Generating Safe Trajectories in Stochastic Dynamic Enviroments

University of St Andrews

CS4099: Major Software Project

Author:
Alexander Wallar

Supervisor:
Dr. Michael Weir

March 25, 2015

Abstract

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated.

The main text of this project report is NNN words long, including project specification and plan.

Contents

List of Figures

Chapter 1

Introduction

Chapter 2

Context Survey

Chapter 3

Ethics

Since this project does not use any personal or classified information and the algorithms developed have not been tested with humans as obstacles, this project does not raise any ethical concerns. One could argue that the algorithms developed could be used in the robot uprising, but that is not the intention of the work and the robot uprising is not set to come for another few decades.

Chapter 4

Software Development Framework

Chapter 5

Objectives

5.1 Primary

The main objective of this work is to develop an algorithm that generates a quantitatively safe trajectory for a robot through an uncertain dynamic environment by utilizing information about how dynamic obstacles are going to move in the future. This can be simply stated as determining the curvature that minimizes the line integral over the dynamic cost distribution for a given set of dynamic obstacles.

$$J(C, A) = \int\limits_{C} \exp\Big(P(x, y, t_0, t_m, A) + 1\Big)\, \mathrm{d}s \tag{5.1}$$

Eq. 5.1 describes the objective function, J, that needs to be minimized with respect to the curvature in order to determine the safest path through the environment. In Eq. 5.1, the function P is the cost surface for a given time interval and set of obstacles. More description about P is given in Sec. 7.1.2 and is formally defined in Eq. 7.2. More precisely, the objective of this work to develop an algorithm that will provide an approximate solution to Eq. 5.2 which will return the minimum cost path through a environment for a given set of obstacles. The solution is described in Sec. 7.2

$$\Gamma(A) = \arg\min_{C}\ J(C, A) \tag{5.2}$$

5.2 Secondary

The main secondary objective for this work is to show that the proposed solution can provide safer paths than standard planners such as potential fields by leveraging information about how the obstacles move through the environment. Quantitative and qualitative experiments have been conducted that provide evidence that the proposed solution does indeed produce safer paths based on the safety metrics that have been devised for this work. These results are shown in Ch. 10.

Chapter 6

Planner Methodology

As with any research project, many different attempts were made to come up with a solution to the objectives state in Ch. 5. Three different techniques were developed in sequence to try and provide a solution that best matched the sought behaviour for the planner. The first attempt was a simple potential field that would take into account the predicted trajectories of the obstacles, and leverage this information to provide safer paths. The second attempt included generating a probabilistic roadmap in relative space-time and using stock graph search algorithms such as Dijkstra's algorithm and Edmonds' algorithm to derive low costs paths through the environment. The last attempt, and the most successful is a planner that uses a two dimensional probabilistic roadmap to sample the search space and then uses Best First Search to expand nodes in space-time to determine the minimum cost path through the dynamic environment. These three attempts are described individually and more detail in this section.

6.1 Potential Fields

Using potential fields was an initial attempt to plan through uncertain dynamic environments since they are frequently used to plan around dynamic obstacles due to their reactive behaviour. The difference between the standard potential field implementations and the one developed for this project was that the repulsive obstacle field at a position (x, y) was proportional to the cost distribution at (x, y) for a given time interval. This is shown more formally in Eq. 6.1.

$$U_{rep}(p, t_0, t_m, A) = k \cdot P(p_x, p_y, t_0, t_m, A) \tag{6.1}$$

In Eq. 6.1, the function P is defined in Eq. 7.2, and $k > 0$ is a constant. The attractive potential field was kept the same as the standard potential field implementation for robotic motion planning as shown in Eq. 6.2

$$U_{\text{att}}(p, g) = \frac{c}{||p - g||^2 + \epsilon} \tag{6.2}$$

In Eq. 6.2, ϵ is a constant such that $0 < \epsilon < c$ and is used to ensure that the function does not have a singularity and c is a scaling constant such that $c > 0$. The potential field planner would use the sum of these two fields to measure the potential through the environment in order to eventually reach the goal by successively moving to the area within the robot's sensing radius that had the minimal potential. Algo. 1 describes more formally how the potential field planner generates a path through the environment.

Through some qualitative testing, these types of potential fields were still leading the robot into unsafe areas and caused the robot to collide with the dynamic obstacles regardless of velocity of the obstacles and the velocity of the robot. After some manipulation of the constants used for the repulsive and attractive

Algorithm 1 PF(q, g, O, A, R)

$q_{min} \leftarrow q$
$p_{min} \leftarrow \infty$
$\theta \leftarrow 0$
while $\theta \leq 2\pi$ **do**
  $q' \leftarrow q + \delta t \cdot s \cdot \textsc{Rot}(\theta)$
  $p \leftarrow U_{rep}(q', O \cup A) + U_{att}(q', g)$
  if $p < p_{min}$ **then**
    $p_{min} \leftarrow p$
    $q_{min} \leftarrow q'$
  $\theta \leftarrow \theta + \delta\theta$
  for all $a \in A$ **do**
    $\textsc{Step}(a)$
if $||q_{min} - g|| < R$ **then**
  return $\{p_{min}\}$
return $\{q_{min}\} \cup \mathrm{PF}(q_{min}, g, O, A, R)$

potentials, there was only a nominal improvement which lead the author to move towards sampling based motion planning techniques which are outlined in Sec. 6.2 and Sec. 6.3.

6.2 Space-time Roadmap

The second attempt at devising a solution to the primary objective in Ch. 5 was to create a three dimensional probabilistic roadmap that can capture the connectivity of a two dimensional surface in space-time. A probabilistic roadmap (PRM) is a method of creating an undirected, weighted graph, (V, E, W), that represents the connectivity of the search space by randomly sampling points and connecting them such that if $(i, j) \in E$, then both i and j must not collide with an obstacle, $||i - j|| \leq d$ where d indicates the maximum distance away connected nodes can be from one another, and there must not be a collision with any obstacle along the edge from i and j [1]. For the attempted space-time PRM, each node would be a vector, (x, y, t), which represents a two dimensional location, (x, y), at a certain absolute time t, the graph was directed such that for an edge, (i, j), $i_t < j_t$, and instead of randomly sampling a point in the environment, a node in the graph would be randomly selected and propagated forward in time by some random change in time such that the constraints for the roadmap are still satisfied.

With the generated roadmap, the first thought was to use a graph search algorithm such as A* or Dijkstra's algorithm to find the path through the environment that had the lowest overall weight. The weight for an edge, (i, j), defined as the line integral over the cost surface for a given set of dynamic obstacles with a time interval of $[i_t, j_t]$. The notion of a cost distribution is described in Ch. 7 and the formal equation for this line integral is given in Eq. 7.5. This space-time roadmap approach yielded mixed results. The robot would sometimes evade the obstacles, but with the incidence, the planner would lead the robot directly into a collision with a dynamic obstacle. After some testing, it was discovered that since graph search algorithms seek to find the path with minimum combined weight through the graph, these algorithms are biased to return paths with a smaller number of vertices. This is because paths with a larger number of vertices will have a higher overall weight. Since shortest path algorithms try to minimize this overall weight, paths which may be safer but may take longer not be returned by these algorithms.

To overcome this, instead of searching over the entire graph, the search could be occur over the minimum spanning tree of the graph. Since the roadmap is a directed graph, Edmonds' algorithm [2] was used since other minimum spanning tree algorithms such as Kruskal's algorithm and Prim's greedy algorithm only work on undirected graphs. Using the minimum spanning tree would minimize the maximum cost associated with a path from the initial configuration to the goal configuration thus moving the robot away from high cost areas in space-time. Through qualitative analysis, this approach was shown to still lead the robot to high cost areas and even into collisions with obstacles.

This approach using a three dimensional probabilistic roadmap did not work in practice regardless of the search algorithm because of the number of nodes that need to be sampled in space-time in order for it to be effective. The roadmap indicates where in the environment the robot is able to travel to from a starting location in space-time. If the number of nodes is too small the, the goal may not even be in the graph, and thus the robot will never reach it. Also, the less nodes in the graph the less optimal the generated path is, but the more nodes added to the graph, the more computationally difficult it becomes to search. Lastly, the main flaw with this approach is that biases the sampling to areas that already have a high sample density and therefore may not sample nodes in the goal area without having a high number of nodes in the graph.

6.3 Probabilistic Roadmap With Best First Search

.

Chapter 7

Design

7.1 Dynamic Obstacles

As a main component of this work, dynamic obstacles needed to be designed such that one could quantify their trajectories, initial configurations, and their level of uncertainty. This section introduces the definition of a dynamic obstacle used throughout this work along with how it is represented to the planner and its simulated & predicted equations of motion.

7.1.1 Definition

A dynamic obstacle is defined as a 5-tuple, $a = (I, \dot{\zeta}, \epsilon, \xi, T)$ where I is the initial configuration of the obstacle, $\dot{\zeta}$ is a function, $\dot{\zeta} : \mathbb{R}^+ \to \mathbb{R}^2$, representing the velocity of the obstacle, ϵ is used to define a random variable $\rho \sim \mathcal{U}(-\epsilon, \epsilon)$ that is injects noise into an obstacle's trajectory shown in Eq. 7.4 where $\mathcal{U}$ is a uniform distribution, ξ is the current configuration used for prediction, and T is the time that the obstacle was in configuration ξ. The variables ξ and T are dynamic variables and are updated throughout the execution of the algorithm and are used to determine when it is appropriate for the algorithm replan and find a new path through the environment using more up to date information. This is explained in Sec. 7.2. The variables ξ and T are initially set to I and 0 respectively.

7.1.2 Cost Function

Unlike in the previous work, dynamic obstacles are represented by cost distributions that resemble probability density functions. The difference being is that these cost distributions do not have a unit integral. These cost distributions are used to describe where the obstacle is going to be in within a time interval and can be generated by a third party system, such as a motion capture system. There is an assumption that for a given interval, $\mathcal{T} = [t_0, t_m]$, the highest cost with the smallest uncertainty will be at $t = t_0$ and the lowest cost with the highest uncertainty will be at $t = t_m$. Under this assumption, the cost function models how the obstacle may diverge from its current trajectory as time increases. With these assumptions, the cost function, $P_a : \mathbb{R}^2 \times (\mathbb{R}^+)^2 \to \mathbb{R}$, represents represents the cost surface for a given obstacle within a given time interval. Eq. 7.1 formally defines the cost function for a single obstacle.

$$P_a(x, y, t_0, t_m) = \int_{t_0}^{t_m} \mathcal{N}(\zeta_a(t), \alpha \cdot (t - t_0)^2 + \beta, x, y) \cdot (t_m - t)^\gamma \, \mathrm{d}t \tag{7.1}$$

In Eq. 7.1, $\mathcal{N}(\mu, \sigma^2, x, y)$ is the evaluation of a 3D normal distribution centered at (μ_x, μ_y) with a variance of σ^2 at (x, y). Fig. 7.1 shows an example of $\mathcal{N}$. This equation models how the uncertainty of obstacle trajectory prediction increases over time by increasing the standard deviation of the Gaussian distribution as the time increases. Likewise, this function multiplies the Gaussian distribution by a factor of $(t_m - t)^\gamma$ where $\gamma \geq 1$ which gives higher costs to times closer to t_0.

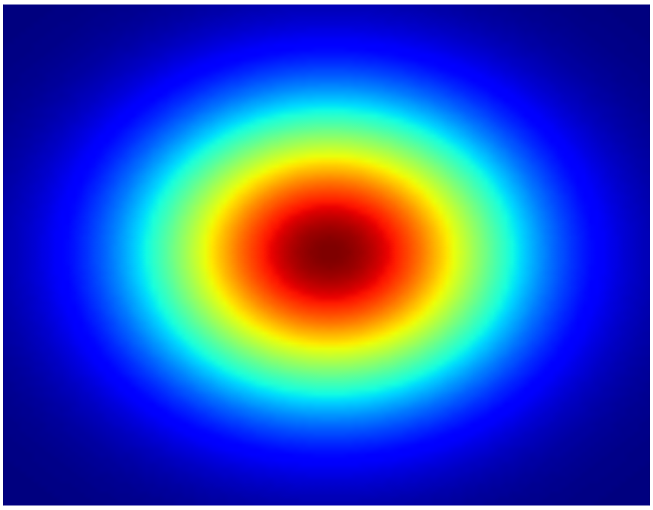

Figure 7.1:

A cost function is also needed that can incorporate the cost distributions for multiple dynamic obstacles within the environment. The cost function used in this work, $P : \mathbb{R}^2 \times (\mathbb{R}^+)^2 \times \mathcal{A} \to \mathbb{R}$ where $\mathcal{A}$ is the set of all possible sets of dynamic obstacles, calculates the average cost at a point (x, y) within a given time interval for a given set of agents. This is shown formally in Eq. 7.2.

$$P(x, y, t_0, t_m, A) = \frac{\sum\limits_{a \in A} P_a(x, y, t_0, t_m)}{|A|} \tag{7.2}$$

An example of how P changes over time is shown in Fig. 7.2. In that example, two dynamic obstacles are placed in the scene and given sinusoidal velocities. In this example the time interval, δt, is kept constant throughout the simulation, i.e. $t_m = t_0 + \delta t$, for all $t_0 \in [0, T - \delta t]$ where T is the length of the simulation. Since the velocity does not remain constant in the example, the cost distribution elongates an shrinks based on the acceleration of the obstacle. For instance, in the first and last images in Fig. 7.2, the cost is contained to a small area due to the velocity equations of the obstacles being at their minimum and in the fourth image,the cost is more spread out through the environment because the velocity is at its maximum.

7.1.3 Equations of Motion

The motion of a dynamic obstacle is defined by the velocity equation, the initial configuration, the amount of uncertainty. Defining the obstacle's trajectory in terms of its velocity makes it easier to model when creating scenes. The equation of motion for the dynamic obstacle is shown in Eq. 7.3.

$$\zeta_a(t) = \begin{cases} \xi_a + \displaystyle\int_{T_a}^{t} \dot{\zeta}_a(\lambda)\, \mathrm{d}\lambda & \text{if } t \geq T_a \\ \tilde{\zeta}_a(t) & \text{if } t < T_a \end{cases} \tag{7.3}$$

In Eq. 7.3, $\tilde{\zeta}_a$ represents the observed trajectory of the obstacle whereas ζ_a corresponds to the predicted trajectory of the obstacle. This disambiguation is needed because the planner needs to be able to extrapolate the future movements of a dynamic obstacle. The variables, ξ_a and T_a are dynamically

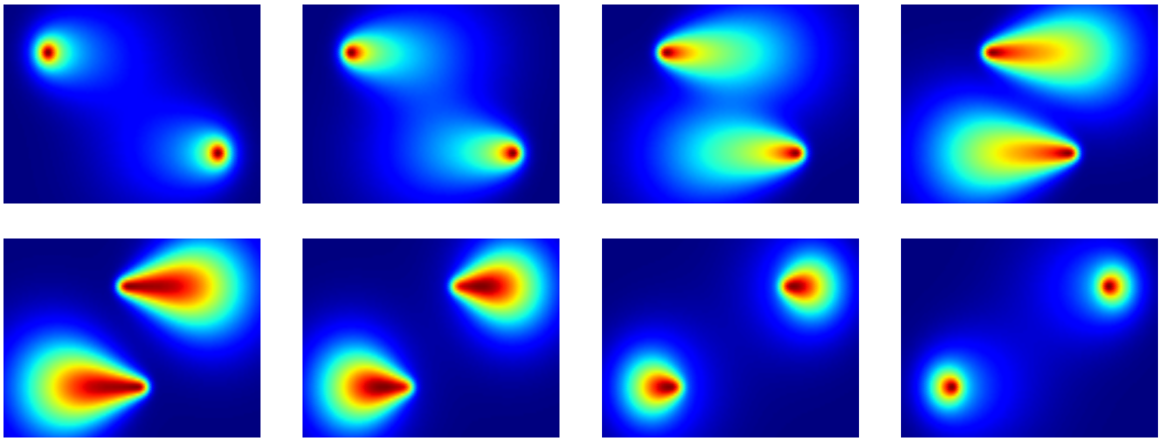

Figure 7.2: Cost distributions indicating the likelihood that an agent will be at a certain location within a given time interval. These figures show how this distribution changes over time (left to right, top to bottom)

updated when the planner replans and are initially set to I_a and 0 respectively. The need for this and how it is designed will be discussed in Sec. 7.2

For the experiments, the motion of the obstacles are simulated by adding a random variable, $\rho \sim \mathcal{U}(-\epsilon, \epsilon)$ to the trajectory during the integration of the velocity equation. This form of stochasticity allows the obstacle to diverge from its specified trajectory whilst maintaining the same velocity equation. This means that the obstacle will not exhibit random motion around its specified path, but rather is able to diverge completely. Also, by adding the random variable to the velocity equation during integration, the obstacle will not "jump" to a new location, but will gradually diverge and because it is analogous to the way the random variable was used during the numerical integration in the code. The definition of $\tilde{\zeta_a}$ is shown in Eq. 7.4. For this equation, it is assumed that the function only computes a value for any given time value, t, only once.

$$\tilde{\zeta_a}(t) = I_a + \int_0^t \dot{\zeta}_a(\lambda) + \rho \, \mathrm{d}\lambda \tag{7.4}$$

7.2 Planning Algorithm

$$C(i, j, t_0, t_m, A) = \int_0^1 \exp\Big(P(x(\lambda), y(\lambda), t_0, t_m, A) + 1\Big) \cdot ||i - j||_2 \, \mathrm{d}\lambda \tag{7.5}$$

Where $x(\lambda) = (j_x - i_x) \cdot \lambda + i_x$ and $y(\lambda) = (j_y - i_y) \cdot \lambda + i_y$ are the parametric equations of the line from i to j.

Algorithm 2 $\textsc{Roadmap}(n, d, w, h, O)$

Input:
n: Maximum number of samples
d: Maximum distance between neighbouring nodes
O: Set of obstacles
Output:
An unweighted graph of points describing the connectivity of the environment

for $i = 1$ **to** n **do**
    $q \leftarrow \textsc{RandomPoint2D}(w, h)$
    if $\bigwedge_{o \in O} \neg\textsc{Collision}(o, q)$ **then**
        $V \leftarrow V \cup \{q\}$
for all $q_i \in V$ **do**
    for all $q_j \in V$ **do**
        if $q_i \neq q_j \wedge ||q_i - q_j|| \leq d$ **then**
            $E \leftarrow E \cup \{(q_i, q_j)\}$
return (V, E)

Algorithm 3 $\textsc{GetPath}(n, d, w, h, \delta, p, g, O, A, R)$

Input:
n: Maximum number of samples for the roadmap
d: Maximum distance between neighbouring nodes in the roadmap
w: Width of the scene
h: Height of the scene
Output:

$(V, E) \leftarrow \textsc{Roadmap}(n, d, w, h, O)$
$\Pi \leftarrow \emptyset$
$q \leftarrow p$
$t \leftarrow 0$
while $||\textsc{Back}(\Pi) - g||_2 > R$ **do**
    $\pi \leftarrow \textsc{SearchGraph}(V, E, R, A, q, g, t)$
    for all $(i, t') \in \pi$ **do**
        $\Pi \leftarrow \Pi \cup \{i\}$
        for all $a \in A$ **do**
            $\textsc{Step}(a)$
        if $\bigvee_{a \in A} ||\tilde{\zeta_a}(t') - \zeta_a(t')|| > \delta$ **then**
            for all $a \in A$ **do**
                $\textsc{Update}(\zeta_a, \tilde{\zeta_a})$
            $q \leftarrow i$
            $t \leftarrow t'$
            break
return Π

Algorithm 4 SEARCHGRAPH(V, E, R, A, p, g, T)

```
Q ← PriorityQueue()
D ← Dictionary()
𝒫 ← Dictionary()
Insert(Q, p, T)
while ¬Empty(Q) do
    (q, t) ← Pop(Q)
    if ||q − g|| ≤ R then
        return BacktrackPath(p, g, 𝒫)
    S ← ∅
    N ← Neighbours(V, E, q)
    for all n ∈ N do
        t′ ← ||q − n||/s + t
        𝒫_(n,t′) ← (q, t)
        c ← ψ · C(q, n, t, t′, A) + ω · D_η
        D_n ← D_n + 1
        Q ← Insert(Q, (n, t′), c)
```

Algorithm 5 BACKTRACKPATH$(p, g, \mathcal{P})$

```
q ← g
S ← Stack()
while 𝒫_q ≠ p do
    S ← Push(S, q)
    (q, t) ← 𝒫_q,t)
S ← Push(S, p)
return S
```

Chapter 8

Implementation

Chapter 9

Experimental Setup

9.1 Metrics

$$MinDist(\Pi) = \min_{t \in \mathcal{T}} \min_{a \in A} ||\zeta_a(t) - \Pi(t)|| \tag{9.1}$$

$$MaxCost(\Pi) = \max_{t \in \mathcal{T}} P_A(\Pi(t)) \tag{9.2}$$

$$AvgCost(\Pi) = \int_{\mathcal{T}} P_A(\Pi(t)) \, \mathrm{d}t \tag{9.3}$$

Chapter 10

Results

10.1 Safety

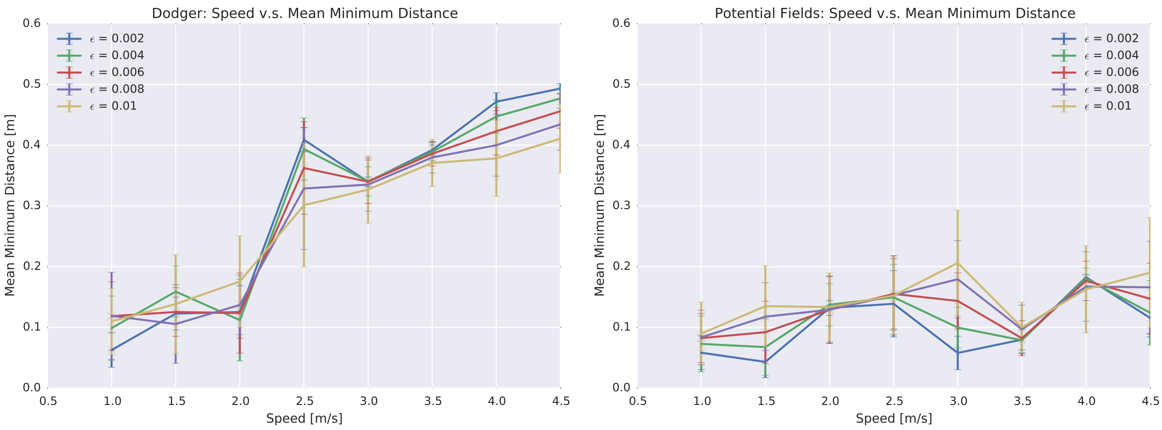

Figure 10.1: Plots showing how the average minimum distance to the obstacles changes as the speed increases for various amounts of obstacle position uncertainties

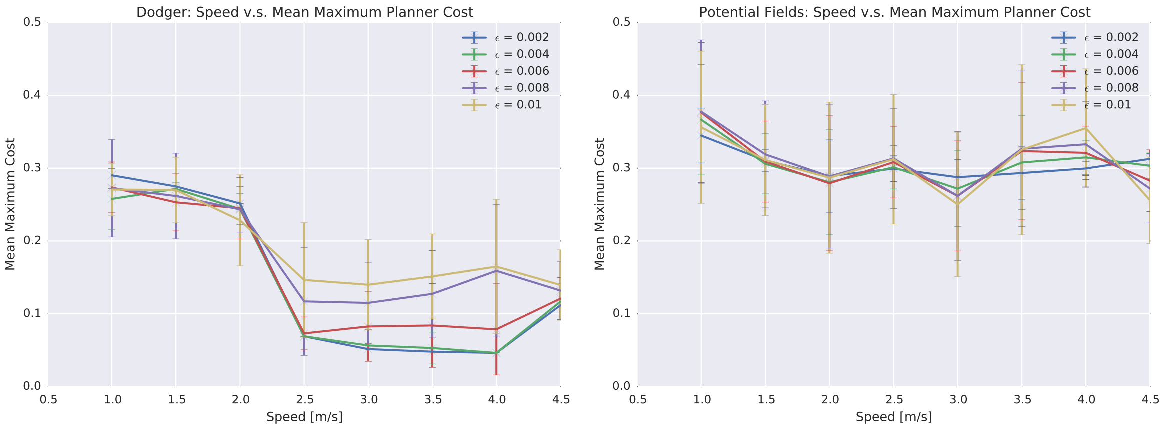

Figure 10.2:

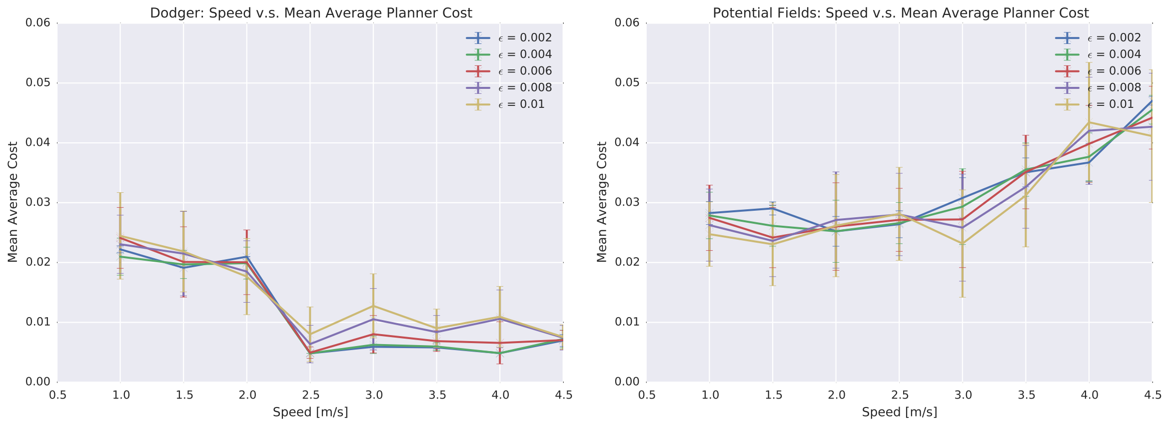

Figure 10.3:

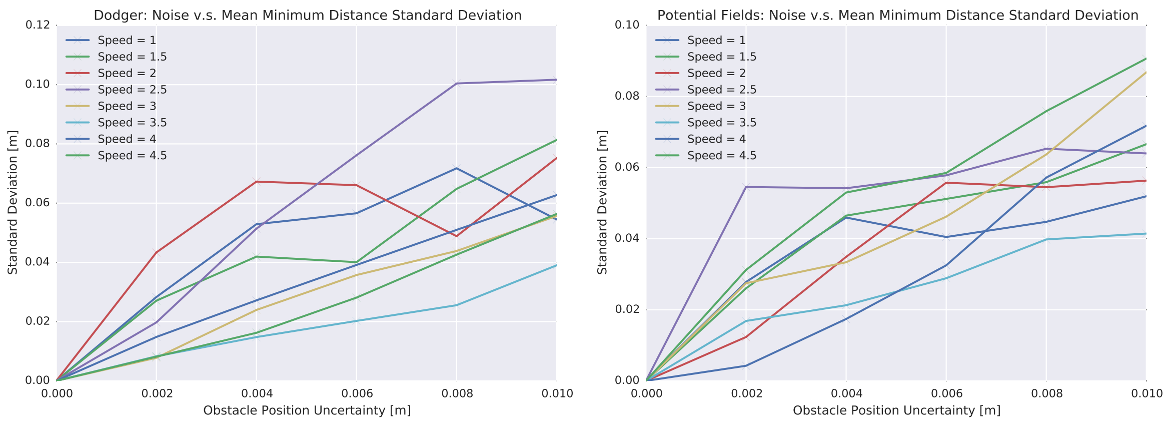

Figure 10.4:

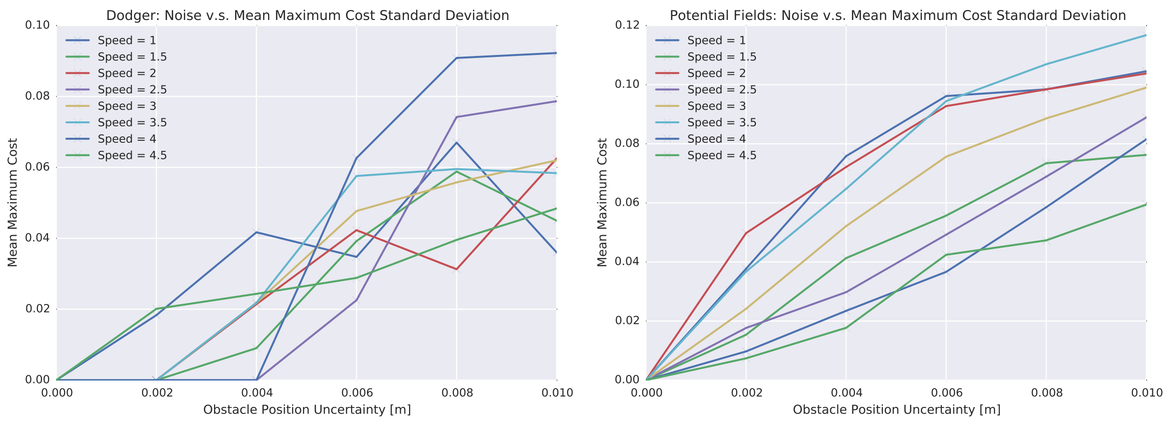

Figure 10.5:

10.1.1 Variance

10.2 Computational Time

10.2.1 Variance

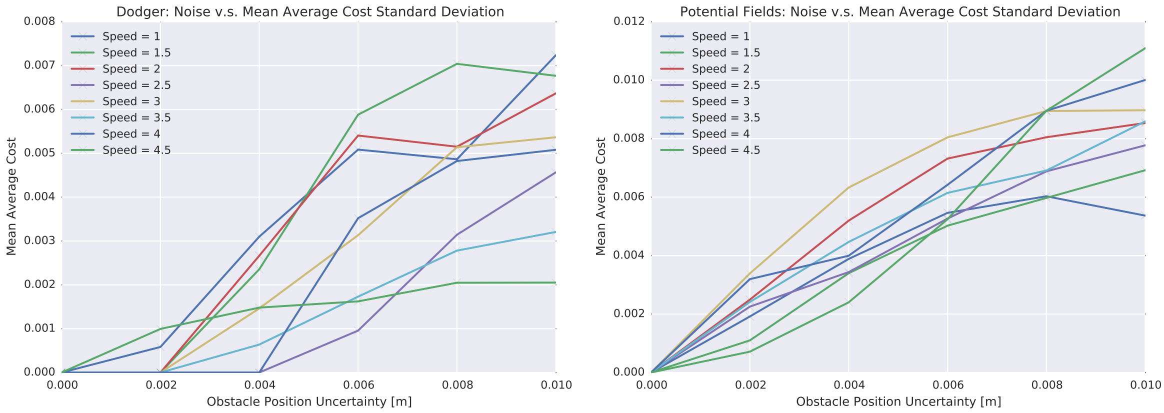

Figure 10.6:

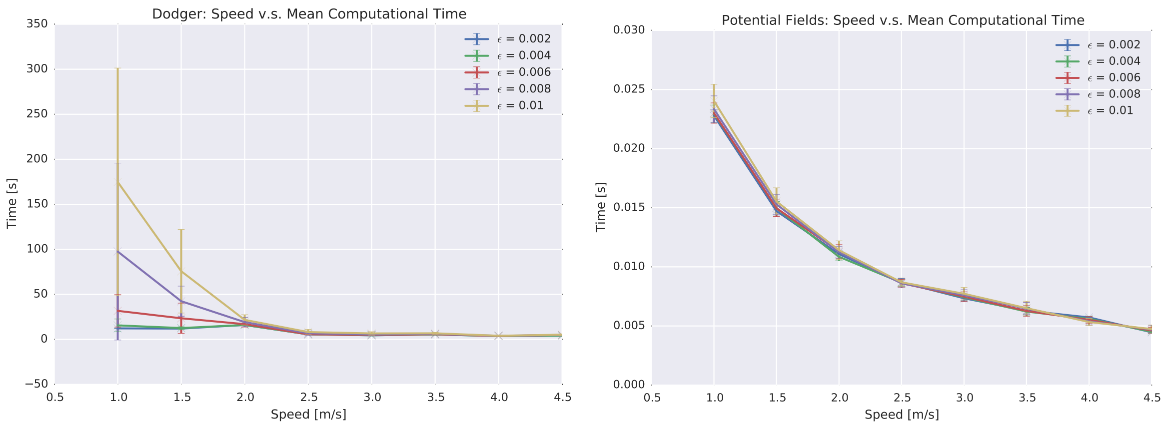

Figure 10.7: Plots showing how the computational time changes as the speed increases for various amounts of obstacle position uncertainties

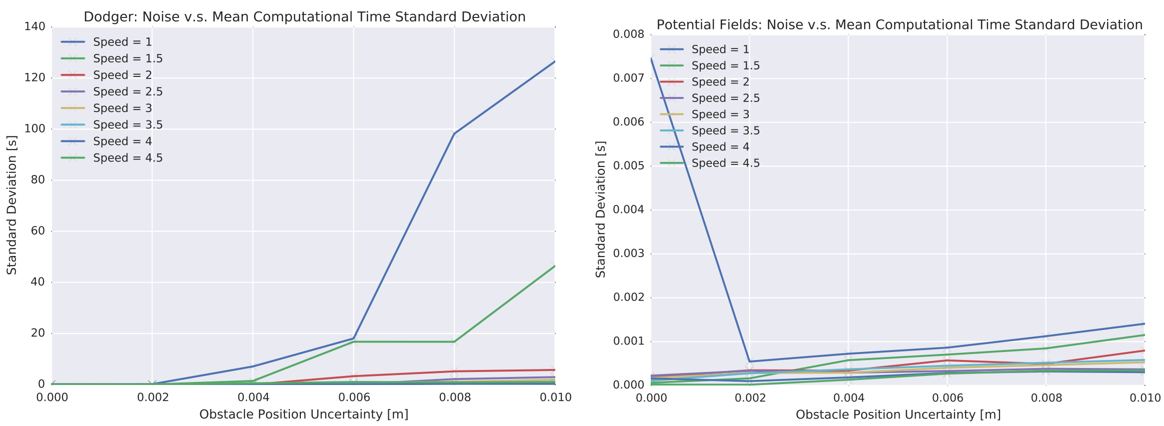

Figure 10.8:

Chapter 11

Discussion

Chapter 12

Acknowledgements

The author would like to thank Dr. Michael Weir and Dr. Erion Plaku for their valuable insight throughout the development of this project and the School of Computer Science for a truly phenomenal undergraduate education. The author would also like to thank his wife, Jessica, for putting up with the late nights and limited contact whilst this project slowly enveloped his life.

Bibliography

[1] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[2] J. Edmonds, "Optimum branching," *Journal OF Research of the National Bureau of Standards*, November 1967.