Appears in the International Joint Conference on Artificial Intelligence
Montreal, Canada, August 20-25, 1995

Formalizing Behavior-based Planning for Nonholonomic Robots *

Vikram Manikonda
vikram@isr.umd.edu
Dept. of Electrical Engineering

James Hendler
hendler@cs.umd.edu
Dept. of Computer Science

P.S. Krishnaprasad
krishna@isr.umd.edu
Dept. of Electrical Engineering

&

Institute for Systems Research
The University of Maryland
College Park, MD 20742

Abstract

In this paper we present a formalization of behavior-based planning for nonholonomic robotic systems. This work provides a framework that integrates features of reactive planning models with modern control-theory-based robotic approaches in the area of path-planning for nonholonomic robots. In particular, 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 is such as to allow descriptions of triggers (generated by sensors) in the language. Feedback and feedforward control laws are selected and executed by the triggering events. We demonstrate the use of MDLe in the area of motion planning for nonholonomic robots. Such models impose limitations on stabilizability via smooth feedback, i.e. 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 the paper is described. We demonstrate obstacle avoidance with limited range sensors as a test of this planner.

1 Introduction

The problems of obstacle avoidance and path planning with autonomous mobile robots have been studied in various settings. [Lumelsky, 1987; Lozano-Perez, 1980; Khatib, 1986; Koditschek, 1987; Shahidi *et al.*, 1991]. 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 size of the obstacles and in addition include kinematic constraints that are nonholonomic (see section 1.1 for more details on nonholonomic constraints). A few examples of nonholonomic systems are, models of a front wheel drive car, dextrous manipulation or assembly with robotic hands, attitude control of a satellite etc. 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 [Laumond, 1990; Barraquand and Latombe, 1989; Mirtich and Canny, 1992; Hu and Brady, 1995]. 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 [Murray and Sastry, 1990; Sussmann, 1991; Coron, 1992; de Wit and Sordalen, 1992]. They also assume near complete *a priori* information about the world and only account for small changes in the environment.

On the other hand behavior-based robots [Brooks, 1986; Arkin, 1992], that use real-time "sensor-based" approaches, have been able to handle more realistic models of sensing. Perhaps best known among this work is Brooks' use of 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. [Horswill, 1993]). Although this approach has significant advantages from the point of view of architectural design and programming flexibility, it has resisted mathematical formalization and is not amenable to tests for optimality. Comparing two sets of behaviors, even within the same task, is complex and the domain-dependent nature

*This research was supported in parts by grants from the National Science Foundation's Engineering Research Centers Program: NSFD CDR 8803012, the AFOSR University Research Initiative Program, under grant AFOSR-90-0105, and AFOSR-F49620-92-J-0500, from NSF(IRI-9306580), ONR (N00014-J-91-1451), AFOSR (F49620-93-1-0065), the ARPA/Rome Laboratory Planning Initiative (F30602-93-C-0039 and by ARI (MDA-903-92-R-0035, subcontract through Microelectronics and Design, Inc.)

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. In addition nature of these systems makes the improvement of behaviors over time difficult. [1]

The inability to integrate the underlying geometry with real time sensor information stems from the lack of a powerful enough framework to integrate the two approaches. 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 steer nonholonomic robots. As the first step, 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[Brockett, 1990]) allows descriptions of triggers (generated by sensors) in the language. Feedback and feedforward control laws are selected and executed by the triggering events.

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 [Brockett, 1983], 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 obstacles detection. In addition, the system assumes no *a priori* information on the location and shapes of the obstacles.

In section 1.1 we give a brief description of nonholonomic constraints that motivates the mathematical model of the motion description language. In section 2 we present details of MDLe, a language that provides a formal basis for specifying behaviors. An example of a path planner using the formalism of the language is presented in section 3. We then describe how we can update world models in section 3.4 and provide examples of the system's performance. Section 4 includes final remarks and future directions for research.

1.1 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

$$A(q)\dot{q} = 0 \tag{1}$$

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

[1]However, see [Stein, 1994] for an approach to map learning.

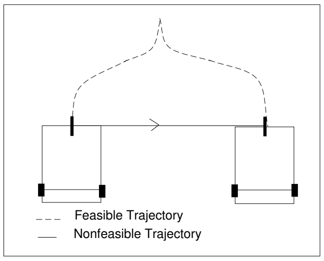

Figure 1: Nonfeasible trajectories due to nonholonomic constraints

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

$$A(q)\dot{q} = 0 \quad \Rightarrow \quad \frac{\partial h}{\partial q}\dot{q} = 0 \tag{2}$$

An integrable kinematic constraint is hence equivalent to a holonomic constraint. Kinematic constraints that are not integrable are said to be *nonholonomic*. The constraint (1) 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 1). 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 (1) can be written in the form of a drift free control system

$$\dot{x} = \sum_{i=1}^{m} b_i(x)u_i; \tag{3}$$

with state $x(t)$ and control $u(t)$, and each b_i is a vector field. Often such drift free (completely) nonholonomic systems are controllable (cf. [Murray *et al.*, 1994]). In the rest of the paper we assume that a nonholonomic robot is modeled by differential equations of the form (2), and the system is controllable.

2 Language for Motion Planning

To formalize behaviors that control a nonholonomic robot, we must provide a formal basis for the motions (in terms of control laws) and a basis for such items as receipt of sensing information and behavioral changes (interrupts) upon receiving certain such items. MDLe provides a formal language for specifying these items in a *kinetic state machine*[2] which can be thought of as the continuous analog of a finite automaton. In the framework of MDLe these kinetic state machines are governed by differential equations of the form

$$\dot{x} = \sum_{i=1}^{m} b_i(x)u_i; \quad y = h(x) \in \mathbb{R}^p \tag{4}$$

[2]The concept of kinetic state machines was first introduced by Brockett [Brockett, 1990].

where

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

$$\begin{array}{rccc} u_i : & \mathbb{R}^+ \times \mathbb{R}^p & \to & \mathbb{R} \\ & (t, y(t)) & \mapsto & u_i(t, y(t)) \end{array}$$

and each b_i is a vector field in $\mathbb{R}^n$.

We now define the *atoms* of the motion language as triples of the form (U, ξ, T) where

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

(where u_i is as defined earlier). To formalize the notion of behavior change caused by the presence of obstacles, we define

$$\begin{array}{rccc} \xi : & \mathbb{R}^k & \to & \{0,1\} \\ & s(t) & \mapsto & \xi(s(t)) \end{array}$$

a boolean function, $T \in \mathbb{R}^+$ and $s(\cdot) : [0, T] \to \mathbb{R}^k$ is a k dimensional signal that represents the output of the k sensors. ξ can be interpreted as an interrupt or trigger to the system which is activated in a case of emergency (for example, we might have such an interrupt used when the robot gets too close to an obstacle). We denote $\widehat{T}$, the time (measured with respect to the initiation of the atom) at which an interrupt was received i.e. ξ changes state from 1 to 0.

If at time t_0 the kinetic state machine receives an input atom (U, ξ, T) the state will evolve governed by the differential equation (1), as

$$\dot{x} = B(x)U, \quad \forall\, t,\ t_0 \le t \le\ t_0 + min[\widehat{T}, T].$$

If the kinetic state machine receives an input string $(U_1, \xi_1, T_1) \cdots (U_n, \xi_n, T_n)$ then the state x will evolve according to

$$\begin{array}{rcll} \dot{x} & = & B(x)U_1, & t_0 \le t \le\ t_0 + min[\widehat{T_1}, T_1]. \\ \vdots & & & \\ \dot{x} & = & B(x)U_n, & t_0 + \cdots + min[\widehat{T_{n-1}}, T_{n-1}] \\ & & & \le t \le\ t_0 + \cdots + min[\widehat{T_n}, T_n]. \end{array} \tag{5}$$

Hence we may denote a kinetic state machine as a seven-tuple $(\mathcal{U}, \mathcal{X}, \mathcal{Y}, \mathcal{S}, B, h, \xi)$, where

$\mathcal{U} = C^\infty(\mathbb{R}^+ \times \mathbb{R}^p; \mathbb{R}^m)$ is an input (control) space,
$\mathcal{X} = \mathbb{R}^n$ is the state space,
$\mathcal{Y} = \mathbb{R}^p$ is an output space,
$\mathcal{S} \subset \mathbb{R}^k$ is the sensor signal space,
B is an $\mathbb{R}^{n \times m}$ matrix (kinematic constraints matrix),
$h : \mathcal{X} \to \mathcal{Y}$ maps the state space to the output space and
$\xi : \mathcal{S} \to \{0,1\}$ maps the sensor output to the set $\{0,1\}$.

A major point of departure from Brockett's MDL, is that we find it necessary to bring input scaling into the picture as this provides considerable flexibility and also provides a basis for the comparison of behaviors (as will be discussed later).

Definition: Given an atom, (U, ξ, T), define $(\alpha U, \xi, \beta T)$, $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}^+$ as the corresponding *scaled atom* and denote it as $(\alpha, \beta)(U, \xi, T)$.

Definition: An *alphabet* Σ is a finite set of atoms, i.e (U, ξ, T) triples. Thus $\Sigma = \{(U_1, \xi_1, T_1), \cdots, (U_n, \xi_n, T_n)\}$ for some finite n or equivalently $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$ where σ_i denotes the triple (U_i, ξ_i, T_i), such that $\sigma_i \neq (\alpha, \beta)(\sigma_j)$ $\alpha \in \mathbb{R}, \beta \in \mathbb{R}^+$ and $i = 1, \cdots n$, $j = 1 \cdots n$.

To use this scaling we introduce the notions of extended alphabet and language. (To simplify notation in the rest of the discussion we denote the scaled atom $(1,1)\sigma_i$ simply by σ_i.)

Definition: An *extended alphabet* Σ_e is the infinite set of scaled atoms, i.e. triples $(\alpha U, \xi, \beta T)$ derived from the alphabet Σ.

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

This now lets us formally define a behavior as a string in this language, and also to define the complexity (length) of a behavior as well as describe its duration in a particular planning sequence:

Definition: A *behavior* π is an element (i.e. word) of the extended language Σ_e^* [3]. For example, given an alphabet $\Sigma = \{\sigma_1, \sigma_2\}$, a behavior π could be the string $(\alpha_1, \beta_1)\sigma_1(\alpha_2, \beta_2)\sigma_2(\alpha_3, \beta_3)\sigma_1$.

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

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

$$\pi = (\alpha_{i1}, \beta_{i1})(U_{i1}, \xi_{i1}, T_{i1}) \cdots (\alpha_{il}, \beta_{il})(U_{il}, \xi_{il}, T_{il})$$

executed beginning at time t_0 is the sum of the time intervals for which each of the atoms in the behavior was executed. That is,

$$T(\pi) = t_0 + min[\widehat{T_1}, \beta_{i1} T_{i1}] + \cdots + min[\widehat{T_n}, \beta_{il} T_{il}] \tag{6}$$

Using this formal definition of behavior, we can now define motion-planning problems in terms of sequences of behaviors that enable the robot to reach a goal:

Definition: 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 $|\Gamma| = \sum |\pi_i|$ and the duration of the plan is given by $T(\Gamma) = \sum T(\pi_i)$. In a particular context there may be more than one plan that achieves a given goal.

At first, it appears that, to generate a plan to steer a system from a give 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, which when concatenated will achieve the required goal. We refer to these plans that are generated on limited information to achieve some subgoal as *partial plans* Γ^p.

Given a system which can use a set of partial plans to reach a goal, one obvious way to produce a plan to steer

[3] To account for constraints one might limit behaviors to lie in a sublanguage $B \subset \Sigma_e^*$. This will be explored in future work.

the system from a given initial state x_o to a final state x_f is to use these partial plan to reach the final state, and then to store the result as a plan consisting of: $\Gamma = \widehat{\Gamma_1}^p \widehat{\Gamma_2}^p \cdots \widehat{\Gamma_n}^p$ where $\widehat{\Gamma_i}^p$ is the partial plan consisting of only those behaviors and atoms in each behavior that have been executed for $t > 0$.
The length of a plan is given by $|\Gamma| = \sum_{i=1}^{n} |\pi_i|$ and the time of execution of the plan is given by $T(\Gamma) = \sum_{i=1}^{n} T(\pi_i)$.
(Note that 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 because there may be several σ's with the same ξ's. For example, Let us consider a behavior $\pi_i = \sigma_1\sigma_2\sigma_3 \cdots \sigma_n$. Let us assume that the atom σ_1 is interrupted by ξ_1 at $\widehat{T}$. Now as explained earlier σ_2 will begin to execute. But if $\xi_2 = \xi_1$ the atom σ_2 will not be executed and (depending of course on ξ_3) σ_3 will begin to execute.

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

$$\Theta(\Gamma) = T(\Gamma) + \tau|\Gamma| \qquad (7)$$

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 rather 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 but this time taking a longer time to achieve the goal. This indicates a trade-off between two competing requirements which is captured by the performance measure (4).
We now define the optimal performance of a plan as

$$\Theta(\Gamma)_{optimal} = min\{T(\Gamma) + \tau|\Gamma|\}. \qquad (8)$$

Here the minimization is performed over the subset of plans generated by the subset B of admissible behaviors.

Due to space limitations, the above has been necessarily terse and short on examples. For more detail and for examples of kinetic state machines, behaviors and performance measures the interested reader is referred to [Manikonda *et al.*, 1995].

3 Path Planning and Obstacle Avoidance

Using the language defined above, we now present a planner which allows us to do task-planning and obstacle avoidance for a nonholonomic robot with limited range sensors. As should be clear from the above, under our framework this involves the generation of partial plans given local sensor information, such that in a global sense the robot is steered towards the goal. Although in general the partial plans generated will largely depend on the choice of the state machine (the differential equations governing the robot and the sensor information), in this section we describe a fairly general purpose planner and its implementation details in MDLe. The task of the planner is out lined as follows:

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.
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 (8). 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. Given an updated world and partial plans generate an optimal plan.

Planning is being done at two levels - global and local. For local planning, obstacle 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.

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. In the remainder of this section we describe this planner in more detail.

Critical to the planner is sensor information (location and calibration of sensors), the generation of "control points" and obstacle free neighborhoods, and selecting of atoms. Each of these is explained in some detail in the following subsections. One should note here that the planner could be used with most nonholonomic robot, 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.

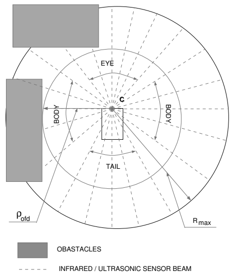

Figure 2: Location and Calibration of IR sensors

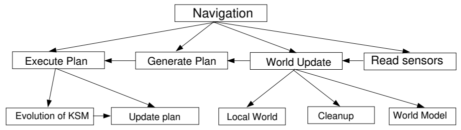

Figure 3: Navigation Task Decomposition

3.1 Location and Calibration of Sensors on the Robot

For purposes of obstacle detection, ranging and mapping, the planner assumes that the robots are provided with infrared/ultrasonic sensors located as shown in the Fig. 2. Bump sensors are located along the body of the robot. Optical encoders mounted on the motors provide position and velocity feedback. The infrared sensors are classified as ${S^i}_{eye}$ if they are located in the eye, as ${S^i}_{body}$ if they are located along the sides or as ${S^i}_{tail}$ if they are located at the rear of the robot. The sensors return distance information and we shall refer to the distance information returned as ${\rho^i}_{eye}$ or ${\rho^i}_{body}$ depending on which sensor detects an object along its line of sight. The sensors are calibrated such that the maximum range of each of these sensors is on the boundary of a circle of radius R_{max} centered at C. The sensors located at the rear of the robot are normally turned off and are turned on only when the robot is reversing. While reversing they function in a manner similar to those of the eye.

Definition : The obstacle free disk $B_{ofd}(C, \rho_{ofd})$ is defined as a disk of radius $\rho_{ofd} \quad (\rho_{ofd} \leq R_{max})$ centered at C (see Fig. 2) such that there are no obstacles in this disk, i.e. $\rho_{ofd} = \min[{\rho^i}_{eye}, {\rho^j}_{body}, {\rho^k}_{tail}],\ i = 1, ...n,\ \ j = 1...n\ \&k = 1...n$. Any trajectory that lies within this ball does not intersect the boundaries of obstacles.

As the distance between the robot and the obstacles decreases, due to the nonholonomic constraints steering might become more difficult and the planner might need to adopt a different control scheme to steer the robot. Let $R_{crt} \leq R_{max}$ denote the radius of the ball centered at C such that the planner needs to use a different set of controls to steer the robot from x_o to $x_f \in \partial\rho_{ofd}$, the boundary of the obstacle free disk. (R_{crt} can be determined empirically or analytically depending on the robot.)

Hence the problem of generating partial plans reduces to (i) planning in an obstacle free disk with $\rho_{ofd} > R_{crt}$ and (ii) planning in an obstacle free disk such that $\rho_{ofd} \leq R_{crt}$. Each of the cases is discussed later.

3.2 Navigation Task Composition

The task of navigation and obstacle avoidance can be decomposed into a number of subtasks each being executed at a certain level in the control architecture. Refer to Fig. 3 for a diagram of task decomposition.

Read sensors involves reading the distance information returned by the sensors and determining the the largest obstacle free disk centered around the robot such that the trajectories that lie in this disk are guaranteed not to intersect with the boundaries of the obstacle. In the implementation of the control strategy, the sensors are read continuously and distance information and ρ_{ofd} (the radius of the obstacle free disk) are updated into a global variable which is used to update the world and is also used by **execute plan** as an input to the interrupt function ξ. (Observe that under our approach we model the world as a network of obstacle free regions rather than mapping the obstacles.)

Update world involves periodic addition of nodes to the network. In the actual implementation of the planner we differentiate between the local world and the global world. The local world is represented as a linked list of intersecting obstacle-free disks, each of which is added to the list in the order they were visited over some finite interval of time. Each node contains information regarding the sensor distance information and ρ_{ofd}.

Partial plans use this information to generate behaviors that steer the robot to the boundary of the disk. After the complete or partial execution of the partial plan, the robot generates the next partial plan based on the sensor information available at that instant. Note that even though the sensors are continuously being monitored, a new node is added to the list only when a new partial plan needs to be generated. Hence as shown in Fig. 4 the world is updated at time instances corresponding to the points A, B, C and so on. The global world is obtained after patching together local world information and using the **clean up** operation to remove redundant information.

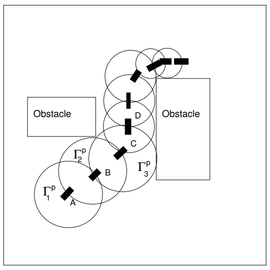

Figure 4: World Update

Generate Plan/Partial Plan - This involves the interpretation of the sensor information, and language to generate a sequence of behaviors that will steer the robot from its current location to a desired location. If a partial plan is being generated the desired location lies on the boundary of the obstacle free disk. The generation of plans/partial plans is discussed in further detail in subsections 3.3 and 3.4

In the implementation of the planner, the data structure of a plan is a linked list of atoms, each atom being represented by a structure that has information regarding the scaling factors (α, β), the interrupt function ξ, the maximum time of execution of the kinetic state machine, the inputs (controls) and a pointer to the kinetic state machine. As mentioned earlier it is possible that while executing a partial plan only some of the behaviors may be executed as planned, some may be executed only for a fraction of the the intended execution time and others may not be executed at all. **Update Plan** updates the fields of each atom of the partial plan after it has completed its execution. Once the plans/partial plans have been generated, these plans have to be executed. **Execute Plan** involves decomposition of the plan into behaviors and further into atoms. The kinetic state machine is allowed to evolve as explained in section 2, equation 2. Fig. 5 shows some paths generated by the planner for a robot modeled along the lines of a unicycle.[4] 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 hence preventing any collisions with obstacles.

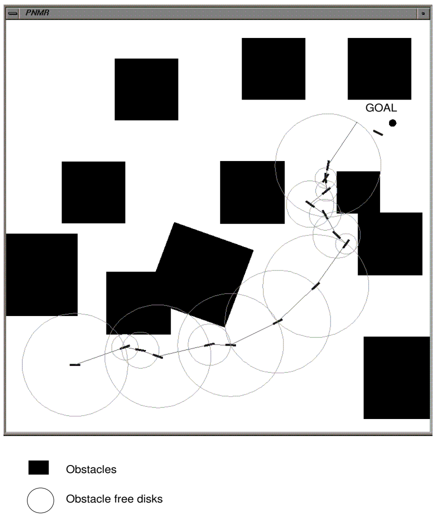

Figure 5: Paths Generated by the Planner

As the local world is a list of nodes, each of which is added to the list as the robot generates partial plans it is possible that while generating a plan the robot may have to revisit a node, and hence introducing redundant information into the world. **Cleanup** keeps track of these redundancies in the map and deletes/adds nodes in the list.

3.3 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 field theory - 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 vision, 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 if the inputs to the equations governing the motion of the robot, we use the sum total of both the attractive and repulsive forces is used 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

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

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 hence making the kinematic model more realistic.

By intelligently choosing weights on the charges (see [Manikonda, 1994] for more details) we can ensure that the robot either avoids the obstacle or gets close enough to an obstacle such that $\rho_{ofd} \leq \rho_{crt}$ 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 :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

[4]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 of the eye 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.

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 of various control strategies.

Tracing Boundaries The real behavior-based aspects of our planner can be seen when tracing the boundaries of obstacles. Planning in $B(C, R_{crt})$ 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 wrong. Hence we use a heuristic function $f(x) = D(X_{robot}, X_{init})$, 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: 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.

3.4 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 suggest 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 ρ_{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_i^p. Further let us assume that n such partial plans were generated. Once the plan has been generated the planner makes the following observations.
(i). If $B_i \subset B_j,\ i = 1, \cdots n, j = 1, \cdots n$ (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_i^p, 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_{i+1}^p that steers the robot form C_{i+1} to C_{i+2} can be replaced by a partial plan $\widehat{\Gamma}_i^p$ that steers the robot from C_i to C_{i+2}. Since $B_i \subset B_{i+1}$ it is obvious that $\Theta(\widehat{\Gamma}_i^p) < \Theta(\Gamma_i^p \Gamma_{i+1}^p)$
(ii). Observe that since C_{i+1} lies on the boundary of C_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 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 $C_{i+j},\ j \in \{2, \cdots n\}$ 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 $C_{i+j+k}, j \in 2, \cdots n, k = 1, \cdots, n$ 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 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 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.
Remarks: (i). One should note that 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 could simply imply choosing the right 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.

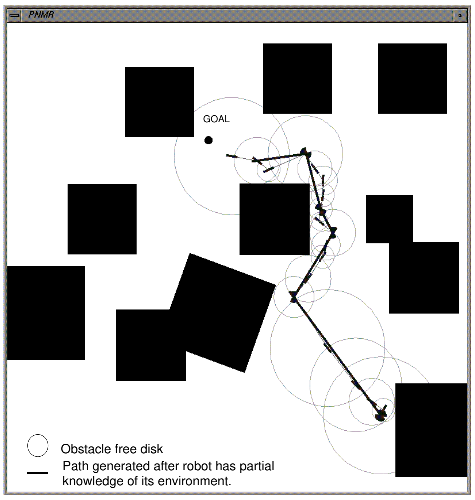

Figure 6: Paths Generated by the Planner

4 Final Remarks

This paper is an attempt to bring together aspects of nonholonomic motion planning with robots as discussed by researchers in the communities of behavior-based planning and control theory. We provide a language, a framework and a hybrid architecture to integrate features of reactive planning methods with control-theoretic approaches to steer nonholonomic robots. The hybrid language permits planning using a set of behaviors but 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, generate maps, etc. It is clear that in a task such as motion-planning using 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 modern control-theory-based approaches. Our system shows that these two can be smoothly integrated, at least for this form 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.

References

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

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

[Brockett, 1983] R. W. Brockett. Asymptotic stability and feedback stabilization. In *Differential Geometric Control Theory*, pages 181–191. Birkhauser, 1983.

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

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

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

[de Wit and Sordalen, 1992] 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.

[Horswill, 1993] I. Horswill. Polly: A vision-based artificial agent. *Proceedings of the Eleventh Conference on Artificial Intelligence*, July 1993.

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

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

[Koditschek, 1987] D.E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *IEEE International Conference on Robotics and Automation*, pages 1–6, Raleigh, NC, Mar 1987.

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

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

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

[Manikonda *et al.*, 1995] V. Manikonda, P.S. Krishnaprasad, and J. Hendler. A motion description language and hybrid architecure for motion planning with nonholonomic robots. In *IEEE International Conference on Robotics and Automation*, May 1995.

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

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

[Murray and Sastry, 1990] R. M. Murray and S. S. Sastry. Steering nonholonomic systems using sinusoids. In *Proceedings of the 29th IEEE Conference on Decision and Control*, pages 2097–2101, Honolulu,HI, December 1990.

[Murray *et al.*, 1994] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[Shahidi *et al.*, 1991] R. Shahidi, M.A. Shayman, and P.S. Krishnaprasad. Mobile robot navigation using potential functions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2047–2053, Sacremento,California, April 1991.

[Stein, 1994] L.A. Stein. Imagination and situated cognition. *J. Experimental and theoretical AI*, 6(4), Dec 1994.

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