Exploring multi-armed bandit decision-making strategies in an underwater vehicle testbed

Jonathan Valverde Lizano, '16

Submitted to the
Department of Mechanical and Aerospace Engineering
Princeton University
in partial fulfillment of the requirements of
Undergraduate Independent Work.

Final Report

April 28, 2016

Naomi E. Leonard
Egemen Kolemen
MAE 442
107 pages
Adviser Copy
The electronic
version of this
document contains
embedded videos

This thesis represents my own work in accordance with University regulations.

Abstract

The problem of field estimation, or finding the spatial distribution of a resource in space, has many applications in problems of robotic search. This thesis approaches the problem in the framework of a Gaussian Multi-Armed Bandit (MAB) task, a problem in which an agent must learn about an unknown environment while maximizing expected reward. The arms correspond to discretized points in space, and the smoothness of the field is modeled as spatial correlation between the arms. Upper Confidence Limit (UCL), an algorithm developed by Reverdy et al. in 2014 for Gaussian MAB problems with correlated arms and prior knowledge, is then applied to this problem.

The smoothness of the field is measured by a parameter known as the length scale. In real world applications, the agent can only have an estimate of this length scale. This thesis explores the performance of UCL with correlation in comparison to other algorithms when the estimate of the length scale is correct. The effect of overestimates and underestimates in the length scale is then explored for fields of different smoothness. The search task is finally implemented in a testbed with a robot, giving additional metrics of performance.

The simulations showed that knowledge of the spatial correlation of the arms can result in improvements in performance using UCL when compared to other algorithms that do not account for correlation. In addition, it is shown that, in the cases studied, best performance is not actually obtained for the correct length scale estimate, but for some particular overestimate. This effect must be studied further, but the results suggest that an agent performing this search should use an overestimate and not an underestimate of the length scale that describes the field, provided this overestimate is not grossly inaccurate.

Contents

List of Tables

List of Figures

List of Symbols

N	The number of arms in an MAB	4
T	The time horizon, or the total number of time steps in an MAB	4
r_t	The reward obtained at time t by an agent in an MAB	4
m_i	The mean of arm i in an MAB	4
m^*	The maximum mean reward of all arms in an MAB	4
R_t	The expected regret at time t	4
n_i^T	The number of times that arm i has been chosen up to time T	4
Δ_i	The expected regret due to picking arm i instead of the optimal arm	4
σ_s^2	The variance of the reward samples of each of the arms in a Gaussian MAB	5
Q_i^t	The heuristic value associated with arm i at time t used to choose an arm	5
$\bar{m}_i^t$	The empirical mean of arm i at time t	5
C_i^t	A measure of the uncertainty of the reward of arm i at time t	5
α_t	A function of time chosen in Bayes-UCB to obtain upper confidence bounds for the arms that hold with increasingly high probability in terms of time	6
μ_i^0	The prior on the mean of arm i given to the agent in UCL	6
σ_0^2	The variance on the prior given to the agent in UCL	6
μ_i^t	The belief state of the mean of arm i at time t in UCL	7
$(\sigma_i^t)^2$	The belief state of the variance of arm i at time t in UCL	7
δ^2	A measure of how heavily priors on the means are weighted in UCL	7
K	A tunable parameter in Bayes-UCB affecting the upper confidence bounds Q_i^T	7
Φ^{-1}	The inverse cdf of the standard Gaussian variable	7
ζ	A measure of how good a set of priors given to UCL is, given the arm's deviation from the actual set of means and the level of confidence indicated by σ_0^2	7

Chapter 1

Introduction

The field of robotic search has been increasingly explored in recent publications. One particular problem is that of estimating a field, or building an estimate of a distributed parameter as a function of time and location [8]. Applications include odor plume detection [7] and mapping of forest fires [9]. Another application is the problem of finding the concentration of dissolved oxygen or spilled oil in the ocean. In the latter case, efforts could then be focused on removing oil from places with high concentration.

A robot performing this search task should move around the space enough to guarantee, at least probabilistically, that it is not ignoring a patch with a high resource concentration. At the same time, it would be desired that the robot focus most of its attention to parts of the field with high concentration of the resource to take more measurements and improve its knowledge of the distribution in those areas. A fundamental problem is then finding the optimal balance between exploration and exploitation [11].

It is possible to approach the problem of search in a resource distribution using a Multi-armed Bandit (MAB) framework. In MAB problems, explained more in depth in Chapter 2, an agent is presented with a number of options, each of which gives a reward according to a probability distribution. The goal is to maximize the expected reward in this uncertain environment. Applied to the problem of field estimation, the space is discretized, and each patch corresponds to an arm in an MAB. The uncertainty in measurements is simulated as the sampling process having a Gaussian distribution. This results in a Gaussian MAB.

Deterministic sequencing algorithms for MABs have been proposed, which consist of setting alternating blocks of exploration and exploitation. In the exploration phase,

the arms are played at random, while in the exploitation phase the arm with the highest expected reward is played [12]. The approach used in this project focuses on stationary distributions and takes advantage of the spatial correlation of the rewards, using the Upper Confidence Limit (UCL) algorithm developed by Reverdy et al. [11].

Reverdy et al. sought to incorporate prior knowledge of the field into an MAB search. They began by studying the Upper Confidence Bound (UCB) algorithm developed by Kaufmann et al. [4], which approaches the MAB problem with no prior knowledge. This algorithm was proven by Kaufmann et al. to achieve the best possible performance given by the bound found by Lai and Robbins [6], within a constant factor. They then treated the problem from a Bayesian perspective following an approach by Kaufmann et al. to allow the incorporation of prior beliefs on the arms [4], and obtained improvements in performance. Reverdy et al. then extended the algorithm for the case of correlated arms [11].

In the problem of field estimation, the smoothness of the field can be simulated by generating a correlation structure that represents the correlation of any two arms based on the distance between them. This smoothness is measured by the length scale, which is a metric of how spatially correlated the arms are. A field based on this length scale can then be generated using the correlation structure. Afterwards, a robot can be programmed with the UCL algorithm to determine what points in space to visit and take samples from.

This process enables the agent to perform better than agents that have no knowledge of the spatial correlation of the arms. It is well understood that a knowledge of this length scale allows the agent to perform better: sampling an arm gives the agent information about the arms around it, reducing the need for exploration and allowing faster convergence towards an optimal arm. However, in the case of robotic search in the ocean, the length scale of the field cannot really be known exactly, since one would likely only have an approximate model that determines a theoretical length scale. An important question is then how making an incorrect estimate of the length scale can impact performance. This issue has not been studied in depth before.

The purpose of this thesis is to evaluate and compare the performance of UCL with correlation as the estimate of the length scale is varied while the length scale associated with the field is kept constant. Chapter 2 gives a review of MAB problems and describes the setup used to perform the robotic search. Chapter 3 describes implementation issues: the control scheme used for the vehicles and the generation of a correlated field. Chapter 4 presents a simulation study on a correlated field comparing the performance of different MAB algorithms against UCL with correlation, and a

study evaluating the performance of UCL with correlation when the estimate of the length scale is incorrect. Finally, Chapter 5 presents an experimental study of this last simulation highlighting implementation issues.

Chapter 2

Background

2.1 MAB Problems

This section follows the review of MAB problems given by Reverdy et al. [11].

In a Multi-Armed Bandit, an agent is presented with N options, usually called "arms". Each arm provides rewards according to a distribution, which is not known to the agent. The problem is for the agent to choose a sequence of arms to maximize the cumulative expected reward, which requires a balance of exploration (choosing new arms to gather information about their rewards) and exploitation (choosing arms that have been known to give high rewards to maximize reward).

In this section, we follow primarily the notation given by Reverdy et al. [11]. Given a time horizon $T \in \mathbb{N}$, for time $t \in \{1, ..., T\}$, the agent obtains a reward $r_t \in \mathbb{R}$ by playing arm i_t. Each reward from arm $i \in \{1, ..., N\}$ is "sampled from a stationary distribution p_i and has an unknown mean $m_i \in \mathbb{R}$". The agent's "objective is to maximize the cumulative expected reward $\sum_{t=1}^{T} m_{i_t}$ by selecting a sequence of arms $\{i_t\}_{t\in\{1,...,T\}}$." By defining $m^* := \max\{m_i | i \in \{1, ..., N\}\}$, and $R_t = m^* - m_{i_t}$ or the "expected regret at time t", this objective can be formulated equivalently as minimizing the cumulative expected regret:

$$\sum_{t=1}^{T} R_t = Tm^* - \sum_{i=1}^{N} m_i \mathbf{E}[n_i^T] = \sum_{i=1}^{N} \Delta_i \mathbf{E}[n_i^T]. \tag{2.1.1}$$

Here, n_i^T is the number of times that arm i is chosen up to time T, and $\Delta_i := m^* - m_i$ is the "expected regret due to picking arm i instead of" the optimal arm.

Thus, at any given time t, exploration corresponds to choosing the arm that "is

estimated to have the highest mean at time t", and exploration refers to choosing "any other arm". The problem becomes a matter of finding the right balance of exploration and exploitation: too much exploitation might result in a suboptimal arm being played an excessive number of times, while too much exploration results in a higher number of suboptimal arm choices. Enough exploration is needed "to learn which arm is most rewarding" to then exploit this arm.

Lai and Robbins showed that the expected number of times that a suboptimal arm is chosen has a logarithmic lower bound in terms of time, which sets a logarithmic lower bound for the cumulative expected regret as well [6].

In the MAB problems considered here, the arms have a Gaussian distribution with mean m_i and variance σ_s^2. This is known as the Gaussian MAB problem. The means are unknown to the agent, but knowledge of the variance is assumed, by previous measurements or knowledge about sensor error. In this case, Reverdy et al. [11] showed, using the bound by Lai and Robbins [6], that:

$$\mathbf{E}[n_i^T] \geq \left(\frac{2\sigma_s^2}{\Delta_i^2} + o(1) \right) \log T. \tag{2.1.2}$$

This means that higher values of σ_s result in higher regret rates, given that the samples obtained are not as significant, while higher values of Δ_i result in lower regret rates, since the arms are easier to distinguish [11].

2.1.1 The UCB algorithm

The upper-confidence-bound algorithm (UCB), developed by Auer et al. [2] achieves logarithmic regret without any prior knowledge of the rewards. In the initialization phase, each arm is played once. Next, at each following time step, a heuristic value Q_i^t is calculated for each arm i, which gives an upper bound on the expected reward of the arm. The arm chosen in time step t is whichever has the maximum value of Q_i^t. The heuristic value is calculated as follows:

$$Q_i^t = \bar{m}_i^t + C_i^t. \tag{2.1.3}$$

Here $\bar{m}_i^t$ is the empirical mean of the arm, and C_i^t is a "measure of the uncertainty in the reward of arm i at time t. In the specific variant used in this project, we use:

$$C_i^t = \sigma_s \sqrt{\frac{2 \log t}{n_i^t}} \tag{2.1.4}$$

where n_i^t is the number of times that arm i has been played at time t. This has the effect of balancing exploration and exploitation: if an arm has a high mean and a high standard deviation, it is more likely to be chosen, but if it is chosen very often, the term C_i^t decreases, and other arms will likely be chosen instead. This variant of the algorithm for a Gaussian MAB was termed UCB1-normal by the authors, and achieves logarithmic regret. However, the general UCB1 algorithm they developed achieves logarithmic regret with a larger constant than the optimal one.

2.1.2 Deterministic UCL algorithm with uncorrelated priors

Kaufmann et al. [4] treated the MAB problem from a Bayesian perspective, which allows the incorporation of priors into the algorithm. They proposed using the quantile function of the posterior reward distribution as the heuristic function Q_i^t.

As explained by Reverdy et al. [11], for a random variable $X \in \mathbb{R} \cup \{\pm\infty\}$ with a probability distribution function $f(x)$, the cumulative distribution function (cdf) of X is defined as $F(X) = \mathbf{P}[X \leq x]$. The quantile function $F^{-1}(p)$ is then defined with domain $[0, 1]$ and codomain $\mathbb{R} \cup \{\pm\infty\}$, such that:

$$\mathbb{P}(X \leq F^{-1}(p)) = p. \tag{2.1.5}$$

Thus, $F^{-1}(p)$ "is an upper confidence bound, i.e. an upper bound that holds with probability, or confidence level, p" [11]. Given a distribution $p_i(r)$ with cdf F of option i, we can take $Q_i = F^{-1}(p)$ to find an upper bound that holds with confidence p. Choosing a high p results in a bound that holds with high probability.

Kaufmann et al. [4] chose $p = 1 - \alpha_t$, where $\alpha_t = 1/(t(\log T)^c)$ so that α_t is of order $1/t$. This has the effect of generating better bounds as time advances, so that the agent is more likely to choose the optimal arm. This algorithm was named by the authors as Bayes-UCB.

Based on this Bayesian algorithm, Reverdy et al. [11] developed the upper-credible-limit (UCL) algorithm for the MAB problem. We will focus here on the case of a deterministic decision maker (the authors also developed a stochastic version of the algorithm). The prior on arm i is a random Gaussian with mean μ_i^0 and variance σ_0^2. Moreover, "conditioned on the number of visits n_i^t to arm i and the empirical mean $\bar{m}_i^t$, the mean reward at arm i at time t is a Gaussian random variable". [11] This variable has mean and variance:

$$\mu_i^t := \frac{\delta^2 \mu_i^0 + n_i^t \bar{m}_i^t}{\delta^2 + n_i^t} \tag{2.1.6}$$

$$(\sigma_i^t)^2 := \frac{\sigma_s^2}{\delta^2 + n_i^t} \tag{2.1.7}$$

where $\delta^2 = \sigma_s^2/\sigma_0^2$. At each time step t, the selects the arm i with maximum:

$$Q_i^t = \mu_i^t + \sigma_i^t \Phi^{-1}(1 - 1/(Kt)). \tag{2.1.8}$$

K is a tunable parameter, chosen as $\sqrt{2\pi e}$ by the authors. In addition $\Phi^{-1} : (0,1) \to \mathbb{R}$ is the inverse cdf of the standard Gaussian variable.

Reverdy et al. [11] showed that this algorithm achieves, in general, logarithmic regret. The regret growth rate is similar to that of the UCB algorithm for uninformative priors (large σ_0^2). For informative priors (small σ_0^2), the performance depends on the quality of those priors. The authors define a metric on the priors $\zeta := \max\{|m_i - \mu_i^0|/\sigma_0 | i \in \{1, ..., N\}\}$, with good priors corresponding to small values of ζ, and bad ones corresponding to big values of ζ. Good priors achieve logarithmic regret, while bad priors can result in super-logarithmic regret growth.

2.1.3 Deterministic UCL algorithm with correlated priors

Reverdy et al. [11] extended this UCL algorithm for correlated priors. In this case, the priors are in the form of estimates on the means $\mu_0 \in \mathbb{R}^{N\times 1}$ and a correlation matrix $\Sigma_0 \in \mathbb{R}^{N\times N}$ (which must be positive-definite). Defining $\phi_t \in \mathbb{R}^{N\times 1}$ for $t \in \{1, ..., T\}$, such that $(\phi_t)_{(k,1)} = 1$ if $k = i_t$ and 0 otherwise, where i_t is the arm chosen at time t. The belief state (μ_t, Σ_t) is then updated as follows [5]:

$$\begin{aligned} q &= \frac{r_t \phi_t}{\sigma_s^2} + \Lambda_{t-1}\mu_{t-1} \\ \Lambda_t &= \frac{\phi_t \phi_t^T}{\sigma_s^2} + \Lambda_{t-1},\ \Sigma_t = \Lambda_t^{-1} \\ \mu_t &= \Sigma_t q. \end{aligned} \tag{2.1.9}$$

These equations reduce to Equations (2.1.6) and (2.1.7) in the case of uncorrelated priors, when $\Sigma_0 = \sigma_0^2 I_N$. Once the belief state has been updated, Equation (2.1.8) is used to choose an arm, as in the uncorrelated case.

In this document, for simplicity, UCL with correlation will often be referred to as "correlated UCL", and UCL without correlation as "uncorrelated UCL".

2.2 Vehicles and Testbed

The underwater vehicles and testbed used in this project were developed by the Dynamical Control Systems Laboratory at Princeton University under the direction of Prof. Naomi Leonard.

Figure 2.1 shows a picture of one of the underwater vehicles used in this project. The robots have a body based on a symmetric airfoil extruded in the vertical direction and made out of a buoyant spongy material and plastic. The body is stabilized by ballast in a keel beneath it, the weight of which can be adjusted to allow the robot to stay at a constant height when submerged. Three servos allow the body to move: one which controls a vertical propeller, and two which control a tail with a propeller. The vertical propeller is placed normal to the top surface of the body of the robot. The tail of the robot consists mostly of a servo with a propeller attached. Another servo controls the angle that this servo makes with the robot. The tail is placed behind the robot, at the narrow end of the airfoil, parallel to the horizontal plane. At the neutral position, the propeller lines with the plane of symmetry of the robot. When the tail is rotated, it does so by a vertical axis on this same plane of symmetry, allowing the robot to turn left or right.

An Arduino, which is a single-board microcontroller, is placed inside a sealed compartment inside the robot. It is connected to a pressure sensor and controls the servos. A tether connects a main computer, from which control schemes are executed with the Arduino.

Figure 2.2 shows the laboratory setup. A water tank, 2.4 m high and with a radius of 3.2 m serves as the testbed for the robots. Four cameras provide a view of the top of the tank. A visual recognition system is used to identify the horizontal position of the robot. The vehicle's top surface and propeller have a bright magenta color for this purpose. To correct for the effect of parallax and water refraction, the reading from the pressure sensor is used to determine the height of the robot with respect to the cameras. For this purpose, the surface of the tank is kept at a fairly constant height throughout the experiments by means of a visual reference in the tank, and the pressure at the top and at the bottom of the tank are calibrated whenever the robot is first ran during a day.

The communication between the main computer, the cameras, and the Arduino is

handled by Rosserial, a protocol that allows communication between multiple devices. Control is executed from Matlab and through this system.

Figure 2.1: Underwater vehicle for use in the testbed. Picture taken by Peter Landgren and used with his permission.

Figure 2.2: Laboratory testbed with water tank and robots. Picture taken by Peter Landgren and used with his permission.

Chapter 3

Implementation

3.1 Vehicle Control

A control scheme developed by Peter Landgren was used to make the robot reach a desired location in the tank.

The main challenge with the control of the robot is that the tether is a source of nonlinearity that cannot be easily modeled. The tether pulls the robot because of its weight and friction with the bottom of the tank. This force depends on the tether's position in the bottom of the tank, and is thus also dependent on the current and past trajectory of the robot. To counteract the variable pull of the tether, a PI control is an appropriate choice. It eliminates the possibility of a steady state error [1], and can reactively correct the pulling force of the tether.

Consider the problem of making the robot track a reference height. Assuming the robot is neutrally buoyant, that the only actuator that can affect the vertical dynamics is the vertical propeller, and that no other forces act on the robot aside from the tether pull, then the dynamics are given by:

$$m_{rob}\ddot{z} = F_T - b\dot{z} + du \tag{3.1.1}$$

where z is the vertical position of the robot in the tank measured from the bottom, u is the input into the motor, m_{rob} is the mass of the robot, F_T is the vertical component of the force exerted by the tether as a function of time, b is a damping coefficient measuring the drag of the robot in the water approximated as a constant, and d is a coefficient describing the proportionality between the input to the vertical propeller and the thrust exerted, approximated as a constant. Parsons and Preston [10], who

developed the robots, determined most of these parameters: $b = 16.89$ N-s/m, and $m_{rob} = 7.5$ kg. From their work, the approximation can be made that $d \approx 0.1$ N.

If the robot moves higher, a longer section of the tether is being lifted by the robot, so we can make the approximation that the force of the tether is proportional to z. The dynamics then reduce to:

$$m_{rob}\ddot{z} = -fz - b\dot{z} + du \tag{3.1.2}$$

where f a coefficient describing the proportionality of the pull of the tether with respect to z. We can find the transfer function of the system by taking the Laplace transform:

$$m_{rob}s^2\boldsymbol{Z} = -f\boldsymbol{Z} - bs\boldsymbol{Z} + d\boldsymbol{U} \tag{3.1.3}$$

$$\Rightarrow \boldsymbol{P} := \frac{\boldsymbol{Z}}{\boldsymbol{U}} = \frac{d}{m_{rob}s^2 + sb + f} \tag{3.1.4}$$

where $\boldsymbol{U}$ and $\boldsymbol{Z}$ are the Laplace transforms of u and z, respectively, and $\boldsymbol{P}$ is the transfer function from u to z. A controller with transfer function $\boldsymbol{C}$ can then be used to track a reference height r with this plant as shown in Figure 3.1.

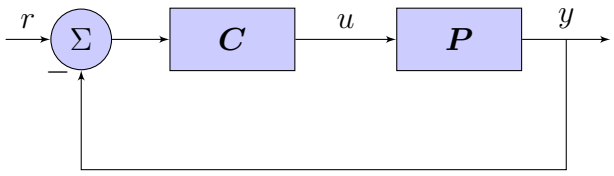

Figure 3.1: Feedback loop for tracking a reference height

For a PI controller, we have $\boldsymbol{C} = K_p + K_i/s$ for constants K_i and K_p. The transfer function from r to y then becomes:

$$\frac{\boldsymbol{PC}}{1 + \boldsymbol{PC}} = \frac{d(sK_p + K_i)}{m_{rob}s^3 + bs^2 + (f + dK_p)s + dK_i} \tag{3.1.5}$$

Note that the parameter K_p allows for compensation for the tether force in the denominator, so that K_p can be chosen so that the tether has no effect on the stability of the system. However, the model used here is only an approximation of the tether pull, and the true dynamics are more complex and path-dependent. For example, if the tether becomes twisted, it can exert significantly more force than when it is not. Alternatively, when the robot approaches the bottom of the tank, the rigidity of

the tether can actually provide an upward force on the robot. These effects can be alleviated by integral control, which allows the system to achieve zero steady state error in spite of the variable nature of the tether pull. Since the dynamics of the tether complicate the control problem, the gains in the controller were chosen experimentally by Peter Landgren. In that control scheme, $K_p = 600$, and $K_i = 25$. Figure 3.2 shows a step response of 1 m for the closed loop, ignoring the effect of the tether. After 1.75 s, the robot stays within 10 cm of the desired height.

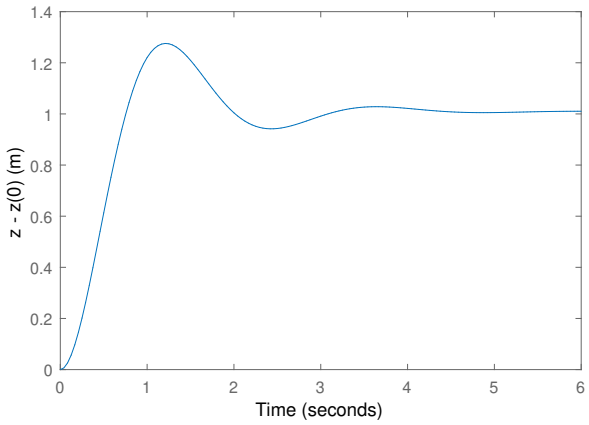

Figure 3.2: Step response for tracking of a reference height

Thus, the controller developed by Peter Landgren used a PI controller to control the vertical propeller of the robot. A similar approach was used in that control scheme for horizontal adjustments, using two different controllers: an outer control for when the robot's distance to its destination was greater than 0.75 m, and an inner control for when the distance was smaller or equal than this number. This inner controller was used to reduce overshoot and provide more steady position holding capabilities. Given the practical limitation of the maximum capacity of the motors, when the calculated output exceeded this limit, the corresponding servo was simply set at its maximum allowable capacity.

For the outer horizontal controller, the angle between the heading direction and the direction to the destination was calculated. If this difference was greater than 90°, then the thrust was applied at a 90° angle, with intensity proportional to the difference in angle. If the difference was smaller than 90°, two components for the thrust were calculated: a tangential component proportional to the difference in angle (with the same gain as in the case of a difference angle greater than 90°), and a parallel component proportional to the distance to the destination. The tail angle was then

adjusted so that the tail's thrust would be in the direction of the resulting thrust vector, and the thrust intensity was calculated as the magnitude of this vector.

The inner horizontal controller was mostly the same as the outer horizontal controller. The main difference is that the parallel thrust component was calculated using a PI control scheme, as opposed to the P controller. The purpose of this inner controller was to bring the robot closer to its destination, so elimination of steady-state error was desirable.

One issue that the current version of the controller does not account for is the induced spin on the robot due to actuation of the top propeller. In a future iteration, the horizontal controller should compensate for this effect. Another issue is that the tail of the robot can get misaligned, so that the control scheme results in the robot going in circles when getting close to its destination if the tail is badly aligned. This problem currently requires frequent checks of the tail before running the robot in the tank. Finally, it is still possible for the tether to become so tangled that the robot is unable to overcome its pull. Substitution of the tether by wireless communication would simplify most of these issues, and better control schemes could then be applied.

3.2 Field Generation

To simulate a continuous field in the tank, correlated rewards in a discretization of the space were generated. This discretization was first made by using a square grid in which the points were set a fixed distance apart. This distance was tuned by experiment to obtain significant difference in performance between different algorithms, while keeping the running time needed practical. Points were generated at a distance of at least 0.5 meters from the edge of the tank, to give the robot clearance from the wall.

Once these points were set, spatially correlated rewards were generated for them following the procedure outlined in Johnson et al. [3]. A correlation matrix $\Sigma \in \mathbb{R}^{N \times N}$ was generated as follows:

$$\Sigma_{i,j} = \begin{cases} 1, & \text{if } i = j \\ e^{-d_{i,j}/\lambda^*}, & \text{if } i \neq j \end{cases} \tag{3.2.1}$$

where $d_{i,j}$ is the distance between arms i and j, and λ^* is the length scale of the field. λ^* determines how strongly correlated the arms are. Duplicating the value of λ^* will mean that an arm will need to be half the distance to be equally as correlated to

any given arm. Thus, higher values of λ^* result in smoother fields, and a sample of an arm gives more information about the arms around it than in the case of a small value of λ^*. In the limit $\lambda^* \to \infty$, the arms become uncorrelated, and in the limit $\lambda^* \to 0$, the arms become uniform.

Parameters m_r and σ_r^2 are then chosen. m_r is the mean around which the correlated rewards will be distributed, and σ_r^2 is the variance of the correlated rewards about m_r. A vector $R \in \mathbb{R}^{N\times 1}$ is generated, with each entry being as random Gaussian variable with mean 0 and variance σ_r^2.

A vector $C \in \mathbb{R}^{N\times 1}$ of N correlated rewards is then generated as follows:

$$C = m_r J_{N,1} + QR \tag{3.2.2}$$

where $J_{N,1}$ is an $N \times 1$ matrix of ones, and $Q \in \mathbb{R}^{N\times N}$ is calculated using the Cholesky decomposition from Σ, so that $QQ^T = \Sigma$.

The correlation matrix Σ_0 given to UCL with correlation with a degree of confidence indicated by σ_0^2 and based on a length scale prior λ is calculated similarly as Σ:

$$\Sigma_{0_{i,j}} = \begin{cases} \sigma_0^2, & \text{if } i = j \\ \sigma_0^2 e^{-d_{i,j}/\lambda}, & \text{if } i \neq j \end{cases} \tag{3.2.3}$$

Figure 3.3 shows intensity plots for reward fields for the tank generated using different values of λ^*, with an edge length of 0.1 m, $\sigma_r = 25$, and $m_r = 75$. The field is generated for a circular region, so the surrounding region in these plots is not part of the reward field (and thus has been set to have no influence on the scale of the plots). For $\lambda^* = 10^{-3}$ m the arms are effectively uncorrelated, and the field corresponds mostly to noise. As λ^* is increased to 0.2 m, small clusters of similarly valued arms begin to form in the area. At $\lambda^* = 0.8$ m, larger clusters form, and at $\lambda^* = 10^3$ m, the clusters are bigger, and the field is more uniform. Note as well that as λ^* increases, the range of rewards becomes narrower, so that with $\lambda^* = 10^3$ m any two arms differ by less than 5, but with $\lambda^* = 10^{-3}$ m, the biggest difference is greater than 150. As can be seen, the length scale effectively captures the smoothness of the field.

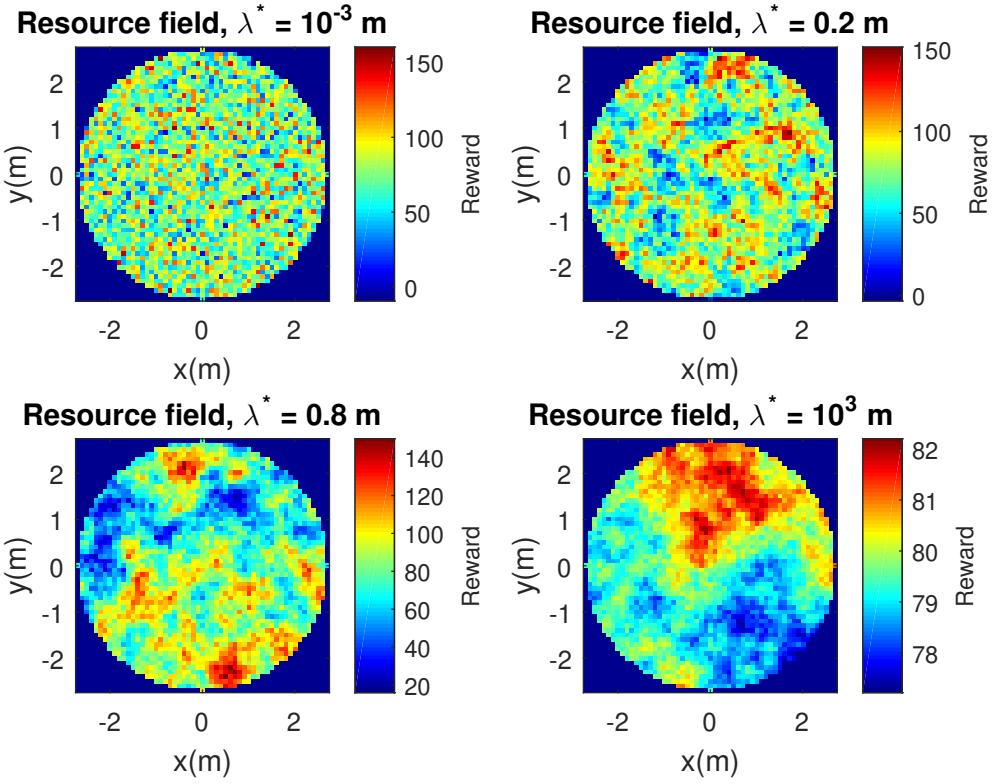

Figure 3.3: Resource fields generated using different values of λ^*

Chapter 4

Simulation Studies

In a robotic field estimation task, the smoothness of the field can vary depending on the specific application and environmental conditions. Therefore, to evaluate the applicability of UCL with correlation for this problem, we must assess the performance of this algorithm for different values of λ^*. Moreover, knowledge of the exact value of λ^* is unlikely to be available, so it is desirable to gain an understanding of how the performance of the algorithm is affected when the length scale estimate λ differs from λ^*. The simulation studies in this chapter seek to explore these issues.

The algorithms and calculations in this chapter were all implemented using MATLAB (see Appendix B for specific implementations). Unless indicated otherwise, the procedure used to generate regret curves for different algorithms was as indicated in these paragraphs. The rewards were generated using $m_r = 75$ and $\sigma_r = 25$. σ_s was set to 30, and σ_0 was set to 20. At each trial, an appropriate set of means was generated. Each algorithm was run on the same means for that trial.

If using UCL with or without correlation, a set of priors on the rewards was generated and given to both versions of the algorithm on that trial. The priors were either a prior on a "master mean" corresponding to a uniform prior of m_r for all the arms, or a "specific prior" generated by taking the actual rewards and adding Gaussian noise with standard deviation of $\sigma_0 = 20$, which results in good priors (low value of ζ). Each of these methods has its advantages. The specific priors provide a more detailed description of the field. On the other hand, the master mean better describes the way the rewards were generated, since this was done by taking m_r and adding a random Gaussian variable to each arm using the correlation structure. Moreover, in real world applications, it would be easier for an agent to have access to a master mean estimate than estimates of the arms at every point in space.

If using UCL with correlation, Σ_0 was generated for a particular length scale estimate λ with the parameter $\sigma_0 = 20$, using the procedure described in Chapter 3. The average of all the runs was taken, and confidence intervals of 95% were calculated.

4.1 Simulation 1: Effect of length scale on UCL with correlation (1D)

4.1.1 Method

The purpose of this study was to evaluate the advantage that UCL with correlated priors can have over UCB and UCL without correlation, when the estimate of the length scale is correct ($\lambda = \lambda^*$), and for a fixed time horizon and number of arms. For each of the two variants of the UCL algorithm, performance both with the master mean and with specific priors was tested.

10 arms were generated in a one-dimensional space, arranged in a line, each 1 meter apart from its immediate neighbors. Correlated means were generated for a length scale λ^* using the procedure described in Chapter 3. The Σ_0 matrix input to UCL with correlation was generated using $\lambda = \lambda^*$. 2000 time steps were used in each run, and the average was taken over 500 trials.

4.1.2 Results

Figures 4.1, 4.2, and 4.3 show the average regret obtained for the algorithms with $\lambda^* = 1, 5, 10$ m, respectively, with bands representing 95% confidence intervals. In all cases, UCB performs the worst, as expected.

At $\lambda^* = 1$ m, the relative performance of each variant of the algorithm is as expected. UCL with correlation performs better than UCL without correlation for the same kind of priors, although this difference is small. Moreover, any of the two variations of the algorithm performs better with the specific priors.

At $\lambda^* = 5$ m, an unexpected trend appears. Although the same initial order by performance as in the case $\lambda^* = 1$ m is observed in the first time steps of the algorithm, UCL with correlation and the specific means very quickly starts to perform worse than UCL with correlation and the master mean. It also starts to perform worse than the uncorrelated UCL with the specific priors, and is close to reaching the performance of uncorrelated UCL with a master mean. In addition, unlike in the case where $\lambda^* = 1$

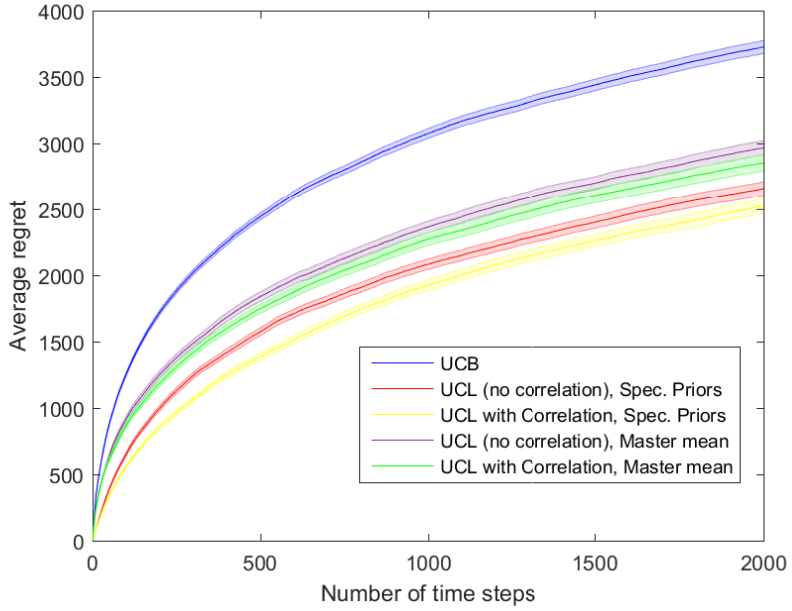

Figure 4.1: Average cumulative regret for 1D arms, $\lambda^* = 1$ m

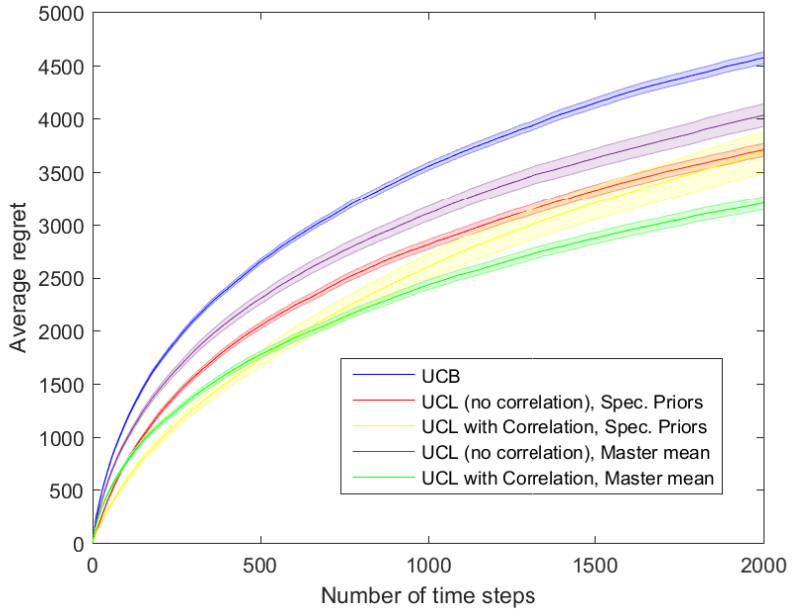

Figure 4.2: Average cumulative regret for 1D arms, $\lambda^* = 5$ m

m, UCL with correlation and the master mean performs better than UCL with the specific means.

At $\lambda^* = 10$ m, the gap between the two regret curves of uncorrelated UCL and that of UCL with correlation with the master mean grows wider. In contrast, the

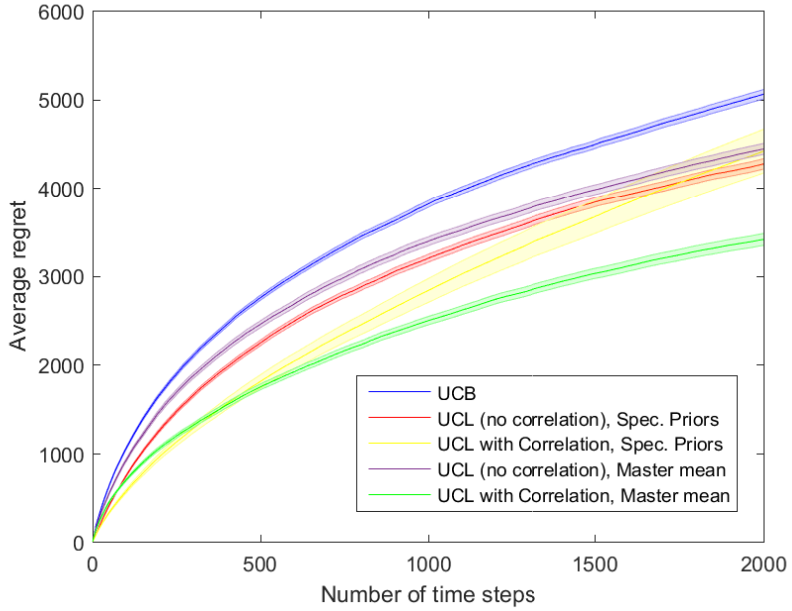

Figure 4.3: Average cumulative regret for 1D arms, $\lambda^* = 10$ m

regret curve of UCL with correlation and the specific priors now intersects those of UCL without correlation earlier. As λ^* keeps increasing beyond this value, this effect tends to become stronger: UCL with correlation and the master mean performs increasingly better than UCL with the specific priors, and UCL with correlation and the specific priors achieves linear regret. Eventually, the means generated are so uniform that most of the regret curves are close to one another, except for that of UCL with correlation and the specific means, which performs worse (see Figure A.1 Appendix A for one such example).

Finally, we can notice that the gap between the regret curves of UCB and UCL without correlation and specific priors narrows as λ^* is increased.

4.1.3 Discussion

As expected, UCL with correlation performs similarly to uncorrelated UCL when λ^* is small: as λ^* approaches zero, the arms become effectively uncorrelated. Moreover, the correlation matrix given to UCL with correlation approaches $\sigma_0^2 I_N$, so performance should approach that of UCL without correlation. In addition, larger values of λ^* result in more uniform reward fields with smaller difference in rewards between the arms, so it is to be expected that the gap in performance between UCB and uncorrelated UCL with specific priors is decreased as λ^* is increased. This is because the

specific priors provided to uncorrelated UCL are a more significant advantage with respect to an uninformed agent performing UCB when the rewards are less uniform.

Given a set of priors on the means, we would expect UCL with correlation to perform better than uncorrelated UCL. In addition, for either UCL with or without correlation, we would expect performance to be better with the specific priors than with the master mean. Indeed, this is the case for uncorrelated UCL in all 3 simulations. Moreover, when using the master mean, UCL with correlation performs better than uncorrelated UCL. Nonetheless, in apparent contradiction with these trends, UCL with correlation and specific priors, which should have the best performance, performs worse than the other algorithms for $\lambda^* = 5$ m and $\lambda^* = 10$ m. It could even be the case that this decay in performance happens eventually, no matter the length scale, so that provided a long enough time horizon, this would also happen with $\lambda^* = 1$ m.

Since the performance seems to be better at the beginning than that of the other algorithms, it seems to be the case that this algorithm initially makes very well informed choices, but eventually converges incorrectly and chooses the wrong arm. In fact, Reverdy et al. [11] remarked that while UCL with correlation converges faster than uncorrelated UCL, correctness depends on how well the prior Σ_0 describes the actual correlation structure of the rewards. Although the arms were generated using the same correlation structure that was provided as a prior, it may be the case that the master mean better complements Σ_0 as a representation of the field. The specific priors are generated with noise added to the actual rewards, resulting in an initial belief state that does not reflect the smoothness of the arms as accurately as the original field.

This noise may also cause UCL with correlation to converge quickly to an arm that is close in space to the optimal one. In fact, since high values of λ^* result in more uniform priors, and since the noise magnitude used to generate the specific priors is kept constant, this would mean that the algorithm could be more easily misled into exploiting a near-optimal arm that is close in space to the optimal one. Afterwards, even if both arms were sampled, the sampling variance would make it harder to differentiate between them if the field is very smooth. This would explain the initial advantage over the other algorithms: the algorithm has more quickly identified the general area around which the optimal arm lies, given the initial combined knowledge of priors on the arms and their correlation structure, but may often exploit a sub-optimal arm.

An important question is then why the combination of the specific priors and

knowledge of the correlation structure results in this bad performance, when either of these alone does not seem to cause the behavior. UCL without correlation, and correlated UCL with the master mean do not exhibit performance worse than that of UCB in any of the simulations. In the case of having the master mean as a uniform prior, the agent is not mislead into any particular arm by its priors on the means. In the case of uncorrelated UCL, the assumption that the arms are uncorrelated causes the agent to explore further, so even if the agent is initially misled, its more frequent visits to arms that initially seem sub-optimal allow it to obtain a more accurate belief about what the optimal arm is.

We can conclude that in the cases analyzed, the performance of UCL with correlation and a master mean was reliably better. A prior on the means was only useful at smaller length scales and during the first series of time steps. Moreover, at smaller length scales, the advantage of having the correct priors was not as significant. In contrast, UCL with the master mean performed better than the other algorithms at a length scale of 10 and higher. Additionally, by comparing its performance with UCL without correlation and specific priors, we can conclude that knowledge of the correlation structure of the rewards was more useful than good priors. The addition of the good priors granted an advantage during the first few time steps, but in the long term, it was best for the algorithm to be provided with the average of the rewards.

4.2 Simulation 2: Effect of length scale prior variation on UCL with correlation in a robotic search task

4.2.1 Method

This study sought to simulate the performance of a robot in the testbed executing an MAB search using a UCL algorithm with correlated priors on a spatially correlated field generated using a length scale λ^*. The goal was to evaluate performance as λ was varied for a fixed λ^*. Given the results from Simulation 1, the priors given to UCL with correlation were in the form of the master mean, for more reliable performance. The arms corresponded to discretized points in a circular area in the tank, with 0.5 m clearance from the tank walls. The discretization consisted of a grid of points with an edge of 0.8 m. Performance was evaluated by calculating the average accumulated

regret and the average distance traveled. This distance was approximated as the total length of a series of straight lines joining each pair of consecutive points visited.

The length scale λ^* was set as 1, 5, or 25 m for three different simulations. For each of these simulations, UCL with correlation was tested for $\lambda = 1, 5, 25$ m. Thus, in one case, $\lambda^* = 1$ m, all runs of UCL but one had an overestimate of the length scale, and in the case $\lambda^* = 25$ m, all the runs of the algorithm but one had an underestimate. These three numbers were chosen as the possible length scales so that they were a geometric progression by a factor of 5.

Finally, to help visualize the trends obtained by increasing or decreasing the length scale, the cumulative regret and the total distance traveled at $t = 1000$ for UCL with correlation were calculated for different values of λ/λ^* ranging between 0.2 and 40, including both 1/5 and 5 for comparison with the previous simulations in this study. These calculations were made for all three values of λ^* used.

1000 time steps were simulated, and the average was taken over 1000 runs. For comparison purposes, UCB and UCL without correlation (and the specific priors) were also run.

4.2.2 Results

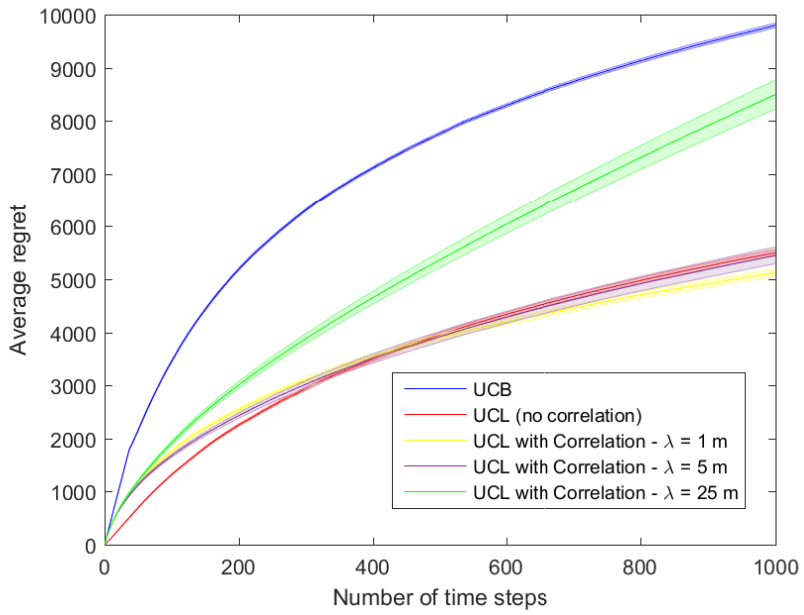

Figure 4.4: Average cumulative regret for agent in tank, $\lambda^* = 1$ m

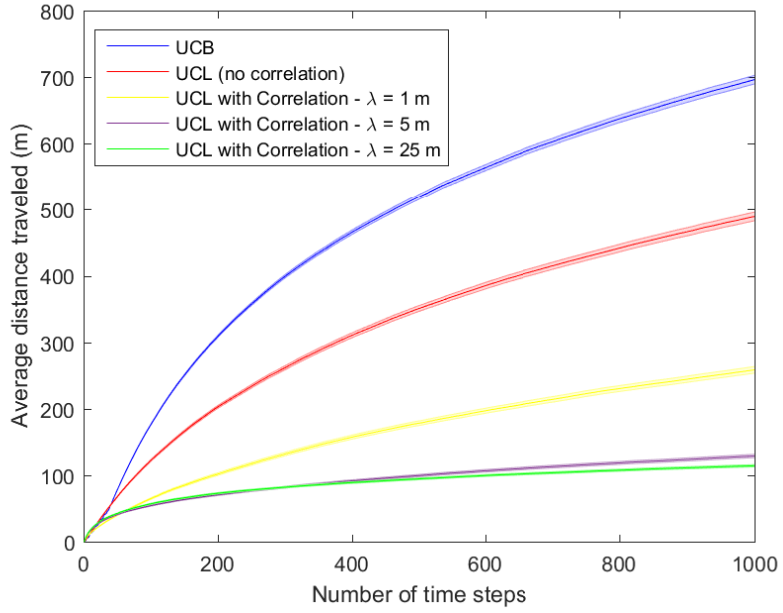

Figure 4.5: Average distance traveled for agent in tank, $\lambda^* = 1$ m

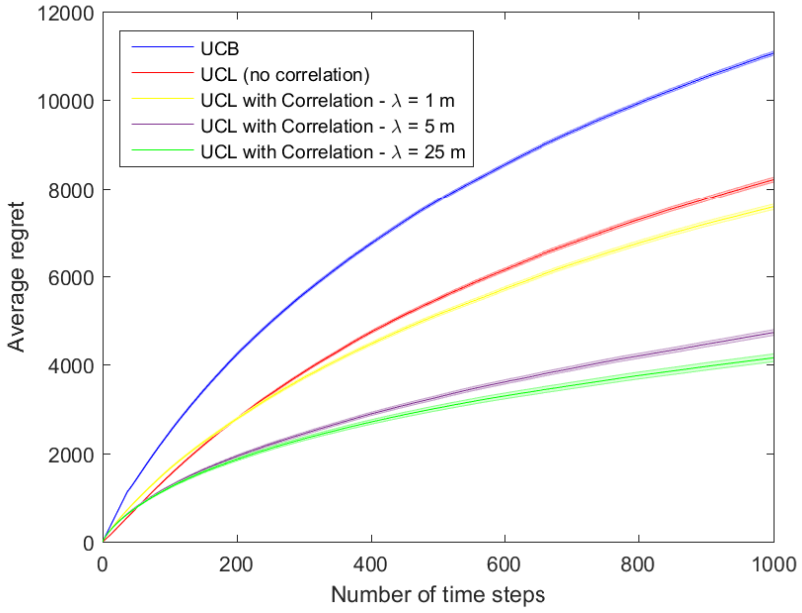

Figure 4.6: Average cumulative regret for agent in tank, $\lambda^* = 5$ m

Figures 4.4, 4.6, and 4.8 show the average cumulative regret for the varying values of λ^*, and Figures 4.5, 4.7, and 4.9 show the average distance traveled curves for these same values, with confidence intervals of 95%.

In the case of $\lambda^* = 1$ m, performance in regret for UCL with $\lambda = \lambda^*$ is close to

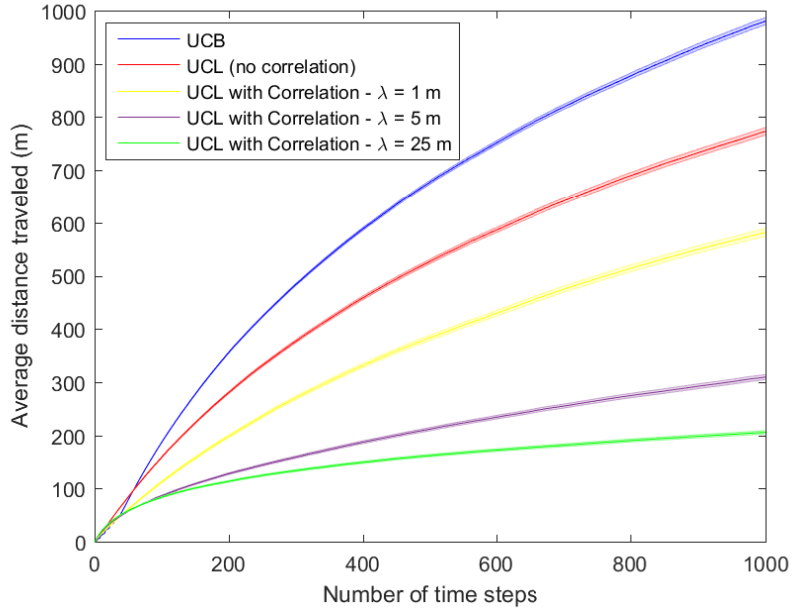

Figure 4.7: Average distance traveled for agent in tank, $\lambda^* = 5$ m

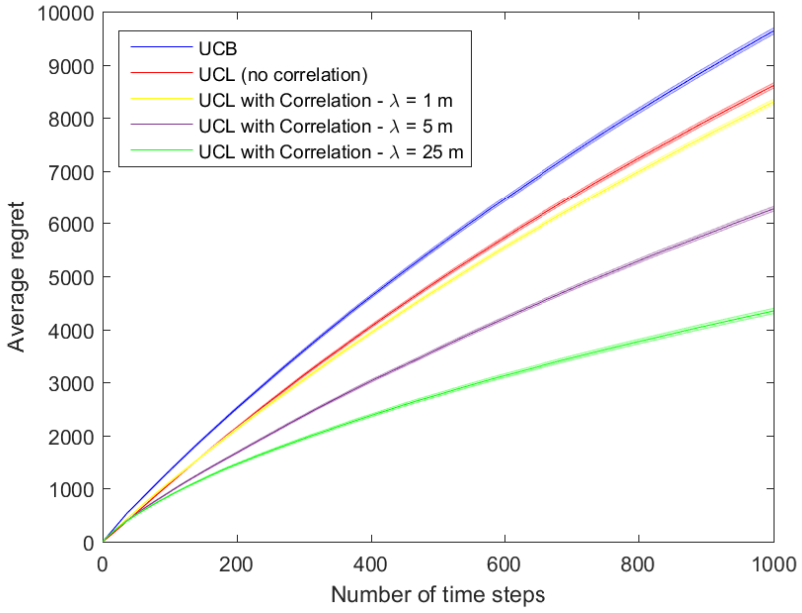

Figure 4.8: Average cumulative regret for agent in tank, $\lambda^* = 25$ m

that of uncorrelated UCL, but has began to be better than it towards the end of the time horizon, consistent with the general trends observed in Section 4.1. In this case, the overestimate $\lambda = 5$ m causes an initial reduction in regret, but this backfires towards the end of the time horizon, where it results in performance closer to that

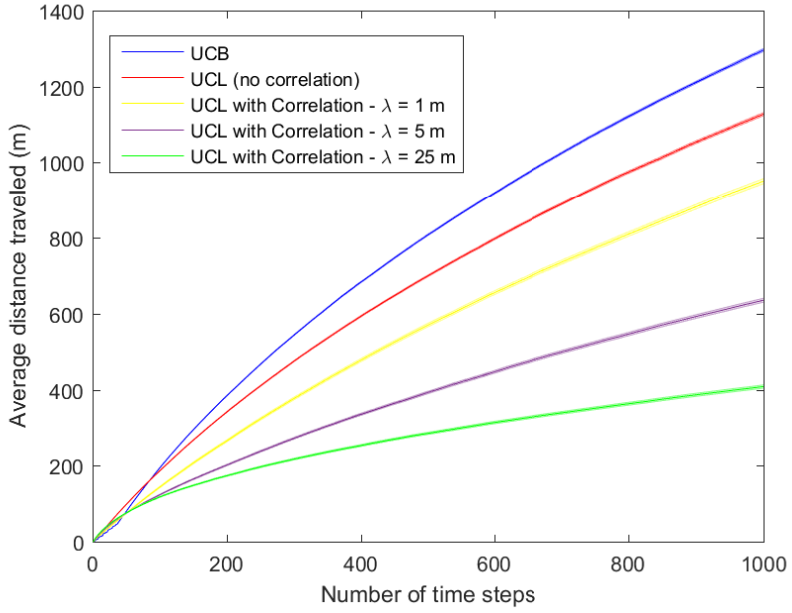

Figure 4.9: Average distance traveled for agent in tank, $\lambda^* = 25$ m

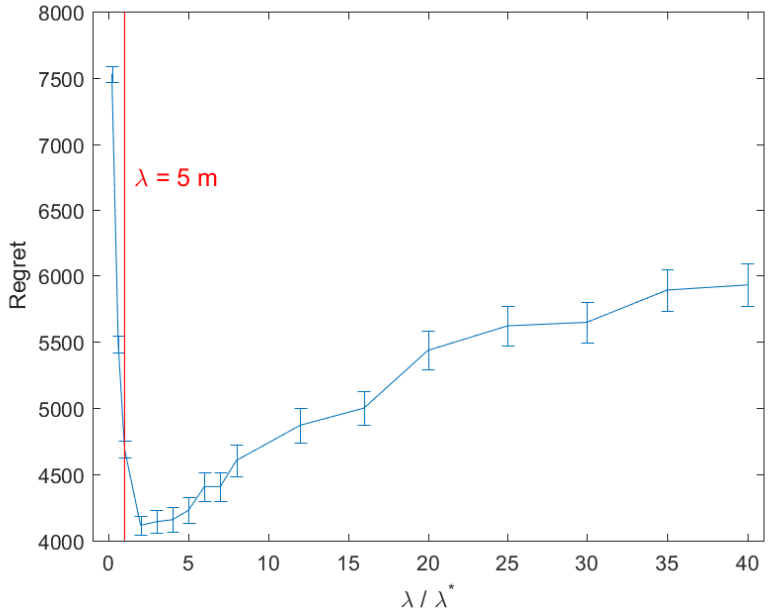

Figure 4.10: Average cumulative regret at $t = 1000$ and $\lambda^* = 5$ m for different values of λ/λ^* using UCL with correlation

of uncorrelated UCL. On the other hand, the overestimate $\lambda = 25$ m results in really bad performance: the regret keeps rising above that of uncorrelated UCL, and even begins to reach that of UCB. In addition, the bigger error bars for this overestimate

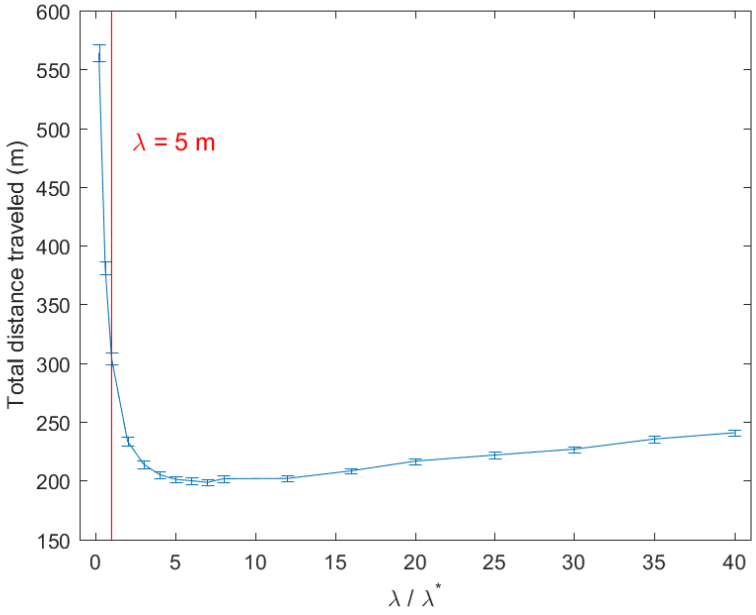

Figure 4.11: Average distance traveled at $t = 1000$ and $\lambda^* = 5$ m for different values of λ/λ^* using UCL with correlation

show higher variability in cumulative regret, and therefore less reliability in long term performance for this length scale estimate. Finally, note that uncorrelated UCL initially performs better than all three runs of correlated UCL, consistent with the results of Section 4.1.

The plot of the average distance shows some more straightforward behavior. The robot using UCB travels the most, then that using uncorrelated UCL, and then those using UCL with correlation in order of λ, from smallest to highest.

For $\lambda^* = 5$ m, the most noticeable change in the cumulative regret is that now UCL with the correct length scale ($\lambda = 5$ m) performs much better than UCL without correlation. The length underestimate of 1 m results in performance close to that of uncorrelated UCL, but this performance improves with respect to that algorithm as time advances. Surprisingly, the overestimate of the length scale, $\lambda = 25$ m, results in lower cumulative regret than $\lambda = 5$ m.

The order of performance by distance traveled is the same as in the case of $\lambda^* = 1$ m, but most notably the gap between the curves of $\lambda = 25$ m and $\lambda = 5$ m has widened; an overestimate of the length scale by a factor of 5 now results in a more significant reduction in distance.

For $\lambda^* = 25$ m, consistent with the results from Section 4.1, the regret curves are steeper than in the other two cases for the same time horizon. Performance with

$\lambda = 1$ m is close to that of uncorrelated UCL. Moreover, bigger values of λ result in smaller accumulated regret.

The distance curves are also steeper, and have the same order by performance as before.

Figures 4.10 and 4.11 show the plots of average cumulative regret and total distance traveled, respectively, in terms of λ/λ^*, at $t = 1000$ and $\lambda^* = 5$ m for correlated UCL, with error bars corresponding to 95% confidence intervals. In the interval tested, minima for the regret and for the distance traveled occur at values of λ/λ^* greater than 1. The plots corresponding to $\lambda^* = 1$ m and $\lambda^* = 25$ m are similar and have this same property for the local minima, and are are shown in Section A.2 in Appendix A.

4.2.3 Discussion

A larger value of λ means that the agent gathers more information about the arms in the vicinity of an arm it samples. This causes the algorithm to converge faster. In the case of small values of λ, an arm does not give much information about the arms around it, and the agent is forced to explore further. The cost of this exploration is an increase in the distance traveled. Consistent with our expectations, we see a reduced travel distance for the robot as the length scale increases for all values of λ^* in Figures 4.5, 4.7, and 4.9, but the distance begins to increase again as λ/λ^* keeps increasing, as can be seen in Figure 4.11 for $\lambda^* = 5$ m and for the other length scales in Figures A.4 and A.5. This indicates that if the length scale is grossly overestimated, the distance traveled can actually increase. A possible explanation for this will be provided later in this section.

In the runs for $\lambda^* = 5$ m and $\lambda^* = 25$ m, underestimating the length scale by a factor of 5 or more results in a higher accumulated regret. In these cases, the agent explores more at the cost of visiting the optimal arm less. Given that underestimating the length scale also results in the agent traveling a longer distance, performance is worse by both metrics.

On the other hand, the results of overestimating the length scale on regret are less straightforward. Overestimating by a factor of 5 when $\lambda^* = 1$ m results in similar regret rates within the time horizon tested, albeit this gap widens with increasing number of time steps. Overestimating by a factor of 25 in this case is fatal, and results in performance that eventually gets worse than that of UCB. This happens because the agent exploits suboptimal arms. Surprisingly, when $\lambda^* = 5$ m, overestimating by

a factor of 5 results in lower accumulated regret. In this specific case, given that the average distance traveled by overestimating by a factor of 5 is also lower, the agent performs best by both metrics by overestimating λ.

These observations are better informed by the data in Figure 4.10. The local minimum of the cumulative regret at time $t = 1000$ appears to occur at some $\lambda/\lambda^* > 1$ This property is also observed for $\lambda^* = 1$ m and $\lambda^* = 25$ m (see Section A.2). It would seem that for each value of λ^* the optimal estimate λ is higher than λ^*. Increasing λ beyond this optimal overestimate results in worse performance, until eventually the regret curve becomes linear, worse in performance than UCB (see Section A.3 in Appendix A for one such example). One possible way of explaining this is that an overestimate that is low enough might combine the benefits of a faster convergence towards exploitation while having a similar degree of a correlation belief between the arms. In other words, although an overestimate of λ should cause an increase in regret because the agent's belief of an arm is influenced excessively by the samples of arms around it (when compared to the real correlation that arms have on each other in the field), as λ is increased beyond λ^* this might initially happen slower than the reduction in regret caused by higher degrees of exploitation. As λ is increased beyond λ^*, the quicker convergence to exploitation provides an initial reduction in regret, until it results in suboptimal arms being exploited.

A similar explanation can be made for the trends in Figure 4.11. Although faster convergence reduces the amount of exploration, doing so also increases excessively the estimates of sub-optimal arms that are too far away from the optimal arm. This results in more arms being visited, and the distance being increased. The result of these two opposite effects could then also be that the overestimate results in the best performance.

Therefore, in the tests provided, it is best to overestimate the length scale than to underestimate it, since overestimating can result in smaller accumulated regret and distance traveled. This suggests that a robot running this search task should err on making its estimate of the length scale big, but not unreasonably so: the cost of overestimating the length scale by a factor of 25 can be fatal, so an accurate model is still desirable. The threshold for λ/λ^* after which performance gets worse is smaller for smaller values of λ^*, so the estimate must be more accurate if the field is less smooth.

Chapter 5

Experimental Study

5.1 Method

The purpose of this experiment was to implement the robotic search simulation described in Section 4.2, in order to gain a better understanding of the results from that simulation and the difference in performance for UCL with correlation for different length estimates λ given a fixed λ^*.

The parameters used to generate the means and discretization of the tank were the same as those in the simulation from Section 4.2. The parameter λ^* was chosen as 5 m, and the length scales used for UCL with correlation were 1, 5 and 25 m, to observe the effects of increasing or decreasing the length scale by a factor of 5.

The surface of the tank was discretized with an edge length of 0.8 m, and each of these points was assigned a reward based on a length scale λ^*, as in Section 4.2. UCB, uncorrelated UCL with specific priors, and correlated UCL with a master mean and three different length scale priors were all ran with this resource field, for a single trial and for 120 time steps. The number of time steps was reduced from 1000 for practical reasons with the robot runs. A trial was chosen that had regret and distance traveled plots representative of those of an average run, which was calculated using 1000 trials.

Once this trial was chosen, the search was simulated in the tank for the UCL with correlation algorithm with the three different values of λ. For each simulation, the sequence of arms and spatial locations generated for the trial were input to the controller described in Section 3.1. First, the robot was made to move to the first point in the search. Once within 20 cm of its destination, the destination was changed.

This was done repeatedly until all the destination points were reached.

Each run was recorded with a camera on top of the tank. For each run, time was calculated from the moment the first arm was reached to the moment the last arm was reached. The distance was calculated similarly, by keeping a record of the coordinates of the robot, taken once roughly every 0.1 seconds, and then adding the straight line distances between consecutive points. The average speed was then calculated from these two quantities.

5.2 Results

Figure 5.1 shows the accumulated regret by the agent for the five different search schemes, and Figure 5.2 shows the average regret with 95% confidence intervals for comparison. Note the order by performance towards the end of the interval is the same for the single trial as for the average. The general behavior is similar to that of the average, with similar values towards the end of the interval, in 3 cases within 1.5 % of the average values (for uncorrelated UCL and UCL with correlation and $\lambda = 1, 5$ m).

Likewise, Figures 5.3 and 5.4 show the (straight-line) distance traveled and the average over 1000 runs, respectively. The order by performance by the end of the interval is the same as in the average case, and the relative differences between the curves are similar, although the actual values are not as close as in the case of the regret curves.

The same trends in average regret and distance traveled are observed as in Section 4.2, except that UCL with correlation and $\lambda = 1$ m has not yet achieved a smaller regret than uncorrelated UCL. Nonetheless, we know that this happens eventually, as can be seen in Figure 4.6, and in fact the gap in regret has already started to close between these two by time step 120.

Thus, the sample trial chosen is representative of the average performance.

Figure 5.5 shows color plots for the resource field and the number of visits for each of the 5 algorithms ran on the field. Note that the three squares at each corner of the plots are not part of the discretized space and are set to have no influence on the color scale. In each of the plots, the color scale is set from the minimum value to the maximum of the quantity being represented, so that the intensity is always relative to these two extremes.

UCB shows the most exploration, with a maximum number of visits on a single arm under 20, and a significant concentration of visits around half of the tank.

Furthermore, the optimal arm is not the most visited one. On the other hand, uncorrelated UCL shows less exploration than UCB. The most visited point has 33 visits, and the visits are more concentrated around the optimal arm.

With UCL with correlation, we observe trends consistent with the previous interpretation of the regret and distance curves. With increasing values of λ, the visits become more concentrated around the optimal arm. The maximum number of visits for $\lambda = 1, 5, 25$ m are 24, 48, and 72, respectively. Only at $\lambda = 25$ m is the optimal arm the most visited point. At $\lambda = 5$ m, an adjacent arm is exploited the most, and at $\lambda = 1$, an arm diagonal to it is. However, given the spatial correlation of the rewards, these are near optimal arms..

Table 5.2 shows a comparison of time, total distance traveled, and average speed for all 3 runs. The underestimation of the length scale, with $\lambda = 1$ m, results in 69.1 s of extra running time when compared to the best of the other two runs, and 31.2 m of extra distance. In contrast, the difference between the other two runs is smaller: 44.8 s and 3.8 m. While the run with $\lambda = 5$ m results in the most time taken to complete the search, $\lambda = 25$ m resulted in the most distance traveled. This is unexpected and seemingly inconsistent with the straight line calculation distance indicating that the bigger length scale resulted in a smaller distance traveled. Nonetheless, the difference is smaller than when comparing to $\lambda = 1$ m.

Figures 5.6, 5.7, and 5.8 show the videos of the robot search runs for $\lambda = 1, 5, 25$ m, respectively, with the resource field color map overlaid over the image. The videos have been sped up by a factor of 8. Note that all of the algorithms converge near the optimal arm as time advances, and that whenever the robot explores an area of the field with low rewards, it travels far away from that area.

λ	Time (s)	Dist. Traveled (m)	Avg. Speed (m/s)
1	455.1	118.3	0.260
5	386.0	83.3	0.216
25	341.2	87.1	0.255

Table 5.1: Comparison of robot performance for different values of λ

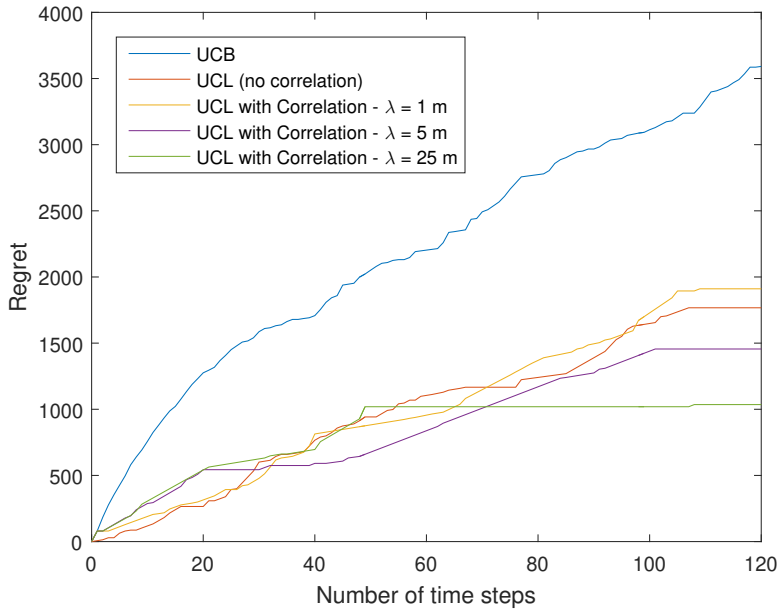

Figure 5.1: Regret accumulated by the robot, $\lambda^* = 5$ m

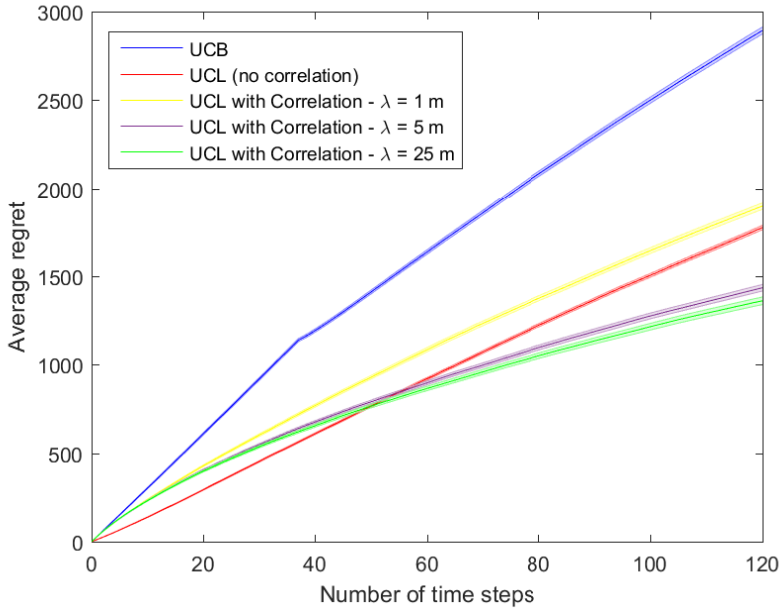

Figure 5.2: Average simulated regret accumulated by the robot, $\lambda^* = 5$ m

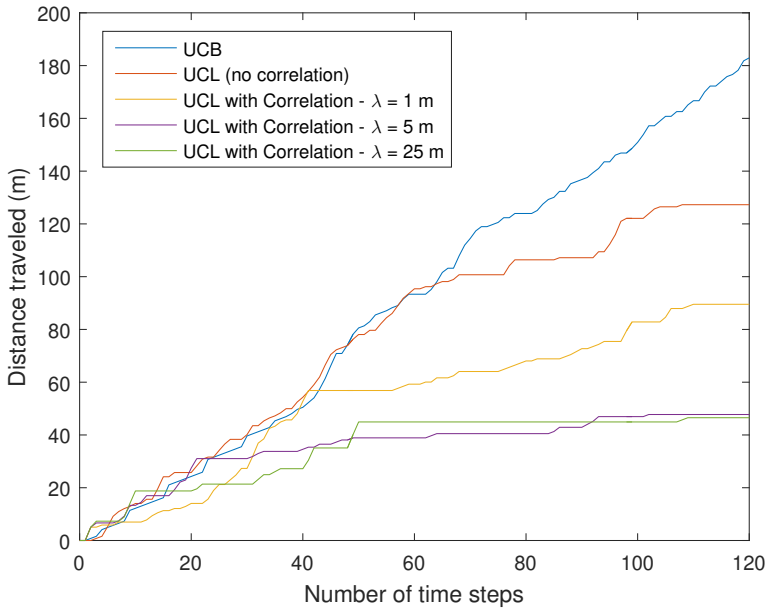

Figure 5.3: Simulated distance traveled by the robot $\lambda^* = 5$ m

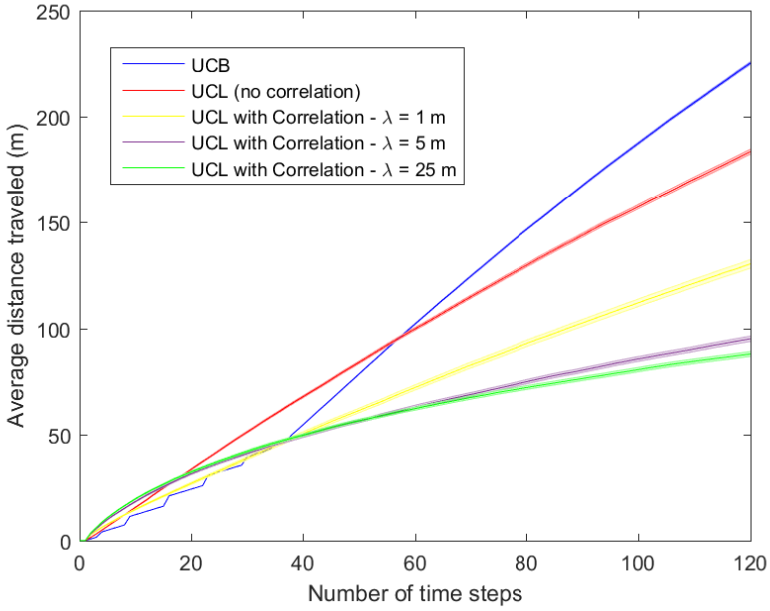

Figure 5.4: Average simulated distance traveled accumulated by the robot, $\lambda^* = 5$ m

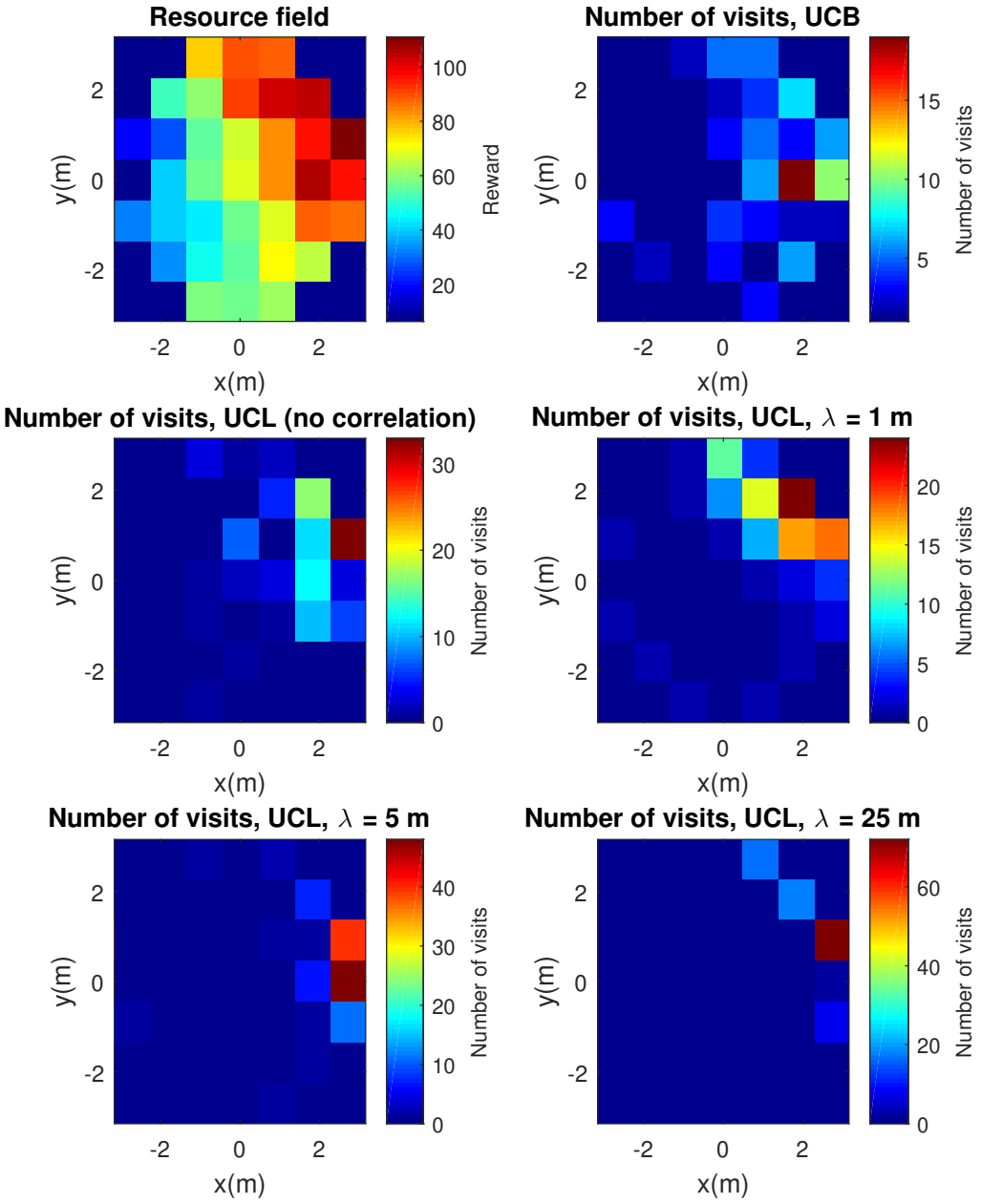

Figure 5.5: Resource Field and Number of Visits per Point for UCB, UCL without correlation, and UCL with correlation for different values of λ

Video of Robot Run

Figure 5.6: Robot Search Task Video, $\lambda = 1$ m

Video of Robot Run

Figure 5.7: Robot Search Task Video, $\lambda = 5$ m

Video of Robot Run

Figure 5.8: Robot Search Task Video, $\lambda = 25$ m

5.3 Discussion

In the three runs executed, performance, in general, improved with growing values of λ. As observed and reasoned before, larger length scale priors result in more exploitation and earlier convergence. All correlated UCL runs converged close to the optimal arm, for all values of λ. This resulted in increasingly smaller values of accumulated regret as λ was increased. The straight line total distance decreased as a result of the quicker convergence; constant exploitation of an arm means less travel for the robot.

Regarding the simulation, the results give insight as to why overestimating the length scale can result in better performance. Note that this particular resource field is almost a gradient from one end of the tank to another. This means that when the robot explores the region of lowest rewards at the left, it is likely driven away as much as possible from it. In fact, we see this with the first two time steps of each of the robots: first, a square in the center left region is visited. This square has a small reward, and in the second time step, the robot visits the square farthest away from this, which is a point with high reward. If λ is bigger, but not big enough that the estimates are heavily distorted, this effect may cause quicker convergence to the optimal arm. An arm with a small reward in this particular field is on the opposite end of the optimal arm, and a higher value of λ means that this estimate will affect more the arms around it. Given that the optimal arm is at the opposite end, this causes the robot to begin focusing on the opposite side of the tank quicker, where the best rewards lie. This reasoning is consistent with the discussion for Simulation 2 in Section 4.2.

In spite of this, we must keep in mind that this is a particular trial, which however was representative of the average performance. Given the number of discretization points, there is a relatively high probability that the optimal arm will lie on the edge of the tank. In fact, it is very unlikely that it will lie in the middle 9 squares: 9/37. Thus, the probability that the optimal arm lies outside of this area is approximately 75.7%, and this is close to or at the edge of the discretization area. Given the smoothness of the field, it could be likely that the least favorable rewards would be located in the opposite edge of the tank. Thus, we could expect more fields that resemble a gradient from one edge to the other to appear.

If this is the reason that overestimating the length scale results in better performance, it is possible that increasing the area of the field would make this effect diminish or even disappear, because doing so would decrease the fraction of points

that lie on the outer ring of the field (the length of the circumference grows proportional to the radius, while the area grows proportional to the radius squared). This would mean that less fields would look like a gradient, so there could be more instances in which overestimating the length scale by 5 would hurt the agent's chances of finding the optimal arm quickly. These results suggest that, although overestimating the length scale can result in better performance, the benefits of this overestimate could diminish or disappear for other discretization configurations.

With regards to the performance of the robot in executing the tasks, the tendency towards more exploitation given by increasing λ is shown, in general, in the metrics summarized in Table 5.2. The time taken was reduced with increasing values of λ. While the increase in actual distance traveled from $\lambda = 5$ m to $\lambda = 25$ m is unexpected, we must keep into account from Figure 5.3 that the theoretical straight line distances were close to each other. The control scheme used does not really ensure a straight line trajectory, so this slight difference could be easily reversed. Moreover, there were factors that could further increase the distance traveled in any given run. The tether pulled the robot in ways that depended on the robot's previous trajectory, and this occasionally caused the robot to struggle to reach a point because it had to drag the cable through the bottom of the tank. Moreover, although the robot's tail fin was aligned before each run, slight misadjustments could result in occasional overshoots in the form of ample curves when the robot traveled long distances at full speed. Additionally, although the control scheme used sought to keep the robot at a constant height, the robot was free to move in the vertical direction subject to the dynamics of the system, possibly adding to the distance traveled. Finally, the straight line distance is an approximation that does not take into account the effects of inertia, which increase the distance traveled in sharp turns. All of these nuances and dynamics may cause the difference in average speed.

Chapter 6

Conclusions

A spatial field estimation task was approached as a Gaussian MAB problem with arms corresponding to discretized points in space. The smoothness of the field was modeled by a spatial correlation structure for the arms, based on a parameter λ^*, the length scale of the field. An algorithm for Gaussian MAB problems with correlated arms, UCL, was then used in this problem. A robot was put on a tank to perform the search. Performance was then evaluated under different estimates of the length scale prior λ provided to the algorithm.

Simulations showed that the UCL algorithm can exploit this knowledge of the correlation and obtain significant reduction in regret when compared to the more traditional UCB algorithm. For the fields generated of 1D arms with varying degrees of smoothness, it was shown that for UCL this knowledge of the correlation can be even more effective in reducing regret than good priors (priors with an appropriate degree of confidence, when considering how much they differ from the true values of the rewards). It was also found that, in this task and with the noise level used, UCL with correlation worked more reliably when the priors provided are the mean of all the arms, instead of a noisy estimate of the reward at each point.

A spatial search in a 2D circular surface corresponding to a discretized version of the tank surface was then simulated for different values of λ^*, when UCL with correlation is provided with estimates of the length scale λ that do not necessarily correspond to λ^*, and when the priors on the means provided correspond to the mean of the field. Performance was evaluated in terms of regret and distance traveled. Although it was shown that an underestimate in length scale resulted in worse performance, it was also found that better performance was attained both in distance traveled and regret for overestimates of the length scale, provided these estimates

were not grossly inaccurate. The size of this threshold was found to be dependent on the length scale of the field λ^*.

Finally, this 2D search task was implemented using a robot in the testbed, using a representative run of the second simulation study, with $\lambda^* = 5$ m. A run with performance similar to that of the average was chosen. Visualizations of the field and the number of visits for each point in the discretization provided insight as to how changing the length scale affects performance. Performance was also evaluated by measuring the time taken by the robot to complete the task, the total distance traveled, and the average speed.

It was found that with the length scale estimates tested, and consistent with the results of the second simulation study, smaller length scale estimates resulted in more exploration. Accordingly, the simulations showed that smaller values of λ resulted in a longer distance traveled and a longer time to finish the run. A discrepancy was found between the simulated distance and the actual distance traveled in two runs, but it was likely due to control issues and approximations of the simulated distance.

The experiments presented show that UCL with correlation can be used effectively to perform a spatial field estimation task under a Gaussian MAB framework for fields of varying smoothness. The effectiveness of this approach varies depending on how well the length scale estimate matches the actual length scale that describes the spatial correlation of points in the field. It was shown for fields of varying smoothness that an overestimate of the length scale can result in better performance. This leads to the conclusion that, in the search task provided, an agent performing UCL with correlation that can only estimate the length scale of field should make an overestimate instead of an underestimate, with the consideration that this overestimate must not exceed some threshold, the size of which depends on the smoothness of the field.

Future work can focus on exploring these effects on performance in a wider range of fields with varying field length scales, number of dimensions, number of points in the discretization, and area and shape of the discretized region. A better understanding of the conditions under which an overestimate of the length scale can yield better results than a correct estimate is necessary. If the optimal estimate is also an overestimate in a wider range of conditions, a way of estimating the optimal value of λ would be desirable. This would dictate the tolerance needed in the estimate of the length scale. In addition, performance in distance traveled can be improved by introducing transition costs into the model. The studies in this thesis could then be repeated for such an implementation. Finally, the effects of length scale estimates could be studied in the multi-agent problem. In that case, the space is separated into areas,

each assigned to a robot that searches only in that area. Each robot has access to the information obtained by the other robots. For smooth fields and good estimates of the length scale, the measurements taken by other robots would give a robot more information about its area. This should result in faster convergence and better performance than when the robots have no knowledge of the spatial correlation.

Bibliography

[1] K. J. Astrom and R. M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, NJ, 2008.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[3] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Pearson, Upper Saddle River, NJ, 6 edition, 2007.

[4] E. Kaufmann, O. Cappe´, and A. Garivier. On bayesian upper confidence bounds for bandit problems. *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, pages 592—-600, 2012.

[5] S. M. Kay. *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*. Pentice-Hall, Englewood Cliffs, NJ, 1993.

[6] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, 1985.

[7] A. Marjovi and L. Marques. Optimal spatial formation of swarm robotic gas sensors in odor plume finding. *Autonomous Robots*, 35(2):93–109, 2013.

[8] M. F. Mysorewala, L. Cheded, and D. O. Popa. A distributed multi-robot adaptive sampling scheme for the estimation of the spatial distribution in widespread fields. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):1–19, 2012.

[9] M. F. Mysorewala, D. O. Popa, and F. L. Lewis. Multi-scale adaptive sampling with mobile agents for mapping of forest fires. *Journal of Intelligent and Robotic Systems*, 54(4), 2008.

[10] B. Parsons and J. Preston. The beluga project: Development of a testbed for autonomous underwater vehicles. Princeton University Senior Thesis, April 2011.

[11] P. B. Reverdy, V. Srivastava, and N. E. Leonard. Modeling human decision making in generalized gaussian multiarmed bandits. *Proceedings of the IEEE*, 102(4):544–571, 2014.

[12] S. Vakili, K. Liu, and Q. Zhao. Deterministic sequencing of exploration and exploitation for multi-armed bandit problems. *IEEE Journal of Selected Topics in Signal Processing*, 7(5):759–767, 2013.

Appendix A

Additional Plots

A.1 Additional results of Simulation 1

The plot in this section provides an additional result of the experiment in Section 4.1, for $\lambda^* = 100$ m. Note that UCL with correlation and a master mean gives the lowest regret, but with the specific priors, it gives the highest regret.

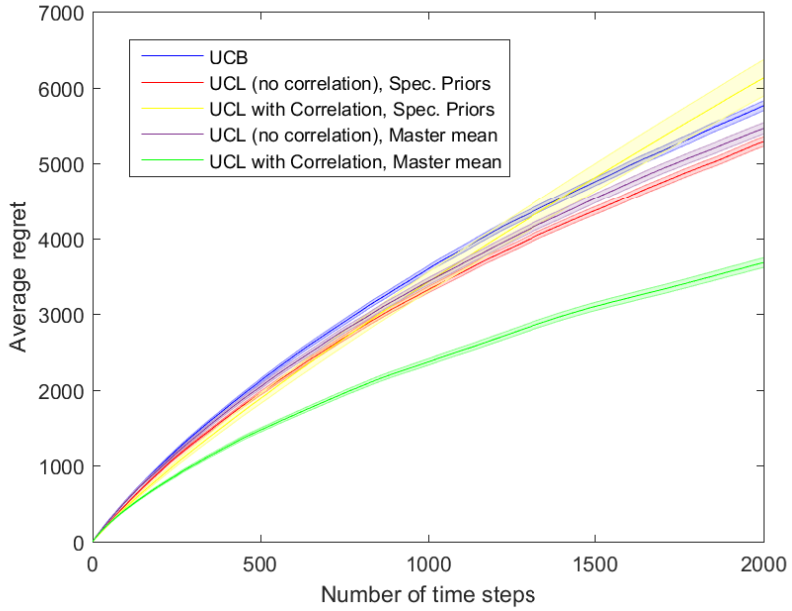

Figure A.1: Average cumulative regret for 1D arms, $\lambda^* = 100$ m

A.2 Plots of performance of UCL with correlation for varying values of λ/λ^*

The plots in this section are the results of the study described in Section 4.2 and complement the graphs already included there.

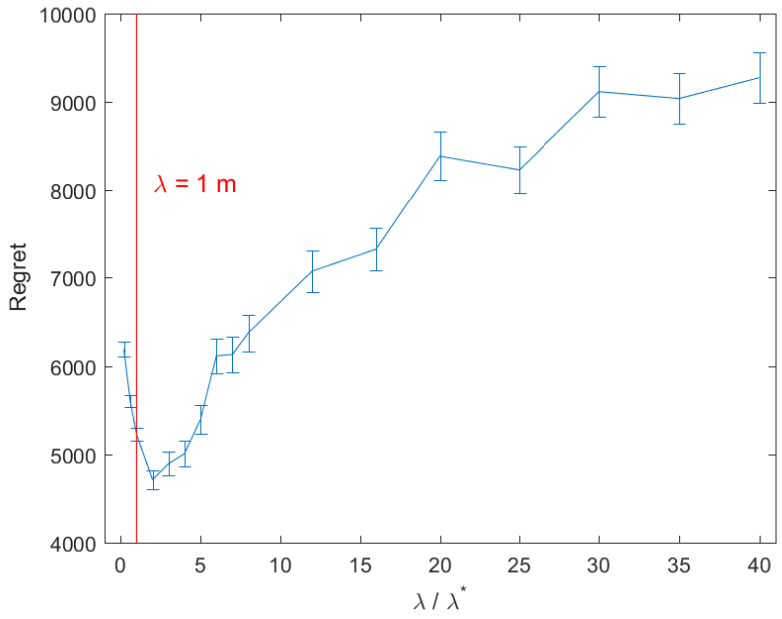

Figure A.2: Average cumulative regret at $t = 1000$ and $\lambda^* = 1$ m for different values of λ/λ^* using UCL with correlation

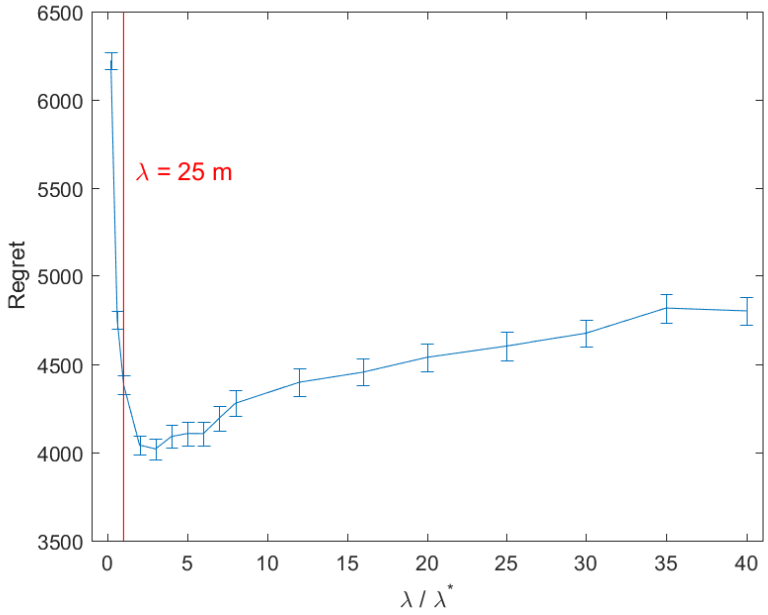

Figure A.3: Average cumulative regret at $t = 1000$ and $\lambda^* = 25$ m for different values of λ/λ^* using UCL with correlation

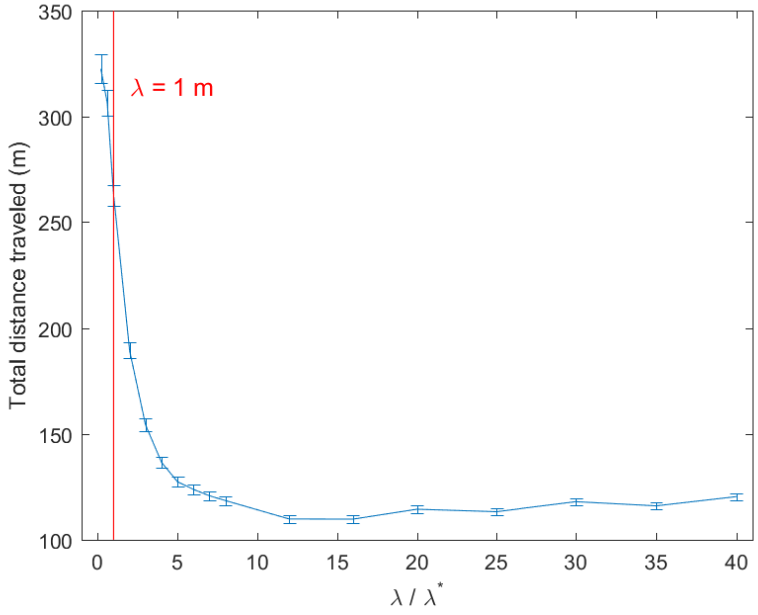

Figure A.4: Average distance traveled at $t = 1000$ and $\lambda^* = 1$ m for different values of λ/λ^* using UCL with correlation

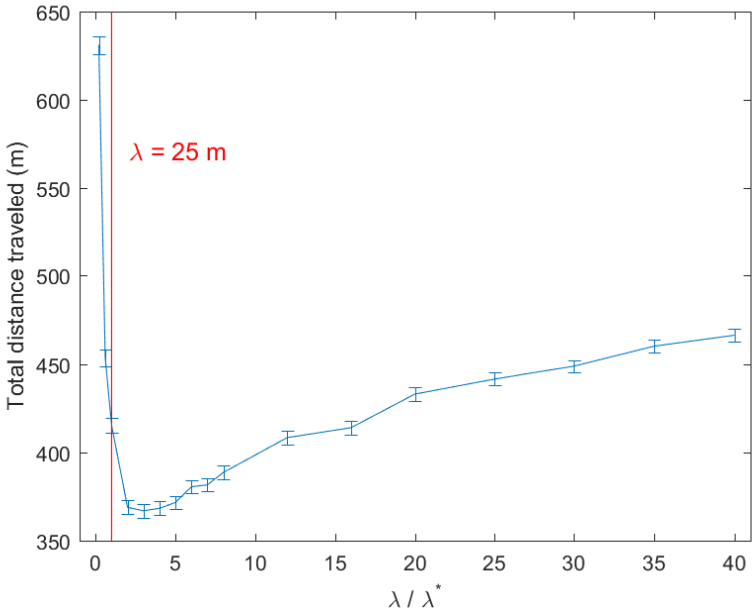

Figure A.5: Average distance traveled at $t = 1000$ and $\lambda^* = 25$ m for different values of λ/λ^* using UCL with correlation

A.3 Effect of gross overestimation of λ in 2D task

Figures A.6 and A.7 show the average cumulative regret and total distance traveled for UCB, UCL, and UCL with correlation for different length scales, with bands corresponding to confidence intervals of 95%. The means are correlated as in the simulation in Section 4.2, with $\lambda^* = 5$ m, but in this case, the overestimate of the length scale is 10^4 m, several orders of magnitude above $\lambda^* = 5$ m. Note that, for the overestimate, regret approaches linear behavior, and performance is worse than that of UCB. The distance traveled is higher than that obtained with $\lambda = \lambda^*$.

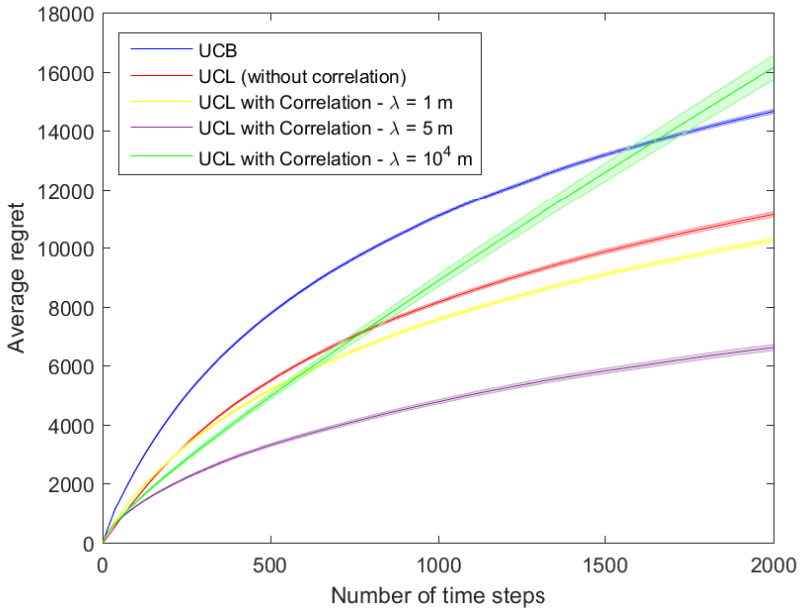

Figure A.6: Average cumulative regret for agent in the tank, $\lambda^* = 5$ m, when UCL with correlation is provided with a gross overestimate of the length scale ($\lambda >> \lambda^*$)

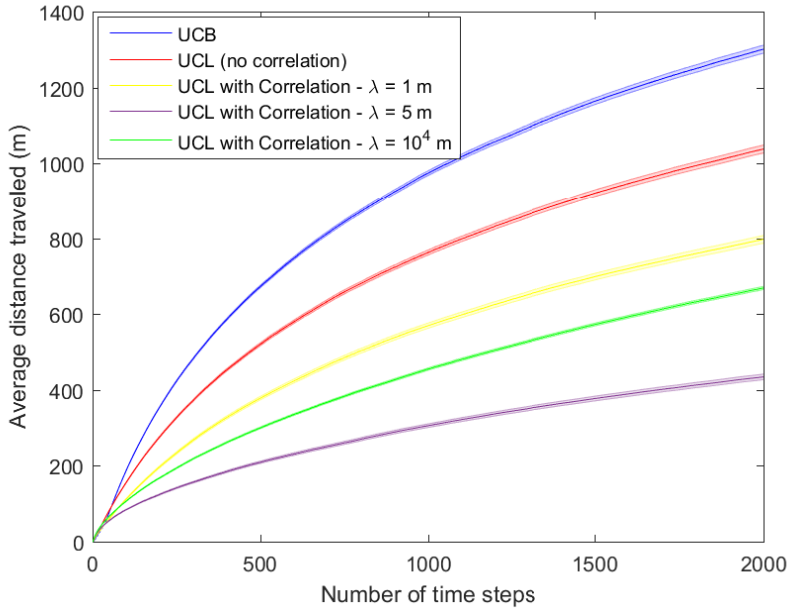

Figure A.7: Average distance traveled for agent in the tank, $\lambda^* = 5$ m, when UCL with correlation is provided with a gross overestimate of the length scale ($\lambda >> \lambda^*$)

Appendix B

MATLAB Programs

B.1 UCB

B.1.1 ucb.m

```
1  % Simulates a multi-armed bandit algorithm with a UCB for nTSteps
2  % (including an initial exploration phase with all arms).
3  % Returns the reward and the regret as a function of the number of
4  % iterations, starting at 0, as column vectors. Also returns the ...
       sequence
5  % of arms played.
6  % m is a vector representing the means of the arms, and stdDev their
7  % standard deviations.
8
9  function [reward, regret, arms] = ucb(m, stdDev, nTSteps)
10
11 [nArms, ¬] = size(m);
12 arms = zeros(nTSteps, 1);
13
14 if nTSteps < nArms + 1
15     error('The number of time steps must be greater than the ...
           number of arms')
16 end
```

```
17
18
19 reward = zeros(nTSteps + 1, 1);  % The accumulated reward at time t
20 regret = zeros(nTSteps + 1, 1);  % The accumulated regret at time t
21 n = zeros(nArms, 1);   % The number of times that each arm has ...
       been played
22 mEmp = zeros(nArms, 1);          % The empirical mean of each arm
23 maxM = max(m);                   % The highest mean of all the arms
24
25
26 % Stage 1: Explore. Sample each arm once.
27 [samplesExp, rewardExp, regretExp] = sampleAll(m, stdDev, nArms, ...
       maxM);
28 for k = 1 : nArms
29     arms(k) = k;
30 end
31 mEmp(1:nArms) = samplesExp;
32 reward(1:(nArms + 1)) = rewardExp;
33 regret(1:(nArms + 1)) = regretExp;
34
35 n = ones(nArms, 1);    % The number of times that each arm has ...
       been played
36
37 % Stage 2: play the arm with the highest q_i
38 for k = (2 + nArms) : (nTSteps + 1)
39     % Choose the arm. q_i is the highest value that arm i can have
40     q = mEmp + stdDev .* sqrt(2 * log(k - 1) ./ n);
41     [¬, arm] = max(q);
42
43     % Play the arm and update data
44     arms(k - 1) = arm;
45     sample = normrnd(m(arm), stdDev(arm));
46     mEmp(arm) = (mEmp(arm) * n(arm) + sample) / (n(arm) + 1);
47     n(arm) = n(arm) + 1;
48     reward(k) = reward(k - 1) + sample;
49     regret(k) = regret(k - 1) + maxM - m(arm);
```

```
50 end
51
52
53 end
```

B.1.2 sampleAll.m

```
1  % Samples all of the arms once. The medians are specified in m, the
2  % standard deviations in stdDev, and nArms is the number of arms. ...
       The max
3  % median is specified in maxM. Returns the samples as a column ...
       vector of
4  % size nArms. Returns the reward and regret, assuming that no ...
       arms have
5  % been played before, as a function of the number of iterations, ...
       starting
6  % at 0.
7
8  function [samples, reward, regret] = sampleAll(m, stdDev, nArms, ...
       maxM)
9
10 samples = zeros(nArms, 1);
11 regret = zeros(nArms + 1, 1);
12 reward = zeros(nArms + 1, 1);
13
14 for k = 1 : nArms
15     samples(k) = normrnd(m(k), stdDev(k));
16     reward(k + 1) = reward(k) + samples(k);
17     regret(k + 1) = regret(k) + maxM - m(k);
18 end
```

B.1.3 ucbAvg.m

```
% Runs UCB a number of times. In each trial the algorithm is
% repeated nTSteps number of times. Returns the average reward ...
    and regret
% as functions of the number of time steps from 0, and errors ...
    corresponding
% to 95% confidence intervals.
% Input means, a matrix of means to be used for each trial. Each ...
    column
% represents a mean to perform a trial with. The matrix must have ...
    nTrials
% columns.

function [rewardErr, regretErr, rewardAvg, regretAvg] =...
    ucbAvg(means, stdDev, nTSteps)

[¬, nTrials] = size(means);

rewardAvg = zeros(nTSteps + 1, 1);
regretAvg = zeros(nTSteps + 1, 1);

% Run ucb nTrials times with different means each time.
reward = zeros(nTSteps + 1, nTrials);
regret = zeros(nTSteps + 1, nTrials);
for k = 1 : nTrials
    [newReward, newRegret, ¬] = ucb(means(:, k), stdDev, nTSteps);
    reward(:,k) = newReward;
    regret(:,k) = newRegret;
    rewardAvg = rewardAvg + newReward;
    regretAvg = regretAvg + newRegret;
end

% Compute average.
rewardAvg = rewardAvg / nTrials;
regretAvg = regretAvg / nTrials;

```

```
% Compute error
rewardErr = confInterval(reward, rewardAvg);
regretErr = confInterval(regret, regretAvg);

end
```

B.2 UCL without Correlation

B.2.1 ucl.m

```
% Simulates a multi-armed bandit algorithm with a UCL for nTSteps
% Returns the reward and the regret as a function of the number of
% time steps, starting at 0, as column vectors, and the sequence ...
    of arms.
% m is a vector representing the means of the arms, and stdDev their
% standard deviations. The priors are in the form of miu_0, a ...
    column vector
% of means, and variance variance_0.

function [reward, regret, arms] = ucl(m, stdDev, miu_0, ...
    variance_0, nTSteps)

[nArms, ¬] = size(m);
arms = zeros(nTSteps, 1);

reward = zeros(nTSteps + 1, 1);  % The accumulated reward at time t
regret = zeros(nTSteps + 1, 1);  % The accumulated regret at time t
mEmp = zeros(nArms, 1);          % The empirical mean of each arm
maxM = max(m);                   % The highest mean of all the arms

n = zeros(nArms, 1);     % The number of times that each arm has ...
    been played

% Play the arm with the highest q_i
```

```
21 ΔSqrd = (stdDev.^2 / variance_0); % Δ^2 = sigma_s^2 / sigma_0^2
22 K = sqrt(2*pi*exp(1));              % Tunable parameter for ...
       inverse CDF
23
24 for k = 2 : (nTSteps + 1)
25     % Choose the arm. q_i is the highest value that arm i can have
26     miu_t = (ΔSqrd .* miu_0 + n .* mEmp) ./ (ΔSqrd + n);
27     variance_t = (stdDev.^2) ./ (ΔSqrd + n);
28     % Use the inverse cdf of the standard Gaussian random variable
29     invGaussian = norminv((1 - 1/K/(k - 1)) , 0, 1);
30     q_t = miu_t + sqrt(variance_t) .* invGaussian;
31     [¬, arm] = max(q_t);
32
33     % Play the arm and update data
34     arms(k - 1) = arm;
35     sample = normrnd(m(arm), stdDev(arm));
36
37     mEmp(arm) = (mEmp(arm) * n(arm) + sample) / (n(arm) + 1);
38     n(arm) = n(arm) + 1;
39     reward(k) = reward(k - 1) + sample;
40     regret(k) = regret(k - 1) + maxM - m(arm);
41 end
42
43
44 end
```

B.2.2 uclAvg.m

```
1 % Runs UCB a number of times. In each iteration the algorithm is
2 % repeated nTSteps number of times. Returns the average reward ...
      and regret
3 % as functions of the number of time steps from 0, and errors ...
      corresponding
```

```
4  % to 95% confidence intervals. noiseMag is the magnitude noise ...
       that is used
5  % to corrupt the actual means into the priors.
6  % Input means and priors, matrices of means and priors of these ...
       means,
7  % respectively, to be used for each trial. Each column represents ...
       a mean
8  % (or prior) to perform a trial with. The matrices must have nTrials
9  % columns.
10
11 function [rewardErr, regretErr, rewardAvg, regretAvg] =...
12     uclAvg(means, stdDev, priors, variance_0, nTSteps)
13
14 [¬, nTrials] = size(means);
15
16 rewardAvg = zeros(nTSteps + 1, 1);
17 regretAvg = zeros(nTSteps + 1, 1);
18
19 % Run ucl nTrials times, with different priors and means each time.
20 reward = zeros(nTSteps + 1, nTrials);
21 regret = zeros(nTSteps + 1, nTrials);
22 for k = 1 : nTrials
23     [newReward, newRegret, ¬] = ucl(means(:, k), stdDev, ...
           priors(:, k), ...
24                                     variance_0, nTSteps);
25     reward(:,k) = newReward;
26     regret(:,k) = newRegret;
27     rewardAvg = rewardAvg + newReward;
28     regretAvg = regretAvg + newRegret;
29 end
30
31 % Compute average.
32 rewardAvg = rewardAvg / nTrials;
33 regretAvg = regretAvg / nTrials;
34
35 % Compute error
```

```
36 rewardErr = confInterval(reward, rewardAvg);
37 regretErr = confInterval(regret, regretAvg);
38
39 end
```

B.3 UCL with correlation

B.3.1 uclCorr.m

```
1  % Simulates a multi-armed bandit algorithm with a UCL for ...
       nTSteps, with
2  % correlated priors.
3  % Returns the reward and the regret as a function of the number of
4  % time steps, starting at 0, as column vectors. Returns the ...
       sequence of
5  % arms played.
6  % m is a vector representing the means of the arms, and stdDev their
7  % standard deviation. The priors are in the form of miu_0, a ...
       column vector
8  % of means, and correlation matrix sigma_0.
9
10 function [reward, regret, arms] = uclCorr(m, stdDev, miu_0, ...
       sigma_0, nTSteps)
11
12 [nArms, ¬] = size(m);
13 arms = zeros(nTSteps, 1);
14
15 reward = zeros(nTSteps + 1, 1);  % The accumulated reward at time t
16 regret = zeros(nTSteps + 1, 1);  % The accumulated regret at time t
17 mEmp = zeros(nArms, 1);          % The empirical mean of each arm
18 maxM = max(m);                   % The highest mean of all the arms
19
20 n = zeros(nArms, 1);     % The number of times that each arm has ...
       been played
```

```
21
22 % Play the arm with the highest q_i
23 K = sqrt(2*pi*exp(1));              % Tunable parameter for ...
       inverse CDF
24 lambda = inv(sigma_0);
25 miu = miu_0;
26 variance_t = diag(sigma_0);
27
28 for k = 2 : (nTSteps + 1)
29     % Choose the arm. q_i is the highest value that arm i can have
30     % Use the inverse cdf of the standard Gaussian random variable
31     invGaussian = norminv((1 - 1/K/(k - 1)) , 0, 1);
32     q_t = miu + sqrt(variance_t) .* invGaussian;
33     [¬, arm] = max(q_t);
34
35     % Play the arm and update data
36     arms(k - 1) = arm;
37     r = normrnd(m(arm), stdDev);
38
39     mEmp(arm) = (mEmp(arm) * n(arm) + r) / (n(arm) + 1);
40     n(arm) = n(arm) + 1;
41     reward(k) = reward(k - 1) + r;
42     regret(k) = regret(k - 1) + maxM - m(arm);
43
44     % Update belief state
45     phi = zeros(nArms, 1);
46     phi(arm, 1) = 1;
47     q = r * phi / stdDev^2 + lambda * miu;
48     lambda = phi * (phi') / stdDev^2 + lambda;
49     sigma_t = inv(lambda);
50     miu = sigma_t * q;
51
52     variance_t = diag(sigma_t);
53 end
54
55
```

```
56 end
```

B.3.2 uclCorrAvg.m

```
1  % Runs UCL with correlated priors for a number of times.
2  % In each iteration the algorithm is repeated nTSteps number of ...
       times.
3  % Returns the average reward and regret as functions of the ...
       number of time
4  % steps starting from 0, and error bars corresponding to 95% ...
       confidence
5  % intervals. noiseMag is the magnitude noise
6  % that is used to corrupt the actual means into the priors.
7  % Input means and priors, matrices of means and priors of these ...
       means,
8  % respectively, to be used for each trial. Each column represents ...
       a mean
9  % (or prior) to perform a trial with. The matrices must have nTrials
10 % columns.
11
12
13 function [rewardErr, regretErr, rewardAvg, regretAvg] =...
14     uclCorrAvg(means, stdDev, priors, sigma_0, nTSteps)
15
16 [¬, nTrials] = size(means);
17
18 rewardAvg = zeros(nTSteps + 1, 1);
19 regretAvg = zeros(nTSteps + 1, 1);
20
21 % Run ucl nTrials times, with different priors and means each time.
22 reward = zeros(nTSteps + 1, nTrials);
23 regret = zeros(nTSteps + 1, nTrials);
24 for k = 1 : nTrials
```

```
25      [newReward, newRegret, ¬] = uclCorr(means(:, k), stdDev, ...
            priors(:, k),...
26                                          sigma_0, nTSteps);
27      reward(:,k) = newReward;
28      regret(:,k) = newRegret;
29      rewardAvg = rewardAvg + newReward;
30      regretAvg = regretAvg + newRegret;
31  end
32
33  % Compute average.
34  rewardAvg = rewardAvg / nTrials;
35  regretAvg = regretAvg / nTrials;
36
37  % Compute error
38  rewardErr = confInterval(reward, rewardAvg);
39  regretErr = confInterval(regret, regretAvg);
40
41  end
```

B.4 1D Arm Study

B.4.1 corrArms1D.m

```
1  % Generates the means and covariance matrix corresponding to a 1D
2  % discretized line, with nArms arms, dScale as the length scale ...
       (assuming
3  % the distance between any two consecutive arms is 1), variance_0 ...
       is the
4  % variance of the arms (related to the confidence of the priors), and
5  % randVar the variance of the reward means generated with respect ...
       to one
6  % another. mean is the mean that the rewards will have
7
```

```
function [m, sigma_0] = corrArms1D(nArms, dScale, variance_0, ...
    randVar,...
                          mean)

dUnit = 1;   % Length between two consecutive arms

% Step 1: Generate covariance matrix
sigma_0 = zeros(nArms, nArms);
for k1 = 1 : nArms
    for k2 = 1 : nArms
        if k1 ≠ k2
            sigma_0(k1, k2) = exp(-abs(k1 - k2) * dUnit / dScale);
        else
            sigma_0(k1, k2) = 1;
        end
    end
end

% Step 2: Generate correlated means
R = normrnd(0, sqrt(randVar), nArms, 1);
Q = (chol(sigma_0))';
m = mean + Q*R; % rewards

% Step 3: Scale the correlation matrix
sigma_0 = sigma_0 * variance_0;

end
```

B.4.2 uclPriorComp.m

```
% Compares specific priors and a "master mean" for UCL with/without
% correlation with UCB, where the means are correlated using the ...
    length
% scale given in dScale.
```

```
4  % Each algorithm is ran nTrials number of times, for nTSteps time ...
       steps.
5  % Plots the average regret of all five conditions.
6
7  function uclPriorComp(nTSteps, nTrials, nArms, dScale)
8
9  MAX_STDDEV = 30;           % The maximum standard deviation
10 stdDev = MAX_STDDEV;              % Standard deviation
11 stdDevV = MAX_STDDEV * ones(nArms, 1); % Standard deviation vector
12
13 % Parameters for good/bad priors with appropriate/high confidence
14 lowNoise = 20;
15 highNoise = 50;
16 lowVariance = 400;
17 highVariance = 1e6;
18
19 % Parameters for 1D arms
20 randVar = 625;
21 mean = 75;
22
23 % Generate means and priors
24 means = zeros(nArms, nTrials);
25 goodPriorsMM = 75 * ones(nArms, nTrials);
26 goodPriors = zeros(nArms, nTrials);
27 [¬, sigma_0] = corrArms1D(nArms, dScale, lowVariance, randVar, mean);
28 for k = 1 : nTrials
29     [means(:, k), ¬] = corrArms1D(nArms, dScale, lowVariance, ...
           randVar, mean);
30     goodPriors(:, k) = means(:, k) + lowNoise * randn(nArms, 1);
31 end
32
33 % UCB
34 [¬, rErrUCB, ¬, regretUCB] = ucbAvg(means, stdDevV, nTSteps);
35
36 % UCL without correlation. Good priors and high confidence
```

```
[¬, rErrGH, ¬, regretGH] = uclAvg(means, stdDevV, goodPriors, ...
    lowVariance, ...
                              nTSteps);

% UCL without correlation. Master mean prior
[¬, rErrGHMM, ¬, regretGHMM] = uclAvg(means, stdDevV, ...
    goodPriorsMM, lowVariance, ...
                              nTSteps);

% UCL with correlation. Good priors and high confidence
[¬, rErrCOR, ¬, regretCOR] = uclCorrAvg(means, stdDev, ...
    goodPriors, sigma_0, ...
                              nTSteps);

% UCL with correlation. Master mean prior
[¬, rErrCORMM, ¬, regretCORMM] = uclCorrAvg(means, stdDev, ...
    goodPriorsMM, sigma_0, ...
                              nTSteps);

n = linspace(0, nTSteps, nTSteps + 1);

figure(1)
purple = [0.4940    0.1840    0.5560];
shadedErrorBar(n, regretUCB, rErrUCB, ...
    {'color','b','Linewidth',0.2,'DisplayName','UCB'},1);
hold on
shadedErrorBar(n, regretGH, rErrGH, ...
    {'color','r','Linewidth',0.2,'DisplayName','UCL - Specific ...
    Priors'},1);
hold on
shadedErrorBar(n, regretCOR, rErrCOR, ...
    {'color','y','Linewidth',0.2,'DisplayName','UCL with ...
    Correlation, Spec. Priors'},1);
hold on
shadedErrorBar(n, regretGHMM, rErrGHMM, ...
    {'color',purple,'Linewidth',0.2,'DisplayName','UCL - Master ...
```

```
    mean'},1);
hold on
shadedErrorBar(n, regretCORMM, rErrCORMM, ...
    {'color','g','Linewidth',0.2,'DisplayName','UCL with ...
    Correlation, Master mean'},1);
legend show;
xlabel('Number of time steps');
ylabel('Average regret');
title('Average regret vs. number of time steps for 1D correlated ...
    priors');
hold off

end
```

B.5 2D Scale Study

B.5.1 corrRewards.m

```
% Generates the means and correlation matrix corresponding to ...
    arms with
% coordinates given by the vectors x, y, and z. Uses dScale as ...
    the length
% scale, variance_0 is the variance of the arms (related to the ...
    confidence
% of the priors), and randVar the variance of the reward means ...
    generated
% with respect to one another. mean is the mean that the rewards ...
    will have.

function [m, sigma_0] = corrRewards(x, y, z, dScale, variance_0, ...
    randVar,...
                        mean)

nArms = length(x);
```

```
11
12 % Step 1: Generate covariance matrix
13 sigma_0 = zeros(nArms, nArms);
14 for k1 = 1 : nArms
15     for k2 = 1 : nArms
16         if k1 ≠ k2
17             distance = sqrt([x(k1) - x(k2)]^2 + [y(k1) - y(k2)]^2 ...
                   + ...
18                  + [z(k1) - z(k2)]^2);
19             sigma_0(k1, k2) = exp(-distance / dScale);
20         else
21             sigma_0(k1, k2) = 1;
22         end
23     end
24 end
25
26 % Step 2: Generate correlated means
27 R = normrnd(0, sqrt(randVar), nArms, 1);
28 Q = (chol(sigma_0))';
29 m = mean + Q*R; % rewards
30
31 % Step 3: Scale the correlation matrix
32 sigma_0 = sigma_0 * variance_0;
33
34 end
```

B.5.2 discretePoints.m

```
1 % Generates discrete points of the tank. The grid has an edge ...
     specified by
2 % "edge". Outputs x, y, and z, vector coordinates of each point.
3
4 function [x, y, z] = discretePoints(edge)
5
```

```
radius = 3.2; % Radius of the tank
height = 2.4; % Height of the tank
clearance = 0.5; % Minimum clearance from the edge of the tank
maxRadius = radius - clearance; % Max radius for a point

numUnits = floor(maxRadius/edge);  % Number of points along an ...
    axis radius

zCoord = 2;

x = [];
y = [];
z = [];

for k1 = -numUnits : numUnits
    for k2 = -numUnits : numUnits
        xCoord = k1 * edge;
        yCoord = k2 * edge;
        if (sqrt(xCoord^2 + yCoord^2) <= maxRadius)
            x(end + 1) = xCoord;
            y(end + 1) = yCoord;
            z(end + 1) = zCoord;
        end
    end
end

figure(1)
scatter(x, y)

end
```

B.5.3 distanceTraveled.m

```
% Calculates the total distance traveled during an MAB search, in ...
    terms of
% time

function distances = distanceTraveled(x, y, z, arms)

nTSteps = length(arms);

distances = zeros(nTSteps + 1, 1);

for k = 1 : (nTSteps - 1)
    distance = sqrt([x(arms(k)) - x(arms(k + 1))]^2 + ...
                    [y(arms(k)) - y(arms(k + 1))]^2 + ...
                    [z(arms(k)) - z(arms(k + 1))]^2);
    distances(k + 2) = distances(k + 1) + distance;
end

end
```

B.5.4 tankUcbAvg.m

```
% Extends the functionality of ucbAvg by generating the average ...
    distance
% traveled.

function [rewardErr, regretErr, rewardAvg, regretAvg, ...
    distanceAvg, distErr] =...
     tankUcbAvg(x, y, z, means, stdDev, nTSteps)

[¬, nTrials] = size(means);

rewardAvg = zeros(nTSteps + 1, 1);
regretAvg = zeros(nTSteps + 1, 1);

```

```
12 distanceAvg = zeros(nTSteps + 1, 1);
13
14 % Run ucb nTrials times with different means each time.
15 reward = zeros(nTSteps + 1, nTrials);
16 regret = zeros(nTSteps + 1, nTrials);
17 distance = zeros(nTSteps + 1, nTrials);
18 for k = 1 : nTrials
19     [newReward, newRegret, arms] = ucb(means(:, k), stdDev, nTSteps);
20     reward(:,k) = newReward;
21     regret(:,k) = newRegret;
22     rewardAvg = rewardAvg + newReward;
23     regretAvg = regretAvg + newRegret;
24
25     % Calculate distance
26     newDistance = distanceTraveled(x, y, z, arms);
27     distance(:, k) = newDistance;
28     distanceAvg = distanceAvg + newDistance;
29 end
30
31 % Compute average.
32 rewardAvg = rewardAvg / nTrials;
33 regretAvg = regretAvg / nTrials;
34 distanceAvg = distanceAvg / nTrials;
35
36 % Compute error
37 rewardErr = confInterval(reward, rewardAvg);
38 regretErr = confInterval(regret, regretAvg);
39 distErr = confInterval(distance, distanceAvg);
40
41 end
```

B.5.5 tankUclAvg.m

```
% Extends the functionality of uclAvg by generating the average ...
    distance
% traveled.

function [rewardErr, regretErr, rewardAvg, regretAvg, ...
    distanceAvg, distErr] =...
     tankUclAvg(x, y, z, means, stdDev, priors, variance_0, nTSteps)

[¬, nTrials] = size(means);

rewardAvg = zeros(nTSteps + 1, 1);
regretAvg = zeros(nTSteps + 1, 1);

distanceAvg = zeros(nTSteps + 1, 1);

% Run ucl nTrials times, with different priors and means each time.
reward = zeros(nTSteps + 1, nTrials);
regret = zeros(nTSteps + 1, nTrials);
distance = zeros(nTSteps + 1, nTrials);
for k = 1 : nTrials
    [newReward, newRegret, arms] = ucl(means(:, k), stdDev, ...
        priors(:, k), ...
                                    variance_0, nTSteps);
    reward(:,k) = newReward;
    regret(:,k) = newRegret;
    rewardAvg = rewardAvg + newReward;
    regretAvg = regretAvg + newRegret;

    % Calculate distance
    newDistance = distanceTraveled(x, y, z, arms);
    distance(:, k) = newDistance;
    distanceAvg = distanceAvg + newDistance;
end

% Compute average.
rewardAvg = rewardAvg / nTrials;
```

```
34 regretAvg = regretAvg / nTrials;
35 distanceAvg = distanceAvg / nTrials;
36
37 % Compute error
38 rewardErr = confInterval(reward, rewardAvg);
39 regretErr = confInterval(regret, regretAvg);
40 distErr = confInterval(distance, distanceAvg);
41
42 end
```

B.5.6 tankUclCorrAvg.m

```
1 % Extends the functionality of tankUclAvg by generating the ...
     average distance
2 % traveled.
3
4 function [rewardErr, regretErr, rewardAvg, regretAvg, ...
     distanceAvg, distErr] =...
5     tankUclCorrAvg(x, y, z, means, stdDev, priors, sigma_0, nTSteps)
6
7 [¬, nTrials] = size(means);
8
9 rewardAvg = zeros(nTSteps + 1, 1);
10 regretAvg = zeros(nTSteps + 1, 1);
11
12 distanceAvg = zeros(nTSteps + 1, 1);
13
14 % Run ucl nTrials times, with different priors and means each time.
15 reward = zeros(nTSteps + 1, nTrials);
16 regret = zeros(nTSteps + 1, nTrials);
17 distance = zeros(nTSteps + 1, nTrials);
18 for k = 1 : nTrials
19     [newReward, newRegret, arms] = uclCorr(means(:, k), stdDev, ...
         priors(:, k),...
```

```
                                                sigma_0, nTSteps);
    reward(:,k) = newReward;
    regret(:,k) = newRegret;
    rewardAvg = rewardAvg + newReward;
    regretAvg = regretAvg + newRegret;

    % Calculate distance
    newDistance = distanceTraveled(x, y, z, arms);
    distance(:, k) = newDistance;
    distanceAvg = distanceAvg + newDistance;
end

% Compute average.
rewardAvg = rewardAvg / nTrials;
regretAvg = regretAvg / nTrials;
distanceAvg = distanceAvg / nTrials;

% Compute error
rewardErr = confInterval(reward, rewardAvg);
regretErr = confInterval(regret, regretAvg);
distErr = confInterval(distance, distanceAvg);

end
```

B.5.7 uclSStudy2D.m

```
% Simulates robotic search in 2D in the tank for nTSteps using ...
    nTrials, for
% UCL with correlation using three length scale priors: 1, 5, and ...
    25. Uses
% dScale as the length scale of the field.

function uclSStudy2D(nTSteps, nTrials, dScale)

```

```
MAX_STDDEV = 30;          % The maximum standard deviation
stdDev = MAX_STDDEV;              % Standard deviation

% Parameters for good/bad priors with low/high confidence
lowNoise = 20;
highNoise = 50;
lowVariance = 400;
highVariance = 1e6;

% Parameters for 1D arms
randVar = 625;
mean = 75;

% Scale parameters
smallD = 1;
midD = 5;
bigD = 25;


% Discretize the tank and generate the rewards
[x, y, z] = discretePoints(0.8);
nArms = length(x);
means = zeros(nArms, nTrials);
goodPriors = zeros(nArms, nTrials);
goodPriorsMM = 75 * ones(nArms, nTrials);
for k = 1 : nTrials
    [means(:, k), ¬] = corrRewards(x, y, z, dScale, lowVariance, ...
        randVar, mean);
    goodPriors(:, k) = means(:, k) + lowNoise * randn(nArms, 1);
end


stdDevV = MAX_STDDEV * ones(nArms, 1); % Standard deviation vector

% Generate covariance matrices for study
```

```
[¬, sigma0Small] = corrRewards(x, y, z, smallD, lowVariance, ...
    randVar, mean);
[¬, sigma0Mid] = corrRewards(x, y, z, midD, lowVariance, randVar, ...
    mean);
[¬, sigma0Big] = corrRewards(x, y, z, bigD, lowVariance, randVar, ...
    mean);

% UCB
[¬, rErrUCB, ¬, regretUCB, dUCB, dErrUCB] = tankUcbAvg(x, y, z, ...
    means, stdDevV, nTSteps);

% UCL without correlation. Good priors and high confidence
[¬, rErrGH, ¬, regretGH, dGH, dErrGH] = tankUclAvg(x, y, z, ...
    means, stdDevV, goodPriors, ...
                              lowVariance, nTSteps);

% UCL with correlation. Good priors and high confidence. 3 ...
    different length
% scales.
[¬, rErrCorS, ¬, regretCorS, dCorS, dErrCorS] = tankUclCorrAvg(x, ...
    y, z, means, stdDev, ...
                                    goodPriorsMM, sigma0Small, ...
                                        nTSteps);
[¬, rErrCorM, ¬, regretCorM, dCorM, dErrCorM] = tankUclCorrAvg(x, ...
    y, z, means, stdDev, ...
                                    goodPriorsMM, sigma0Mid, ...
                                        nTSteps);
[¬, rErrCorB, ¬, regretCorB, dCorB, dErrCorB] = tankUclCorrAvg(x, ...
    y, z, means, stdDev, ...
                                    goodPriorsMM, sigma0Big, ...
                                        nTSteps);

n = linspace(0, nTSteps, nTSteps + 1);

% Plot regret
figure(1)
```

```
65 purple = [0.4940    0.1840    0.5560];
66 shadedErrorBar(n, regretUCB, rErrUCB, ...
       {'color','b','Linewidth',0.2,'DisplayName','UCB'},1);
67 hold on
68 shadedErrorBar(n, regretGH, rErrGH, ...
       {'color','r','Linewidth',0.2,'DisplayName','UCL'},1);
69 hold on
70 shadedErrorBar(n, regretCorS, rErrCorS, ...
       {'color','y','Linewidth',0.2,'DisplayName','UCL with ...
       Correlation - \lambda = 1'},1);
71 hold on
72 shadedErrorBar(n, regretCorM, rErrCorM, ...
       {'color',purple,'Linewidth',0.2,'DisplayName','UCL with ...
       Correlation - \lambda = 5'},1);
73 hold on
74 shadedErrorBar(n, regretCorB, rErrCorB, ...
       {'color','g','Linewidth',0.2,'DisplayName','UCL with ...
       Correlation - \lambda = 25'},1);
75 hold on
76 legend show;
77 xlabel('Number of time steps');
78 ylabel('Average regret');
79 title('Average regret vs. number of time steps for 2D correlated ...
       priors');
80 hold off
81
82 % Plot distances
83 figure(2)
84 purple = [0.4940    0.1840    0.5560];
85 shadedErrorBar(n, dUCB, dErrUCB, ...
       {'color','b','Linewidth',0.2,'DisplayName','UCB'},1);
86 hold on
87 shadedErrorBar(n, dGH, dErrGH, ...
       {'color','r','Linewidth',0.2,'DisplayName','UCL'},1);
88 hold on
```

```
89 shadedErrorBar(n, dCorS, dErrCorS, ...
       {'color','y','Linewidth',0.2,'DisplayName','UCL with ...
       Correlation - \lambda = 1'},1);
90 hold on
91 shadedErrorBar(n, dCorM, dErrCorM, ...
       {'color',purple,'Linewidth',0.2,'DisplayName','UCL with ...
       Correlation - \lambda = 5'},1);
92 hold on
93 shadedErrorBar(n, dCorB, dErrCorB, ...
       {'color','g','Linewidth',0.2,'DisplayName','UCL with ...
       Correlation - \lambda = 25'},1);
94 hold on
95 legend show;
96 xlabel('Number of time steps');
97 ylabel('Average distance traveled (m)');
98 title('Average distance traveled vs. number of time steps for 2D ...
       correlated priors');
99 hold off
100
101 end
```

B.5.8 uclSuperSStudy2D.m

```
1  % Generates curves of regret and distance traveled vs. length ...
       scale, for
2  % nTSteps and using nTrials.
3  function uclSuperSStudy2D(nTSteps, nTrials, dScale)
4
5  MAX_STDDEV = 30;           % The maximum standard deviation
6  stdDev = MAX_STDDEV;              % Standard deviation
7
8  % Parameters for good/bad priors with low/high confidence
9  lowNoise = 20;
10 highNoise = 50;
```

```
lowVariance = 400;
highVariance = 1e6;

% Parameters for 1D arms
randVar = 625;
mean = 75;

% Scale parameters
factor = dScale / 5;


% Generate a set of length scales to try
scales = factor * [1 3 5 10 15 20 25 30 35 40 60 80 100 125 150 ...
    175 200];
nScales = length(scales);

% Regrets and distances plus their errors
regret = zeros(nScales, 1);
rErr = zeros(nScales, 1);
distance = zeros(nScales, 1);
dErr = zeros(nScales, 1);

% Discretize the tank and generate the rewards
[x, y, z] = discretePoints(0.8);
nArms = length(x);
means = zeros(nArms, nTrials);
goodPriors = zeros(nArms, nTrials);
goodPriorsMM = 75 * ones(nArms, nTrials);
for k = 1 : nTrials
    [means(:, k), ¬] = corrRewards(x, y, z, dScale, lowVariance, ...
        randVar, mean);
end

for k = 1 : nScales
    [¬, sigma0Dist] = corrRewards(x, y, z, scales(k), ...
        lowVariance, randVar, mean);
```

```
44      [¬, rErrScale, ¬, regretCorScale, dCorScale, dErrCorScale] = ...
45          tankUclCorrAvg(x, y, z, means, stdDev, goodPriorsMM, ...
                sigma0Dist, nTSteps);
46      regret(k) = regretCorScale(nTSteps + 1)
47      rErr(k) = rErrScale(nTSteps + 1)
48      distance(k) = dCorScale(nTSteps + 1)
49      dErr(k) = dErrCorScale(nTSteps + 1)
50  end
51
52  %%
53  figure(1)
54  errorbar(scales, regret, rErr)
55  title('Accumulated Regret vs. \lambda')
56  xlabel('\lambda (m)')
57  ylabel('Regret')
58
59  figure(2)
60  errorbar(scales, distance, dErr)
61  title('Distance traveled vs. \lambda')
62  xlabel('\lambda (m)')
63  ylabel('Total distance traveled (m)')
64
65  save('SuperStudy', 'regret', 'rErr', 'distance', 'dErr', 'scales');
66
67
68  end
```

B.6 Experimental Study

B.6.1 makeGrid.m

```
1  % Makes a matrix that can be used in conjunction with a color ...
       plot to
```

```
% display the distribution of the values in "values". Assumes the ...
    grid has
% sides of length "edge"

function grid = makeGrid(x, y, z, values, edge)

radius = 3.2; % Radius of the tank
clearance = 0.5; % Minimum clearance from the edge of the tank
maxRadius = radius - clearance; % Max radius for a point
numUnits = floor(maxRadius/edge); % Number of points along an ...
    axis radius

grid = zeros(2*numUnits + 1);

minValue = min(values);

valueIndex = 1;
for k1 = -numUnits : numUnits
    for k2 = -numUnits : numUnits
        xCoord = k1 * edge;
        yCoord = k2 * edge;
        if (sqrt(xCoord^2 + yCoord^2) <= maxRadius)
            grid(k2 + numUnits + 1, k1 + numUnits + 1) = ...
                values(valueIndex);
            valueIndex = valueIndex + 1;
        else
            grid(k2 + numUnits + 1, k1 + numUnits + 1) = minValue;
        end
    end
end

end
```

B.6.2 robotScaleStudy.m

```
% Runs a 2D scale study 1 time for the number of steps indicated ...
    by nTSteps
% using the length scale indicated by dScale.

function robotScaleStudy(nTSteps, dScale)

MAX_STDDEV = 30;          % The maximum standard deviation
stdDev = MAX_STDDEV;              % Standard deviation

% Parameters for good/bad priors with low/high confidence
lowNoise = 20;
highNoise = 50;
lowVariance = 400;
highVariance = 1e6;

% Parameters for means
randVar = 625;
mean = 75;

% Scale parameters
factor = 5;
midD = dScale;
smallD = midD/factor;
bigD = midD * factor;

% Tank parameters
edge = 0.8;  % Edge of the point grid
radius = 3.2; % Radius of the tank
clearance = 0.5; % Minimum clearance from the edge of the tank
maxRadius = radius - clearance; % Max radius for a point

% Discretize the tank and generate the rewards
[x, y, z] = discretePoints(edge);
nArms = length(x);
```

```
34 [means, ¬] = corrRewards(x, y, z, dScale, lowVariance, randVar, ...
       mean);
35 goodPriors = means + lowNoise * randn(nArms, 1);
36 goodPriorsMM = 75 * ones(nArms, 1);
37
38 stdDevV = MAX_STDDEV * ones(nArms, 1); % Standard deviation vector
39
40 % Generate covariance matrices for study
41 [¬, sigma0Small] = corrRewards(x, y, z, smallD, lowVariance, ...
       randVar, mean);
42 [¬, sigma0Mid] = corrRewards(x, y, z, midD, lowVariance, randVar, ...
       mean);
43 [¬, sigma0Big] = corrRewards(x, y, z, bigD, lowVariance, randVar, ...
       mean);
44
45 % UCB
46 [¬, regretUCB, armsUCB] = ucb(means, stdDevV, nTSteps);
47
48 % UCL without correlation. Good priors and high confidence
49 [¬, regretGH, armsGH] = ucl(means, stdDevV, goodPriors, ...
       lowVariance, ...
50                              nTSteps);
51
52 % UCL with correlation. Good priors and high confidence. 3 ...
       different length
53 % scales.
54 [¬, regretCorS, armsCorS] = uclCorr(means, stdDev, goodPriorsMM, ...
       sigma0Small, ...
55                                     nTSteps);
56 [¬, regretCorM, armsCorM] = uclCorr(means, stdDev, goodPriorsMM, ...
       sigma0Mid, ...
57                                     nTSteps);
58 [¬, regretCorB, armsCorB] = uclCorr(means, stdDev, goodPriorsMM, ...
       sigma0Big, ...
59                                     nTSteps);
60
```

```
% Calculate distances traveled
dUCB = distanceTraveled(x, y, z, armsUCB);
dGH = distanceTraveled(x, y, z, armsGH);
dCorS = distanceTraveled(x, y, z, armsCorS);
dCorM = distanceTraveled(x, y, z, armsCorM);
dCorB = distanceTraveled(x, y, z, armsCorB);

% Calculate times that each point was visited
visitsUCB = timesVisited(x, y, z, armsUCB);
visitsGH = timesVisited(x, y, z, armsGH);
visitsCorS = timesVisited(x, y, z, armsCorS);
visitsCorM = timesVisited(x, y, z, armsCorM);
visitsCorB = timesVisited(x, y, z, armsCorB);
gridVisUCB = makeGrid(x, y, z, visitsUCB, edge);
gridVisGH = makeGrid(x, y, z, visitsGH, edge);
gridVisCorS = makeGrid(x, y, z, visitsCorS, edge);
gridVisCorM = makeGrid(x, y, z, visitsCorM, edge);
gridVisCorB = makeGrid(x, y, z, visitsCorB, edge);

n = linspace(0, nTSteps, nTSteps + 1);

% Plot regret
figure(1)
plot(n, regretUCB, n, regretGH, n, regretCorS, n, regretCorM, n, ...
    regretCorB);
xlabel('Number of time steps');
ylabel('Regret');
title('Regret vs. number of time steps for robot search');
legend('UCB', 'UCL, Good Priors, high confidence', ...
    'UCL, \lambda = 1', 'UCL, \lambda = 5',...
    'UCL, \lambda = 25');

% Plot distance traveled
figure(2)
plot(n, dUCB, n, dGH, n, dCorS, n, dCorM, n, dCorB);
xlabel('Number of time steps');
```

```
96  ylabel('Distance traveled (m)');
97  title('Distance traveled vs. number of time steps for robot search');
98  legend('UCB', 'UCL, Good Priors, high confidence', ...
99      'UCL, \lambda = 1', 'UCL, \lambda = 5',...
100     'UCL, \lambda = 25');
101
102 % Plot resource field vs. frequency of visits per point
103 areaLims = [-maxRadius maxRadius];
104
105 figure(3)
106 % Reward
107 subplot(3, 2, 1);
108 resourceGrid = makeGrid(x, y, z, means, edge);
109 imagesc(areaLims, areaLims, resourceGrid)
110 colormap jet
111 c = colorbar;
112 set(gca,'YDir','normal')
113 c.Label.String = 'Reward';
114 title('Resource field')
115 xlabel('x(m)')
116 ylabel('y(m)')
117
118 % Visit plots
119 subplot(3, 2, 2);
120 imagesc(areaLims, areaLims, gridVisUCB)
121 colormap jet
122 c = colorbar;
123 set(gca,'YDir','normal')
124 c.Label.String = 'Number of visits';
125 title('Number of visits, UCB')
126 xlabel('x(m)')
127 ylabel('y(m)')
128
129 subplot(3, 2, 3);
130 imagesc(areaLims, areaLims, gridVisGH)
131 colormap jet
```

```
132 c = colorbar;
133 set(gca,'YDir','normal')
134 c.Label.String = 'Number of visits';
135 title('Number of visits, UCL')
136 xlabel('x(m)')
137 ylabel('y(m)')
138
139 subplot(3, 2, 4);
140 imagesc(areaLims, areaLims, gridVisCorS)
141 colormap jet
142 c = colorbar;
143 set(gca,'YDir','normal')
144 c.Label.String = 'Number of visits';
145 title('Number of visits, UCL, \lambda = 1')
146 xlabel('x(m)')
147 ylabel('y(m)')
148
149 subplot(3, 2, 5);
150 imagesc(areaLims, areaLims, gridVisCorM)
151 colormap jet
152 c = colorbar;
153 set(gca,'YDir','normal')
154 c.Label.String = 'Number of visits';
155 title('Number of visits, UCL, \lambda = 5')
156 xlabel('x(m)')
157 ylabel('y(m)')
158
159 subplot(3, 2, 6);
160 imagesc(areaLims, areaLims, gridVisCorB)
161 colormap jet
162 c = colorbar;
163 set(gca,'YDir','normal')
164 c.Label.String = 'Number of visits';
165 title('Number of visits, UCL, \lambda = 25')
166 xlabel('x(m)')
167 ylabel('y(m)')
```

```
168
169 % Export to a workspace for robot execution
170 arms = armsUCB;
171 save('UCB', 'x', 'y', 'z', 'arms', 'regretUCB', 'gridVisUCB', ...
        'dUCB');
172 arms = armsGH;
173 save('UCL', 'x', 'y', 'z', 'arms', 'regretGH', 'gridVisGH', 'dGH');
174 arms = armsCorS;
175 save('UCLCorS', 'x', 'y', 'z', 'arms', 'regretCorS', ...
        'gridVisCorS', 'dCorS');
176 arms = armsCorM;
177 save('UCLCorM', 'x', 'y', 'z', 'arms', 'regretCorM', ...
        'gridVisCorM', 'dCorM');
178 arms = armsCorB;
179 save('UCLCorB', 'x', 'y', 'z', 'arms', 'regretCorB', ...
        'gridVisCorB', 'dCorB');
180 % Save run data
181 save('RunData', 'edge', 'means', 'resourceGrid', 'dScale', ...
        'midD', 'bigD', 'smallD');
182
183
184 end
```

B.6.3 timesVisited.m

```
1 % Calculates the total number of times that each point was ...
      visited in an
2 % MAB task
3
4 function visits = timesVisited(x, y, z, arms)
5
6 nTSteps = length(arms);
7
8 nArms = length(x);
```

```
9
10 visits = zeros(nArms, 1);
11
12 for k = 1 : nTSteps
13     arm = arms(k);
14     visits(arm) = visits(arm) + 1;
15 end
16
17 end
```

B.7 Vehicle Control

B.7.1 vertStepDemo.m

```
1 % Simulates a step response using the vertical controller for the ...
    robot
2
3 d = 0.1;
4 m = 7.5;
5 b = 16.89;
6 Kp = 600;
7 Ki = 25;
8 Plant = tf([d], [m b 0]);
9 Controller = tf([Kp Ki], [1 0]);
10
11 System = Plant * Controller / (1 + Plant * Controller);
12
13 step(System)
14 ylabel('z - z(0) (m)')
15 title('')
```

B.7.2 visitor.m

```
% Runs "visit" control law, for the locations specified in filename.
% Warning: must clear all variables before running

function visitor(filename)

n_robots = 1;    % number of robots being used. Must agree with ...
    ROS code

% set up the coordinates to visit
% xCoord, yCoord, and zCoord are vectors containing the ...
    coordinates of each
% of the points
% arms: a vector indicating the sequence of points to visit

global x y z arms;
load(filename);

% set up the robot object
clear m
initial_poses(1,:) = [0 0 0 0];
runtime = 10000;
m = Belugas(initial_poses, @visit, 'direct', runtime, 'sim', false);
m.connect

%% send the control command

m.enable_control;
pause;

%% stop the vehicles

m.stop

%% shut down

```

```
34 m.shutdown
35
36 end
```

B.7.3 visit.m

```
1  function [ commands ] = visit(t, states)
2  % Visits a set of points in space. Developed by Peter Landgren, ...
       adapted by
3  % Jonathan Valverde Lizano.
4
5  %PI control with integral contol based on robot heading
6
7  % Due to the global variables being used, must remember to clear ...
       u_t before
8  % running.
9
10 % Notes: -Currently ignoring spin from top propeller
11 %        -Not calculating ErrorSum (Integral of distance to ...
       destination)
12 %        -Not keeping a history of commands
13
14 % u_t: a vector containing the times per run of control
15 % IntZValue: Integral of height with respect to destination
16 % IntRValue: Integral of horizontal distance to destination
17 % stepNumber:  Index of waypoints being reached
18 % atOuter:   Boolean vector that, for each robot, indicates ...
       whether the
19 %            previous iteration of the control was in the outer ...
       part of the
20 %            circle around the destination
21 global u_t IntZValue IntRValue stepNumber atOuter;
22
23 % x, y, and z are vectors containing the coordinates of each
```

```
24 % of the points
25 % arms: arms(s, r) is the index of the sth arm that is played by ...
       the rth
26 % robot
27 global x y z arms;
28
29 % Has the simulation finished?
30 global isFinished;
31
32 % Logs of run
33 global posLog tLog tInit tFinal;
34
35 n_robots = size(states, 1);
36 commands = zeros(n_robots,3);
37
38 %Control Gains
39 k_d_alpha = 100; %Set alpha porportional gain value
40 k_d_r = 100; %Set radius proportional gain value
41 k_d_z = 600; %Set vertical propostional gain value
42 k_i_r = 10; %Set radius integral gain value
43 k_i_z = 25; %Set height (z) integral gain value
44
45 % Radius to switch to inner controller
46 R_inner = .75;
47
48 % Max inputs
49 u_horizontal_max = 256;
50 u_vertical_max = 256;
51 omega_max = 1.0;
52
53 % Have all the robots reached their waypoints?
54 allDestReached = true;
55 rDest = 0.15; % Radius at which a robot has reached its destination
56
57 if numel(u_t) == 0
58     stepNumber = 1;
```

```
59 end
60
61 for robot = 1 : n_robots
62     % Check for state estimate errors
63     if isequal(size(states),[0,0])
64         commands(robot,:) = [0 0 0];
65         disp('State estimator error if statement entered');
66         return;
67     end
68
69     %% ------------ STAGE 1: Determine WAYPOINT (Destination) ...
           ------------
70     % Change of waypoint
71     if numel(u_t) == 0
72         atOuter(robot) = true;
73         IntRValue(robot) = 0;
74         IntZValue(robot) = 0;
75     end
76
77     destIndex = arms(stepNumber, robot);
78
79     waypoint = [x(destIndex) y(destIndex) z(destIndex)];
80
81     %% ----------------- STAGE 2: Position Calculations ...
           ------------------
82     state = states(robot,:);
83     x_Δ = state(1) - waypoint(1);
84     y_Δ = state(2) - waypoint(2);
85     r = sqrt(x_Δ^2+y_Δ^2);
86     dz = waypoint(3)-state(3);
87
88     % Has destination been reached?
89     if r > rDest
90         allDestReached = false;
91     end
92
```

```
93      %Calculating alpha
94      a = atan(x_Δ / y_Δ);
95      if y_Δ ≥0
96          alpha = pi/2 + a + state(6);
97          if abs(alpha) > pi
98              alpha = alpha - 2*pi;
99          end
100     elseif y_Δ < 0
101         alpha = -pi/2 + a + state(6);
102         if abs(alpha) > pi
103             alpha = alpha + 2*pi;
104         end
105     else
106         disp('Error: setting alpha, no half entered');
107         alpha = 0;
108     end
109     if abs(alpha)>pi
110         disp('Error: alpha outside anticipated range');
111     end
112
113     %% ---------------- STAGE 3: Calculate Vertical input ...
            ----------------
114
115     % Calculate input needed to correct for height
116     u_vertical_proportional = k_d_z * dz;
117     if numel(u_t) ≠ 0
118         % Calculate z integral
119         IntZValue(robot) = IntZValue(robot) + (t-u_t(end))*dz;
120     end
121
122     u_vertical_integral = k_i_z * IntZValue(robot);
123     u_vertical = u_vertical_proportional + u_vertical_integral;
124
125     % Check if input exceeds max
126     if abs(u_vertical) > u_vertical_max
127         u_vertical = u_vertical_max * sign(u_vertical);
```

```
128     end
129
130     %% -------- STAGE 4: Calculate Inputs for Horizontal Position ...
            --------
131
132     % ------------- Outer Control -------------
133
134     if r ≥ R_inner
135         atOuter(robot) = true;
136
137         u_tangential_alpha = k_d_alpha * alpha; %Set u_tangential
138         u_tangential = u_tangential_alpha;
139         if abs(alpha) ≤ pi/2;
140             u_parallel_proportional = k_d_r * r * cos(alpha);
141         else
142             u_parallel_proportional = 0;
143         end
144         u_parallel = u_parallel_proportional;
145
146         u_final = sqrt(u_parallel^2+u_tangential^2);
147         if u_parallel ≠ 0;
148             omega = atan(u_tangential / u_parallel);
149         else
150             omega = pi/2 * sign(u_tangential);
151         end
152
153     % ------------- Inner Control -------------
154     else
155         % In case the robot leaves the inner radius
156         if (atOuter(robot))
157             IntRValue(robot) = 0;
158         end
159         atOuter(robot) = false;
160
161         alpha_inner = alpha;
162         if abs(alpha_inner) > pi/2
```

```
163             alpha_inner = ...
                    (pi/2-(abs(alpha_inner)-pi/2))*sign(alpha_inner);
164         end
165
166         if numel(u_t) ~= 0 %Radius integral control
167             %Calculate radius integral control
168             IntRValue(robot) = IntRValue(robot) + ...
                    (t-u_t(end))*(r*cos(alpha));
169         end
170
171
172         u_parallel_proportional = k_d_r * r * cos(alpha);
173         u_parallel_integral = k_i_r * IntRValue(robot);
174         u_parallel = u_parallel_proportional + u_parallel_integral;
175         u_tangential_alpha = k_d_alpha*alpha_inner;
176         u_tangential = u_tangential_alpha;
177
178         u_final = sqrt(u_parallel^2+u_tangential^2) * ...
                sign(u_parallel);
179         omega = atan(u_tangential / u_parallel);
180         if abs(alpha) > pi/2
181             omega = omega * -1;
182         end
183     end
184
185     % Check if input exceeds max
186     if abs(u_final) > u_horizontal_max
187         u_final = u_horizontal_max * sign(u_final);
188         disp('Max u_horizontal exceeded');
189     end
190
191     %Normalize omega to 0-abs(omega_max)
192     omega = omega / (pi/2) * omega_max;
193
194     % Correct for small r
195     if r <= .15
```

```
196
197         u_final = u_final / .15 * r;
198     end
199
200     %% ------------------ STAGE 5: Send control command ...
            ------------------
201     command = [u_final omega u_vertical];
202     if isFinished
203         command = [0 0 0];
204     end
205     commands(robot,:) = command;
206
207     %% ------------------ STAGE 6: Update logs ...
            --------------------------
208     if stepNumber > 1
209         posLog(end + 1, :) = [state(1) state(2) state(3)];
210     end
211 end
212
213 u_t(end+1) = t;
214
215
216 %% Update logs
217 if stepNumber > 1
218     tLog(end + 1) = t;
219 end
220
221 %% ------ Change of waypoint if all robots have reached ...
       destinations ------
222 totSteps = size(arms, 1);
223 if allDestReached
224     if stepNumber == totSteps
225         tFinal = t;
226         isFinished = true;
227         disp('Execution completed');
228     else
```

```
229         if stepNumber == 1
230             tInit = t;
231         end
232         stepNumber = stepNumber + 1;
233         for robot = 1 : n_robots
234             atOuter(robot) = true;
235             IntRValue(robot) = 0;
236             IntZValue(robot) = 0;
237         end
238     end
239 end
240
241 end
```

B.8 Miscellaneous

B.8.1 confInterval.m

```
1  % Computes error bar for a random variable x
2
3  function [error] = confInterval(x, xAvg)
4
5  CONFIDENCE = 0.95;
6
7  [¬, nTrials] = size(x);
8
9  sumError = 0;
10 for k = 1 : nTrials
11     sumError = sumError + (xAvg - x(:, k)).^2 / nTrials;
12 end
13
14 error = sqrt(sumError) / sqrt(nTrials) * CONFIDENCE;
15
16 end
```

B.8.2 fieldVisualizer.m

```
% Generates fields using different length scales and visualizes them
function fieldVisualizer

% Parameters for good/bad priors with low/high confidence
lowVariance = 400;

% Parameters for means
randVar = 625;
mean = 75;

% Length scales
dTiny = 1e-3;
dMid = 0.2;
dMore = 0.8;
dHuge = 1e3;

% Tank parameters
edge = 0.1;  % Edge of the point grid
radius = 3.2; % Radius of the tank
clearance = 0.5; % Minimum clearance from the edge of the tank
maxRadius = radius - clearance; % Max radius for a point

% Discretize the tank and generate the rewards
[x, y, z] = discretePoints(edge);
[mTiny, ¬] = corrRewards(x, y, z, dTiny, lowVariance, randVar, mean);
[mMid, ¬] = corrRewards(x, y, z, dMid, lowVariance, randVar, mean);
[mMore, ¬] = corrRewards(x, y, z, dMore, lowVariance, randVar, mean);
[mHuge, ¬] = corrRewards(x, y, z, dHuge, lowVariance, randVar, mean);

% Make nice grids to plot these resource fields in
```

```
gridTiny = makeGrid(x, y, z, mTiny, edge);
gridMid = makeGrid(x, y, z, mMid, edge);
gridMore = makeGrid(x, y, z, mMore, edge);
gridHuge = makeGrid(x, y, z, mHuge, edge);

% Plot resource fields
areaLims = [-maxRadius maxRadius];

subplot(2, 2, 1);
imagesc(areaLims, areaLims, gridTiny)
colormap jet
c = colorbar;
set(gca,'YDir','normal')
c.Label.String = 'Reward';
title('Resource field, \lambda^* = 10^{-3}')
xlabel('x(m)')
ylabel('y(m)')

subplot(2, 2, 2);
imagesc(areaLims, areaLims, gridMid)
colormap jet
c = colorbar;
set(gca,'YDir','normal')
c.Label.String = 'Reward';
title('Resource field, \lambda^* = 0.2')
xlabel('x(m)')
ylabel('y(m)')


subplot(2, 2, 3);
imagesc(areaLims, areaLims, gridMore)
colormap jet
c = colorbar;
set(gca,'YDir','normal')
c.Label.String = 'Reward';
title('Resource field, \lambda^* = 0.8')
```

```
67 xlabel('x(m)')
68 ylabel('y(m)')
69
70
71 subplot(2, 2, 4);
72 imagesc(areaLims, areaLims, gridHuge)
73 colormap jet
74 c = colorbar;
75 set(gca,'YDir','normal')
76 c.Label.String = 'Reward';
77 title('Resource field, \lambda^* = 10^3')
78 xlabel('x(m)')
79 ylabel('y(m)')
80
81
82
83 end
```