Probability - Driven Motion Planning for Mobile Robots

Aleksandar Timcenko Peter Allen
Computer Science Dept.
Columbia University
New York, NY 10027

Abstract

This paper proposes a path-planning method for mobile robots in the presence of uncertainty. We analyze environment and control uncertainty and propose methods for incorporating each of them into the planning algorithm.

We model the environment using the pyramid structure that encodes the information on occupancy probabilities for each pixel as well as the partial information on conditional probabilities among different pixels. This structure allows for efficient and accurate computation of collision probabilities in the presence of environment uncertainty.

The control uncertainty is mainly characterized by its expansion in space and time and is accordingly modeled by a stochastic differential equation that mathematically captures this phenomenon. Models that we develop are inevitably approximate but experiments confirm that they can be used as a reasonable model for motion planning.

We have conducted a series of experiments on the mobile platform and we present some of these results in the paper.

This work was supported in part by DARPA contract DACA-76-92-C-007, NSF grants IRI-86-57151, CDA-90-24735, North American Philips Laboratories, Siemens Corporation and Rockwell International.

1 Introduction

Realistic robot systems are flawed by uncertainty. Handling the uncertainty is necessary for improving robot's performance. Some well known motion planning techniques, such as Voronoi diagrams [8] and cell decomposition [12], often referred to as "global", are based on graph searches and they assume that a robot's sensing, control and knowledge of an environment are perfect. This assumption, albeit never absolutely true, is realistic in non-cluttered environments when the required accuracy in the goal is not critical. The simple — and usually quite sufficient — approach is to slightly "grow" the obstacles and "shrink" the goal in the configuration space to compensate for all present uncertainties. Motions planned under these assumptions are usually called *gross* motions.

Nevertheless, the necessity for a more elaborate treatment of uncertainties exists. Intuitively, by conservatively "growing" the obstacles we may either run out of free space or the goal region may disappear. Thus, we need a planning methodology capable of coping with inherent uncertainties in a more elaborate way. More precisely, we need a tool that allows us to suppress the unwanted effects of different uncertainties — for example, even if our robot "slips" from the prescribed trajectory, we want to be able to guide it towards the goal anyway.

The classical work in robotic planning in the presence of uncertainties has been done by Lozano-Pérez and colleagues [6, 4, 2]. This planning strategy is referred to as a worst-case planning since the computed plans are *guaranteed* to succeed under assumed conditions. There have been several attempts to generalize this planning strategy to average case (see, for example, [1]). Some further developments are [3] and [10].

Output: A path planning algorithm for real-sized robots in the presence of environment and control uncertainty

Identify sources of uncertainty;
Propose method for path planning for a point robot in the presence of environment uncertainty;
Extend this algorithm for the case of real-sized robots;
Propose the mathematical model of control uncertainty;
Experimentally establish the model of control uncertainty;
Merge the path planning algorithm with the model of control uncertainty;

Figure 1: Overview of the planning method

The following is the overview of the work presented in this paper (see figure 1). The first step is the identification of sources of uncertainty. We will consider two types of uncertainty: environment and control uncertainty. Next step is to come up with a path planning algorithm for point robots in the presence of environment uncertainty. Point robots are robots whose size is negligible compared to the size of obstacles. This algorithm is developed in section 2. The extension of this algorithm for path planning of real-sized robots is given in section 3. We establish the mathematical model of control uncertainty for an experimental robotic system (a mobile platform) in section 4. The integration of the path planning algorithm from section 3 and the control uncertainty model from section 4 is presented as the final result of this paper in section 5, where we give the path planning algorithm for real-sized robots in the presence of environment and control uncertainty.

This paper is based on the first author's doctoral dissertation [13].

2 Path Planning for Point Robots with Environment Uncertainty

The problem we want to address in this section is how to move a "point robot" (a robot whose size is negligible compared to the size of obstacles) among obstacles in a 2D environment from an initial point to a goal point, observing certain dynamical properties of the planned trajectory such as motion duration and maximal impact force. The positions of obstacles are not completely known. What is known are the probabilities that a given point is occupied by an obstacle. We call such an environment "semi-static" since the obstacles' appearance, position and orientation are allowed to change in between consecutive task execution instances. The scenario for a semi-static environment is a conveyor belt or a machine feeder that brings a part to be processed each time in a slightly different posture. Another possible scenario is the floor of

a warehouse with the objects in it having static but not completely known positions.

The model of a semi-static environment will be the widely used concept of an occupancy grid [7], obtained by averaging a series of images of the 2D workspace from an overhead camera. The lighting and coloring of obstacles is assumed to allow for unique distinction between objects and the background. As a series of images is taken, the brightness of each pixel is computed to be proportional to the frequency of pixel's occupancy by an obstacle. As a solution of the find-path/trajectory parameterization problem we propose a combination of a grid search technique based on the A* search algorithm and a kinetic energy-type criterion optimization. We call this criterion the *task execution difficulty index*.

2.1 Path planning with probability-induced metric

The main idea that we employ for path planning is the choice of a probability-dependent metric. The mathematical entity that defines a metric in a given space is a metric tensor. We can think of a metric tensor as an $n \times n$ matrix where n is the dimensionality of the space in question, in our case $n = 2$. The infinitesimal distance $\mathrm{d}s$ is defined through a metric tensor g_{ij} by the formula $\mathrm{d}s^2 = g_{ij}\mathrm{d}\mathbf{q}^i\mathbf{q}^j$ where $\mathrm{d}\mathbf{q}$ is an infinitesimal vector.

We will utilize the freedom to choose the metric tensor in order to cope with the uncertainty. Intuitively, we can think of uncertain areas as "less desirable" and penalize trajectories that go through those areas by considering them to be longer. The metric tensor defines the pseudo distance in the configuration space. This pseudo distance should be a function of the occupancy probability Ψ_{fail} with the following properties:

$$\begin{cases} \mathrm{d}s = \sqrt{\delta_{ij}\mathrm{d}q^i\mathrm{d}q^j}, & \Psi_{\text{fail}} = 0 \\ \mathrm{d}s \text{ increases}, & \Psi_{\text{fail}} \text{ increases} \\ \mathrm{d}s = \infty, & \Psi_{\text{fail}} = 1 \end{cases} \tag{1}$$

where $\Psi_{\text{fail}} = \Psi_{\text{fail}}(\mathbf{q})$ is the probability that the point $\mathbf{q}$ is occupied by an obstacle, which would cause the robot in this position to collide with this obstacle and consequently cause the plan to fail. The infinitesimal distance $\mathrm{d}s$ is defined by the metric tensor as stated above.

It is known in theoretical mechanics that the shortest path between two points is given by a geodesic line that connects those two points. This geodesic line is a solution of a differential equation that depends on the metric tensor. The condition that distance becomes infinite if a point is certainly part of an obstacle guarantees that the paths obtained as solutions of this differential equation will avoid obstacles, while the condition that $\mathrm{d}s$ rises as b rises results in a tendency to "prefer" paths through areas that are unlikely to be populated by obstacles over areas with substantial uncertainty.

We have chosen g_{ij} to be

$$g_{ij} = \delta_{ij}\phi(\Psi_{\text{fail}}) \tag{2}$$

where δ_{ij} is Kronecker tensor (i.e. identity matrix), ϕ is an appropriately chosen scalar function and $\Psi_{\text{fail}} = \Psi_{\text{fail}}(\mathbf{q})$ is the probability that point $\mathbf{q}$ is part of an obstacle.

Once the pseudo-distance is defined, the path planning is accomplished by a grid search using the A* algorithm. This algorithm is guaranteed to find a path if one exists. The cost of getting from one point to another is assumed to be the pseudo distance between those two points. This guarantees that the path that the A* algorithm finds is the shortest path with respect to the assumed distance metric.

The heuristic function in the A* algorithm is chosen to be the Euclidean distance between the current point and the goal point.

2.2 Trajectory time parameterization

In this subsection we discuss the velocity planning along a precomputed path. This separation of path and velocity planning is the approximation that we are forced to make due to the numerical instability of simultaneous optimization of the path and the velocity.

The velocity is planned through an optimization of a criterion that will guarantee that the computed result is "the best" in a certain sense. The choice of a criterion will define the meaning of "the best". An attractive choice for a criterion is undoubtedly a mathematical embodiment of the concept of "difficulty" of a task. Now we will define an optimization criterion that we will call *task execution difficulty*. It combines motion velocity with the knowledge about the environment presented through the metric tensor. The metric tensor in itself contains the probabilistic information on obstacle arrangement in the environment, as in formula 1.

Definition 1 *A* task execution difficulty index *is*

$$\mathcal{D} = \frac{1}{2}\int_0^\tau g_{ij}\mathbf{v}^i\mathbf{v}^j\mathrm{d}t \tag{3}$$

for positioning a point robot from $\mathbf{q}(0) = \mathbf{q}_{\text{start}}$ *to* $\mathbf{q}(\tau) = \mathbf{q}_{\text{goal}}$ *in time* τ, *where* $\mathbf{v}^i$ *is a velocity vector* $\mathrm{d}\mathbf{q}^i/\mathrm{d}t = \mathbf{v}^i$ *and* g_{ij} *is a certain two time covariant tensor.*

For the purpose of planning velocity we choose the same metric tensor as for path planning. The optimal velocity profile along the path can be analytically found and the solution is (see [14] for details):

$$v = v_0/\sqrt{\phi} \tag{4}$$

where v_0 is an initial velocity.

In order to determine the initial velocity v_0 we need to impose a condition on total motion duration τ. Simple analytic manipulation yields

$$v_0 = \frac{1}{\tau}\int\sqrt{\phi}\mathrm{d}s$$

where integrals are taken from $\mathbf{q}_{\text{start}}$ to $\mathbf{q}_{\text{goal}}$.

The importance of formula 4 is that we now have a solution for velocity v along the trajectory in closed form. This solution depends only on *local* properties of the trajectory — the probability that the current point is occupied by an obstacle — and so it can be computed efficiently.

2.3 Simulation results

The metric tensor we have chosen is of the form

$$g_{ij} = \delta_{ij}\phi(\Psi_{\text{fail}}) = \delta_{ij}/\sqrt{1-(\Psi_{\text{fail}})^\rho}$$

where ρ is a parameter . In our simulations we have assumed the value $\rho = 2$.

Figures 2 through 6 demonstrate the behavior of the planner as we increase the uncertainty further. The path becomes more conservative, totally abandoning the narrowing channel between the central and bottommost obstacles. As the uncertainty increases even more, the planner chooses the third topologically different path, presented in figure 5. Finally, if the uncertainty is blown up so that almost no information remains, the planner just follows a straight line path from the start to the goal (figure 6).

3 Path Planning for Real Robots with Environment Uncertainty

The approach to the motion planning for real-sized robots that we examine in this section is to estimate the probability of an arbitrarily-shaped robot hitting an obstacle in a given configuration based on the same occupancy grid as in the previous section. This would both keep the storage requirements low and allow for the metric tensor method to be applied directly. Also, this method is general in a sense that the data-gathering phase does not need to be repeated

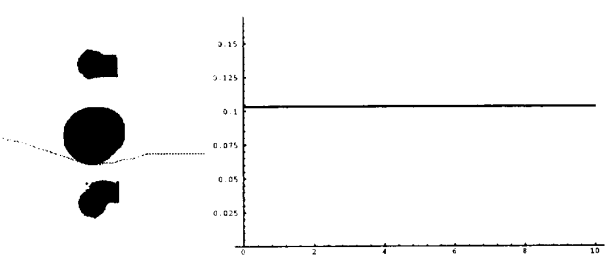

Figure 2: Path and velocity, $\sigma = 0.0$

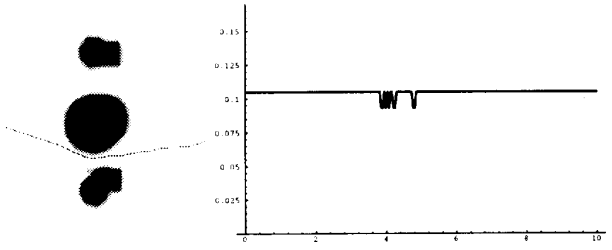

Figure 3: Path and velocity, $\sigma = 0.01$

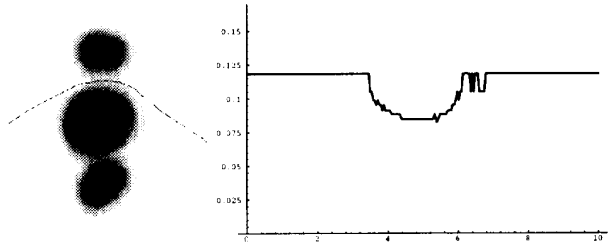

Figure 4: Path and velocity, $\sigma = 0.03$

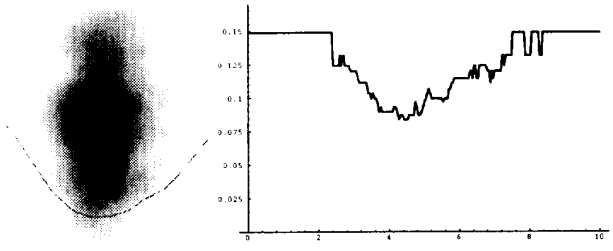

Figure 5: Path and velocity, $\sigma = 0.08$

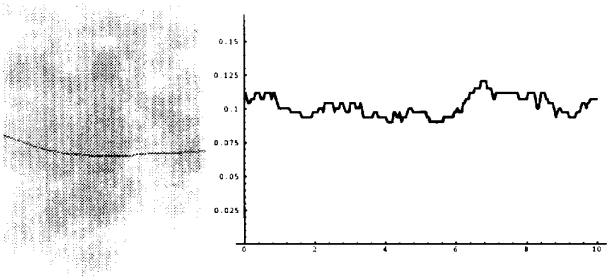

Figure 6: Path and velocity, $\sigma = 0.5$

for a robot with a different shape, but rather the same occupancy grid may be used again. The price that we have to pay is that the probability of an impact is just an estimate of a real probability based on certain heuristics. Only experiments can prove or disprove the choices we have made.

In a summary, the algorithm for robot motion planning in 2D that we describe in this section has the following favorable characteristics:

- *Robot's shape independence*: The algorithm allows for planning paths for differently shaped and possibly shape-changing robots.
- *Velocity planning*: The algorithm utilizes a similar criterion optimization mechanism as described in section 2 for the robot's velocity planning.
- *Efficiency*: The search through high-dimensional configuration space is avoided. The algorithm is implemented as a set of parallel processes that may run on a distributed system.

The algorithm has been implemented and the experiments have been conducted on the mobile platform in a simple laboratory environment (see figure 11).

This section explains how we compute the collision probability for a robot in a given configuration. It is very difficult to compute this probability exactly. In order to do this, one would need to know all conditional dependencies between pixels that robot covers in a given configuration. This would require unattainable amount of information. For this reason we are forced to adopt some approximate solution. The approximation that we propose is to define an operator that combines pixel probabilities based on some approximate dependency information obtained during the generation of the environment. This operator, denoted $\oplus$, needs to posses certain algebraic properties in order to allow pixel probabilities to be combined. It depends on the pixel dependency parameter, denoted ℓ.

Let $\mathcal{A}$ be a planar robot, and $\mathcal{A}(\mathbf{q}, \theta)$ is the set of pixels covered by the robot in configuration $(\mathbf{q}, \theta)$. For a point robot, the cardinality of $\mathcal{A}$ is $||\mathcal{A}(\mathbf{q}, \theta)|| = 1$ (robot occupies exactly one pixel). For the real robot, $||\mathcal{A}(\mathbf{q}, \theta)|| > 1$.

Let $\mathcal{CB}$ be a set of obstacles in the plane. The probability Ψ_{fail} that configuration $(\mathbf{q}, \theta)$ is unattainable is the probability that any of the pixels $\mathbf{q}'$ from $\mathcal{A}(\mathbf{q}, \theta)$ are part of the obstacle:

$$\Psi_{\text{fail}}(\mathbf{q}, \theta) = \Psi \left\{ \bigvee_{\mathbf{q}' \in \mathcal{A}(\mathbf{q}, \theta)} \mathbf{q}' \in \mathcal{CB} \right\} \tag{5}$$

In order to be able to combine occupancy grid probabilities for all pixels $\mathbf{q}' \in \mathcal{A}(\mathbf{q}, \theta)$ the binary operation $\oplus$ has to satisfy certain additional conditions. Particularly, we require that the algebraic structure $(\oplus, [0, 1])$ is a commutative group. This condition is important for Ψ_{fail} as given in 5 to be well-defined. It guarantees, through associativity and commutativity of $\oplus$, that the particular order in which we combine probabilities in 5 is unimportant.

Using the symbol $\oplus$ we can now rewrite formula 5 in the following form:

$$\Psi_{\text{fail}}(\mathbf{q}, \theta) = \bigoplus_{\mathbf{q}' \in \mathcal{A}(\mathbf{q},\theta)} \Psi(\mathbf{q}') \tag{6}$$

where $\Psi(\mathbf{q}')$ stands for the probability that $\mathbf{q}' \in \mathcal{CB}$.

The final form for $\oplus$ that we chose after carefully considering $\oplus$'s algebraic properties, is

$$\Psi_A \oplus \Psi_B = 1 - ((1 - \Psi_A)^\ell + (1 - \Psi_B)^\ell) - 1)^{1/\ell} \tag{7}$$

We use a combination of two hierarchical data structures in the algorithm that computes the collision probability: a quad tree [11] and a pyramid [5]. A quad tree is used to represent the robot and a pyramid is used to represent the environment.

A quad tree is a hierarchical data structure that allows an efficient representation of a 2D space in a form of a tree data structure. In its original definition, as given in [11], each node in a tree is a leaf or a non-leaf, each non-leaf has exactly four sons, named NW, NE, SW, SE (for north-west, etc.), and each leaf is said to be either BLACK, if corresponding set of pixels entirely belongs to an object in an environment, or WHITE, if the entire set of pixels represented by the leaf in question is in free space in the environment. Non-leaf nodes are considered to be MIXED, which simply means that they have offsprings that are both BLACK and WHITE.

The problem in representing the environment with a quad tree is that it is not sufficient to compute the collision probability for a given robot in the given configuration due to the fact that it is not possible to deduce what is the probability that a set of pixels (in this case, a square) is occupied by an obstacle based on probabilities for each pixel that constitutes the set. This information has to be acquired directly from the environment through averaging a series of snapshots, each discretized with a set of resolution levels. For this purpose we need more general data structure, a pyramid [5]. The algorithm for building a pyramid works with a series of snapshots of the environment by averaging a series of binary (BLACK / WHITE) images with different resolutions for each snapshot. The result is a tree structure with each node bearing the probability that the representing set of pixels contains an obstacle. These probabilities are used to determine the dependency parameter ℓ in each non-leaf node. A dependency parameter is not defined for leaf nodes. In order to determine ℓ we apply formula 7 to probabilities computed for a node's children. This is done as follows. Let Ψ be the probability in the node for which we are computing the dependency parameter, and $\Psi_{\text{NE}}, \Psi_{\text{NW}}, \Psi_{\text{SE}}, \Psi_{\text{SW}}$ probabilities in node's NE, NW, SE and SW son. Knowing that the operator $\oplus$ is associative, we can combine probabilities $\Psi_{\text{NE}}, \Psi_{\text{NW}}, \Psi_{\text{SE}}$ and Ψ_{SW} using formula 7. This combination should equal Ψ. This condition represents an equation in ℓ since the operator $\oplus$ is parameterized by ℓ. Solving These equations for each non-leaf node we compute the dependency parameters.

Now we present a simplified version of the algorithm that computes the collision probability for a given robot in a given configuration. Let $\mathcal{A} = \{P_1, P_2, \ldots, P_n\}$ be a set of pixels in the image covered by the robot in a given configuration, and let T be a quad tree that represents this set. Pixels $P_1, \ldots, P_n$ are either leaves of T or belong to sets of pixels (squares) represented by leaves in T. We also assume that there is available a pyramid structure that represents the environment. The algorithm for computing the probability that a node from T is in a state of collision with an obstacle from the environment traverses the tree T in depth-first manner. If the node is BLACK in T, that means that all pixels represented by this node belong to $\mathcal{A}$. Thus, the probability returned is the probability for that node found in the pyramid. If the node has the value WHITE in T this means that no pixels represented by the node are in $\mathcal{A}$, thus the probability of collision is 0. However, if the node is MIXED in T, we apply formula 7 to combine probabilities in the node's sons, obtained recursively. The dependency parameter ℓ is available from the pyramid for each non-leaf node.

The full account on this algorithm can be found in [13].

Now that we have the algorithm for computing the collision probability we use this probability to determine the pseudo-distance between pixels the same way it is done in the previous section. Given this information, the A* search algorithm can find the shortest path between start and goal configurations in the robot's configuration space. Also, the motion velocity (formula 4) depends only on the collision probability, and thus can be computed once the collision probability is found.

Development of the control uncertainty model:

Output: SDE that models the control uncertainty for a given system;

begin
Select a SDE that, based on the initial understanding of the system models its control uncertainty;
Conduct experiments that measure different aspects of control uncertainty
(e.g. position errors due to linear motions, orientation errors due to rotations, etc.)
do
Numerically fit the coefficients of the SDE to the measurement results;
until (SDE closely reflects the measured data);
{ *The next step is optional* }
Statistically examine the obtained model;
if (The discrepancies between the measurements and the model statistically significant)
Go back to the first step and choose another SDE;
Verify the model experimentally;
end.

Figure 7: Development of the control uncertainty model

4 Control Uncertainty Model

In this section we will analyze the *propagation* of uncertainty in space and time and the ways to capture it in mathematical form. Our primary goal is to show that the uncertainty can be *measured* and statistically *modeled* through stochastic differential equations that capture its dynamic nature. The ability to measure the uncertainty is important; we cannot proceed with planning motions unless we know both the nature and the quantity of the uncertainty present in the system.

The primary conclusion drawn in this section is twofold:

- The uncertainty in robotic systems dynamically changes in space and time.
- The control uncertainty of a wide class of robotic systems can be modeled through stochastic differential equations. Those models, obtained through experiments, can be statistically verified.

The development of the control uncertainty model for a given system is shown in figure 7. This algorithm comprises in several steps. The first step is the choice of the SDE that models the control uncertainty. We will concentrate on a linear SDE that results in Gaussian random variables. This is certainly an approximation but we argue that by careful experimental measurement models obtained this way may serve as a reasonable source of information on the nature if control uncertainty.

The SDE that we are looking for as a model of the control uncertainty is, in its most general form [9],

$$\mathrm{d}\Delta\mathbf{q} = \mathbf{A}(\Delta\mathbf{q}, \mathbf{v}^c, t)\mathrm{d}t + \Sigma(\Delta\mathbf{q}, \mathbf{v}^c, t)\mathrm{d}\mathbf{W} \tag{8}$$

where $\Delta\mathbf{q} = [\Delta x\ \Delta y\ \Delta\theta]^T = \mathbf{q}^m - \mathbf{q}^c$ is the difference between the actual robot's position $\mathbf{q}^m$ and the commanded position $\mathbf{q}^c$, $\mathbf{v}^c$ is the commanded

motion velocity and $\mathbf{W}$ is the random Wiener process. Vector functions $\mathbf{A}$ and Σ define the SDE that models the control uncertainty.

In order to simplify the mathematical exposition and the experimental estimation of parameters of SDEs that model the control uncertainty in our testbed systems, we will make the following assumptions that result in the Gaussian noise model:

- the commanded motion velocity $\mathbf{v}^c$ does not depend on the robot's position $\mathbf{q}^m$
- $\mathbf{A}$ is either independent from $\Delta\mathbf{q}$ or proportional to $\Delta\mathbf{q}$. It may however depend non-linearly on other motion parameters, such as the commanded motion velocity $\mathbf{v}^c$.
- Σ is independent from $\Delta\mathbf{q}$. However, similarly to $\mathbf{A}$ its dependency on other motion parameters, such as the commanded velocity $\mathbf{v}^c$, is arbitrary.

These assumptions greatly reduce the complexity and allow for a closed-form solution of the SDE. The justification for them can only come from the experimental use of models that we obtain. This is the case that we will analyze in the reminder of this section.

With the assumptions that we made, the control uncertainty model 8 can be rewritten in the form

$$\mathrm{d}\Delta\mathbf{q} = A\Delta\mathbf{q}\mathrm{d}t + \Sigma\mathrm{d}\mathbf{W} \tag{9}$$

where A and Σ are matrices that define the model. These matrices do not dependent on $\Delta\mathbf{q}$. The purpose of control uncertainty modeling is determining these two matrices.

We have conducted a series of experiments aimed towards estimating matrices A and Σ (see [13]). The result that we obtained is

$$A = A_\omega = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & v \\ 0 & 0 & 0 \end{bmatrix}$$

and

$$\Sigma = \begin{bmatrix} c_1 v^{3/2} & 0.0 & 0.0 \\ 0.0 & c_2 v^{3/2} & 0.0 \\ 0.0 & 0.0 & c_3 \omega^{3/2} \end{bmatrix}$$

Where c_1, c_2 and c_3 are numerical constants. Using the least square fit, their values are estimated to be $c_1 = 0.048$, $c_2 = 0.0055$ and $c_3 = 18.44$.

From the SDE 9, we compute the covariance matrix V that models the error $\Delta\mathbf{q}$. This covariance matrix is

$$V = \int \begin{bmatrix} c_1^2 v^3 & 0.0 & 0.0 \\ 0.0 & c_2^2 v^3 + 2c_3^2 v \int v \int \omega^3 \mathrm{d}t\mathrm{d}t & c_3^2 v \int \omega^3 \mathrm{d}t \\ 0.0 & c_3^2 v \int \omega^3 \mathrm{d}t & c_3^2 \omega^3 \end{bmatrix} \mathrm{d}t \tag{10}$$

What does this model mean? Let us explain this on an example. Let a commanded trajectory be a straight line motion of length 1m and velocity 0.1m/s followed by an arc of radius 1m and angle 90°, with a peripheral velocity of 0.1m/s. The question we want to answer is what is the distribution of robot's position and orientation upon the completion of this motion.

The overall trajectory length is $1 + \pi/2 = 2.57$ meters. The angular velocity during the rotation is 0.1 radian/sec. The overall motion duration is 25.7sec. Let v_{ij} be V's element in place i, j. We have that $v_{33} = 339.9 \int \omega^3 \mathrm{d}t \times 10^{-4} = 339.9 \times 0.1^2 \times 1.57 \times 10^{-4} = 5.336 \times 10^{-4}$. This is the variance in $\Delta\theta$. The standard deviation is $\sqrt{5.336} \times 10^{-2}$ or about 1.3°.

If we proceed with calculation of v_{ij}'s eventually we obtain

$$V = \begin{bmatrix} 0.6042 & 0.0 & 0.0 \\ 0.0 & 4.393 & 4.190 \\ 0.0 & 4.190 & 5.336 \end{bmatrix} \times 10^{-4}$$

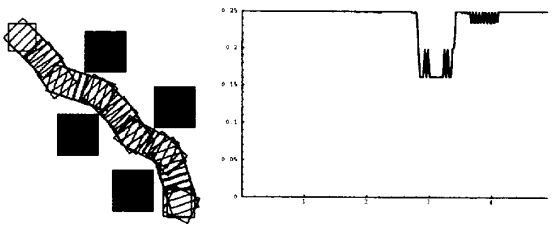

Figure 8: Path planning with the control uncertainty model, $v_0 = 0.25$m/s

which means that the error in x direction (the direction of motion) will have the standard deviation of $\sqrt{0.6042} \times 10^{-2} = 0.0078$ or about 0.8 centimeters, the error in y direction (perpendicular to the direction of motion) will have the standard deviation of $\sqrt{4.393} \times 10^{-2} = 0.02$ or about 2 centimeters. Thus, the overall error is small, due to the low motion velocity.

The model 9 has been implemented in the planning algorithm that takes into consideration the control uncertainty. This algorithm is presented in the following section.

5 The Integration

The objective of this section is a planning algorithm for real-sized robots that incorporates the model of control uncertainty. We present the unification of the path planning method presented in section 3 with the control uncertainty model from section 4. The properties of the planning algorithm from section 3 allow for straightforward generalization that accommodates the control uncertainty.

At the end of this section we present simulation and experimental results. The experiments were done on the mobile platform examined in section 4 and they justify the stochastic model developed there.

Knowing the distribution of robot's position and orientation along the trajectory we can compute the *expected* value of the probability that the robot will be in the free space $\mathcal{C}_f$. Let this expected probability be Ψ^m_{free}. It can be computed from the probability $\Psi_{\text{free}}(\mathbf{q})$ that the robot is in $\mathcal{C}_f$ in configuration $\mathbf{q}$ and the distribution density $\psi_{\Delta\mathbf{q}}$ of the control error $\Delta\mathbf{q}$ using the formula for the expectation:

$$\Psi^m_{\text{free}} = \int \psi_{\Delta\mathbf{q}}(\Delta\mathbf{q})\Psi_{\text{free}}(\mathbf{q} + \Delta\mathbf{q})\mathrm{d}\Delta\mathbf{q} \tag{11}$$

When the probability Ψ^m_{free} is computed the path planning problem can be reduced to the path planning algorithm from section 3. The computation of the integral in 11 is a numerical problem that can be successfully solved using the Gauss-Hermite integration method.

When we introduce the control uncertainty model into the planning algorithm, the resulting trajectory changes in order to accommodate for the increasing uncertainty in robot's position. This uncertainty is a function of robot's velocity and gets higher as robots moves faster.

We have performed a series of simulations with three different initial motion velocities. Figure 8 shows the trajectory computed for the initial velocity $v_0 = 0.25$m/s together with the motion velocity as a function of the path length. The uncertainty does not get very high due to the low motion velocity. However, the robot visibly slows down in the area when it is closest to the obstacles.

The next experiment was performed with the initial velocity $v_0 = 0.5$m/s. The results are given in figure 9. We notice that the trajectory, while still close to the trajectory in figure 8, bends away from the obstacles. Also the motion velocity further decreases in the vicinity of the obstacles.

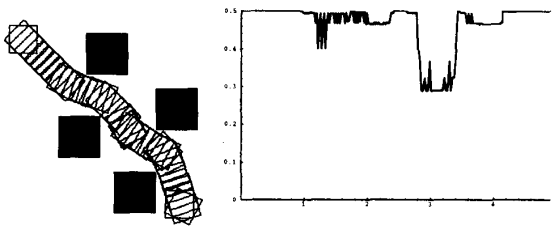

Figure 9: Path planning with the control uncertainty model, $v_0 = 0.5$m/s

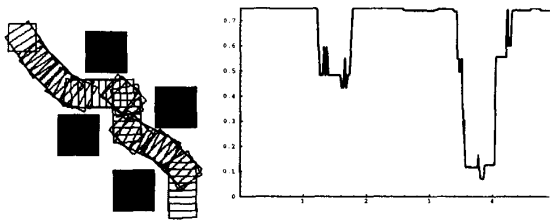

Figure 10: Path planning with the control uncertainty model, $v_0 = 0.75$m/s

The biggest difference from the shortest path trajectory is in figure 10. This figure was obtained with the initial velocity $v_0 = 0.75$m/s. The robot attempts to stay away from obstacles as far as possible while further lowering the velocity. It is important to notice that the lowest success probability along the path was below 0.5. This indicates that due to the control uncertainty the robot is likely to collide with an obstacle. This is the behaviour that we noticed on the actual experiments with the real system.

Figure 11 shows the actual robot performing one of the plans shown above. These experiments are also presented on the video tape that accompanies this work.

6 Conclusion

In this paper we considered the problem of motion planning for mobile robots in the presence of environment and control uncertainty. We developed computationally efficient approximations of optimal trajectories based on the computation of the collision probability. The collision probability serves as a unifying principle in the development of the algorithm, from the case of a point robot through the most general case of a real-sized robot in the presence of control and environment uncertainty.

With regards to the control uncertainty, we developed a new mathematical model that is based on stochastic differential equations. This is important because SDEs are the correct mathematical entity to capture the expanding nature of the control uncertainty.

We have conducted experiments with our algorithm on the mobile platform. The results that we obtained show that the planning algorithms works reasonably well even when only partial knowledge about the environment is available and when the system is not perfect.

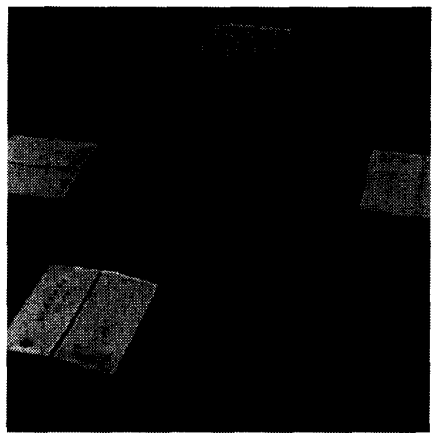

Figure 11: The mobile platform

References

[1] R. C. Brost and A. D. Christiansen. Probabilistic analysis of manipulation tasks: A research agenda. In *Proceedings of the IEEE Conference on Robotics and Automation*, 1993.

[2] B. R. Donald. *Error Detection and Recovery in Robotics*. Springer-Verlag, 1987.

[3] B. R. Donald. Planning multi-step error detection and recovery strategies. *International Journal of Robotics Research*, 9(1):3–60, 1990.

[4] M. Erdmann. Using backprojections for fine motion planning with uncertainty. *International Journal of Robotics Research*, 5(1), 1986.

[5] B. K. P. Horn. *Robot Vision*. MIT Press, 1989.

[6] T. Lozano-Peréz, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[7] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[8] C. Ó'Dúnlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

[9] B. K. Oksendal. *Stochastic differential equations: an introduction with applications*. Springer-Verlag, 2 edition, 1989.

[10] D. K. Pai and M. C. Leu. Uncertainty and compliance of robot manipulators with applications to task feasibility. *International Journal of Robotics Research*, 10(3):200–213, 1991.

[11] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, 1984.

[12] J. T. Schwartz, M. Sharir, and J. Hopcroft. *Planning, Geometry and Complexity of Robot Motion*. Ablex, Norwood, New Jersey, 1987.

[13] A. Timcenko. *Planning Robot Motions in the presence of Uncertainty*. PhD thesis, Columbia University, 1994.

[14] A. Timcenko and P. Allen. Planning velocity profiles from task-level constraints and environment uncertainties. In *Proc. IEEE International Conf. on Robotics and Automation*, 1993.