Generating Safe Paths in Dynamic Environments By Extracting Minimum Cost Trajectories Using Obstacle Position Probability Distributions and Replanning

University of
St Andrews

CS4099: Major Software Project

Author:
Alexander Wallar

Supervisor:
Dr. Michael Weir

March 21, 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

1 Design **3**

1.1 Agents 3

1.2 Planning Algorithm 4

2 Experimental Setup **6**

3 Discussion **7**

Chapter 1

Design

1.1 Agents

$$P_a(x, y, t_0, t_m) = \int_{t_0}^{t_m} \mathcal{N}(\zeta_a(t), \alpha \cdot (t - t_0)^2 + \beta, x, y) \cdot (t_m - t)^\gamma \mathrm{d}t \tag{1.1}$$

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

$$P_A(x, y, t_0, t_m) = \frac{\sum_{a \in A} P_a(x, y, t_0, t_m)}{|A|} \tag{1.2}$$

$$\tilde{\zeta_a}(t) = \begin{cases} \tilde{\zeta_a}(t - \delta t) + \zeta_a'(t) \cdot \delta t + \rho & \text{if } t > 0 \\ \zeta_a^{(0)} & \text{if } t = 0 \end{cases} \tag{1.3}$$

Where $\rho \sim \mathcal{U}(-\epsilon, \epsilon)$, $\epsilon > 0$, and $\zeta_a^{(0)}$ is the initial position of the obstacle.

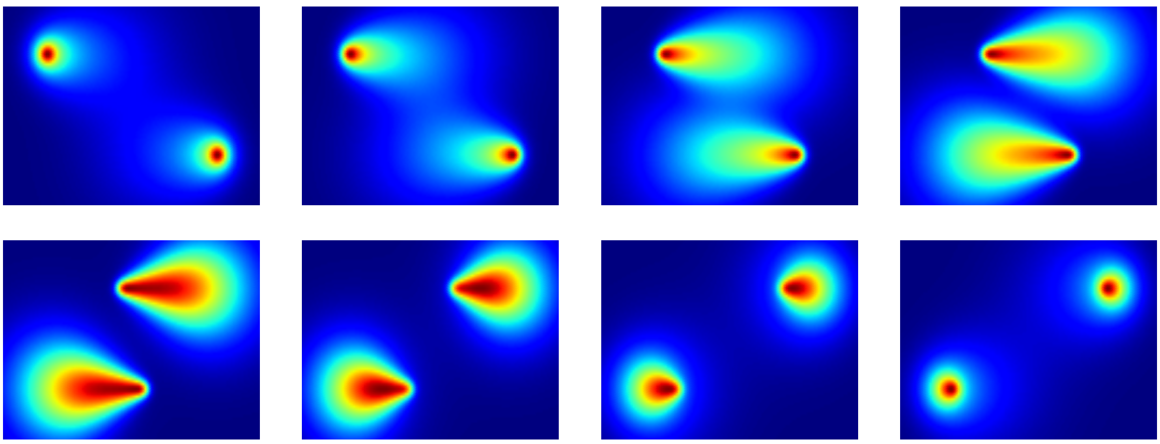

Figure 1.1: Cost distributions indicating the likelihood that an agent will be at a certain location within a given time interval. These figures show how this distribution changes over time (left to right, top to bottom)

1.2 Planning Algorithm

$$C_A(i,j) = \int_0^1 \exp\Big(P_A(x(\lambda), y(\lambda), i_t, j_t)\Big) \cdot ||i-j||_2 \mathrm{d}\lambda \tag{1.4}$$

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 1 $\textsc{Roadmap}(n, d, w, h, O)$

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 2 $\textsc{GetPath}(n, d, w, h, \delta, p, g, O, A, R)$

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:

$(V, E) \leftarrow \textsc{Roadmap}(n, d, w, h, O)$
$\Pi \leftarrow \textsc{Set}()$
$q \leftarrow p$
while $||\textsc{Back}(\Pi) - g||_2 > R$ **do**
  $\pi \leftarrow \textsc{SearchGraph}(V, E, R, A, q, g)$
  for all $i \in \pi$ **do**
    $\Pi \leftarrow \Pi \cup \{i\}$
    for all $a \in A$ **do**
      $\textsc{Step}(a)$
    if $\bigvee_{a \in A} ||\tilde{\zeta_a}(i_t) - \zeta_a(i_t)|| > \delta$ **then**
      for all $a \in A$ **do**
        $\textsc{Update}(\zeta_a, \tilde{\zeta_a})$
      $q \leftarrow i$
      break
return Π

Algorithm 3 SearchGraph(V, E, R, A, p, g)

$Q \leftarrow$ PriorityQueue()
$D \leftarrow$ Dictionary()
$\Pi \leftarrow$ Dictionary()
Insert$(Q, p, 0)$
while $\neg$Empty(Q) **do**
  $q, w \leftarrow$ Pop(Q)
  if $||q - g||_2 \leq R$ **then**
    return BacktrackPath(p, g, Π)
  $N \leftarrow$ GetTemporalNeighbours(V, E, q)
  for all $n \in N$ **do**
    $\Pi_n \leftarrow q$
    $c \leftarrow \psi \cdot C_A(q, n) + \omega \cdot D_n$
    $D_n \leftarrow D_n + 1$
    if $w > c$ **then**
      $c \leftarrow w$
    $Q \leftarrow$ Insert(Q, n, q)

Algorithm 4 GetTemporalNeighbours(V, E, q)

$S \leftarrow Set()$
$N \leftarrow$ Neighbours(V, E, q)
for all $n \in N$ **do**
  $t \leftarrow ||q - n||_2/s + q_t$
  $S \leftarrow S \cup \{(n_x, n_y, t)\}$

Algorithm 5 BacktrackPath(p, g, Π)

$q \leftarrow g$
$S \leftarrow$ Stack()
while $\Pi_q \neq p$ **do**
  $S \leftarrow$ Push(S, q)
  $q \leftarrow \Pi_q$
$S \leftarrow$ Push(S, p)
return S

Chapter 2

Experimental Setup

Algorithm 6 $\text{PF}(q, g, O, A, R)$

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

Chapter 3

Discussion