PARAMETER SELECTION IN THE DYNAMIC WINDOW APPROACH ROBOT COLLISION AVOIDANCE ALGORITHM USING BAYESIAN OPTIMIZATION

by

CHINONSO OVUEGBE, B.S

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

COMMITTEE MEMBERS:
Adel Alaeddini, Ph.D., Co-Chair
Pranav Bhounsule, Ph.D. , Co-Chair
Lyle Hood, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Mechanical Engineering
May 2020

DEDICATION

This thesis is dedicated to all members of the Timothy - Ovuegbe family, specifically my parents and uncle, whom I have learned a lot from.

ACKNOWLEDGEMENTS

Special thanks to Dr Adel Alaeddini for his constant support and guidance throughout the development of the ideas in this thesis and as a member of his research group. I would also like to thank Dr Pranav Bhounsule for introducing me to the field of Robotics and inspiring me to take on take on challenging problems to expand on my knowledge and skill set. I am thankful to Dr Lyle Hood for taking out time to serve on my committee, and providing thoughtful encouragement and advice on areas I could improve on.

Thanks to all the members of the Advanced Data and Engineering Lab as well as the Robotics and Motion Lab. Specifically, Rajitha Meka who provided me with vital resources on applying Bayesian Optimization to my work. Stanford Martinez explained the intuition behind Gaussian Process and hard coded some examples in MATLAB. Syed-Hasib Akhter, Ernesto Hernandez, Robert Brothers, Sal Escheveste and Andrew Wattereus (deceased) always kept me company, in the lab, with daily nerdy conversations and occasional banters. Finally, I would like to thank Cayla Jimenez for her excellent handling of administrative work and logistics that contributed to the success of this thesis.

May 2020

PARAMETER SELECTION IN THE DYNAMIC WINDOW APPROACH ROBOT COLLISION AVOIDANCE ALGORITHM USING BAYESIAN OPTIMIZATION

Chinonso Ovuegbe, M.S
The University of Texas at San Antonio, 2020

Supervising Professors: Adel Alaeddini, Ph.D. and Pranav Bhounsule, Ph.D.

The Dynamic Window Approach algorithm is a classical local collision avoidance algorithm used in mobile robot navigation to generate obstacle free trajectories that are feasible based on the motion dynamics of the robot. Trajectory selection is guided by a navigation function, a sum of sub functions that evaluate the Heading direction, Distance to obstacle and Forward velocity of the robot. These sub-functions are each weighted using parameter constants within the interval $\{0, 1\}$. Parameter weights play a role in determining the overall success of navigation in terms of reaching the goal point.

In this thesis, Bayesian optimization, a Machine - Learning based optimization method for expensive black box functions, is applied to select optimal parameters that yield goal reaching trajectories within a shorter time span. A navigation simulation was built in MATLAB to model the motion dynamics of a rigid point in a 2-D Cartesian space coupled with obstacle avoidance based on the Dynamic Window Approach algorithm. A surrogate model was developed based on Gaussian Process Regression using a training set of 20 initial evaluations comprising of metrics that define the navigation simulation.Furthermore, an expected improvement function is iteratively sampled over the surrogate model to yield parameter sets that minimize a cost function f^*.

Results show that the optimal parameter weights generated from the optimization process all yielded successful goal reaching navigation outcomes in simulation environments with varying number and arrangements of static obstacles. Also, minimizing navigation time penalizes the Distance to obstacle cost, as a result, resulting trajectories tend to be close to obstacles within the navigation space.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Work presented in this thesis show an application of Bayesian optimization in selecting optimal weights for the Navigation function in the Dynamic Window Approach obstacle avoidance algorithm. The research objective here is to minimize the overall travel time and inform the selection of parameters that successfully meet the navigation goal. The navigation consists of three sub functions that guide optimal trajectory selection in the Dynamic window obstacle avoidance algorithm. These three sub functions - Heading, Distance to obstacle, and velocity are multiplied by constant weights that range from $\{0, 1\}$. The magnitudes of these constants have an effect in the resulting navigation trajectories of the robot. A high weighting ratio of one constant over the other can cause an uneven bias in the sub-function which can affect the overall robot trajectory - in some cases trajectories generated fail to reach the goal location while some others take longer paths, thereby taking too long. To mitigate this problem, Bayesian Optimization (Bayes-Opt), a machine-learning based optimization for expensive to evaluate black box functions, is adopted to generate parameters that balance the corresponding weighting of the navigation sub-functions. Bayes-Opt is modeled for functions that are expensive to evaluate, therefore limited evaluations are available for such functions. Concretely, applying Bayes-Opt requires prior evaluations of the function and generating a prior probabilistic belief of the function model, then updating the model as more evaluations are available - this is known as the surrogate. As this surrogate model is updated, an acquisition function also suggests areas that yield an approximate maximum or minimum of that function. Examples of real world applications of Bayes-Opt include Hyper-parameter optimization of Neural networks, Parameter optimization in A/B testing, etc.

In our application, the function is modeled as a simulation of the navigation algorithm with static obstacles, therefore generating an evaluation entails running the simulation for a single evaluation using the sub-function weight constants as inputs, where an evaluation output is time, average minimum distance to obstacle and distance to goal location at the end of simulation. Furthermore, although there is some knowledge as to how the navigation algorithm works, there is

limited knowledge on the inner workings of how the evaluations of the navigation simulation are generated, due to stochasticity in the prior evaluations using randomly generated weight constants. As a result of the costs associated with generating an output and limited knowledge on how the evaluations are generated, the navigation algorithm can be termed an expensive black box function, and Bayes-Opt can be justifiably applied to yield constants that only generate trajectories that lead to goal point and minimize overall navigation time.

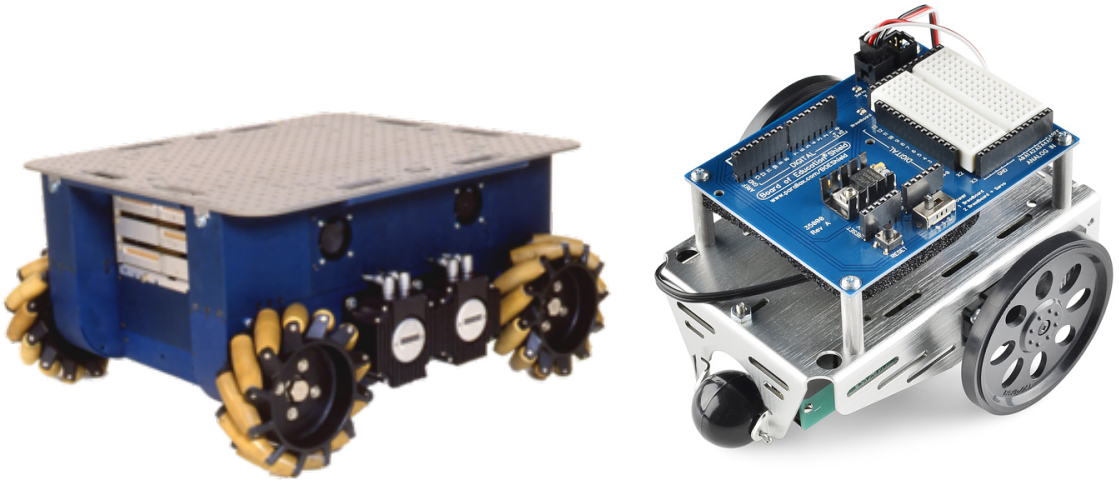
1.1 Thesis Contribution

As with most optimization problems, the weights of the cost function can effectively affect the convergence. In Neural networks, hyper-parameters play an important role in optimization convergence which is why selecting optimal hyper-parameters is a major priority in most Neural network based approaches. The basis of the work done here is the Dynamic Window Obstacle avoidance algorithm. This classical algorithm is designed to discretely generate feasible trajectories that guide a robot safely through obstacles to a goal point. The objective function in this algorithm is multiplied by a set of constants and the weights of these constants can affect the selection of trajectories thereby affecting overall navigation outcomes. In some cases, using weights that create high biases cause trajectories that do not lead to goal or take too long to reach goal point. As a result, this problem was approached from a statistical standpoint- with little knowledge of how the navigation works, if there is some available data showing what constant combinations yield good and bad navigation metrics, then an optimization objective can be modeled to yield outputs that solve an optimization problem to yield parameter constants that improve navigation. The experiments and results presented in this thesis demonstrate an application of Bayesian Optimization to select weight constants that generate trajectories that lead to the goal and prioritize much faster local routes in discrete time intervals. Work done here demonstrates, with vital results, that with limited overall data evaluations (≤ 32 observations) of the navigation function Bayesian optimization can be applied to yield parameter sets that work for multiple navigation spaces with different obstacle arrangements.

CHAPTER 2: LITERATURE REVIEW

2.1 Navigation in Mobile Robots

Mobile robots are equipped with mobile bases that enable lateral and longitudinal locomotion around surrounding environment. However, locomotion differ based on engineering design and planned dynamics of the robot. Mobile robots with holonomic mobile bases are capable of locomotion control in all degrees of freedom (i.e robots with omni-directional drives) while non-holonomic drives have limited control in all motion axes (i.e robots with differential drive) [7]. Locomotion is guided in mobile robots by controlling the velocities and positions with respect to time.



(a) Omni-directional Mobile robot (b) Differential drive Mobile robot

Figure 2.1: Holonomic and Non-Holonomic examples.

Mobile robots equipped with sensors and on-board processing units are able to navigate autonomously in static or dynamic environments. Prior to executing locomotion, the robot needs some information about its environment relating to the location of obstacles, span of navigation space, etc.

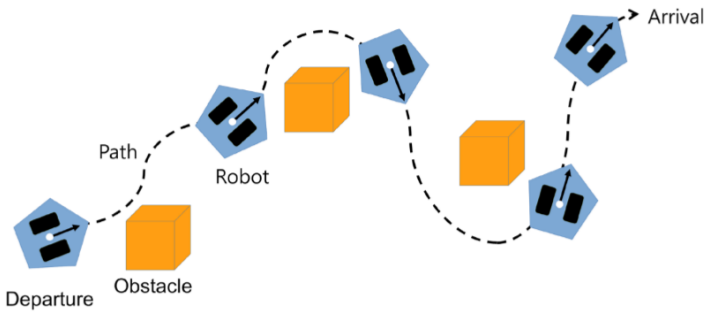


Figure 2.2: Robot Navigation.

The robot also needs to constantly know its position within the navigation space. Using a stream of data, the robot should find an optimized route amongst a set of trajectories which minimize or maximize a certain function [23]. Factoring all these, there are four major areas of mobile navigation: Mapping, Pose Estimation, Sensing, Path Planning.

2.1.1 Mapping

Mapping generates a 2D / 3D representation of the environment with estimates of landmarks within the navigation space. This map represents a *priori* known environment which is used to develop path planning strategies. Realistically, and in most cases, there is no available prior knowledge of the navigation environment. Here, a different approach is used the process of map building is incremental, where the robot is manually controlled and uses sensors (LiDAR, Sonar etc) to collect data which is processed using various algorithms (Extended Kalman Filters etc) to build a map. This is commonly known as Simultaneous Localization and Mapping (SLAM). In a SLAM module, the robot incrementally generates a global map, using probabilistic estimates of surrounding landmarks, and simultaneously estimates its position within the environment [1]. SLAM solves the problem of not having an initial planning of the navigation environment.

2.1.2 Pose Estimation

In order to achieve navigation objectives within the space, a robot needs to know its pose (position and orientation) at any time within the navigation environment. Pose estimation varies on a use case

basis. In most outdoor environments, GPS data can be used to define the pose of the robot, however GPS information is not accurate in indoor environments. For these applications, sensor data is processed using algorithms to output probabilistic estimates of the robotâs pose. A conventional approach is the dead reckoning which relies on the amount of rotation of the wheel coupled with inertial measurements from an Inertial Measurement Unit (IMU) to yield values for position (x, y) and orientation (θ) on a 2-D co-ordinate frame. Also, pose estimates are more accurate with more sensor data (i.e odometry, vision etc) [8].

2.1.3 Sensing

Online sensing allows the robot to keep track of changes, such as changes in obstacle position within its environment in order to adjust its path accordingly. The robot uses data from a range of sensors to interpret these changes in its surroundings. Sensors here include distance based or vision based. Distance based sensors measure relative distances of objects by shooting ultrasound waves or radars. Distance is calculated by measuring the time it takes for the waves to bounce back and return with reference to the speed of sound [31]. Examples of distance-based sensors include ultrasonic sensors, sonar sensors and radar. Vision based sensors record data as captured images of objects, these sensor types include- LiDAR, Camera, Real sense, etc. Robots equipped with more or a diverse sensor suites are usually more robust than robots with less sensors. However, the control strategy and kinematics are also contributing factors.

2.1.4 Path Planning

Given a starting pose, goal location, map describing the navigation space and other relevant data from sensor suite, autonomous robots use path planning algorithms to generate optimal paths that do not collide with any obstacles and lead to a goal destination. Two categories of path planning methods currently exist: Global and Local path planning. Global path planning algorithms are able to sample the whole configuration map, as a result generate an optimal path or simply a shortest distance path that do not collide with any static obstacle. Most global planning algorithms use a

sampling-based approach where the navigation space is naively or randomly searched, and nodes are created if the sampled space is collision free. These nodes keep developing until a clear connection is established, linking the robotâs starting pose to the goal. Given multiple sets of collision free paths, an optimization objective is done to select the optimal path. A global planner commonly used in many robotic path planning tasks is the Rapidly Exploring Random Trees (RRT) which incrementally generates tree-like trajectories that safely lead to goal in the navigation space [17] [18]. Although it is commonly used due to easy integration in higher dimensional configuration spaces, RRT does not generate optimal paths.

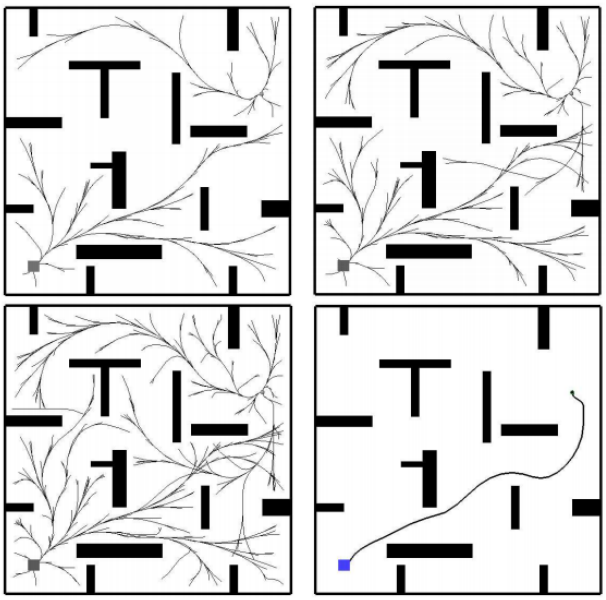


Figure 2.3: Multiple stages of RRT during planning.

Other popular variants of global path planners are the A* search algorithm [13], a derivative of the Dijkstra algorithm [28], which generates optimal paths by minimizing a cost function that models the distance from robot starting point to the goal. Common problems with global path planners are that they do not work well in dynamic environments. Since initial search is done with a global static map, a change in obstacle location means a path must be re-planned to accommodate new changes in optimization objective. Also, global path planners do not account for the geometry or kinematic constraints such as acceleration/ deceleration limits, max allowable speed, rotational speed of the robot, therefore on several occasions the path planner may generate non-feasible paths

based on the dynamic constraints of the physical robot. To address this issue, robotics researchers developed local planning methods known as active obstacle avoidance. Local planners factor in the geometry and kinematic limits of robots to generate feasible trajectories. Common local planning algorithms include Artificial Potential Fields (APF) [16].From a more general perspective, APF algorithm models a potential field in the navigation such that there is a hypothetical attractive force field between the robot point and the goal, and a repellent force between robot point and obstacles in the navigation space. The sum of the negative and positive potential fields yields a potential in which the negative gradient directs the path of motion representing the trajectory. APF algorithms are more prevalent in robotic manipulation. A popular local planner is the Dynamic Window Approach (DWA) algorithm which is basically a local obstacle avoidance that works well in dynamic environments. In most physical applications of robotic navigation, global and local planning are combined to generate optimal and feasible waypoints for robotic navigation. The popular Robotic Operating System (ROS),used by researchers and hobbyist alike to prototype robotic designs, uses the Dijkstra algorithm as a global planner and the DWA as a local planner in its path planning navigation architecture [29] [30].

2.2 Dynamic Window Approach

A general overview of the DWA algorithm was introduced in 2.1.4. For the basis of the work done in this paper, the Dynamic Window Approach (DWA) algorithm will be elaborated in more detail. The DWA is a reactive collision avoidance algorithm that incorporates the geometry and motion dynamics of the robot [10]. As with most other local planners, the goal of this planning algorithm is to generate feasible motion and steering commands in short time intervals that direct the robot to the goal. Based on the kinematic limitations of the robot, a 2-D dimensional velocity search space is created by considering all possible velocity pairs (v, w) that are reachable based on its kinematics. This search space (Vs) is then limited to only select velocities pairs that enable the robot to come to a stop in the vicinity of an obstacle considering the max deceleration of the robot. These velocities are known as admissible velocities (Va). A velocity pair is considered admissible

based on (2.1).

$$V_a = (v, w) | v \leq \sqrt{2 \cdot dist(v, w) \cdot v_b} \wedge w \leq \sqrt{2 \cdot dist(v, w) \cdot w_b} \tag{2.1}$$

Where (v, w) are the linear (v) and angular (w) velocity pairs. Given a velocity pair, the sub-function dist(v, w) generates the distance to the closest obstacle from the robot. Also, given that v_b and w_b represent the linear and angular accelerations for breakage respectively, based on the expression in (2.1), a velocity pair is admissible if the robot can come to a stop before colliding into an obstacle. Once admissible velocity pairs have been established, a dynamic window is created. The dynamic window is a sub velocity search space constituting velocity pairs from V_d that are reachable within the next time interval given acceleration constraints. Velocity pairs are considered reachable based on (2.2)

$$V_d = (v, w) | v \epsilon [v_a - v \cdot t, v_a + v \cdot t], w \epsilon [w_a - w \cdot t, w_a + w \cdot t] \tag{2.2}$$

Velocity pairs in the dynamic window V_d depend on selected admissible pairs and time. Based on (2.2), sets in V_d are time based, therefore velocities that are reachable given the current time step and current velocity, which constitute the dynamic window. Typically, a set of possible trajectories is generated at every time step using all velocity pairs in the dynamic window. The global velocity search space constitutes the possible velocities according to the specifications of the robot (V_s), admissible velocities (V_a) and finally the dynamic window (V_d)

$$V_{space} = V_s \cap V_a \cap V_d \tag{2.3}$$

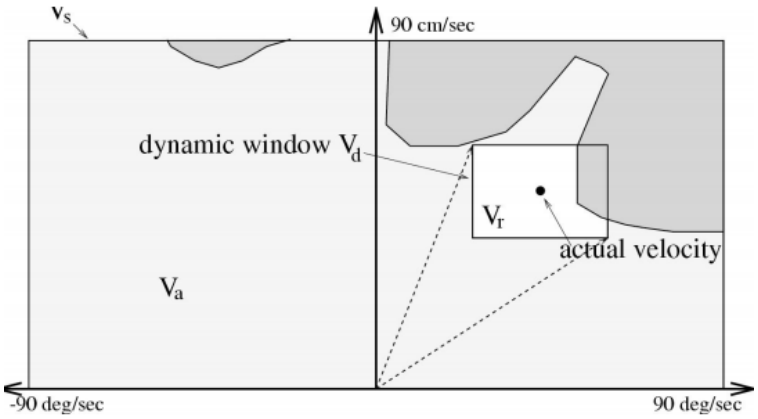


Figure 2.4: Velocity search space

In the space of these possible velocity pairs within the dynamic window that guide the robot through a path, there is an optimal velocity pair that yields the best local trajectory in the current time frame. This optimization problem is guided by a heuristic navigation function

$$G(v, w) = \alpha \cdot angle(v, w) + \beta \cdot dist(v, w) + \gamma \cdot vel(v, w) \tag{2.4}$$

Recollect that the v, w pairs are selected from the dynamic window (V_d), optimal velocity pairs are those that maximize $G(v, w)$. The sub-function $angle(v, w)$ evaluates the positioning of the robot calculated by determining the angle of the goal point with reference to the current heading direction of the robot. The $dist(v, w)$ evaluates the distance to the closest obstacle on a path and the $vel(v, w)$ expression represents the translational velocity of the robot. In some cases, the velocity(v,w) is used to measure the forward progress of the robot. The weighting constants α, β, γ are values between $\{0, 1\}$ that affect optimization of the navigation function which as a result affects the trajectory of the robot from start position to the target. All 3 sub-functions in the navigation function and their weights contribute to the optimal velocity pair selection which define the trajectory per time step.

Assigning higher weights to the distance to obstacle and forward velocity functions results in trajectories that do not move towards the goal. Also, bias towards the angle and velocity means the robot is not incentivized to maintain a safe distance away from obstacles within its path. Moreover,

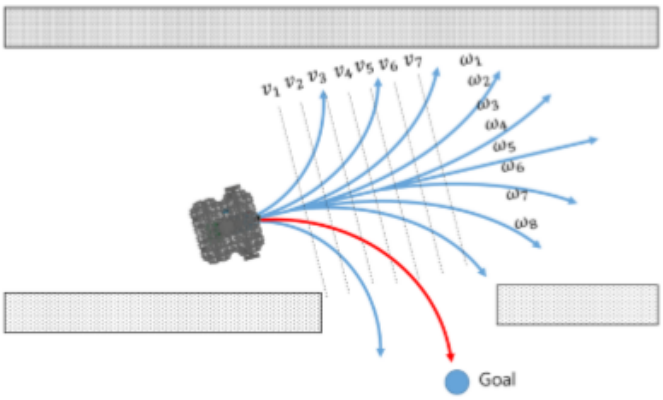


Figure 2.5: Trajectory roll-out based on v, w pair selection

the navigation function can be adjusted to augment the selection of trajectories based on the use case. A prime example is an application of the DWA for global obstacle avoidance in which the navigation function selects velocity pairs based on maximization object of velocity, a reward to stay within an optimized global path and a reward for reaching the goal [6] . The ROS navigation stack uses DWA as a local planner in combination with various global planners [29]. According to documentation provided in [29], overall path planning is implemented as follows:

- Discretely sample robot's 2D velocity space in user defined time intervals
- For each sampled velocity, perform a simulation from robotâs current state to figure out trajectory direction for that time step
- Evaluate and reject trajectories that collide with obstacles while selecting the one that maximizes the objective function which constitutes: closeness to obstacles, speed, closeness to goal, proximity to a global path
- Pick the trajectory with the max score and send velocity pairs to mobile base
- Repeat as necessary

As shown above, the algorithm uses a global planner to generate an optimal path, so the DWA navigation function is rewarded for maintaining that global optimal path.

2.3 Bayesian Optimization

Optimization problems are designed to maximize or minimize an cost function given a constrained or unconstrained set of design inputs. The cost function is a mathematical model of a physical or abstract system or a function that models certain function variables to be optimized. Therefore, if a system can be modeled mathematically, then it can be optimized. Optimization problems are solved based on the nature of the objective function model and design parameters. Certain optimizers use derivative or gradient information from the objective function to find a maxima or minima while others use other techniques such as exhaustive search, particle swarm, genetics to optimize objective functions with no gradient information. In this paper,Bayesian optimization is adopted as the primary optimization method. Bayesian optimization (BayesOpt) is a machine learning based optimization technique,used to optimize objective functions that are expensive to evaluate. [26] [21] [15] . In context, an expensive evaluation can mean it takes an exceedingly long time to generate evaluations, requires large computational resources, some monetary expense in running experiment and getting results etc. Also, there is no mathematical information or model on the objective function. In common nomenclature, the objective function is a black box function. In this case, there is some stochasticity in function evaluations and no linearity or convexity. As a result, it is impossible to apply gradient based optimization techniques as there is no gradient information from evaluations. The general objective for BayesOpt is to maximize or minimize a black box function $f(x)$, where the function is continuous.

$$max f(x) \tag{2.5}$$

The input parameter x has dimensions of $\mathbb{R}^d$, where d < 20 for most successful implementations. x is drawn from A, a feasible set typically a hyper-rectangle or a d dimensional simplex [11]. In the optimization objective in (2.5), $f(x)$ is an expensive black box function with some noise. To successfully implement BayesOpt, input parameters of d dimension can be generated using a space filling design approach. Since there is a limited evaluation budget, $f(x)$ can be sampled using a

few points in x in order to establish a surrogate model based on a prior observation of the function using Gaussian process regression. From this prior, an acquisition function is sampled over the prior distribution to suggest input points that likely maximize or minimize $f(x)$. BayesOpt is done iteratively as the acquisition function recommends a point in the input space until convergence [11].

2.3.1 Latin Hyper-cube Design Sampling

Computer simulated experiments require careful design of parameters in order to generate useful results that explain a physical system. One of the various methods in designing a close to randomized input parameter sets for a simulation is the Latin Hyper-cube Sampling (LHS). LHS is a statistical method of generating optimal random sample points that maximize an input bounded sample space [20] [14]. For a multidimensional input set, let x be an input parameter set matrix $\mathbb{R}^{m \cdot d}$, with m points and d dimensions that are independent.To design an LHS distribution, the range m of x is divided equally into m grid like segments with equal rows and columns. A point is generated independently on each segment using a probability function. Once samples are generated for each segment, all points are combined randomly to d dimensional pairs [19]

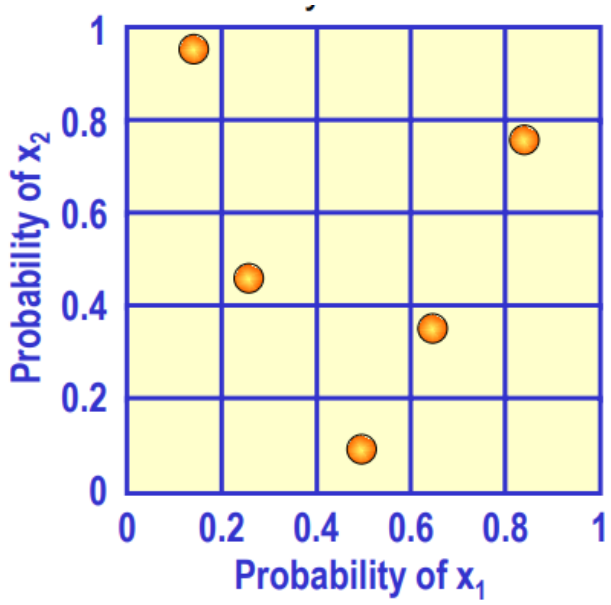


Figure 2.6: LHS sample distribution for 2 dimensional inputs

The advantage of using LHS sample points over randomly generated points is that they provide

data points for simulations based on well thought out statistical inferences and LHS points are well distributed over the input space both for small and large data-sets eliminating point clustering that may emanate from randomly generated samples [27].

2.3.2 Gaussian Process Regression

In chapter 2, Bayes-Opt was introduced and Gaussian Process was described as a means to generate a surrogate function model of an expensive unknown function. In more detail, Gaussian Process (GP) regression is basically a machine learning based approach to model function distributions [11]. GPs are subsets of Bayesian statistical methods which enable us to make probabilistic inferences on events based on a previous limited knowledge or assumptions of an event [4]

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.6}$$

Machine learning models such as the linear regression model a function using a parametric model with the goal of reducing the sum of squared error. In GP regression, a prior model is established based on a probabilistic inference or likelihood that the set of function evaluations within the input bounds are probabilistic. This function distribution is then updated as more data retrieved from the function. Let f be an expensive to evaluate black box function, with very minimal or no prior evaluations, of the function. A probabilistic inference can be made to describe the evaluations of f within the bounded input space, $y_prob = [f(x_1) \cdots f(x_p)]$. This Gaussian prior distribution defined by a mean vector and a co-variance matrix. The mean vector is established by running mean function μ at every point in the input space $x, \mu = (mean(x_i) \cdots mean(x_p))$. Co-variance matrix is evaluated by running a kernel function at a pair of x points,

$$\kappa_{ij} = kernel(x_i, x_j) \tag{2.7}$$

Where, $\kappa_i j$ is the co-variance matrix or a positive definite kernel function. The kernel function is generally a similarity function that assigns a metric to quantify the similarities between two input

points. This is done to infer that points with close similarities have similar function evaluations, so if input points x_i and x_j are similar then it is expected that their evaluations $f(x_i)$ and $f(x_j)$ will be similar [11] [24]. There are other variants of kernel functions i.e squared exponential kernel, power exponential, Martern kernel that can be used to quantify similarities. However, a more common kernel is the squared exponential kernel which is defined with the expression

$$kernel(x_i, x_j) = \alpha \cdot \exp(-(\frac{1}{2 * l^2}) \cdot (x_i - x_j)^T \cdot (x_i - x_j)) \tag{2.8}$$

Where l is a smoothing control and α is the vertical varication, which can be optimized by maximizing the likelihood metric. Using the mean and kernel functions, a GP regression generates a prior distribution $p(f(X))$ given only a d dimension input space X with no prior samples. Given some evaluations of the function y, the GP prior can be made a GP posterior $p(f|X, y)$, which is used as a regression function to make predictions when provided with new data points

$$\begin{pmatrix} y \\ f^* \end{pmatrix} \eta(0, \begin{pmatrix} K_y & K_* \\ K_*^T & K_{**} \end{pmatrix}) \tag{2.9}$$

(2.9) represents the prediction function to predict $f*$ based on a probability distribution η with a zero mean μ and a co-variance matrix κ, given training and test data sets. The kernel function in the co-variance matrix are sub matrices evaluated using the kernel in (2.9) as such

$$K_y = kernel(X, X) + \alpha \cdot I \tag{2.10}$$

$$K_* = kernel(X, X_*) \tag{2.11}$$

$$K_{**} = kernel(X_*, X_*) \tag{2.12}$$

For a posterior distribution given new data sets, an updated mean and co-variance can be cal-

culated using

$$\mu_* = K_*^T K_y^{-1} y \tag{2.13}$$

$$E_* = K_{**} - K_*^T \cdot K_y^{-1} \cdot K_* \tag{2.14}$$

2.3.3 Expected Improvement Acquisition Function

Acquisition functions measure the usefulness of sampling one or more points of a black box function in a bid to locate the maximum or minimum of the function. Expected improvement (EI) is a popular and easy-to-use acquisition function defines an expectation of improvement metric that suggests a sample point along the function that possibly yields an optimization improvement [21] [11]. For a Gaussian Process surrogate model of a black box function, assuming the best sample so far for a maximum optimization objective is $f_n(x)$, and another sample is made at with an arbitrary point within the input bounds, this current sample is the best so far if it is greater than $f_n(x)$. Since the value of $f(x)$ cannot be determined without sampling the objective function, multiple samples of the Gaussian posterior can be taken with the EI function and the point that yields the maximum expected improvement is returned as an output. A closed form Expected Improvement expression as defined by the expression

$$EI(x) = (f^* - \mu)\theta(\frac{f^* - \mu}{\sigma}) + \sigma\phi(\frac{f^* - \mu}{\sigma}) \tag{2.15}$$

In (2.15), the functions θ and ϕ represent probability and cumulative distribution functions of a standard normal distribution respectively. The mean μ and standard deviation σ are expressions that define the Gaussian Posterior model, therefore for any point within the input bounds of the posterior distribution that yields the maximum EI value, it is expected that there is an improvement, at that point, towards the optimization objective.

2.3.4 Bayesian Optimization - A simple overview

This section aims to illustrate the intuition behind Bayesian Optimization and Gaussian process with the aid of images, excluding mathematical terminologies presented in the preceding sections. Figure 2.7 shows two trend-lines. Recall that the premise of Bayes optimization is to find optimal points for unknown functions with limited evaluations. Therefore Gaussian process is utilized to build a surrogate model which is sampled liberally to find optimal points. Assuming the shape of a function, represented in dashed red, is unknown. However, we would like to build a map of this shape and use the map to find points that yield the maxima. Well, since there is no data or true information regarding the distribution of this function within an input space, a prior distribution is built. The blue horizontal line represents this prior distribution in Figure 2.7. This distribution is Gaussian, defined by a mean distribution and standard deviation of points within the input space. As there is currently no data to describe the function this distribution is a zero mean.

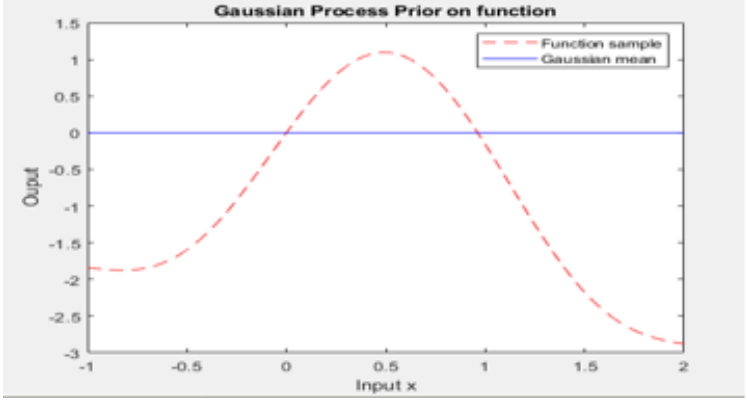
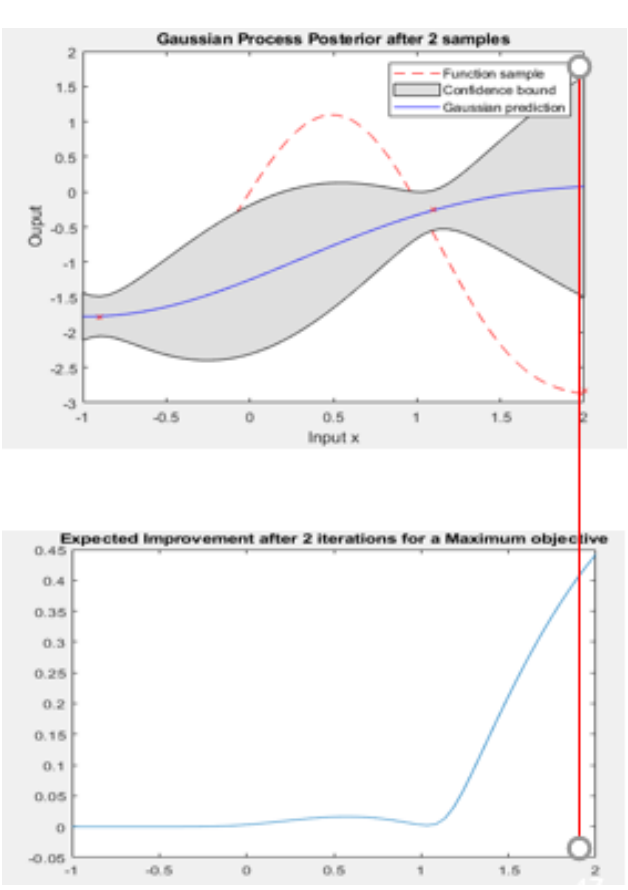


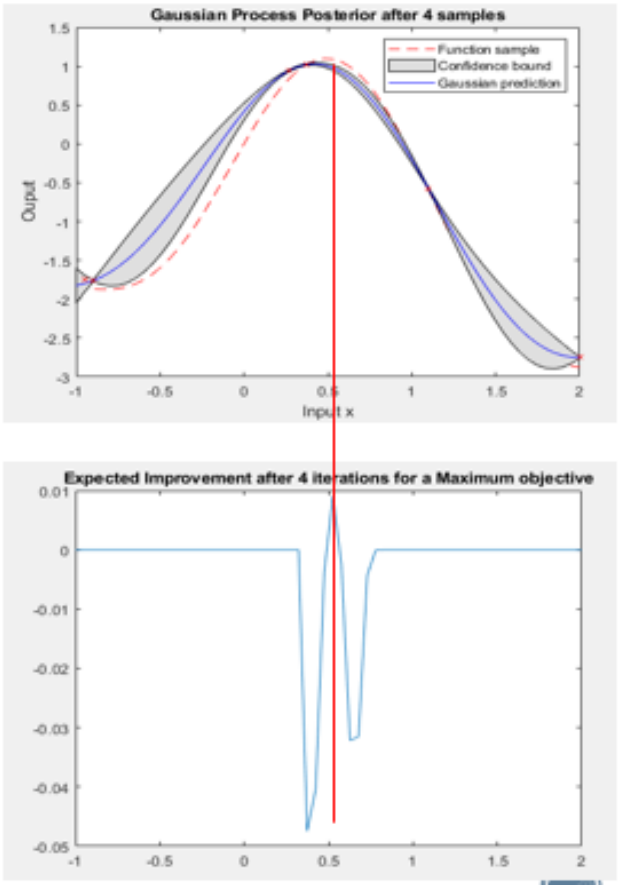
Figure 2.7: Gaussian Distribution of a function with no prior data

Now as some data is generated by sampling the actual function, the Gaussian distribution is updated to model the shape of the actual function as shown in Figure 2.8. Also, keep in mind that the maxima of the function is desired. Therefore as the Gaussian distribution is updated, the EI function is sampled over the distribution to find a point, within the input space, with the highest possibility of an improvement towards the optimization goal. The point that corresponds to the highest single EI value over the input space is the optimal point so far. This point is then used to sample the function and update the Gaussian surrogate model. In addition to finding optimal

points, there is also an incentive for the EI to max in areas with less data point distribution. There should be an exploration and exploitation balancing. If the EI function is more exploratory then it may take longer to converge at an optimal point. In contrast, if the EI function exploits the knowledge of a previously known minima or maxima, then it may want to converge at that point, even if yields a local minima or maxima for non-convex or non-concave functions.



(a) Gaussian Posterior and EI after 2 samples



(b) Gaussian Posterior and EI after 4 samples

Figure 2.8: Gaussian Posterior and EI with some observations.

As more samples are made available, an improvement is made towards the Gaussian surrogate model, as well as the optimization objective as shown in Figure 2.9.

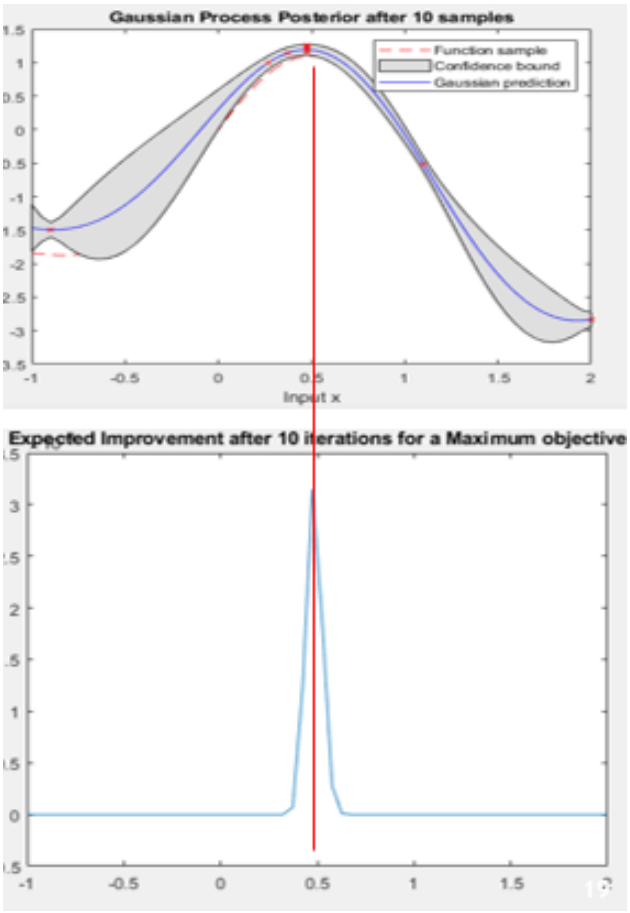


Figure 2.9: Gaussian Distribution and Expected Improvement after 10 observations

CHAPTER 3: METHODOLOGY

3.1 Dynamic Window Approach Simulation

3.1.1 Simulation Environment

A navigation simulation was developed in MATLAB using static obstacles at different (x, y) co-ordinate locations. The code for this simulation was a derivative of an open source python simulation project developed by Atsushi Sakai [25]. The objective of the simulation was to evaluate and visualize overall trajectories from robot start point to the goal location using optimized parameters from Bayes-Opt. Three navigation environments were created with different co-ordinate locations of static obstacles. Figure 3.1 shows the environment used for training. This training environment was designed with the expectation that a difficult to navigate training environment would yield parameters that work in a wider range of environments. Therefore, obstacle clusters were manually set around the goal co-ordinate to observe how trajectories circumvent these obstacle points and aim for the goal location.

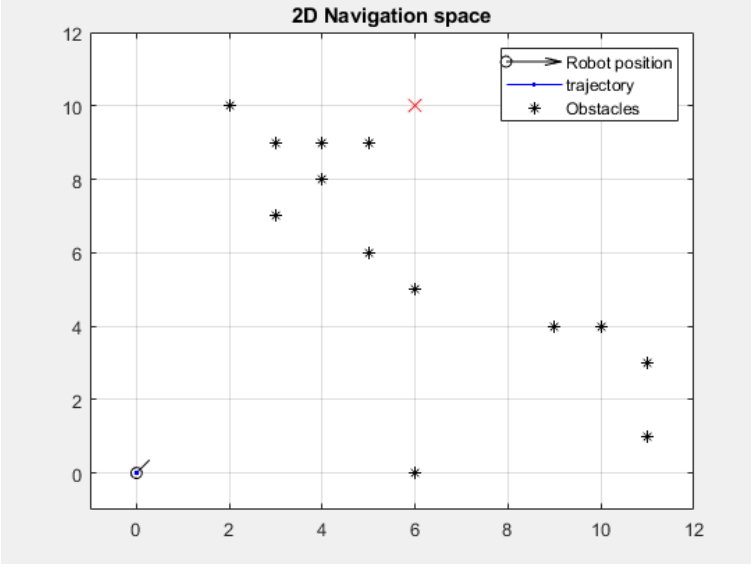
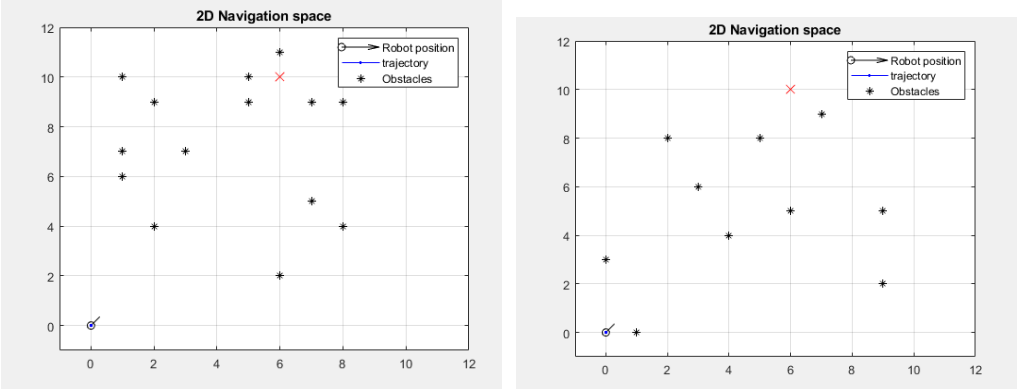


Figure 3.1: Navigation Training Environment.

The obstacle configuration in 4.14 ,used to test optimal parameter weights, are 10 and 15 ran-

domly distributed point co-ordinates within an input range $\{0, 10\}$ respectively. It is expected that all parameter weight combinations generated after training is completed, will successfully navigate to the goal point.



(a) Test Environment 1 (b) Test Environment 2

Figure 3.2: Navigation Environments for Testing Optimal Parameters.

3.1.2 Robot Kinematics

The robot used in this simulation is modeled as a rigid point moving in a 2-D co-ordinate frame, as a result, physical variables such as wheel rotation, friction, differential speeds at both wheels were ignored. Since physical constraints were excepted, motion control strategies to maintain stable locomotion in the event of environmental disturbances were also ignored. The robot's position and orientation was defined by its point location in a 2-D reference frame (x, y) and an angle ϕ. Angle ϕ represents the relative angle of the robot with respect to the global reference frame, as shown in Figure 3.4, where Y_g, X_g are the global frames and Y_l, X_l are the local frames.

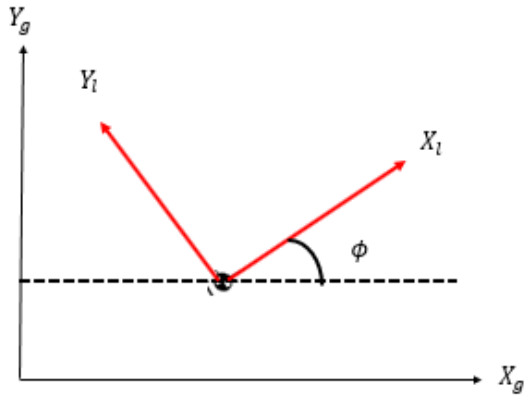


Figure 3.3: Robot Point in 2-D Global Reference Frame.

In the simulation, starting points for the x, y co-ordinate pair was (0, 0) and ϕ $\frac{\pi}{2}$. In order to yield a near realistic motion model for the robot point, motion variables were defined to magnitudes that constrain motion as shown in Table 3.1

Table 3.1: Robot Motion Settings

Kinematic Variables	Magnitude
Max Linear Velocity (m/s)	1.0
Max Angular Velocity (rad/s^2)	0.3491
Max Acceleration (m/s^2)	0.2
Linear velocity resolution (m/s)	0.01
Angular velocity resolution (rad/s^2)	0.0175

The Linear and angular velocity resolution are essentially the smallest velocity increments with respect to time. The point movement was simply guided by the linear and angular velocity pairs that maximized the navigation function per time step.

3.1.3 Updating the Robot Pose

At any time step increment, the position and orientation (pose) of the robot, identified by (x, y) and ϕ needs to be defined and updated. This update is basically guided by kinematic equations define the linear and angular velocities. The position co-ordinates (x, y) are updated using the expression in Equation (3.1) and (3.2) respectively. Also, orientation ϕ in units of degrees is updated using

Equation (3.3)

$$X_{i+1} = X_i + V \cdot cos(\phi_i) \cdot dt \tag{3.1}$$

$$Y_{i+1} = Y_i + V \cdot sin(\phi_i) \cdot dt \tag{3.2}$$

$$\phi_{i+1} = \phi_i + \omega * dt \tag{3.3}$$

Here, (X_i, Y_i) and ϕ_i represent the robot's current position and orientation in the 2-D reference frame, dt represents the time increment and ω is the angular velocity .

A state vector is defined to describe the state of the robot on the basis of the robot current co-ordinate, yaw angle and velocity pairs (v,w). Essentially, this vector is updated per time step as an appropriate velocity pair that maximizes the navigation function is chosen.

3.1.4 Generating the Dynamic Window

A velocity search space was generated for the robot using motion settings from Table 3.1 to select safe velocity pairs as described in 2.2. A vector V_s was defined to store values of the max and minimum linear and angular velocities allowable. Minimum linear and angular velocities in this case are 0 and the negative of the max angular velocity. From this allowable velocity vector, a vector V_a is generated that includes the current velocity, time, and resolution limits to generate a search space with only velocity pairs that are reachable within the next time step dt. The dynamic window is generated from this velocity search space vector V_a. Furthermore, vectors V_a and V_d are regularly updated at every time step as the robot updates its pose.

3.1.5 Calculating the Robot Heading

The Heading function keeps the robot facing towards the goal. This sub-function is evaluated by $180 - \theta_{corr}$. The variable θ_{corr} is the angle between the robot local horizontal reference frame and a reference vertical line at 90^o from the global horizontal reference frame X_g. θ is calculated by

taking the $\arctan$ of the difference between the corresponding (x, y) co-ordinate locations of the robot and the goal. A perfect alignment between and the goal point would yield approximately or close to 0 for θ_{corr}. Therefore, keeping this value at a minimum would keep the robot at an orientation facing the goal at every time step. Since the goal of the DWA algorithm is to maximize the overall navigation per time step, if θ is large then the heading is penalized, so velocity control actions that minimize the value of θ_{corr} are always desired.

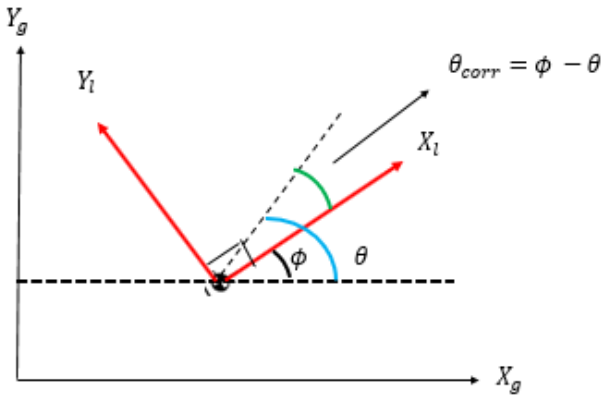


Figure 3.4: Robot Heading on a 2-D Co-ordinate.

3.1.6 Calculating the distance to obstacle

The distance to obstacle function measures the distance to the closest obstacle from the robot within the navigation space. This term is evaluated by calculating the Euclidean norm distance between the current robot coordinate to the coordinate of each obstacle. Since the function runs within a loop, it always calculates distances for every obstacle point, as the loop runs, without taking into consideration if that point is close enough to the robot. To resolve this, a condition is set to only consider euclidean distances that are less than 1.5 as distances to the closest obstacle, then this variable is used as an output for the navigation function.

3.1.7 Generating Cost Function

A cost function is needed to measure the performance of Bayes-opt. Generally, cost functions are modeled as error functions between predictions and actual values. In this case, the cost function is modeled using metrics that define the navigation simulation. Since the goal of this thesis is to find optimized weights for the navigation that reach goal at shorter time spans, the time taken for the robot to reach the goal was recorded including the average euclidean distance to obstacle and the euclidean distance to goal point at the end of a simulation trial. Using these metrics, a cost function was modeled as such

$$f^* = \frac{time}{max(time)} + max(avg(obstacledist)) * avg(obstacledist) + disttogoalpoint \tag{3.4}$$

Time taken for the robot to reach goal point is a major part metric of the function model. However, a sum of time, average obstacle distance and the distance to goal could also yield parameter weights that reach goal and maintain good clearances away from obstacles. The reasoning behind this cost function is that since the distance to goal metric and time are relatively large for unsuccessful trials, then a minimization objective can be set to converge to f^* outputs that take shorter times and reach goal

$$\begin{aligned} &\underset{\alpha,\beta,\gamma}{\text{minimize}} \quad f^* \\ &\text{subject to} \quad \alpha, \beta, \gamma \in \{0, 1\} \end{aligned} \tag{3.5}$$

3.2 Implementing Bayesian Optimization

3.2.1 Bayes- Opt Algorithm

The following pseudo code outlines the sequential implementation of Bayesian Optimization with Dynamic window obstacle avoidance. The loop condition is true when the number of overall evaluation trials of the navigation $N(32)$ are not exhausted. Before the looping condition, the

training set matrix X is used to generate 20 initial evaluations $Ytrain$. These evaluations are used as a starting point for the Gaussian surrogate model, so as more points are sampled, the training set is updated. The output after every iteration are the optimal parameter set that minimize the cost function f^*.

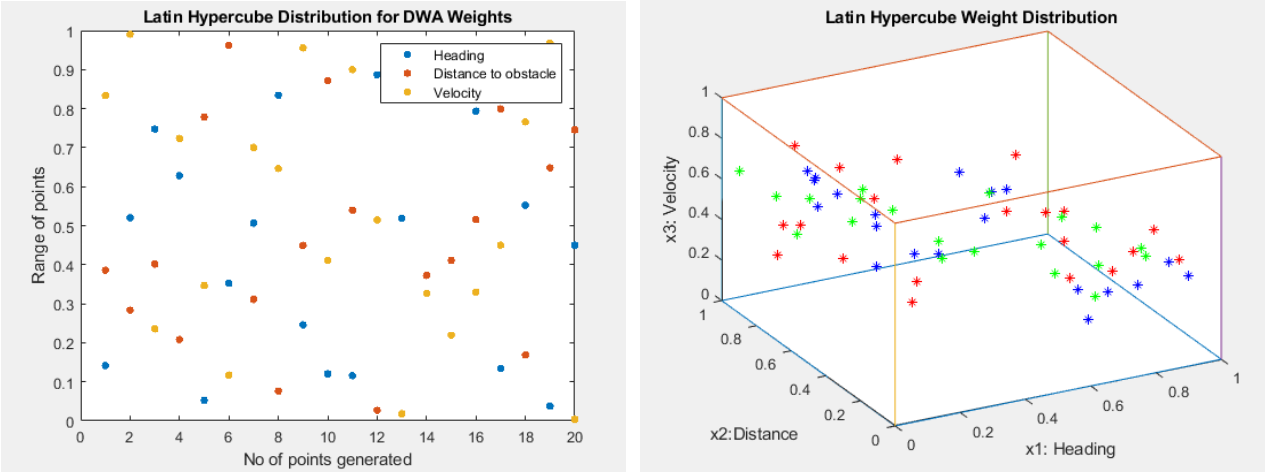
Algorithm 3.1 Implementing Bayesian Optimization with Navigation Simulation

Result: Approximate optimal parameter set
Generate Test and Training set X_{train}, X_{test} using LHD design function
Run navigation simulation using training set X_{train} to generate f^*prior evaluation in Y_{train}
while *n < N* **do**
    Run GPML function with current evaluations $Ytrain$ to generate Posterior distribution
    Run EI function to output next index corresponding to max EI
    Get next sample location in test set Z using index
    Run Navigation simulation to generate evaluations f^*
    Update training set X_{train} with sampling point and corresponding output vector with evaluation $Ytrain$
end

3.2.2 Generating weights using Latin Hyper-cube Design

A distribution of input weight parameters combinations that make up the training and test sets were used was generated using a Latin Hyper-cube design, a space filling random number process. Two distributions of parameter sets were generated: A training set and test set. The training set $X \epsilon\ \mathbb{R}^{20x3}$ is used as an input to generate 20 prior evaluations of the navigation simulation. Each column represents weights corresponding to sub-functions in the navigation function: Heading (α), Distance to Obstacle(β) and velocity(γ). Values are taken row-wise as inputs to the navigation simulation which is run 20 times to generate 20 prior evaluations.

The test set $Z \epsilon \mathbb{R}^{50x3}$ is a Latin Hyper-cube distribution of parameters from where the Expected Improvement function suggests the next sample. In other words, once the Gaussian posterior is sampled, the next sample location for an improvement in optimization is taken from Z. We chose a Latin Hyper-cube distribution over random distributions as they provide better space filling properties, therefore provide a better distribution of numbers within the intervals.



(a) 2-D LHD Weight Distribution

(b) 3-D LHD Weight Distribution

Figure 3.5: Latin Hyper-cube Generated Sample of Training Set X_{train}.

3.2.3 Generating the Gaussian Posterior

The Gaussian process prior and posterior distributions were generated using an external MATLAB Gaussian Process Machine Learning Library (GPML). We opted to using this library because it optimizes the l and α term in Equation 8 when generating the co-variance matrix, based on maximizing a likelihood singular value which represents the probability that a point has similar properties with a neighbouring point in the distribution. The GPML function takes in the training set and corresponding evaluations $Y \epsilon \mathbb{R}^{20x1}$ after running the navigation simulation with those points. The output is a vectored distribution of mean and standard deviation which describes the prior and posterior in Equation 8. Using sample points from the training set X_{train} a Gaussian process prior was established which is then transformed to a posterior as sample points from the test set Z_{test} are obtained. One an important note, the dimensions of the training set does not remain constant. As the optimization is implemented to look for optimal points, the Gaussian posterior distribution is also consecutively updated using training set data.

3.2.4 Expected Improvement Function

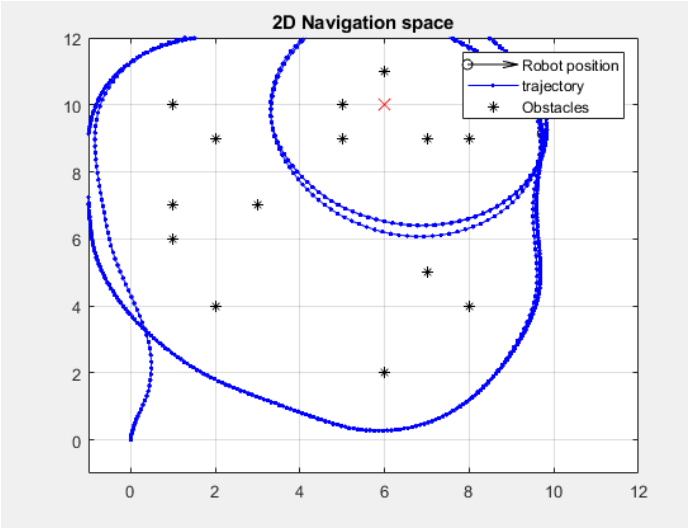
The Expected Improvement (EI) function samples the Gaussian posterior distribution and suggests a point that yields an improvement based on the optimization objective. An EI function is developed to take in evaluations of f^* and the Gaussian process function - the mean and standard deviations that describe the current posterior distribution. Concretely, the EI function is sampled over the surrogate for every update in the posterior distribution. Since the optimization goal is to minimize f^*, the minimum in f^* vector is evaluated for every posterior update and subtracted from the mean distribution vector generated from the GPML function and then apply Equation (2.15) to compute the EI. More importantly, the output of the EI is the index that corresponds to the negative of the maximum of all values of computed expected improvements to the optimization objective. This index is then used to extract the next sample location from the test set Z. This process is done repeatedly until we exhaust the budget for function evaluations.

CHAPTER 4: RESULTS AND DISCUSSION

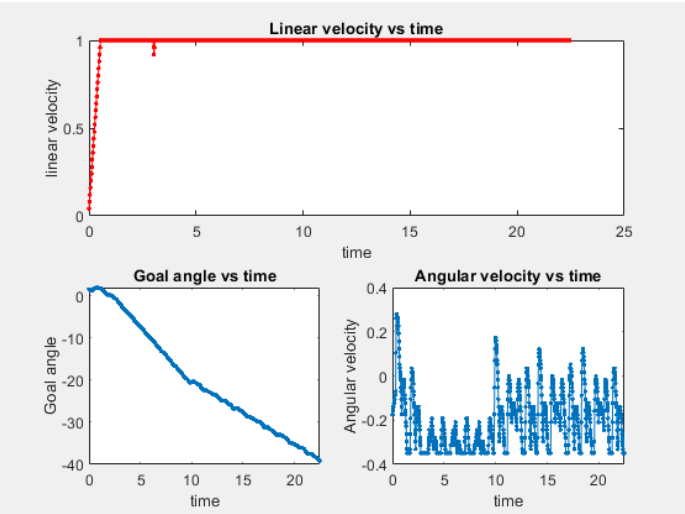
In this chapter, results obtained from running the simulations will be outlined and discussed. To restate the objectives of this thesis, Bayes-Opt is applied to generate optimal weighting parameters (α, β, γ) for the navigation function, in the DWA algorithm, that successfully reach goal point in shorter time spans . Furthermore, due to minimal research information on how these parameters can be tuned to improve trajectory generation, methods are proposed to inform this task.

4.1 Evaluating Obstacle Avoidance before Optimization

As discussed in Chapter 3, the navigation simulation was built to demonstrate the DWA algorithm to obstacle avoidance and also test approximate optimal parameter sets generated from Baye-Opt. Using LHD generated parameter training sets X, the navigation simulation was implemented to observe trajectory plots and evaluates that characterize each simulation. It is important to note that the LHD parameters are varied combinations, therefore diverse navigation outcomes are desirable. Generating prior training data with a more clustered obstacle space was chosen with the expectation that parameters generated from here would perform better in less or closely clustered environments. A caution is that over clustering a navigation space is not be feasible because the robot could always get stuck at point as there are no feasible trajectories to circumvent the obstacle(s). Figure 4.1 shows a navigation case where the robot failed to reach the goal using the input parameter sets from the 2nd row in Table 4.1.



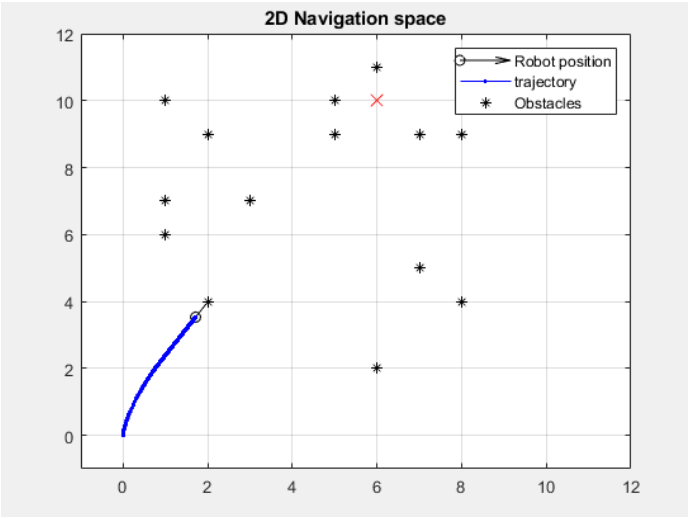
(a) Trajectory map



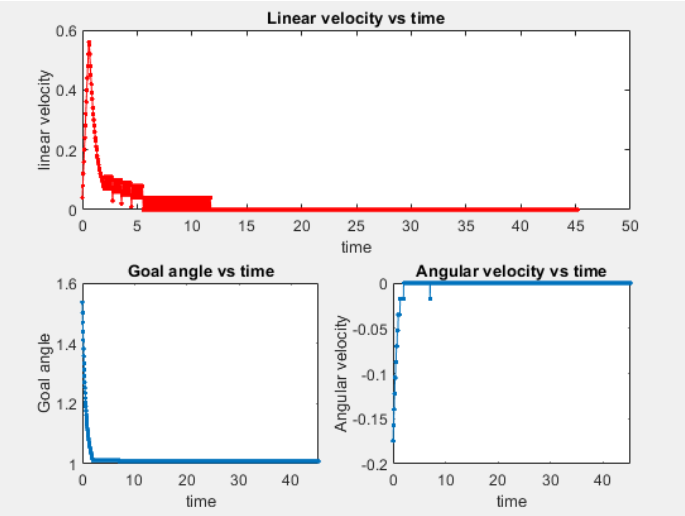
(b) Linear, Angular Velocity and Goal vs Time

Figure 4.1: Navigation Plots using α, β, γ weights from initial training set X_{train}.

These parameter combinations seem to be biased towards the velocity, thereby penalizing other parameter weightings. As a result, trajectories generated do not lead to the goal point. Figure 4.2 are navigation results corresponding to the parameters in the 9th row of Table 4.1.



(a) Trajectory map

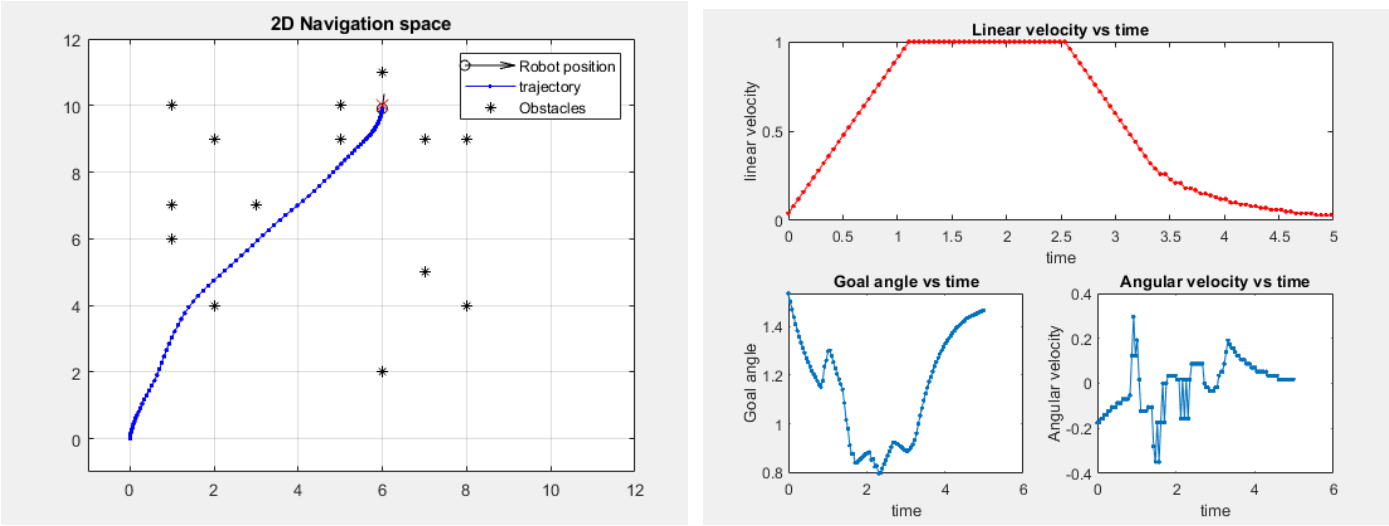


(b) Linear, Angular Velocity and Goal vs Time

Figure 4.2: Navigation Plots using α, β, γ weights from initial training set X_{train}.

Here the robot seem to be stuck at that point. As discussed in Chapter 2, the DWA algorithm is designed to avoid collisions, therefore at a time step, if there are no velocities to feasibly navigate

the robot around an obstacle, then the robot stops at that particular point to prevent colliding with an obstacle. The linear velocity plot also shows the drop in velocity around the 5s time stamp which stays at zero until the simulation is stopped. Figure 4.3 and 4.4 are navigation outputs corresponding to parameters in the 3rd and 12th row respectively. These navigation outcomes are ideally a representation of desirable trajectories, without factoring time taken or optimality of path (shortest path possible).

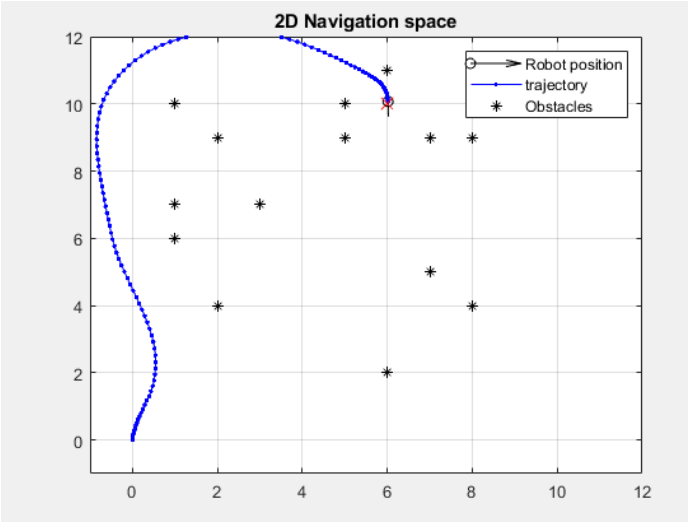


(a) Trajectory map

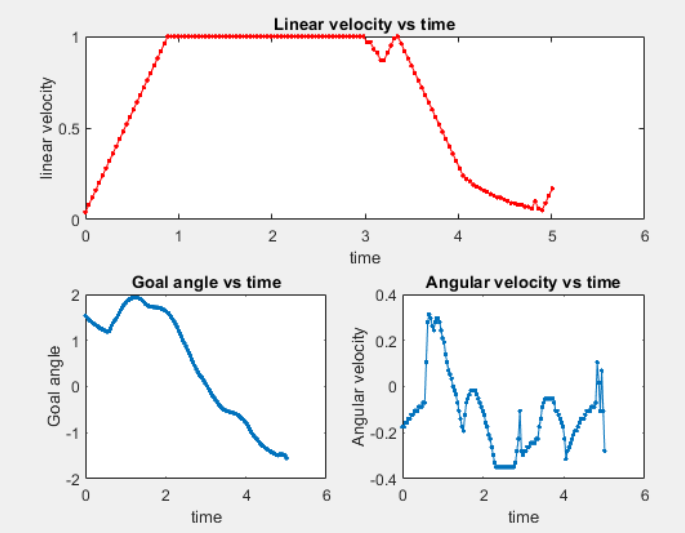
(b) Linear, Angular Velocity and Goal vs Time

Figure 4.3: Navigation Plots using α, β, γ weights from initial training set X_{train}

Although these two navigation plots successfully reached the goal point, they take a different path indicating some bias on a parameter. The trajectory in Figure 4.3 had a higher weighting on the velocity parameter, even higher than the corresponding weighting in Figure 4.2. However, the heading weighting - which is responsible for orienting the robot towards the goal - is a bit higher which might be a possible explanation as to why it successfully reached the goal. The trajectory in Figure 4.4 kept a much larger obstacle distance on average among other successful runs as shown in Table 4.3.



(a) Navigation space 1



(b) Linear, Angular Velocity and Goal vs Time

Figure 4.4: Navigation Plots using α, β, γ weights from initial training set X_{train}.

On the contrary, the parameters do not indicate a much higher bias (i.e highest weighting in the set) towards the distance to obstacle weights. It is expected, from experience, that this trajectory should be closely similar to Figure 4.2, as their velocity parameter weights are quite close numerically. Therefore this trajectory is a possible exception as it was unique for all 20 trajectories generating running the simulation. Table 4.1 shows the parameter set distribution used as the training set to generate a Gaussian prior distribution. Each row represents a parameter set used to run the simulation per entry to generate the metrics in Table 4.2.

Table 4.1: Results of Navigation metrics using initial weight distribution in X_{train}

Input			Output		
α	β	γ	Time (s)	Average min obstacle distance	Distance to goal
0.1414	0.3859	0.8334	22.677	0.562	3.829
0.5203	0.2836	0.9906	5.0232	0.153	0.095
0.7478	0.4013	0.2355	17.575	0.175	0.09
0.6283	0.2083	0.7237	5.029	0.128	0.096
0.0527	0.7783	0.3466	24.309	1.163	6.829
0.3523	0.9624	0.1172	45.207	0.512	7.760
0.5064	0.3112	0.7007	4.820	0.173	0.097
0.8343	0.0762	0.6465	5.114	0.080	0.096
0.2458	0.4496	0.9555	5.015	0.329	0.073
0.1205	0.8717	0.4114	24.517	1.106	6.830
0.1157	0.5395	0.9000	24.391	1.100	6.799
0.8866	0.0270	0.5146	5.119	0.077	0.096
0.5192	0.8486	0.0182	34.810	9.218	1.891
0.9270	0.3725	0.3267	13.287	0.151	0.097
0.8861	0.4113	0.2191	43.683	0.111	7.380
0.7937	0.5157	0.3299	16.451	0.176	0.098
0.1336	0.7995	0.4502	24.579	1.124	6.812
0.5524	0.1689	0.7661	4.960	0.089	0.098
0.0374	0.6483	0.9672	24.398	1.163	6.847
0.4501	0.7453	0.0040	34.539	1.788	9.234

The data in Table 4.1 shows important metrics used to evaluate the navigation simulation. As discussed, LHD generated parameters were used to allow for a diverse set of parameters. Parameters that maximize point space within the bounds $\{0, 1\}$ were desired, as a result the metrics emanating from these parameter sets reflected this diversity. From Table 4.2 50% of parameter sets used successfully reached the goal. Navigation data entries with distance to goal values greater than 0.098 did not reach the goal location. Furthermore, the corresponding time for these entries were a bit high because the simulation was programmed to keep running until the distance to goal value is less than 0.1, otherwise it is force stopped using an iteration count. This was uniform for all simulation trials. On average, it takes about 5 seconds for the robot to converge at the goal point from the starting point, so any entry exceeding this time stamp did not reach the goal. However, an exception was the entry in row 14, which took 13.287 seconds but successfully reached the goal

location. Using Equation 3.1, f^* values were generated as shown in the last column of Table 4.1.

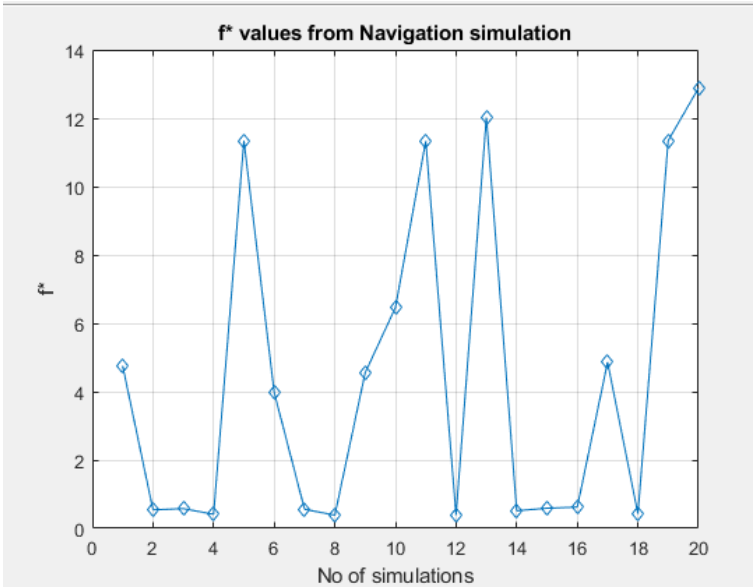


Figure 4.5: Cost function evaluations f^* using all 20 initial parameter sets in X_{train} before optimization.

4.2 Bayesian Optimization Results

4.2.1 Gaussian Regression Surrogate Models

Using prior evaluations f^*, from training set in Table 4.2, a Gaussian prior distribution is built to model the distribution of f^* as shown in Figure 4.5. The blue trend-line in Figure 4.6 shows the regression model over f^* using all 20 prior evaluations of f^*. As this regression distribution model is probabilistic, there is no expectation of a perfect fit over the actual function evaluation (the red trend-line).

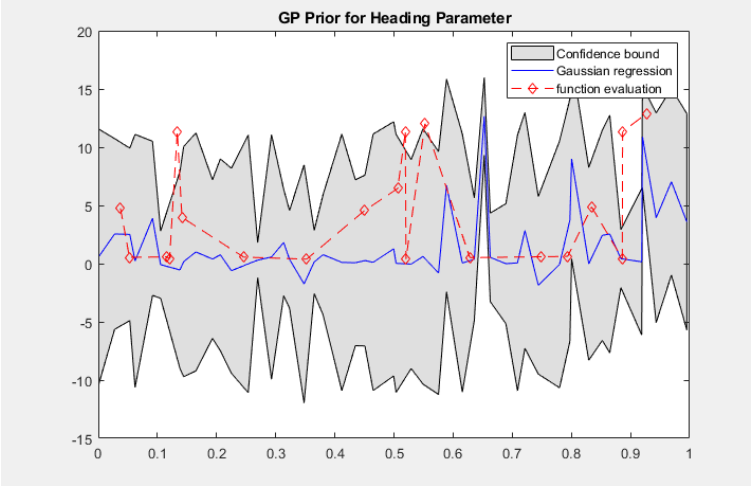


Figure 4.6: Gaussian Regression prior over Cost function f^*.

Based on a minimal f^* objective, the EI function is sampled over the Gaussian prior to suggest improvement areas corresponding to a maximum Expected Improvement over the distribution limits. As more evaluations are available, using parameter sets corresponding to the peak point (max EI) from test set, the regression model is updated to better fit the navigation evaluations f^*.

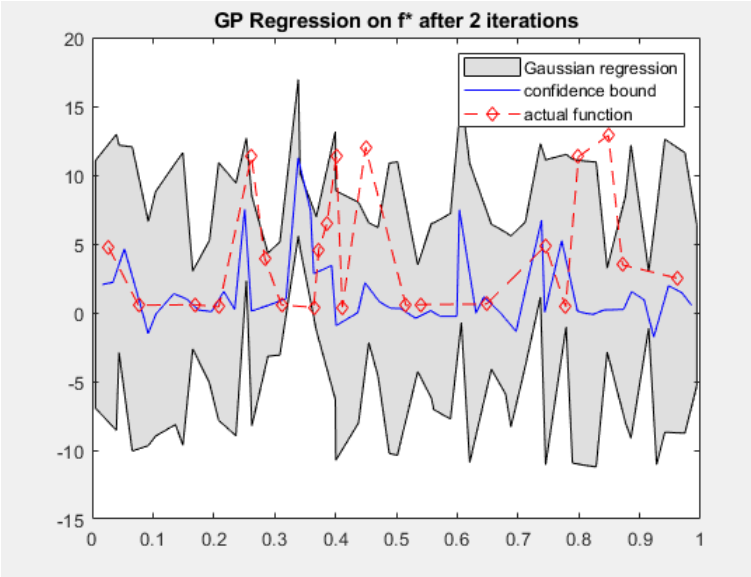


Figure 4.7: Gaussian Regression after 2 iterations.

In other words, evaluations from parameters in test set is used to update prior distribution which also improves the regression model as there is more knowledge of f^* with more evaluation points.

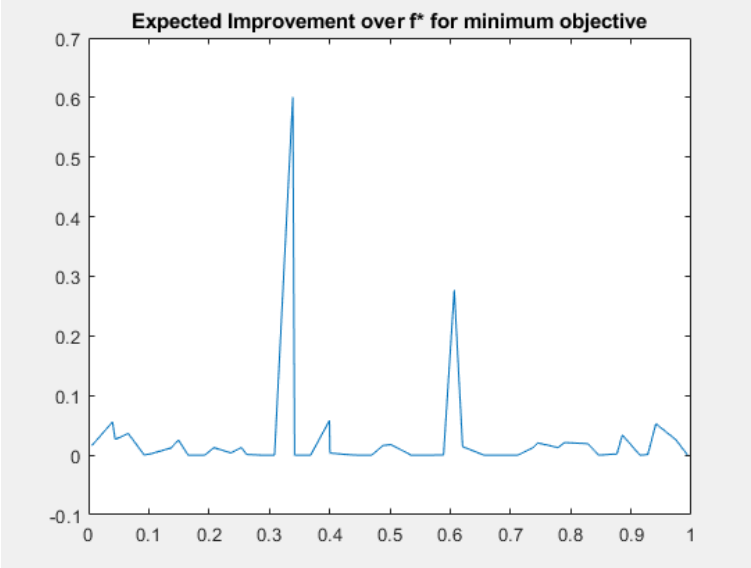


Figure 4.8: Expected Improvement over Gaussian Regression.

Due to limited evaluations of the navigation simulation (32 total evaluations), this process is repeated for 12 iterations to generate a final regression model after exhausting all evaluations.

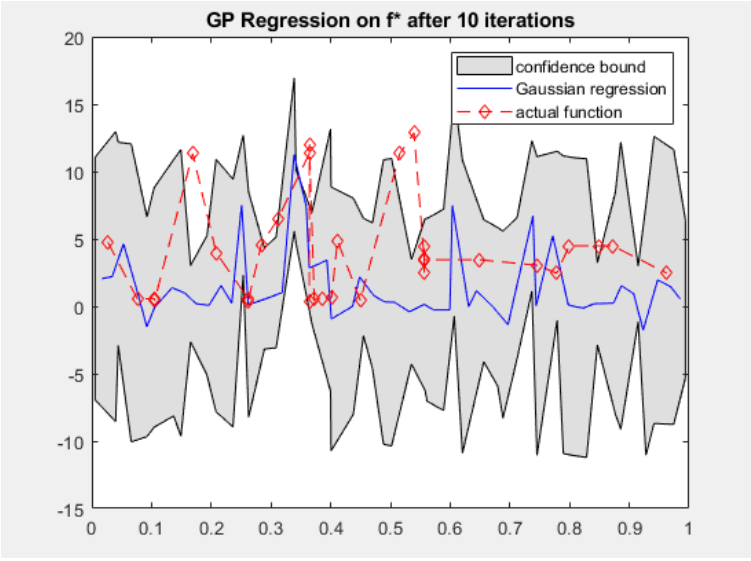


Figure 4.9: Gaussian Regression posterior after 12 Iterations.

Figure 4.9 also shows additional evaluations using parameters corresponding to the max Expected Improvement. Most of these new evaluations fall between 0-5 suggesting an improvement in the optimization objective to minimize f^*. Perhaps, there were more evaluations of the simulations allowed, there may not be a significant improvement in the regression model. This is because

as optimization begins to converge to the minimum, newer points will only cluster at areas of convergence which as a result, provide much vital information as to the shape of the function within the bounds.

4.2.2 Evaluating Optimization

Each parameter evaluation on Gaussian regression posterior is an improvement towards the optimization objective ($minf^*$). In comparison to prior evaluations before optimization in Table 4.2 and Figure 4.5, an overall reduction in f^* evaluations as shown in Figure 4.10. In more context, Figure 4.10 shows all the cost function evaluations sampled on the Gaussian posterior distribution. Selected optimal parameter sets would be the parameters with f^* evaluations at approximately 2.5.

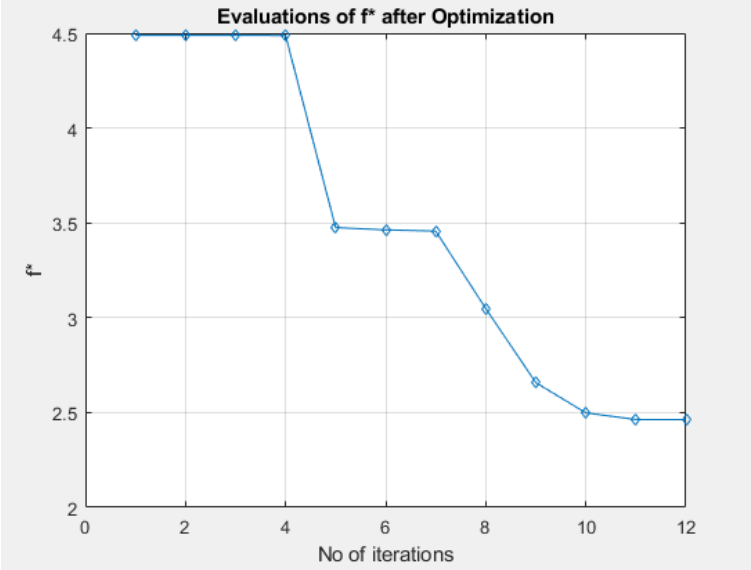


Figure 4.10: Cost function f^* evaluation using parameter weights corresponding to max EI over Gaussian surrogate

The point distribution in Figure 4.10 are cost function evaluations using parameter sets generated after the EI function samples the current shape of the Gaussian distribution. Remember that the GP model is being updated as the f^* is evaluated using metrics from the navigation simulation, so these evaluations are fed back into the training set output vector Y_{train}, which is subsequently used to update the GP distribution. This process is done iteratively. Minimizing f^* also minimizes the time as there is a linear relationship between both values. Figure 4.11 shows a reduction in

time through the optimization.

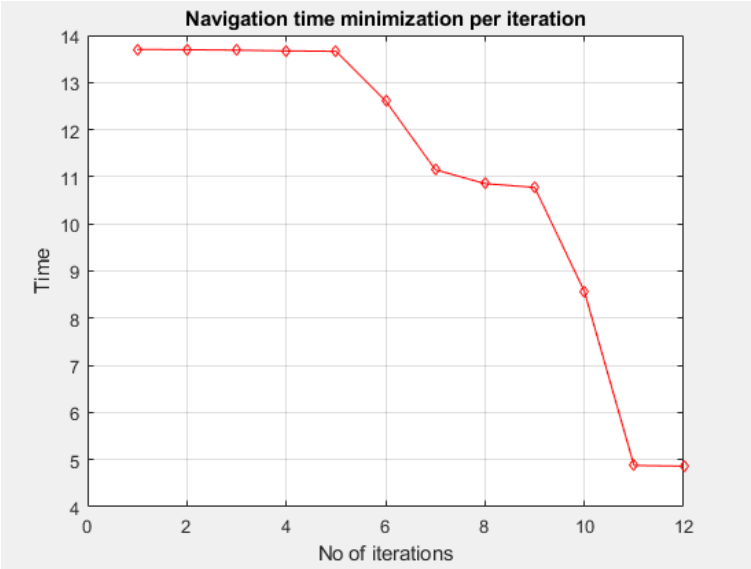
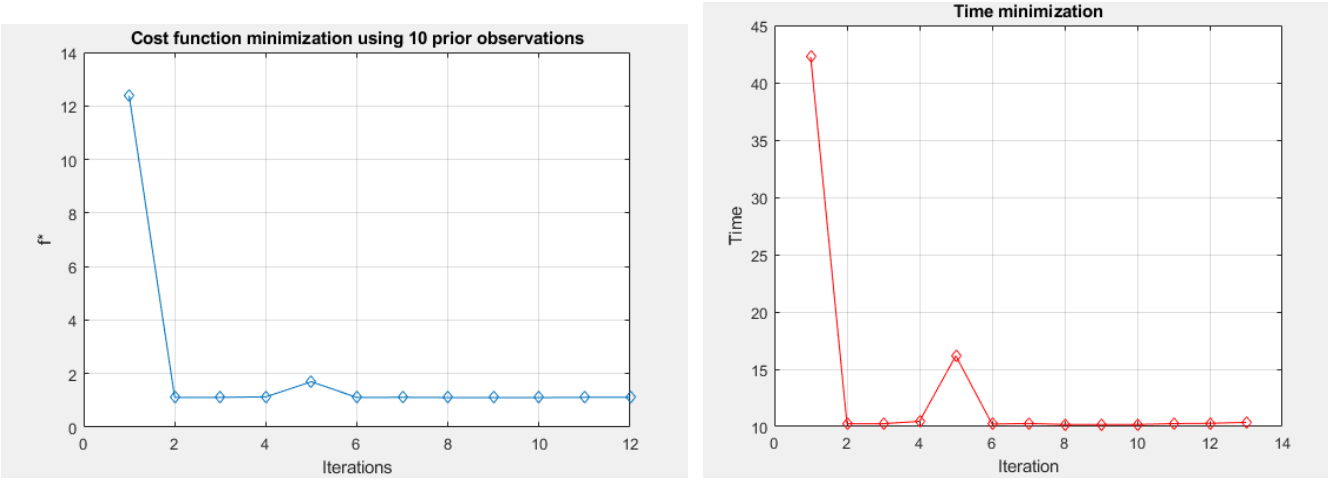


Figure 4.11: Time evaluations per optimized parameter.

It is important to note that the convergence trend of time or f^* actually depends on the size of variability of the training set and the corresponding evaluations. The starting point in the first iteration in Figure 4.10 seems to begin at a much lower value as compared to high f^* evaluations in Figure 4.5 before optimization. An explanation for this is that a Gaussian regression prior model is built in Figure 4.6 based on prior f^* evaluations in Figure 4.5, and it is from this regression model the optimization is based on. Furthermore, to begin the optimization, the Expected Improvement samples the regression model against the minimum evaluation value f^* so far and suggests an expected minimum based on sample from regression model. If using the input parameters in the training set only generated evaluations that did not reach goal therefore a high cost function value and time evaluations, then it may take longer and more iterations to find an approximate minimum. In order to explore other possible convergence options, Bayes-Opt was implemented using 10 prior evaluations as compared to results generated using 20 prior evaluations in Figure 4.10 and 4.10. The purpose of reducing the training set X_{train} is to observe the cost and time values from sampling a Gaussian distribution with limited evaluations compared to results using 20+ evaluations to iteratively build the GP distribution. The plots in Figure 4.12 illustrate the

cost function and time plots using parameters corresponding to the max EI, sampled on the GP regression. Since the current results are based on a much limited GP distribution, the resultant time with optimal parameters (10s), is much higher than results from Figure 4.11, where optimal time was approximately 5s. A cogent explanation for this is that since the GP regression used in Figure 4.11 was built with a training set of 20 simulation evaluations, the EI samples a better probabilistic distribution using more data points. As a result, convergence to a smaller time or cost value is more probable because the GP model is a better regression model. Another observation in the cost and time plots in Figure 4.12 is that convergence is much faster than that of Figure 4.11. A possible reason still ties down to the nature of the GP regression which the EI function samples. Since the regression used in Figure 4.12 is less accurate, optimization seemed to get stuck at a local minimum. Therefore, additional evaluation points on the regression can impose new minimums which is the case in Figure 4.11. Overall, it is preferable to spend available computational resources on generating real data to build better probabilistic regression models before EI is sampled to look for an approximate minima or maxima. Moreover, once the EI finds an optimal point, the next few points tends to converge at that point, therefore, using these new points does not relatively help in improving the regression model.



(a) Cost function

(b) Time minimization

Figure 4.12: Cost function and time plots using 10 evaluations for GP Prior distribution

4.2.3 Testing Optimized Parameter sets

One of the goals of this thesis is to generate parameter(s) combinations that work in various navigation environments. Parameter combinations corresponding to the last three evaluations in Figure 4.10, were selected as test weights for the three different simulation environments in Figure 3.2. Table 4.2, 4.3 and 4.4 show navigation metrics using parameters from optimization. All parameter combinations successfully reached the goal coordinates as shown by their distance to goal values. Both parameters had the lowest overall time in Environment 1. This is because navigation environment 1 was used to generate training data to build the regression model and sample new parameters during optimization. The first parameter weights achieved a lower overall time as compared to the second, however, obstacle distance suffers and is low on average. This translates to more straight trajectories that are close to obstacles. The second parameter weights took more time to reach final location, but they maintained larger distances away from obstacles on average. Here, trajectories that have some curvature seem to circumvent around an obstacle to maintain some distance.

Table 4.2: Overview of Navigation in Environment 1

Optimized Input			Output		
α^*	β^*	γ^*	Time(s)	Average min obstacle distance	Distance to goal
0.6884	0.1051	0.6686	4.667	0.083904	0.09
0.7454	0.5565	0.3385	9.4444	0.52476	0.09

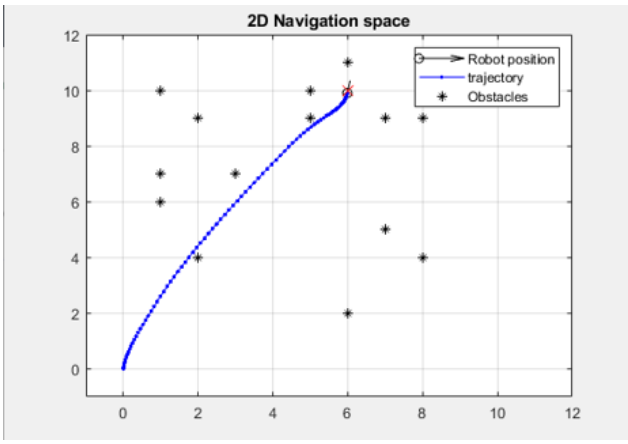
Table 4.3: Overview of Navigation in Environment 2

Optimized Input			Output		
α^*	β^*	γ^*	Time(s)	Average min obstacle distance	Distance to goal
0.6884	0.1051	0.6686	4.8459	0.087714	0.09
0.7454	0.5565	0.3385	16.74	0.18179	0.09

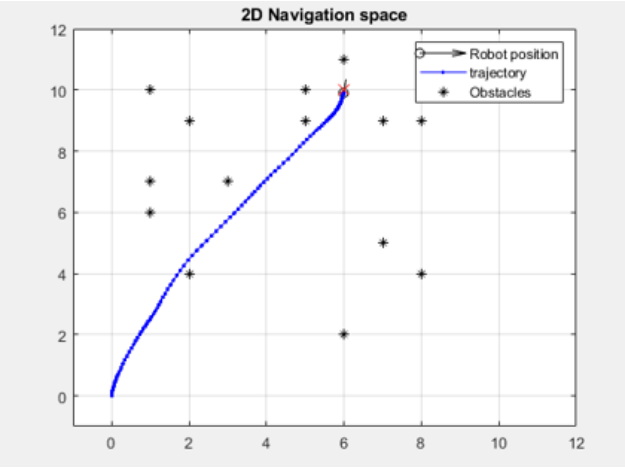
Table 4.4: Overview of Navigation in Environment 3

Optimized Input			Output		
α^*	β^*	γ^*	Time(s)	Average min obstacle distance	Distance to goal
0.6884	0.1051	0.6686	4.8901	0.245	0.09
0.7454	0.5565	0.3385	13.60	0.44351	0.09

Figure 4.13 illustrate trajectory images generated using optimized weights in Table 4.2, 4.3 and 4.4 in comparison with weights used in literature. In [10], a combination of [0.2, 0.1, 0.2] corresponding to $\alpha^*, \beta^*, \gamma^*$ was used to weight each sub-function in the DWA obstacle avoidance algorithm. The navigation trajectories generated using the weight combinations in [10] and weights optimal weights from Bayes-opt were observed. In the space of time, the parameter weights from Bayes-Opt averaged 4.84s in two different obstacle environments. In contrast, the weights in [10] averaged 6.293s in similar obstacle environments. Although parameters from Bayes-opt seem to run faster in the same conditions, their trajectories are more likely to stay closer to obstacles in the navigation path. The trajectory images in Figure 4.13(a) and 4.14(a), generated using optimal Bayes-Opt weights [0.6884, 0.1051, 0.6686], are more forward driven with not much incentive for the robot point to circumvent or keep some distance away from obstacles. Trajectories in Figure 4.13(b) and 4.14(b), generated using weights from [10], seem to maintain some distance away from an approaching obstacle. Furthermore, average minimum euclidean distance to obstacle was evaluated as 0.27691 using weights in [10] as compared to the same values for the corresponding optimal weights shown in Table 4.2,4.3 and 4.4. In respect to these observations, it is important to note that time minimization was a part of the optimization objective, as a result of this, parameter combinations generated devalue the obstacle distance costs to reduce the overall navigation time. Again the optimization process is data-driven, so parameter weight combinations that yield lower navigation times are more likely to be selected from the test set buffer based on time results using parameter weights from training set.

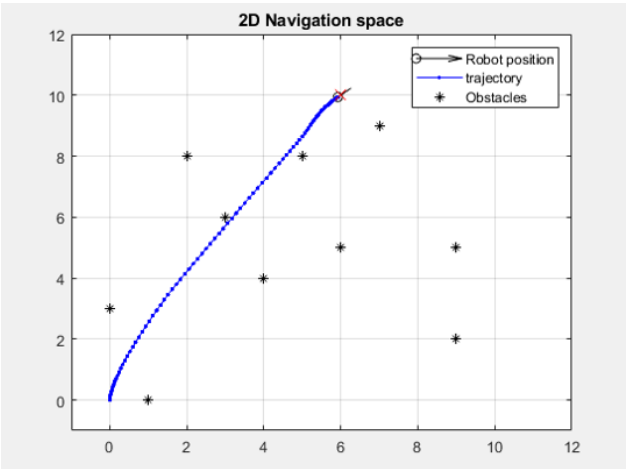


(a) Trajectory generated using optimal weights in Table 4.3

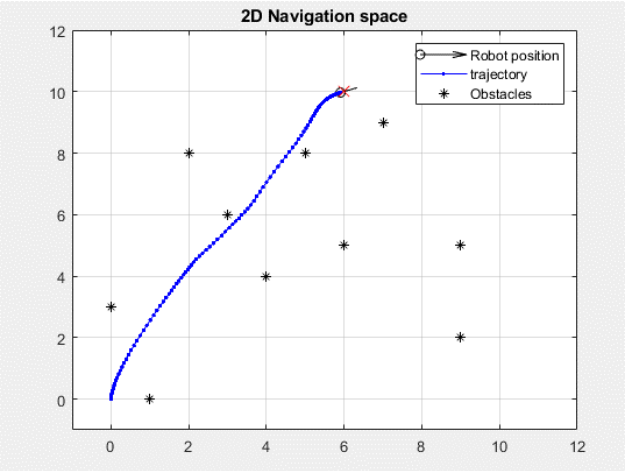


(b) Trajectory generated using weights in Literature

Figure 4.13: Comparison of Trajectories in Test Environment 1.



(a) Trajectory generated using optimal weights in Table 4.3



(b) Trajectory generated using weights in Literature

Figure 4.14: Comparison of Trajectories on Test Environment 2.

CHAPTER 5: CONCLUSION

Bayesian optimization was introduced to select optimal parameter weights for the Dynamic window approach algorithm for local collision avoidance. A MATLAB based simulation that models the motion dynamics of a robot as a point moving in a 2-D coordinate was incorporated into the functions of the DWA algorithm to select velocity controls that cause locomotion, in that optimal velocity pairs (v, w) from DWA algorithm are used as control actions to update robot location per time step. This optimization approach is data-driven, so there is a need to maximize points in the input space in order to generate a diverse range of evaluations. As a result, Latin Hypercube space filling design was used to generate X_{train} and X_{test} with 20 and 50 points for all α, β, γ respectively. Using a training output set Y_{train} comprising of cost evaluations corresponding to parameter sets in X_{train} , a Gaussian Regression model was developed and iteratively updated as new data is generated after running the navigation simulation. This regression model is theoretically distributed over the input bounds $\{0, 1\}$, so the parameter sets corresponding to the max EI are an approximately optimal and within the bounds $\{0, 1\}$.

5.1 Summary of Results

Two parameter weights in Table 4.2, 4.3 and 4.4 were selected after optimization as final optimal parameters to satisfy the navigation objectives desired. Both optimal parameters sets $(\alpha^*, \beta^*, \gamma^*)$ yielded trajectories that guided the robot to the goal co-ordinate, however navigation results using parameter set in Table 4.2 reached the goal point in a shorter average time than the others in Table 4.3 and 4.4. A brief explanation for this is that the navigation environment used in Table 4.2 was equally used in training, so it expected that optimal parameters would perform much better in time compared to newer environments. Parameter sets in Table 4.4 took a lot more time on average, however maintained a good clearance away from obstacles along its trajectory. In a more deeper sense, trajectories with lower navigation times have smoother paths but stay close to obstacles within its close range. Comparatively, trajectories with longer times have haphazard

paths to circumvent and stay far away from close obstacles in the next location. Trajectories and time metrics of optimal parameters from Bayes-Opt were compared with navigation metrics using weights from [10]. Parameter weights from Bayes-Opt used lesser navigation times but had forward leaning trajectories that were close to obstacles. In contrast, weights in [10] took a longer time but had trajectories that kept some distance away from approaching obstacles. These deductions only hold true for this simulation used in this thesis, so navigation outcomes may be different in other environments. However, reasonable trade-offs on navigation time and obstacle distance can be made using results and deductions from this thesis for local navigation approaches.

5.2 Future Directions

5.2.1 A more realistic simulation environment

In the MATLAB based simulation used to demonstrate obstacle avoidance, the robot kinematics was a simple kinematic model of a point moving in a 2-D Cartesian co-ordinate. Therefore, more realistic parameters such as wheel rotation and differential speeds were not factored into the forward and inverse kinematics calculation. Also, static obstacles were used to demonstrate obstacle avoidance. Realistically, navigation environments are not static, even in indoor environments. Autonomous mobile robots in real world scenarios encounter moving objects, clustered locations and in some cases unexpected jerking from external forces. Running training modules for Bayesian optimization with a robot simulation that incorporates a more realistic dynamics and environment will more than likely yield optimal parameter sets that work in a host of environments. In addition to utilizing more realistic environments, external robot simulation environments such as Gazebo, Open-Sim could be used. These simulators have a wide availability of robots with their respective kinematic models for navigation testing. So there is no need to generate kinematic equations or physical models of the environment.

5.2.2 Parameter weights for better Obstacle clearance

Results from this thesis illustrate that overall trajectories with smaller navigation times tend to stay close to obstacles along the selected path. Although this action does not result to collision with obstacle as the obstacle avoidance prompts the robot to stop in if a collision is predicted, trajectories that maintain good clearances from obstacles have an added safety factor than trajectories that are closer to obstacles. A starting point to solve this problem using the same methodology in this thesis, is to model an appropriate cost function that factors in a metric that is tied to the obstacle distance value and choosing an optimization objective that maximizes this value.

BIBLIOGRAPHY

[1] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[2] Romain Benassi, Julien Bect, and Emmanuel Vazquez. Robust gaussian process-based global optimization using a fully bayesian expected improvement criterion. In *International Conference on Learning and Intelligent Optimization*, pages 176–190. Springer, 2011.

[3] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[4] George EP Box and George C Tiao. *Bayesian inference in statistical analysis*, volume 40. John Wiley & Sons, 2011.

[5] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[6] Oliver Brock and Oussama Khatib. High-speed navigation using the global dynamic window approach. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 1, pages 341–346. IEEE, 1999.

[7] Riichiro Damoto, Wendy Cheng, and Shigeo Hirose. Holonomic omnidirectional vehicle with new omni-wheel mechanism. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 1, pages 773–778. IEEE, 2001.

[8] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 17(3):229–241, 2001.

[9] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Controlling synchro-drive robots with the dynamic window approach to collision avoidance. In *Proceedings of IEEE/RSJ Interna-*

tional Conference on Intelligent Robots and Systems. IROS'96, volume 3, pages 1280–1287. IEEE, 1996.

[10] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[11] Peter I Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[12] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[13] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[14] Ronald L Iman and WJ Conover. Small sample sensitivity analysis techniques for computer models. with an application to risk assessment. *Communications in statistics-theory and methods*, 9(17):1749–1842, 1980.

[15] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[16] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 500–505. IEEE, 1985.

[17] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[18] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[19] Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[20] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[21] Jonas Mockus. The bayesian approach to local optimization. In *Bayesian Approach to Global Optimization*, pages 125–156. Springer, 1989.

[22] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[23] YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, and TaeHoon Lim. Ros robot programming. *ROBOTIS, December*, 2017.

[24] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *Journal of machine learning research*, 11(Nov):3011–3015, 2010.

[25] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. Pythonrobotics: a python code collection of robotics algorithms. *arXiv preprint arXiv:1808.10703*, 2018.

[26] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[27] Felipe AC Viana. A tutorial on latin hypercube design of experiments. *Quality and reliability engineering international*, 32(5):1975–1985, 2016.

[28] Huijuan Wang, Yuan Yu, and Quanbo Yuan. Application of dijkstra algorithm in robot path-planning. In *2011 second international conference on mechanic automation and control engineering*, pages 1067–1069. IEEE, 2011.

[29] ROS Wiki. Documentation-ros wiki. *URL: http://www. ros. org*.

[30] ROS Wiki. Writing a global path planner as plugin in ros, 2015.

[31] Peng Zhang. *Advanced industrial control technology*. William Andrew, 2010.

VITA

Chinonso Ovuegbe is an international student from Nigeria. He earned a Bachelors degree in Mechanical Engineering from the University of Texas at San Antonio. He finally found his interest in Robotics and will earn his Master's degree upon completion of this thesis. He plans on getting a real word experience in Robotics and probably a PhD in the near future.