2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Control and Planning for Vehicles with Uncertainty in Dynamics

Daniel Mellinger and Vijay Kumar

Abstract— **This paper describes a motion planning algorithm that accounts for uncertainty in the dynamics of vehicles. This noise is a function of the type of controller employed on the vehicle and the characteristics of the terrain and can cause the robot to deviate from a planned trajectory and collide with obstacles. Our motion planning algorithm finds trajectories that balance the trade-off between conventional performance measures such as time and energy versus safety. The key is a characterization of the vehicle's ability to follow planned paths, which allows the algorithm to explicitly calculate probabilities of successful traversal for different trajectory segments. We illustrate the method with a six-legged Rhex-like robot by experimentally characterizing different gaits (controllers) on different terrains and demonstrating the hexapod navigating a multi-terrain environment.**

I. Introduction

Planning and control for a mobile robot is a core problem in robotics and has been addressed by many researchers [1]–[4]. Planning generally refers to finding the path in space for a robot to follow while control refers to the methods used to force the robot to follow the planned path. Both aspects work together to achieve some objective which is often the fastest possible execution of a task. This requires planners that can find the shortest distance path that is followable by the given robot and controllers that can accurately track a path while moving the robot as fast as possible along the path. Here we define a trajectory as the combination of a path and the controllers that are used to follow it.

In this paper we consider what happens when significant process noise is added to the dynamics of the system so that controllers cannot track paths with non-negligible error. This uncertainty is generally a function of the type of controller used and the nature of the terrain. There is significant work on the problem of planning with uncertainty in the environment [5], [6]. Other forms of uncertainty have also been considered. In [7] the problem of finding a path for UAVs to avoid being detected by adversaries is addressed. A minimum risk path is found which accounts for uncertainty in the location of the adversaries. In [8] and [9] uncertainty in state estimation is considered. This uncertainty is corrected for by localization algorithms which locate features of the environment in known maps. In order to minimize uncertainty these planners essentially find paths which pass near features in the environment that provide a good degree of localization so as to avoid becoming lost. Our work differs from this approach in that we assume that even with perfect localization the noise in the dynamics is still significant enough to deviate from the paths. Additionally, neither of these approaches explicitly model the dependence of the uncertainty on the type of controller and the characteristics of the terrain.

In this paper we describe a planner that finds trajectories with high probabilities of success while optimizing conventional measures of performance such as execution time, length of path, or energy. Depending on the trade-off between risk and such performance measures as speed, it can find plans that employ fast but potentially "noisy" controllers in open space and slower but more precise controllers when the robot is close to obstacles. In order to generate plans with these features we formulate the planning problem as an optimal control problem minimizing a scalar objective function, J, that is a function of a performance measure associated with the trajectory and the probability that the trajectory is executed successfully. We discretize the problem so that the optimal plan can be found using a search-based planning algorithm.

Section II lays out the generic continuous optimization problem and Section IV explains how we discretize and solve this problem using the incremental planning framework. In Section III we describe a six-legged robot called the RDK to which we apply this method. We experimentally characterize the performance of different controllers for this robot over different terrains to build empirical models that are then used for the planning algorithm. In Section V plans generated with this method for different scenarios are presented. Finally, we demonstrate the algorithm on the RDK along several trajectories using only a laser range-finder for localization.

II. Problem Formulation

The general hierarchical planning and control structure is represented in Figure 1. At the bottom level are the motor controllers for the locomotion elements (feet, wheels, rotors). At the next level we assume there are several types of controllers, $\kappa \in \{1, ..., m_\kappa\}$, that can be used to follow a desired path. These controllers can be simple controllers as in Controller 2 of Figure 1 or more complex hybrid controllers like Controllers 1 and m_κ. At the top level the planner generates trajectories which can be followed by these controllers.

We let the state of the robot be $x \in C$, where C is the configuration space. The controllers take x_{des} and $\dot{x}_{des}$ as inputs and return the low level control signal u. We assume that all controllers nominally follow the path but vary in how closely they actually stay to the desired path. We assume that the faster controllers are inherently noisier than the slower

D. Mellinger and V. Kumar are with the GRASP Laboratory, Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104 USA {dmel,kumar}@seas.upenn.edu

We gratefully acknowledge support from: NSF grant no. IIS-0427313, ARO grant no. W911NF-05-1-0219, ONR grants no. N00014-07-1-0829 and N00014-08-1-0696, and ARL grant no. W911NF-08-2-0004.

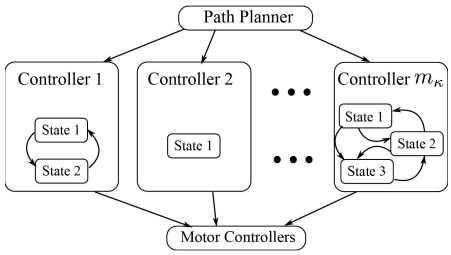

Fig. 1. General Hierarchical Planning and Control Structure

controllers because if the fastest controller was also the most accurate it would simply be used all the time. This method allows C to be classified into some number of different terrain types, $\rho \in \{1,...,m_\rho\}$. Note that the noise on the system, ν, is a function of the controller type, κ, the terrain type, ρ, and the control input u. The generic equations that govern the motion of the system can then be written as:

$$\dot{x} = f(x,u) + \nu(\kappa,\rho,u)$$
$$u = h_\kappa(x, x_{des}, \dot{x}_{des})$$

where $\nu(\kappa,\rho,0) = 0$ so that the dynamics have no drift term. In this work we do not explicitly characterize the noise on the system, ν, but instead its measurable affect on controller performance.

We let C_{free} be the set of collision-free robot states, and C_{obs} be the set of states that result in collisions with obstacles. We wish to find a collision-free path through the environment, $\gamma(\xi) : [0,1] \rightarrow C_{free}$, from some given start state, x_{start}, to a goal state, x_{goal}. Here ξ represents a path coordinate that ranges from 0 at the start of the path to 1 at the end. We constrain $\gamma(\xi)$ to be a followable path for the given robot, for example a path for a Dubin's car is limited to some minimum turning radius. The controller type can change throughout the trajectory so we let the controller type being used at ξ be represented as $\kappa(\xi)$. The problem is to find a path and controllers to be used along that path (a trajectory) that minimize some cost function. We state the generic continuous problem here:

$$\begin{aligned} \text{minimize} \quad & J(\gamma(\xi),\kappa(\xi)) \\ \text{such that} \quad & \gamma(\xi) \in C_{free} \text{ for } \xi \in [0,1] \\ & \gamma(\xi) \text{ is admissible} \\ & \gamma(0) = x_{start} \\ & \gamma(1) = x_{goal} \end{aligned} \tag{1}$$

III. Experimental Setup

A. Overview

We implemented our algorithm on the six-legged RDK, shown in Figure 2, which is similar to the RHex robot [10]. This system is a specific instance of the control structure in Figure 1. At the bottom layer, the six legs are controlled to track trajectories using PD control on the hip motors. At the next layer, different types of gaits described in Section III-B send leg trajectories to the motors. The controllers described in Section III-C utilize the gaits to follow line segments. At the highest level our planner generates the trajectories for the different controllers to follow.

B. Gaits

The alternating tripod gait is used as the starting point for all gaits. To move forward (or backward), the legs within each tripod rotate forward (or backward) along identical trajectories so that one tripod comes into contact with the ground as the other tripod leaves the ground. The exact trajectories followed by legs in this gait are determined by the four parameters shown in Figure 2 where $\phi = 0$ represents the straight down leg position [10]. Here ϕ_s is the stance angle, ϕ_0 is the angular leg offset, t_c is the period of the gait, and t_s is the stance time.

Turning while walking is achieved by changing the relative time legs are in the stance phase between the left and right sets of legs, $t_s(left) = t_s(base) + \Delta t_s$ and $t_s(right) = t_s(base) - \Delta t_s$ [10]. Here a positive Δt_s causes the legs on the right side of the robot to move faster while in contact with the ground than the legs on the left which causes the robot to turn left while walking forward. Turning in place can be achieved by rotating legs on opposite sides of the robot in opposite directions.

Using these parametrized trajectories we found a set of base gaits parameters (ϕ_s, ϕ_0, t_c, $t_s(base)$) that works well at about 0.39 m/s, the slow gait, one that works well at about 0.62 m/s, the fast gait, and one that works well for turning in place. All of these parameters are shown in Table I and were found through testing on carpet. We fix ϕ_0 to be constant across all gaits so the robot reaches the position where one tripod is at $\pi + \phi_0$ and the other is at ϕ_0 twice during every cycle as shown in Figure 2. For this reason, we allow the controllers to switch to a different gait or change Δt_s every half cycle.

The body velocities and angular velocities achieved using several sets of gait parameters are shown in Figure 3. Here each point represents data captured from about 25 steps for a particular set of gait parameters. This plot shows that we were not able to achieve a turning radius of less than about 1 meter for turning while walking forward or backward. This unique set of achievable motions led to the creation of the hybrid controller for path following described in the next section.

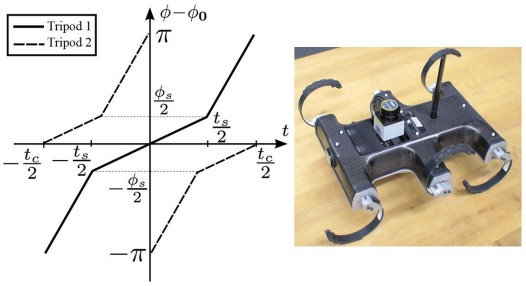

Fig. 2. Left: Leg Trajectories for Forward Alternating Tripod Gait, Right: RDK Hexapod Robot - www.sandboxinnovations.com

C. Controllers

We use a controller that switches between the turn-in-place gait and turn-while-walking gait to follow a line. The feedback controller uses the heading angle error, θ_{error} =

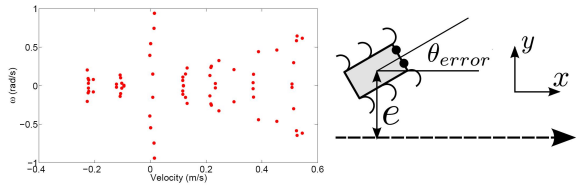

Fig. 3. Left: Achievable Angular Velocities and Velocities for RDK, Right: Line Following Controller

$\theta - \theta_{des}$, and the distance from the center of the robot to the line, e, as shown in Figure 3. Note that the orientation of the line in space is arbitrary since θ_{error} and e are measured relative to the line being tracked.

While the heading error is greater than some maximum heading error, θ_{max}, the robot is commanded to turn in place to reduce the heading error. While the heading error is below θ_{max} the turn-while-walking gait is used and we force the error dynamics to obey $\ddot{e} = -k_d\dot{e} - k_p e$. The gains of this controller are chosen so that the tracking error is critically damped and some rise time, t_r, is achieved by setting $k_p = (3.3/t_r)^2$ and $k_d = 2\sqrt{k_p}$. In this manner only one parameter, t_r, is changed during tuning.

The small angle assumption is appropriate since the robot turns in place whenever θ_{error} is large. Also, for a given base gait v is approximately constant. Under these assumptions $\dot{e} \approx v\theta_{error}$ and $\ddot{e} \approx v\omega$.

As discussed previously, we have control over a parameter, Δt_s, that relates to the turning rate, ω. Through experimentation we have found that this relationship is close to linear while $|\Delta t_s|$ is less than some Δt_s^*. In this range $\omega \approx \eta \Delta t_s$ where η is determined experimentally for each set of base gait parameters. Putting it all together we have a control law:

$$\Delta t_s = max(min(-\frac{k_d}{\eta}\theta_{error} - \frac{k_p}{v\eta}, \Delta t_s^*), -\Delta t_s^*) \qquad (2)$$

This is similar to the controller used in [11] except our method adds the ability to turn in place when the angle error is above some threshold, saturation of the turning rate parameter at some magnitude, and a systematic method for choosing gains. The parameters for the three gaits we developed are shown in Table I. The slow controller ($\kappa = 1$) uses the slow gait and the turn-in-place gait while the fast controller ($\kappa = 2$) uses the fast gait and the turn-in-place gait. For both controllers $t_r = 3s$ and $\theta_{max} = 15°$.

TABLE I

GAIT PARAMETERS AND EXPERIMENTAL DETERMINED VALUES

Gait	$t_s(s)$	$t_c(s)$	$\phi_s(rad)$	$\phi_0(rad)$	$\Delta t_s^*(s)$	$v(\frac{m}{s})$	$\eta(rad)$
Slow	0.3	0.5	0.9	−0.2	0.1	0.39	2.4
Fast	0.25	0.5	1.3	−0.2	0.05	0.62	6.45
Turn-in-Place	0.6	1.0	0.5	−0.2	N/A	N/A	N/A

D. Controller Performance

To characterize controller performance a Vicon motion capture system was used for state estimation. Two types of terrain were tested, a hard carpet ($\rho = 1$) and a rocky terrain($\rho = 2$) consisting of irregular-shaped rocks of average diameter 5cm glued to pieces of plywood. For each combination of controller and terrain type the robot was run over a 2 meter straight line segment trajectory 38 times. Ten representative trajectories are shown in Figure 4 for each combination of controller and terrain. This figure clearly shows how the rocky terrain degrades the tracking performance for both the slow and fast controllers. Videos of these tests can be seen in the video attachment accompanying this paper.

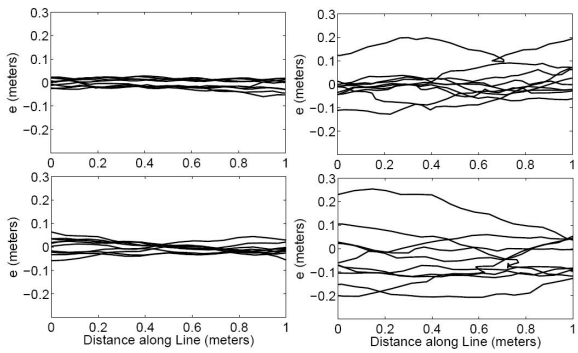

Fig. 4. Ten Representative Trajectories - Slow/Carpet: $\kappa = 1$, $\rho = 1$ (top left), Slow/Rocky: $\kappa = 1$, $\rho = 2$ (top right), Fast/Carpet: $\kappa = 2$, $\rho = 1$ (bottom left), Fast/Rocky: $\kappa = 2$, $\rho = 2$ (bottom right)

IV. PLANNING ALGORITHM

A. Discrete Formulation and Assumptions

We solve the optimal control problem (1) by discretizing C_{free} and using a search-based planning algorithm. We use the A*-based ARA* algorithm [3] which is complete and has optimality guarantees. Normal A* uses a heuristic function, $h(s)$, which is an estimate of the cost to a goal. Weighted A* uses an inflated heuristic, $\epsilon h(s)$ which often results in fewer state expansions and faster searches. The trajectories are guaranteed to be suboptimal by less than the ϵ that the heuristic is scaled. ARA* uses a large heuristic scaling factor and decreases ϵ until $\epsilon = 1$. The constraints of making the path admissible are enforced by using action spaces that obey the constraints as in [2]. To obtain a good heuristic function we run a 2D Dijkstra's search from the goal state using the fastest controller over the 2D discretization of C_{free}. This heuristic gives an underestimate of the cost for the full 3D planning problem.

We based our code on an open source C++ implementation of ARA* [12]. Our algorithm can find a trajectory from the current position to the goal quickly (0.3 secs for a time horizon of 20-30 seconds and a 5 cm length discretization and 22.5° angular discretization of a 4×4m area). This enables the robot to replan at a fast rate and quickly react to changes in a sensed environment. Thus, this approach naturally integrates deliberative planners and reactive controllers.

B. Estimating the Probability of Successful Trajectory Execution

We approximate the condition of navigating the trajectory successfully as staying within a corridor in which there are no obstacles inside the corridor. We define $\delta(\xi)$ as the distance to the closest collision state in C_{obs} at path coordinate ξ. Note that for a point robot $\delta(\xi)$ is simply the distance to the closest obstacle. We define $e(\xi)$ as the distance from the robot to the path at path coordinate ξ. These parameters are illustrated in Figure 5 for a point robot.

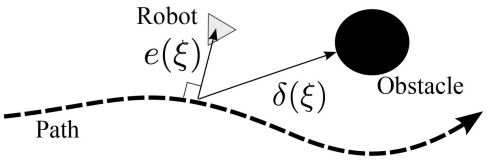

Fig. 5. Point Robot Following a Path Near an Obstacle

As long as the $e(\xi)$ is always less than $\delta(\xi)$ then the robot executes the trajectory successfully. We can write

$$P(S_{cor}) = P(e(\xi) \leq \delta(\xi) \text{ for } 0 \leq \xi \leq 1) \tag{3}$$

where the probably of staying within a corridor around the trajectory is denoted as $P(S_{cor})$.

1) Successful Execution of Trajectory Segments: For planning we must consider smaller segments of the trajectory, so we assume that the trajectory is broken down into n segments. We define the variable q as the segment the robot is currently navigating and δ_i as the minimum value of $\delta(\xi)$ over segment i. We define the event of navigating segment i of the trajectory successfully as S_i.

$$S_i = e \leq \delta_i \text{ while } q = i \tag{4}$$

The probability that the robot stays within the corridor over the entire trajectory is greater than the probability that all segments are navigated successfully, $P(S_{cor}) \geq P(S_1 \wedge S_2 \wedge ... \wedge S_n)$. Note that the LHS and RHS of this equation are approximately equal when the segment length is small relative to the obstacle size. We can then rewrite this equation as a product of conditional probabilities involving navigating individual segments successfully.

$$\begin{aligned} P(S_{cor}) \geq\ & P(S_1)P(S_2|S_1)P(S_3|S_1 \wedge S_2)... \\ & ...P(S_n|S_1 \wedge S_2 \wedge ... \wedge S_{n-1}) \end{aligned} \tag{5}$$

The term $P(S_2|S_1)$ is the probability of successfully executing trajectory segment 2 (S_2) given that segment 1 was successfully executed (S_1). Similarly, the subsequent terms in this equation represent the probability that a segment is traversed successfully given that all the previous ones were.

We next simplify this expression into something which can be used in planning. It is clear that S_i are not independent since the position of the robot at the end of segment $i-1$ is the position of the robot at the beginning of segment i. Here we will make the assumption that the successful traversal of all segments prior to segment i can be approximated by the condition that the robot is within δ_i of the trajectory at the start of segment i. Note that if segment $i-1$ was traversed successfully then the robot is only guaranteed to be within δ_{i-1} of the trajectory at the start of segment i. But since subsequent segments are physically close to each other $\delta_{i-1} \approx \delta_i$. For simplicity we define a new term for this quantity, $P(\tilde{S}_i)$.

$$\begin{aligned} P(\tilde{S}_i) &= P(S_i \,|\, e < \delta_i \text{ at start of segment }) \\ &\approx P(S_i|S_1 \wedge S_2 \wedge ... \wedge S_{i-1}) \end{aligned} \tag{6}$$

This assumption allows (5) to be written in a much simpler form. Using (6) the 2nd through nth terms of Equation (5) can be approximated as $P(\tilde{S}_i)$. The first term in (5) can be approximated as $P(\tilde{S}_1)$ since the plan is formed from the actual start location of the robot which is guaranteed to be on the path.

$$P(S_{cor}) \geq \prod_{i=1}^{n} P(\tilde{S}_i) \tag{7}$$

Now we see that the probability of successfully staying within a corridor around a trajectory is multiplicative in terms that are functions of n segments of the trajectory. To incorporate this cost into the search-based planning framework we take the logarithm of this quantity to make it additive in each segment of the trajectory.

2) Segment Length: The length of the segment plays an important role in the probability that it is successfully traversed. For a given environment, a short segment with a given δ_i should have a higher probability of success than a longer segment with the same δ_i. In order to account for this length each segment i must have a distance metric, l_i, associated with it. Then for each controller over each terrain we determine the probability of success for a particular characteristic length, say L, for all $\delta > 0$. From this data we can determine the probability of success of a segment i of any length, l_i, given $P(\tilde{S}(L))$ for δ_i:

$$P(\tilde{S}(l_i)) \approx P(\tilde{S}(L))^{\frac{l_i}{L}} \tag{8}$$

C. Incorporating the Cost Function into the Planner

We next incorporate this probability into a cost function which minimizes the time to completion of a trajectory and maximizes the log of the probability of successfully staying within a corridor around the trajectory.

$$J(\gamma(\xi), \kappa(\xi)) = t_{traj} - \mu log(P(S_{cor}))$$

The discretized version of the cost function is additive in terms that depend on individual segments and is approximately equal to continuous cost function if the segment lengths are small relative to the size of the obstacles:

$$J(\gamma(\xi), \kappa(\xi)) \approx \sum_{i=1}^{n} t_i - \mu log(P(\tilde{S}_i)) = \sum J_i \tag{9}$$

Here, t_i is the time to traverse segment i and μ is a scaling parameter that defines that relative weight between the time to completion versus the probability of success. Setting μ to 0 will yield the minimum time solution with no regard to the safety of the trajectory. A large μ will result in a trajectory with a high probability of success but at the cost of taking longer to execute.

When ARA* expands a node of the graph a segment is formed which connects two nodes of the graph and is collision free. From the map of the environment the distance to the closest collision state for each segment, δ_i, and the type of terrain the segment is on are found. We consider that if any part of the segment touches rough terrain then that segment is on rough terrain. For each controller, we find $P(\tilde{S}(L))$ for δ_i and the terrain type from experimental data and then use (8) and l_i to find $P(\tilde{S}_i)$. The time to execute the segment, t_i, is calculated based on the length of the path and the controller speed. The controller which minimizes J_i is chosen for that segment. When the planning algorithm terminates and a path is found we have the controller type which should be used over each segment of the path.

Here, a cost for switching controller types could be added but it would multiply the dimension of the state space by the number of controller types. In our application, the fast and slow controllers are very similar so we do not add a cost for switching between them. Note also that we assign no cost for switching between tracking line segments of different angles. This is valid if the turn-in-place gait turns exactly on a point but becomes less accurate the more the turn-in-place gait drifts from a point.

V. Results

A. Data For Planner

From the data presented in Section III-D, we find the probability of staying within $\delta > 0$ of the line for some characteristic length, L, given that the robot started within δ of the line. We chose L to be 1 meter, so the 38 trials were broken down into 76 1-meter sections. The results of this analysis are shown in Figure 6. As expected, running the vehicle faster decreases tracking performance for both terrains. Also, the rocky terrain decreases performance for both vehicle speeds.

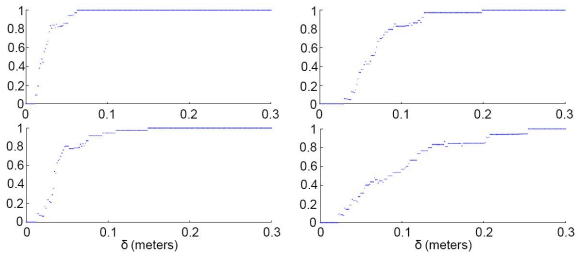

Fig. 6. $P(\tilde{S}(1 \text{ meter}))$ vs. δ - Slow/Carpet: $\kappa = 1$, $\rho = 1$ (top left), Slow/Rocky: $\kappa = 1$, $\rho = 2$ (top right), Fast/Carpet: $\kappa = 2$, $\rho = 1$ (bottom left), Fast/Rocky: $\kappa = 2$, $\rho = 2$ (bottom right)

From the average velocities reported in Table II, note that both controllers result in slightly lower velocity on the rocky terrain than the carpet. The body of the RDK is 35 cm long so these controllers result in velocities ranging from about 1.1 to 1.8 body lengths per second.

B. Example 1

The velocity and probability data were encoded into the cost function used in the planner. The planner was run on a 4×4m map shown in Figure 7 with a length discretization

TABLE II
Controller Speeds

	Carpet: $\rho = 1$	Rocky Terrain: $\rho = 2$
Slow Controller: $\kappa = 1$	0.391 m/s	0.356 m/s
Fast Controller: $\kappa = 2$	0.623 m/s	0.591 m/s

of 5cm and an angle discretization of 22.5°. By changing μ, five distinct plans with different probabilities of success and times to completion emerge from this relatively simple scenario and are shown in Figure 7. Note that these plans account for the finite size of the robot and its legs (50×40cm). In (1) the planner finds the trajectory with minimum time to completion but requires the robot to pass very close to the leftmost obstacle. In (2) the probability of success is slightly increased by the robot traveling through the middle of the first gap. In (3), the robot uses the slow controller through the first gap which further increases the probability of success. The probability of success jumps to around 75 percent in (4) as the robot uses the slow controller through a gap which has a shorter section of rocky terrain. Finally, in (5) the planner finds the trajectory with 100 percent probability of success by traveling far enough away from all obstacles.

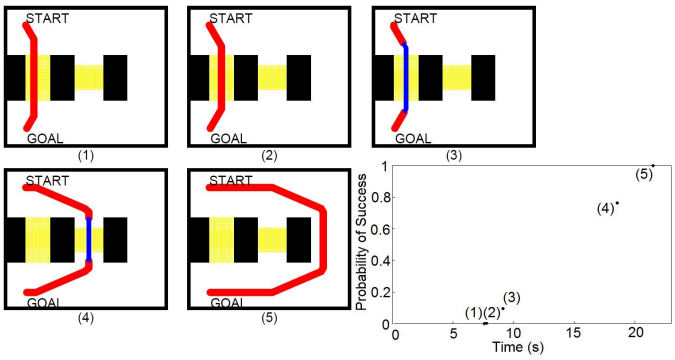

Fig. 7. Planned Trajectories in Order from Fastest (1) to Safest (5), Black-Obstacles, White-Carpet, Yellow-Rocky Terrain, Red-Fast Controller, Blue-Slow Controller and Probabilities of Success and Completion Times

C. Example 2

In this example we demonstrate how our method can be used to find trajectories for complex environments with multiple terrain types. We label the four terrain types as easy, tough, tougher, and toughest, each with progressively worse controller performance. This situation is similar to a real-world situation where precise characterization of the robot's performance on different terrains is not possible. Here we generate plans for a point robot which has the option to use a fast and slow controller. Figure 8 shows some of the many trajectories that can be generated by varying μ. As the robot moves to the right in this map the gaps are safer to navigate because the terrain becomes easier, the gaps becomes wider, or the lengths of the difficult terrain segments decrease. These trajectories have higher probabilities of success but longer execution times as shown in the right portion of Figure 8.

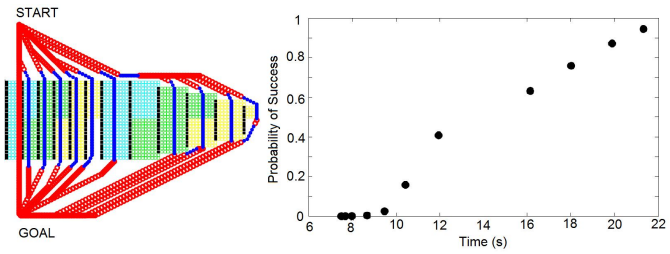

Fig. 8. Multiple Trajectories for Complex Scenario with Four Terrain Types (White-Easy, Yellow-Tough, Green-Tougher, Cyan-Toughest) and Two Controllers (Red-Fast, Blue-Slow)

D. Experimental Implementation

It is not possible to use the Vicon system outside of the lab for state estimation so we used a laser-based localization method based on Section 4.5 of [13] that uses a 2D occupancy map of the environment and scans from a Hokuyo URG-04LX mounted on the RDK. In order to easily extend our planner to any localization method with non-negligible error we just add a buffer of the accuracy of the localization system to all obstacles. For example, our laser-based localization is accurate to approximately 5 cm so we buffer all obstacles by this amount.

For the real world implementation of this planner we use a 5×4m map shown in Figure 9 discretized to 5cm and 22.5°. For a given start and goal location changing μ produces trajectories with different probabilities of success and completion times. Figure 9 shows a sample of three such plans that are found for three different μ values. In (1), a fast but high risk trajectory is found where the fast controller is used throughout the trajectory. In (2), a medium risk trajectory, a slightly longer path is chosen and the slow controller is used when moving close to obstacles. In (3), a low risk trajectory is chosen by avoiding the rocky terrain entirely and taking a long but safe path.

The RDK was run along these trajectories using the laser range-finder for localization. The high risk trajectory was executed successfully less often than the medium risk trajectory. The low risk trajectory was successful on all trials. The data for nine of these experiments are shown in Figure 9. Videos of some of these runs can be seen in the video attachment accompanying this paper. In these trials, the high risk trajectory is executed in 28 seconds, the medium risk in 30.5 seconds, and the low risk in 33 seconds.

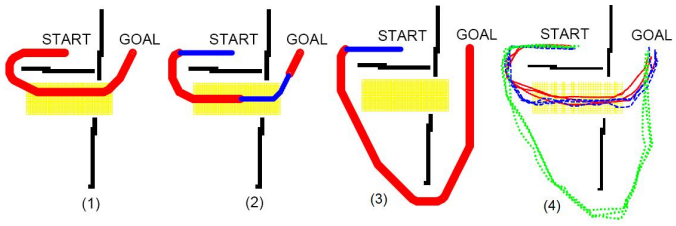

Fig. 9. (1-3) Planned Trajectories from High (1) to Low Risk (3), Black-Obstacles, White-Carpet, Yellow-Rocky Terrain, Red-Fast Controller, Blue-Slow Controller; (4) Experimental Data for RDK Following Trajectories (1)-Solid Red, (2)-Dashed Blue, and (3)-Dotted Green

VI. Concluding Remarks

In this paper, we have addressed the problem of planning to account for uncertainty in vehicle dynamics. We efficiently solve the problem by incorporating it into the search-based planning framework. Our planner chooses trajectories which balance the trade-off between safety and speed. We demonstrated the method on the RDK hexapod robot which executed several trajectories of varying degrees of safety and speed in a multi-terrain environment using only a laser range-finder for localization.

Future work includes extending this method to larger environments and more complex terrain. Larger, more open environments will likely introduce significant errors in the laser localization method. We plan to address this issue by introducing localization errors which are a function of environment into the process model. Extremely difficulty terrain will introduce even greater errors into the dynamics of the system. We plan to deal with this by developing new controllers based on a low-level dynamic model of the RDK.

VII. Acknowledgments

The authors thank Jon Fink, Adam Komoroski, Dr. Haldun Komsuoglu, Alex Kushleyev, Dr. Max Likhachev, and Dr. Nathan Michael for their contributions to the work.

References

[1] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[2] M. Likhachev and D. Ferguson, "Planning long dynamically-feasible maneuvers for autonomous vehicles," *Proceedings of Robotics: Science and Systems*, 2008.

[3] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality." *Advances in Neural Information Processing Systems*, vol. 16, 2003.

[4] R. Philippsen and R. Siegwart, "Smooth and efficient obstacle avoidance for a tour guide robot," *International Conference on Robotics and Automation*, 2003.

[5] M. Likhachev and A. Stentz, "PPCP: Efficient probabilistic planning with clear preferences in partially-known environments," *Proceedings of the National Conference on Artificial Intelligence*, 2006.

[6] M. Spaan and N. Vlassis, "A point-based pomdp algorithm for robot planning," *International Conference on Robotics and Automation*, 2004.

[7] M. Jun and R. D'Andrea, "Path planning for unmanned aerial vehicles in uncertain and adversarial environments," *Cooperative Control: Models, Applications and Algorithms*, pp. 95–111, 2002.

[8] J. Gonzalez and A. Stentz, "Planning with uncertainty in position using high-resolution maps," *International Conference on Robotics and Automation*, 2007.

[9] R. He, S. Prentice, and N. Roy, "Planning in information space for a quadrotor helicopter in a GPS-denied environment," *International Conference on Robotics and Automation*, 2008.

[10] U. Saranli, M. Buehler, and D. Koditschek, "Rhex - a simple and highly mobile hexapod robot," *International Journal of Robotics Research*, 2001.

[11] S. Skaff, G. Kantor, D. Maiwand, and A. Rizzi, "Inertial navigation and visual line following for a dynamical hexapod robot," *Intl. Conference on Intelligent Robots and Systems*, 2003.

[12] M. Likhachev. Search-based planning library (SBPL), http://www.seas.upenn.edu/ maximl/software.html.

[13] T. Bailey, "Mobile robot localisation and mapping in extensive outdoor environments," Ph.D. dissertation, The University of Sydney, 2002.