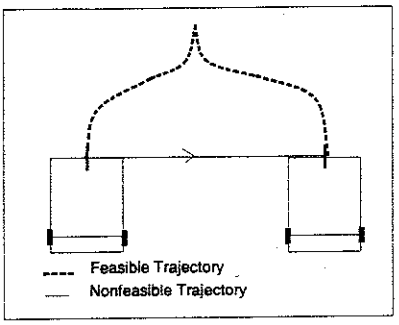

FIGURE 6.6. Nonfeasible trajectories due to nonholonomic constraints.

Proposition 6.4.1 Given an obstacle-free environment and a kinetic state machine that is governed by the differential equation

$$\dot{x} = \sum_{i=1}^{m} b_i(x)u_i; \ x \in \mathbb{R}^n, u \in \mathbb{R}^m \tag{10}$$

such that the control Lie algebra (i.e., the vector space spanned at any point by all the Lie brackets of the vector fields b_i) has rank n, then there exists an alphabet Σ (respectively Σ_e) which can be used to generate behaviors (and hence plans) to steer the system from a given initial state x_o to a final state x_f.

PROOF. From Chow's [19] theorem we know that if the control Lie algebra has rank n then the system is controllable. This implies that there exist piecewise constant controls $u : [0, T] \to \mathbb{R}^m$, $T \geq 0$, that steer the system from any initial state $x_o(0)$ to any final state $x_f(T)$.

A simple alphabet that can be used to generate behaviors consists of m triples of the form $(U_1, 1, 1), \ldots, (U_m, 1, 1)$ $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}^+$ where

$$\begin{aligned} U_1 &= (1, 0, 0, 0, \ldots, 0)' \\ U_2 &= (0, 1, 0, 0, \ldots, 0)' \\ &\vdots \\ U_m &= (0, 0, 0, \ldots, 0, 1)' \end{aligned}$$

□

While writing down the equations of motion it is sufficient to consider the evolution of the state $x \in \mathcal{C}$ (the configuration space of the robot). For path planning and obstacle avoidance one needs to be concerned with the "material points of the robot," location, and calibration of sensors. In our planner we identify the material points of the robot with a closed subset, denoted by B^r, of $\mathbb{R}^3$. Hence $B^r : \mathcal{C} \to \mathcal{F} : x \mapsto B^r(x)$, where $\mathcal{F}$ is the space of all closed connected (simply connected) subsets of $\mathbb{R}^3$. Further we define $B^r_k : \mathcal{C} \to \mathcal{F}$ such that at any given time $t = t'$ (i) $B^r(x(t')) \subset B^r_k(x(t'))$ and (ii) $d(\partial B^r_k, \partial B^r) = k$, where $d(\mathcal{X}, \mathcal{Y}) = \min_{x \in \mathcal{X},\, y \in \mathcal{Y}} \|d(x, y)\|$ and ∂B^r denotes the boundary of the set B^r. Let's assume that the robot is equipped with limited-range sensors that can detect

an obstacle in a B^r_ρ neighborhood. of x. Define an interrupt ξ_{nc} as follows.

$$\xi_{nc}(x) = \begin{cases} 0, & \forall x \in B^r_\epsilon, \quad \epsilon < \rho, \\ 1, & \forall x \in B^r_\rho \backslash B^r_\epsilon. \end{cases} \tag{11}$$

Proposition 6.4.2 Given a kinetic state machine governed by the differential equation (9) and a behavior with an associated interrupt $\widetilde{\xi}^b_i$, then the same behaviour with an interrupt of the form

$$\xi^b_i = \widetilde{\xi}^b_i \wedge \xi_{nc}$$

where ξ_{nc} is as defined by (11) will result in a collision-free path.

Since the robot is equipped with limited-range sensors in our planner the task of path planning is reduced to steering in obstacle-free neighborhoods denoted by B^r_{ofd}. Using a potential function approach (see Section 6.5.1 for details) a point $x_f \in \partial B^r_{ofd}$ is identified to which the planner steers the robot. As the size of B^r_{ofd} decreases due to nonholonomic constraints finding a control s.t. $x(t) \in B^r_{ofd}$ cannot always be guaranteed. But for a certain class of robots that are locally locally controllable (LLC)[5] we can guarantee this.

Proposition 6.4.3 If the system defend by (9) is LLC then there exists a behavior $(\cdot, \xi^b, T^b)$ such that $x(t) \in B^r_{ofd}$, $\forall t \in [0, T^b]$.

Observe that by a choice of ξ^b as defined in (11) collision avoidance is guaranteed even if material points of the robot leave B^r_{ofd}.

6.5 PNMR: Path Planner for Nonholonomic Mobile Robots

The task of the planner is to use the limited-range sensor information, to generate partial plans that result in collision-free feasible trajectories. Planning is done at two levels—global and local. For local planning, collision-free (non)feasible paths are generated using potential functions assuming that the robot is holonomic. A partial plan (feasible path) is then generated that obeys the constraints in the configuration variables. As feasible trajectories are only approximations to the trajectories generated using potential functions, collision with obstacles could occur while tracing them. While the robot is in motion, collisions are avoided by using the sensor information to trigger interrupts as described previously. The task of the path planning is outlined as follows (see Fig. 6.7):

1. Interpret local sensor information to generate a "control point" and an obstacle-free neighborhood containing this "control point" to which the robot is to be steered.

[5]The system (9) is said to be locally locally controllable at x_0 if given any $\epsilon > 0$ there exists a $\delta > 0$ such that all points in the δ-neighborhood of x_0 can be linked by a trajectory of (9) which does not leave the ϵ-neighborhood

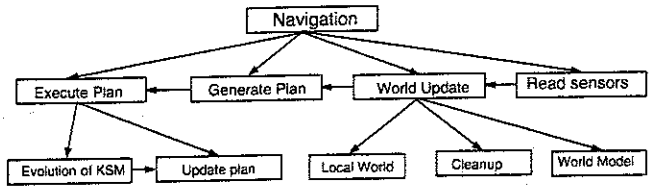

FIGURE 6.7. Navigation task decomposition.

2. From the given alphabet select atoms (U, ξ, T) that could be used to steer the robot (in general, depending on the richness of the alphabet (Σ), there could be more than one behavior to steer the robot to the control point).
3. Calculate the scaling factor α (crucial, as it determines the speed of the robot). Having calculated α, calculate or approximate β, the duration for which each atom is to be executed.
4. Generate an optimal partial plan, by minimizing the performance measure (6). The minimization is performed over the admissible behaviors.
5. Execute the partial plan and update runtime information regarding actual time of execution of behaviors in the partial plan, sensor information, etc.
6. Generate an optimal plan given partial plans and an updated world model.

At a global level, heuristics, along with the world map generated while the robot is *en route* to the goal, are used to solve the problem of cycles. One should note here that the planner could be used with most nonholonomic robots, by selecting the corresponding alphabet and associating rules with the selection of atoms. In our simulations we have assumed that the robot is modeled along the lines of a unicycle. See [30] for details on the implementation of the planner.

6.5.1 *Planning in the Obstacle-Free Disk*

To find the best direction of travel in the obstacle-free disk we use the approach of potential functions. As in the earlier work on path planning with potential functions, the idea behind our construction is based on electrostatic fields. Charges of the same sign repel and charges of the opposite sign attract in accordance with Coulomb's law. Hence we assign a positive charge distribution to the obstacles and the mobile robot and a negative charge distribution to the goal. The idea is to construct a vector field which will give the best direction of travel based on the location of the obstacles and the goal.

The robot is approximated to a point robot and as sensors can detect only points on the boundaries of the obstacles that lie in their line of sight, we treat obstacles as a collection of point charges and assign charges to them depending on which sensor detects them. The intersection of the resultant gradient field with the circumference of the obstacle-free disks gives the desired location to which robot is to be steered. (One should observe here that unlike earlier approaches the gradient field is not directly used to steer the robot. As integral curves of the resultant gradient field may not result in feasible trajectories we use the resultant gradient field only to

determine the scaling factors and x_f on the circumference of the obstacle-free disk.)

Once the initial and desired final state of the robot is known, control inputs are chosen from those available in the language to generate feasible trajectories to steer the robot from the initial location to the final desired location. If more than one such control achieves the task then the performance measure can be used to select the optimum one.

As we are using a kinematic model, of the robot an underlying assumption is that the robot is moving at low velocities and we can bring the robot to a halt simply by turning off the controls. To determine scaling factors, which are directly related to the velocities of the inputs to the equations governing the motion of the robot, we use the sum total of both the attractive and repulsive forces to determine the bounds on the velocities and hence the bounds on the scaling factors. A simple example of such a function is given by

$$g(f_a, f_r) = \begin{cases} v_{\max}, & \text{if } \dfrac{1}{k(\|f_a\| + \|f_r\|)} > v_{\max}, \\ \dfrac{1}{k(\|f_a\| + \|f_r\|)}, & \text{if } 0 \leq \dfrac{1}{k(\|f_a\| + \|f_r\|)} \leq v_{\max}, \end{cases}$$

where f_a and f_r are the net attractive and repulsive forces acting on the robot. Observe that when the robot is close to either the obstacle, the goal or both, it moves with a lower velocity, thus making the kinematic model more realistic.

By intelligently choosing weights on the charges (see [28] for more details) we can ensure that the robot either avoids the obstacle or gets close enough to an obstacle[6] such that $B^r_{ofd} \leq B^r_{crt}$ in which case it traces the boundaries of obstacles to a point where it finds an edge or is heading in the direction of the goal.

Remark 6.5.1 It is important to mention here that as we are making no assumptions on the location, sizes, or shapes of the obstacles. Guaranteeing the existence of a path is very difficult, though empirical results have shown that if a path exists the robot has more often than not found it. More importance here is stressed on the ability of the planner to integrate real-time sensor information with control-theory-based approaches to steer nonholonomic systems in a systematic way. As mentioned earlier, nonholonomic robots impose limitations on stabilizability via smooth feedback and the planner developed under the framework of the language provides an elegant way of piecing together various control strategies.

6.5.2 *Tracing Boundaries*

Planning in B^r_{crt} is a closed-loop planning strategy which essentially results in a *trace* behavior that traces the boundaries of the obstacles. Given the limited sensor and world information it is probable that the direction of trace may have been

[6] In the implementation of the planner we identify a critical radius in which the robot changes its control strategy

wrong. Hence we use a heuristic function $f(x) = D(X_{\text{robot}}, X_{\text{init}})$, the Euclidean distance between X_{robot} and X_{init} (the point at which the trace behavior was started) as an estimate of how far the robot has strayed from the goal. The robot traces the boundary as long as $f < f_s$ where f_s is some permitted distance from where the trace behavior was started. If $f > f_s$ then we retrace path and trace the boundary of the obstacle in the opposite direction. If terminal conditions for the trace are not met, we set $f_s = \alpha f_s$, $\alpha > 1$, and repeat.

Remark 6.5.2 Retracing a path under this framework is a rather simple task. Observing that the system is a drift-free system, retracing involves executing the past n partial plans in a reverse order with $(-\alpha)$ scaling factor.

Fig. 6.8 shows some paths generated by the planner for a robot modeled along the lines of a unicycle.[7] It is important to note that while the plan is being executed the sensors are being continuously scanned and are present in a low-level feedback loop, thus preventing any collisions with obstacles.

6.5.3 *World Model Update*

Once the robot has explored the environment using limited-range sensors, it is natural to expect the robot to generate plans of a better performance if it has to repeat the same task or move to goal that lie in the explore regions. We use a "learning algorithm" that improves the performance of a plan to bring it closer to optimal.

As described above, the plan to steer a robot consists of a sequence of partial plans, where each partial plan steers the robot in some obstacle-free disk of radius B^r_{ofd}. In the rest of this section let us denote each of the obstacle-free disks which were used to generate the ith partial plan as B_i and the ith partial plan as Γ^p_i. Further let us assume that n such partial plans were generated. Once the plan has been generated the planner uses following:

(i). If $B_i \subset B_j$, $i = 1, \ldots, n$, $j = 1, \ldots, n$ (i.e., B_i is contained in B_j) then obviously B_i contains redundant information. Thus if $B_i \subset B_{i+1}$ the partial plan Γ^p_i, that steers the robot from C_i to C_{i+1}, where C_i is the center of the obstacle-free disk, and partial plan Γ^p_{i+1} that steers the robot form C_{i+1} to C_{i+2} can be replaced by a partial plan $\widehat{\Gamma}^p_i$ that steers the robot from C_i to C_{i+2}. Since $B_i \subset B_{i+1}$ it is obvious that $\Theta(\widehat{\Gamma}^p_i) < \Theta(\Gamma^p_i \Gamma^p_{i+1})$

(ii). Observe that since C_{i+1} lies on the boundary of B_i, we are guaranteed the existence of a trivial nonfeasible trajectory (the straight line joining C_i with C_{i+2}) that lies entirely in $B_i \bigcup B_{i+1}$, i.e., the obstacle-free area enclosed by these

[7] It should be pointed out here that the obstacle-free disks generated by the planner violate the exact definition given above, but this is because in the simulator we have used only sensors in the "front" of the robot to generate obstacle-free disks. For now, those obstacles that are not detected by the sensors are treated as being in the blind spots of the robot.

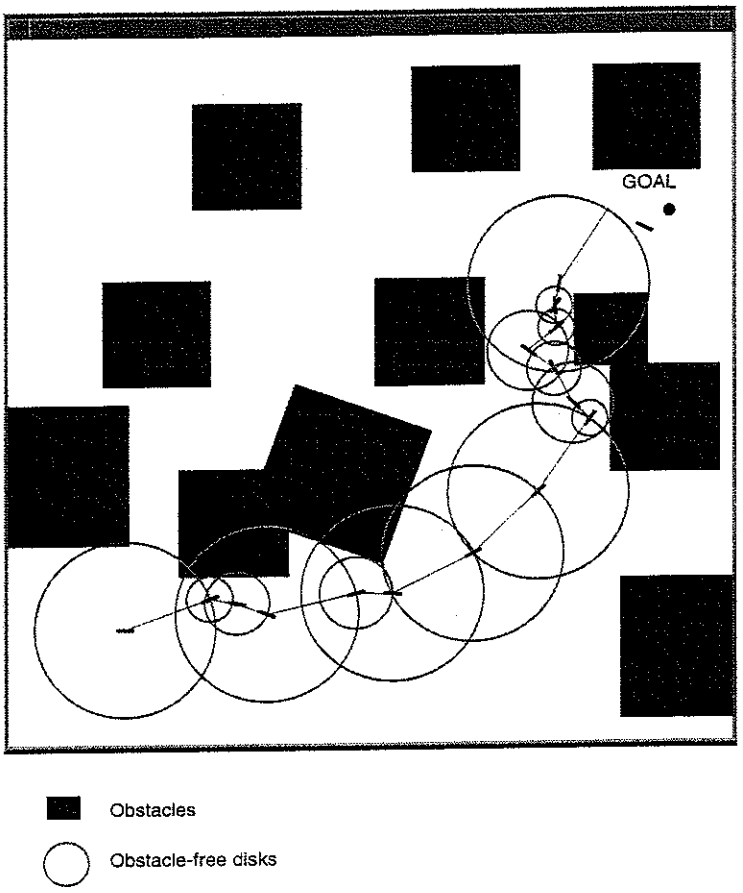

FIGURE 6.8. Paths generated by the planner.

two intersecting obstacle-free disks. Hence if there exists a partial plan $\widehat{\Gamma}_i^p$ that generates a feasible trajectory that can track this nonfeasible trajectory and lie entirely in $B_i \bigcup B_{i+1}$ such that $\Theta(\widehat{\Gamma}_i^p) < \Theta(\Gamma_i^p \Gamma_{i+1}^p)$ we can replace $\Gamma_i^p \Gamma_{i+1}^p$ by $\widehat{\Gamma}_i^p$

(iii). After the execution of (i) and (ii) we now have partial plans that steer the robot from C_i to C_{i+j}, $j \in \{2, \ldots, n\}$ such that the trajectory lies entirely in $\bigcup_i^{i+j} B_i$. The planner now explores the possibility of finding (non)feasible trajectories from C_i to C_{i+j+k}, $j \in 2, \ldots n, k = 1, \ldots, n$ such that these trajectories lie entirely in $\bigcup_i^{i+j+k} B_i$ and the performance of the plan that generates this trajectory is better than the earlier one.

Fig. 6.9 shows paths generated by the planner after it has gained partial knowledge of the world it has explored in its first attempt to reach the goal. The bold solid lines denote the new trajectories (partial plans) generated after partial knowledge

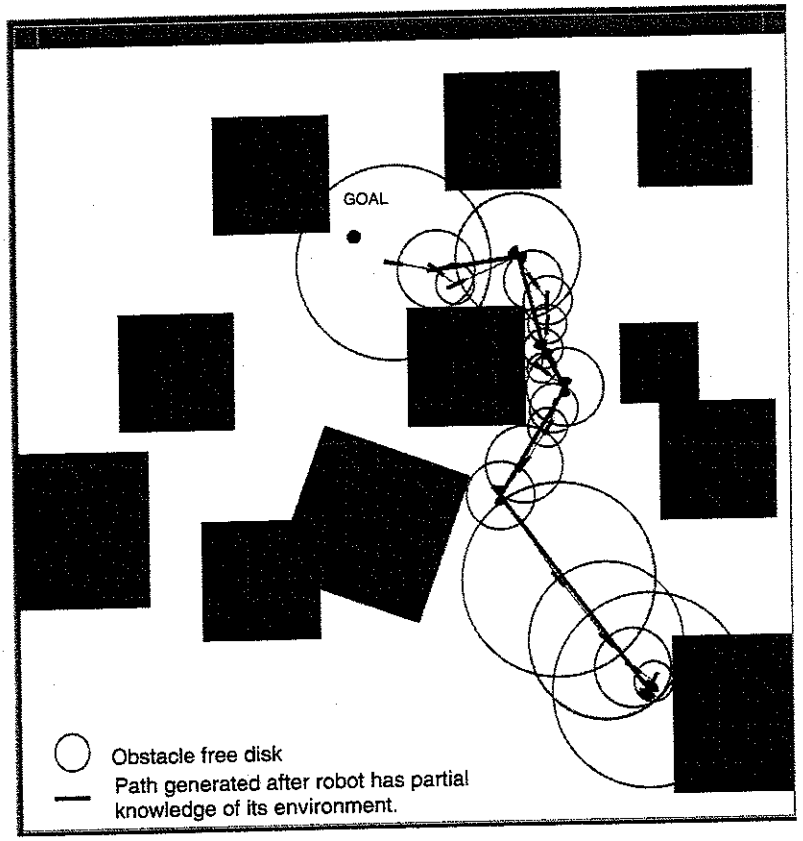

FIGURE 6.9. Paths generated by the planner.

of the world has been gained. It clearly shows an improvement in the performance of the planner as the length of the plan is nearly a third of the plan generated in the first attempt.

Remark 6.5.3 (i). Generating plans of better performance does not necessarily imply that $|\widehat{\Gamma}_i^p| < |\Gamma_i^p|$ where $\widehat{\Gamma}_i^p$ is the new plan, but rather could simply imply the choice of better scaling factors α, β such that $T(\widehat{\Gamma}_i^p) < T(\Gamma_i^p)$.

(ii). One need not restrict the generation on nonfeasible trajectories to straight line segments, but could instead use arc or even curves that best fit the centers of these obstacle-free disks.

6.6 Conclusions

This discussion is an attempt to bring together the perspectives on motion control as developed through the research programs in artificial intelligence, control theory, and the behavioral sciences. Here we provide a language, a framework,

and a hybrid architecture to integrate features of reactive planning methods with control-theoretic approaches to motion control. The hybrid language permits composition of plans from sets of behaviors. The issue of compositionality is of active interest (see, for instance, the article of Bienenstock and Geman in [6]), and the approach taken here may have some appeal in this regard. At the same time the incorporation of differential equations in the language makes it possible to formalize, compare, and generate behaviors that improve over time, construct maps, etc. It is clear that in the arena of motion-planning of systems with complex dynamics, under-actuated controls, and limited-range sensors, it is helpful to be able to switch between behaviors that rely on the direct coupling of sensory information and actuators, and steering using control-theoretic approaches. Our approach shows that these two can be smoothly integrated, at least for problems of nonholonomic robot path planning. Future work includes extending the language to continue formalization of behaviors, including multiple kinetic state machines in the language, and implementation of the planner to control a physical, as opposed to a simulated, robot.

Acknowledgments: This research was supported in part by grants from the National Science Foundation's Engineering Research Centers Program NSFD CDR 8803012, by the Army Research Office under Smart Structures URI Contract No. DAAL03-92-G0121, and under the ODDR&E MURI97 Program Grant No. DAAG55-97-1-0114 to the Center for Dynamics and Control of Smart Structures (through Harvard University), by the Office of Naval Research under ONR (N00014-J-91-1451), by the Advanced Research Projects Agency under (N00014-94-1090, DAST-95-C003, F30602-93-C-0039), by the Army Research Lab (DAAH049610297) and by the National Science Foundation under Grant EEC 94-02384, and under the Learning and Intelligent Systems Initiative Grant CMS9720334.

REFERENCES

[1] M.A. Arbib. Schema theory. In *Encyclopedia of Artificial Intelligence*, pages 1427–1443. Wiley-Interscience, New York, 1992.

[2] M.A. Arbib. Schema-theoretic models of arm, hand, and eye movements. In P. Rudomin, M.A. Arbib, F. Cervantes-Perez, and R. Romo, editors, *Neuroscience: From Neural Networks to Artificial Intelligence*, pages 43–60. Springer-Verlag, New York, 1993.

[3] C.R. Arkin. Motor schema-based mobile robot navigation. *The Int. Journal of Robotics Research*, 8(4):92–112, 1988.

[4] C.R. Arkin. Behaviour-based robot navigation for extended domains. *Adaptive Behaviour*, 1(2):201–225, 1992.

[5] N. Bernstein. *The Coordination and Regulation of Movement*. Pergamon Press, Oxford, 1967.

[6] E. Bienestock and S. Geman. Compositionality in neural systems. In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 223–226. MIT Press, Cambridge, MA, 1995.

[7] B. Blumberg and T. Galyean. Multi-level direction of autonomous creatures for real-time virtual envorinments. In *Computer Graphics Proceedings, SIGGRAPH-95*, pages 47–54, 1995.

[8] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. Technical Report STAN-CS-89-1257, Stanford University, May 1989.

[9] R. W. Brockett. Asymptotic stability and feedback stabilization. In R.W. Brockett, R.S. Millman, and H.J. Sussmann, editors, *Differential Geometric Control Theory*, pages 181–191. Birkhauser, Boston, 1983.

[10] R.W. Brockett. Robotic manipulators and the product of exponential formula. In P.A. Fuhrmann, editor, *Mathematical Theory of Networks and Systems*, pages 120–129. Springer-Verlag, New York, 1984.

[11] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[12] R.W. Brockett. On the computer control of movement. In *Proc. of the 1988 IEEE Conference on Robotics and Automation*, pages 534–540. IEEE, New York, 1988.

[13] R.W. Brockett. Formal languages for motion description and map making. In *Robotics*, pages 181–193. American Mathematical Society, Providence, RI, 1990.

[14] R. W. Brockett. Hybrid models for motion control. In H. Trentelman and J.C. Willems, editors, *Perspectives in Control*, pages 29–51. Birkhauser, Boston, 1993.

[15] J.-M. Coron. Global asymptotic stabilization for controllable systems. *Mathematics of Control, Signals and Systems*, 5(3), 1992.

[16] C. Canudas de Wit and O.J. Sordalen. Exponential stabilization of mobile robots with nonholonomic constraints. *IEEE Transactions on Automatic Control*, 37(11):1791–1797, November 1992.

[17] C. Fernandes, L. Gurvits, and Z.X. Li. Foundations of nonholonomic motion planning. In Z. X. Li and J. F. Canny, editors, *Nonholonomic Motion Planning*. Kluwer, 1993.

[18] H. Hu and M. Brady. A Bayesian approach to real-time obstacle avoidance for a mobile robot. *Autonomous Robots*, 1(1):69–92, 1995.

[19] Robert Hermann. Accessibility problems for path systems. In *Differential Geometry and the Calculus of Variations*, Chapter 18, pages 241–257. Math Sci Press, Brookline, MA, 1968. 2nd Edition.

[20] I. Horswill. Polly: A vision-based artificial agent. *Proc. of the Eleventh Conference on Artificial Intelligence*, MIT Press, Cambridge, MA, 1993.

[21] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The Int. Journal of Robotics Research*, 5(1):90–99, Spring 1986.

[22] D.E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *Proc. of the IEEE Int. Conference on Robotics and Automation*, pages 1–6. IEEE, New York, 1987.

[23] D.M. Lyons and M.A. Arbib. A formal model of computation for sensory-based robotics. *IEEE Tran. on Robotics and Automation*, 5(3):280–293, June 1989.

[24] J.P. Laumond. Nonholonomic motion planning versus controllability via the multibody car system example. Technical Report STAN-CS-90-1345, Stanford University, December 1990.

[25] N.E. Leonard and P.S. Krishnaprasad. Averaging for attitude control and motion planning. In *Proc. of the 32nd IEEE Conference on Decision and Control*, pages 3098–3104. IEEE, New York, 1993.

[26] T. Lozano-Perez. Spatial planning: A configuration space approach. AI memo 605, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1980.

[27] V.J. Lumelsky. Algorithmic and complexity issues of robot motion in an uncertain environment. *Journal of Complexity*, 3:146–182, 1987.

[28] Vikram Manikonda. A hybrid control strategy for path planning and obstacle avoidance with nonholonomic robots. Master's thesis, University of Maryland, College Park, MD, 1994.

[29] B. Mirtich and J.F. Canny. Using skeletons for nonholonomic path planniing among obstacles. In *Proc. of the Int. Conference on Robotics and Automation*, pages 2533–2540. IEEE, New York, 1992.

[30] V. Manikonda, J. Hendler, and P.S. Krishnaprasad. Formalizing behavior-based planning for nonholonomic robots. In *Proc. of the 14th Int. Joint Conference on Artificial Intelligence*, pages 142–149, Morgan Kaufmann, San Mateo, CA, 1995.

[31] V. Manikonda, P.S. Krishnaprasad, and J. Hendler. A motion description language and hybrid architecure for motion planning with nonholonomic robots. In *Proc. of the IEEE Int. Conference on Robotics and Automation*. IEEE, New York, 1995.

[32] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Boca Raton, FL, 1994.

[33] R.M. Murray and S.S. Sastry. Steering nonholonomic systems using sinusoids. In *Proc. of the 29th IEEE Conference on ecision and Control*, pages 2097–2101. IEEE, New York, 1990.

[34] J.B. Pomet. Explicit design of time-varying stabilizing control laws for a class of controllable systems without drift. *Systems and Control letters*, 18:147–158, 1992.

[35] R. Shahidi, M.A. Shayman, and P.S. Krishnaprasad. Mobile robot navigation using potential functions. In *Proc. of the IEEE Int. Conference on Robotics and Automation*, pages 2047–2053. IEEE, New York, 1991.

[36] H.J. Sussmann. Local controllability and motion planning for some classes of systems without drift. In *Proc. of the 30th Conference on Decision and Control*, pages 1110–1114. IEEE, New York, 1991.

[37] J.C. Willems. Paradigms and puzzles in the theory of dynamical systems. *IEEE Transactions on Automatic Control*, 36:259–294, 1991.