LEARNING-AIDED AUTONOMOUS EXPLORATION

by

Shi Bai

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Shi Bai, Candidate

ADVISORY COMMITTEE

Brendan Englot, Chairman Date

Kishore Pochiraju Date

Philipppos Mordohai Date

Kostas Bekris Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2018
LEARNING-AIDED AUTONOMOUS EXPLORATION

ABSTRACT

We consider an autonomous exploration problem in which a range-sensing mobile robot is guided through an a priori unknown environment by a controller that chooses the robot’s next sensing action, one action at a time. The controller uses the mutual information between the robot’s sensor observations and an occupancy grid map as the metric to determine the best action. However, it is computationally expensive to estimate the mutual information for sensing actions supported by a dense, long-range sensor, over a high-resolution map, in an action space that may grow exponentially in a robot’s degrees of freedom. This poses challenges to the real-time viability of such an approach.

This thesis proposes a series of novel, real-time viable algorithms to select an informative robot sensing action, including: (a) applying Gaussian process regression to estimate mutual information over a large set of possible actions, (b) applying Bayesian optimization to actively select candidate viewpoints for evaluation, and (c) employing a deep neural network to learn offline from a large database of maps with relevant characteristics, to select the most informative action online within an unknown map. Both computational and physical indoor and outdoor mobile robot experiments have been performed to demonstrate the capabilities and limitations of these approaches.

Author: Shi Bai
Advisor: Brendan Englot
Date: May 1st, 2018
Department: Mechanical Engineering
Degree: Doctor of Philosophy
To my parents and my family.
Acknowledgments

I would like to thank my advisor Dr. Brendan Englot, for the one on one meetings every week which have been a great vehicle for discussions not only encouraging but inspiring. He has been a very smart and supportive leader who cares the best for his student. I am also impressed by the management of our lab, that people seem relaxed however very productive. I want to thank Dr. Kishore Pochiraju for the time we have been working together; I am very likely to have left the PhD program without him. I want to thank Dr. Philippos Mordohai for numerous tutorials, encouragement and advice he gave me since when I started my Master’s program at Stevens. I also want to thank Dr. Kostas Bekris for his precious time being my committee memeber as well as the valuable comments on my thesis.

I had a great time working in our lab, and I want to thank all the members of RFAL: Frankie Chen, Tixiao Shan, Jinkun Wang, John Martin, Erik Pearson, Dengwei Gao, Xiangyu Xu, Dong Cui, Sarah Bertussi and Paul Szenher. I hope it won’t be too hard for us to get together in the future. However if it is, I will miss you guys, as this is the best lab I have ever been in and I am grateful that I was part of it. I want to thank Dr. Mishah Salman for all the encouragement and it has been a great time to be his teaching assistant. Yizhe Chang, who has been my roomate for quite a few years, I am really glad to met him and I wish him all the best with his career in academia. I want to thank Chenhui Zhao, for being the most easy going guy during our lunch time and cafe time, and I wish him always surrounded by friends like us. I want to thank Jennifer Field for her help and I wish her a successful career in future as well. Last but not least, I also want to thank all the friends I met at Stevens. 春秋不长，日夜不短，望再相见。

Finally I want to thank my parents and my family that without you none of
the above may happen.
Table of Contents

Abstract ................................................................. iii

Dedication ............................................................... v

Acknowledgments ......................................................... vi

Table of Contents ......................................................... viii

List of Tables ............................................................ xii

List of Figures ........................................................... xiii

1 Introduction ............................................................. 1
   1.1 Next Best Viewpoint ............................................. 2
   1.2 Map Representations ............................................. 3
       1.2.1 Occupancy Grid Map ....................................... 3
       1.2.2 OctoMap .................................................. 4
   1.3 Motivation and Related Work .................................. 6
       1.3.1 Frontier-based Exploration ............................... 6
       1.3.2 Information-Theoretic Exploration ...................... 7
   1.4 Thesis Structure ................................................ 9
       1.4.1 Contributions ............................................. 9
# Information-Theoretic Exploration with Regression Model

2.1 Regression Models

2.1.1 Gaussian Process Regression

2.1.2 Support Vector Regression

2.2 GP Regression Exploration Algorithm

2.2.1 Algorithm Description

2.2.2 Time Complexity Analysis

2.2.3 Computational Results

2.3 Conclusion

# Information-Theoretic Exploration with Active Sampling

3.1 Active Learning

3.2 Bayesian Optimization

3.2.1 Related Work

3.3 Bayesian Optimization Exploration Algorithm

3.3.1 Algorithm Description

3.3.2 Time Complexity Analysis

3.3.3 Computational Results

3.4 Conclusion

# Exploration with Deep Learning

4.1 Related Work

4.2 Towards Supervised Learning

4.2.1 Problem Formulation

4.2.2 Label Calculation

4.2.3 Neural Networks

4.3 Computational Experiments
5 Hardware and Software Implementations for Physical Robot Experiments

5.1 Collaborations

5.2 Simulation Environment
  5.2.1 MATLAB and Python

5.3 Implementation for Ground Robot
  5.3.1 Turtlebot 2 Software Package
  5.3.2 Clearpath Jackal Software Package
  5.3.3 A Dense 3D Lidar: spin_hokuyo
  5.3.4 jackal_exploration_3d

5.4 Real Robot Experiments
  5.4.1 Indoor Environment
  5.4.2 Hoboken Pier A

6 Future Work

6.1 About Mutual Information Evaluation
  6.1.1 Reinforcement Learning

6.2 Non Myopic Approach

Bibliography
List of Tables

2.1 The results shown here are the average of 100 computational trials over the unstructured map shown in Figure 4. For the six test cases examined (in which “DA” refers to the deterministic approach, derived from Sobol samples; “GP” refers to the Gaussian process approach, and “SV” refers to the support vector approach), at top we show a comparison of the mean and standard deviation of computation time required per sensing action, and at bottom we show a comparison of the mean and standard deviation of the total number of steps taken by the robot in the course of driving its entropy to the minimum designated value. ................................. 24

3.1 The results shown here are the average of 100 trials over the maze map shown in Figure 3.3a. For the six test cases examined (in which “QMC” refers to the quasi Monte Carlo approach, derived from Sobol samples; “GP” refers to the batch GP regression approach and “BayOpt” refers to Bayesian optimization approach), at top we give a comparison of the mean and standard deviation of computation time required per sensing action, and at bottom we show a comparison of the mean and standard deviation of the total number of steps taken by the robot in the course of driving its entropy to the minimum designated value. .............................. 40
4.1 Accuracy of Different Networks ............................. 50
List of Figures

1.1 An example of autonomous mapping and exploration with a ground robot in a 2d environment. The occupancy grid map is shown in color according to height, while the red cells represent frontiers between unknown and free cells. ................................. 1

1.2 An example of an occupancy grid map. The white cells represent free space, black cells represent occupied space and grey cells represent unknown space. ................................. 4

1.3 Octomap with resolution of 0.05 meter. ................................. 5

1.4 Octomap with resolution of 0.1 meter. ................................. 5

1.5 Octomap with resolution of 0.2 meter. Different resolutions can be extracted from the same Octomap and it is up to the user which level of detail will be extracted. ................................. 6

1.6 An Example of frontiers in 2D (left) and 3D (right). Color: occupied cells in 3D colored by height, Red: frontier cells. ................................. 7
2.1 An illustration of two steps of the proposed decision-making process. In (a), the current occupancy map is used to predict the mutual information at a set of discrete sensing locations. In (b), a Gaussian process (GP) regression is performed over a local region of the space of sensing actions, using the data from (a) as training data. The darkest of the blue cells in (a) represents the current location of the robot, and the black cells represent known obstacles in the robot’s occupancy grid. The color scale from blue to red indicates increasing mutual information (MI).

2.2 Gaussian process (GP) regression using samples from the Sobol sequence. In all images, the robot’s current location is the same as depicted in Figure 1, and the black cells represent known obstacles in the robot’s occupancy grid. The color scale from blue to red indicates increasing mutual information (MI).

2.3 The “maze” map used in our experiments is shown on the left. The dimensions of the map are approximately 6 by 9 meters. An occupancy map produced by exploration with GP regression, showing the trajectory of the robot, is given on the right.

2.4 The “unstructured” map used in our experiments is shown on the left. The dimensions of the map are approximately 7 by 9 meters. An occupancy map produced by exploration with GP regression, showing the trajectory of the robot, is given on the right.

2.5 The results of 100 exploration trials randomly initialized in their respective maps, for each of four parameterizations. The mean entropy reduction is given over the number of sensing actions performed by the robot for all test cases considered.
2.6 Representative cases when Gaussian Process regression makes a more informative decision than the best sample whose information gain was explicitly evaluated. The green star represents the most informative Sobol sample, and the red star represents the action expected to be most informative from the Gaussian process regression. In cases (c) and (d) from the unstructured map, the candidate actions lie in different homotopy classes. .......................................................... 25

2.7 Comparison between GP estimation and ground truth (when GP estimation failed to capture the correct trend of the MI field). The color indicates the MI value from blue (lowest) to red (highest). ............... 26

3.1 An illustration of the proposed decision-making process. At each iteration, an existing set of sampled actions in the robot’s current field of view (pixels shown in solid colors), whose anticipated MI has been explicitly evaluated by ray-casting, is used to select a new sensing action (purple circle at bottom left). The robot’s current location is at the large red square, and the colors of the other solid pixels indicate their anticipated MI (with higher MI in red). ......................... 29

3.2 Using Bayesian optimization (BayOpt) to select a highly informative sensing action. ................................. 30

3.3 The three different environments used in our computational experiments are visualized as completed occupancy grids. A representative execution trace of the robot exploration process using Bayesian optimization is illustrated in each map, where nodes represent the sensing actions used to construct the map, and edges represent the paths traveled between them. ........................................... 33
3.4 The results of 100 simulated robot exploration trials, for each of six parameterizations, using the maps of Figure 3. The mean entropy reduction is given over the number of sensing actions performed by the robot for all test cases considered.

3.6 Representative cases when batch Gaussian Process regression makes poor-quality predictions and causes the robot to become stuck in place, from the Seattle map.

4.1 Illustrating the proposed decision-making process: a map patch of an exploration-in-progress is provided as input to the neural network, and the output is a suggested action.

4.2 Flowchart showing both training and testing of a deep neural network. The upper half demonstrates the training phase while the lower half shows testing. We note that the labels were calculated offline before the training phase.

4.3 An illustration of the action sets formulated for particular image patches used as training samples. The red circle represents the robot, and the blue dots represent the actions considered, with a single red dot representing the optimal action selected. Up to 36 actions may be considered.

4.4 A few examples of all the maps (10881) used for the purposes of training and testing in this study. Occupied map cells are black and free map cells are white.
4.5 A comparison of GoogleNet with an exhaustive information-theoretic approach over a series of exploration trials paired with different randomly-generated maps, in which a robot explores until it fails to collect new information. The y-axis data has been normalized with respect to the information gain obtained from the exhaustive approach. 51

4.6 A representative trajectory obtained from allowing a trained deep neural network to choose each subsequent sensing action in a robot’s exploration of an *a priori* unknown environment. Starting from the upper left corner of the map, the neural network recommended informative robot sensing actions until a dead-end in the map was reached. 52

4.7 Visualization of the intermediate layers of GoogleNet and AlexNet. 54

4.8 Filters, GooglNet conv1 layer. 55

4.9 Blobs after conv1 layer. 56

4.10 Filters, GooglNet conv2 layer. 57

4.11 Blobs after conv2 layer. 58

4.12 Filters, AlexNet conv1 layer. 59

4.13 Filters, AlexNet conv2 layer. 60

5.1 Framework for 2D Autonomous Exploration. 63

5.2 ROS Package *turtlebot_exploration_3d*. 64

5.3 Github title page of *turtlebot_exploration_3d*. 64

5.4 Clearpath Jackal. Equipped with Hokuyo and Velodyne Laser Range Finders. 65

5.5 Point cloud of Stevens Park. 65

5.6 Occupancy Map of Stevens Campus. 65

5.7 Hardware setup and outcome of the *spin_hokuyo* ROS package. 67
5.8 Framework for 3D Autonomous Exploration. ........................................ 68
5.9 An occupancy grid map generated in ABS Room 108, using our open
source repository for Bayesian optimization-enabled exploration with
the Turtlebot 2. .......................................................... 69
5.10 Exploration results with Turtlebot 2, in ABS 108. ............................. 70
5.11 Hoboken Pier A. .......................................................... 71
5.12 Jackal exploration experiments on Hoboken Pier A. .......................... 72
5.13 An Octomap generated by the proposed approach at Hoboken Pier A. 73

6.1 Comparison between myopic and non-myopic approaches on Clearpath
Jackal robot from Pier A in Hoboken, NJ. ........................................ 75
Chapter 1

Introduction

Figure 1.1: An example of autonomous mapping and exploration with a ground robot in a 2d environment. The occupancy grid map is shown in color according to height, while the red cells represent frontiers between unknown and free cells.

Mapping, localization and path planning are three essential components of autonomous navigation. In order for a robot to perform autonomously, it has to know at first the surrounding environment, which is often represented using a map. Though in some cases, a previously acquired map is available, there are many applications that require a robot to have the ability to build a map. Depending on the density and accuracy of the sensor being used, we may need to make inferences over the map. Localization is solving the problem of relative positioning given a map or partially known map. It is also possible to have uncertainty associated with localization results. Finally, path planning takes place when both a map and robust localization are
available. A path planner may not only generate a valid path efficiently, but also consider different criteria such as distance, safety and accuracy.

In this thesis we consider a problem involving all the aspects touched in the previous paragraph. We assume a robot wakes up in an a priori unknown environment and we have developed a series of algorithms which allow the robot to generate a map autonomously. We are aiming to make the mapping and exploration process as efficient as possible with respect to the new information gained at each step.

1.1 Next Best Viewpoint

Autonomous exploration considers a mobile robot that has no prior knowledge of the contents of its environment and must make repeated decisions about where to travel next, comprising an autonomous exploration problem [30].

Specifically, we formulate an information-theoretic exploration problem in which the long-term goal is to reduce entropy throughout the robot’s environment map, and the short-term goal is to perform the sensing action in each iteration that will maximize mutual information, along the lines of [31]. We assume the robot is equipped with a range sensor and uses an occupancy grid [32] to represent and reason about the environment.

Motivated by the recent work of [33], which proved that a controller driven by mutual information maximization attracts a robot to unexplored space, our aim is to implement a mutual information maximization approach that is amenable to real-time decision making and scalable to higher-dimensional systems. The approach of [33], although successful, requires a predictive evaluation of the mutual information achieved by performing every possible sensing action within a robot’s finely discretized action space. We hope to cut down the complexity by evaluating only a select number
of actions, and using these as training data for a supervised learning procedure that will predict information gain throughout the continuous action space.

The main challenge of finding the next best viewpoint is how to scale, as there can be a large number of candidate viewpoints.

1.2 Map Representations

There have been two grid-based mapping frameworks typically used in this context: occupancy grid [30] allow easy access to any cell, however they scale poorly, and OctoMap [1] is a compressed way of storing a map, but needs extra processing time for accessing a specific grid cell.

1.2.1 Occupancy Grid Map

Occupancy grid map represents the map with a set of random variables, which each of the random variable describes the possibility of the corresponding space being occupied [30]. A formal denotation of the probability of occupancy is usually $p(m|z, x)$, where $m$ denotes the resulting map under observations $z$ and robot state $x$. However the joint probability distribution among all cells defined in this form will have dimensionality up to $2^N$, where $N$ is the number of grid cells. This is too big to compute so the standard approach is to break down the correlation between grids, in other words assuming the occupancy of cells is independent among even a cell’s immediate neighbors. Instead, we can make the following approximation:

$$p(m|z, x) = p(m_i|z, x)$$ (1.1)

where $m_i$ denotes each grid instead of the whole map.
An example of an occupancy grid map is shown in Figure 1.2, which is constructed by a ground robot equipped with a laser ranger finder.

1.2.2 OctoMap

An OctoMap applies an octree data structure to compress an occupancy grid map, which significantly reduces the memory consumption [1]. An example of a 3D octomap taken in the same sensor configuration and same environment (as shown in Figure 1.2) can be found in Figures 1.3, 1.4 and 1.5. The maps shown in these figures are from the same octomap however with a different resolution.
Figure 1.3: Octomap with resolution of 0.05 meter.

Figure 1.4: Octomap with resolution of 0.1 meter.
1.3 Motivation and Related Work

1.3.1 Frontier-based Exploration

Frontiers

Given an a prior unmapped environment, an autonomous exploration task requires a robot to navigate to unknown areas to complete the map. So it is intuitive to drive the robot to the boundaries of the free areas, at the intersection between free and unknown areas, in other words “frontiers”. Figure 1.6a shows an example of the “frontiers”; while in 2D cases frontiers may be sufficient, it can be misleading in 3D cases.
Figure 1.6: An Example of frontiers in 2D (left) and 3D (right). Color: occupied cells in 3D colored by height, Red: frontier cells.

**Related work**

Frontier based methods provide a straightforward solution to the question of “Where to go?” [20], however it is a non-trivial problem to extract frontiers, especially in 3D cases [21]. Frontier based methods can also be applied to multi-robot exploration tasks [22], which extract multiple goals by segmenting the frontiers.

**1.3.2 Information-Theoretic Exploration**

**Map Entropy**

Though frontiers will likely drive a robot to complete a mapping and exploration task, the approach doesn’t have a metric of how much more information can be acquired from the environment. An information-theoretic method focuses on how much information can be acquired or how fast the map can be finished. First we need to introduce the metric of information for a map. We define Shannon’s entropy [15]...
over an occupancy grid map $m$ as follows:

$$H(m) = - \sum_i \sum_j p(m_{i,j}) \log p(m_{i,j})$$  \hfill (1.2)$$

where index $i$ refers to the individual grid cells of the map and index $j$ refers to the possible outcomes of the Bernoulli random variable that represents each grid cell, which is either free or occupied. Cells whose contents have never been observed are characterized as $p(m_{i,j}) = 0.5$, contributing one unit of entropy per cell. Cells whose contents are perfectly known contribute no entropy to the summation.

**Mutual Information**

We use mutual information $I(m, x_i)$ to evaluate the expected information gain with respect to a specific configuration $x_i$, defined as follows:

$$I(m, x_i) = H(m) - H(m|x_i)$$  \hfill (1.3)$$

where $H(m)$ is the current entropy of the map, and $H(m|x_i)$ is the expected entropy of the map given a new sensor observation at configuration $x_i$.

**Related work**

Among the earliest information-theoretic exploration strategies are those proposed by Whaite and Ferrie [43] and Elfes [31]. The former work proposes exploring an a priori unknown environment with the goal of minimizing entropy, and the latter work specifically proposes exploring to maximize the mutual information between sensor observations and an occupancy grid map. More recent works in information-theoretic exploration have considered the trade-off between maximizing mutual information
and managing the localization uncertainty in a robot’s simultaneous localization and mapping (SLAM) process [44], [4], [50], in addition to the selection of trajectories that maximize map accuracy [45]. Efforts to reduce the computational cost of evaluating mutual information over many possible future measurements have considered small, carefully selected sets of candidate trajectories, using a skeletonization of the known occupancy map [5] and the evaluation of information gain over a finite number of motion primitives [6], [7] or 3D viewpoints [8], or exclusively along the frontiers between known and unknown map regions [14], [47], which is effective in 2D environments. Limiting consideration to local neighborhoods of configurations permits efficient exploration by manipulators in 3D environments [9].

1.4 Thesis Structure

From chapter 2 to 4, we will describe each proposed approach in detail with computational experiments. All of the open source repositories and hardware configurations and experiments using physical robot platforms are explained in detail in chapter 5. Chapter 6 will talk about future work.

1.4.1 Contributions

The novel contributions of this thesis are as follows:

- We propose the first Gaussian Process Regression-Enabled Information-Theoretic Exploration algorithm and validate it with computational experiments. Details are given in Chapter 2 as well as in [34].

- We propose the first Bayesian Optimization-Enabled Information-Theoretic Exploration algorithm and validate it with computational experiments. Details are given in Chapter 3 as well as in [23].
• We propose the first *Deep Learning-Enabled Information-Theoretic Exploration* algorithm and validate it with computational experiments. Details are given in Chapter 4 as well as in [53].

• We have implemented the *Bayesian Optimization algorithm* of Chapter 3 on indoor and outdoor ground robots and we have published its supporting software as an open source repository for use with the Robot Operating System (ROS) [64].
Chapter 2
Information-Theoretic Exploration with Regression Model

In this chapter we propose to model the objective function of mutual information using two different regression models, Gaussian Process Regression and Support Vector Regression. The objective function here is like a black box, as we don’t know the mathematical form of the objective function and it is computationally expensive to calculate. Our goal is to pick the optimal configuration $x^*$ that maximizes the expected information gain.

$$x^* = \arg\max_{x_i \in C_{\text{action}}} I(m, x_i)$$  \hspace{1cm} (2.1)

In 2.1, $C_{\text{action}}$ represents the subset of the configuration space from which the robot’s next sensing action will be selected, typically within a short distance of the robot’s current location. We show that the efficiency of exploration can be improved by using regression models, with almost zero extra computational cost.

We first briefly introduce both Gaussian Process (GP) regression and Support Vector regression in section 2.1, then we focus on the GP regression implementation, as the frameworks for both regressions give nearly identical results.

2.1 Regression Models

2.1.1 Gaussian Process Regression

We assume a set of training data $x$ represents the candidate sensing configurations $x_i$ for which $I(m, x_i)$ has been computed. The values of $I(m, x_i)$ for all $x_i \in C_{\text{action}}$ comprise the set of training outputs $y$. Gaussian process regression [49] estimates the
output values and corresponding covariance associated with a set of test configurations \( \mathbf{x}_* \), according to Equations 2.2 and 2.3. The test configurations \( \mathbf{x}_* \) will be finely discretized, with the same resolution as the occupancy grid map.

\[
\bar{y}_* = k(\mathbf{x}_*, \mathbf{x})[k(\mathbf{x}, \mathbf{x}) + \sigma_n^2 I]^{-1} y
\]

\[
cov(y_*) = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{x})[k(\mathbf{x}, \mathbf{x}) + \sigma_n^2 I]^{-1} k(\mathbf{x}, \mathbf{x}_*)
\]

In the above equations, \( \bar{y}_* \) are the estimated values \( I(m, x_i) \) for the test data \( \mathbf{x}_* \), \( cov(y_*) \) is the covariance associated with these outputs, \( \sigma_n^2 \) is a vector of Gaussian noise variances associated with the observed outputs \( \mathbf{y} \), and \( k(\mathbf{x}, \mathbf{x}') \) is the kernel function, which gives a covariance matrix relating all pairs of inputs. The hyperparameters of the kernel function, which typically influence such characteristics as smoothness and length scales, can be trained using a preliminary set of representative training data.

### 2.1.2 Support Vector Regression

Support vector (SV) regression is an adaptation of the support vector machine used for regression rather than classification \[18\]. With the same training set \( \mathbf{x} \) of candidate configurations as training inputs and \( I(m, x_i) \) for all \( x_i \in C_{\text{action}} \) as the set of training outputs \( \mathbf{y} \), support vector regression estimates the output values associated with the test configuration \( \mathbf{x}_* \), but not their corresponding covariances, as in Gaussian process regression.

We use \( \epsilon \)-support vector regression, in which we aim to find an approximate function with deviation less that \( \epsilon \) from each output value at training time. The basic form for the hypothesis of a support vector regression estimator for test data \( \mathbf{x}_* \) is as
follows:

\[
\bar{y}_s = \sum_{i=1}^{l} (-\alpha_i + \alpha_i^*)k(x_{s_i}, x) + b
\]  \hspace{1cm} (2.4)

where \(k(\cdot)\) denotes the Matérn kernel function mapping from test inputs to features and \(\alpha_i^* - \alpha_i\) denotes the learned weight for the \(i\)-th feature in \(k(\cdot)\), from the solution to the dual problem described in [19].

### 2.2 GP Regression Exploration Algorithm

Our vehicle for capturing correlation among a discrete set of candidate actions will be Gaussian process regression [49], a method that has met with recent success in predicting outcomes within unknown regions of robot action [2] and observation [3] spaces. An example of the regression performed in a single decision-making step is illustrated in figure 2.1. The output of this regression is used to select the maximally informative sensing action from a continuous, local region of the robot’s action space.

Gaussian process regression has been applied to the problem of robot action selection and control in a variety of contexts. It has been used to solve optimal control problems [2], [10], generate paths that reduce localization uncertainty [11], and select actions that are likely to observe physical objects of interest [12]. It has also been used to aid the inspection of structures by predicting the regions of variability in greatest need of additional measurement [13]. Gaussian process regression has also been used to generate maps that support the evaluation of informative actions [6], [14]. However, to the best of our knowledge, it has not been applied to the problem of action selection for the exploration of unknown environments modeled by occupancy maps.

We define the space of mobile robot sensing actions to be the configuration
space $C \subseteq \mathbb{R}^d$, a subset of $d$-dimensional Euclidean space. We assume the robot’s range sensor provides a 360-degree field of view, and that its occupancy grid map is discretized finely enough to represent the configuration space, in addition to serving as the robot’s model of the environment. In the absence of obstacles, the robot is assumed capable of travel from any grid cell in the map to any other cell. A fundamental presumption in this formulation is that the robot’s action space is a subset of the spatial configuration space; this, along with our other assumptions, are similar to those made in [33].

![Figure 2.1](image)

(a) MI of discrete sensing actions.  
(b) MI predicted by GP regression.

Figure 2.1: An illustration of two steps of the proposed decision-making process. In (a), the current occupancy map is used to predict the mutual information at a set of discrete sensing locations. In (b), a Gaussian process (GP) regression is performed over a local region of the space of sensing actions, using the data from (a) as training data. The darkest of the blue cells in (a) represents the current location of the robot, and the black cells represent known obstacles in the robot’s occupancy grid. The color scale from blue to red indicates increasing mutual information (MI).

We adopt a Matérn kernel function for this application, given by 2.5.

$$k(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left( \frac{\sqrt{2\nu} |x - x'|}{\ell} \right)^{\nu} K_{\nu} \left( \frac{\sqrt{2\nu} |x - x'|}{\ell} \right)$$  \hspace{1cm} (2.5)

In 2.5, $\nu$ is a parameter used to vary the smoothness of the covariance, $\ell$ is a char-
acteristic length, \( \Gamma \) is the gamma function, and \( K_\nu \) is a modified Bessel function. In contrast with the squared exponential kernel function, which is more commonly used in Gaussian process regression [49], the Matérn kernel can be tuned to capture sharp variations in the estimated outputs. This has met with success in Gaussian process occupancy mapping, in which sharp and sudden transitions in occupancy probability due to obstacles are successfully modeled [14], [50]. Similarly, we anticipate sharp variations in mutual information due to the presence of obstacles, which will obstruct the visibility of some areas and permit the observation of others.

Example regressions performed over two sets of candidate training data are given in Figure 2.2. The training data \( \mathbf{x} \) is sampled from the two-dimensional Sobol sequence [16]. This pseudorandom sequence is selected to impose some regularity on the training data, as demonstrated by the field of samples shown in Subfigure (a). Subfigures (b), (c), and (e) represent the explicit evaluation of expected mutual information over several series of candidate views. Subfigures (d) and (f) represent the use of Gaussian process regression to predict the mutual information achieved by all actions within the finely discretized grid representing \( \mathcal{C}_{\text{action}} \), the continuous action space. The boundaries of \( \mathcal{C}_{\text{action}} \) are set according to the limits of the robot’s field of view at its present location.
Figure 2.2: Gaussian process (GP) regression using samples from the Sobol sequence. In all images, the robot’s current location is the same as depicted in Figure 1, and the black cells represent known obstacles in the robot’s occupancy grid. The color scale from blue to red indicates increasing mutual information (MI).
2.2.1 Algorithm Description

**Algorithm 1** AutonomousExploration($x_{\text{init}}, m_{\text{init}}, \text{InfoThreshold, } N_{\text{samples}}$)

1: $x_k \leftarrow x_{\text{init}}; \ m_k \leftarrow m_{\text{init}}; \ \text{ActionHistory} \leftarrow x_{\text{init}}$
2: for $k \in \{1, 2, ..., \text{NumIterations}\}$ do
3: \hspace{1em} ActionSet $\leftarrow \emptyset$
4: \hspace{1em} MISet $\leftarrow \emptyset$
5: \hspace{1em} for $x_i \in C_{\text{action}}(x_k, N_{\text{samples}})$ do
6: \hspace{2em} $MI \leftarrow \text{ObservationPrediction}(x_i, m_k)$
7: \hspace{2em} MISet $\leftarrow$ MISet $\cup$ MI
8: \hspace{2em} if $MI > \text{InfoThreshold}$ then
9: \hspace{3em} ActionSet $\leftarrow$ ActionSet $\cup$ $x_i$
10: \hspace{2em} end if
11: \hspace{1em} end for
12: \hspace{1em} ActionSet $\leftarrow$ GaussianProcessRegression(ActionSet, MISet, $x_k, m_k$);
13: \hspace{1em} if ActionSet $\neq \emptyset$ then
14: \hspace{2em} $x_{k+1} \leftarrow$ BestAction(ActionSet);
15: \hspace{2em} ActionHistory $\leftarrow$ ActionHistory $\cup$ $x_{k+1}$
16: \hspace{1em} else
17: \hspace{2em} $x_{k+1} \leftarrow$ ActionHistory(PreviousAction);
18: \hspace{2em} ActionHistory $\leftarrow$ ActionHistory $\setminus$ $x_k$
19: \hspace{2em} end if
20: \hspace{1em} end for
21: $m_{k+1} \leftarrow \text{MapUpdate}(x_{k+1})$

The exploration process proceeds according to Algorithm 1. On each iteration, an action set $C_{\text{action}}$ is formulated within the sensor field of view at the robot’s current location, and a designated number of sampled actions within the set is evaluated per Equation 1.3, drawn from a Sobol sequence. Actions whose mutual information surpasses a designated threshold $\text{InfoThreshold}$ are added to a set of approved candidate actions $\text{ActionSet}$. All of the mutual information data are then used to perform a Gaussian process regression to estimate the information gain of all other members of $C_{\text{action}}$ whose mutual information was not explicitly computed. Actions whose estimated mutual information exceeds the $\text{InfoThreshold}$ are also added to the set of candidate actions $\text{ActionSet}$. If at least one action is identified whose information
Algorithm 2 MI = ObservationPrediction($x_i$, $m_i$)

1: $m \leftarrow m_i$;
2: for beam$_j \in$ SensorBeams($x_i$) do
3:  IntersectCell $\leftarrow$ IntersectionDetected(beam$_j$, $m$);
4:  if IntersectCell $\neq \emptyset$ then
5:    $r_j = \text{knnsearch}(x_i, \text{IntersectCell})$;
6:  else
7:    $r_j = \text{MaxSensorRange}$;
8:  end if
9:  $m \leftarrow \text{EntropyUpdate}(x_i, r_j)$;
10: end for
11: $MI = \text{Entropy}(m_i) - \text{Entropy}(m)$;
12: return $MI$;

If the gain surpasses the threshold, the robot performs the maximally informative sensing action. However, if none of the actions evaluated surpasses the threshold, the robot takes a step backwards and considers the actions at a previous location along the route traveled, where there may have been informative candidate actions that were not yet performed. The algorithm repeats until the designated number of iterations is performed, or map entropy drops below a user-designated lower limit.

Algorithm 2 gives the specific steps required to explicitly evaluate the mutual information at a designated sample action $x_i$. This entails a ray tracing computation along each of the robot’s sensor beams, returning the ranges to the nearest obstacles intersected, if any. New entropy values are estimated in all cells that are anticipated to be intersected by a sensor beam, and the new expected map entropy is used to compute the expected mutual information after performing the designated sensing action. Algorithm 2 is a sub-routine of Algorithm 1 used to evaluate every Sobol sample from the action space whose mutual information is explicitly computed.
2.2.2 Time Complexity Analysis

When using Gaussian process regression, the computational complexity of Algorithm 1 is given in 2.6:

\[
O(N_{\text{steps}}(N_{\text{samples}}N_{\text{beams}}N_{\text{cells}} + N_{\text{samples}}^3 + N_{\text{samples}}^2 N_{\text{actions}}))
\]  

(2.6)

where \(N_{\text{steps}}\) is the total number of sensing actions taken by the robot in the course of exploration, \(N_{\text{samples}}\) is the number of designated configurations whose mutual information is explicitly evaluated, \(N_{\text{actions}}\) is the total number of actions comprising \(C_{\text{action}}\) that are estimated using Gaussian process regression, \(N_{\text{beams}}\) is the number of beams emitted by the robot’s range sensor, and \(N_{\text{cells}}\) is the worst-case number of occupancy grid cells that a beam may intersect. The term \(N_{\text{samples}}N_{\text{beams}}N_{\text{cells}}\) represents the cost of explicitly evaluating mutual information in all cells intersected by the robot’s sensor, for all designated actions \(N_{\text{samples}}\). The term \(N_{\text{samples}}^3 + N_{\text{samples}}^2 N_{\text{actions}}\) represents the cost of performing the subsequent Gaussian process regression, which requires the inversion of a matrix that is square in \(N_{\text{samples}}\), and its subsequent multiplication with cross-covariance terms that scale with \(N_{\text{actions}}\), the total number of sensing actions recovered from the “test data” of the Gaussian process regression. In practice, we have worked with \(10 \leq N_{\text{samples}} \leq 20\), \(N_{\text{actions}} \sim 300\), \(N_{\text{beams}} = 360\), and \(N_{\text{cells}} \sim 25\), and we have found that in this range, the complexity of the procedure is dominated by the first term, with the cost of the Gaussian process regression relatively minor in comparison to the cost of the mutual information computation. Hence, a much larger number of sensing actions can be evaluated approximately for a small additional cost on top of the initial evaluation of information gain over the original set of samples. Specific examples will be highlighted in the following section.

When using support vector regression, the complexity of Algorithm 1 is given
where training still takes on worst-case cubic complexity (which occurs when the upper bound on the coefficients $\alpha_i$ is large) but testing is linear in the number of candidate sensing actions. Once again, the complexity of the procedure is dominated by the $N_{\text{samples}}N_{\text{beams}}N_{\text{cells}}$ term, and the cost of the regression is minor in comparison to the cost of mutual information computation.

### 2.2.3 Computational Results

#### Experimental Setup

We explored the performance of our algorithm using two different maps: 1) a “maze” map representing an indoor environment (shown in Figure 2.3) and 2) an “unstructured” environment representing a forest-like environment (shown in Figure 2.4). In our simulations, we assume the robot is equipped with a laser scanner with a $360^\circ$ field of view and $1^\circ$ resolution. The range of the laser scanner was set to 1 meter, and all sensing actions considered were within a 0.5m range of the robot, ensuring that the next sensing action lies within the robot’s current field of view to the extent that its outcome can be reasonably predicted by a mutual information evaluation over the existing map. The exploration process was simulated using MATLAB.

We initialized the robot randomly within each map and simulated 100 instances of exploration for each of the following cases: a) choosing the best action among 10 Sobol samples (as depicted in Figure 2.2c) choosing the best action among 20 Sobol samples (as depicted in Figure 2.2e) using 10 Sobol samples as the basis for Gaussian process regression, choosing the best action from the approximately continuous action
space (as depicted in Figure 2.2d), and d) using 20 Sobol samples as the basis for Gaussian process regression, again choosing the best action from the approximately continuous action space (as depicted in Figure 2.2f). The robot is permitted to explore until its map entropy falls below a designated threshold, after which the simulation terminates. The Gaussian process regression computations were performed with the aid of the Gaussian processes for machine learning (GPML) MATLAB library [17]. The computation required for each trial was distributed across four cores of an Intel Xeon 5 3.0 GHz processor using the MATLAB Parallel Computing Toolbox, and a computer equipped with 4GB RAM.

Results

As noted in Section 2.2.2, the time consumed by Gaussian process regression across the entirely of the action space is substantially less than evaluating the mutual information across the much smaller designated set of actions drawn from a Sobol sequence. Figure 2.5 gives results showing the performance of the six problem parameterizations over the maps of Figures 2.3 and 2.4. In the maze map, Gaussian
process exploration drives down entropy faster than each respective case that chooses the most informative action from the explicitly evaluated sample set. In this case, all exploration methods nearly always select sensing actions from the same homotopy class, but choosing the approximately continuous action that is expected to be most informative, with the aid of Gaussian process regression, tends to point the robot in a more advantageous direction than the most informative Sobol sample.

In the unstructured map, all parameterizations using Gaussian process regression perform better across the board, even when less computational effort is invested in establishing a training data set. In this case, Gaussian process regression occasionally selects actions from a different homotopy class than the competing method using explicitly computed mutual information only, resulting in fundamentally different paths among the different parameterizations. The use of Gaussian process regression to select moves from the continuous space of sensing actions accumulates a more significant advantage, such that regression over 10 samples performs better than explicitly evaluating the mutual information at 20 samples. Hence, more informative outcomes are selected with substantially less computational effort. Representative
Figure 2.5: The results of 100 exploration trials randomly initialized in their respective maps, for each of four parameterizations. The mean entropy reduction is given over the number of sensing actions performed by the robot for all test cases considered.

(a) Results from the maze map of Fig. 3. (b) Results from the unstructured map of Fig. 4.

2.3 Conclusion

The result from the previous section has shown Gaussian Process Regression did improve the efficiency of exploration, however it is also worth the time to compare the estimation using GPs with the “ground truth” at each step (the ground truth is the evaluation of MI on all grid cells in the action space).
Table 2.1: The results shown here are the average of 100 computational trials over the unstructured map shown in Figure 4. For the six test cases examined (in which “DA” refers to the deterministic approach, derived from Sobol samples, “GP” refers to the Gaussian process approach, and “SV” refers to the support vector approach), at top we show a comparison of the mean and standard deviation of computation time required per sensing action, and at bottom we show a comparison of the mean and standard deviation of the total number of steps taken by the robot in the course of driving its entropy to the minimum designated value.

Thus a relatively intensive computational experiment has been performed which has evaluated all the actions per step. The number of actions in the unstructured map, for example, ranged from 800 to around 2000 per step, and as a result it is nearly 100 times slower than the trials in previous section (taking more than 4 days to finish only around 40 trials.). The plots shown in Figure 2.6 are representative cases when GP estimation failed to capture the correct trend of the MI field.

Of course if we increase the number of samples in the training set for the GP, we can achieve better results of the estimated MI field. However the major computational cost comes from the evaluation of MI, so the increase in training samples will significantly affect our real-time processing capability. There are two potential directions of future work on this topic discussed in the following two chapters.
Figure 2.6: Representative cases when Gaussian Process regression makes a more informative decision than the best sample whose information gain was explicitly evaluated. The green star represents the most informative Sobol sample, and the red star represents the action expected to be most informative from the Gaussian process regression. In cases (c) and (d) from the unstructured map, the candidate actions lie in different homotopy classes.
Figure 2.7: Comparison between GP estimation and ground truth (when GP estimation failed to capture the correct trend of the MI field). The color indicates the MI value from blue (lowest) to red (highest).
Chapter 3
Information-Theoretic Exploration with Active Sampling

In this chapter, we propose actively selecting the candidate actions whose MI will be explicitly evaluated, using Bayesian optimization [36], which is an efficient approach when a cost function is expensive to evaluate [37], [38]. The use of such techniques in robot gait optimization [39], environmental monitoring [40, 41], and rough-terrain navigation [42] have shown the method to be effective in a variety of robotics applications. Specifically, we will estimate a robot’s MI objective function using the posterior mean function of a Gaussian process (GP) [28]. An example of the proposed approach actively selecting a candidate for MI evaluation is illustrated in Figure 3.1. At each iteration of Bayesian optimization, a candidate sensing action is suggested and evaluated, then added to the pool of sensing actions used to approximate the MI objective function. The GP that estimates MI also forms the basis for the acquisition function used to select the next candidate sensing action. The most informative action in the pool will be executed.

3.1 Active Learning

The work in this chapter was inspired by active learning, which is a semi-supervised machine learning technique. The main idea is to improve on the quasi Monte Carlo sampling of candidate sensing actions. Instead of the quasi Monte Carlo (e.g. sobol samples) sampling of a limited number of actions to be evaluated, their selection may be guided through active learning [25] [28].

Let $\mathcal{C}_{\text{known}}$ denote the evaluated actions and $\mathcal{C}_{\text{unknown}}$ denotes all the actions left unevaluated; then an ideal active learning algorithm should result in $\mathcal{C}_{\text{chosen}}$ (which
is a subset of $\mathcal{C}_\text{unknown}$) which satisfies the following:

$$\mathcal{C}_{\text{chosen}} = \arg\min_{C_{\text{chosen}} \in \mathcal{C}_\text{unknown}} | MI_{\text{GP estimation}} - MI_{\text{Ground Truth}} | \quad (3.1)$$

### 3.2 Bayesian Optimization

The candidate sensing action suggested by Bayesian optimization is computed using an *acquisition function*, which can take on high values where GP regression predicts high values of the MI objective function, and also in unexplored areas where uncertainty is high. An acquisition function may focus solely on improving the value of the current solution, or it can also adopt the framework of a multi-armed bandit problem, in which a tradeoff is managed among exploration and exploitation of our model of the objective function. We will adopt such an approach, using the acquisition function of the Gaussian process upper confidence bound (GP-UCB) algorithm [48], which is given in Equation 3.2:

$$x_t = \arg\max_{x \in \mathcal{C}_{\text{action}}} \mu(x) + \beta \sigma(x) \quad (3.2)$$

where $\beta$ is the tradeoff parameter between exploration and exploitation. $\mu(x)$ and $\sigma(x)$ are the predicted mean and variance derived from Gaussian process regression. It has also been proven in [48] that cumulative regret can be bounded using an optimal choice of the $\beta$ parameter.

An example of how the acquisition function is used and updated is shown in Figure 3.2. An initial acquisition function based on 8 pseudo-random samples is shown in Figure 3.2a, and Figure 3.2b shows the acquisition function after one iteration of Bayesian optimization. Figure 3.2c shows the “ground truth” prediction of MI, which
Figure 3.1: An illustration of the proposed decision-making process. At each iteration, an existing set of sampled actions in the robot’s current field of view (pixels shown in solid colors), whose anticipated MI has been explicitly evaluated by ray-casting, is used to select a new sensing action (purple circle at bottom left). The robot’s current location is at the large red square, and the colors of the other solid pixels indicate their anticipated MI (with higher MI in red).

is exhaustively evaluated over all the possible actions in $C_{\text{action}}$ using ray-casting over the occupancy map. Figures 3.2d and 3.2e show the sensing actions suggested by Bayesian optimization in consecutive iterations, using the functions illustrated in the figures above. In Figure 3.2f, the best sensing action obtained from this approach is shown alongside the best action from the approach of [34], which applies a GP regression to 10 pseudo-random samples, and chooses the action predicted to offer the highest MI.

3.2.1 Related Work

Bayesian optimization has been applied to robotics problems in domains spanning from policy learning to perception. It has been used to monitor high-concentration areas of unknown spatially and temporally varying scalar fields [40, 41], minimize the vibrations experienced while navigating rough-terrain environments [42], optimize the speed and smoothness of a bipedal gait [39], and learn policies that reduce localization uncertainty in the presence of unknown landmarks [11]. However, to the best of our knowledge, Bayesian optimization has not been applied to the problem of action
selection for the exploration of unknown environments modeled by occupancy maps, the goal of which is rapid and complete discovery of the contents of the environment.

3.3 Bayesian Optimization Exploration Algorithm

![Figure 3.2: Using Bayesian optimization (BayOpt) to select a highly informative sensing action.](image)

3.3.1 Algorithm Description

The exploration process proceeds according to Algorithm 1. In every iteration, an action set $C_{\text{action}}$ is formulated within the sensor field of view at the robot’s current location, and a designated number of sampled actions within the set is evaluated per Equation 1.3, drawn from a Sobol sequence to ensure a low-discrepancy set of samples [16]. After the evaluation of mutual information over the selected actions, a candidate action suggested by Equation 3.2 will be evaluated and added to the set of approved candidate actions $ActionSet$. This updated set of actions will be used for choosing
Algorithm 3 *AutonomousExploration*(\(x_{\text{init}}, m_{\text{init}}, \text{InfoThreshold}, N_{\text{samples}}, N_{\text{iterations}}\))

1: \(x_k \leftarrow x_{\text{init}}; m_k \leftarrow m_{\text{init}}; \text{ActionHistory} \leftarrow x_{\text{init}};\)
2: \(\text{while ActionHistory} \neq \emptyset \text{ do}\)
3: \(\text{ActionSet} \leftarrow \emptyset;\)
4: \(\text{MISet} \leftarrow \emptyset;\)
5: \(\text{for } x_i \in C_{\text{action}}(x_k, N_{\text{samples}}) \text{ do}\)
6: \(MI \leftarrow \text{ObservationPrediction}(x_i, m_k);\)
7: \(\text{MISet} \leftarrow \text{MISet} \cup MI;\)
8: \(\text{ActionSet} \leftarrow \text{ActionSet} \cup x_i;\)
9: \(\text{end for}\)
10: \(\text{for } N_{\text{iterations}} \text{ do}\)
11: \(x_{\text{opt}} \leftarrow \text{BayesianOptimization}(\text{ActionSet}, \text{MISet});\)
12: \(MI \leftarrow \text{ObservationPrediction}(x_{\text{opt}}, m_k);\)
13: \(\text{MISet} \leftarrow \text{MISet} \cup MI;\)
14: \(\text{ActionSet} \leftarrow \text{ActionSet} \cup x_{\text{opt}};\)
15: \(\text{end for}\)
16: \(\text{if } \max(\text{ActionSet}) > \text{InfoThreshold} \text{ then}\)
17: \(x_{k+1} \leftarrow \text{BestAction}(\text{ActionSet});\)
18: \(\text{ActionHistory} \leftarrow \text{ActionHistory} \cup x_{k+1};\)
19: \(\text{else}\)
20: \(x_{k+1} \leftarrow \text{ActionHistory}(\text{PreviousAction});\)
21: \(\text{ActionHistory} \leftarrow \text{ActionHistory} \setminus x_k;\)
22: \(\text{end if}\)
23: \(m_{k+1} \leftarrow \text{MapUpdate}(x_{k+1});\)
24: \(\text{end while}\)

the next sample, and so on. After the designated number of iterations of Bayesian optimization, the most informative action will be selected from \(\text{ActionSet}\) whose mutual information has been explicitly computed. If at least one action is identified whose information gain surpasses the designated threshold (which typically has value slightly greater than zero), the robot performs the maximally informative sensing action. However, if none of the actions evaluated surpasses the threshold, the robot takes a step backwards and considers the actions at a previous location along the route traveled, where there may have been informative candidate actions that were not yet performed. The algorithm terminates when the previously taken action set
ActionHistory is empty. While the emphasis of this work is the efficient operation of a mutual information controller that selects informative sensing actions one-by-one, we note that more sophisticated global planning is possible to avoid becoming stuck in the “dead ends” that have already been explored [5], [46].

### 3.3.2 Time Complexity Analysis

The computational complexity of Algorithm 1 at every step of its while-loop is given in (3.3):

\[
O(N_{\text{samples}}N_{\text{beams}}N_{\text{cells}}) + O(N_{\text{iterations}}(N_{\text{samples}}^3 + N_{\text{samples}}^2N_{\text{actions}})),
\]

where \(N_{\text{samples}}\) is the number of designated configurations whose mutual information is explicitly evaluated, \(N_{\text{actions}}\) is the total number of actions comprising \(\mathcal{C}_{\text{action}}\) that are estimated using Gaussian process regression, \(N_{\text{beams}}\) is the number of beams emitted by the robot’s range sensor, and \(N_{\text{cells}}\) is the worst-case number of occupancy grid cells that a beam may intersect. \(N_{\text{iterations}}\) is the number of GP regressions performed, equivalent to the number of iterations in which Bayesian optimization is applied. The term \(N_{\text{samples}}N_{\text{beams}}N_{\text{cells}}\) represents the cost of explicitly evaluating mutual information in all cells intersected by the robot’s sensor, for all designated actions \(N_{\text{samples}}\). The term \(N_{\text{samples}}^3 + N_{\text{samples}}^2N_{\text{actions}}\) represents the cost of performing the subsequent Gaussian process regression, which requires the inversion of a matrix that is square in \(N_{\text{samples}}\), and its subsequent multiplication with cross-covariance terms that scale with \(N_{\text{actions}}\), the total number of sensing actions recovered from the “test data” of the Gaussian process regression. In practice, we have worked with \(10 \leq N_{\text{samples}} \leq 20\), \(N_{\text{actions}} \sim 300\), \(N_{\text{iterations}} \sim 5\), \(N_{\text{beams}} = 360\) (note that \(N_{\text{beams}}\) can often be much
Figure 3.3: The three different environments used in our computational experiments are visualized as completed occupancy grids. A representative execution trace of the robot exploration process using Bayesian optimization is illustrated in each map, where nodes represent the sensing actions used to construct the map, and edges represent the paths traveled between them.

larger, e.g. 307200 for a Kinect sensor), and $N_{cells} \sim 25$, and we have found that in this range, the complexity of the procedure is dominated by the first term of (3.3), with the cost of the Gaussian process regression relatively minor in comparison to the cost of the mutual information computation. Hence, a much larger number of sensing actions can be evaluated approximately for a small additional cost on top of the initial evaluation of information gain over the original set of samples. Specific examples will be highlighted in the following section.

3.3.3 Computational Results

Experimental Setup

In our simulation of robot exploration, we assume a mobile robot is equipped with a laser scanner with a 360° field of view and 1° resolution. We assume the scanner is noiseless, and that a grid cell of the occupancy map is determined with certainty to be either free or occupied if it is intersected by any of the sensor’s beams. The range of the laser scanner is assumed to be 1 meter, each grid cell is 0.01 meter in dimension, and
Figure 3.4: The results of 100 simulated robot exploration trials, for each of six parameterizations, using the maps of Figure 3. The mean entropy reduction is given over the number of sensing actions performed by the robot for all test cases considered.

all sensing actions considered are within 0.5 meters of the robot’s current location, ensuring that the next sensing action lies within the current field of view to the extent that its outcome can be reasonably predicted by an MI evaluation over the existing map. We also assume the robot is able to localize accurately, which is often feasible with the aid of laser scan-matching [35], [52]. GP regression computations were performed with the aid of the Gaussian processes for machine learning (GPML) MATLAB library [17]. The hyperparameters selected for the Matérn kernel, tuned to optimize marginal likelihood using a small set of representative training data, were held constant across all maps studied, as was the acquisition function parameter $\beta$ of Equation (4), which was tuned optimally following the procedure recommended in [48].

We explored the performance of our algorithm using three different maps: 1) a synthetic “maze” map representing an indoor environment (shown in Figure 3.3a); 2) the “Seattle map” from the Radish repository [51] (shown in Figure 3.3b), where any gaps in the building perimeter have been closed manually; and 3) a synthetic
“unstructured” map representing a forest-like environment (shown in Figure 3.3c). The exploration process was simulated using MATLAB.

We initialized the robot randomly within each map and simulated 100 instances of exploration for each of the following cases: a) choosing the best action among 10 or 20 Sobol samples, termed the quasi Monte Carlo (QMC) approach below, b) using 10 or 20 Sobol samples as the basis for Gaussian process regression, and choosing the best action from the approximately continuous action space (as in [34]), and c) using 8 or 16 Sobol samples to bootstrap Bayesian optimization, and choosing the best action after 2 or 4 subsequent iterations of Bayesian optimization, respectively. The robot will terminate its exploration process after taking a designated number of steps (50-250 steps depending on the specific map), unless it terminates automatically beforehand according to Algorithm 1. This designated number of steps is introduced to allow the exploration process to terminate in instances when a robot becomes stuck in a local region of the map without eliminating all of the map’s entropy. The computation required for each trial was distributed across four cores of an Intel i5 3.0 GHz processor using the MATLAB Parallel Computing Toolbox, on a computer equipped with 4GB RAM running Windows 7.

Results

Figure 3.3 gives results showing the performance of the six problem parameterizations over the maps of Figure 3.4. In the maze map, Bayesian optimization-based exploration drives down entropy to the maximum extent allowable at a faster rate than both the “batch” GP approach and the QMC approach for each number of sampled actions examined. Additionally, when relying on only 10 sampled actions to predict MI, Bayesian optimization achieves nearly the same performance as evaluating 20 actions from a predetermined QMC sample set (the “Sample 20” case in Fig. 3.4a).
In the Seattle map, batch GP regression has difficulty when applied to only 10 sampled actions, in which case the final map entropy does not reduce to the same level as the other two approaches. However, for both 10 and 20 sampled actions, Bayesian optimization reduces entropy to the maximal extent encountered, at rates comparable to all competing methods. Two representative examples of the batch GP approach encountering difficulty over this map are shown in Figure 3.6. In these cases, Bayesian optimization, in contrast, produces good suggestions of informative actions that will lead to further exploration. In the first example (Fig. 3.5a to 3.5d), batch GP regression predicts that the robot’s current location will yield the largest information gain of any local sensing action (Fig. 3.5a), causing the robot to stay in the current map region and remain “stuck” until the algorithm terminates. In the second example (Fig. 3.6a to 3.6f), the robot’s first step is shown as the red square in Figure 3.6b and the batch GP regression decision (Fig. 3.6a) is represented by the solid green circle. After the robot moves to the second location shown in Figure 3.6d, GP regression suggests a return to the previous position per the inference result shown in Figure 3.6c, thus causing the robot to loop between the two sensing actions until the algorithm terminates. In both of these examples, one iteration of Bayesian optimization (Fig. 3.5c and 3.6e) applied to the existing samples will suggest decisions similar to the MI “ground truth” (Fig. 3.6f).

Finally, in the unstructured map, all parameterizations using GP regression and Bayesian optimization perform better across the board, even when less computational effort is invested in establishing a training data set. In this case, GP regression and Bayesian optimization occasionally select actions from a different homotopy class than the competing QMC method, resulting in fundamentally different paths among the different parameterizations. The use of GP regression or Bayesian optimization to select moves from the continuous space of sensing actions accumulates a more signif-
(a) Batch GP inference indicates the robot’s current location is the most informative viewpoint.

(b) Recommended action from (a) (red square) relative to the sample set used.

(c) Bayesian optimization suggests a more informative action, leading out of the room.

(d) Recommended action from (c) (red circle) relative to the sample set used.
Figure 3.6: Representative cases when batch Gaussian Process regression makes poor-quality predictions and causes the robot to become stuck in place, from the Seattle map.
icant advantage, such that regression over 10 samples performs better than explicitly evaluating the MI at 20 samples. Hence, more informative outcomes are selected with substantially less computational effort. Representative trajectories of the robot when using Bayesian optimization are given in Figure 3.3, for each map. These trajectories represent full exploration of their respective environments, reaching the lower limit of map entropy sufficient for termination of Algorithm 1.

Finally, Table 3.1 gives the computation time required, and the number of steps taken by the robot in the exploration process, for all examples implemented over the maze map of Figure 3.3a. The computational cost of Bayesian optimization and GP regression are slightly better than the QMC approach, because the former two will drive the robot to more open areas containing fewer obstructions, which reduces the computational cost of the first term in (3.2). At 20 sampled actions, the Bayesian optimization approach begins to incur a heavier computational cost than the batch GP approach, due the additional number of GP regressions required.

3.4 Conclusion

We have proposed a novel approach to predict mutual information using Bayesian optimization, for the purpose of exploring \textit{a priori} unknown environments and producing a comprehensive occupancy map. In the examples considered, Bayesian optimization facilitates the selection of competitive informative sensing actions compared to batch GP regression, always reducing map entropy to an equivalent extent and at an equivalent rate, if not superior. The benefits of actively selecting additional sensing actions to include in an MI prediction are most evident in complex environments with narrow corridors, where significant gains can be made with marginal additional computational effort. However, if the number of active samples selected
Table 3.1: The results shown here are the average of 100 trials over the maze map shown in Figure 3.3a. For the six test cases examined (in which “QMC” refers to the quasi Monte Carlo approach, derived from Sobol samples; “GP” refers to the batch GP regression approach and “BayOpt” refers to Bayesian optimization approach), at top we give a comparison of the mean and standard deviation of computation time required per sensing action, and at bottom we show a comparison of the mean and standard deviation of the total number of steps taken by the robot in the course of driving its entropy to the minimum designated value.

with Bayesian optimization were to increase substantially, the number of explicit MI evaluations $N_{samples}$ would grow, and the procedure’s computational complexity may no longer be real-time viable. An appropriate tradeoff must be established between the marginal improvements in information gain with additional Bayesian optimization samples, and the bandwidth of the robot’s decision-making process.

A key area for future work is the adaptation of this approach to ground robots with 3D range-sensing capability, and a reduced angular field of view such that a robot’s angular orientation defines a unique sensing action. It is anticipated that the proposed Bayesian optimization approach will offer improved scalability of mutual information controllers to higher-dimensional action spaces, extracting the maximal benefit from every candidate sensing action for which MI is evaluated, and leveraging the power of Gaussian processes to interpolate among a sparse set of sampled actions. However, the suitability of the Matérn kernel function for capturing mutual
information over the SE2 configuration space must first be validated, to ensure the results obtained in the Euclidean action spaces explored in this work are extensible to action spaces of different topologies.
Chapter 4
Exploration with Deep Learning

In previous chapters we introduced two approaches which can improve the computational efficiency of information-theoretic autonomous exploration. However, these information theoretic approaches [34], [23] and [24] require a computationally intensive real-time evaluation of a set of candidate sensing actions, and this set may grow significantly when a robot has high degrees of freedom, and must map large-scale 3D environments. In this chapter, we propose and evaluate a methodology (shown in Figure 4.1) to choose sensing actions for exploration using a deep neural network, with the aim of selecting optimal or near-optimal sensing actions by utilizing the structure of the local region around the robot, thus avoiding intensive ray casting during the calculation of the expected information gain.

![Figure 4.1](image.png)

Figure 4.1: Illustrating the proposed decision-making process: a map patch of an exploration-in-progress is provided as input to the neural network, and the output is a suggested action.

4.1 Related Work

Deep neural networks have been successfully applied to challenging problems such as image recognition [70], robot manipulation [71] and control [65], [66]. Recent work on
obstacle avoidance [72] successfully trained a deep neural network which took RGB images as input and generated steering angles as output, while a robot was moving forward at constant speed. A novel approach for visual navigation proposed in [73] also took RGB images as input, and the learned model was able to recognize the cues for navigation to a target for which only the apperance of the target was exposed to the network. Another work in [68] used deep neural network to detect exit locations from building blueprints. As it is helpful to have rich features available from RGB images, fewer works on deep neural networks have focused on range sensors. One recent work in this area [67] learned to detect vehicles from LIDAR scans.

4.2 Towards Supervised Learning

4.2.1 Problem Formulation

Instead of choosing the sensing action of maximum expected information gain after a computationally expensive evaluation of all the candidate sensing actions, we train a neural network to predict the optimal sensing action using a database of maps with similar characteristics but different topology and layout. A flowchart demonstrating both the training and testing phases is shown in Figure 4.2. We train and test the neural network on locally visible portions of randomly-generated 2D maps of indoor environments, and we believe the locally visible portion has the cue to the optimal action. Every training sample of an exploration-in-progress is labeled with the sensing action of maximum expected information gain, and the trained model is tested over different map examples that are withheld from the training data set.

As illustrated in Figure 4.2, images containing a randomly-seeded robot’s locally visible portion of a randomly generated map are repeatedly sampled and set aside for either training or testing. For each image, the label is an index corresponding to
the expected optimal action that allows the robot to collect the maximum amount of new information from its sensor. We consider these locally visible, uniformly-sized regions instead of the entirety of a robot’s currently acquired map with the aim of achieving a scalable, computationally efficient approach that may be applied to environments of arbitrary size. However, this is done with the understanding that we are training an information-seeking controller rather than a global optimization method.

As a simulated robot explores each member of a set of pre-generated 2D maps, its labeling process is driven by a similar approach to that of [33] and [24] to select the anticipated optimal action at every step. More details about how the optimal actions were selected are described below in Section 4.2.2. As in [33], [34], and [23], we train the robot to select only one sensing action at a time. The implications of extending the proposed method to systems with higher-dimensional state and action spaces will be discussed in Section 4.4.

4.2.2 Label Calculation

We define the space of mobile robot sensing actions to be the configuration space $\mathcal{C} \subseteq \mathbb{R}^d$, a subset of $d$-dimensional Euclidean space. Specifically we consider two-
four-dimensional maps in which a robot is permitted to move a fixed distance with every step, with ten-degree resolution in its heading angle, resulting in up to 36 possible actions at each decision-making step. We use an image patch centered at the current location of the robot as the training input, and we use the index of the optimal action $x^*$ as the training label. We assume the robot’s range sensor provides a 360-degree field of view with a designated maximum sensing range, and that its occupancy grid map is discretized finely enough to represent the configuration space, in addition to serving as the robot’s model of the environment. In the absence of obstacles, the robot is assumed capable of travel from any grid cell in the map to any other cell.

We define Shannon’s entropy [15] over an occupancy grid map $m$ as follows:

$$H(m) = - \sum_i \sum_j p(m_{i,j}) \log p(m_{i,j}),$$  \hspace{1cm} (4.1)$$

where index $i$ refers to the individual grid cells of the map and index $j$ refers to the possible outcomes of the Bernoulli random variable that represents each grid cell, which is either free or occupied. Cells whose contents have never been observed are characterized as $p(m_{i,j}) = 0.5$, contributing one unit of entropy per cell. Cells whose contents are perfectly known contribute no entropy to the summation.

We use mutual information $I(m, x_i)$ to evaluate the expected information gain with respect to a specific configuration $x_i$, defined as follows:

$$I(m, x_i) = H(m) - H(m|x_i)$$  \hspace{1cm} (4.2)$$

where $H(m)$ is the current entropy of the map, and $H(m|x_i)$ is the expected entropy of the map given a new (predicted) sensor observation at configuration $x_i$. Our goal is to select the optimal robot configuration $x^*$ whose sensor observations will maximize
Figure 4.3: An illustration of the action sets formulated for particular image patches used as training samples. The red circle represents the robot, and the blue dots represent the actions considered, with a single red dot representing the optimal action selected. Up to 36 actions may be considered.

the expected information gain.

\[ x^* = \arg\max_{x_i \in C_{\text{action}}} I(m, x_i) \]  \hspace{1cm} (4.3)

In Equation 4.3, \( C_{\text{action}} \) represents the subset of the configuration space from which the robot’s next sensing action will be selected, which in our case is comprised of actions at a designated step distance from the robot’s current location, spanning the full range of feasible heading angles. The robot is moved to each of these prospective locations, its sensor rays are projected into the environment, and the expected information gain of each action is tallied, assuming that unobserved space is unoccupied, and rays will pass through unless a previously-observed obstacle presents an occlusion.

4.2.3 Neural Networks

Deep neural networks [54] have been successfully applied to image classification problems such as the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [70] in recent years. We have selected deep neural network image classification method-
ologies representing mature frameworks that have been extensively tested and characterized, noting that the development of a novel structure for a deep neural network is non-trivial and out of the scope of this particular study. Below is a list of the different neural networks tested with our range-sensing robot exploration dataset:

**AlexNet**

AlexNet [55] was the pioneering winner of the 2012 ILSVRC, after which deep neural networks have repeatedly won the ILSVRC competition.

**VGG**

Though VGG [56] was not the winner of 2014 ILSVRC, it is well known for the use of a much smaller filter size compared to AlexNet, although the network contains many more layers. It is designed on the premise that back-to-back layers can achieve a large effective receptive field, which reduces the total number of parameters needed.

**GoogleNet**

GoogleNet [57] was the winner of 2014 ILSVRC, which introduced the ‘Inception Module’ as a novel structure. Instead of stacking convolution and pooling layers sequentially, the proposed structure has convolution and pooling layers in parallel, which requires many fewer parameters than AlexNet.

**ResNet**

The Residual Network [58] was the winner of 2015 ILSVRC, which introduced a residual structure connecting every 3 layers. The Residual Network may have a thousand layers and can be more robust to overfitting.
Figure 4.4: A few examples of all the maps (10881) used for the purposes of training and testing in this study. Occupied map cells are black and free map cells are white.

**Locally Connected Layers**

Locally connected layers [59] allow different weights when the filter is applied at different positions to the input. It was selected with hopes of recognizing the surrounding structures and their locations. The locally connected neural network used here is similar to AlexNet, however the number of filters has been reduced.

**4.3 Computational Experiments**

**4.3.1 Training setup**

Using a “dungeon map” auto generator from [60], we generated 10,881 random 2D maps. A few typical maps can be found in the first column of Figure 4.4. These maps all have the same resolution of 640 by 480, consisting of randomly-sized rooms connected by corridors. Every map is comprised of one single connected component. The map data set was split into two parts, training and testing, with a ratio of 9:1.
for the respective sets. A robot is initialized at a random location in each map and takes the optimal action using the method described in Section 4.2.2. If the optimal action’s information gain surpasses a threshold (the threshold is slightly greater than zero), the robot performs the action and adds the sample into the training or testing data set, depending on which set the map has been assigned to. The robot will switch to a new map if there is no information gain after an action has been executed. While gathering the training samples, We kept taking the optimal action so that the robot can explore the map as much as possible. Adding some random actions may improve the completeness of the training data and could be a future direction to the proposed approach. The total number of training samples collected was approximately 185,000, which amounts to approximately 20 samples per map. The number of testing samples collected was approximately 18,000. Every sample generated was used in the respective training and testing phases for all networks.

In representing the map-derived images, usually an occupied cell will have a grayscale value of zero associated with its respective pixel, however we have added an offset of 5 to the occupied value just to avoid confusion in training the neural network. The grayscale pixel values of free and unknown grid cells used were 255 and 127 respectively.

The neural networks described above were trained using an Nvidia Titan X graphics card using caffe [63], while the labeling was performed in MATLAB on a Intel i7 4790K CPU. Labeling the training data required several days, and the training process also required several days for a single neural network. The completion of mapping and exploration tasks with respect to reducing 100% of a map’s entropy is not enforced here, as we limit our scope to learning informative decision-making that chooses one sensing action at a time. However, there are a number of suitable methods (e.g., a frontier-based approach or global planning over the existing map)
that could be used to “rescue” a robot that reaches a dead-end, which could work in concert with the proposed neural network enabled information-based controller.

4.3.2 Computational Results

Accuracy in Testing Phase over Individual Samples

We compared “top-1” and “top-5” accuracy between the different neural networks described in Section 4.2.3. Top-1 accuracy represents the percentage of instances in which the neural network chose the correct label for its testing samples (i.e., it chose the action offering the maximum expected information gain per Eq. 4.2 , and top-5 accuracy represents the percentage of instances in which, if the neural network is allowed to pick 5 actions, one of them is the top-1 action.

Table 4.1: Accuracy of Different Networks

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>AlexNet</th>
<th>VGG</th>
<th>GoogleNet</th>
<th>ResNet</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1</td>
<td>68.081</td>
<td>11.389</td>
<td>69.195</td>
<td>11.314</td>
<td>65.676</td>
</tr>
<tr>
<td>Top 5</td>
<td>97.405</td>
<td>20.449</td>
<td>97.692</td>
<td>38.892</td>
<td>96.286</td>
</tr>
<tr>
<td>Size</td>
<td>large</td>
<td>large</td>
<td>smallest</td>
<td>large</td>
<td>small</td>
</tr>
</tbody>
</table>

As shown in Table 4.1, GoogleNet has the highest accuracy in both the top-1 and top-5 cases, and also requires fewer parameters than other formulations. AlexNet came very close in accuracy compared to GoogleNet, however it requires more parameters. A locally connected neural network gave slightly less accuracy than AlexNet, but requires fewer parameters.

Both VGG and ResNet are not learning much in this application, though both of them have great success on image classification challenges. We suspect that it may be preferable to have a larger filter size (e.g., the first filter layer of AlexNet is 11, and for GoogleNet it is 7). A small filter size (e.g., most of the VGG and ResNet filters have a filter size of 3) may not capture the features well enough in our case.
Figure 4.5: A comparison of GoogleNet with an exhaustive information-theoretic approach over a series of exploration trials paired with different randomly-generated maps, in which a robot explores until it fails to collect new information. The y-axis data has been normalized with respect to the information gain obtained from the exhaustive approach.

**Application to Full Robot Exploration Sequences**

In order to evaluate the performance of a trained neural network over full robot exploration sequences, we generated several new dungeon maps for comparison between GoogleNet, the best-performing framework over our 18,000 testing samples, and an exhaustive information-theoretic approach that selects sensing actions according to Equation 4.2. The robot was initialized from a random location and repeatedly queries the neural network for five actions; all five actions are evaluated for MI and the robot takes the action offering maximum MI. The baseline approach in this scheme was a mutual information controller which exhaustively evaluates the expected information gain of all the possible actions at every decision-making step (up to 36 actions when
Figure 4.6: A representative trajectory obtained from allowing a trained deep neural network to choose each subsequent sensing action in a robot’s exploration of an *a priori* unknown environment. Starting from the upper left corner of the map, the neural network recommended informative robot sensing actions until a dead-end in the map was reached.

none are obstructed by obstacles). Note that we consider only the “next best view”, without providing a multi-step rescue action when the robot runs into a dead end during the training of our neural networks, and thus the trained neural network does not have the capability of recovering from a dead end (as mentioned above, a number of suitable heuristics can assist in these instances). Thus we will terminate an exploration sequence and switch to a new map when there is no information gain after an action has been executed, for both the neural network and the exhaustive mutual information controller, after several attempts.

We use the metric of average information gain per step (within the same map) to compare the performance between the two approaches, and the results are summarized in Figure 4.5. We compared the performance among 11 maps, all of which are *a priori* unknown at the start of each trial. The robot is allowed to move for at least 20 steps, after which the trial will terminate exploration of the current map if there is no information gained by the robot’s last action executed. If the robot continues to acquire new information, the trial is allowed to proceed until the robot encounters
an iteration in which it fails to acquire new information. All trials are shown in their entirety in the video attachment.

Despite the 97% top-5 accuracy observed from GoogleNet, it does not achieve the same level of performance as the exhaustive information-theoretic approach over full robot exploration sequences. We suspect a contributing factor is that all of the training samples were generated after taking the optimal action at a previous step, and the neural network may struggle to recover after one or more sub-optimal actions are taken by the robot in practice. If we consider that, for each decision-making step, the neural network will be able to select a top-5 action 97% of the time, then the possibility of doing this 20 steps in a row drops to only 54.38%.

Although the neural network does not outperform the exhaustive mutual information controller with respect to entropy reduction, the former is more computationally efficient than the latter. We ran the exhaustive information-theoretic approach on Intel i3 3.7Ghz and i5 3.5Ghz CPUs; the neural network approach was run on Nvidia GTX 960 and Titan GPUs, respectively. The average computation time for each decision-making step of the exploration process is summarized in Table 4.1. We note that the neural network approach, with constant computation time in test, only depends on the size of the network once trained, while the exhaustive approach may grow significantly when extended to higher dimensional state, action, and workspaces.

### 4.3.3 Network Visualization

We show plots of a few intermediate filters of GoogleNet and AlexNet as well as the output from those filters in Figure 4.7 to 4.13. The first convolution layer of GoogleNet has 64 filters of size 7 by 7, which are combined into a 56 by 56 matrix shown in Figure 4.8. The output of the first convolution layer was also combined into one matrix shown in Figure 4.9. The second convolution layer of GoogleNet has
Figure 4.7: Visualization of the intermediate layers of GoogleNet and AlexNet.

192 filters of size 3 by 3, which is shown in Figure 4.10, and its output is shown in Figure 4.11. The output of the first GoogleNet convolution layer indicates some edges (between free and occupied, and explored and unexplored space) being extracted by the neural network in Figure 4.12, and the output of the second convolution layer, shown in Figure 4.13, indicates the extraction of edges, and larger chunks of free, occupied, and unknown space.

4.4 Conclusions

We have proposed a novel approach to predict a robot’s most informative exploratory action using a deep neural network, for the purpose of exploring \textit{a priori} unknown environments with a range-sensing mobile robot. The proposed approach is more efficient in the testing phase than an information-theoretic optimization approach, with $O(1)$ computational complexity, and achieves competitive performance that has the potential to guide an autonomous mobile robot, if used in concert with other standard
Figure 4.8: Filters, GoogLeNet conv1 layer.
Figure 4.9: Blobs after conv1 layer.
Figure 4.10: Filters, GooglNet conv2 layer.
Figure 4.11: Blobs after conv2 layer.
Figure 4.12: Filters, AlexNet conv1 layer.
Figure 4.13: Filters, AlexNet conv2 layer.
heuristics that can “rescue” a robot from dead ends. Assuming that the required offline training resources are available, the proposed approach has the potential to scale to higher-dimensional robots, and 3D maps, by virtue of the ease of testing. However, the specific means for training a neural network to explore higher-dimensional action spaces and workspaces, and to perhaps consider longer sequences of exploratory actions, remain an area of interest for future work, as does supplementing our training data with flawed scenarios that reflect the occasional selection of sub-optimal actions.
Chapter 5
Hardware and Software Implementations for Physical Robot Experiments

5.1 Collaborations

The open source repositories introduced in this chapter were developed with members of the Robust Field Autonomy Lab: Tixiao Shan, Xiangyu Xu, Paul Szenher, Sarah Bertussi and Fanfei Chen. The details are listed as follows:

- MATLAB and Python simulations were developed with Fanfei Chen.
- The `turtlebot_exploration_3d` ROS package was developed with Tixiao Shan and Xiangyu Xu.
- The `spin_hokuyo` ROS package was developed mainly by Paul Szenher and Sarah Bertussi during their summer internship in 2017.
- The `jackal_exploration_3d` ROS package was developed with Tixiao Shan.

5.2 Simulation Environment

5.2.1 MATLAB and Python

We first developed a simulation framework in MATLAB for 2D robot exploration. The main purpose of this framework is to simulate robot exploration in 2D workspaces, and most of the computational experiments mentioned in the previous chapters were performed using this MATLAB framework.

The inverse sensor model was implemented using the MATLAB built-in function `polyxpoly`, which works fine but is computationally expensive. We then rewrote the
framework in Python and updated the inverse sensor model with ray tracing similar to the *raycast* function in the Octomap [1] implementation. We also tried to perform ray tracing in parallel, however the overhead cost for ray tracing is too expensive. Thus it turns out to be more efficient to perform the candidate viewpoint evaluations in parallel.

5.3 Implementation for Ground Robot

After the validation of our algorithms in simulation, we developed a few Robot Operating System (ROS) [64] software packages in ROS for real robot platforms. These ROS packages subscribe to a map topic (e.g. *octomap*), reason about the preferred robot viewpoint locations and send the desired robot goal to a navigation package.

5.3.1 Turtlebot 2 Software Package

Flowchart

![Flowchart](image)

Figure 5.1: Framework for 2D Autonomous Exploration.
We published a ROS package for turtlebot exploration in 3d environment in October, 2016. Now we have 31 Star and 14 Fork, also an average of 10-20 clones bi-weekly. The Turtlebot 2 was equipped with a Hokuyo URG04 scanning laser rangefinder used for localization, and an Orbbec Astra depth camera used for 3D mapping.
5.3.2 Clearpath Jackal Software Package

Figure 5.4: Clearpath Jackal. Equipped with Hokuyo and Velodyne Laser Range Finders.

Figure 5.5: Point cloud of Stevens Park.

Figure 5.6: Occupancy Map of Stevens Campus.
The Clearpath Jackal is a rugged ground vehicle with an embedded computer and payload capacity for a variety of sensors. With appropriate sensors and algorithms, the vehicle is capable of autonomous navigation in outdoor environments, even without a map or GPS.

**Challenges and Customization**

We wanted to build a ground robot platform capable of autonomous navigation in outdoor environments. We chose the Clearpath Jackal (shown in Figure 5.4), which is capable of mapping outdoor environments such as those shown in Figure 5.5 and 5.6 within 5-10 minutes, when driven by a human operator. The map can be stored in both point cloud and occupancy map format.

We performed a few upgrades to the platform to address some of the early challenges we identified. One of the challenges we encountered was producing a real-time onboard SLAM solution given the limited computational resources available. We tried *ethzasl icp mapper* first and it was found to be too computationally expensive for even an Intel i7 4770 microprocessor, though it does provide accurate localization and mapping results. We then tried Lidar odometry and mapping (LOAM) [69] which is very efficient, however it focuses more on high performance mapping, while localization was running at a relatively low frequency (e.g. 0.5Hz). In cases where the assumption is valid, we have assumed the terrain is flat so that we can use 2D SLAM solutions (e.g. HectorSLAM [62] or gmapping [61]). Another upgrade we performed was adding a router which enabled data transmission through a wireless network. Finally, we added a customized 3D lidar as our mapping sensor, which is described in detail in the next subsection.
5.3.3 A Dense 3D Lidar: spin_hokuyo

We built a 3d lidar based on attaching a 2d lidar to a tilting station, in order to get a dense and cost efficient range sensor than most of the 3d lidars on the market. The package has two modes, single sweep and continuous sweep. Depending on the applications, it can support collecting one dense lidar scan at a specific location or streaming lidar scans on the fly with a SLAM node providing pose information.

(a) Spin hokuyo hardware. (b) An Octomap captured by the package.

Figure 5.7: Hardware setup and outcome of the spin_hokuyo ROS package.

5.3.4 jackal_exploration_3d

We extended our turtlebot_exploration_3d ROS package, to make it work on the Clearpath Jackal. We added more sensors and extended on a few capabilities, including path planning and SLAM. Ultimately, a Velodyne VLP-16 volumetric scanning lidar was used to support localization, and our customized spin_hokuyo sensor was used for 3D mapping.
5.4 Real Robot Experiments

We performed experiments both indoors and outdoors with the two ground robots mentioned above.

We had two main test fields for the Bayesian optimization approach we proposed. The first one has been a indoor office-like environment and the second one has been outdoor forest-like environment. Both environments has relatively flat terrain which allowed us to run 2D SLAM for localization. We choose turtlebot for the indoor environment and Clearpath Jackal for the outdoor one.

5.4.1 Indoor Environment

We tested our implementation of Bayesian optimization-enabled exploration on a Turtlebot 2 equipped with a depth camera and planar laser rangefinder, and compared it to the baseline approach (introduced earlier in Chapter 2) in room 108 (shown in Figure 5.9) of ABS building at Stevens. We performed 3 trials for each approach and plotted the results of entropy reduction versus time, shown in Figure 5.10.
Figure 5.9: An occupancy grid map generated in ABS Room 108, using our open source repository for Bayesian optimization-enabled exploration with the Turtlebot 2.
As shown in Figure 5.10, the proposed Bayesian Optimization approach more rapidly eliminates occupancy map entropy than the baseline approach.
5.4.2 Hoboken Pier A

Figure 5.11: Hoboken Pier A.

Hoboken Pier A (shown in Figure 5.11) is an ideal environment for testing autonomous exploration using our Clearpath Jackal platform. The trees are evenly distributed on a flat terrain, which is fully traversable by the robot. We made the assumption that the terrain is completely flat and utilized 2D SLAM for localization, using a single planar “slice” of a Velodyne VLP-16’s scans of the surrounding trees, and the GMapping SLAM framework [61].
Figure 5.12: Jackal exploration experiments on Hoboken Pier A.
As shown in Figure 5.12, we can see that the Bayesian Optimization approach is not outperforming the baseline approach. We assume the reason is due to the limited number of candidate sensing actions being evaluated at each decision-making step, as this is a much larger environment compared to the indoor environment in ABS 108. The outcome of the Bayesian Optimization-Enabled Information-Theoretic Exploration approach is shown in Figure 5.13.
Chapter 6
Future Work

6.1 About Mutual Information Evaluation

During the calculation of mutual information, we assume that every unknown grid cell will be free if intersected by a sensor ray. This may be an accurate assumption for some cases in which environments are largely open and unobstructed, however this can also be a poor assumption in the exploration of cluttered environments.

6.1.1 Reinforcement Learning

A collaboration with Fanfei Chen, a fellow lab member of RFAL, has been looking into learning the actual amount of mutual information that a robot stands to gain at a given decision-making step of the exploration process. In this work, we are planning to train a deep neural network to learn the true amount of mutual information using the framework of reinforcement learning. A policy encoded in a neural network has been trained to predict mutual information while the exploration can be simulated in many synthetic maps. The policy can be trained very efficiently presents the potential to outperform the supervised learning approach in Chapter 4.

6.2 Non Myopic Approach

We would like to extend our approach to a non-myopic solution of autonomous exploration, in which a robot plans to collect more than one viewpoint. We have obtained some preliminary results (shown in Figure 6.1) of a multi-observation approach which segments the robot’s action space and connects selected viewpoints from different seg-
ments into a path.

Figure 6.1: Comparison between myopic and non-myopic approaches on Clearpath Jackal robot from Pier A in Hoboken, NJ.

In this non myopic setup, we segment the action space into cells and connect the selected actions from each cell to form a single trajectory. The criteria for selecting actions from each grid here has been simply selecting the action that offers the maximum expected mutual information within the cell. At present, we connect the selected actions through connecting nearest neighbors. We can see that the non myopic setup here outperforms the baseline approach used to select one sensing action at a time in the previous chapters of this thesis, and we hope this can be a promising direction for future research.
Bibliography


