Ryan Spring
rds4
11/25/2014 Project 5 – Planning Under Uncertainty

Project Description / Motivation

The objective of this project is to model a steerable needle with action uncertainty in different environments. The steerable needle is modeled with a Dubin's car. The car has a bang-bang controller that causes the robot to turn completely in the left or right direction. The action uncertainty is represented using Gaussian white noise that prevents the car from turning a deterministic manner. All mechanical systems operate within certain tolerances, creating an amount of action uncertainty. A requirement for general purpose robots is to function robustly under ambiguous conditions in the real world.

Approach

First, I constructed some basic environments and the dynamics of the Dubin's car according the section III of the SMR paper. I verified my setup and the Dubin's car by using an RRT Control planner. My next step was to implement the Stochastic Motion Roadmap (SMR) to generate the policy for the Dubin's car. I modified the RRT control planner because I wanted to use the Nearest Neighbors data structure to efficiently find the closest node in the SMR to a random sample. I divided the SMR planner into four different parts. Sampling the configuration space and learning the transition probabilities composed the learning phase. The value iteration algorithms and executing the policy in the environment were a part of the query phase. After implementing the basic algorithm, I debugged several major errors such as segmentation faults, updating the current state values with the future value estimate, and using too much Gaussian noise. I fixed the segmentation faults by using smart pointers to prevent the program from deleting the same object multiple times. I verified that the SMR planner was running correctly by running a policy for 100 iterations to measure the probability of success was reasonable and visualizing a correct path generated by the policy. Finally, I added multi-threaded support for generating the transition probabilities and the value iteration algorithm to improve the performance of the SMR planner.

Experiments

My three primary experiments were to measure the effect of increasing the standard deviation of the Gaussian noise, the number of states in the SMR representation, and the number of the threads running the algorithm. I used the OMPL benchmark class to generate the SMR policy and then use the policy to execute a path to the goal for 100 iterations.

Computer Specifications

- Intel Core i7-4930K 3.20 GHz
- 16 GB RAM
- Nvidia GeForce GTX 660 Ti
- OMPL VirtualBox w/ Ubuntu 14.04 LTS

Environments

The environments are derived from the examples in the SMR paper. The primary difference is the environments are 1 x 1 squares while the environments in the paper are sized 10 x 10.

Ryan Spring
rds4
11/25/2014 Project 5 – Planning Under Uncertainty

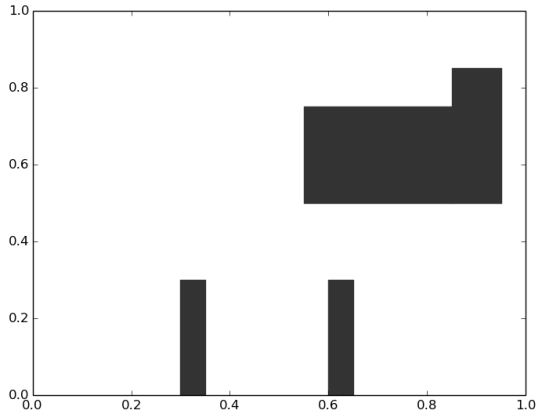


Figure 1 Environment 1

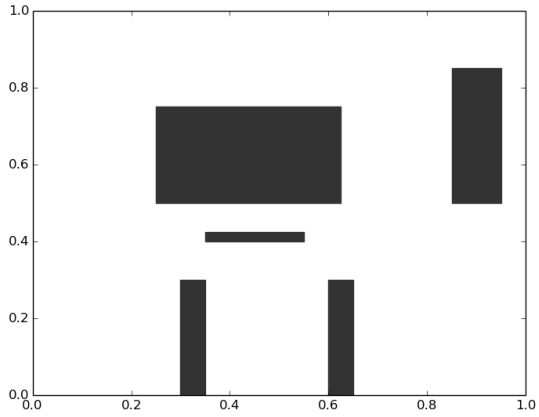


Figure 2 Environment 2

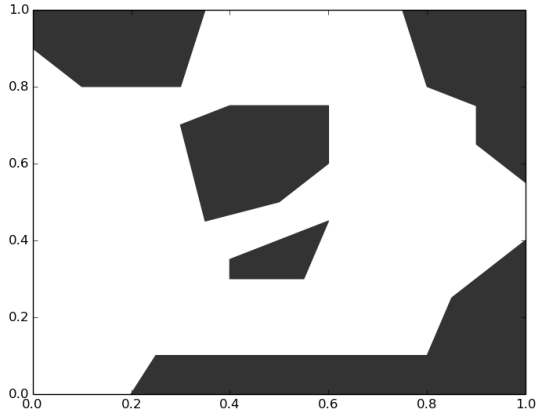


Figure 3 Environment 3

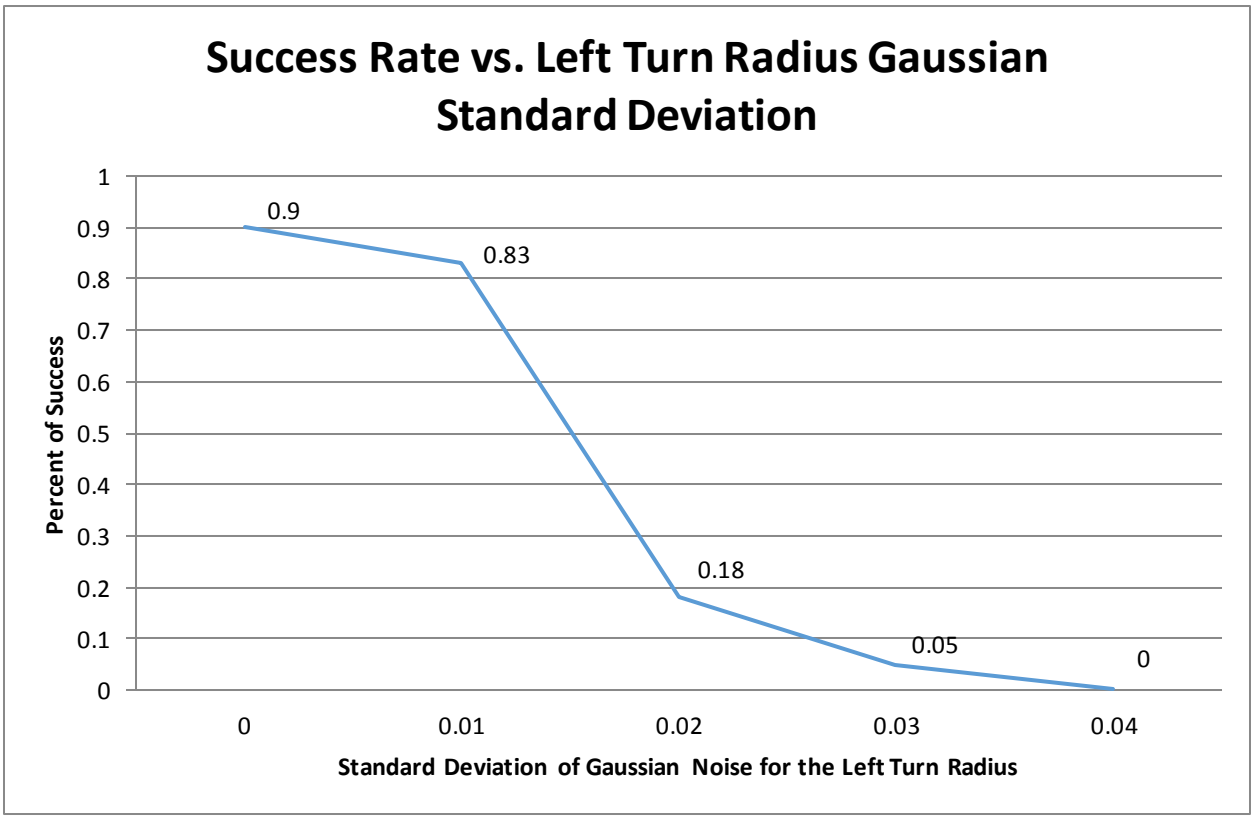
Ryan Spring
rds4
11/25/2014 Project 5 – Planning Under Uncertainty

Results

The default settings for the SMR policy experiments is 50,000 nodes, 20 transition samples per nodes, left turning radius ~ N(0.05, 0.01), right turning radius ~N(0.05, 0.02), left forward motion ~ N(0.25, 0.1), right forward motion ~ N(0.25, 0.2), and environment 3.

How sensitive is the probability of success to the standard deviation of the Gaussian noise distribution?

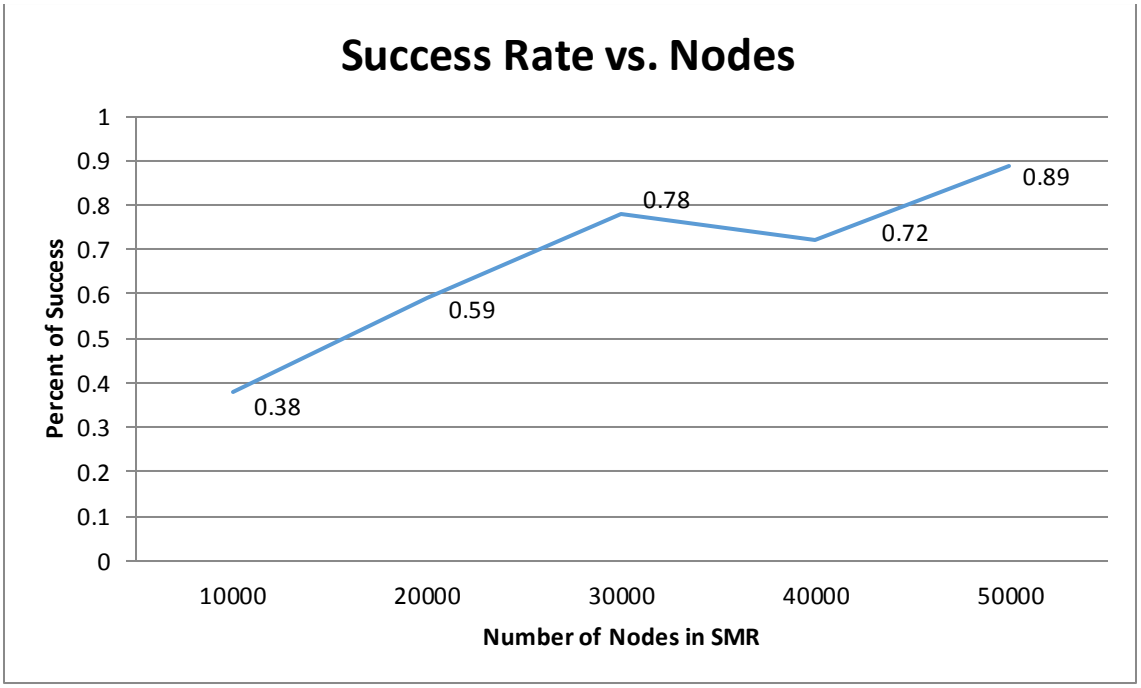
The standard deviation for right turns is double the standard deviation for left turns. The probability of success drops dramatically as the Gaussian noise increases from 0.01 to 0.02. This change indicates that the SMR algorithm is very sensitive to the amount of uncertainty in the system.



How does the number of states in the SMR representation affect the probability of success?

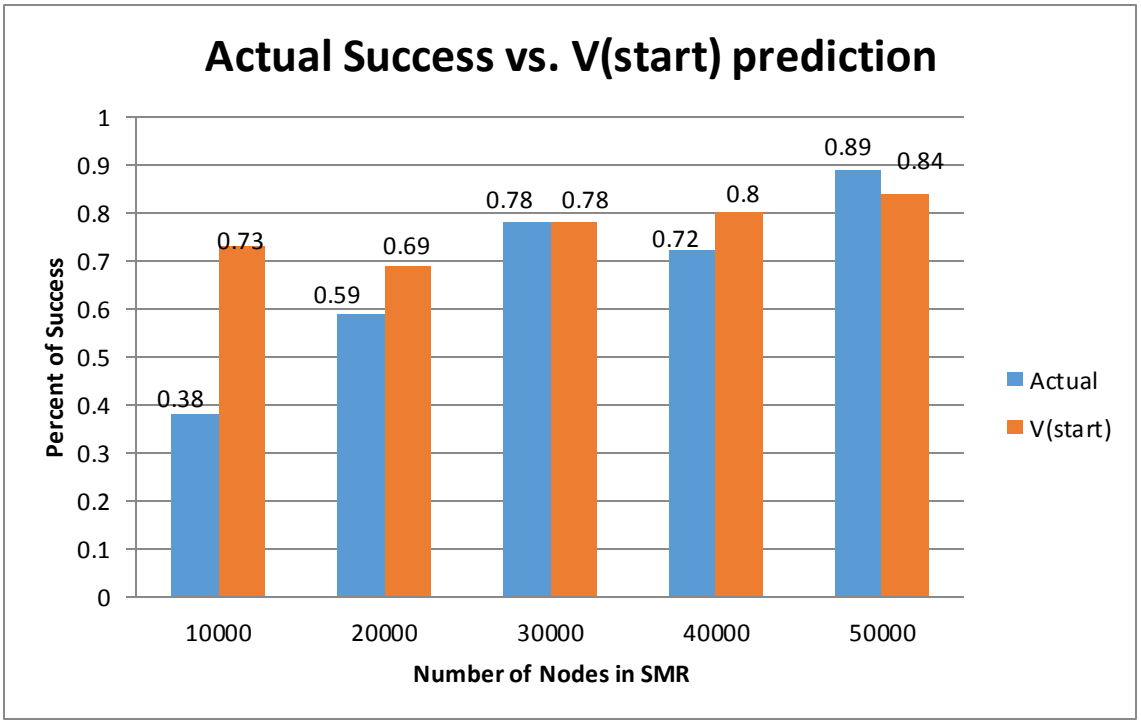
As the number of states increases, the probability of success increases. The growth of the probability of success seems to plateau as the number of nodes of increases. The number of nodes in the SMR graph corresponds with the fidelity of the model. If the number of nodes is low, the nearest neighbor to a state is too far away to accurately map to a random sample. If the number of nodes is high, the saturation of the configuration space is high. Thus, there are multiple nodes that are a potential nearest neighbor for a random sample so the accuracy of the matching improves more slowly.

Ryan Spring
rds4
11/25/2014 Project 5 – Planning Under Uncertainty



Is there any correlation in the difference of V (start) and the simulated probability of success from the start as n increases?

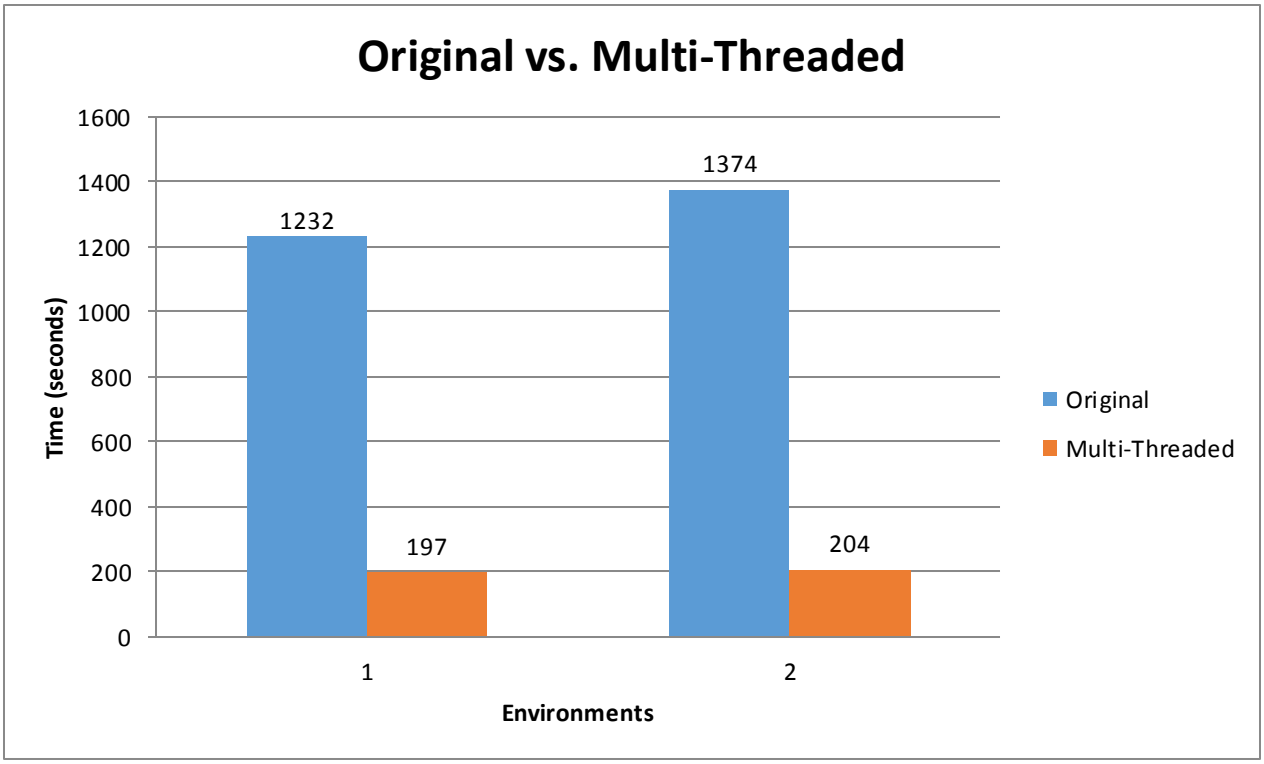
When the number of nodes in the SMR representation is low, the difference between the V (start) and actual success rate is high. However, when the representation contains more than 30,000 nodes, the difference decreases to less than 10% of the probability of success.



Ryan Spring
rds4
11/25/2014 Project 5 – Planning Under Uncertainty

What is the performance increase of multi-threaded support?

The multi-threaded version utilized the 6 threads allocated to the VirtualBox. The performance improved on average 6.5 times between the original and multi-threaded versions without altering the probability of success for the SMR policy.



Conclusion / Difficulty / Time

The SMR policy usually generates a similar path to the goal. The main difference between paths occur when the policy over shoots the goal and it loops around multiple times to reach the goal. The policy does not always reach the goal but the success rate was between 75 – 90% depending on the number of nodes in the SMR representation, the size of the goal region, and the amount of action uncertainty. The biggest hassle was presetting the Gaussian noise distribution according to the SMR paper. I thought using the standard deviation in the paper would act as a good baseline. I learned that to debug systems with uncertainty you should start without a deterministic system and see if the algorithm solves that problem first.

Difficulty: 4 out of 10

Time: about 24 hours – About 8 hours coding the algorithm and 16 hours debugging issues