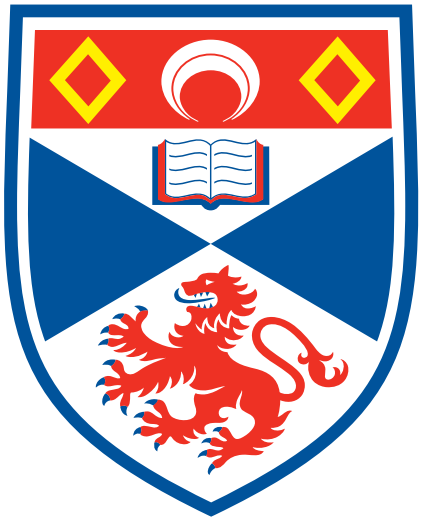
Generating Safe Trajectories in Stochastic Dynamic Environments



University of
St Andrews

CS4099: Major Software Project

Author:
Alexander Wallar

Supervisor:
Dr. Michael Weir

March 27, 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

Software Development Framework

Chapter 4

Objectives

4.1 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.

$$J(C, A) = \int_C \exp\Big(P(x, y, t_0, t_m, A) + 1\Big)\, \mathrm{d}s \tag{4.1}$$

Eq. 4.1 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. 4.1, the function P is the cost surface for a given time interval and set of obstacles. More description about P is given in Sec. 6.1.2 and is formally defined in Eq. 6.2. More precisely, the objective of this work to develop an algorithm that will provide an approximate solution to Eq. 4.2 which will return the minimum cost path through a environment for a given set of obstacles. The solution is described in Sec. 6.2

$$\Gamma(A) = \arg\min_C \; J(C, A) \tag{4.2}$$

4.2 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. 9.

Chapter 5

Design

As with any research project, many different attempts were made to come up with a solution to the objectives state in Ch. 4. 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.

5.1 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 [1, 2, 3]. 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. 5.1.

$$U_{rep}(p, t_0, t_m, A) = k \cdot P(p_x, p_y, t_0, t_m, A) \tag{5.1}$$

In Eq. 5.1, the function P is defined in Eq. 6.2, 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. 5.2

$$U_{att}(p, g) = \frac{c}{||p - g||^2 + \epsilon} \tag{5.2}$$

In Eq. 5.2, 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**
  $q' \leftarrow q + \delta t \cdot s \cdot \text{ROT}(\theta)$
  $p \leftarrow U_{rep}(q', O \cup A) + U_{att}(q', g)$
  if $p < p_{min}$ **then**
    $p_{min} \leftarrow p$
    $q_{min} \leftarrow q'$
  $\theta \leftarrow \theta + \delta\theta$
  for all $a \in A$ **do**
    STEP(a)
if $||q_{min} - g|| < R$ **then**
  return $\{p_{min}\}$
return $\{q_{min}\} \cup \text{PF}(q_{min}, g, O, A, R)$

potentials, there was only a nominal improvement which lead the author to move towards sampling based motion planning techniques which are outlined in Sec. 5.2 and Sec. 5.3.

5.2 Space-time Roadmap

The second attempt at devising a solution to the primary objective in Ch. 4 was to create a three dimensional probabilistic roadmap that can capture the connectivity of a two dimensional surface in space-time. A spatio-temporal probabilistic roadmap (PRM) is a directed, weighted graph, (V, E), that represents the spatio-temporal connectivity of the search space by randomly sampling points and connecting them such that if $((i, t), (j, t')) \in E$, then both i and j must not collide with an obstacle at times t and t' respectively, $||i - j|| \leq d$ where d indicates the maximum distance away connected nodes can be from one another, there must not be a collision with any obstacle along the edge from i and j in the time interval $[t, t']$, $|t - t'| < \delta t$, and $t < t'$ [4, 5]. 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, 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. This algorithm is shown more formally in Algo. 2.

With the generated roadmap, the first thought was to use a graph search algorithm such as A* [6] or Dijkstra's algorithm [7] 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. 6 and the formal equation for this line integral is given in Eq. 6.5. 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 [8] was used since other minimum spanning tree algorithms such as Kruskal's algorithm [9] and Prim's greedy algorithm [10] 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.

Algorithm 2 TemporalRoadmap$(N, d, \delta t, s, p, O, A)$

Input:
n: Maximum number of samples
d: Maximum distance between neighbouring nodes
O: Set of obstacles
Output:
An weighted directed graph of points describing where it is possible for the robot to move from its initial configuration.

$V \leftarrow \{(p, 0)\}$
for $i = 1$ **to** N **do**
  $(n, t) \leftarrow$ RandomSelection(V)
  $t' \leftarrow t +$ UniformRandom$(\varepsilon, \delta t)$
  $\theta \leftarrow$ UniformRandom$(0, 2\pi)$
  $q \leftarrow n + s \cdot t' \cdot$ Rot(θ)
  if $\bigwedge_{o \in O} \neg$Collision(o, q) **then**
    for all $(v, \tau) \in V$ **do**
      if $||v - q|| < d \wedge |\tau - t'| < \delta t$ **then**
        if $\tau < t'$ **then**
          $E \leftarrow E \cup \{((v, t), (v, t'), C(v, q, \tau, t', A))\}$
        else
          $E \leftarrow E \cup \{((q, t'), (v, t), C(q, v, t', \tau, A))\}$
    $V \leftarrow V \cup \{(q, t)\}$
return (V, E, W)

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.

5.3 Probabilistic Roadmap With Best First Search

After consideration for other sampling based motion planning techniques such as rapidly exploring random trees [11] and expansive space trees [12] the author chose to explore using a custom graph search algorithm over a two dimensional probabilistic roadmap due to the lack of ability for classical sampling based techniques to deal with time-dependent edge costs. The idea was to use a probabilistic roadmap to capture the connectivity of the two dimensional environment and to generate a search tree through the roadmap that encodes the temporal information for each point in a tree node and is therefore able to account for time-dependent edge costs. This graph search algorithm is a temporal analogue of best-first search which tries to expand the best current node in a search tree based on some heuristic [13]. The heuristic in this project is to expand the node in the search tree that has the minimum cost as defined in Eq. 6.5. A more complete and in depth description of this method is given in Ch. 6.

Chapter 6

Methodology

6.1 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.

6.1.1 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, $\dot{\zeta} : \mathbb{R}^+ \rightarrow \mathbb{R}^2$, 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. 6.4 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. 6.2. The variables ξ and T are initially set to I and 0 respectively. It is assumed that the robot has access to this information about the dynamic obstacles and will use it to safely around them. Information such as the initial configuration and the velocity equation for each dynamic obstacle are assumed to be determined by an external system that is either using a machine learning technique to deduce these properties by using information about where the dynamic obstacle has been before or by an external planning system such as in a warehouse that is commanding the velocities and configurations of these dynamic obstacles. This is the same assumption as that made by Phillips et al [14] and Narayanan et al [15].

6.1.2 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, $P_a : \mathbb{R}^2 \times (\mathbb{R}^+)^2 \rightarrow \mathbb{R}$, represents represents the cost surface for a given obstacle within a given time interval. Eq. 6.1 formally defines the cost function for a single obstacle.

$$P_a(x, y, t_0, t_m) = \int_{t_0}^{t_m} \mathcal{N}(\zeta_a(t), \alpha \cdot (t - t_0)^2 + \beta, x, y) \cdot (t_m - t)^\gamma \, \mathrm{d}t \tag{6.1}$$

In Eq. 6.1, $\mathcal{N}(\mu, \sigma^2, x, y)$ is the evaluation of a 3D normal distribution centered at (μ_x, μ_y) with a variance of σ^2 at (x, y). Fig. 6.1 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.

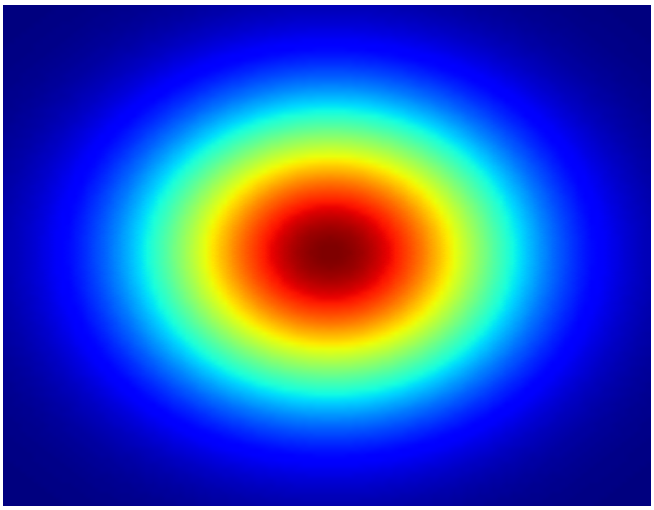


Figure 6.1:

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. 6.2.

$$P(x, y, t_0, t_m, A) = \frac{\sum_{a \in A} P_a(x, y, t_0, t_m)}{|A|} \tag{6.2}$$

An example of how P changes over time is shown in Fig. 6.2. 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. 6.2, 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.

6.1.3 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. 6.3.

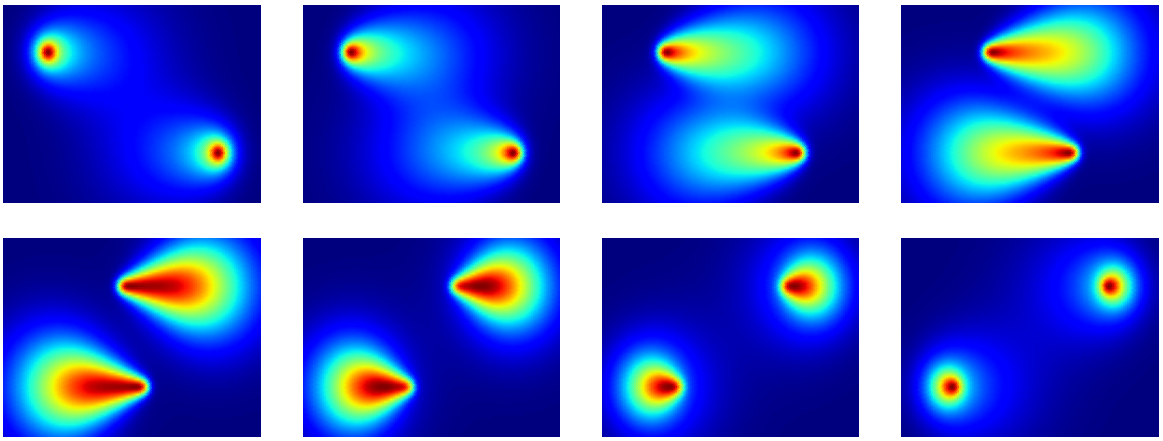


Figure 6.2: Cost distributions indicating the likelihood that an agent will be at a certain location within a given time interval. These figures show how this distribution changes over time (left to right, top to bottom)

$$\zeta_a(t) = \begin{cases} \xi_a + \int_{T_a}^{t} \dot{\zeta}_a(\lambda)\,\mathrm{d}\lambda & \text{if } t \geq T_a \\ \tilde{\zeta}_a(t) & \text{if } t < T_a \end{cases} \tag{6.3}$$

In Eq. 6.3, $\tilde{\zeta}_a$ 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 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. 6.2

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. 6.4. For this equation, it is assumed that the function only computes a value for any given time value, t, only once.

$$\tilde{\zeta}_a(t) = I_a + \int_{0}^{t} \dot{\zeta}_a(\lambda) + \rho\,\mathrm{d}\lambda \tag{6.4}$$

By increasing the level of stochasticity, ϵ, for a given dynamic obstacle, it will be more likely to diverge from its current path. This means that the uncertainty of a given dynamic obstacle can be parametrized by its value for ϵ.

The motion of a dynamic obstacle is described by it's velocity and a starting location in order for the planner to be able to continue to predict an obstacle's motion when it diverges from its current path. If the obstacle's motion was described by parametric equations of its position, it would not be possible to continue to predict where it is going to move once it is no longer following its prescribed path.

6.2 Planning Algorithm

The planning algorithm for Dodger has three major components. First, a two dimensional probabilistic roadmap is constructed over the search space in order to capture the spatial connectivity of the environment. The planner then uses a best-first (BestFS) graph search algorithm to determine the safest path from the initial position to the goal position by creating a temporal search tree through the graph in order to account for time-dependent edge costs. Once the robot has an initial path to follow, it will incrementally pursue this path whilst updating its information about the location of the dynamic obstacles. If any of the obstacles deviate from their predicted path, the planner will generate a new path (replan) for the robot using this information. These three components allow the planner to reduce the number of samples over time by reusing the same two dimensional roadmap and allows the planner to account for dynamic obstacles with stochastic motion by replanning. These are improvements to the first sampling based technique described in Sec. 5.2

6.2.1 Building the Roadmap

The first component of the planning algorithm is the underlying two dimensional roadmap which represents the spatial connectivity of the environment. This roadmap is constructed using a standard variant of the probabilistic roadmap algorithm created by Kavraki et al [4]. A probabilistic roadmap is a undirected graph, (V, E), created by randomly sampling n configurations in the configuration space of the robot and connecting them such that if $(i, j) \in E$, then $||i - j|| < d$, both i and j are not within any of the static obstacles, and there is no collision along the geometric edge from i to j. In this work the configuration space is $\mathbb{R}^2$ and the edge between two nodes is a straight line. The complexity of the graph is parametrized by the number of samples, n, and the maximum distance between nodes, d. By increasing the number of samples, the density of samples increases and therefore the average degree for a node in the graph increases. Likewise, if the maximum distance between nodes increases, so does the average degree for a node in the graph. Pseudocode is provided for constructing a probabilistic roadmap in Algo. 3.

Algorithm 3 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

```
1: for k = 1 to n do
2:     q ← RandomPoint2D(w, h)
3:     if ⋀_{o∈O} ¬Collision(o, q) then
4:         V ← V ∪ {q}
5: for all i ∈ V do
6:     for all j ∈ V do
7:         if i ≠ j ∧ ||i − j|| ≤ d ∧ ⋀_{o∈O} ¬Collision(o, i, j) then
8:             E ← E ∪ {(i, j)}
9: return (V, E)
```

In Algo. 3, first at most n points are sampled and added to the set of vertices, V, if and only if there is not a collision with any of the static obstacles. Essentially the Cartesian square of V is iterated over and any two points that are not the same, whose distance is less than d, and if there is no collision along the edge between these two points are added to the edge set, E. Finally, the vertices and edges are returned as a tuple representing the graph.

6.2.2 Searching the Graph

$$C(i, j, t_0, t_m, A) = \int_0^1 \exp\Big(P(x(\lambda), y(\lambda), t_0, t_m, A) + 1\Big) \cdot ||i - j||_2 \, \mathrm{d}\lambda \tag{6.5}$$

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 4 SearchGraph(V, E, R, A, p, g, T)

$Q \leftarrow$ PriorityQueue()
$D \leftarrow$ Dictionary()
$\mathcal{P} \leftarrow$ Dictionary()
Insert(Q, p, T)
while $\neg$Empty(Q) **do**
  $(q, t) \leftarrow$ Pop(Q)
  if $||q - g|| \leq R$ **then**
    return BacktrackPath$(p, g, \mathcal{P})$
  $S \leftarrow \emptyset$
  $N \leftarrow$ Neighbours(V, E, q)
  for all $n \in N$ **do**
    $t' \leftarrow ||q - n||/s + t$
    $\mathcal{P}_{(n,t')} \leftarrow (q, t)$
    $c \leftarrow \psi \cdot C(q, n, t, t', A) + \omega \cdot D_\eta$
    $D_n \leftarrow D_n + 1$
    $Q \leftarrow$ Insert$(Q, (n, t'), c)$

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

$q \leftarrow g$
$S \leftarrow$ Stack()
while $\mathcal{P}_q \neq p$ **do**
  $S \leftarrow$ Push(S, q)
  $(q, t) \leftarrow \mathcal{P}_{q,t})$
$S \leftarrow$ Push(S, p)
return S

6.2.3 Replanning

Algorithm 6 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:

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

Chapter 7

Implementation

7.1 Initial

Due to the ease of development and the author's experience with the language, Python was chosen as the initial language for the implementation. All three attempts described in Ch. 5, were implemented and a suite of tests scripts and visualization mechanisms were created in order to incrementally assess how each planner was behaving. However after the implementation was completed for the final design described in Sec. 5.3, it was discovered that Python could not produce solution paths (without replanning) through dynamic environments in a real-time scenario. Since Python could not search the graph within an acceptable amount of time, it could be only used for *a priori* planning and therefore could not be used in stochastic environments in which replanning would be needed. Due to this drawback, the author decided to rewrite the entire implementation in C++. Since this initial attempt is complete, it is publicly available at [16].

7.2 Final

The final implementation was written in C++ due to the tremendous speed improvement which made planning in real-time viable. Due to the object oriented structure of C++, the planner and the associated data structures were encapsulated in classes which made it very easy for people to use the code as an API. While designing the software, extra effort was put into making the code easily usable by other people who are not necessarily experts in robotics or motion planning. The system has an easy import mechanism and several examples on how to use the different planners that have been implemented.

In order to visualize the paths that the planner has generated, the C++ code can exports the paths of the robot and the dynamic obstacles to a JSON file and a Python script then can read and parse the generated JSON and display the paths using either Matplotlib or RViz. Matplotlib is an open source library for creating plots from data [17]. It is very similar to Matlab or Mathematica except it runs natively in Python. RViz is a core component of the Robotic Operating System (ROS) which can render dynamic scenes in three dimensions and can visualize native geometrical and navigation messages from ROS [18]. Since the paths are exported to JSON, a standard format with parsers in many languages, third party software can be developed that can parse the generated JSON file and analysis or visualized the generated paths. Likewise, the `Dodger`library can be imported by a third party program to control robots from the generated paths or to visualize the paths using other visualization tools such as OpenGL. A diagram of this tool-chain is shown in Fig. 7.1.

By separating the visualization code completely from the planner code, the software is more easily portable to different visualization frameworks and ensures that the user does not need to have a certain graphical software or ROS installed for the software to be able to compile and use the library. This allowed the planner code to be entirely self contained and simple to compile with a provided CMake file

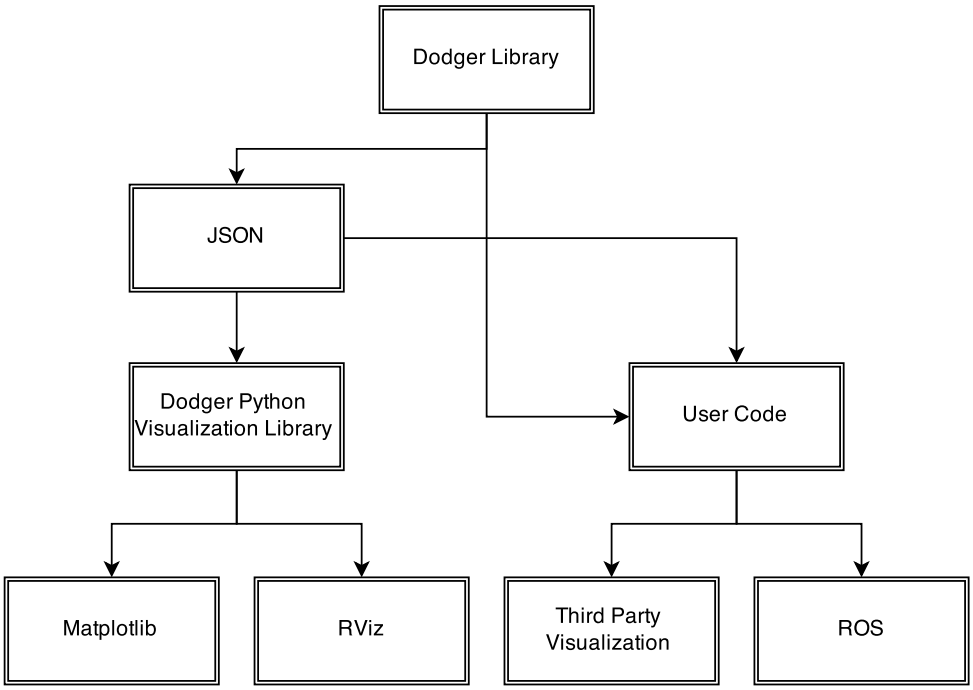


Figure 7.1:

which compiles the C++ code into a static library. Likewise, this design decision in the implementation allows the generated paths to be saved and either analysed or visualized at another time. Also, since the planner had no dependencies, experiments could be run on the servers provided without needing root access. This code is also made publicly available at [19].

Chapter 8

Experimental Setup

8.1 Metrics

$$MinDist(\Pi) = \min_{t \in \mathcal{T}} \min_{a \in A} ||\zeta_a(t) - \Pi(t)|| \tag{8.1}$$

$$MaxCost(\Pi) = \max_{t \in \mathcal{T}} P_A(\Pi(t)) \tag{8.2}$$

$$AvgCost(\Pi) = \int_{\mathcal{T}} P_A(\Pi(t))\,\mathrm{d}t \tag{8.3}$$

Chapter 9

Results

9.1 Safety

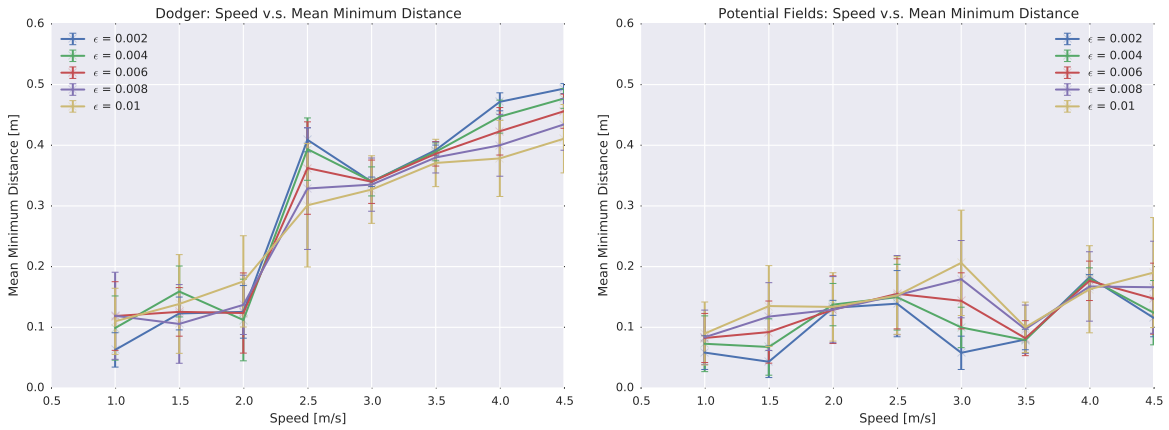


Figure 9.1: Plots showing how the average minimum distance to the obstacles changes as the speed increases for various amounts of obstacle position uncertainties

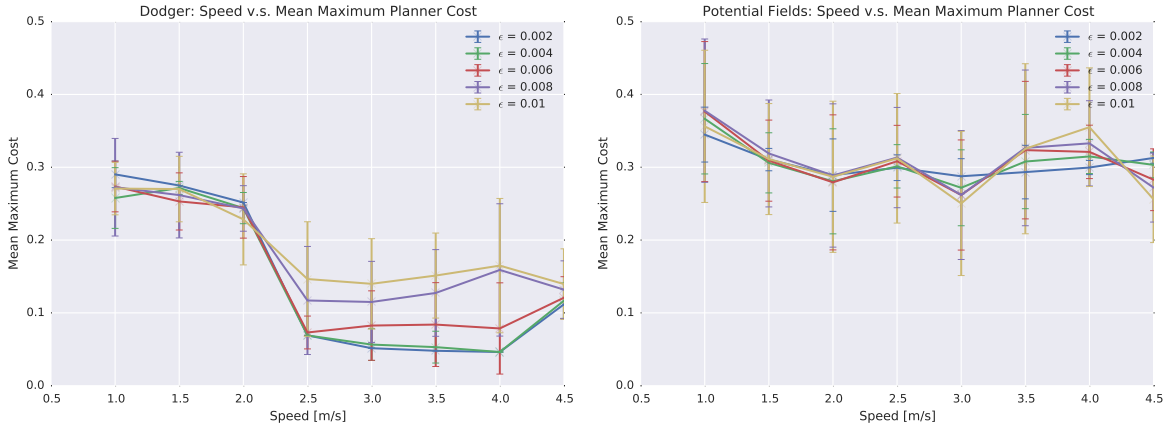


Figure 9.2:

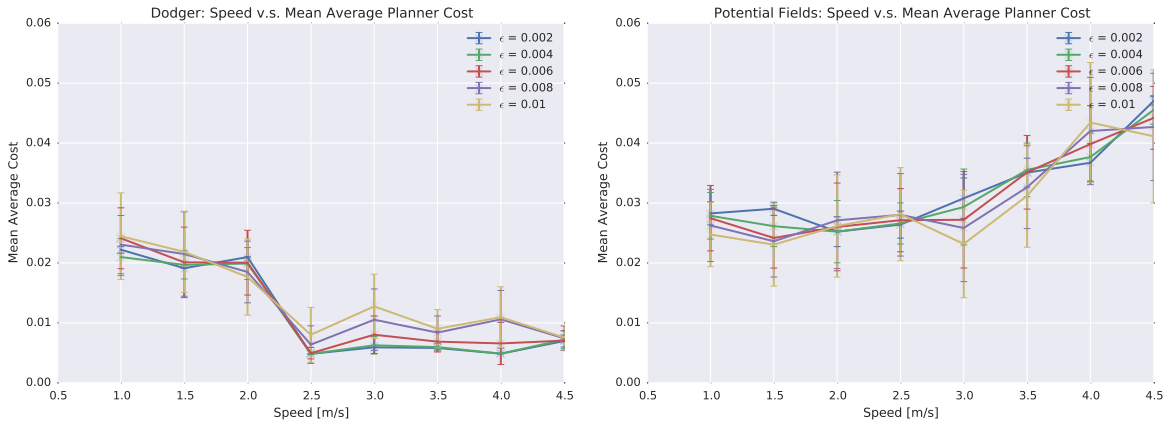


Figure 9.3:

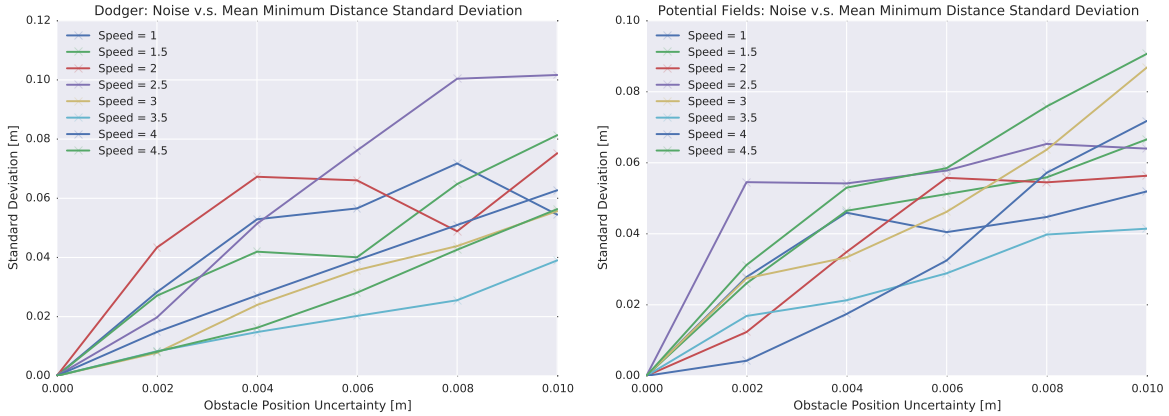


Figure 9.4:

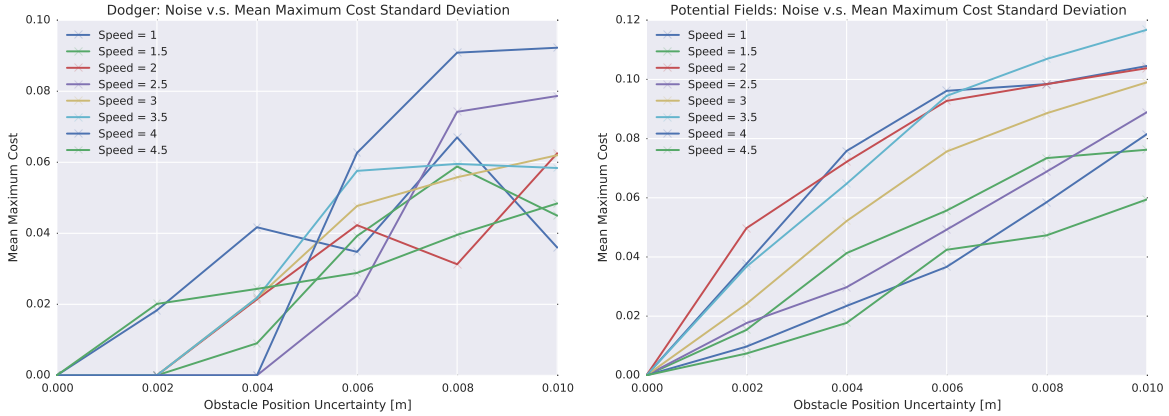


Figure 9.5:

9.1.1 Variance

9.2 Computational Time

9.2.1 Variance

9.3 Behaviour

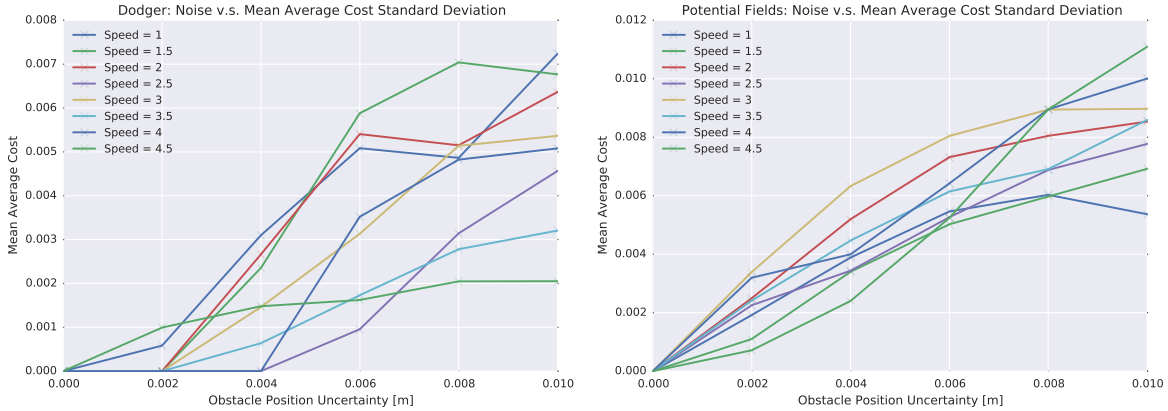


Figure 9.6:

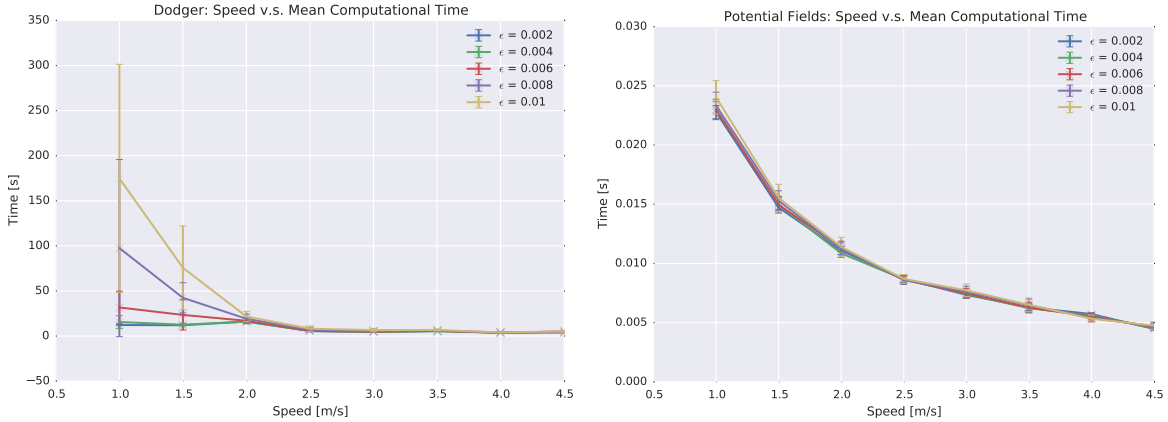


Figure 9.7: Plots showing how the computational time changes as the speed increases for various amounts of obstacle position uncertainties

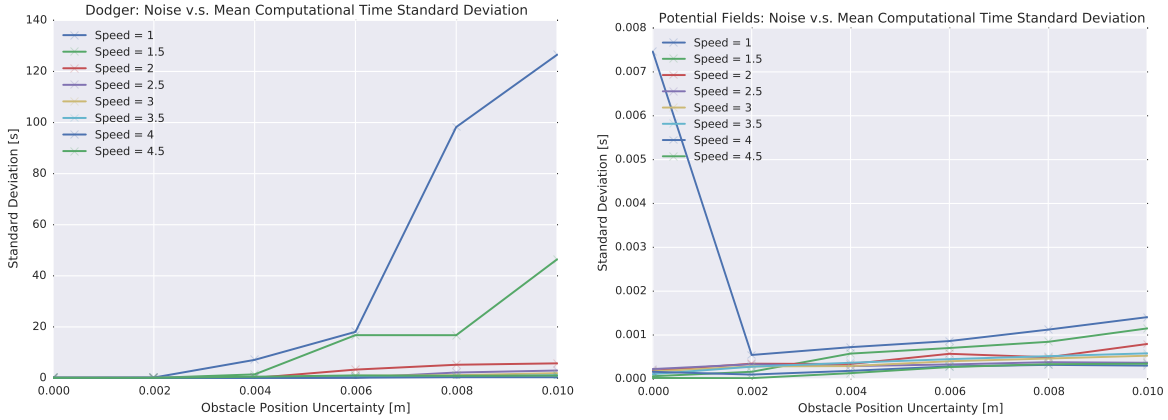


Figure 9.8:

Chapter 10

Discussion

Chapter 11

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. A possible vessel for this uprising is shown in Fig. 11.1 [20].

Figure 11.1: A picture of a possible vessel for the robot uprising

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] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 1398–1404, IEEE, 1991.

[2] A. Wallar and E. Plaku, "Path planning for swarms by combining probabilistic roadmaps and potential fields," in *Towards Autonomous Robotic Systems*, pp. 417–428, Springer, 2014.

[3] A. Wallar and E. Plaku, "Path planning for swarms in dynamic environments by combining probabilistic roadmaps and potential fields," in *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pp. 1–8, IEEE, 2014.

[4] 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, pp. 566–580, Aug 1996.

[5] G. Alankus, N. Atay, C. Lu, and O. Bayazit, "Spatiotemporal query strategies for navigation in dynamic sensor network environments," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 3718–3725, Aug 2005.

[6] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, pp. 100–107, July 1968.

[7] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

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

[9] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, p. 4850, 1956.

[10] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, p. 13891401, 1957.

[11] S. M. LaValle, "Rapidly-exploring random trees a new tool for path planning," 1998.

[12] J. M. Phillips, N. Bedrossian, and E. Kavraki, "Guided expansive spaces trees: A search strategy for motion-and cost-constrained state spaces," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 4, pp. 3968–3973, IEEE, 2004.

[13] R. E. Korf, "Linear-space best-first search," *Artificial Intelligence*, vol. 62, no. 1, pp. 41–78, 1993.

[14] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5628–5635, IEEE, 2011.

[15] V. Narayanan, M. Phillips, and M. Likhachev, "Anytime safe interval path planning for dynamic environments," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4708–4715, IEEE, 2012.

[16] A. Wallar, "Racer: Path planning in stochastic dynamic environments in python." `https://github.com/wallarelvo/racer`, 2015.

[17] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[18] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, 2009.

[19] A. Wallar, "Dodger: Path planning in stochastic dynamic environments in c++." `https://github.com/wallarelvo/Dodger`, 2015.

[20] R. McKee, "Skynet: 5 Things You Didn't Know." `http://uk.askmen.com/entertainment/special_feature_300/339_skynet-5-things-you-didnt-know.html`.