Stealth Navigation: Covert Path Planning Using Spotlights

Salma Berriche, Ralph Skliros, Anotida Ntini, Kavyalakshmi Erode Vivekanandhan, Ali Badawy

***Abstract*—This project implements a "stealth robot" that uses Q-learning to navigate from a given location to another on a known map while avoiding detection by "searchlights". We outline the algorithms and framework used in our covert path planning implementation as well as our experiments and their results.**

***Index Terms*—Covert path planning, Reinforcement learning, Q-Learning, Bresenham's line algorithm**

I. Introduction

Stealth robots have a wide range of applications including high-risk situations such as covertly delivering supplies to a specified destination whilst minimising the risk of capture. Other uses could be a guard robot on patrol remaining unobserved by intruders, a service robot avoiding collisions with people and physical obstacles in its environment, and even robots that play games such as hide and seek. The principles behind covert path planning could also be applied in video games, typically in the case of non-playable characters. The authors in [1] outline further potential uses for stealth robots.

This report discusses our a custom stealth reinforcement learning environment designed to be used with a Q-learning agent.

II. Related Work

A. Obstacle Avoidance

The problem of avoiding searchlights can be thought of as path planning and obstacle avoidance. For example, [2] proposes a path planning algorithm based on the Markov Decision Process (MDP) which can be used in dynamic environments to guide a robot to a goal. The algorithm can find a correct path in real time even in the presence of moving obstacles by constantly recalculating a candidate new path. This idea is useful for the project as the can searchlights can be seen as moving obstacles.

[3] describes a Q-learning approach to discover the best strategies for obstacle avoidance. The robot can learn without a model of the environment since Q-learning is off-policy. This would be useful to implement since the robot does not know the positions and behaviour of the searchlights, with the understanding that convergence to the optimal policy can be slow.

B. Covert Path Planning

[4] introduces the problem of covert path planning i.e. a robot navigating in an environment to approach or track a specific target without getting spotted. A Dark Path algorithm is proposed, which involves a two-dimensional array of the map where each cell's cost is computed based on the costs of its neighbours, distance from the goal, and visibility of the robot. The limitation of this approach is that it requires complete knowledge of the environment and goal location.

[5] uses a Q-learning based path planning approach to train autonomous underwater vehicles (AUVs) to avoid detection. Probability of detection is calculated using the signal-to-noise ratio from the sonar sensors. However, the algorithm is restricted to environments where obstacles and sensors are static.

III. Algorithms and Framework

In our stealth robot project, the key challenge is to enable a robotic agent to navigate through complex and dynamic environments while remaining undetected. This involves not only understanding and reacting to the environment in real time but also planning and executing covert paths that minimize the risk of detection. To achieve this, our solution integrates a sophisticated Q-Learning algorithm for decision-making and path planning with advanced image processing techniques. These components work in tandem within a robust Robot Operating System (ROS) framework, ensuring efficient and adaptable operations.

A. Q-learning for path planning

Q-Learning, an off-policy temporal difference algorithm, lies at the heart of the robot's decision-making process.

The Q-Learning process involves maintaining a Q-value for each state-action pair, which represents the expected utility of taking a given action in a given state and then following the optimal policy thereafter. The Q-values are initialized to a default value and updated as the robot interacts with its environment. Algorithm 1 shows pseudocode for Q-learning.

The key steps in Q-learning are:

1) **Initialization:** The algorithm initializes parameters such the number of actions, learning rate (α), discount factor (γ), and exploration rate (ϵ).
2) **Learning:** Each time the agent takes an action and receives feedback (rewards) from the environment (in the form of a reward), the Q-values $Q(s, a)$ are updated, reflecting the learned value of taking action a in state s.
3) **Action Selection:** The agent uses an ϵ-greedy policy where it most often chooses the action with the highest Q-value in the current state but occasionally chooses a random action for exploration. The Q-learning algorithm

Algorithm 1 Q-learning algorithm for estimating $\pi \sim \pi_*$ [6]

```
1: Parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
2: Initialise $Q(s, a)$ for all $s \in \mathcal{S}^+$, arbitrarily except that $Q(terminal, \cdot) = 0$
3: Loop for each episode:
4:     Initialise $S$
5:     Choose $A$ from $S$ using policy derived from $Q$ (e.g. $\epsilon$-greedy)
6:     Take action $A$, observe $R$, $S'$
7:     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
8:     $S \leftarrow S'$
9: until $S$ is terminal
```

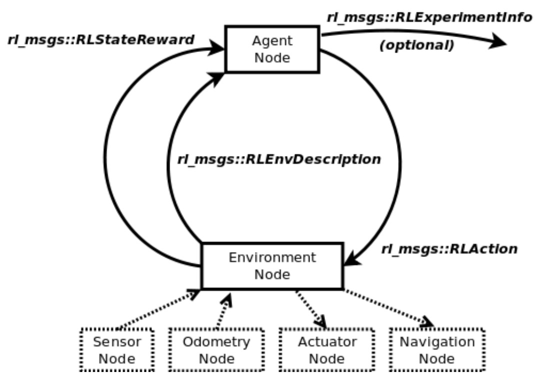

Fig. 1. Communication between ROS reinforcement learning agents and environments [8]

will converge to an optimal policy $\pi \sim \pi_*$ as long as all state-action pairs continue to be updated.

We opted to use the Q-learning agent from `rl-texplore-ros-pkg` in order to focus our efforts on building a reinforcement learning environment involving searchlights with custom ranges, with the robot's objective to reach the goal undetected. This `rl_agent` node publishes its chosen actions according to its ϵ-greedy policy and keeps track of a Q-matrix which it learns from exploring the environment and receiving rewards.

B. Image Processing for Environmental Understanding

The robot's ability to understand and interact with its environment is enhanced through image processing techniques. Our project includes scripts for converting images to black and white and transforming PNG images to ROS (Robot Operating System) maps. These processes are crucial for simplifying the robot's perception of its surroundings and enabling efficient path planning.

Our `image_to_bw.py` script uses OpenCV [7] to process image data and convert images to black and white, making it suitable for conversion to a ROS map. `png_to_ros_map.py` transforms PNG images into ROS-compatible maps. `floorplan` 5 and `map1` 4 were created using this method.

C. Reinforcement Learning Environment

In the context of reinforcement learning within the ROS framework, Fig. 1 illustrates the feedback loop between a reinforcement learning agent and its environment. `rl_msgs`[8] defines several message types used in our environment implementation:

- An `RlEnvDescription` messages is passed from environment to agent on startup; it contains key information such as the range of rewards and whether the task is deterministic or stochastic and episodic or not. In the case of covert path planning, the policy is stochastic (mapping each state to a probability distribution over actions) and the task is episodic; terminal states are reaching a goal point and being detected by a searchlight.
- `RLStateReward` provides the agent with its new state, the reward received, and whether its transition was terminal or not. Our implementation publishes an `RLStateReward` message every time an `RLAction` message is passed to the environment.
- `RLAction` messages are passed from the agent to the environment to inform it of its chosen action. We defined actions as the robot moving a given distance up, down, left, or right, and states as the truncation of the robot's two-dimensional coordinates obtained from AMCL.

Our environment node sets up ROS publishers for messages to pass to the agent, and subscribers for messages to receive from it. The `amcl` [9] package is used for AMCL (Adaptive Monte Carlo Localisation) based on the robot's odometry data, not only for visualisation in Rviz but also to aid calculation of distance-based reward factors.

D. Physically-Based Searchlight Modeling

A large factor in calculating the state reward of the robot within the environment is driven by searchlights placed around the scene. The illumination effect of the searchlights approximates the physical behavior of conical light sources. Each light is parameterized by a position vector, a direction angle, a "field of view" angle, and a detection range: R. We model the attenuation of light intensity I from the distance d to the source using the inverse square law given by:

$$I = 1/d^2 \tag{1}$$

The first challenge was to adapt equation 1 to meet the imposed restraints of the parameters. We first limit the upper bound of the equation to 1, which will be used to calculate the maximum penalty for detection. This is done by adding 1 to the denominator, preventing the equation from exploding as $d \rightarrow 0$.

$$I = 1/(1 + \beta d^2) \tag{2}$$

Here, the coefficient β is calculated such that, at the distance $d = R$, $I = \delta$, where δ is a sufficiently small threshold close to 0. The reason for this is that the original equation, while upper bounded at $I = 1$, is asymptotic as $d \rightarrow \infty$. To satisfy these constraints, equation 3 is used to obtain a suitable value for β, and therefore a calibrated attenuation model.

$$\beta = (1 - \delta)/\delta R^2 \tag{3}$$

Using this method, we are able to balance the physical properties of light intensity, while also having finer control over the simulation.

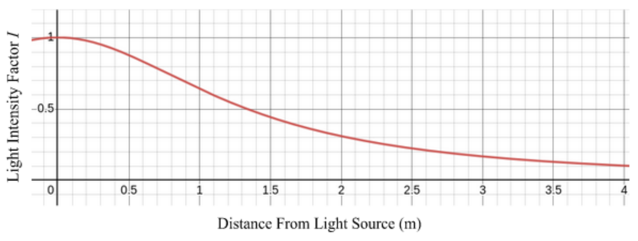

Fig. 2. A graph showing the light intensity factor by distance from the light source where $R = 4m$ and $\delta = 0.1$

Lastly, the factor we use to calculate the state reward of the robot is the total visibility scores from all of the individual light sources. This is based on the physical intuition that light intensity is additive; crossing paths of light results in a brighter intersection. This total visibility is a key component of the reward function provided by our environment to the Q-learning agent.

E. Environmental Occlusion

We employ a variation on Bresenham's line algorithm [10] to iterate over pixels in integer map coordinates along the line formed between two arbitrary points. The line segment is then continuously checked against the occupancy grid to see if the path is blocked.

Algorithm 2 Adapted Bresenham's line algorithm [10] for efficient environmental occlusion detection

```
    Parameters: Searchlight coordinates $(x_1, y_1)$, Robot coordinates $(x_2, y_2)$
 2: $\Delta x \leftarrow (x_2 - x_1)$
    $\Delta y \leftarrow (y_2 - y_1)$
 4: $\epsilon \leftarrow 0, y \leftarrow y_1$
    for $x \leftarrow x_1$ to $x_2$ do
 6:     if Grid Cell $(x, y)$ is occupied then
            return 0
 8:     else if $(2(\epsilon + \Delta y) < \Delta x)$ then
            $\epsilon \leftarrow \epsilon + \Delta y$
10:     else
            $y \leftarrow y + 1$
12:         $\epsilon \leftarrow \epsilon + \Delta y - \Delta x$
        end if
14: end for
    return Intensity at distance $d = \sqrt{\Delta x^2 + \Delta y^2}$
```

The time complexity of Algorithm 2 is $O(|\Delta x|)$, where Δx is the integer difference between the searchlight and robot's respective x coordinates. Additional speed is also realised by the avoidance of (repeated) expensive operations such as floating-point arithmetic, division ,and multiplication; Only integer operations are required.

F. Reward Function

The reward function is based on two main components: the total visibility of the robot to the searchlights (as calculated in Section III-D) and the Euclidean distance of the robot from the goal.

There are three different weighted sums of these components used for rewards: 1) a high reward for reaching the goal, which gives a higher reward the closer the robot is within the acceptance radius, 2) a high penalty for being detected by one or more searchlights which takes distance from the goal into account, and 3) a weighted sum of the robot's visibility and goal distance used for non-terminal states.

IV. Experimental Results

We define four reinforcement learning maps, each with a set start state, goal point, and searchlights with varying ranges and intensities. Figs. 3-6 show reinforcement learning experiments in action in RViz. In these diagrams, the dark pink sphere is the goal position and the red arrow and pink circle are the robot's estimated pose and covariance from AMCL. The green spheres are the searchlights with the red lines indicating their visibility ranges. The simulated robot's 'true' movement can also be visualised in the ROS Stage simultaneously when reinforcement learning experiments are run using the provided launch file.

Our environment node allows for reinforcement learning episodes to be run successively, allowing the agent to learn the optimal policy over time. This policy includes both navigating to the goal and avoidance of searchlights with varying ranges due to the nature of the reward function. `rospy.loginfo()` is used to output useful information during execution, such as visibility, distance from goal, and reward for the most recent action. This also allows us to observe the reward increase over these episodes indicating convergence towards the optimal policy.

However, this convergence can be slow, with initial episodes taking especially long to reach a terminal state. Adding components based on time and/or distance travelled to the reward function would be beneficial; the policy would be learned in such a way that it prioritises shorter and/or faster paths to the goal.

V. Conclusions

The environmental setup for calculating state rewards appears to model our proposed situation reasonably well, using a combination of physically based lighting calculations and highly customisable parameters to simulate spotlight-illuminated internal environments. Additionally, Q-learning is an effective method to solve the problem of covert path planning, with experiments showing that the robot slowly but surely learned the optimal policy for given maps over time.

VI. Next Steps

A. Environment

One possible next step is to tackle environments with dynamic ranges or moving searchlights. This would require

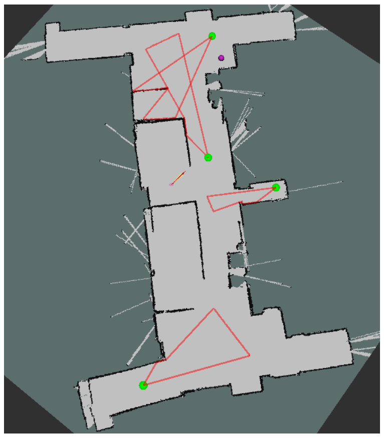

Fig. 3. `lgfloor` map (from `socspioneer` package) with four searchlights, including two with overlapping ranges.

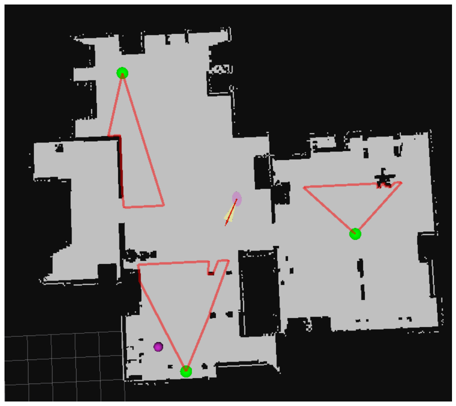

Fig. 4. `map1` with three searchlights, including one close to the goal, making it challenging to navigate there without being detected.

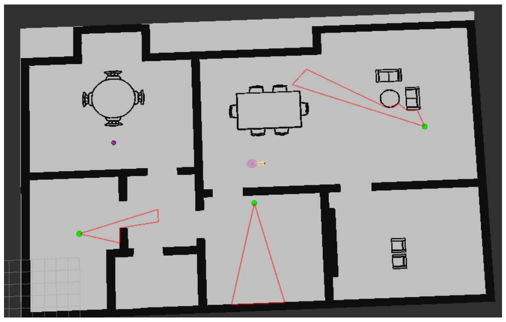

Fig. 5. `floorplan` map featuring searchlights in different rooms on the way to the goal.

a more complex reward function that takes the movement patterns of searchlights into account when calculating the robot's visibility to searchlights.

B. Visibility Model

Further improvements could be made to the lighting model used in the environment. Firstly, taking into account the bounce lighting within the scene which may increase the robot's visibility. Secondly, refining the robot's visibility calculations by how much surface area is exposed to the light,

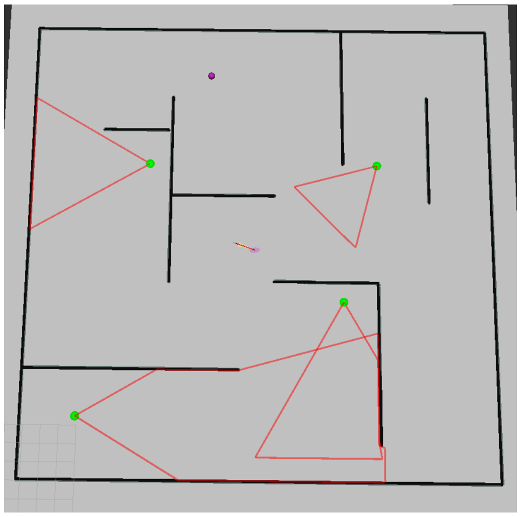

Fig. 6. `simple-maze` map including searchlights with very wide ranges.

in the hopes that the agent might learn to minimize exposure to a directional light source through adjusting its orientation. Furthermore, adding an intensity drop off near the edges of

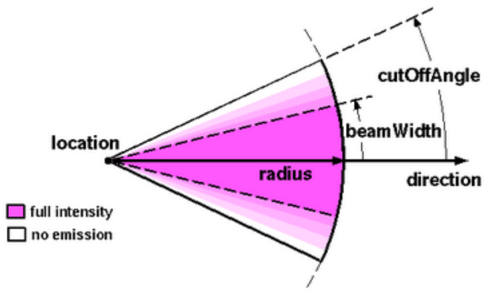

Fig. 7. A diagram showing the proposed light attenuation at the cone edges

the defined light cone for the searchlights as illustrated in Fig. 7 would further increase the physical accuracy of the model. All of the suggestions add realism into the environment, which may encourage different learning behaviours from the agent, making it more applicable to real world environments.

Our code is available at `https://git.cs.bham.ac.uk/smb114/ir-final-project`.

References

[1] Mohamed Al Marzouqi and Ray A. Jarvis. "Robotic Covert Path Planning: A Survey". In: *IEEE 5th International Conference on Robotics, Automation and Mechatronics* (2011).

[2] Yu-Ju Chen, Bing-Gang Jhong, and Mei-Yung Chen. "A Real-Time Path Planning Algorithm Based on the Markov Decision Process in a Dynamic Environment for Wheeled Mobile Robots". In: *Actuators* (2022).

[3] Khawla Almazourei, Ibrahim Kamel, and Tamer Rabie. "Dynamic Obstacle Avoidance and Path Planning through Reinforcement Learning". In: *Applied Sciences* (2023).

[4] Mohamed Marzouqi and Ray A. Jarvis. "Covert Path Planning for Autonomous Robot Navigation in Known Environments". In: *Australasian Conference on Robotics and Automation* (2004).

[5] Emre Tascioglu and Ahmet Gunes. "Path-planning with minimum probability of detection for AUVs using reinforcement learning". In: *Innovations in Intelligent Systems and Applications Conference* (2022).

[6] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2018.

[7] *`opencv-python` Documentation*. Accessed November 30th, 2023. URL: https://pypi.org/project/opencv-python.

[8] *`rl_msgs` Documentation*. Accessed November 28th, 2023. URL: http://wiki.ros.org/rl_msgs.

[9] *`amcl` Documentation*. Accessed November 30th, 2023. URL: http://wiki.ros.org/amcl.

[10] P Koopman. "Bresenham line-drawing algorithm". In: *Forth Dimensions* (1987).