TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ALI GHIASI

PATH PLANNING OF A ROBOT WITH UNCERTAIN OBSERVATIONS

Master's thesis

Examiner: Professor Risto Ritala
Examiner and topic approved by the Faculty Council of the Faculty of Automation, Mechanical and Materials Engineering on December 7th, 2011.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Program in Machine Automation
GHIASI, ALI: PATH PLANNING OF A ROBOT WITH UNCERTAIN OBSERVATIONS
Master of Science Thesis, 87 Pages, 22 Appendix pages
November 2012
Major subject: Mechatronics
Examiner: Professor Risto Ritala
Keywords: Robot navigation, Path finding, POMDP, Dynamic programming, Value iteration

The purpose of the work is to assess the performance and further improve a solution to the problem of autonomous robot optimal path planning under uncertainty. The path finding happens on a 2D plane modeled by an overlaid lattice. The idea in the solution is to combine deterministic and stochastic approaches. First assuming complete knowledge of the environment, the deterministic path planning problem is solved resulting in an optimal path; after that knowing that there may also be some unmapped static or slowly and randomly moving obstacles present in the environment; the online stochastic solution uses dynamic programming method to solve the path finding with obstacle avoidance problem.

The proposed solution was rigorously put to test with different parameters and under various configurations to evaluate its performance and identify its weaknesses. The results of conducted experiments revealed notable achievements along with excellent opportunities for improvements. Hence, attempts were made to seize those opportunities and enhance the performance of the solution. The outcomes of those efforts were 35 % increase in the success rate and reduction in the time required for the solution to reach its goal by over 97 %.

ACKNOWLEDGEMENTS

This work would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to Dr. Risto Ritala, my advising professor whose patience and encouragement I will never forget. Dr. Ritala has been on my side as I hurdle all the obstacles in the completion of this research work.

I would like to express my thanks to Miika Rajala, for his assistance in early phases of this work, and my colleagues and staff in the Automation Science and Engineering Department for their moral support and use of facilities.

Also I am inclined to thank the Administrators of the Faculty of Automation, Mechanical and Materials Engineering, for their untiring effort in preparation of a relaxed atmosphere nurturing students' growth.

Last but not the least, I would like to dedicate this work to my parents, my sister and my brother and one above all, my beloved wife, for if it wasn't for her unconditional love, support and reassurance I would have thrown in the towel a long time ago, thank you so much dear Leyla.

Ali Ghiasi
November 22nd, 2012

CONTENTS

LIST OF SYMBOLS

A_m: Motion actions
A_o: Observation actions
L_i: Location of the machine
L_{idle}: Idle location
L_{target}: Target location
P_{0b1}, P_{ob2}: Observations erring probabilities
S_i: State space model
T_0: Short time horizon
p_0: Initial occupancy belief
$\tilde{r}(l_i)$: Reward of occupying a node
r_{coll}: Collision reward
q: Propagation probability of random walkers
CI: Collision Incidents
m, n: Dimensions of the environment
O: Obstacle occupancy
r: Reward function
SC: Steps Counts
T: state transition function
α: Discount rate
A: Allowed actions
OB: Observations
i: Time or time-step index
$p(O(L_i))$: Occupation probability of locations
t: Time or time-step

LIST OF ABBREVIATIONS

CAR: Collision Avoidance Rate
CC: Command Controller
COC: Collision or Crash Cost
DE: Dynamic Environment
DP: Design Parameter
IMs: Incautious Motions
MDP: Markov decision process
MLs: Motion Loops
MOLs: Multiple directional Observations Loops
NEC: Normal Environment Conditions
OLs: Observation Loops
OPF: Online Path Finding
OPP: Offline Path Planning
OTSR: Observation steps to Total number of Steps Ratio
OU: Occupancy Updater
POMDP: Partially observable Markov decision process
SE: Static Environment
SLC: Secondary Loop Controller
SOLs: Single directional Observations Loops
STH: Short Time Horizon

1. INTRODUCTION

1.1. Autonomous robot navigation

Over the past few decades and with swiftly evolving technologies, developing autonomous intelligent robots has sparked an immense fascination amongst many researchers, especially those in the Artificial Intelligence (AI) community. In particular, autonomous robot navigation has attracted lot of attention and large amount of resources due to its importance and broad applications in defense and military, aeronautics and space and many other technological and industrial organizations.

Strictly speaking from path finding perspective, navigation design of autonomous robot comes down to three key features:

- The ability to plan optimal paths.
- The ability to move towards a target location in a real-time environment.
- The ability to circumvent obstacles and correct the course to avoid collision.

Initially, these aspects were dealt with separately. However, with extensive research and development of novel methods, capable of handling complex problems in real-world applications, the areas are now pursued in combination [1].

The first phase, planning an optimal path, is the process of finding executable geometric paths for a robot from a start position to its destination and determining optimal one if more than one path exists. A geometric path consists of a set of parameters expressing how a robot is located in the physical world. For example, the simplest configuration of a robot in a Cartesian space is its coordinate pair.

The second feature, motion planning in real-time world, takes time into account. It refers to a sequence of executable kinematic and dynamic motion actions that enables a robot to maneuver itself in the environment [2]. In this work with the robot only simulated, the motion planning is reduced to choosing the optimum action from a list of available activities at each time step.

The final feature approaches the world with a more realistic notion, assuming that the environmental information is generally only partially available or completely unavailable in advance [1]. Therefore, it is necessary to perceive uncertain position and behavior of the obstacles in order to navigate safely toward the goal and to modify robot’s conduct accordingly. The quality of robot’s perception depends on the sensors providing a probabilistic estimation of the occupied and free space and the portion of the environment used to update the world model according to the current sensor observation [3]. In addition to unknown environment, other factors are also affecting

autonomous robot's ability to navigate towards its destination, such as localization uncertainty due to motion imperfection. Therefore an autonomous robot should be able to take those factors into consideration while planning motions and evading objects [4].

1.2. A motivational example

Good examples that vastly inspired and motivated the author of the thesis are the two autonomous rovers of the ongoing Mars Exploration Rover Mission (MER), Spirit and Opportunity, exploring the planet Mars (see *Figure 1.1*).

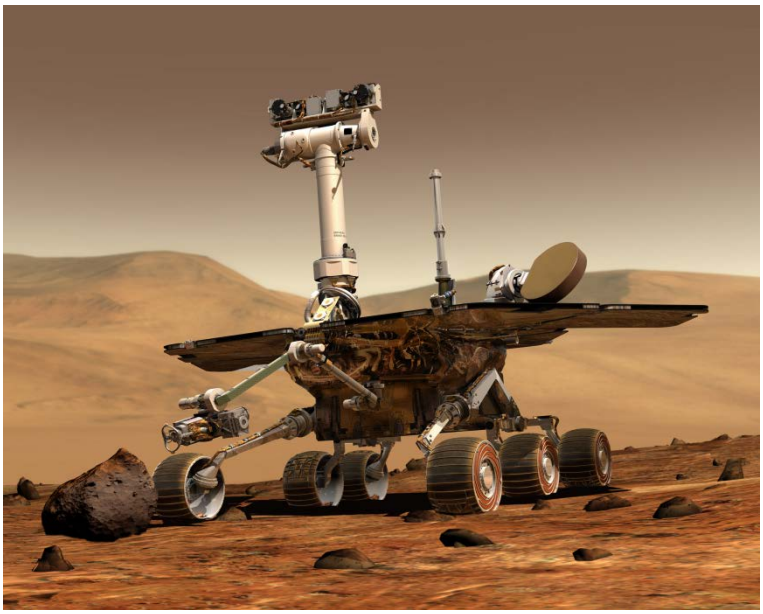

Figure 1.1: A concept portrays a NASA Mars Exploration Rover on the surface of Mars [5].

Since 2003, the twin rovers have been exploring and navigating through the Martian surface. Considering the difficulties of guiding the robots from the Earth, the robots have largely been acting autonomously. Thus, from navigational point of view, their missions have been exemplary cases of path planning and execution under uncertainty and environmental constraints [5].

1.3. Organization of this work

The thesis is divided into eight Chapters. The first Chapter consists of introductory material. Chapter two is dedicated to theoretical background giving a brief understanding of planning methods and algorithms to the reader, with emphasis on the methodologies employed in this work. It is then followed by the problem of optimal path finding with static obstacle avoidance and its formulation in Chapter three. The fourth Chapter aims to explain the pre-existing solution to the problem while its shortcomings and their rectifications are covered in Chapter five. The navigation problem with dynamic obstacle avoidance and its implementation are discussed in the sixth Chapter. The results of numerous experiments with different version of the

program and the statistics gathered from the performance of the robot are illustrated in Chapter seven. Finally the thesis is concluded and some future directions are outlined in the eighth Chapter.

1.4. Author's Contribution

The original program was handed to the author for assessment by Prof. Ritala. In its original form the program was able to some extent to navigate in a static environment. Since then, the author has been responsible for evaluating the program, identifying its limitations and finding solutions to improve the performance.

The results of these activities are manifested in form of statistics and charts of robot behavior, mainly addressed in first half of Chapter 5 and Section 7.1.1. Moreover, the author implemented new methods and auxiliary components enhancing the robot's performance, most notably the controller module which is covered in the latter half of the said Chapter. The modified program was then put to the test extensively with their results featured in sections 7.1.2 and 7.1.3.

The author was to add to the robot the capability of navigating in a dynamic environment. To achieve this, some modules were added and adjustments were applied to the program. These activities are addressed in Chapter 6. The program was tested in this state to assess its performance and the results are illustrated in Section 7.2.

2. THEORETICAL BACKGROUND

2.1. Planning elements

Although there are broad classes and models of planning, they all virtually share the same basic elements. These elements are state, initial and goal states, actions, a plan and a criterion.

In general, planning methods incorporate state space models capturing all possible situations that could occur. The states can be either discrete (finite, or countable infinite) or continuous (uncountable infinite) or combinations of them. The planning problem is to reach the robot's specified goal state, from an initial state. The actions seek to achieve this by manipulating the state. Resulting changes in the state may be expressed as a state-valued function.

The plan is a specific policy, mapping from state space to action space. It may be as simple as an explicit sequence of actions or more complex. The desired outcome of a plan is set by a criterion and one that maximizes the given criterion is an optimal plan.

In many applications, it is difficult to formulate the right criterion to optimize and when it can be formulated, it may be impossible to obtain a practical algorithm that computes optimal plans. In such cases, sub-optimal solutions may be devised instead to formulate an approximate criterion. For problems that involve probabilistic uncertainty optimization arises more frequently. The probabilities are often utilized to obtain the best performance in terms of expected costs [6].

2.2. Discrete planning methods

In this work the planning problem considered has a finite state space. Therefore in this thesis there will not be any aspects regarding planning in neither continuous state space nor will there be any need for geometric models or differential equations to characterize the planning problem. At its core a discrete path planning method is a systematic graph search. Being systematic is the key requirement for these or any search algorithm. In a finite graph, being systematic translates to the algorithm visiting every available state and keeping track of them, which in finite time enables it to correctly declare whether or not a solution exists [7].

The rest of this section is mainly based on the reference [6]. Before starting with particular algorithms, it is beneficial to outline general forward search method. It works by starting at one vertex and exploring adjacent nodes until the destination node is

reached. At any point during the search, three types of states may come forth. A node may,

- Have remained unvisited
- Have been visited and every potential next state also has been visited, “dead”
- Have been visited but there are still potential next state left, “alive”

The set of *alive* states is accumulated in a priority line up, for which a priority function must be specified. This algorithm serves as an entry point for other search algorithms since the only significant difference between them is the particular function used to sort the queue. Some search algorithms require a cost to be computed and associated with every state which may be used to sorting, or otherwise enable the generation of the plan on completion of the algorithm. Here the optimal cost to return to the initial state could be stored with each state instead of storing pointers. The action sequence, which leads to any visited state, is sufficiently determined by this cost alone. In the following several search algorithms, each of which is a special case of this algorithm, will be introduced.

Breadth first is a method that specifies the priority function as a First-In First-Out (FIFO) queue, which selects states using the first-come, first-serve principle. This causes the search frontier to grow uniformly and is therefore referred to as breadth-first search. Breadth first guarantees that the first solution found will use the smallest number of steps.

Depth first is a variant of previous in which the priority function is made a stack (Last-In, First-Out; or LIFO), thus aggressive exploration of the state transition graph occurs, as opposed to the uniform expansion of breadth-first search. The algorithm is called depth-first search because the search dives quickly into the graph inclining toward investigating longer plans very early. This aggressive behavior might seem desirable though the particular choice of longer plans is arbitrary. Actions are applied in loop in whatever order they happen to be defined. The search could easily focus on one “direction” and completely miss large portions of the search space as the number of iterations tends to infinity.

Dijkstra’s algorithm is an algorithm for finding single-source shortest paths in a graph, which is a special form of dynamic programming. Assuming that in the graph representation of a discrete planning problem, every edge is associated with a nonnegative cost to apply the action. Then the total cost of a plan is the sum of the edge costs over the path from the initial state to a goal state. The priority queue is sorted according to a function called the cost-to-come. For each state, the value called the optimal cost-to-come from the initial state is defined. This optimal cost is obtained by summing edge costs, over all possible paths from start to each state and using the path that produces the least cumulative cost [8].

The A^* search algorithm is an extension of Dijkstra's algorithm that tries to reduce the total number of states explored by incorporating a heuristic estimate of the cost to get to the goal from a given state known as cost-to-go. In some problems however, it is hard or impossible to find a heuristic that is both efficient to evaluate and provides good search guidance [9].

Best first sorts the priority queue according to an estimate of the optimal cost-to-go. The solutions obtained in this way are not necessarily optimal. However, in many cases (not guaranteed), the algorithm runs much faster since fewer vertexes are explored. The worst-case performance of best-first search is worse than that of A^* search and dynamic programming [9, 10].

Iterative deepening tries to find all states that are located in a certain distance i, or less from the initial state. The work is discarded if the goal is not found, then, it seeks all states of distance $i + 1$ or less instead. This algorithm generally iterates indefinitely until the goal is reached. It is usually preferable approach if the search tree has a large branching factor which could occur if there are many more vertices in the next level than in the current level or if there are many actions per state and only a few states are revisited. The iterative deepening method has better worst-case performance than breadth-first search for many problems. Furthermore, the space requirements are reduced because the queue in breadth-first search is usually much larger than for depth-first search. The A^* idea can be combined with iterative deepening to yield IDA^* in which case the allowed total cost gradually increases in each iteration [9].

The value iteration algorithm iteratively computes optimal cost-to-go (or cost-to-come) functions over the entire state space. It differs from other graph search methods in two key aspects: providing the optimal path from any state to the goal (rather than given start state) and being computationally more expensive. Under some special conditions, the value iteration algorithm can be reduced to Dijkstra's algorithm. It can solve a vast collection of optimal planning problems, including those that involve variable-length plans, stochastic uncertainties, imperfect state measurements, and many other complications [11].

The algorithms mentioned so far start from the initial state and proceed toward the goal; hence the forward search. It is also possible on the other hand, to do a backward version of the tree search algorithm, i.e. from goal back to the start, or pursue a bidirectional approach that grows two search trees, one from the initial state and one from a goal state. Nevertheless, the algorithms' ideas in these approaches remain the same.

2.3. Fully observable versus partially observable states

Classical robotics often assumes that sensors can measure the full state of the environment which arguably is an unrealistic assumption. The lack of perfect sensors

has two ramifications. Firstly, robot control must be robust with respect to state uncertainty. Secondly, it must cope with future, anticipated uncertainty, and choose actions accordingly [12]. In order to behave effectively in a partially observable environment, it is necessary to use previous state information to aid in the disambiguation of the states. For instance all the previous information could be compiled into current state belief and expressed as probability distributions on state values. Then using system models, the future state belief can be predicted. Once measurement data is obtained, the state information at the time instant of measurement and further on is updated.

2.3.1. Estimation of environment's state

The interaction of a robot and its environment can be modeled as a coupled dynamical system, in which the robot can manipulate its environment by taking actions, and in which it can perceive its environment through sensor measurements. The dynamics of the robot and its environment are characterized in the form of two probabilistic laws: the state transition distribution, and the measurement distribution. The state transition distribution characterizes how state changes over time, possible as the effect of a robot action. The measurement distribution characterizes how measurement data depends on states. Both laws are probabilistic, accounting for the inherent uncertainty in state evolution and sensing. The Bayes filter assumes that the state is a complete summary of the past.

The belief of a system state is the probability distribution over the state, given all past sensor measurement data and all past controls. The Bayes filter is the principal algorithm for calculating the belief in robotics. The Bayes filter is recursive; the belief at time t is calculated from the belief and action at time $t-1$ and measurement data at t [12].

2.4. Planning under uncertainty

Classical robotics often assumes that the environment state is fully observable and the effects of actions are deterministic. In practice, however, the robot and its environments are stochastic in nature and thus uncertain.

Robots, like all systems, naturally have some limits to their capabilities. They can accomplish their delegated tasks within the reach but beyond that there is no guarantee. Additionally, there always will be some imprecision in robots' performance even within their limits. These inaccuracies could occur in form of measurement errors, motion errors and etc. The probability of these errors might be known or not [20]. The environments are in constant state of change. Thus no matter how accurate the surveys have been, there will be always inconsistencies in their maps [18]. Setting forth few examples, objects are where they should not or vice versa, topography of the

environment has changed or because of a heavy rain some lands surfaces are compromised significantly. Any of those might happen even as soon as the survey itself and remain unnoticed. On top of them, the problem gets worse as the time passes by [19].

Considering these facts, it is insufficient to plan a single sequence of actions and blindly execute it at run-time; but rather it is mandated that the robot receives measurement data, updates its state belief and reacts accordingly [12]. Note that in theory, the plan can be generated offline as a mapping from belief state to action space. Then the only run-time action is to receive measurement data, update the belief and then use the offline-computed control law to react. However, computing the plan offline may be impossible in practice due to computational complexity. Instead the problem may be solved online as a mapping of current belief to action space.

The planning problem may be formulated as sequence of decision problems such that the outcomes of actions are not known with certainty. In this work the state uncertainty is associated with the obstacle occupancies whereas the current and future locations of the robot are known without uncertainty. POMDP is one instrumental mathematical framework for modeling decision making in such circumstances and it is briefly introduced in the following.

2.4.1. Partially observable Markov decision process

Markov decision processes (MDP) serve as a basis for solving the more complex partially observable problems that ultimately is of interest. An MDP is a model, in which the decision maker takes as input the state of the world and generates as output actions, which themselves affect the state of the world. In the MDP framework, it is assumed that, although there may be a great deal of uncertainty about the effects of a robot's actions, there is never any uncertainty about the robot's current state –it has complete and perfect perceptual abilities [6].

A Markov decision process can be described as a tuple (S, A, T, r) where, S is the state representation, A is the allowed actions set, T is the transition function and r is the reward function. In this model, the next state and the expected reward depend only on the previous state and the action taken; even if we were to condition on additional previous states, the transition probabilities and the expected rewards would remain the same (Markov property). The core problem of MDPs is to find a policy for the decision maker; a function that specifies the action that the decision maker will choose when in any of the states. The problem can be solved in many ways such as by value iteration dynamic programming [13].

A Partially Observable Markov Decision Process, POMDP, is a generalization of Markov Decision Processes. A POMDP models a decision process in which it is assumed that the system dynamics are determined by an MDP, but the robot cannot

directly observe the underlying state. Instead, it must estimate a probability distribution over the set of possible states, known as the belief state, based on a set of observations and observation probabilities, and the underlying MDP [16]. The belief state is estimated as described in subsection 2.3.1.

A POMDP can be described as a tuple (S, A, T, r, OB, P_{ob}) where, OB is the set of possible observations and P_{ob} is a set of observation probabilities as functions of the state. In this form the decision maker's goal is to maximize expected discounted future reward. Solutions to this problem can be found in same way as for MDPs [13]. The difficult is in that the solution of MDP maps the state space to action space whereas that of POMDP maps the space of *state beliefs* on actions space. Therefore in practice POMDPs are often computationally virtually intractable to solve exactly, so methods have been developed that approximate solutions [14].

3. THE PROBLEM FORMULATION

3.1. The generic problem

In this section the generic problem of path planning is formulated using the POMDP framework introduced in section 2.4.1. Although the problem is quite general, it is still simplified to a degree. The assumption adopted for the simplification is described further into the section. In preparation of this Chapter reference [15] is extensively used.

In this work the area is modeled by a $N \times M$ square lattice with nodes coordinates (m, n), where $m \in [1: M], n \in [1: N]$. At any time step i, the state consists of the robot's location and obstacle occupancies, i.e. $S_i = \{L_i; O(m, n)\}$. As the main objective, the robot must get from its present location to a target location L_{target}. From there the robot then moves to idle position L_{idle}, where it stays. In a static environment it is assumed that the obstacles do not move. The $O(m, n)$ takes a binary value depending whether there is an obstacle at (m, n) or not. Note that both $O(L_{target})$ and $O(L_{idle})$ are equal to zero.

At each time step the robot may either move or observe. Hence the action space is $A = A_m \cup A_o$, in which A_m and A_o denote the motion actions and observation actions respectively.

Figure 3.1 demonstrates the allowed moves which are one step in the four directions, i.e. $A_m = \{E, W, N, S\}$. At the edges of the area certain moves may take the robot over the edge. Therefore those moves are excluded in path planning.

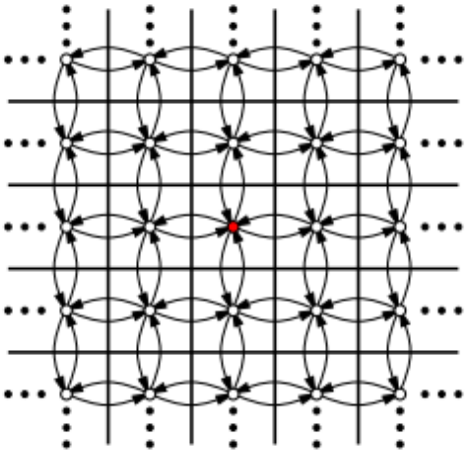

Figure 3.1: A part of the lattice; east, west, north and south moves are allowed [6].

It is assumed that within each time step, the robot is able to accurately localize itself. Therefore the transition model is deterministic. For example should the robot take an eastbound motion, its future location would have probability distributions as described in *Equation 3.1*.

Equation 3.1

$$P_{L_{i+1}|L_i,a_i=E}\left(L_{i+1}=(m_{i+1},n_{i+1})\,|\,L_i=(m_i,n_i),a_i=E\right)$$
$$=\begin{cases}1 & m_{i+1}=m_i+1,n_{i+1}=n_i\\ 0 & other(m_{i+1},n_{i+1})\end{cases}$$
$$a_i\in A$$

The robot has eight observation possibilities. The action space is defined as $A_o=\{(E,NE),(E,SE),(S,SE),(S,SW),(W,SW),(W,NW),(N,NW),(N,NE)\}$. Each observation is about the occupation of two locations by obstacles (see *Figure 3.2*) and thus providing two bits of information.

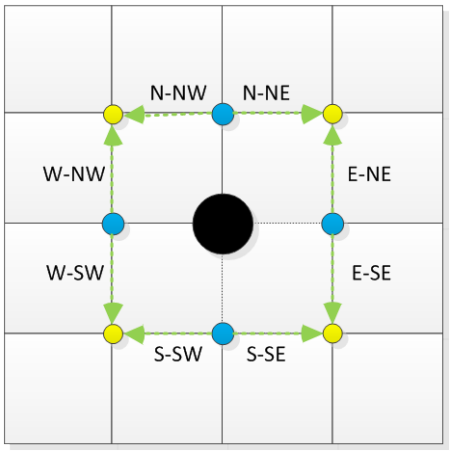

Figure 3.2: Observations' possibilities and observable points. The black circle represents the robot's current location.

The information obtained from observations is described for instance as in *Equation 3.2*.

Equation 3.2

$$p^{(E,SE)}_{S^{(E,SE)}_{i+1}|(L_{i+1},O\{m,n\})}\left(S^{(E,SE)}_{i+1}=(O(m_{i+1}+1,n_{i+1}),O(m_{i+1}+1,n_{i+1}+1))\,|\,(L_{i+1},O\{m,n\})\right)$$
$$=p^{(E,SE)}_{S^{(E,SE)}_{i+1}|(L_{i+1},O\{m,n\})}\left(S^{(E,SE)}_{i+1}(1)=O(m_{i+1}+1,n_{i+1})\,|\,(L_{i+1},O\{m,n\})\right)\cdot$$
$$p^{(E,SE)}_{S^{(E,SE)}_{i+1}|(L_{i+1},O\{m,n\})}\left(S^{(E,SE)}_{i+1}(2)=O(m_{i+1}+1,n_{i+1}+1)\,|\,(L_{i+1},O\{m,n\})\right)$$
$$=(1-P_{OB1})(1-P_{OB2})$$

Variables P_{OB1} and P_{OB2}, respectively denote the erring probabilities of the nearer and further points colored blue and yellow in *Figure 3.2*. The erring probabilities are independent on whether the site is occupied or not (symmetric erring).

Each node on the lattice is associated with a small negative value which defines its cost of occupation. In real maps these values are determined by taking into account the specifications of the area such as topography, surface material, or any minor obstacle [17]. In this work however, those values, denoted by $\tilde{r}(L_i)$, are synthetically generated random numbers, between zero and minus one. To simulate a more lifelike situation, sets of known static obstacles are scattered in the area map. If the robot moves to a location of an obstacle a collision occurs. The cost allocated to occupied locations, represented by r_{coll}, is significantly higher, by their absolute value, than at the other locations. Finally r_{end} represents the destination's reward while the *idle* state has zero reward. The values are stored in an array known as the reward function as described in *Equation 3.3*.

Equation 3.3

$$r(L_i, O(L_i)) = \begin{cases} \tilde{r}(L_i) & O(L_i) = 0 \\ r_{coll} & O(L_i) = 1 \\ r_{end} & L_{target} \\ 0 & L_{idle} \end{cases}$$

In some cases the obstacle occupancies are uncertain. If the occupancy probabilities from some time instant onward are fixed (no obstacle movement, no observations), the resulting path planning problem is deterministic with rewards given in *Equation 3.4.*

Equation 3.4

$$r(L_i, p(O(L_i))) = \begin{cases} \tilde{r}(L_i) & p(O(L_i)) < 0.55 \\ p(O(L_i)) \cdot r_{coll} + (1 - p(O(L_i)))\tilde{r}(L_i) & p(O(L_i)) \geq 0.55 \\ r_{end} & L_{target} \\ 0 & L_{idle} \end{cases}$$

Note that in this work, the collision reward is only applied to occupation probabilities equal or larger than 55 %. Furthermore if the time evolution of the occupation probabilities is known, these results in a deterministic problem with time varying rewards read as in *Equation 3.5*, which can be solved as value iteration on a time expanded graph.

Equation 3.5

$$r(L_i, p(O(L_i;t))) = \begin{cases} \tilde{r}(L_i) & p(O(L_i)) < 0.55 \\ p(O(L_i;t)) \cdot r_{coll} + (1 - p(O(L_i;t)))\tilde{r}(L_i) & p(O(L_i)) \geq 0.55 \\ r_{end} & L_{target} \\ 0 & L_{idle} \end{cases}$$

A final note on the reward function is that motions or observations have the same costs but as they are mutually exclusive, the robot remains at its present location while making an observation and adds a cost of $\tilde{r}(L_i)$ compared to moving along the shortest path.

The path planning problem is then formulated as in *Equation 3.6*.

Equation 3.6

$$V\left[L_i, \left\{P_i^{O(m,n)}\right\}_{m,n=1}^{M,N}\right] = \max_{\{a_{i'}\}_{i'=i}^{\infty} \in \{A_M \cup A_O\}_{i'=i}^{\infty}} E\left\{\sum_{i'=i+1}^{\infty} r\left(L_{i'}, O(L_{i'})\right)\right\}$$

$$P_i^{(O(m,n))} = p\left(O(m,n) = 1, i\right)$$

Where $P_i^{O(m,n)}$ describes the probabilistic current area map. The set consists of ones for known occupied locations, zeros for locations known to be not occupied and values belonging to $(0,1)$ when the occupancies of locations are uncertain.

The problem may be solved by receding horizon principle so that at time i, an entire optimal plan is generated and the first action is implemented accordingly. Then at time $i+1$, if new information is obtained through information channels the planning is repeated. Otherwise no re-planning is needed and the solution may proceed with the initial optimal plan.

Obviously the search space is huge. Depending on the area dimensions, both an offline solution and an online solution with a large horizon have to deal with immense search trees. Therefore, some approximations must be applied in order to solve the problem.

3.2. The simplified problem

3.2.1. Applied approximations

As described earlier in this chapter, the search space for the generic problem is vast due to the long time horizon till the *idle* state and the large number of occupation probabilities.

The first approximation is proposed to reduce the search space. Here, it is assumed that in the planning at time i, the search space is radically narrowed so that only for a short time horizon T_0 both observation and motion actions are considered after which only motion actions are deemed available. The problem is then scaled down to the one presented in *Equation 3.7*.

Equation 3.7

$$V\left[L_i, \left\{p_i^{O(m,n)}\right\}_{m,n=1}^{M,N}\right]$$

$$\approx \max_{\{a_{i'}\}_{i'=i}^{T_0-1} \in \{A_M \cup A_O\}_{i'=i}^{T_0-1}} E\left\{\sum_{i'=i+1}^{i+T_0} r\left(L_{i'}, O(L_{i'})\right) + \max_{\{a_{i'}\}_{i'=i+T_0}^{\infty} \in \{A_M\}_{i'=i+T_0}^{\infty}} \sum_{i'=i+T_0+1}^{\infty} r\left(L_{i'}, O(L_{i'})\right)\right\}$$

$$= \max_{\{a_{i'}\}_{i'=i}^{T_0-1} \in \{A_M \cup A_O\}_{i'=i}^{T_0-1}} E\left\{\sum_{i'=i+1}^{i+T_0} r\left(L_{i'}, O(L_{i'})\right) + V^{(0)}\left[L_{i+T_0}, \left\{p_{i+T_0}^{O(m,n)}\right\}_{m,n=1}^{M,N}\right]\right\}$$

In the equation the variable $V^{(0)}$ is the optimal value of being at location L_{i+T_0}.

The problem can now be solved with on-line dynamic programming. In an exhaustive search until the short time horizon T_0, all possible action sequences are considered and robot's location and occupation probabilities are calculated resulting in some $L_{i+T_0}, \left\{P_{i+T_0}^{O(m,n)}\right\}_{m,n=1}^{M,N}$, noting that occupation probabilities are weighted with their prior information. Then the corresponding $V^{(0)}$ part is solved and the value is back propagated to i. Although solving for $V^{(0)}$ is a straight forward problem, the computational burden in *Equation 3.7* comes from its many repetitions. As there are four move actions with no associated data values and eight observation actions, each with four possible data outcomes, the branching factor of the on-line optimization is 36. Hence the number of $V^{(0)}$ problems to be solved for a given T_0 is 36^{T_0}. An extreme approximation is to replace occupancy information in the $V^{(0)}$ problem with the original map. The transformed problem is then read as in *Equation 3.8*.

Equation 3.8

$$V\left[L_i, \left\{p_i^{O(m,n)}\right\}_{m,n=1}^{M,N}\right] \approx \max_{\{a_{i'}\}_{i'=i}^{T_0-1} \in \{A_M \cup A_o\}_{i'=i}^{T_0-1}} E\left\{\sum_{i'=i+1}^{i+T_0} r\left(L_{i'}, O(L_{i'})\right) + V^{(0)}\left[L_{i+T_0}, \left\{p^{(O(m,n),map)}\right\}_{m,n=1}^{M,N}\right]\right\}$$

In its current form, solution of the problem till T_0 demands the same exhaustive tree search as before. However, the $V^{(0)}$ part could be solved off-line once and the end rewards for on-line optimization could be read from a lookup table.

3.2.2. Offline deterministic problem

This path planning problem with known fixed obstacle occupancies and action space limited to only motion actions is a straight forward deterministic problem. The optimal value problem in *Equation 3.8* is then reformulated and read as in *Equation 3.9*.

Equation 3.9

$$V^{(0)}\left[L_i\right] = \max_{\{a_{i'}\}_{i'=i}^{\infty} \in \{A_M\}_{i'=i}^{\infty}} E\left\{\sum_{i'=i+1}^{\infty} r\left(L_{i'}\right)\right\} = \max_{a_i}\left[r\left(L_{i+1}(a_i)\right) + V^{(0)}\left[L_{i+1}(a_i)\right]\right]$$

To solve the problem, value iteration algorithm may be employed. First the equation is solved with an arbitrary $V^{(0,k)}$ on the right hand side resulting in some $V^{(0,k+1)}$ on the left hand side. Then the calculation is repeated with the obtained value and the iterations will continue till the convergence. The value iteration algorithm guarantees the convergence in finite number of steps.

4. THE ORIGINAL PROGRAM

The proposed solution to the path finding problem consists of two methodologies: an offline deterministic problem and an online stochastic one. Hence in this program, the offline path planner (OPP) solves the deterministic part of the problem using value iteration algorithm; whereas the online path finder (OPF), utilizes dynamic programming method to solve the POMDP body of the problem. In this arrangement, the OPP is a sub-process of the OPF. Schematic of the entire program is depicted in *Figure 4.1*.

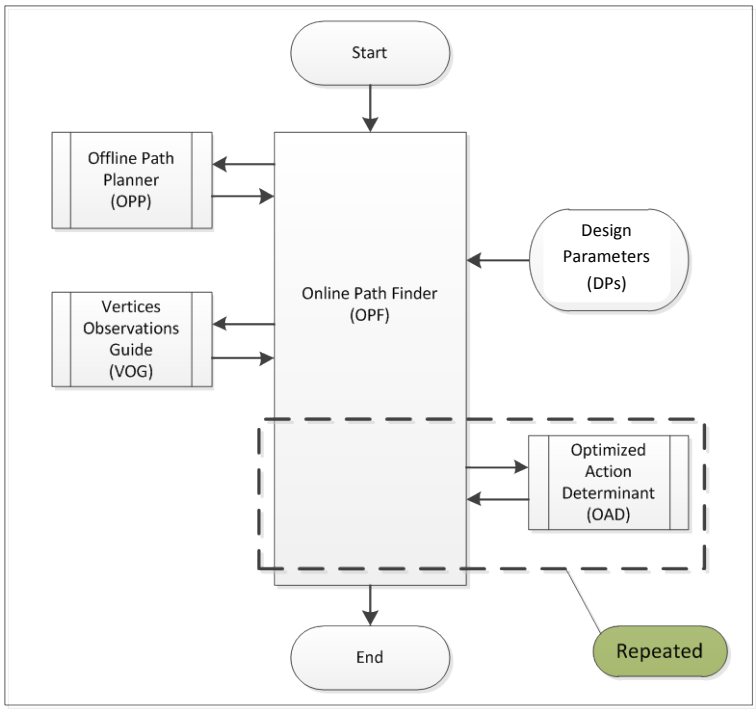

Figure 4.1: The original program's schematic.

In this chapter, the implementations of program components are described in order that they appear in the structure.

4.1. Offline path planner (OPP):

The offline path planner finds a deterministic optimal path to the destination. Based on the ideas developed in section 3.2, it also provides a table of optimal end rewards for the online stochastic part of the problem. The aim of this section is to provide an insight into the implementation of the solution. The codes for this module can be found in Appendix 1.

The synthetic environment, through which the robot tries to find its way, was described in section 3.1. To create the model, the OPP acquires the dimensions of the area and accordingly generates site rewards as random numbers between zero and minus one. Then the nodes known to be occupied with static obstacles are assigned with a very large negative reward. Finally the destination point is marked by a very high reward and the *idle* state node is marked zero. These values, stored in an array, serve as the reward function.

In certain circumstances, the OPP module may be re-executed. Therefore the function has the capability to update the reward function to solve the path finding problem. Any obstacle occupation change in the area map is applied to the reward function according to *Equation 3.4*.

The plan is a set of allowed move actions at each location by the robot. In this work along with the square lattice, there are at most four moves from any given node (north, east, south and west). Note that the freedom of motion is reduced at the edges and corners to three and two move actions, respectively. Also form the target location the robot moves to the *idle* site, where it stops all operations. To incorporate such action space in the program a matrix, named Steps, is generated. This matrix specifies where the robot will reach by taking any of the quartet moves from any given point.

The optimal path problem formulated in *Equation 3.9* can now be solved with value iteration algorithm. In the OPP, the optimal value is the sum of the reward obtained making moves and optimal value in the previous iteration, noting that in the first iteration the optimal value is equal to zero. In the later iterations however, the optimal value consists of the highest values from the previous iteration. The iterations will continue for 10000 times although technically according to the algorithm $N \times M$ times should suffice. Then the highest value in each step is cross-referenced with its respective travel, thus extracting optimal actions. Finally starting from the current position of the robot, the optimal deterministic path to the goal is determined from the optimal actions and the step matrix together. The total reward collected from is the sum of the values associated with each node on the optimal path.

4.2. Vertices observation guide (VOG):

The robot gathers information about its surroundings by making observations. The observation mechanism has been described earlier in section 3.1.

In the program implementation in order to identify the observed nodes, the robot utilizes a simple chart, generated by the VOG component. From this table, knowing the current position, from which an observation is being made, and also the direction of the surveillance, which is marked with a number from one to eight, the robot is able to recognize the two observed locations. The codes for this module can be found in Appendix 3.

4.3. Online path finder (OPF):

This section presents the main building block of the program. The OPF mainly prepares the elements of the path finding problem framework for another module which makes the decisions. It is in fact a virtual path finder robot which perceives its environment, makes decisions, carries out actions and updates its state. The codes for this module can be found in Appendix 2.

As explained in section 3.2.1, the robot relies on the POMDP framework with some approximations to devise the Equation 3.8. The problem is then solved with dynamic programming method. There are some parameters required to be defined for problem formulation and its solution method, called the design parameters (DPs) which are introduced in the following.

The first step is to draw a primary and deterministic plan based on known conditions of the area (topography, obstacles and etc.), hence an offline planning. This deterministic plan is the lookup table for the approximated optimization problem presented in section 3.2.1. The offline planner may be called several times if required (more on the issue in sections 5.4 and 5.5).

Depth or the short time horizon (STH) is the length of tree search ahead in time. Within this length full action space is applicable whereas after it the action space is limited to motion actions only.

Local area dimensions define a sample rectangular area surrounding the current position of the robot that the program confines its calculations to it, per instructions of dynamic programming solution. Local area is chosen such that within the given depth the robot will not leave it.

Collision cost is a factor that the robot applies in its calculations to prioritize its choices of actions within the STH length. High cost of collision would lower the expectation of getting higher reward by taking motion steps and consequently the observations become more prominent. On the other hand, a low COC would decrease the cost of motion steps to a point where they are slightly less costly than staying in the same location, which is the robot's state while observing, and hence more motion steps. In the real world the COC depends on various factors ranging from the nature of the obstacle to the quality of the impact. In this work however, the parameter is determined by looking into robot's behavior and statistically evaluating its responses (see section 7.1.1). Suitable COC are assigned based on the STHs.

Observation probabilities set defines the erring probability distribution of the observations made by the robot. It introduces the partially observable part to the POMDP problem formulation.

Global occupancy represents the belief state in the POMDP framework explained in sections 2.4.1 and 3.1. In this work the occupancy belief is naively adopted at first, i.e. it is believed that each node has the same probability of occupancy as the other. Therefore an array of same probabilities is produced, where each member is associated with a particular location on the map.

In the absence of a real environment and to test the robot, the surroundings should be synthesized. The base area, where complete knowledge of the environment is assumed, is generated in the OPP and at the beginning of the process. However, the random obstacles to be avoided are launched to the environment in the OPF, noting that the robot is not informed about whereabouts or number of them. In general the obstacles in the environments are of different sorts; some are stationary and others are dynamic. However, the robot with the present observation mechanism and program cannot differ between the two. Thus all objects are treated as either all static or all dynamic obstacles.

Preparations before commencing the process of path finding continues with generating the vertices observations guide (see section 4.2), setting some practical counters, defining the number of allowed steps and the starting position of the robot.

The optimal path in this work is variable-length plan meaning that the number of steps (iterations) that takes to solve the problem is not fixed [6]. Statistical analysis has shown that if the online stochastic problem is to be solved, it may take up to five times the number of steps required for the offline deterministic problem. Of course having broader action space with observations and not proceeding when observing in the online stochastic process, account for much bigger portion of the difference; however frequency of encounters with random obstacles has an impact on the number of steps as well.

The path finding process is repetitive set of actions and therefore they are performed in a *while loop*. The loop carries on as long as the robot has not reached immediate vicinity of destination. The procedures within the loop are described in the following.

In a dynamic environment the OPF starts its loop actions with calling the RW module to get receive the updated location of the walkers. It is to be noted that positions of the obstacles are not reported to the robot and it only perceives them through its observation actions.

In offline planning mode the whole area is being considered for the optimal path. In the online part on the other hand, this notion would lead to a huge computational burden. The dynamic programming suggests a search tree limited to smaller vicinity, the dimension of which was determined earlier, around the current position of the robot. Within this limited neighborhood the robot solves the decision making problem for the best course of actions. To map the local area around the robot, neighboring nodes within the specified dimensions are identified. The local state of the robot which consists of the

location of the robot its local belief state, local rewards and local allowed actions are then generated.

At this point all the elements required to construct the Equation 3.8 are ready. The next action in the loop for the program is to summon the optimum action determinant (OAD) which actually solves the problem. The implementation of the module is discussed in section 4.4. After execution, the function returns the best immediate action that the robot should take.

Depending on the action that the function instructs, the robot's state or the belief state will be altered. If the optimal action is to move, the belief state about obstacles remains unchanged and the state is updated by the new position of the robot; which is set by utilizing the previous coordinates, the suggested direction of the motion, noting that in that the robot's future location is fully predictable. Conversely, if the optimal action is an observation, the robot should look in the direction specified by the OAD and then update the belief state accordingly. In this work the observations are virtual simulations, in which the observation error probabilities are embedded. The results of these simulations along with previous belief state, update the belief utilizing Bayes formula. In a static environment, the update process consists of marking the observed nodes as occupied or vacant by increased or decreased probability respectively.

With the necessary adjustments made to the state belief, the loop reaches to its turning point so it continues from the top if the robot is still in its midst way. Otherwise, when the robot is immediate neighborhood of the goal, the program exits the loop and continues to its destination approach protocol. The protocol assumes that there is no obstacle present in immediate vicinity of target location and thus the program reduces to choosing a way out of two according to the reward map. Finally when the destination is reached the robot assumes an idle position and stops.

4.4. Optimum action determinant (OAD):

The Optimum action determinant is the module that solves the Equation 3.8. The module recursively calculates the optimal action for the current time instant by considering all the possible action sequences beginning from the time instant considered till the end of the fixed depth. At the end of the short time horizon last move of the robot, the online solution uses the optimal values (cost-to-go) of the offline solution calculated for that grid cell which then finally gives the values for each possible action at current time instant. Only the next action is ever executed from the planned action sequence after which the online optimization is again considered from the beginning at the next time step by utilizing the possible new measurement information obtained if observation action is taken.

The OAD is designed to address motion actions with numbers one to four, corresponding to four directions while returning numbers five to 12 for corresponding observation actions. The codes for this module can be found in Appendix 4.

5. ANOMALIES AND RECTIFICATION

The robot's behavior with the current program demonstrates some anomalies. In this Chapter, first forms of the unwanted behavior and their characterization method are covered. Next the sources of errors are discussed. The Chapter then ends by proposing methods and implementing them to rectify such errors.

5.1. Unwanted behavior and characterization

Numerous tests have shown that the robot's unwanted behavior is mostly manifested in the form of different infinitely repetitive patterns of observation or motion and no advancement along the path. These patterns are discussed in their respective sections 5.1.1 and 5.1.2. Other notable problem is the issue of incautious motion, a hazardous trend which is unraveled in subsection 5.1.3.

5.1.1. Observation loops (OLs)

It has been witnessed in several occasions that the robot stands still and endlessly observes the same direction or different directions. The robot encounters such problem in narrow corridors or neck-like areas and more often when its path is blocked in the front by an obstacle and on the side by walls. This anomaly is of two specific types: single directional observation loop and multiple directions OL.

Single directional observations loop (SOL)

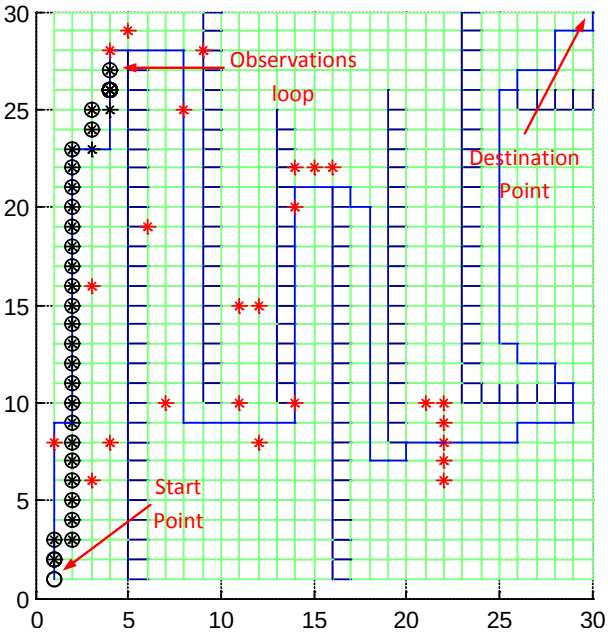

Figure 5.1: An instance of a single directional observation loop.

Figure 5.1 depicts an instance of a single directional observation loop. Note that the robot is incapable of finding a way around with STH equal to three, because of the obstacle in front and the wall to the right.

To further clarify the situation portrayed in the Figure, it is accompanied by *Table 5.1*, in which the initial position (Pos. i), the action performed by the robot (Action) and the destination location after taking an action (Pos. j), are listed.

Table 5.1: States and actions.

No.	Step	$Pos._i$	Action	$Pos._j$
1	80	116	$O:(N,NE)$	116
2	81	116	$O:(N,NE)$	116
3	82	116	$O:(N,NE)$	116
4	83	116	$O:(N,NE)$	116
5	84	116	$O:(N,NE)$	116
6	85	116	$O:(N,NE)$	116
7	86	116	$O:(N,NE)$	116

Theoretically, if the situation in a certain location remains unchanged, Bayes rule assures us that by looking into its direction repeatedly, the accuracy of the information about that position will be increased. *Figures 5.2* and *5.3* demonstrate examples of changes in occupancy probability of a single point which is being observed several times. The data for these figures is obtained by inducing observation loops to the program; and then executing it for at least 20 rounds while recording occupancy probability of specific points after every observation. Note that the points subject to these experiments are the further nodes from the robot that have higher uncertainty.

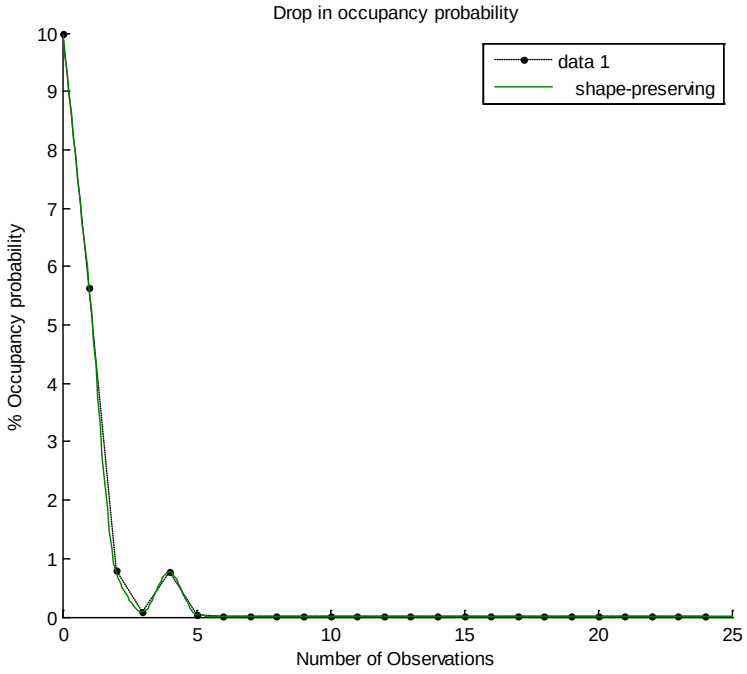

Figure 5.2: Drop in occupancy probability of a single point by repeated uncertain observations. Note that the fluctuation due to a false positive on the fourth observation is settled well before the tenth observation.

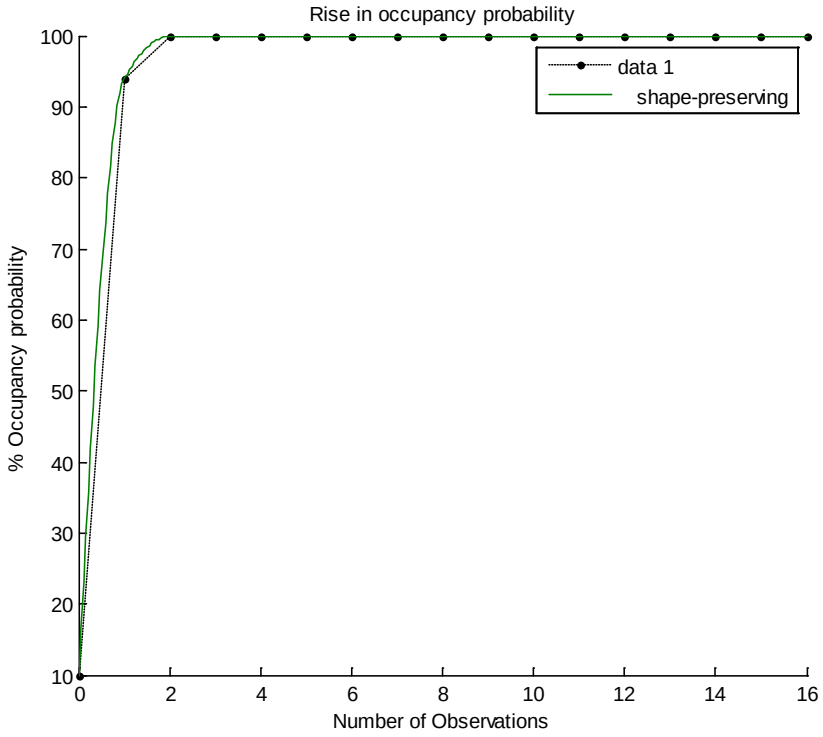

Figure 5.3: Rise in occupancy probability of a single point by repeated uncertain observations.

The Figures clearly illustrate that after around five observations, the occupancy belief practically reaches its final value and hereafter further observations made in that particular way are redundant.

Of course in reality the notion of static environment is naive and perilous; yet, considering our setting, since the likelihood of variation is low enough and by accepting some risk it can be adopted. Still, it is wise to increase the number of redundant observations as a precautionary measure. In this work, seven redundant observations in one direction are allowed before characterizing the robots behavior as a loop.

Multiple directional observation loop (MOL)

Another type of OL occurs when the program commands the robot to observe its surroundings over and over again without taking any further action. The difference here, comparing to the situation above, is the direction of observations which is not necessary the same. An instance of such event is portrayed in *Figure 5.4* and supplemented by *Table 5.2*.

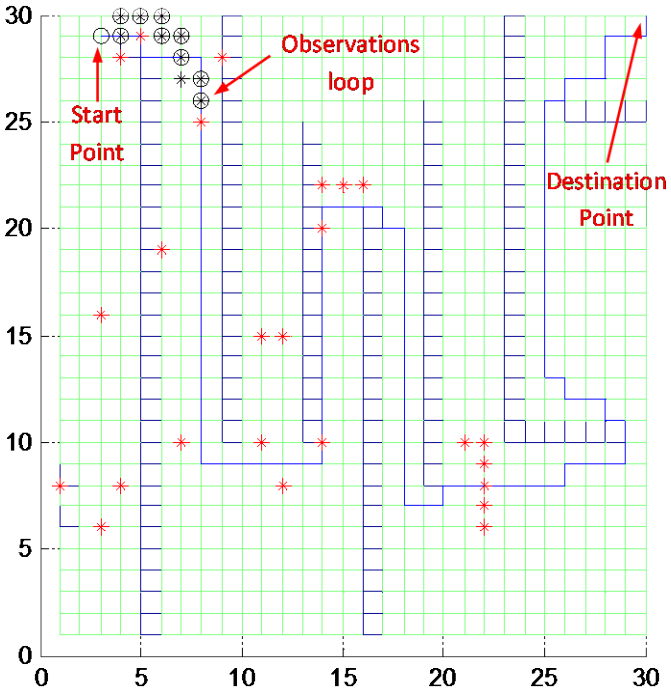

Figure 5.4: An instance of a multiple directional observation loop.

Table 5.2: States and actions.

No.	Step	Pos.$_i$	Action	Pos.$_j$	No.	Step	Pos.$_i$	Action	Pos.$_j$
1	286	525	$O:(W,NW)$	525	19	304	525	$O:(E,NE)$	525
2	287	525	$O:(W,SW)$	525	20	305	525	$O:(E,NE)$	525
3	288	525	$O:(W,NW)$	525	21	306	525	$O:(S,SW)$	525
4	289	525	$O:(W,SW)$	525	22	307	525	$O:(W,SW)$	525
5	290	525	$O:(W,SW)$	525	23	308	525	$O:(W,SW)$	525
6	291	525	$O:(W,SW)$	525	24	309	525	$O:(W,NW)$	525
7	292	525	$O:(E,SE)$	525	25	310	525	$O:(W,NW)$	525
8	293	525	$O:(W,SW)$	525	26	311	525	$O:(W,NW)$	525
9	294	525	$O:(S,SW)$	525	27	312	525	$O:(W,NW)$	525
10	295	525	$O:(S,SW)$	525	28	313	525	$O:(W,NW)$	525
11	296	525	$O:(S,SW)$	525	29	314	525	$O:(W,NW)$	525
12	297	525	$O:(S,SW)$	525	30	315	525	$O:(E,NE)$	525
13	298	525	$O:(S,SE)$	525	31	316	525	$O:(E,NE)$	525
14	299	525	$O:(E,SE)$	525	32	317	525	$O:(E,NE)$	525
15	300	525	$O:(E,NE)$	525	33	318	525	$O:(E,NE)$	525
16	301	525	$O:(E,NE)$	525	34	319	525	$O:(E,NE)$	525
17	302	525	$O:(E,SE)$	525	35	320	525	$O:(E,NE)$	525
18	303	525	$O:(E,NE)$	525					

Following the same logic as stated in the SOL case, the threshold for characterization of the behavior as a loop, including redundancy, is set on 35.

5.1.2. Motion loops (MLs)

Motion loops are identified as the occasions in which the robot performs the same motion sequence again and again and not progressing in its path. Compared to observations loops, they are not very likely to occur; nevertheless they have been observed. Several tests have revealed that in motion loop cases, the incorrect perception of the robot from the environment, i.e. seeing obstacles where they are not, coupled with short time horizon, are the root causes of the problem. As anticipated, these errors are also often happening in narrow corridors or neck-like areas. Reciprocating motions and circular motions are the two categories of the motion loops which will be discussed.

Reciprocating motions

Figure 5.5 is an example of reciprocating motions accompanied by *Table 5.3*.

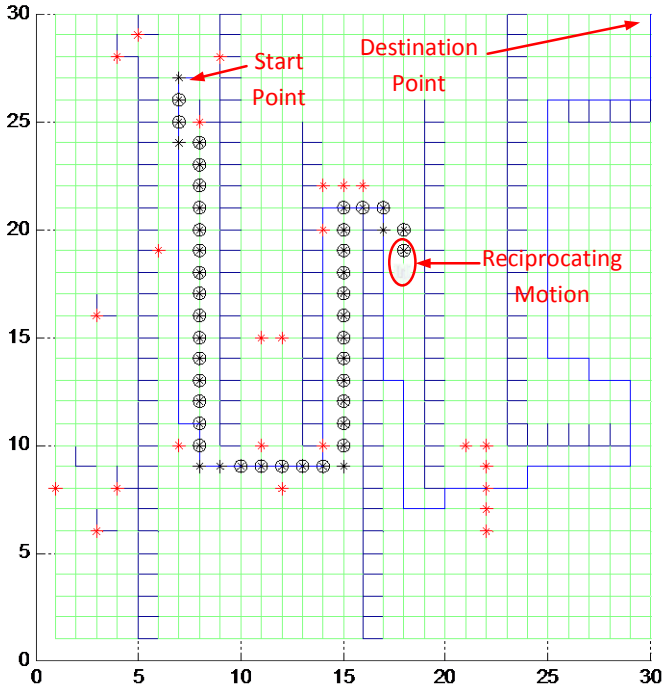

Figure 5.5: An instance of reciprocating motions.

Table 5.3: States and actions.

No.	Step	Pos.$_i$	Action	Pos.$_j$
1	321	528	$M:N$	529
2	322	529	$M:S$	528
3	323	528	$M:N$	529
4	324	529	$M:S$	528
5	324	528	$M:N$	529
6	326	529	$M:S$	528
7	327	528	$M:N$	529
8	328	529	$M:S$	528

As it is clear in the *Table 5.3*, the robot has fallen in a back and forth motion loop. The pattern is easily identified by analyzing the unique values of either action vector or

position vector; in such an event, i.e. reciprocating motion, the unique vector each would have a length equal to two.

The question remains is the number of redundant motions allowed before characterization. In reviewed cases there have been legitimate circumstances that the robot had to be in two consecutive locations for maximum of three times in six serial steps. However, following the same footsteps for the fourth time has been acknowledged as an indicator of a loop.

Circular motion loops

Another form of motion loops which may happen, but not yet observed, are those of circular nature. It must be noted that these motions are termed circular in the sense that they form closed loops. These circular motions are harder to detect due to their variety of shapes and sizes. Nevertheless, the possibility of them happening ever increases as the passage ways in the environment get more crowded with dynamic obstacles.

In this work, the sole circular motion loop covered is a four step loop, i.e. moving on corners of a square as displayed in *Figure 5.6*. It must be noted that this Figure is hypothetical and it is only demonstrated to make the concept clear.

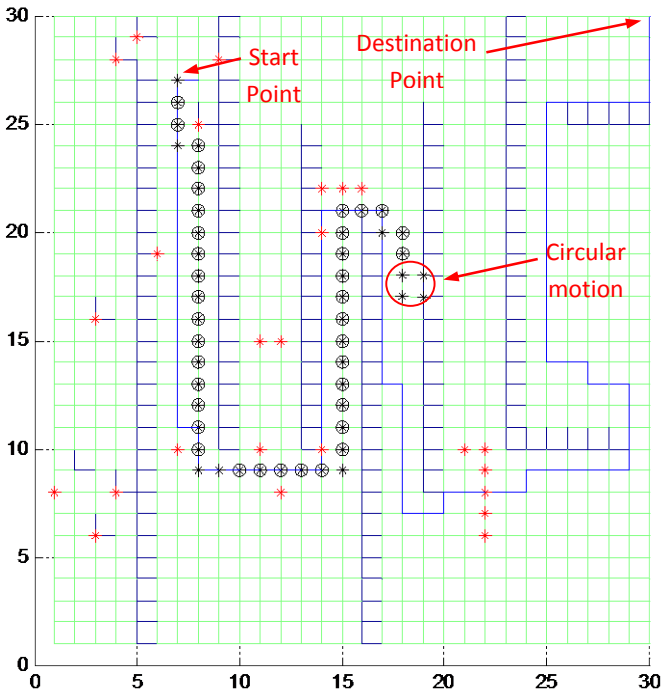

Figure 5.6: An instance of circular motions. Note that this image is hypothetical.

Same as reciprocating motions, this loop is also being identified by analyzing the current and past positions of the robot. If the robot has been occupying the same four locations in its past 16 steps, the pattern is characterized as a circular motion loop. More sophisticated pattern recognition techniques are required in order to distinguish other types of circular motions, which are out of scope of this work.

5.1.3. Incautious motions (IMs)

The robot's wrongful presumption of open space is concerning. It has been witnessed that in some cases, where the robot has enough room to move and has not discovered any obstacle in its immediate past observations, even if there have been some, it proceeds incautiously and without evermore observing its surroundings like depicted in *Figure 5.7*.

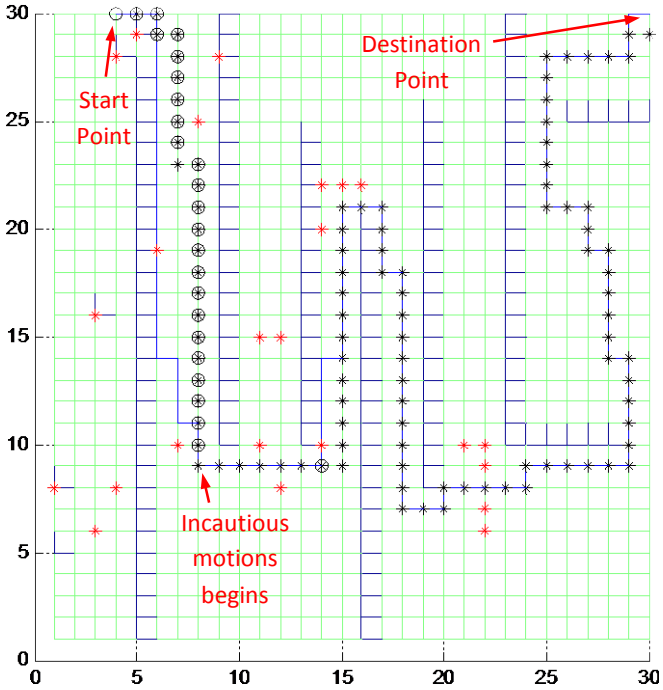

Figure 5.7: Instances of incautious motions.

The nature of this tendency is not yet fully understood which makes it harder to characterize the anomaly. Nevertheless, the robot must observe its surroundings in reasonably spaced steps. According to observation mechanism's design, even in favorable circumstances, the robot would have no reliable observation based information about the passage ahead after taking three consecutive and not retracting motion steps. Therefore, minimum of one observation is required in every three steps. Any occasion in which the robot deviates from this minimum requirement, may be characterized as a case of IM.

5.2. Sources of the errors

The unwanted behavior described in the previous sections is due to two sources. The first one is the approximations applied to the generic problem and the second is inappropriate design parameters. The latter one, however, depends to a great extent on the approximations, so if no approximations were made the design parameters would not have caused any issue. For instance the effect of the collision cost on the robot's

behavior is far less with higher short time horizon (see *Table 7.2*). Therefore, the approximations are considered as the only source of error in this section.

In order to solve the generic problem of *Equation 3.6*, two approximations were made to it. The first one was to assume that the search space radically shrinks at planning time so that full action space is only considered for a short time horizon and after that only motion actions are available. The second was to replace occupancy information in the latter part of the problem with the original map rather than using the full information obtained till that point in time.

The first approximation limits the robots options and causes failure in accurately anticipating steps beyond STH and thus inability to find a maneuvering solution. While a higher STH should eliminate this problem to some extent, it would also increase the computational costs more than exponentially, which obviously counters the objective of the approximation in the first place.

The shortcoming of the second approximation is that since it rates the paths after the short time horizon based on the original map, the results may favor paths that are known by on-line observations to be blocked with high probability. In particular, if the optimal path based on the original map goes through a narrow passage and then this passage is observed to be blocked, the solution does not find an alternative path, unless it is T_0 move steps away or closer.

5.3. Command controller (CC):

In this work, based on fundamentals of feedback control, a Command controller is designed to look for suspicious patterns in the robot's behavior and call for respective corrective action upon recognizing those patterns. A rough schematic of the CC's function is depicted in *Figure 5.8*.

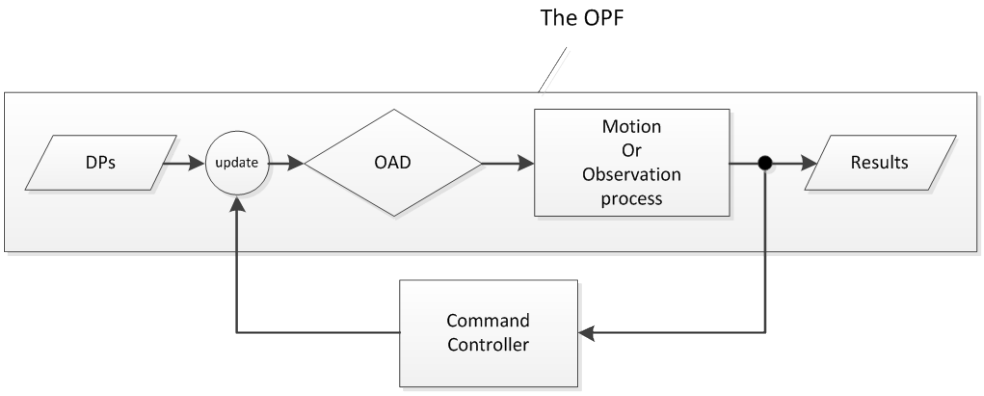

Figure 5.8: Outline of the command controller's function.

The controller unit is introduced into the OPF just before the turning point of the loop (see *Figure 5.9*), where the state is updated with the latest action.

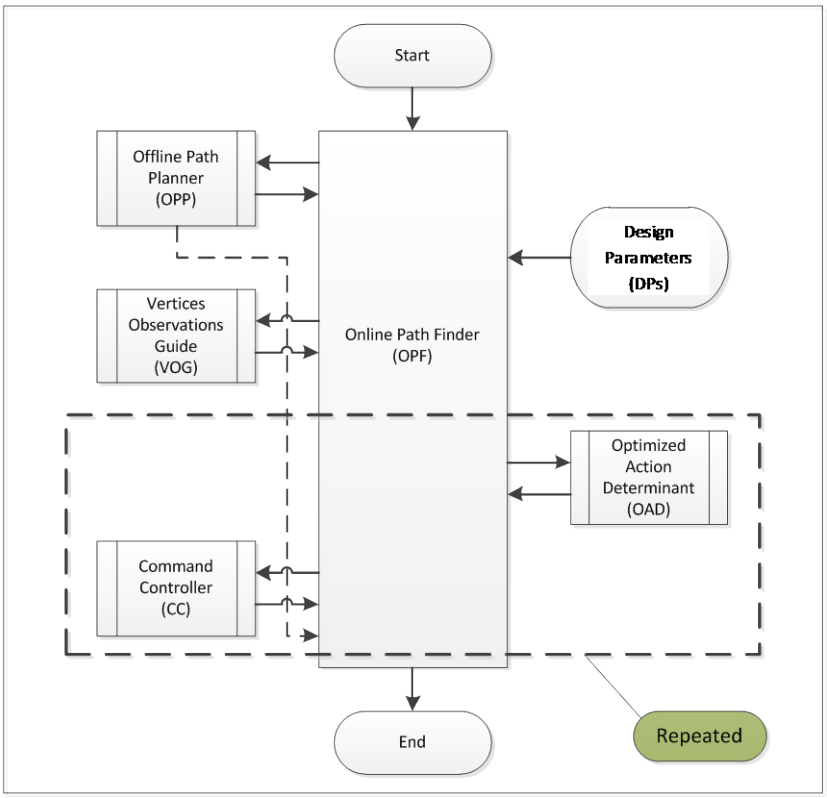

Figure 5.9: The new program's schematic with command controller.

The controller issues warnings which are then translated into suitable responses within the online pathfinder itself. The assignments of reactions are explained in sections 5.5 through 5.7.

In contrast, measures taken to eliminate the unwanted behavior may burden the robot's operations in normal settings. Therefore it is logical to revise the rectifications once they were not required, hence another role assigned to the command controller. The concept and its methodology are described in 5.8

Command controller is perhaps the most significant module in this work without which the robots performance suffers gravely; a proof of that is the overall success rate which is discussed in subsections 7.1.2 and 7.1.3. The codes for the CC unit can be found in Appendix 5 and the corresponding actions added to the online path finder are covered in Appendix 6.

5.4. Corrective measures

The robot's behavior may be manipulated through certain adjustments of its parameters. These parameters are used to solve the path finding problem. Altering them may lead to more accurate solution and consequently more desirable behavior of the robot.

The parameters which are open for adjustments range from the reward function to the COC and the STH. Change in each of the values of these parameters has its consequence in the robot's output.

The approximated *Equation 3.8* always uses the initial reward function which was constituted to be problematic. In such cases, solving the offline planning problem again, i.e. executing the OPP, with updated obstacle information can be applied to determine the actual reward function. The benefit is that if narrow passages are observed to be blocked, alternative paths, differing widely from the optimal path according to the original map, can be found.

The effect of collision cost on the robots behavior was introduced in section 4.3 and more statistics can be found in section 7.1.1. In brief a carefully adjusted COC can cause the robot to favor observations over motions. That is an effect that can be utilized to shape the robots behavior to its advantage.

The importance of the STH and its role in the robot performance cannot be overstated. While a low STH will result in expedited operations of the robot, it also reduces the accuracy and effectiveness of the decisions made, thus leaving the program vulnerable under problematic conditions. An increased STH can then be a solution in the settings where the robot needs a border perspective for instance in order to overcome an obstacle.

5.5. Rectification of loops

Characterizing the errors makes it relatively easy to detect them. The CC recognizes the single directional observation loops if in its past seven steps, the robot has been staying in the same location while performing observations in the same direction. Similarly it detects the multiple directional observation loops when the robot serially performs more than 35 observations in any direction. As for the motion loops, the command controller is triggered if the robot has been travelling between no more than two immediate neighboring locations within eight consecutive time steps or if it has been occupying the same four locations in its past 16 steps. These patterns produce reciprocating motions or circular motions warnings respectively.

Loops, of any kind, are indicatives of blocked paths. The robot obviously has not been able to bypass these blocks with lesser number of steps than the STH. In such conditions updating the end point rewards via new OPP execution, has become known to be effective in more than 90 % of times. Furthermore increasing the STH is another approach that will most probably solve the problem. Its success rate has been estimated at over 99 %. However, the achievement comes with a cost. Raising the STH escalates the computation time dramatically. Thus the methodology utilized consists of two parts. The program initially tries to resolve the loop errors by rerouting. However, should the

OPF fails to obtain a convenient response in few trials, the secondary loop controller intervenes and the STH is adjusted to a higher value.

5.6. Rectification of IMs

The controller module issues IM warning if the observations are accounted for less than 30 % of the total actions performed in past seven steps or the robot has serially moved three steps.

In course of an IM, the first response would be to persuade the robot to make more observations. As it has been stated before, increasing the COC is one way to achieve such a reaction. The amounts of incremental changes are determined based on statistical analysis as will be discussed in section 7.1.1 and *Table 7.2*. A desirable side effect of this process is that by pushing the robot to observe more between its motions, motion loops are also prevented from happening more often.

Although this approach is estimated to be successful in a little more than 98 %, it is still not guaranteed. It is also absolutely essential not to force the robot into observations only mode. Similar to the loop cases the secondary loop controller (SLC) intervention is foreseen to increase in the STH as means to overcome resistant IMs.

5.7. Secondary loop controller (SLC)

During the test runs, error detecting algorithms demonstrated acceptable performance and left no problem undetected; hence dramatic improvement of the robot's overall operation was achieved. However, it was made evident by some trials that reacting to a warning alone and neglecting the past corrections might be a source of problem by itself.

An example is an occasion in which different abnormalities occur so close to each other that reacting to them all, especially with the same corrective response, would be incorrect. In other instances, resistant errors, i.e. repeating instances of the same error in a short period of time, are yet other indicators of detecting algorithms' limitations. *Figure 5.10* and *Table 5.4* depict such situation. As it is apparent in the graph, after taking one motion step and few observations, the robot has fallen into a loop and the SOL warning is issued. However, the warning and its consequential corrective measure fails to achieve any result. In further steps, the problems keep reoccurring despite of being detected and dealt with every time.

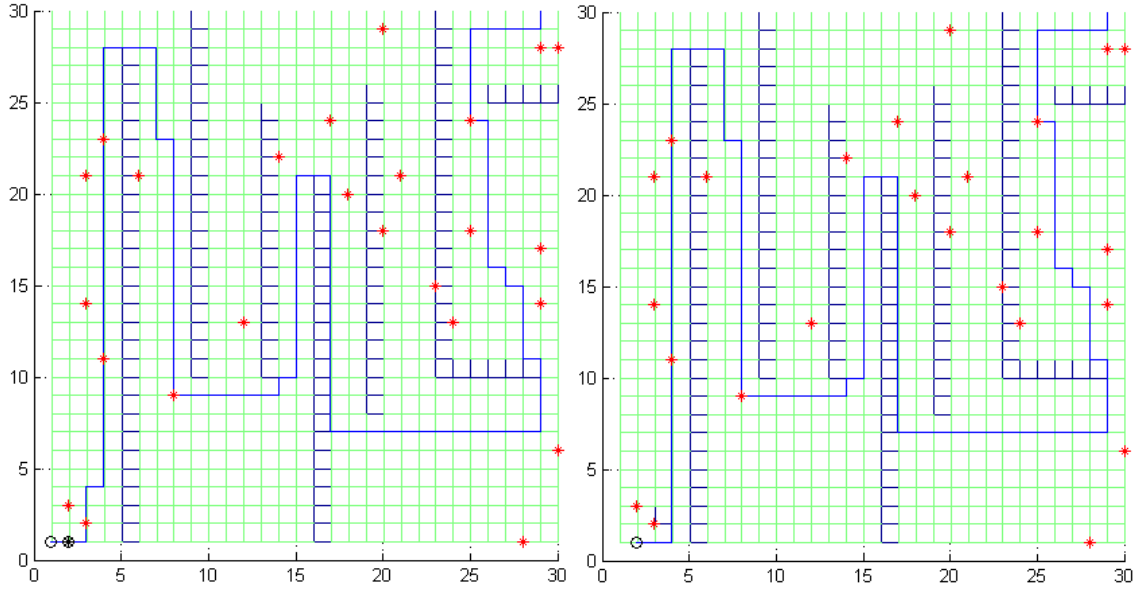

Figure 5.10: Instances of persistent observation loops.

Table 5.4: Execution results during the resistant observation loops.

No.	Step	$Pos._i$	Action	$Pos._j$	War.	No.	Step	$Pos._i$	Action	$Pos._j$	War.
1	1	1	$O:(S,SW)$	1	0	28	6	31	$O:(S,SE)$	31	0
2	2	1	$O:(W,SW)$	1	0	29	7	31	$O:(S,SE)$	31	SOL
3	3	1	$O:(S,SW)$	1	0	30	1	31	$O:(S,SE)$	31	0
4	4	1	$M:E$	31	0	31	2	31	$O:(S,SE)$	31	0
5	5	31	$O:(S,SE)$	31	0	32	3	31	$O:(S,SE)$	31	0
6	6	31	$O:(S,SE)$	31	0	33	4	31	$O:(S,SE)$	31	0
7	7	31	$O:(S,SE)$	31	0	34	5	31	$O:(S,SE)$	31	0
8	8	31	$O:(S,SE)$	31	0	35	6	31	$O:(S,SE)$	31	0
9	9	31	$O:(S,SE)$	31	0	36	7	31	$O:(S,SE)$	31	SOL
10	10	31	$O:(S,SE)$	31	0	37	1	31	$O:(S,SE)$	31	0
11	11	31	$O:(S,SE)$	31	SOL	38	2	31	$O:(S,SE)$	31	0
12	1	31	$O:(E,NE)$	31	0	39	3	31	$O:(S,SE)$	31	0
13	2	31	$O:(E,NE)$	31	0	40	4	31	$O:(S,SE)$	31	0
14	3	31	$O:(E,NE)$	31	0	41	5	31	$O:(S,SE)$	31	0
15	4	31	$O:(E,NE)$	31	0	42	6	31	$O:(S,SE)$	31	0
16	5	31	$O:(S,SE)$	31	0	43	7	31	$O:(S,SE)$	31	SOL
17	6	31	$O:(S,SE)$	31	0	44	1	31	$O:(S,SE)$	31	0
18	7	31	$O:(S,SE)$	31	0	45	2	31	$O:(S,SE)$	31	0
19	8	31	$O:(S,SE)$	31	0	46	3	31	$O:(S,SE)$	31	0
20	9	31	$O:(S,SE)$	31	0	47	4	31	$O:(S,SE)$	31	0
21	10	31	$O:(S,SE)$	31	0	48	5	31	$O:(S,SE)$	31	0
22	11	31	$O:(S,SE)$	31	SOL	49	6	31	$O:(S,SE)$	31	0
23	1	31	$O:(S,SE)$	31	0	50	7	31	$O:(S,SE)$	31	SOL
24	2	31	$O:(S,SE)$	31	0	51	1	31	$O:(S,SE)$	31	0
25	3	31	$O:(S,SE)$	31	0	52	2	31	$O:(S,SE)$	31	0
26	4	31	$O:(S,SE)$	31	0	53	3	31	$O:(S,SE)$	31	0
27	5	31	$O:(S,SE)$	31	0	54	4	31	$O:(S,SE)$	31	0

Considering the preceding discussions and example, the lone logical deduction is to include previous warnings in decision making process before allotting a corrective response. However two questions must be answered: how many past warnings should be revisited and how to identify the correlation between them?

In this work, proposed method is to consider an area around the current position of the robot and search it thoroughly for any issued warning. This approach is believed to be capable of addressing both questions at the same time. It is established on two assumptions; one being that at any given time properly shaped and sized neighborhood is independent of other areas and the other is that in such locality, error inciting factors are similar.

The first assumption leads to the point that in any arbitrary area, errors and their consequent corrective reactions have no impact on neighboring zones at least for some time. Meanwhile, the second supposition implies that errors taken place in vicinity are rooted in the same ground and thus correlated. The method, however risky in fast paced environments, has performed exceptionally well in this work both with static and dynamic obstacles.

The algorithm works by first generating a list of nearby nodes to the current location. These points together usually form a five by five square having the present location of the robot in the middle and smaller at sides and corners (see *Figure 5.11*). The size of this vicinity is currently determined empirically. The square shape of the area is especially suitable when the robot is not moving in a straight line for example when circumventing an obstacle. In addition the area form conveniently makes the motion direction irrelevant.

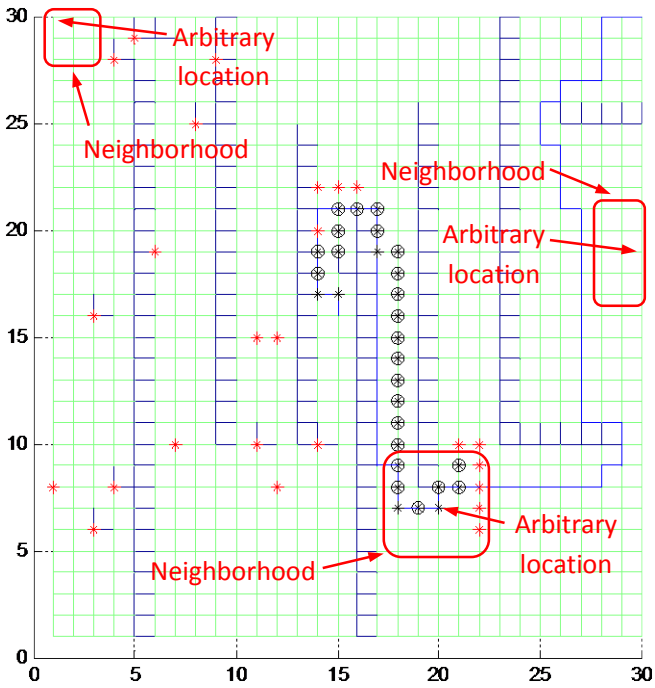

Figure 5.11: Neighborhoods' sizes in different locations on the map.

Then by inspecting the history of the warnings and matching their locations with members of the defined vicinity, warnings and their number of occurrences, if there is any, are specified. If any loop warning has been given out more than two times, in spite

of their types, the SLC will be triggered. Similarly it is activated when the number of IMs has exceeded two.

5.8. Normal condition revision

The robot navigating in an ordinary environment sometimes faces challenging situations. In such conditions adjustments should be applied to achieve the objectives. However, most of the times when the challenging situations are passed, some of the extraordinary measures taken are becoming redundant. Thus if these normal states could be exploited, the overall performance of the robot could be facilitated, which in this work means returning the adjusted parameters to their earlier states. The key to this realization is to identify the circumstances correctly.

Thus far, all the efforts have been directed to recognize alarming patterns. Although the identified unwanted behavior cannot possibly cover all unwanted cases, but they cover an important portion of them which actually have been witnessed during test runs. Therefore a Normal environment condition (NEC), in which the robot can reset its adjusted parameters, can be characterized as a setting in which there has not been a warning issued for some time. Of course the NEC should be utilized with caution as it can lose validity at any time.

As it was stated in the NEC definition, there has to be some time passed without the controller raising any alarm. This is a safety mechanism which makes sure that the robot has had enough time to discover or pass through earlier troubling situations. Currently utilized algorithm in two separate sections, allows 12 and 18 steps to elapse before making its inspections. These numbers are chosen mainly to be higher than the number of steps required to identify an error except for MOLs. However, they are fine-tuned statistically and with some practical considerations.

In the 12th step inspections, the algorithm examines the results of past operations for any IMs warnings and observes whether the robot has been staying in the same spot; while it looks for OTSR over 80 % and the COC more than its initial value. The module is finally activated when it receives false, false, true and true as results of aforementioned assessments.

In the 18th step examinations, the algorithm still investigates whether the robot has been staying in the same spot but it is now concerned by more than only IMs. So the application is prompted when the STH is higher than its initial value, the robot has been moving and there has not been any warning of any kind in the last 18 steps of the operations.

5.8.1. Aiding the operations in NEC

Decreasing the COC and STH values speeds up the computation. Therefore, when detecting the NEC, the robot’s operation is facilitated by returning these parameters to their earlier state.

In the case where the observations account for over 80 % of total actions, the number of observations are presumably unnecessary and they can be lowered to improve the robot’s performance; hence the reduction of the COC. On the other hand, when the robot has been functioning normally, the process is eased by detraction of the STH.

It is important to note that in neither of these modifications the parameters are set lower than their initial values. The reductions go only as far as the increments beginning.

6. DYNAMIC ENVIRONMENT IMPLEMENTATION

In this Chapter a simple dynamic obstacle model is introduced to the optimal path finding problem and the problem is then reformulated accordingly. Finally the program is modified with the necessary changes implemented.

6.1. Moving obstacles

In general the nature of the dynamic obstacles is not fully known and they vary broadly in their behavior. Therefore it is difficult to draw mathematical models of them. However, there are some simpler models of active obstacles that can be utilized to test the robot's performance.

One of those models is the Random Walk model. A random walk is a mathematical formulation of a path that comprises of a series of random steps. A popular random walk model is that of a random walk on a regular grid, where at each step the object moves to neighboring sites of the lattice or stays in its present site according to some probability distribution. In simple symmetric random walk on an infinite lattice, the probabilities of the location jumping to each one of its immediate neighbors are the same as seen in *Figure 6.1* [21].

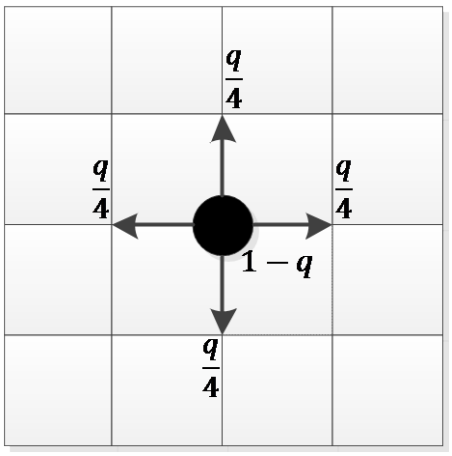

Figure 6.1: Random walker's movement probabilities.

While there are different conditions introduced for the boundaries of finite lattices [21], in this work it is assumed that the grid system is surrounded, for example by fences; thus no object can escape from nor can any enter to the area. Since on the edges and corners of the lattice, there are less leaping choices, logically the probability of

staying and/or moving in available directions should increase. Therefore at the boundaries the random walkers are considered to bounce back to their previous position.

The walkers initially appear in the environment in random locations. As they move, each location may be occupied by more than one walker.

With random walkers as the only moving obstacles it is possible to predict future occupancy probabilities with a mathematical model, as presented in *Equation 6.1*.

Equation 6.1

$$p(O_{i+1}(m,n)) = (1-\frac{q}{4}) \cdot p(O_i(m,n)) + \frac{q}{4}(p(O_i(m-1,n)) + p(O_i(m,n-1)) + p(O_i(m+1,n)) + p(O_i(m,n+1)))$$

As it can be seen the model is recursive in which, $p(O_{i+1}(m,n))$ symbolizes the expected occupancy probability of an arbitrary node, q signifies the propagation probability of random walkers, $p(O_i(m,n))$ stands for the current occupancy probability of said vertex and $p(O_i(x,x))$ are the current occupancy probabilities of its neighboring locations. At edges, respectively corners, there are only 3, respectively 2 jump terms and thus the probability of staying at present location is $(1-\frac{3q}{4})$, respectively $(1-\frac{q}{2})$.

According to the model, in every interval, occupancy probability of each node spreads to its immediate neighboring vertices with a rate equal to q while decreasing itself.

6.2. Problem formulation

In this work the path finding problem in a dynamic environment with known obstacle movement probabilities does not differ much from the one in static environments. Here, the action space, rewards and observations probabilities are the same in both environments. The only dissimilarity is the belief state which changes with respect to time in the dynamic environment. The occupancy belief evolution is modeled by *Equation 6.1*. The optimal path problem is described in the *Equation 3.8* is then reads as in *Equation 6.2*. Note the occupancy belief index.

Equation 6.2

$$V\left[L_i, \left\{p_i^{O_i(m,n)}\right\}_{m,n=1}^{M,N}\right] \approx \max_{\{a_{i'}\}_{i'=i}^{T_0-1} \in \{A_M \cup A_o\}_{i'=i}^{T_0-1}} E\left\{\sum_{i'=i+1}^{i+T_0} r\left(L_{i'}, O_{i'}(L_{i'})\right) + V^{(0)}\left[L_{i+T_0}, \left\{p^{(O(m,n),map)}\right\}_{m,n=1}^{M,N}\right]\right\}$$

6.3. Modifying the program

The program in dynamic environment is closely related to the one in static environment in its core functionality. They have similar modular structure with most algorithms shared. Schematic of the dynamic environment version of the program is depicted in *Figure 6.2*.

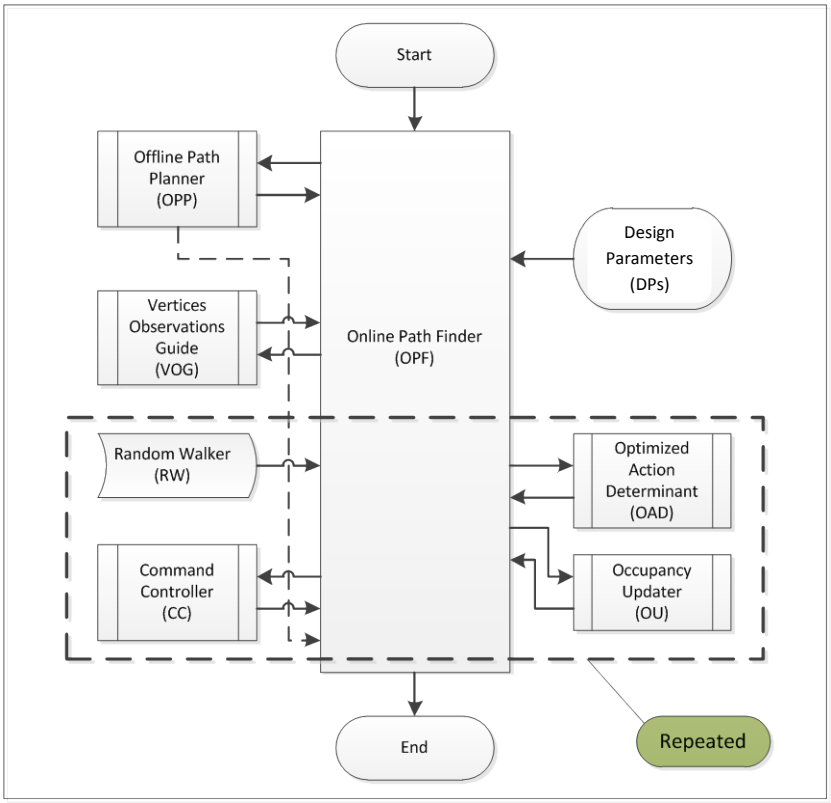

Figure 6.2: Program's variation in dynamic environments.

As it can be seen in the Figure the main difference of the program's two versions is presence of random walker and occupancy updater modules. The occupancy updater is also called within the optimum action determinant. The modifications made to the OPF and OAD are summarized in the following

6.3.1. The OPF modification

The changes in the OPF are limited to introduction of random obstacles and their characteristics which are described in the following. The codes for the program can be found in Appendix 7.

Random walkers' motion probability marks the probability of random walkers to move from their locations to any of their neighboring places. Presence or absence of this parameter denotes one of the differences between the two versions of the OPF. In a static environment, the probability of random walkers moving is zero, hence, absence of the parameter.

In a static environment, the update process consists of marking the observed nodes as occupied or vacant by increased or decreased probability respectively. In a dynamic environment however, there are two stages of updating. First, same as the above and then a function called Occupancy updater is called to update the belief state of the entire map; which is due to the fact that with obstacles moving, occupation of a node increases the occupancy chances of the neighboring locations in the future. The occupancy updater maps the probabilities of moving obstacles for the robot and keeps them updated for each decision making cycle.

6.3.2. Modifications to the OAD

When determining the optimum action in dynamic environment, the OAD needs to update its belief in every recursive cycle before reaching the short time horizon. To do so the OAD calls the occupancy updater function with each branch of its search tree. With each call the OU function further updates the belief state upon its previous updates. Note that the function changes the belief if there it has received any new information obtained through observations. Also note that the OPF creates a reference of the occupancy belief before calling the OAD to restore it to the current time state.

6.4. Random walker (RW):

In this work, the first time that the online path finder summons the RW function, provides it with the area dimensions, the number of random walkers wandering in it and their moving probability. Of course the probability is independent of the OPF and can be defined in the function itself; nevertheless it is introduced in the main structure for better management. The RW unit generates some random initial positions for the walkers and returns them to the OPF. These primary sites cannot coincide with the robots start position nor its destination position. Therefore the locations are omitted from the function's pool of random choices.

In the later calls, the RW module governs the already introduced random walkers' behavior. The unit receives the area grid map, latest position of the random walkers accompanied with the probability of them moving. Then, a random number is generated for each walker. By aggregating the figure into any of five different categories, which are tied with the walkers moving probabilities the next action of the walker is determined. Finally utilization of the grid map, walkers' current positions and the succeeding directions, result into a subsequent position of each walker.

A final note on this module's operations is about its dependence on the program cycles. In this work, the RW function is called with every time step of the program. In this regard, the walkers' timing are totally dominated by each time cycle of the programs calculations. In reality however, such relation is not completely accurate; hence, the room for improving the component in the future. The codes for this module can be found in Appendix 8.

6.5. Occupancy updater (OU)

In the beginning of the path the robot has a rather naive view of the environment ahead, assuming that all the unknown locations have the same occupancy probability. Nonetheless, as it observes and moves toward the destination, the robot may detect some objects.

In a dynamic setting, observing an object in a location equals to change in occupancy probability of its immediate vicinity within the next time step and their neighboring locations in a step after that and so on. It is to be noted that the algorithm ahead requires exact pose estimates for the obstacles; hence it does not solve the general mapping problem [12].

The OU function utilizes an algorithm, shared with few other modules, mapping each node's vicinity and after that constructing the *Equation 6.1* is straight forward. The codes for this module can be found in Appendix 9.

The module can also be employed to update only a fraction of the entire lattice. This capability is particularly designed for the optimized action determinant, which isolates specific sections of the whole map for its operations and needs to know the future possibilities beforehand.

Figure *6.3* simply illustrates propagation of occupancy probabilities from two arbitrary nodes in five interims with following values:

$$p(O_i(11,25)) = 1 \quad , \quad p(O_i(24,25)) = 1 \quad , \quad p(O_i(x,x)) = 0.2 \quad , \quad q = 0.5$$

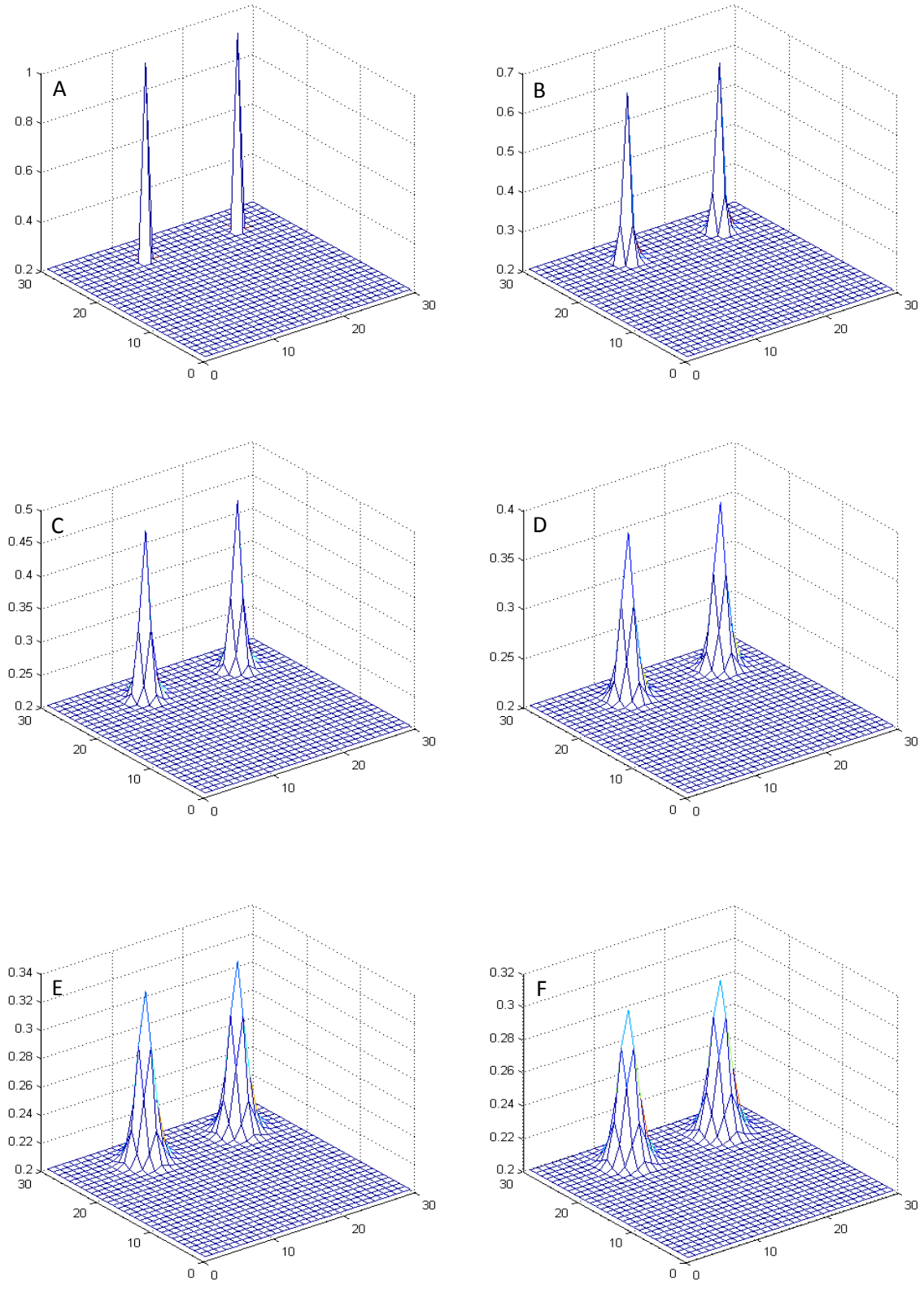

Figure 6.3: Updating occupancy belief in 5 time steps. (A): Initial belief (B), (C), (D), (E), (F): One cycle later.

7. RESULTS

This Chapter aims to illustrate the robot's operations with statistical analysis. The robot's performance in static environment –both with the original program and the new one, is covered in the first Section. Then its behavior when confronting dynamic obstacles is depicted in the second Section.

7.1 Robot's performance in static environment

The main assumption in this Section is that the originally unknown obstacles in the robot's way are static. This piece of information is heavily used in the program. Both the original and new programs were tried out for thousands of times to accurately assess their performance. In the following subsections first the outcomes of the original program's tests are demonstrated and then the new program's results are presented.

7.1.1 The original program

The first set of trials was conducted to determine the basic capabilities of the robot, i.e. statistics of its operations. An instance of such is demonstrated in *Figure 7.1* which was carried out with the following parameters:

$$STH_i = 4\,, \qquad COC_i = -10000\,, \qquad \alpha = 99\,\%\,, \qquad P_o = 10\,\%$$
$$Obstacles\ Count = 25\,, \qquad P_{ob} = \{(0.01\,, 0.99), (0.05\,, 0.95)\}$$

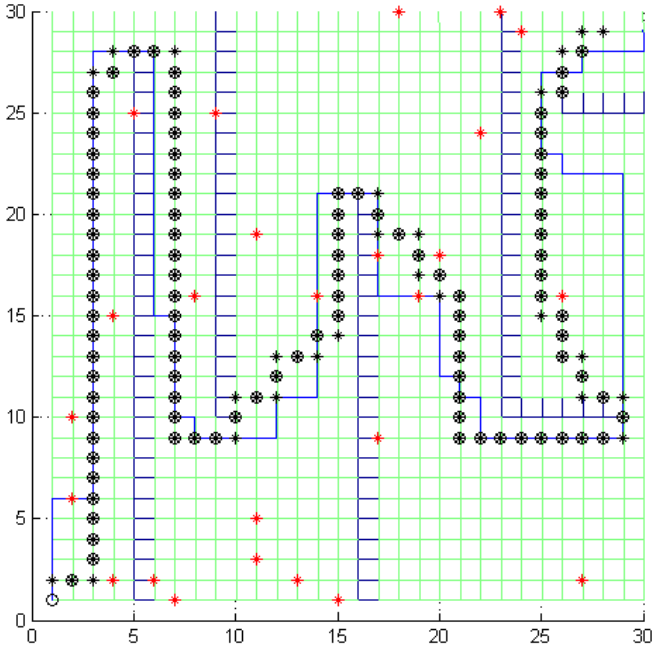

Figure 7.1: Robot's operations on its way to target location.

During these tests some shortcomings of the robot were identified. Amongst them was the robot's inability to complete its travel through narrow corridors or S shape passages in more than 26 % of times with STH as high as four. Of course the robot's confusion behind barriers depends on the obstacle's configuration and shape. Nevertheless, in these tests all obstacles were scattered in the environment randomly. On the other hand when the program does execute successfully like depicted in *Figure 7.1*, another notable problem was recognized to be the computation time required to complete the task. Although getting an estimate of average completion time proved to be very hard, nevertheless among countless trials there has not been a case witnessed where the robot has reached the destination under about 22 minutes. Note that the computation times throughout this work are measured on computers with fast and powerful processors (7 cores clocked at 3.4 GHz). *Table 7.1* is a summary of the results obtained with the original program.

Table 7.1: Original program's results.

Success Rate (%)	Step Count		STH	Collision Avoidance Rate (%)		Computation Time (s)	
	Mean	STD	Constant	Mean	STD	Mean	STD
73.32	227	19	4	99.86	0.09	1492	180

In the table the success rate is defined as the percentage of completed travels. The collision avoidance rate is a measure of the robot's ability to make right decisions circumventing obstacles. The figure is calculated using *Equation 7.1*.

Equation 7.1

$$\overline{CAR} = 100 - Mean\left(\frac{\sum_{100} CI_{25}}{\sum_{100} SC_{25}} \times 100\right)$$

,where the variable CI, denotes the number of collision incidents and the variable SC, symbolizes the steps counts. The tests have been repeated 100 times with 25 random obstacles.

Next set of trials was made to determine the effect of observations error probabilities on the overall performance of the robot. For this purpose, all other parameters were kept constant while the observations probabilities were changing. The reference values for observation probabilities were $P_{ob} = \{(0.025, 0.975), (0.075, 0.925)\}$ and the error probabilities were incremented by steps of 5 %. The program was executed for minimum of 14 times with each observation probability set. The results are illustrated in *Figure 7.2*.

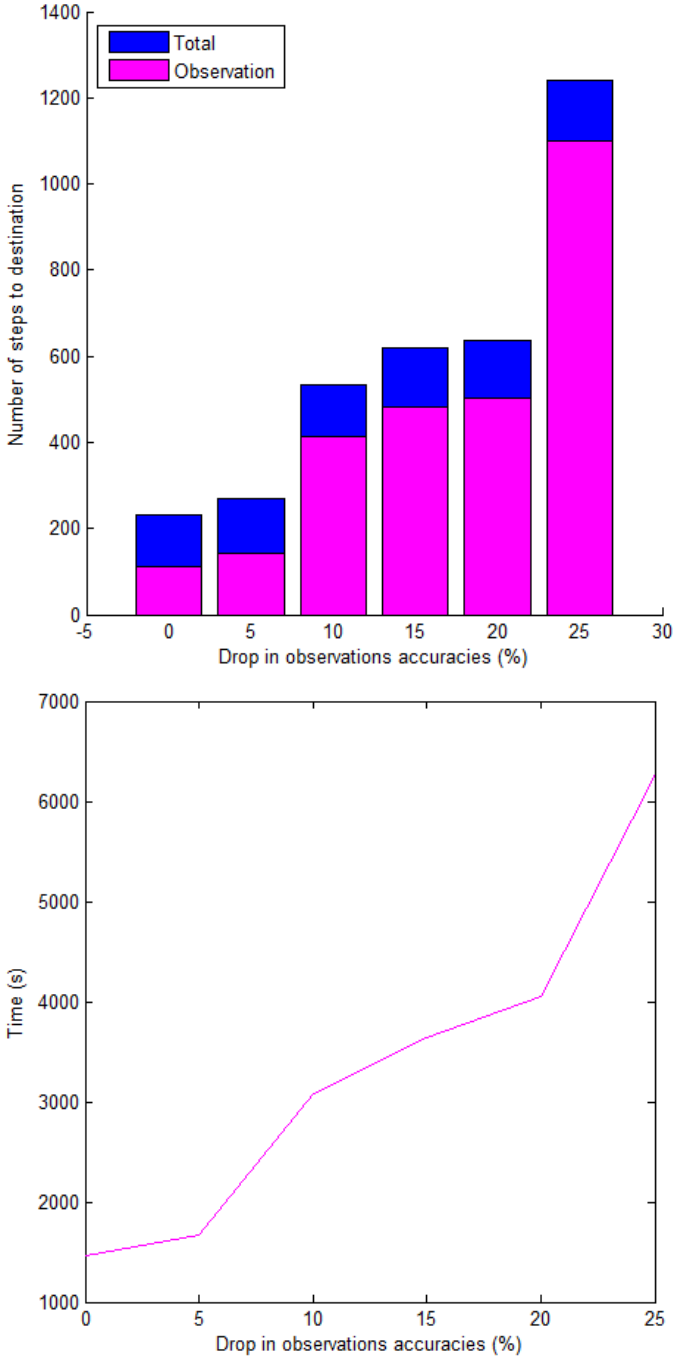

Figure 7.2: Change in SC and number of observations (top) and computation time (bottom) as a result of observation probabilities reduction. The number of motion steps was roughly the same all the time.

Apart from the results depicted in the figure, the success rate of the robot was also extremely compromised with decreasing observation precision. It is estimated that a 5 % increase in the observation error probability reduces the success rate by 25 %. In fact the decline was so steep that when the erring probabilities were increased over 30 %, the robot was not able to complete its travel at all.

The last set of trials was designed to outline the relation between the COC and the robot's performance. In order to measure the impact of COC on the robots behavior, its

expected responses must be defined. The first ever essential robot's behavior is arguably to commence its operations with an observation action. Thus an adequate COC is a value that delivers such response. A measure against which the COC can be assessed is, according to *Equation 7.2*, the ratio of observations to total number of steps.

Equation 7.2

$$OTSR = \frac{Number\ \ of\ \ observations\ \ steps}{Number\ \ of\ \ observations\ \ steps + Number\ \ of\ \ motion\ \ steps} \times 100$$

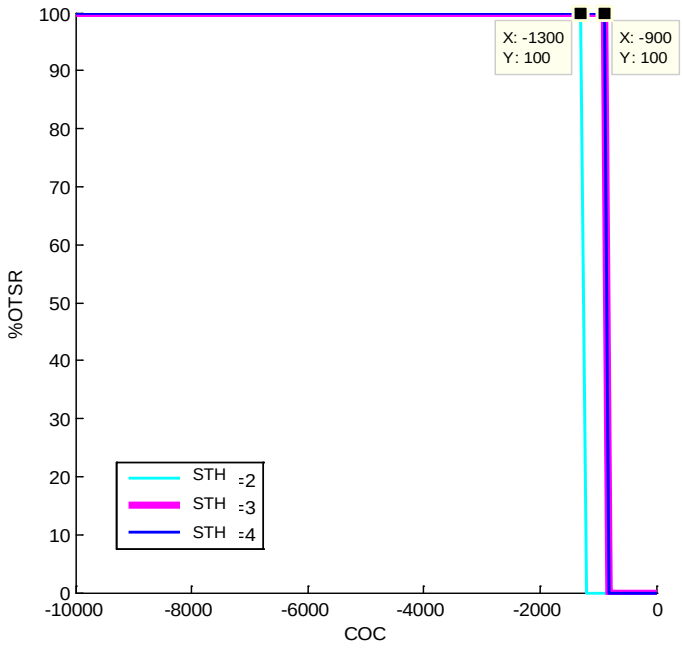

Figure 7.3: The first action, displayed in form of OTSR, as function of COC.

Figure 7.3 is obtained from the data collected from approximately ten thousand trials. In each of mentioned experiments the first action performed by the robot is recorded. Then the number of observations, in form of OTSR, is plotted as a function of the cost of collision. It is apparent in the figure that with the STH more than two, the first robot's output is an observation as soon as the COC reaches to 900. However, with a lower STH, the target OTSR cannot be achieved while the cost of crash is less than 1300.

Moreover, in a normal environment and with no external intervention to the robot's outputs, it is expected from the robot to exhibit OTSR over a certain percentage. In this work, since the robot has only two choices to either move or observe, a normal OTSR should be approximately 50 %. Over the course of 10000 test runs, the COC and the robot's first 100 actions were recorded. To get a better estimate, the robot's entry point to the environment was randomly selected in each trial. Results of these trials are illustrated in *Figure 7.4*.

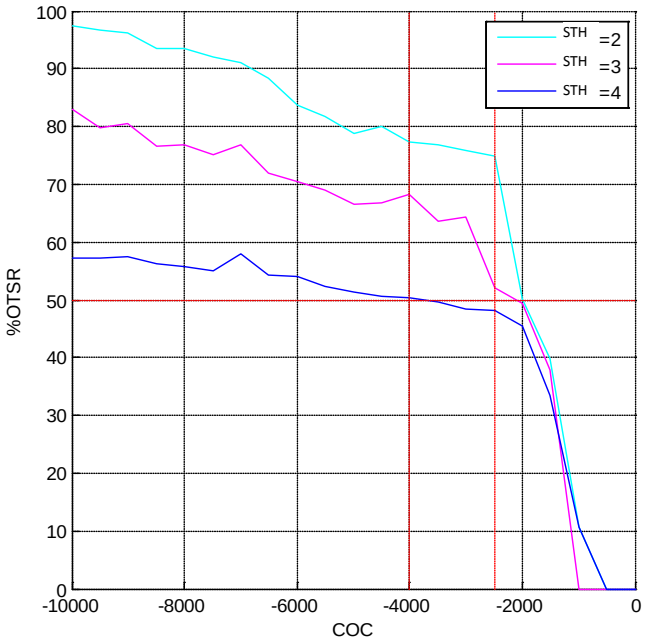

Figure 7.4: OTSR as function of COC, for 100 actions.

Figure 7.4 suggests that the robot is not capable of passing the 50 % mark for OTSR with COC less than 2000 while it might almost hit the observations only mode when the collision cost reaches to 10000. A closer look to the results reveals that beyond certain thresholds, increasing the COC would inflict relatively less rise in the OTSR. Furthermore, these turning points are appearing to be STH-specific; and the illustration implies that the threshold is about 4000 for STH equal to four while it is approximately 2500 for smaller value STHs.

Finally, the last piece of information that can be extracted from the graph is the effect of COCs increments, beyond their initial values on, the OTSR. The *Table 7.2* reveals this relationship. These values are especially helpful when it comes to rectification of robots' behaviors (see section 5.5).

Table 7.2: Data extracted from the Figure.

STH	Initial COC	Increase in the COC (points)	Increase in the OTSR (%)
2	2500	1500	6.16
3	2500	1500	4.5
4	4000	1000	1.16

The performance of the robot has been rather below what is required.

7.1.2 The new program with active CC unit

The new program is designed to reach the destination as fast as possible by overcoming the problems that the original program had difficulties coping with. However, the new program is not cope with all situations; yet, it is performs much better than the old one.

Figure 7.5 through *Figure 7.7* demonstrate the robot's operations within an entire path finding sequence. The initial parameters were in as in subsection 7.1.1 and as follows:

$STH_i = 2$

$COC_i = \begin{cases} -2500 & STH_i \ll 3 \\ -4000 & STH_i > 3 \end{cases}$

$\alpha = 99\ \%$

$P_o = 10\ \%$

$Obstacles\ Count = 25$

$P_{ob} = \{(0.01\ , 0.99), (0.05\ , 0.95)\}$

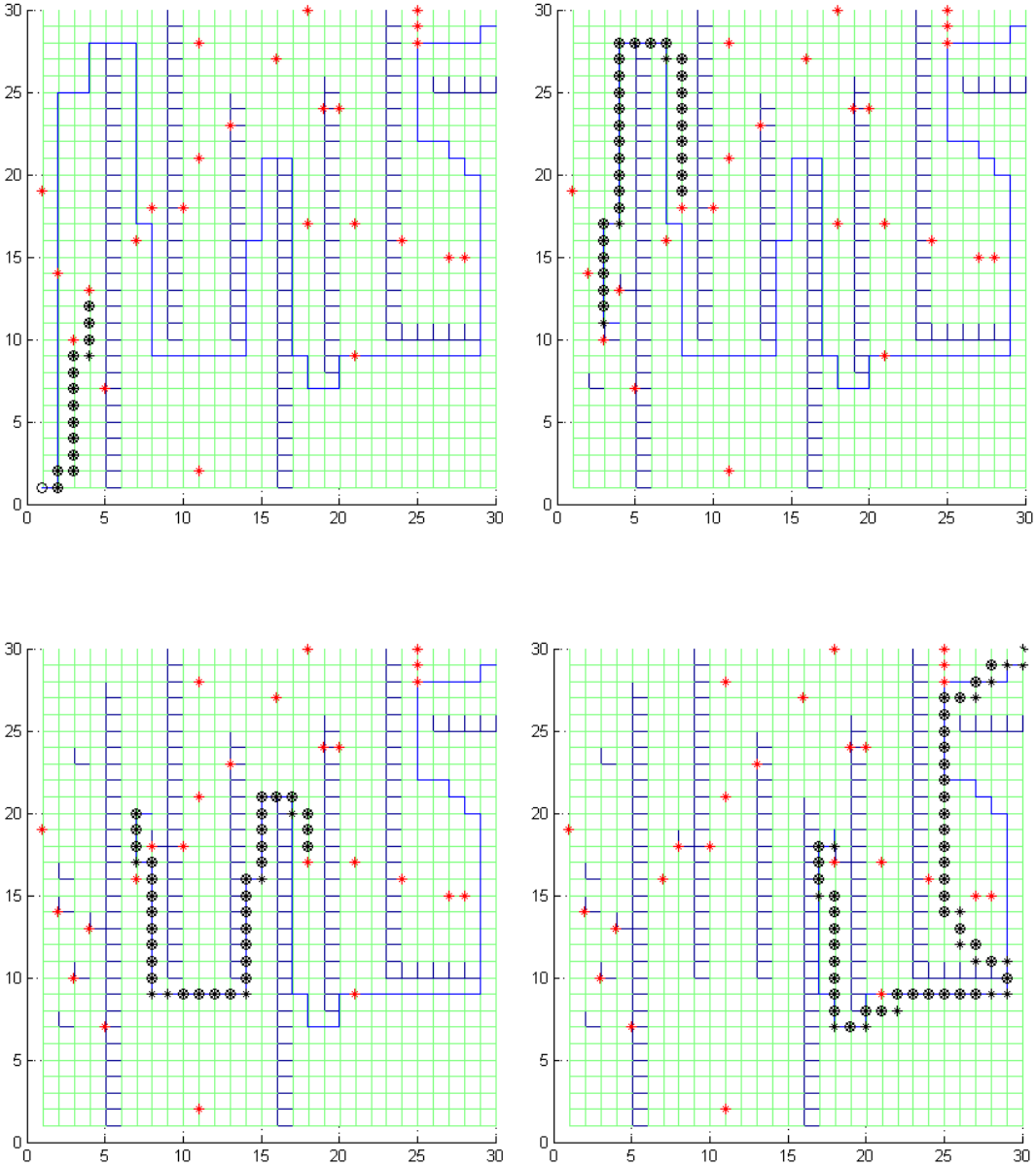

Figure 7.5: Robot's operations on its way to target location.

The first Figure shows the result of online path finding while the next two illustrate full travel path and the evolved belief matrix.

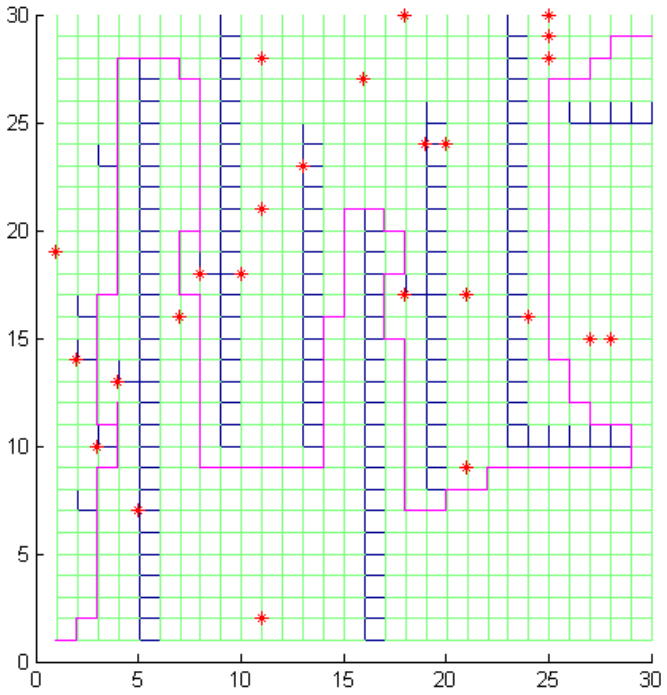

Figure 7.6: Robot's actual travel path.

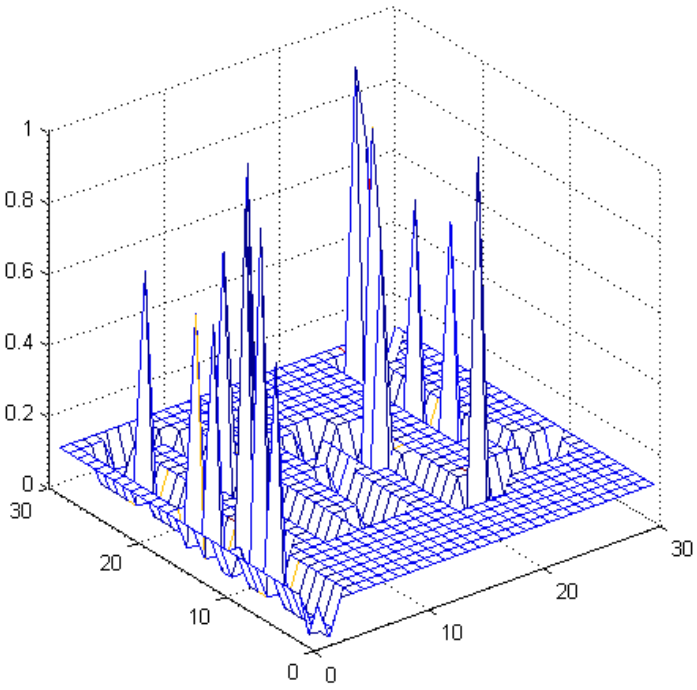

Figure 7.7: Robots final belief of the environments occupancy.

This example is only one of 266 successful trials. As displayed, the robot has been able to reach its goal with a few re-routings and without colliding with any of the random obstacles present. However, the main achievement of this procedure is the computation time required for the robot to arrive at its destination; which has been dramatically decreased compare to the original program. Results extracted from over 200 tests are illustrated in *Table 7.3*.

Table 7.3: The new program's execution results in static environments.

Success Rate (%)	Step Count		Re-Routings		STH		Collision Avoidance Rate (%)		Computation Time (s)	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD	Mean	STD
98.87	373	68	4.39	1.97	2.01	0.17	99.89	0.12	10.5	26.28

According to the statistics the robot can reach its goal nearly 100 % of times. In the conducted experiments, only three of trials were not successful wherein the errors in observations were the reason behind the failures. In those particular circumstances the robot had deemed that all the passages toward the destination are blocked due to wrong observation results and thus terminating the operations.

The number of steps has increased compare to the original program which is due to increased number of observations.

The short time horizon, according to the table of statistics, has not been altering in the programs much. This means that the preliminary control algorithm have had successful operations.

The CAR figure in *Table 7.3* is calculated using *Equation 7.3*.

Equation 7.3

$$\overline{CAR} = Mean\ \left(CAR_i\right) \quad , \quad CAR_i = 100 - \frac{\sum_n CI_i}{\sum_n SC_i} \times 100 \quad , \quad 1 \le i \le 75 \quad , \quad n \approx 15$$

In this equation, for each obstacle count, denoted by subscript i, the test has been repeated n times.

The individual CAR values, estimated from numerous tests with varying number of obstacles, are depicted in *Figure 7.8*. It reveals that the robot is capable of avoiding accidents at a very high rate. Conversely, it also illustrates a downward trend; which is not strange due to the impact that increases in the number of obstacles have on the probability of collision incidents leading to decrease in the CAR.

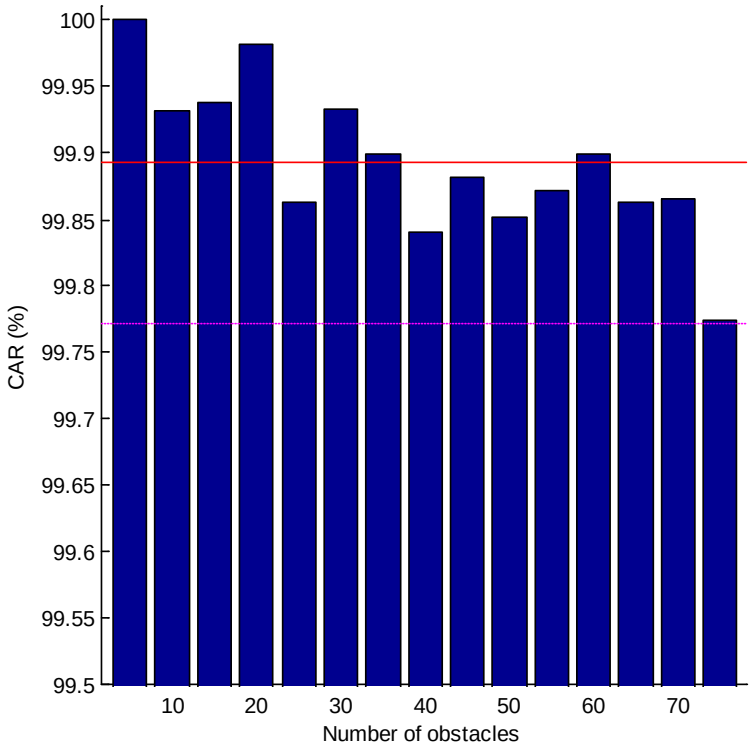

Figure 7.8: Collision avoidance rate as a function of obstacles' number. The red line signifies the average value while the dashed magenta line is the STD.

Finally, the computation time well under a minute is the definitive indicator of the robot's accomplishment. This achievement is directly linked to use of smaller STH which is possible here with the precaution mechanisms but not in the original program.

7.1.3 The new program with inactive CC unit

Receiving encouraging results from the robot raised a question that to what extent the CC module has been responsible for its success. Thus experiments were performed with disabled controller module.

The results pointed out that the robot is almost incapable of completing its task without the module. In over a hundred trails, the robot exhausted all of its allowed steps without even going as far as half of the way toward the destination; that is but only for two rare successful cases. Of course the design parameters, STH and COC, were at their initial points, optimized for the new program; otherwise the program has the exact same level of performance as in its original form.

As a conclusion, it is obvious that the controller component plays a crucial role in the accomplishment of the procedure.

7.2 Robot's performance in dynamic environment

The main objective of the robot with the new program was to reach the destination in a static environment. However, with modifications into the OPF and OAD and

introducing the complementary components it has been able to pursue the target in a setting wherein random walkers are present. *Figure 7.9* and *Figure 7.10* illustrate results of a successful execution of the program. Again the initial DPs are stated below.

$STH_i = 2$

$COC_i = \begin{cases} -2500 & STH_i \ll 3 \\ -4000 & STH_i > 3 \end{cases}$

$\alpha = 99\ \%$

$P_o = 10\ \%$

$q = 10\ \%$

$Walkers\ Count = 8$

$P_{ob} = \{(0.01\ , 0.99), (0.05\ , 0.95)\}$

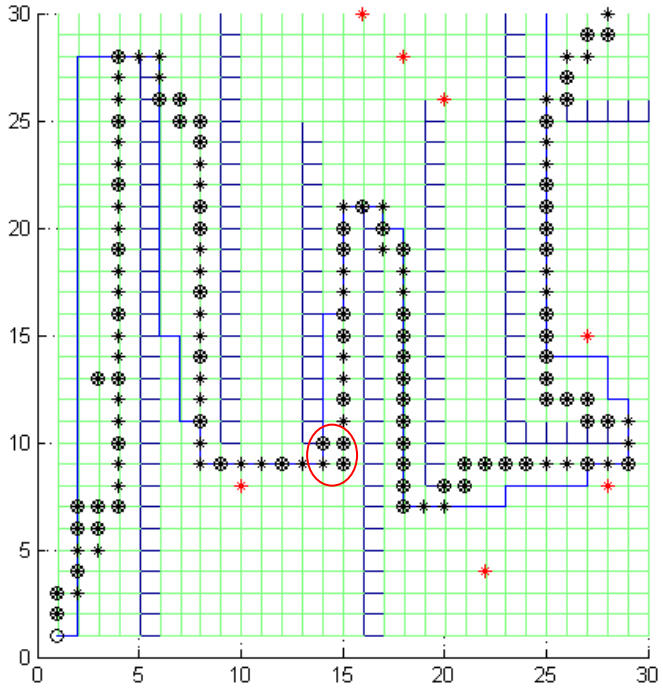

Figure 7.9: Robot's operations on its way to target location.

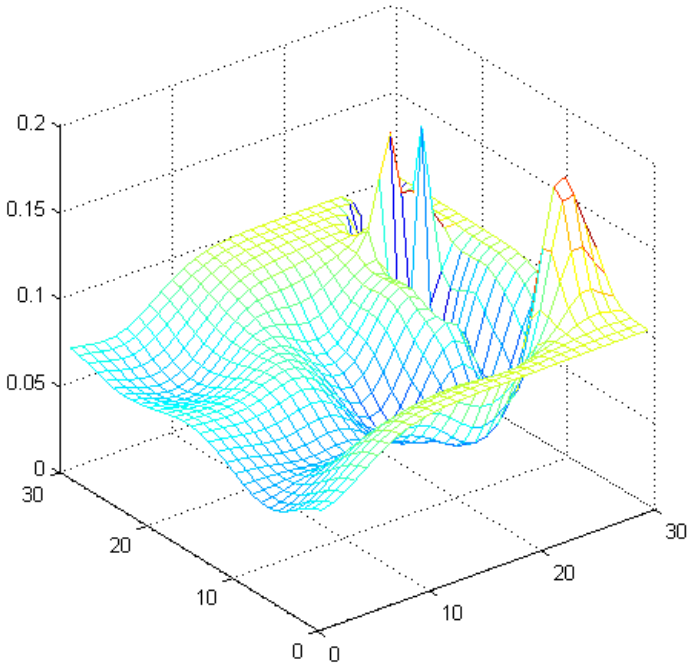

Figure 7.10: Robots final belief of the environments occupancy.

Note that in *Figure 7.9*, one of the random walkers is on the path of the robot. In fact the robot has taken a step back circumventing the obstacle; hence there has not been an actual crash.

In this example, the robot has been able to arrive at its destination without colliding with any of the walkers. Results of one hundred more executions are summarized in the *Table 7.4*.

Table 7.4: Execution results in dynamic environments

Success Rate (%)	Step Count		Re-Routings		STH		Collision Avoidance Rate (%)		Computation Time (s)	
	Mean	STD	Mean	STD	Mean	STD	Mean	STD	Mean	STD
98.21	634	32	1.38	0.62	2.00	0.001	98.62	0.71	36.36	4.69

The drop in the success rate and rise in the SC are both in direct response to the dynamic surroundings. The robot may terminate its operations more often because of higher possibility of random walkers blocking all passages ahead. Also bypassing moving obstacles requires more steps.

Given that the obstacle avoidance is much more difficult in active environments, the robot's CAR seems reasonable enough too. *Figure 7.11* is an illustration of the robot's efforts circumventing random walkers. Compared to those under static environment conditions (see *Table 7.3*) the results here are obviously less accurate.

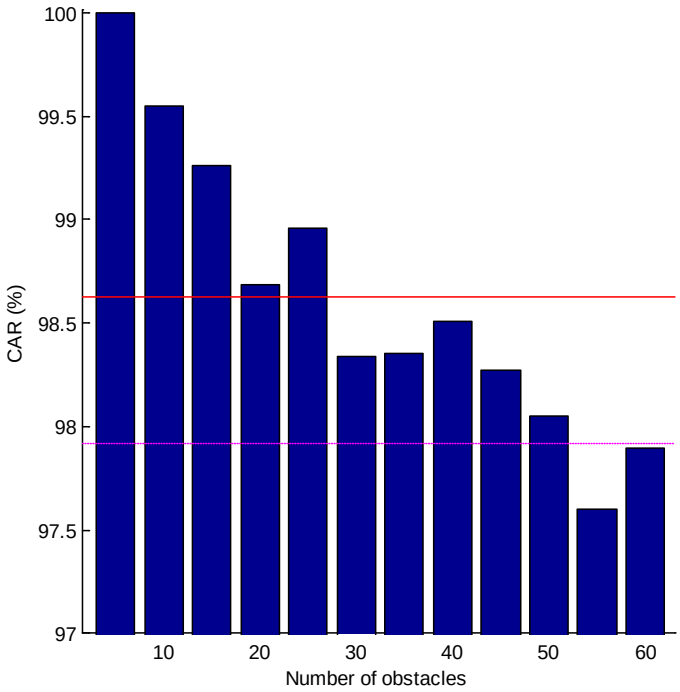

Figure 7.11: Collision avoidance rate as a function of obstacles' number. The red line signifies the average value while the dashed magenta line is the STD.

The decline in the collision avoidance rate however, is not solely due to robots decision making process. On the contrary, it is mainly because of limited observation capacity. For instance, if in a particular time more than one walker approach the robot, even if the robot decides to observe a certain direction it cannot see the other coming walkers and thus a crash is very likely. Thus the robot’s performance is quite satisfactory.

8. CONCLUSIONS AND FUTURE WORK

Path finding utilizing POMDP framework is a well-studied method. Many experiments have been conducted and articles have been published about the solutions to this problem. Thus the benefits and limitations of this process are quite known. The main idea in this work was to assess and to improve currently existing method with optimizing the design parameters and introduction of new components controlling its operations.

Judging from the results the original path finding approach is powerful enough in most circumstances. However, on its own, and with presence of unmapped and difficult obstacles, it does not seem to be capable to complete the task with desired efficiency. The magnitude of the problems sensed in the experiments with static obstacles is gravely amplified when dealing with dynamic obstacles.

In case of the robot facing static obstacles, introduction of a controller, specification of efficient DPs and finally some modifications in the original program were efforts to bring up the robot's performance to its maximum potential. The outcome is a definite improvement, see *Table 8.1*. However, there is still a lot of room for development in various directions.

Table 8.1: Results comparison

	Success Rate (%)	Computation Time (s)	Step count	STH
Original program	73.32	1492	227	4
New program in static environment	98.87	10.5	373	2.01
New program in dynamic environment	98.21	36.36	634	2.00

Assuming that the POMDP, Online programming and Value iteration methodologies are to be kept, the future work that can improve the performance of the approach are at least the following.

- Incorporating machine learning and pattern recognition methods both to improve the observation mechanism enabling it to differentiate between different obstacles.
- Implementing more sophisticated pattern recognition techniques into the controller and equip it with better mathematical models

- Employing more accurate models of dynamic obstacles to improve the OU function of the program

The suggestions can be combined in one component, such as controllers, in order to obtain powerful modules.

As for the program itself, the future work can be towards simplification of the generic problem with more efficient approximations. However, it was suggested by reference [22] and made evident with some practical examples, that the program in its current state does not seem to be fit for larger scale problems. Thus, for future works, alternative approaches such as Monte-Carlo methods [23] are advisable to be exploited.

REFERENCES

1 Bin Wu, Tze Leung Lai, Yuguo Chen. Sequential Planning for Robotic Navigation and Exploration under Uncertainty. Introduction to Modern Robotics. iConcept Press, Queensland, Australia 2012.

2 Harry Chia-Hung Hsu, Robot Path Planning. Wiley Encyclopedia of Computer Science and Engineering. Published Online: 16 MAR 2009.

3 Chiara Fulgenzi, Anne Spalanzani, and Christian Laugier. Dynamic Obstacle Avoidance in uncertain environment combining PVOs and Occupancy Grid. IEEE International Conference on Robotics and Automation. Roma, Italy, 10-14 April 2007.

4 Sylvie C.W. Ong, Shao Wei Png, David Hsu, Wee Sun Lee. POMDPs for Robotic Tasks with Mixed Observability. International Journal of Robotics Research. Volume 29 Issue 8, July 2010.

5 http://marsrovers.jpl.nasa.gov/home/index.html

6 S. M. LaValle: Planning Algorithms, 2006 Cambridge University Press.

7 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms (2nd Ed.). MIT Press, Cambridge, MA, 2001.

8 E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269–271, 1959.

9 J. Pearl. Heuristics. Addison-Wesley, Reading, MA, 1984.

10 Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall.

11 R. Bellman, Dynamic Programming. Princeton University Press, 1957.

12 S. THRUN, W. BURGARD, D. FOX, Probabilistic Robotics; Intelligent Robotics and Autonomous Agents series. The MIT Press, August 19, 2005.

13 Martin L. Puterman. Markov Decision Processes|Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, 1994.

14 Lovejoy, W. (1991). Computationally feasible bounds for partially observed Markov decision processes. Operations Research 39: 162–175.

15 R. Ritala. Notes on path planning with obstacle avoidance. Tampere University of Technology, Tampere, 2011

16 L. P. Kaelbling, M. L. Littman, A. R. Cassandra, Planning and Acting in Partially Observable Stochastic Domains. Journal of Artificial Intelligence, Volume 101 Issue 1-2, May 1998. Pages 99-134.

17 R. Dial. Algorithm 360: Shortest path forest with topological ordering. Communications of the ACM, Volume 12, 1969. Pages 632–633.

18 Zhang, J. and Goodchild, M.F. (2002). Uncertainty in Geographical Information. first published 2002 by Taylor & Francis.

19 Klir, G.J. and Folger, T.A. (1988). Fuzzy sets, uncertainty and information. Prentice-Hall Int. Editions.

20 G. D'Agostini. Bayesian reasoning in data analysis - A critical introduction. World Scientific Publishing 2003.

21 George H. Weiss, Aspects and Applications of the Random Walk. Elsevier Science & Technology Books, Jan 1994.

22 J. Pineau, G. Gordon, and S. Thrun. Anytime point-based approximations for large POMDPs. Journal of Artificial Intelligence Research, Volume 27, 2006. Pages 335–380.

23 D. Silver, J. Veness, Monte-Carlo Planning in Large POMDPs. NIPS 2010: 2164-2172

APPENDIX 1

The Offline Path Planner

```
function [steps, opt_val, rew, alpha, optimal_path,opt_pathx,
opt_pathy,on_the_way_cost, N, M, rew_matr, p_count,
start_grid]=OPP(start_grid, rew, updated_obs, updated_obs_op)
% ---------------------------------------------------------------------
% Constants
% ---------------------------------------------------------------------
% N and M are the web dimensions. 1 is upper left corner, N on upper right and
N*M on lower right
N=30;
M=30;
vl=N*M+1; % Number of locations on the map
alpha=0.99; %Discounting factor
hrew=-100*N*M; % Large negative rewards for the grids known as the "walls"
% ---------------------------------------------------------------------
% known obstacles
% ---------------------------------------------------------------------
% The given "start grid" is the entry pint on the map. In case that the
function is run from the start, the value will be equal to 1.
if start_grid == 1;
  rew=-rand(vl,1); % Reward associated to each grid cell
  rew(N*M)=100*M*N; % Final (goal) reward (large)
  rew(vl)=0; % Stop point reward
  % Walls are defined
  rew((4*N+1):(4*N+N-3))=hrew;
  rew((8*N+10):(8*N+N))=hrew;
  rew((floor(2*M/5)*N+10):(floor(2*M/5)*N+N-6))=hrew;
  rew((floor(M/2)*N+1):(floor(M/2)*N+N-10))=hrew;
  rew((floor(3*M/5)*N+8):(floor(3*M/5)*N+N-5))=hrew;
  rew(((M-8)*N+10):((M-8)*N+N))=hrew;
  rew(((M-8)*N+10):N:((M-3)*N+10))=hrew;
  rew(((M-5)*N+25):N:((M-1)*N+25))=hrew;
end
% ---------------------------------------------------------------------
% Updated obstacles
% ---------------------------------------------------------------------
for i=1: length(updated_obs);
  rew(updated_obs(i))=updated_obs_op(i)*hrew+(1-
updated_obs_op(i))*rew(updated_obs(i));
end
% ---------------------------------------------------------------------
% Possible steps
% ---------------------------------------------------------------------
steps=[1:vl;1:vl;1:vl;1:vl]';
for i=1:M
  steps(((i-1)*N+1):(i*N-1),1)=((i-1)*N+2):(i*N); %east steps
  steps(((i-1)*N+2):(i*N),2)=((i-1)*N+1):(i*N-1); %west steps
end
for i=1:M-1
  steps(((i-1)*N+1):(i*N),3)=(i*N+1):((i+1)*N); %south steps
  steps((i*N+1):((i+1)*N),4)=((i-1)*N+1):(i*N); %north steps
end
steps(N*M,1:4)=vl; % Exit grid
steps(vl,1:4)=vl; % Exit and stay grid
% ---------------------------------------------------------------------
% Value iteration
% ---------------------------------------------------------------------
values=zeros(vl,4); %Optimal value vector
opt_val=zeros(vl,1); %Optimal decision vector
opt_act=zeros(vl,1);
% 10000 updates of the optimal value function for each grid cell
for i=1:10000;
  for j=1:4; % Possible steps/actions for each grid cell
    values(:,j)=rew(steps(:,j))+alpha*opt_val(steps(:,j));
```

```
  end
  opt_val=max(values')'; % Optimal value function (action) selection
end
for k=1:vl% Optimal action for each grid cell according to the value functions
  ind=find(values(k,1:4)==opt_val(k));
  opt_act(k)=ind(1);
end
% Rewards, optimal value function values, and optimal actions in grid
% matrix form
rew_matr=reshape(rew(1:vl-1),N,M);
opt_matr=reshape(opt_val(1:vl-1),N,M);
opt_act_matr=reshape(opt_act(1:vl-1),N,M);
% ---------------------------------------------------------------------
% Optimal path
% ---------------------------------------------------------------------
optimal_path=zeros(vl,1); % Optimal path vector
opt_pathx=zeros(vl,1); % Auxiliary vectors for plotting the optimal path
opt_pathy=zeros(vl,1);
% Calculation from a given start grid
optimal_path(start_grid)=start_grid;
opt_pathx(start_grid)=start_grid;
opt_pathy(start_grid)=start_grid;
p_count=start_grid; % Step counter (starting from the start grid)
% Defining the optimal path by matching the optimal actions and steps
for ii=p_count:vl;
  optimal_path(ii+1)=steps(optimal_path(ii),opt_act(optimal_path(ii)));
  if optimal_path(ii)<(vl)
    opt_pathy(ii)=ceil(optimal_path(ii)/N);
    opt_pathx(ii)=optimal_path(ii)-N*(ceil(optimal_path(ii)/N)-1);
    p_count=p_count+1;
  end
end
if start_grid > 1
  optimal_path(1:start_grid-1)=optimal_path(start_grid);
  opt_pathy(1:start_grid-1)=opt_pathy(start_grid);
  opt_pathx(1:start_grid-1)=opt_pathx(start_grid);
end
% ---------------------------------------------------------------------
% On the way cost to be compared with mean of reward
% ---------------------------------------------------------------------
on_the_way_cost=mean(rew(optimal_path(1:p_count-1)));
```

APPENDIX 2

The Online Path Finder

```
clear all
close all
clc
tic; % timer start
% ---------------------------------------------------------------------
% Primary offline-planning
% ---------------------------------------------------------------------
[steps, opt_val, rew, alpha, optimal_path,opt_pathx,
opt_pathy,on_the_way_cost, N, M, rew_matr, p_count, start_grid]=OPP(1, [], [],
[]);
% ---------------------------------------------------------------------
% System state
% ---------------------------------------------------------------------
% System state is location of WM and belief vector of location of MO
depth=2; % the depth starts at 2 but the program could go deeper in horizon
l_dim=5;
if depth<=3 % Initial Cost of clash
  c_clash=-2500;
elseif depth>3
  c_clash=-4000;
end
c_O=-0; %% Cost of observation
action_history=zeros(1,6); % initial action history
A_dim=2*l_dim+1; % Size of area considered in one on-line decision
p_obs=[0.025 0.975; 0.05 0.95]; %erring probabilities of observations
global_occupancy=0.1*ones(N*M,1); % initial occupancy belief
% Position of MOs, changes are in random positions
no_positions = 35; % number of random obstacles
positions=(randperm(N*M-2, no_positions)+1)'; % random position of obstacles
positions_xy = [positions-N*(ceil(positions/N)-1),ceil(positions/N)];
iroad=p_count*5; % maximum length of the plan
pos_on_the_road=zeros(iroad+1,1);
act_on_the_road=zeros(iroad,1);
pos_on_the_road(1)=optimal_path(start_grid); % Same starting point as in the
% optimal path studied in script OPP
% ---------------------------------------------------------------------
% Observation sets
% ---------------------------------------------------------------------
generate_observation_sets_small;
% ---------------------------------------------------------------------
% Mapping of area of entire problem to local optimization problem
% ---------------------------------------------------------------------
f_c=1; % figure counter
figure(f_c);
set (figure(f_c), 'Position' , [100 100 500 500])
mesh(rew_matr);
hold on;
z_opt=zeros(1,p_count-1);
reff=plot3(opt_pathy(1:p_count-1),opt_pathx(1:p_count-1),z_opt); % refreshing
figure
plot3(positions_xy(:,2),positions_xy(:,1),zeros(no_positions,1),'r*');
set(reff,'XDataSource','opt_pathy(1:p_count-1)');
set(reff,'YDataSource','opt_pathx(1:p_count-1)');
set(reff,'ZDataSource','z_opt');

iloop=1; % counter
call_count=1;
% ---------------------------------------------------------------------
while iloop < iroad+1 & pos_on_the_road(iloop)~=N*M-N &
pos_on_the_road(iloop)~=N*M-1 & pos_on_the_road(iloop)~=N*M-N-1
% ---------------------------------------------------------------------
% Local area
% ---------------------------------------------------------------------
```

```
  pos_WM=pos_on_the_road(iloop);
  N0=floor(pos_WM/N)+1;
  NY_start=max(1,N0-l_dim);
  NY_stop=min(M,N0+l_dim);
  NX_start=max(1,pos_WM-N*floor((pos_WM-1)/N)-l_dim);
  NX_stop=min(N,pos_WM-N*floor((pos_WM-1)/N)+l_dim);
  indsa=[]; % Global coordinates of local area
  for i=NY_start:NY_stop
    indsa=[indsa ((i-1)*N+NX_start:((i-1)*N+NX_stop))]; % mapping vector
  end
% ------------------------------------------------------------------
% WM steps on the local area
% ------------------------------------------------------------------
  loc_steps=steps(indsa,:);
  loc_rew=rew(indsa); %Local immediate rewards
  loc_rew_end=rew(indsa)+alpha*opt_val(indsa); % Rewards at the end step of
% on-line optimization
  loc_obs_sets=obs_sets(indsa,:,:);
  state=zeros(length(indsa)+1,1);
  state(1)=pos_WM;
  state(2:end)=global_occupancy(indsa);
% ------------------------------------------------------------------
% On-line optimization of actions
% ------------------------------------------------------------------
  pos_vis=pos_on_the_road(1:iloop);
[opt_value,opt_act,value]=OAD_S(depth,state,indsa,loc_rew,loc_rew_end,loc_step
s,loc_obs_sets,c_O,c_clash,alpha,p_obs,pos_vis);
% ------------------------------------------------------------------
% After optimization update position or belief depending on action
% ------------------------------------------------------------------
  fprintf('At time step %3i \n',call_count)
  fprintf('optimal action is %2i \n',opt_act)
  if opt_act<5 % Step taken, no measurement made
pos_on_the_road(iloop+1)=steps(pos_on_the_road(iloop),opt_act);
    fprintf('and new position of the WM is %3i \n\n',pos_on_the_road(iloop+1))
    hold on;
plot3(ceil(pos_on_the_road(iloop+1)/N),pos_on_the_road(iloop+1)-
N*(ceil(pos_on_the_road(iloop+1)/N)-1), zeros(length(iloop+1),1),'k*');
    drawnow;
    hold off;
  else % Measurement made, no step taken
    pos_on_the_road(iloop+1)=pos_on_the_road(iloop);
    fprintf('and the WM holds its position at %3i
\n\n',pos_on_the_road(iloop))
    hold on;
plot3(ceil(pos_on_the_road(iloop+1)/N),pos_on_the_road(iloop+1)-
N*(ceil(pos_on_the_road(iloop+1)/N)-1),zeros(length(iloop+1),1),'ko');
    drawnow;
    hold off;
    imeas=opt_act-4; %The optimal measurement
% ------------------------------------------------------------------
% Making the measurements;
% ------------------------------------------------------------------
% each point measured for being occupied or not
    obsind=obs_sets(pos_WM,imeas,:);
    apuind=find(obsind~=0);
    if ~isempty(apuind)
      obsind2=zeros(length(apuind),1);
      for il=1:length(apuind)
        obsind2(il)=obs_sets(pos_WM,imeas,apuind(il));
      end
%Simulate the measurement result (value=1, no observation; =2 is observed)
      result=zeros(length(apuind),1);
      for imi=1:length(apuind)
        if ismember(obsind2(imi),positions)
          ptest=p_obs(imi,2);
        else
          ptest=p_obs(imi,1);
```

```
        end
        if rand<ptest
          result(imi)=1;
        end
      end
      result;
% ------------------------------------------------------------------
% Update occupancy belief
% ------------------------------------------------------------------
      bel_ap=global_occupancy(obsind2);
      bel_post=zeros(length(obsind2),1);
      for iup=1:length(apuind)
        if result(iup)==1
bel_post(iup)=p_obs(iup,2)*bel_ap(iup)/(p_obs(iup,2)*bel_ap(iup)+p_obs(iup,1)*
(1-bel_ap(iup)));
        else
          bel_post(iup)=(1-p_obs(iup,2))*bel_ap(iup)/((1-
p_obs(iup,2))*bel_ap(iup)+(1-p_obs(iup,1))*(1-bel_ap(iup)));
        end
      end
      bel_ap;
      bel_post;
      global_occupancy(obsind2)=bel_post;
    end
  end
  call_count=call_count+1;
  iloop=iloop+1;
end
% ------------------------------------------------------------------
% Destination approach
% ------------------------------------------------------------------
[action_history, corrective_action] = CC(M, N, depth, iloop, call_count,
opt_value, opt_act, pos_on_the_road, action_history, c_clash);
if pos_on_the_road(iloop)==N*M-N-1
  call_count = call_count+1; iloop = iloop+1; opt_value = 0;
  better_move = sort([rew(N*M-1), N*M-1, 3;rew(N*M-N), N*M-N, 1],'descend');
  pos_on_the_road(iloop) = better_move(1,2);
  opt_act = better_move(1,3);
  [action_history, corrective_action] = CC(M, N, depth, iloop, call_count,
opt_value, opt_act, pos_on_the_road, action_history, c_clash);
  fprintf('At time step %3i \n',call_count)
  fprintf('The Destination has been reached \n')
  hold on;
  plot3(ceil(pos_on_the_road(iloop)/N),pos_on_the_road(iloop)-
N*(ceil(pos_on_the_road(iloop)/N)-1), zeros(length(iloop),1),'k*');
  drawnow;
end
if pos_on_the_road(iloop)==N*M-1;
  call_count = call_count+1; iloop = iloop+1; opt_act = 1; opt_value = 0;
  pos_on_the_road(iloop)=N*M;
  [action_history, corrective_action] = CC(M, N, depth, iloop, call_count,
opt_value, opt_act, pos_on_the_road, action_history, c_clash);
  fprintf('At time step %3i \n',call_count)
  fprintf('The Destination has been reached \n')
  hold on;
  plot3(ceil(pos_on_the_road(iloop)/N),pos_on_the_road(iloop)-
N*(ceil(pos_on_the_road(iloop)/N)-1), zeros(length(iloop),1),'k*');
  drawnow;
elseif pos_on_the_road(iloop)==N*M-N;
  call_count = call_count+1; iloop = iloop+1; opt_act = 3; opt_value = 0;
  pos_on_the_road(iloop)=N*M;
  [action_history, corrective_action] = CC(M, N, depth, iloop, call_count,
opt_value, opt_act, pos_on_the_road, action_history, c_clash);
  fprintf('At time step %3i \n',call_count)
  fprintf('The Destination has been reached \n')
  hold on;
  plot3(ceil(pos_on_the_road(iloop)/N),pos_on_the_road(iloop)-
N*(ceil(pos_on_the_road(iloop)/N)-1), zeros(length(iloop),1),'k*');
```

```
  drawnow;
end
T=toc; % timer stop
% ---------------------------------------------------------------------
% Obstacle mapping based on belief
% ---------------------------------------------------------------------
GO_mtrx=reshape(global_occupancy,N,M);
figure (f_c+1)
set (figure(f_c+1), 'Position' , [100 100 500 500])
mesh(GO_mtrx)
% ---------------------------------------------------------------------
% Actual path
% ---------------------------------------------------------------------
opt_of_pathx=zeros(call_count,1);
opt_of_pathy=zeros(call_count,1);
for i=1:call_count
  opt_of_pathy(i)=ceil(action_history(i,4)/N);
  opt_of_pathx(i)=action_history(i,4)-N*(ceil(action_history(i,4)/N)-1);
end
figure (f_c+2)
hold on
set (figure(f_c+2), 'Position' , [100 100 500 500])
mesh(rew_matr);
plot3(positions_xy(:,2),positions_xy(:,1),zeros(no_positions,1),'r*');
plot3(opt_of_pathy,opt_of_pathx,zeros(length(opt_of_pathx)),'m')
```

APPENDIX 3

The Vertices Observations Guide

```
% Observation sets for all positions on the grids; in the order of distance
% ------------------------------------------------------------------------
obs_sets=zeros(N*M,8,2); %First index WM position, 2nd the type of observation
% 3rd points observed
for ii=1:M
  for jj=1:N
    pres_pos=(ii-1)*N+jj;
% ------------------------------------------------------------------------
% NE observation
% ------------------------------------------------------------------------
    icount=0;
    if pres_pos-N>0
      icount=icount+1;
      obs_sets(pres_pos,1,icount)=pres_pos-N;
      if ceil(pres_pos/N)*N~=pres_pos
        icount=icount+1;
        obs_sets(pres_pos,1,icount)=pres_pos-N+1;
      end
    end
% ------------------------------------------------------------------------
% EN observation
% ------------------------------------------------------------------------
    icount=0;
    if ceil(pres_pos/N)*N~=pres_pos;
      icount=icount+1;
      obs_sets(pres_pos,2,icount)=pres_pos+1;
      if pres_pos-N>0
        icount=icount+1;
        obs_sets(pres_pos,2,icount)=pres_pos-N+1;
      end
    end
% ------------------------------------------------------------------------
% ES observation
% ------------------------------------------------------------------------
    icount=0;
    if ceil(pres_pos/N)*N~=pres_pos;
      icount=icount+1;
      obs_sets(pres_pos,3,icount)=pres_pos+1;
      if pres_pos+N<N*M
        icount=icount+1;
        obs_sets(pres_pos,3,icount)=pres_pos+N+1;
      end
    end
% ------------------------------------------------------------------------
% SE observation
% ------------------------------------------------------------------------
    icount=0;
    if pres_pos+N<N*M
      icount=icount+1;
      obs_sets(pres_pos,4,icount)=pres_pos+N;
      if ceil(pres_pos/N)*N~=pres_pos
        icount=icount+1;
        obs_sets(pres_pos,4,icount)=pres_pos+N+1;
      end
    end
% ------------------------------------------------------------------------
% SW observation
% ------------------------------------------------------------------------
    icount=0;
    if pres_pos+N<N*M
      icount=icount+1;
      obs_sets(pres_pos,5,icount)=pres_pos+N;
      if ceil((pres_pos-1)/N)*N+1~=pres_pos
```

```
        icount=icount+1;
        obs_sets(pres_pos,5,icount)=pres_pos+N-1;
      end
    end
% ------------------------------------------------------------------------
% WS observation
% ------------------------------------------------------------------------
    icount=0;
    if ceil((pres_pos-1)/N)*N+1~=pres_pos;
      icount=icount+1;
      obs_sets(pres_pos,6,icount)=pres_pos-1;
      if pres_pos+N<N*M
        icount=icount+1;
        obs_sets(pres_pos,6,icount)=pres_pos+N-1;
      end
    end
% ------------------------------------------------------------------------
% WN observation
% ------------------------------------------------------------------------
    icount=0;
    if ceil((pres_pos-1)/N)*N+1~=pres_pos;
      icount=icount+1;
      obs_sets(pres_pos,7,icount)=pres_pos-1;
      if pres_pos-N>0
        icount=icount+1;
        obs_sets(pres_pos,7,icount)=pres_pos-N-1;
      end
    end
% ------------------------------------------------------------------------
% NW observation
% ------------------------------------------------------------------------
    icount=0;
    if pres_pos-N>0
      icount=icount+1;
      obs_sets(pres_pos,8,icount)=pres_pos-N;
      if ceil((pres_pos-1)/N)*N+1~=pres_pos
        icount=icount+1;
        obs_sets(pres_pos,8,icount)=pres_pos-N-1;
      end
    end
  end
end
clear icount ii jj pres_pos
```

APPENDIX 4

The Optimum Action Determinant

```
function
[opt_value,opt_act,value]=OAD_S(depth,state,indsa,loc_rew,loc_rew_end,loc_step
s,loc_obs_sets,c_O,c_clash,alpha,p_obs,pos_vis)
% ------------------------------------------------------------------
% On-line optimization of actions
% ------------------------------------------------------------------
% From the current WM location, looks n-steps (varying depth) ahead to choose
the optimal immediate (single) action, which can be any of the four move
actions or eight measurement actions. Function calls itself recursive n-1
times. At the last step of the horizon only a move action can be made.
% IN:
% depth: how many stages in time included in the optimization
% state: system state representation = position of the wm and belief about MO
location
% indsa: mapping between local (i) and global (j) WM coordinates j=indsa(i)
% loc_rew: local reward for the position of the WM
% loc_rew_end: local reward including the result of off-line optimization, to
be used if depth=1
% loc_steps: WM move options
% c_O: cost of observation
% c_clash: cost of WM and MO clashing
% p_MO: probability of MO moving to a random direction (NESW)
% alpha: discounting factor
% OUT:
% opt_value=optimal value of the reward function
% opt_state=state resulting from the optimal action
% opt_act=optimal action

% 12 actions possible, specified by 4 move actions in loc_steps and eight
measurement actions. Value related to each action initialized to zero, value
related to each observation initialized to large negative.
value=zeros(12,1);
value(5:12)=-1e6;
% ------------------------------------------------------------------
% Control actions
% ------------------------------------------------------------------
% The depth of the problem is still larger than one (not the final step)
if depth>1
    for i=1:4 % All move actions considered
    % Local coordinate update for the actions considered
    state_new=state;
    incoro=find(indsa==state(1)); % Local coordinate of WM old position
    state_new(1)=loc_steps(incoro,i); % Global coordinate of new position
    incorn=find(indsa==state_new(1)); %Local coordinate of WM new position
% Value of state is immediate state value + immediate value of clash + future
value (recursive call to the function itself)
value(i)=loc_rew(incorn)+c_clash*state_new(incorn+1)+alpha*OAD_S(depth-
1,state_new,indsa,loc_rew,
loc_rew_end,loc_steps,loc_obs_sets,c_O,c_clash,alpha,p_obs,pos_vis);
  end
else % The last/final step of the problem, no call to the function itself
  for i=1:4
    state_new=state;
    incoro=find(indsa==state(1)); % Local coordinate of WM old position
    state_new(1)=loc_steps(incoro,i); % Global coordinate of new position
    incorn=find(indsa==state_new(1)); %Local coordinate of WM new position
Value of state is immediate state value + immediate value of clash; immediate
state value at the last step of the horizon(depth) is the value function value
of the deterministic offline problem.
    value(i)=loc_rew_end(incorn)+c_clash*state_new(incorn+1);
  end
end
% ------------------------------------------------------------------
```

```
% Measurement actions
% -------------------------------------------------------------------
% The depth of the problem is still larger than one (not the final step)
if depth>1
    for imeas=1:8 % All possible measurements considered
    % Local coordinate of the present position
    incoro=find(indsa==state(1));
     % Find the set of sites (in global coordinates) that can be
     % observed from current state with measurement imeas
    obsind=loc_obs_sets(incoro,imeas,:);
    apuind=find(obsind~=0);
    % All the sites that can be observed
    if length(apuind)>0
      % Measurement locations in local coordinates
      obsind2=zeros(length(apuind),1);
      for il=1:length(apuind)
        obsind2(il)=find(indsa==obsind(apuind(il)));
      end
% -------------------------------------------------------------------
% Measurement result combinations
% -------------------------------------------------------------------
% Generate all possible measurement results; 5 sites are observed each giving
% a binary result -> 32 possibilities in the most general case (current
% position not at the edges)
      ggg = (dec2bin(0:31))';
      gcc = textscan(ggg(:),'%1d%1d%1d%1d%1d','CollectOutput',true);
      FM = gcc{1};
      FM=FM(:,5:-1:1);
      % The real observation combinations (results) in this
      % particular observation-position case
      Nmeas=2^length(apuind);
      result=FM(1:2^length(apuind),1:length(apuind));
      result=str2num(int2str(result));
% -------------------------------------------------------------------
% Read prior probabilities of belief state
% -------------------------------------------------------------------
      bel_ap=zeros(length(obsind2),1);
      bel_ap(1:length(obsind2))=state(obsind2+1);
% -------------------------------------------------------------------
% Calculate prior probabilities for each measurement result
% -------------------------------------------------------------------
      pr_ap_meas=zeros(2^length(apuind),1);
      for iapm=1:2^length(apuind)
pr_ap_meas(iapm)=prod(bel_ap.^(result(iapm,:)'))*prod((1-bel_ap).^(1-
result(iapm,:)'));
      end
      bel_post=zeros(length(apuind),2^length(apuind));
% -------------------------------------------------------------------
% Calculate posterior probabilities of states for each measurement result;
% only those sites that are affected by the observation are updated
% -------------------------------------------------------------------
      for iapm2=1:(2^length(apuind))
        for ist=1:length(apuind)
          if result(iapm2,ist)==1
bel_post(ist,iapm2)=p_obs(ist,2)*bel_ap(ist)/(p_obs(ist,2)*bel_ap(ist)+p_obs(i
st,1)*(1-bel_ap(ist)));
          elseif result(iapm2,ist)==0
            bel_post(ist,iapm2)=(1-p_obs(ist,2))*bel_ap(ist)/((1-
p_obs(ist,2))*bel_ap(ist)+(1-p_obs(ist,1))*(1-bel_ap(ist)));
          else
            'Undefined measurement result'
            result(iapm2)
          end
        end
      end
% -------------------------------------------------------------------
% State update for each possible measurement result (no move)
% -------------------------------------------------------------------
```

```
      new_states=zeros(length(state),2^length(apuind));
      for ir2=1:(2^length(apuind))
        new_states(:,ir2)=state;
      new_states(obsind2+1,ir2)=bel_post(1:length(apuind),ir2);
      end
% ------------------------------------------------------------------
% Calculate the expected value of the action (observation) when making this
% particular observation
% ------------------------------------------------------------------
      value(4+imeas)=0;
      for ir3=1:(2^length(apuind))
        incorn=find(indsa==new_states(1,ir3)); % Local coordinate of WM
% Value is state value (no move) cost of observations and expected value of
% futures steps
value(4+imeas)=value(4+imeas)+pr_ap_meas(ir3)*(c_O+loc_rew(incorn)+alpha*OAD_S
(depth-1,new_states(:,ir3),
indsa,loc_rew,loc_rew_end,loc_steps,loc_obs_sets,c_O,c_clash,
alpha,p_obs,pos_vis));
      end

    end
  end
end
% ------------------------------------------------------------------
% Optimal value and the corresponding optimal action
% ------------------------------------------------------------------
opt_value=max(value);
inda=find(value==opt_value);
opt_act=inda(1);
end
```

APPENDIX 5

The Command Controller

```
function [ action_history, corrective_action ] = CC (M, N, depth, iloop,
call_count, opt_value, opt_act, pos_on_the_road, action_history, c_clash)
% this function keeps a tight track of whatever value being generated in the
% online tester or deterministic path finder. Thus the correct actions could
% be carried out at the right times. Corrective action will indicate the right
% path to go in case of encountering an abnormality. corrective action=0: no
% immediate action required, corrective action=100: OL-re-routing is advised,
% corrective action=200: ML-re-routing is advised, corrective action=250:
% reduce the cost of collision, corrective action=300: increase the cost of
% collision, corrective action=350: set the collision cost back to the initial
% and increase the depth, corrective action=400: increase the depth,
% corrective action=500: reduce the depth
% ------------------------------------------------------------------
corrective_action=0;
action_history(call_count, 1)=iloop;
action_history(call_count, 2)=opt_act;
action_history(call_count, 3)=opt_value;
action_history(call_count, 4)=pos_on_the_road(iloop);
action_history(call_count, 5)=c_clash;
action_history(call_count, 6)=depth;
action_history(call_count, 7:8)=0;
% ------------------------------------------------------------------
%% looking for observation loops
% ------------------------------------------------------------------
% repeating single observation
no_sdo=7; % Number of past steps considered for single direction observation
if call_count > no_sdo & action_history(call_count-no_sdo:call_count, 2)>=5 &
~ismember(100,action_history(call_count-no_sdo:call_count, 7)) &
length(unique(action_history(call_count-no_sdo:call_count, 2)))==1;
  corrective_action=100;
  action_history(call_count, 7)=corrective_action;
  action_history(call_count, 8)=1;
end
% repeating observations
no_mdo=35; % Number of past steps considered for Multi direction observation
if call_count > no_mdo & ...
    action_history(call_count-no_mdo:call_count, 2)>=5 &...
    ~ismember(100,action_history(call_count-no_mdo:call_count, 7));
  corrective_action=100;
  action_history(call_count, 7)=corrective_action;
  action_history(call_count, 8)=2;
end
% ------------------------------------------------------------------
% looking for motion loops
% ------------------------------------------------------------------
% reciprocating motion
no_rm=8; % Number of past steps considered for Reciprocating motion
if call_count > no_rm & action_history (call_count-no_rm:call_count, 2) < 5 &
~ismember(200,action_history(call_count-no_rm:call_count, 7)) &
length(unique(action_history(call_count-no_rm:call_count, 4)))==2;
  corrective_action=200;
  action_history(call_count, 7)=corrective_action;
  action_history(call_count, 8)=3;
end
% circular motion
no_cm=16; % Number of past steps to be considered for circular motion
if call_count > 16 & action_history (call_count-no_cm:call_count, 2) < 5 &
~ismember(200,action_history(call_count-no_cm:call_count, 7)) &
length(unique(action_history(call_count-no_cm:call_count, 4)))==4;
  corrective_action=200;
  action_history(call_count, 7)=corrective_action;
  action_history(call_count, 8)=4;
end
```

```
% ------------------------------------------------------------------
% ------------------------------------------------------------------
% looking for incautious motions
% ------------------------------------------------------------------
% looking for the pattern based on observations/motions ratio
no_cr_ca=7; % Number of past steps considered for Cost adjustment, also this
% represents the space between two possible warnings
lim_omr_ca=0.3; % observation to motion ratio limit
if iloop > no_cr_ca && ~ismember(300,action_history(call_count-no_cr_ca:end,
7)) && ~ismember(350,action_history(call_count-no_cr_ca:end, 7))
  pa_op=action_history(call_count-iloop+1:call_count, 2);
  mobo=hist(pa_op, [4 5]); % number of motions and observations in past steps
  omr_ca=mobo(2)/(mobo(1)+mobo(2)); % observation to motion ratio
  if omr_ca <= lim_omr_ca
    if action_history(call_count, 5) > -10000
      corrective_action=300;
      action_history(call_count, 7)=corrective_action;
      action_history(call_count, 8)=5;
    else % this condition can in fact be a part of secondary loop regulator
      corrective_action=350;
      action_history(call_count, 7)=corrective_action;
    end
  end
end
% Incautious motion pattern based on number of motion steps without
% observations
no_im=2; % Number of past steps to be considered for incautious motion
if call_count > no_im & action_history (call_count-no_im:call_count, 2) < 5
  if action_history(call_count, 5) > -10000
    corrective_action=300;
    action_history(call_count, 7)=corrective_action;
    action_history(call_count, 8)=6;
  else % this condition can in fact be a part of secondary loop regulator
    corrective_action=350;
    action_history(call_count, 7)=corrective_action;
  end
end
% ------------------------------------------------------------------
% Secondary loop controller
% ------------------------------------------------------------------
if action_history(call_count, 7)~=0
  % defining the immidiate vicinity around the wm position
  v_dim=2; % dimension on the vicinity
  pos=action_history(call_count, 4);
  ny_t=max(1,floor(pos/N)+1-v_dim);
  ny_p=min(M,floor(pos/N)+1+v_dim);
  nx_t=max(1,pos-N*floor((pos-1)/N)-v_dim);
  nx_p=min(N,pos-N*floor((pos-1)/N)+v_dim);
  vicinity=[]; %% Global coordinates of local area
  for iv=ny_t:ny_p
    vicinity=[vicinity ((iv-1)*N+nx_t:((iv-1)*N+nx_p))];
  end
  % determining when in past the robot has been in a location belonging to the
% vicinity defined above
  vil=length(vicinity);
  vin1=zeros(36, vil);
  for il=1:vil;
    vin=find(action_history(1:end-1, 4)==vicinity(il));
    for jl=1:length(vin);
      vin1(jl,il)=vin(jl);
    end
  end
  [z,n,V]=find(vin1);
  % retrieving the type of the warnings that might have been issued
  viwar=action_history(V, 7);
  n=hist(viwar, [0 160 260 310 360 410 510]);
  % loop controller
  if n(2) > 2
```

```
    corrective_action=400;
    action_history(call_count, 7)=corrective_action;
  end
  if n(4) > 2
    corrective_action=350;
    action_history(call_count, 7)=corrective_action;
  end
end
% ------------------------------------------------------------------
% normal conditions revision
% ------------------------------------------------------------------
% Collision cost reduction
no_cr=12; % Number of past steps considered for Cost reduction
lim_omr=0.8; % observation to motion ratio limit
if iloop > no_cr & unique(action_history (call_count-no_cr:call_count, 7))==0
& unique(action_history (call_count-no_cr:call_count, 4))~=1
  ls=action_history (call_count-iloop+1:call_count, 2); % iloop steps.
  numo=hist(ls, [4 5]); % number of motions and observations in past steps
  omr=numo(2)/(numo(1)+numo(2)); % observation to motion ratio
  if action_history(call_count, 6) <= 3 & action_history(call_count, 5) ~= -
2500
    if omr >= lim_omr
      corrective_action=250;
      action_history(call_count, 7)=corrective_action;
    end
  end
  if action_history(call_count, 6) > 3 & action_history(call_count, 5) ~= -
4000
    if omr >= lim_omr
      corrective_action=250;
      action_history(call_count, 7)=corrective_action;
    end
  end
end
% Depth reduction
no_dr=18; % Number of past steps to be considered for depth reduction
if call_count > no_dr & unique(action_history (call_count-no_cr:call_count,
4))~=1
  if action_history(end, 6) > action_history(1, 6)
    SC=length(unique(action_history(call_count-no_dr: call_count, 7)));
    if unique(action_history(call_count-no_dr: call_count, 7))==0
      corrective_action=500;
      action_history(call_count, 7)=corrective_action;
    elseif SC==2;
      if unique(action_history(call_count-no_dr: call_count, 7))==[0;250]
        corrective_action=500;
        action_history(call_count, 7)=corrective_action;
      end
    end
  end
end
end
```

APPENDIX 6

Modifications to the OPF following the CC

```
% Command controller
% ---------------------------------------------------------------------
% The anomaly controller function is being called to check the results of the
% optimization function and take a decisive action if needed
[action_history, corrective_action] = CC(M, N, depth, iloop, call_count,
opt_value, opt_act, pos_on_the_road, action_history, c_clash);
  if corrective_action==100 | corrective_action==200;
    if corrective_action==100
      fprintf('Abonrmality of type "observations loop" has been detected; Re-
routing \n\n')
    else
      fprintf('Abonrmality of type "Reciprocating or circular motions" has
been detected; Re-routing \n\n')
    end
    start_grid=pos_on_the_road(iloop);
    new_obs=find(global_occupancy >= 0.55);
    new_obs_op=global_occupancy(new_obs);
    [steps, opt_val, rew, alpha, optimal_path,opt_pathx, opt_pathy,
on_the_way_cost, N, M, rew_matr, p_count, start_grid]=OPP(start_grid, rew,
new_obs, new_obs_op);
    iroad=p_count*5;
    pos_on_the_road=zeros(iroad+1,1);
    act_on_the_road=zeros(iroad,1);
    pos_on_the_road(1)=optimal_path(start_grid);
    z_opt=zeros(1,p_count-1);
    refreshdata (reff)
    drawnow
    if optimal_path ~= M*N+1;
      fprintf('All passages are blocked; the programs execution is terminated
\n\n')
      T=toc; % timer stop
      break
    end
    if depth<=3
      c_clash=-4000;
    else
      c_clash=-5000;
    end
    fprintf('The collision cost is temporarily increased \n')
    fprintf('current cost is %2i \n\n',c_clash)
    iloop=0;
  end
  if corrective_action==250;
    if depth <= 3 & c_clash < -2500
      c_clash=c_clash+1500;
      fprintf('The collision cost has been lowered due to smooth operation
\n')
      fprintf('current depth is %2i \n\n',c_clash)
    elseif depth > 3 & c_clash < -4000
      c_clash=c_clash+1000;
      fprintf('The collision cost has been lowered due to smooth operation
\n')
      fprintf('current depth is %2i \n\n',c_clash)
    end
  end
  if corrective_action==300;
    if depth <= 3
      c_clash=c_clash-1500;
      fprintf('Abonrmality of type "incautious motions" has been detected;
collision cost is temporarily increased \n')
      fprintf('current cost is %2i \n\n',c_clash)
    else
      c_clash=c_clash-1000;
```

```
      fprintf('Abonrmality of type "incautious motions" has been detected;
collision cost is temporarily increased \n')
      fprintf('current cost is %2i \n\n',c_clash)
    end
  end
  if corrective_action==350;
    depth=depth+1;
    if depth<=3
      c_clash=-2500;
    else
      c_clash=-4000;
    end
    fprintf('Abonrmality of type "consistent incautious motions" has been
detected; depth is temporarily increased \n')
    fprintf('current depth is %2i \n\n',depth)
    fprintf('The collision cost has been set to initial value \n')
    fprintf('current cost is %2i \n\n',c_clash)
  end
  if corrective_action==400;
    depth=depth+1;
    if depth<=3
      c_clash=-2500;
    else
      c_clash=-4000;
    end
    fprintf('The depth has increased due to unsuccessful re-routing \n')
    fprintf('current depth is %2i \n\n',depth)
    fprintf('The collision cost has been set to initial value \n')
    fprintf('current cost is %2i \n\n',c_clash)
  end
  if corrective_action==500;
    depth=depth-1;
    fprintf('The depth has been lowered due to smooth operation \n')
    fprintf('current depth is %2i \n\n',depth)
  end
```

APPENDIX 7

The OPF in dynamic environment

```
clear all
close all
clc
tic;
%% Primary offline-planning
 [steps, opt_val, rew, alpha, optimal_path,opt_pathx, opt_pathy,...
    on_the_way_cost, N, M, rew_matr, p_count, start_grid]=OPP(1, [], [], []);
%% On-line part
%% System state
% System state is location of WM and belief vector of location of MO
depth=3; % the depth starts at 2 but the program could go deeper in horizen,
should the need ever arises
l_dim=5;
if depth<=3
    c_clash=-2500; %% Initial Cost of clash
elseif depth>3
    c_clash=-4000;
end
cii=0; % Initial collision increment index
record_call_count=0;
c_O=-0; %% Cost of observation
action_history=zeros(1,6); % initial action history
A_dim=2*l_dim+1; % Size of area considered in one on-line decision
p_obs=[0.01 0.99; 0.05 0.95]; %Probabilites of observing MO as a function of
position 1:no MO, 2: is MO
qw=0.1;
% complete ignorance about occupancy in the beginning
global_occupancy=0.1*ones(N*M,1);
% Position of initial MOs, changes are in random positions
no_positions = 8;
positions=RW(N,M, 1, no_positions, [], [], qw ); % random positions
positions_xy = [positions-N*(ceil(positions/N)-1),ceil(positions/N)];
z_positions=zeros(no_positions,1);
% length of the road studied
iroad=p_count*5; % length of the path studied = one step till the off-line
path hits the goal
pos_on_the_road=zeros(iroad+1,1);
act_on_the_road=zeros(iroad,1);
pos_on_the_road(1)=optimal_path(start_grid); % Same starting point as in the
optimal path studied in script OPP
%% Observation sets
generate_observation_sets_small;
%% Mapping of area of entire problem to local optimization problem
f_c=1; % figure counter
figure(f_c);
set (figure(f_c), 'Position' , [100 100 500 500])
mesh(rew_matr);
hold on;
plot3(opt_pathy(1:p_count-1),opt_pathx(1:p_count-1),zeros(1,p_count-1));
reff=plot3(positions_xy(:,2),positions_xy(:,1),z_positions,'r*'); % refreshing
figure
set(reff,'XDataSource','positions_xy(:,2)');
set(reff,'YDataSource','positions_xy(:,1)');
set(reff,'ZDataSource','z_positions');
iloop=1;
call_count=1;
while iloop < iroad+1 & pos_on_the_road(iloop)~=N*M-N &
pos_on_the_road(iloop)~=N*M-1 & pos_on_the_road(iloop)~=N*M-N-1
    %% Random walker
    [positions]=RW([],[],0,no_positions, steps, positions, qw);
    positions_xy = [positions-N*(ceil(positions/N)-1),ceil(positions/N)];
    refreshdata (reff)
    drawnow
```

```
    %% indsa is the mapping vector
    pos_WM=pos_on_the_road(iloop);
    N0=floor(pos_WM/N)+1;
    NY_start=max(1,N0-l_dim);
    NY_stop=min(M,N0+l_dim);
    NX_start=max(1,pos_WM-N*floor((pos_WM-1)/N)-l_dim);
    NX_stop=min(N,pos_WM-N*floor((pos_WM-1)/N)+l_dim);
    indsa=[]; %% Global coordinates of local area
    for i=NY_start:NY_stop
        indsa=[indsa ((i-1)*N+NX_start:((i-1)*N+NX_stop))];
    end
    %% WM steps on the local area
    loc_steps=steps(indsa,:);
    loc_rew=rew(indsa); %Local immediate rewards
    loc_rew_end=rew(indsa)+alpha*opt_val(indsa); % Rewards at the end step of
on-line optimization
    loc_obs_sets=obs_sets(indsa,:,:);
    state=zeros(length(indsa)+1,1);
    state(1)=pos_WM;
    state(2:end)=global_occupancy(indsa); %% To take care of the irregular
global belief
    %% On-line optimization of actions
    global_occupancy_ref=global_occupancy; % referncing global_occupancy
before changes being made to it
    pos_vis=pos_on_the_road(1:iloop);

[opt_value,opt_act,value]=OAD_D(depth,state,indsa,loc_rew,loc_rew_end,loc_step
s...
        ,loc_obs_sets,c_O,c_clash,alpha,p_obs,pos_vis, global_occupancy,
obs_sets, steps, qw);%% observation as chosen/optimized and updating belief
    global_occupancy=global_occupancy_ref; % restoring the global_occupancy to
the original
    %% After optimization update position or belief depending on action
    fprintf('At time step %3i \n',call_count)
    fprintf('optimal action is %2i \n',opt_act)
    if opt_act<5 %% Step taken, no measurement made
pos_on_the_road(iloop+1)=steps(pos_on_the_road(iloop),opt_act);
        fprintf('and new position of the WM is %3i
\n\n',pos_on_the_road(iloop+1))
        hold on;
plot3(ceil(pos_on_the_road(iloop+1)/N),pos_on_the_road(iloop+1)-
N*(ceil(pos_on_the_road(iloop+1)/N)-1), zeros(length(iloop+1),1),'k*');
        drawnow;
        hold off;
    else %% Measurement made, no step taken
        pos_on_the_road(iloop+1)=pos_on_the_road(iloop);
        fprintf('and the WM holds its position at %3i
\n\n',pos_on_the_road(iloop))
        hold on;
plot3(ceil(pos_on_the_road(iloop+1)/N),pos_on_the_road(iloop+1)-
N*(ceil(pos_on_the_road(iloop+1)/N)-1),zeros(length(iloop+1),1),'ko');
        drawnow;
        hold off;
        imeas=opt_act-4; %This is the optimal measurement

        % Making the measurement imeas; each point measured for occupancy
        % or not
        obsind=obs_sets(pos_WM,imeas,:);
        apuind=find(obsind~=0);

        if ~isempty(apuind)
            obsind2=zeros(length(apuind),1);
            for il=1:length(apuind)
                obsind2(il)=obs_sets(pos_WM,imeas,apuind(il));
            end
            %Simulate the measurement result (value=1, no observation; =2
            %is observed)
            result=zeros(length(apuind),1);
```

```
            for imi=1:length(apuind)
                if ismember(obsind2(imi),positions)
                    ptest=p_obs(imi,2);
                else
                    ptest=p_obs(imi,1);
                end
                if rand<ptest
                    result(imi)=1;
                end
            end
            result;
            % Update ocuupancy belief
            bel_ap=global_occupancy(obsind2);
            bel_post=zeros(length(obsind2),1);
            for iup=1:length(apuind)
                if result(iup)==1
bel_post(iup)=p_obs(iup,2)*bel_ap(iup)/(p_obs(iup,2)*bel_ap(iup)+p_obs(iup,1)*
(1-bel_ap(iup)));
                else
                    bel_post(iup)=(1-p_obs(iup,2))*bel_ap(iup)/((1-
p_obs(iup,2))*bel_ap(iup)+(1-p_obs(iup,1))*(1-bel_ap(iup)));
                end
            end
            bel_ap;
            bel_post;
            global_occupancy(obsind2)=bel_post;
        end
        [ global_occupancy ] = occupancy_updater( qw, global_occupancy,
obs_sets, steps, 0, [], [] );
    end
    %% Command controller
%The anomaly controller function is being called to check the
results of the optimizaion function and take a decisive action if needed
    [action_history, corrective_action] = command_controller(M, N, depth,
iloop, call_count, opt_value, opt_act, pos_on_the_road, action_history,
c_clash);

    if corrective_action==100 | corrective_action==200;
        if corrective_action==100
            fprintf('Abonrmality of type "observations loop" has been
detected; Re-routing \n\n')
        else
            fprintf('Abonrmality of type "Reciprocating or circular motions"
has been detected; Re-routing \n\n')
        end
        start_grid=pos_on_the_road(iloop);
        new_obs=find(global_occupancy >= 0.55);
        new_obs_op=global_occupancy(new_obs);
        [steps, opt_val, rew, alpha, optimal_path,opt_pathx, opt_pathy,
on_the_way_cost, N, M, rew_matr, p_count, start_grid]=OPP(start_grid, rew,
new_obs, new_obs_op);
        iroad=p_count*5;
        pos_on_the_road=zeros(iroad+1,1);
        act_on_the_road=zeros(iroad,1);
        pos_on_the_road(1)=optimal_path(start_grid);

        f_c=f_c+1;
        figure(f_c);
        set (figure(f_c), 'Position' , [100 100 500 500])
        mesh(rew_matr);
        hold on;
        plot3(opt_pathy(1:p_count-1),opt_pathx(1:p_count-1),zeros(1,p_count-
1));
reff=plot3(positions_xy(:,2),positions_xy(:,1),z_positions,'r*');
        set(reff,'XDataSource','positions_xy(:,2)');
        set(reff,'YDataSource','positions_xy(:,1)');
        set(reff,'ZDataSource','z_positions');
        if optimal_path ~= M*N+1;
```

```
            fprintf('All passages are blocked; the programs execution is
terminated \n\n')
            break
        end
        if depth<=3
            c_clash=-4000;
        else
            c_clash=-5000;
        end
        fprintf('The collision cost is temporarily increased \n')
        fprintf('current cost is %2i \n\n',c_clash)
        iloop=0;
    end
    if corrective_action==250;
        if depth <= 3 & c_clash < -2500
            c_clash=c_clash+1500;
            fprintf('The collision cost has been lowered due to smooth
operation \n')
            fprintf('current depth is %2i \n\n',c_clash)
        elseif depth > 3 & c_clash < -4000
            c_clash=c_clash+1000;
            fprintf('The collision cost has been lowered due to smooth
operation \n')
            fprintf('current depth is %2i \n\n',c_clash)
        end
    end
    if corrective_action==300;
        if depth <= 3
            c_clash=c_clash-1500;
            fprintf('Abonrmality of type "incautious motions" has been
detected; collision cost is temporarily increased \n')
            fprintf('current cost is %2i \n\n',c_clash)
        else
            c_clash=c_clash-1000;
            fprintf('Abonrmality of type "incautious motions" has been
detected; collision cost is temporarily increased \n')
            fprintf('current cost is %2i \n\n',c_clash)
        end
    end
    if corrective_action==350;
        depth=depth+1;
        if depth<=3
            c_clash=-2500;
        else
            c_clash=-4000;
        end
        fprintf('Abonrmality of type "consistent incautious motions" has been
detected; depth is temporarily increased \n')
        fprintf('current depth is %2i \n\n',depth)
        fprintf('The collision cost has been set to initial value \n')
        fprintf('current cost is %2i \n\n',c_clash)
    end
    if corrective_action==400;
        depth=depth+1;
        if depth<=3
            c_clash=-2500;
        else
            c_clash=-4000;
        end
        fprintf('The depth has increased due to unsuccessful re-routing \n')
        fprintf('current depth is %2i \n\n',depth)
        fprintf('The collision cost has been set to initial value \n')
        fprintf('current cost is %2i \n\n',c_clash)
    end
    if corrective_action==500;
        depth=depth-1;
        fprintf('The depth has been lowered due to smooth operation \n')
        fprintf('current depth is %2i \n\n',depth)
```

```
    end
    call_count=call_count+1;
    iloop=iloop+1;
end
%% Destination approach
 [action_history, corrective_action] = command_controller(M, N, depth, iloop,
call_count, opt_value, opt_act, pos_on_the_road, action_history, c_clash);

if pos_on_the_road(iloop)==N*M-N-1
    call_count = call_count+1; iloop = iloop+1; opt_value = 0;
    better_move = sort([rew(N*M-1), N*M-1, 3;rew(N*M-N), N*M-N, 1],'descend');
    pos_on_the_road(iloop) = better_move(1,2);
    opt_act = better_move(1,3);
    [action_history, corrective_action] = command_controller(M, N, depth,
iloop, call_count, opt_value, opt_act, pos_on_the_road, action_history,
c_clash);
    fprintf('At time step %3i \n',call_count)
    fprintf('The Destination has been reached \n')
    hold on;
    plot3(ceil(pos_on_the_road(iloop)/N),pos_on_the_road(iloop)-
N*(ceil(pos_on_the_road(iloop)/N)-1), zeros(length(iloop),1),'k*');
    drawnow;
end
if pos_on_the_road(iloop)==N*M-1;
    call_count = call_count+1; iloop = iloop+1; opt_act = 1; opt_value = 0;
    pos_on_the_road(iloop)=N*M;
    [action_history, corrective_action] = command_controller(M, N, depth,
iloop, call_count, opt_value, opt_act, pos_on_the_road, action_history,
c_clash);
    fprintf('At time step %3i \n',call_count)
    fprintf('The Destination has been reached \n')
    hold on;
    plot3(ceil(pos_on_the_road(iloop)/N),pos_on_the_road(iloop)-
N*(ceil(pos_on_the_road(iloop)/N)-1), zeros(length(iloop),1),'k*');
    drawnow;
elseif pos_on_the_road(iloop)==N*M-N;
    call_count = call_count+1; iloop = iloop+1; opt_act = 3; opt_value = 0;
    pos_on_the_road(iloop)=N*M;
    [action_history, corrective_action] = command_controller(M, N, depth,
iloop, call_count, opt_value, opt_act, pos_on_the_road, action_history,
c_clash);
    fprintf('At time step %3i \n',call_count)
    fprintf('The Destination has been reached \n')
    hold on;
    plot3(ceil(pos_on_the_road(iloop)/N),pos_on_the_road(iloop)-
N*(ceil(pos_on_the_road(iloop)/N)-1), zeros(length(iloop),1),'k*');
    drawnow;
end
T=toc;
%% Obstruction mapping based on system belief
GO_mtrx=reshape(global_occupancy,N,M);
figure (f_c+1)
set (figure(f_c+1), 'Position' , [100 100 500 500])
mesh(GO_mtrx)

%% Actual path
opt_of_pathx=zeros(call_count,1);
opt_of_pathy=zeros(call_count,1);
for i=1:call_count
    opt_of_pathy(i)=ceil(action_history(i,4)/N);
    opt_of_pathx(i)=action_history(i,4)-N*(ceil(action_history(i,4)/N)-1);
end
figure (f_c+2)
set (figure(f_c+2), 'Position' , [100 100 500 500])
mesh(rew_matr);
hold on;
plot3(opt_of_pathy,opt_of_pathx,zeros(1,length(opt_of_pathx)),'m');
```

APPENDIX 8

The random walker

```
function [ walker_pos] = RW(N,M, first_call, no_walkers, steps, walker_pos, qw
)
% this module governs the movement of the random walker(s). the number of
% walker(s) and probability
% of them moving is introduced to the function by the OPF. it randomly
% chooses an action, based on the probabilities received, for each
% walker and returns the updated position to the online path finder.
if first_call == 1;
    %% Iitial random positions
    walker_pos=(randperm(N*M-2, no_walkers)+1)'; % random positions
else
    %% Random action generator (rag)
    stp=1-qw; % probability of a random walker staying in its place
    mp=qw/4; % probability of a random walker moving from its place
    for jj=1:no_walkers
        rag=rand(1);
        if rag <= stp
            wact(jj)=5; % walker action
        elseif rag > stp & rag <= stp+mp
            wact(jj)=1;
        elseif rag > stp+mp & rag <= stp+2*mp
            wact(jj)=2;
        elseif rag > stp+2*mp & rag <= stp+3*mp
            wact(jj)=3;
        elseif rag > stp+3*mp & rag <= 1
            wact(jj)=4;
        end
        %% Walker position selector
        if wact(jj)<5 %% Step taken
            walker_pos(jj)=steps(walker_pos (jj), wact(jj));
        else
            walker_pos(jj)=walker_pos (jj);
        end
    end
end
end
```

APPENDIX 9

The Occupancy Updater

```
function [ global_occupancy, state ] = occupancy_updater( qw,
global_occupancy, obs_sets, steps, partial, state, indsa )%% Cell Checker
for i=1:900
    CC(i,1)= obs_sets (i, 1, 1)~=0;
    CC(i,2)= obs_sets (i, 2, 1)~=0;
    CC(i,3)= obs_sets (i, 4, 1)~=0;
    CC(i,4)= obs_sets (i, 6, 1)~=0;
end
%% Whole grid update
qw_ied=qw/4; % probability of random walkers moving in each direction
for i=1:899
    global_occupancy(i)=(1-
qw_ied*(CC(i,1)+CC(i,2)+CC(i,3)+CC(i,4)))*global_occupancy(i)+qw_ied*(global_o
ccupancy(steps(i,4))*CC(i,1)+global_occupancy(steps(i,1))*CC(i,2)+global_occup
ancy(steps(i,3))*CC(i,3)+global_occupancy(steps(i,2))*CC(i,4));
end
if partial==1
    state(2:end)=global_occupancy(indsa);
end
end
```