THE UNIVERSITY OF SALFORD

SCHOOL OF COMPUTING, SCIENCE AND ENGINEERING

University of Salford MANCHESTER

Multi-Objective Reinforcement Learning Framework for Unknown Stochastic & Uncertain Environments

JOHN MICHAEL PINDER

Submitted in partial fulfilment of the requirements for the degree of:

DOCTOR OF PHILOSOPHY IN ADVANCED ROBOTICS AND AUTOMONOUS SYSTEMS 2016

Author: John Michael Pinder BSc, MSc

Supervised by: Prof Samia Nefti-Meziani

Co Supervised by: Dr Theodoros Theodoridis

SALFORD – UK

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This dissertation is the product of many years of endurance and effort and is the result of not just one individual, but of many, each one being equally important for its correct development.

To begin, I would like to thank my supervisor Prof Samia Nefti Meziani for this amazing opportunity to learn under her guidance in such a well-equipped robotics department at the University Of Salford. Also thanks to my co-supervisor Dr Theodoros Theodoridis for his initial guidance and introduction to the interesting field of Multi Objective Reinforcement Learning.

I would like to thank and acknowledge the remarkable work of Dr William Hinojosa in the development of the GPFRL algorithm which formed the foundations upon which this research was conducted.

I thank my parents and family who have never stopped supporting me, I thank their patience, commitment, support, and advice when I needed it the most. They always believed in me and encouraged me, showing me love and support making me the person I am today.

Special thanks to my beautiful partner Sarah whose continuing support and perseverance have allowed me to peruse my ambitious goals in life. I appreciate that moving into a house together, financial situations, and coping with the stresses of supporting a PHD has pushed our relationship to the limits but thank you for still believing in me and helping me in any way you could.

ACRONYMS

AI	Artificial Intelligence
DP	Dynamic Programming
FIS	Fuzzy Inference System
US	Ultra Sonic
MC	Monte Carlo Methods
MDP	Markov Decision Process
NN	Neural Networks
PFL	Probabilistic Fuzzy Logic
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
SARSA	State-Action-Reward State-Action
TD	Temporal Difference
MORL	Multi Objective Reinforcement Learning
MOQ	Multi Objective Q-Learning
MOSARSA	Multi Objective State Action Reward State Action
GPFMORL	Generalised Probabilistic Fuzzy Multi Objective Reinforcement Learning
UAV	Unmanned Aerial Vehicle
CDSM	Cellular Decomposed State Matrix
PFM	Potential Field Methodology

GLOSSARY

Actor-critic:

Refers to a class of agent architectures, where the actor plays out a particular policy, while the critic learns to evaluate the actor's policy. Both the actor and critic are simultaneously improving by bootstrapping on each other.

Agent:

A system that is embedded in an environment. The controller or decision-making entity choosing actions and learning to perform a task. Examples include mobile robots, software agents, or industrial controllers.

Average-reward methods:

A framework where the agent's goal is to maximize the expected payoff per step. Average-reward methods are appropriate in problems where the goal is to maximize the long-term performance. They are usually much more difficult to analyse than discounted algorithms.

Discount Factor:

A scalar value between 0 and 1 which determines the present value of future rewards. If the discount factor is 0, the agent is concerned with maximizing immediate rewards. As the discount factor approaches 1, the agent takes more future rewards into account. Algorithms which discount future rewards include Q-learning and TD (lambda).

Discounting:

If rewards received in the far future are worth less than rewards received sooner, they are described as being discounted. Humans and animals appear to discount future rewards hyperbolically; exponential discounting is common in engineering and finance.

Dynamic Programming:

A collection of calculation techniques for finding a policy that maximises reward or minimises costs. Is a class of solution methods for solving sequential decision problems with a compositional cost structure.

Environment:

The external system in which an agent is “embedded” which enables perception and action.

Episode:

A time segment of learning with task dependent starting and ending conditions.

Function Approximation:

Refers to the problem of inducing a function from training examples. Standard approximators include decision trees, neural networks, and nearest-neighbour methods.

Markov Chain:

A model for a random process that evolves over time such that the states (like locations in a maze) occupied in the future are independent of the states in the past given the current state.

Markov Decision Process:

A model for a controlled random process in which an agent's choice of action determines the probabilities of transitions of a Markov chain and lead to rewards (or costs) that need to be maximised (or minimised). Essentially, the outcome of applying an action to a state depends only on the current action and state (and not on preceding actions or states) .

Model:

The agent's view of the environment, which maps state-action pairs to probability distributions over states. Note that not every reinforcement learning agent uses a model of its environment. Basically it’s a mathematical description of the environment.

Model-based algorithms:

These compute value functions using a model of the system dynamics. Adaptive Real-time DP (ARTDP) is a well-known example of a model-based algorithm.

Model-free algorithms:

These directly learn a value function without requiring knowledge of the consequences of doing actions. Q-learning is the best known example of a model-free algorithm.

Monte Carlo Methods:

A class of methods for learning value functions, which estimates the value of a state by running many trials starting at that state, then averages the total rewards received on those trials.

Policy:

The decision-making function of the agent, which represents a mapping from situations to actions. Can be considered a deterministic or stochastic scheme for choosing an action at every state or location.

Policy Evaluation:

Determining the value of each state for a given policy.

Policy Improvement:

Forming a new policy that is better than the current one.

Policy Iteration:

Alternating steps of policy evaluation and policy improvement to converge to an optimal policy.

POMDP

Partially observable Markov decision problem. State information is available only through a set of observations.

Return:

The cumulative (discounted) reward for an entire episode.

Reward:

An immediate, possibly stochastic, payoff that results from performing an action in a state represented by a numerical signal to the learning agent indicating task progress or completion or the degree to which a state or action is desirable. Reward functions can be used to specify a wide range of planning goals

Sensor:

Agents perceive the state of their environment using sensors, which can refer to physical transducers, such as ultrasound, or simulated feature-detectors.

State:

This can be viewed as a summary of the past history of the system, which determines its future evolution.

Temporal Difference Algorithms:

A class of learning methods, based on the idea of comparing temporally successive predictions. Possibly the single most fundamental idea in all of reinforcement learning.

Temporal Difference Prediction Error:

A measure of the inconsistency between estimates of the value function at two successive states. This prediction error can be used to improve the predictions and also to choose good actions.

Unsupervised Learning:

The area of machine learning in which an agent learns from interaction with its environment, rather than from a knowledgeable teacher that specifies the action the agent should take in any given state.

Value Function:

Is a mapping from states to real numbers, where the value of a state represents the long-term reward achieved starting from that state, and executing a particular policy. The key distinguishing feature of RL methods is that they learn policies indirectly, by instead learning value functions. RL methods can be contrasted with direct optimization methods, such as genetic algorithms (GA), which attempt to search the policy space directly. A function defined over states, which gives an estimate of the total (possibly discounted) reward expected in the future, starting from each state, and following a particular policy.

Value Iteration:

A single iteration of policy evaluation followed by policy improvement.

“You only get out of life what you put in”

- Clifford Neal.

ABSTRACT

This dissertation focuses on the problem of uncertainty handling during learning, by agents dealing in stochastic environments by means of Multi Objective Reinforcement Learning (MORL). Most previous investigations into multi objective reinforcement learning have proposed algorithms to deal with the learning performance issues but have neglected the uncertainty present in stochastic environments. The realisation that multiple long term objectives are exhibited in many risky and uncertain real-world decision making problems forms the principle motivation of this research.

This dissertation proposes a novel modification to the single objective GPFRL algorithm (HIinojosa et al, 2008) where, the implementation of a linear scalarisation methodology provides a way to automatically find an optimal policy for multiple objectives under different kinds of uncertainty. The proposed Generalised Probabilistic Fuzzy Multi Objective Reinforcement Learning (GPFMORL) algorithm is further enhanced by the introduction of prospect theory to guarantee convergence by the means of risk evaluation. The simulated grid world increased in complexity as a further two complementary and conflicting objectives were specified whilst also introducing uncertainty in the form of stochastic cross winds.

Results obtained from the GPFMORL grid world simulations were compared against two more classical multi objective algorithms, MOQ and MOSARSA, showing not only a stronger convergence but also a much faster one. Experiments performed on an actual Quad-Copter/Drone demonstrated that the proposed algorithm and developed framework are both feasible and promising for the control of Artificially Intelligent (AI) Unmanned Aerial Vehicles (UAV) in a variety of real-world multi objective applications such as; autonomous landing/delivery or search and rescue.

Furthermore, the observed results of this work showed that the GPFMORL method can find its major real world application in the un-calibrated control of non-linear, multiple inputs, and multiple output systems, especially in multi objective situations with high uncertainty. Proposed novel case study research prototype examples include: Controlled Environment Agriculture for optimising Hydroponic Crop Growth by the proposed “Automated Solar Powered Environmental Controller” (ASPEC). Finally the “Robotic Dementia Medication Administration System” (RDMAS) attempts to optimise liquid medication dispensing via intelligent scheduling to more appropriate times of the day when the patient is more likely to remember to take their medication, based upon previous learned knowledge and experience.

Chapter 1

Introduction

1.1 Motivation

In situations where pre-programmed solutions are difficult or impossible to design, learning algorithms can be used to generate solutions for complex problems. Depending upon the level of available information, one or more types of learning can be applied. According to the connectionist learning approach (Hinton, 1989), these algorithms exist primarily in the form of Unsupervised, Supervised and Reinforcement Learning (RL). Unsupervised learning is only applicable when target information is not available and the agent attempts to form a model based upon association amongst data or clustering. Supervised learning however is much more powerful but requires the knowledge of output patterns corresponding to input data. However, in dynamic environments where the outcome of an action is not immediately known and is subject to change, correct target data may not be available at the moment of learning, this implies that supervised approaches cannot be applied. In these environments, reward information, whose availability can be only sparse, may be the best signal that the agent receives. For such systems, Reinforcement Learning has proven to be a more suitable method than supervised or unsupervised learning when the systems require a selection of actions whose consequences emerge over long periods of time for which input-output data are not available (Berenji and Khedkar, 1992). This dissertation proposes a novel Multi Objective Reinforcement Learning (MORL) algorithm that combines the universal function approximation capabilities of fuzzy systems and probabilistic theory with a linear scalarisation approach. The proposed algorithm seeks to exploit the advantages of both fuzzy inference systems and Probabilistic theory in a multi objective scenario to capture the probabilistic uncertainty of real world stochastic environments. This will allow the agent to choose an action that best satisfies all objectives based upon a probabilistic distribution able to minimise negative outcomes or maximise positive reinforcement of future events.

In recent years a growing number of algorithms for multi objective reinforcement learning have been proposed suggesting that MORL is emerging as a distinct sub-discipline of reinforcement learning research. However the field of MORL is still in its infancy and therefore the literature remains fragmented with very few real world applications developed. The observation that most real-world applications require the simultaneous satisfaction of multiple objectives drove the growth in multi objective optimisation research during the 1990's (Coello et al. 2002). Multi objective optimization concerns minimising each objective so that a lower value is superior to a higher value, however in the context of RL, the superiority is reversed so that the task is to maximise the reward. MORL problems are different to RL problems in the fact that there are two or more objectives to be achieved, each with its own associated reward signal; therefore the reward is a vector rather than a scalar value. If all objectives are directly related or completely independent, they can be combined into a single objective and a policy found that can maximise all objectives. In contrast if the objectives are conflicting then any policy must either maximise only one objective, or represent a trade-off between conflicting objectives usually in the form of a pareto front. The extension of a single objective RL algorithm (GPFRL) into a multi objective algorithm (GPFMORL) introduces new possibilities for variations in the overall aim of the MORL algorithm which shall be described in the following sections

1.2 Background to the research

The recent advances in Artificial Intelligence (AI) and military funded technologies embedded into Micro Arial Vehicles (MAV's) have allowed a wide variety of Unmanned Arial Vehicle (UAV) applications to be researched and advanced such as search and rescue, crop monitoring and even an Amazon drone delivery service. Along with the increasing popularity of quadcopters (drones) there arises the opportunity to take advantage of their full capabilities by endowing them with AI so that they can learn how to react to situations themselves without being reliant upon a human operator or communication link. Furthermore, researchers typically rely on expensive aerial vehicles that are capable of lifting heavier advanced sensors which inherently reduce the operational run time of the UAV impairing the overall effectiveness of task completion when instead the use of computer vision and AI techniques could be more efficient especially for communication failure redundancies.

1.3 Research Goal and Contributions

It can be seen later that the GPFMORL framework is an extended version of the single objective GPFRL algorithm using linear scalarisation and Prospect Theory. The following contribution is derived from the work described in this thesis.

A Generalised Probabilistic Fuzzy Multi Objective Reinforcement Learning method intended for continuous states and actions and is able to learn input-output mappings through interactions with the environment. This method tackles 6 issues in the decision making processes.

- Uncertainty in the outcome of actions is handled by a probabilistic approach.
- Learning from interaction, where system model is not readily available is handled by using reinforcement learning.
- Uncertainty in the inputs, which is handled by using a fuzzy logic control method.
- Choosing the most appropriate action that satisfies multiple objectives using a linear scalarisation technique.
- Dealing with the presence of risk in the form of Prospect Theory (Risk Aversion)

The developed algorithm exhibits a comparatively fast learning speed and flexibility as it can be used for several different systems, where control or decision making under uncertainty is paramount. This concept has been tested through 4 different experiments and 2 additional research prototypes:

- The random walk multi objective deterministic simulations
- The Windy Grid World multi objective stochastic simulations
- Experiments performed using Prospect Theory for uncertainty under risk
- A real time experimental investigation of automatic landing for a quadcopter mobile robot using computer vision as the primary positioning sensor.

1. Working Prototype of the Automated Solar Powered Environmental Controller
2. Partially working prototype for the Robotic Dementia Medication Administration System

1.4 Justification for the research

What takes place when lerning occurs in still largely unknown. The presence of uncertainty and the concept of knowledge itself make it intrinsically difficult if not impossible to model and analyse the final behaviour of a learning system; therefore no system with unsupervised learning should be unsupervised. Instead, it is suggested that the research of methods combining the probability estimation of reinforcement learning, as the one proposed by Hinojosa et al (2011), with other methods that can incorporate the knowledge of human operators; especially for situations where the result of selecting actions or behaviours with a relatively “high” probability of success can have positive consequences. As an example, it can be suggested the use of prospect theory, which is a method of calculating decisions not only based on probabilities of success but also based on a risk evaluation.

1.5 Methodology

Quantitative research is ‘Explaining phenomena by collecting numerical data that are analysed using mathematically based methods (in particular statistics)’. For example, how many iterations (steps) does it take a MORL agent to converge to the optimal policy? While quantitative research is based on numerical data analysed statistically, qualitative research uses non-numerical data. Qualitative research encompasses a wide range of methods, such as interviews, case studies, ethnographic research and discourse analysis. Due to the nature of this autonomous project and the hazardous environment in which it acts, the need for human subjects is eradicated. The performance measure of the system can be acquired using statistical methods directly outputted from the system in terms of a numerical value. The performance may then be compared to other implementations of MORL using further statistical analysis in an attempt to demonstrate its superiority and short falls in contrast with other algorithms.

1.6 Outline of the thesis

- **Chapter 2** Reviews existing Multi Objective Reinforcement Learning literature to build a theoretical foundation upon which the research is based. The chapter starts with a brief introduction to multi objective reinforcement learning and then suggests some important real world applications where MORL may be applied to improve existing methods. A general overview of RL is then described and a taxonomy of RL frameworks is presented highlighting criteria from which the proposed algorithm was created. Several single policy multi objective algorithms are summarised, followed by multiple policy algorithms, most significant RL and Hybrid algorithms. Prospect theory is then briefly introduced finishing with a discussion and conclusions.

- **Chapter 3** Explains in more detail the principal components of the GPFMORL architecture. The mathematical formulisation of a sequential decision making process is explained. I then analyse the importance of uncertainty handling; review the concept of fuzzy inference systems, and then introduce the application of probabilistic theory to fuzzy logic. Finally conclusions are made and the rationale behind extending the GPFRL single objective RL algorithm into an improved multi objective GPFMORL algorithm is justified.

- **Chapter 4** In this chapter, we consider a number of examples regarding our proposed reinforcement learning approach. In sub section 4.2, we describe a multi objective linear scalarisation technique which extends the single objective GPFRL algorithm into a multi objective one (GPFMORL). Prospect theory is then explained in section 4.3 in more detail with integration into our proposed algorithm. Section 4.4 details three unique case study research prototypes developed to demonstrate the feasibility of MORL decision making for crucial applications such as AI UAV's, ASPEC and RDMAS. Subsection 4.4.1 details the various implementations necessary to construct the Unmanned Aerial Vehicle Visual Navigation Framework. Subsection 4.4.2 details various prototypes for the Automated Solar Powered Environmental Controller. Subsection 4.4.3 details the design for a Robotic Dementia Medication Administration System. Section 4.5 explains some Empirical Evaluation methods for MORL finishing with a discussion and conclusion.

- **Chapter 5** This chapter explains several experiments using the proposed GPFMORL algorithm, and compare it to results of two other more conventional MORL algorithms MOQ and MOSARSA. Four experiments are presented in this thesis: Multi Objective Deterministic Environment, Multi Objective Stochastic Windy Hill World Environment, UAV Automatic

Landing Practical Experiment and finally the Controlled Environment Agriculture of Hydroponic Tomatoe Plants using ASPEC.

- **Chapter 6** – Recalls the contributions of this work; it then states some observations and proposes some possible extensions and directions for future research. The chapter ends with some final conclusions derived from this research.

Additionally three appendices complement the chapters above as by demonstating the implementation of MORL as follows: first, appendix A contains matlab programme code used for the random walk simulated experiments. Second, Appendix B contains C++ code used for the practical UAV experiment. Finally Appendix C includes design and development evidence of an alternate case study embedded system prototype “Automated Solar Programmable Environmental Controller (ASPEC)”.

1.7 Delimitations of scope and key assumptions

One of the test labs is approximately 6 m high therefore preliminary experiments reveal that this is sufficient to perceive the environments plan view perspective of around 3m^2. Should larger environments need to be tested, there are other rooms available with a maximum height of approximately 12m high, therefore at most a plan view perspective of a 6m^2 environment can be used for testing purposes. This should satisfy my testing conditions of non-finite environments in an attempt to validate scaled up versions of MORL. It is assumed that the control PC shall be no more than 50m from the AR-Drone at any point in order to ensure reliable Wi-Fi signal strength.

1.8 Conclusion

This chapter laid the foundations for the report. It introduced the research problem and research questions and hypotheses. Subsequently the research was justified, definitions were presented, the methodology was described and justified, the thesis was outlined, and the limitations were given. On these foundations, the report can proceed with a detailed description of the research.

Multi Objective Reinforcement Learning

2.1 Introduction

Automonous robot operation requires the subjects to know its position and movement within the environment. Since no assumptions can be made about the environment, the robot must learn from its environment. The learning task consists of finding a mapping from environmental conditions to behaviours into the most effective actions (policy). One method that is particularly good at learning and improving policies is RL, which is considered to be a significant sub-branch of Machine Learning (ML) most suited to solving sequential decision making problems. Single Objective Reinforcement Learning (SORL) has proven to be very powerful at allowing agents to learn in unknown environments by maximising a value representing a single long term objective. However within the majority of real world decision making problems there usually exists multiple long term objectives. Only recently in the last few years has there been a significant increase in MORL research. However, researchers have yet to develop of a MORL algorithm capable of dealing with risk and uncertainty which is crucial in most real world applications as suggested by the ASPEC and RDMAS research prototypes.

2.2 Case Study Research Prototype Review

2.2.1 Controlled Environment Agriculture

An important real world example of a multi objective problem in the presence of uncertainty is the optimisation of hydroponic crop growth by means of Controlled Environment Agriculture (CEA). Few systems have attempted to automate hydroponic crop growth however no systems take into account uncertain environmental factors which is imperative to predicting the optimal conditions for optimised crop growth. The paper by Saaid et al (2013) proposes a microcontroller system for a hydroponic technique known as deep water culture. Whilst the described system compensates for PH variations with good results, the electrical conductivity of the nutrient solution is neglected, consequently only the PH of the nutrient solution is optimised which is not sufficient for crop growth optimisation. The instrumental system developed by Domingues et al (2012) proved successful at monitoring and fixing the pH and EC of the nutrient solution for hydroponic lettuce. However the quality of crop results obtained were done so via two crops grown in parallel, one in conventional soil and the other using an automated hydroponic system. This does not fairly test the advantages of the automated system; instead it compares automated hydroponic crop growth with manual conventional soil methods which are unsurprisingly not as efficient. Instead a fairer test would have been to test two parallel hydroponic crop growths, one real time-automatic and the other manually controlled hydroponics truly demonstrating the advantages of automated hydroponics in a non-bias testing environment. Whilst the work by Velázquez et al (2013) proposes sound electrical schematic evidence of accurate PH and EC monitoring systems, it neglects other environmental factors such as Temperature which is influential on nutrient absorption. A system with closed loop feedback from nutrient solution sensors in combination with monitoring of uncertain conditions, such as sunlight and temperature, may result in much more effective hydroponic optimisation system, such as the one proposed later in this thesis.

2.2.2 Robotic Medication Administration Systems

As a result of medication errors both in hospitals and at home, adverse drug events (ADE's) cost the NHS around £200 - £400 million per year. In order to reduce these medication errors, Morriss, Frank, et al (2009), proposed barcode technology that verifies medication administration. However whilst this system was effective at detecting medication errors, it was not capable of acting automatically in order to correct these errors which may in fact present the opportunity for more errors to propagate further down the medication administration cycle. Other commercial products include Aesynt's ROBOT-Rx which is the market-leading automated medication dispensing solution. The system increases efficiency and accuracy in a Central Pharmacy, but this is not a personal mobile solution and is the size of a large room which would not be feasible for self-care patients in their own homes.

2.3 Reinforcement Learning

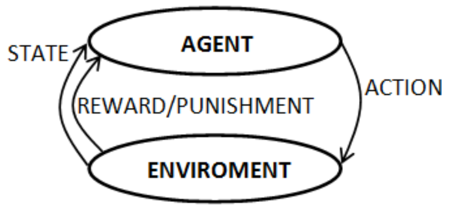


Figure 2.1: Basic Components of a RL

RL uses the notion of positive rewards and negative punishments that are experienced by the agent themselves. In other learning methods such as supervised learning, the best action possible will be made known to the agent after acting in specific way. In practical problems, the best action is sometimes impossible to know, but a certain value of the resulting state could be defined. The task of the agent is to learn from this indirect, delayed reward to choose sequences of actions that yield the greatest cumulative reward. The reward obtained when the agent attempts different actions, is used in reinforcement learning to update either the value function or the policy. The value function determines the potential value of each state where as the policy maps possible perceived states to the best known set of actions. The goal or objectives to be achieved are defined by the reward function without actually specifying how the objectives shall be accomplished.

The primary aim of RL is to maximise the expected total reward over time, also known as return. Interaction within the environment allows the agent to learn the optimal actions to take for each state by selecting actions which result in the largest expected return. In order to update the agents knowledge, a method that involves some kind of temporal difference (TD) may be used **(**Sutton & Barto, 1998**)** that updates the value function. Temporal-difference (TD) methods were formalised and studied by Sutton (1988) as a solution to the problem of making multi-step predictions of future events based on past experience. Before Sutton’s formalisation, well-understood techniques for learning predictions were trained using differences between predictions and the actual future outcomes.TD techniques are used to update the values or value function parameters based on the direct reward and the difference between the current and last potential state value, weighted with a learning parameter (Veen, M. van der. 2011). The update rule is therefore based on the difference between the value of the last state visited and the value of the current state, or a weighted combination of the old expected (long-term) return value and the current one based on current reward and the value of the next state (Veen, M. van der. 2011)**.**

Alternatively Evolutionary Algorithms (EA) can be used to directly update the policies themselves whereby whole policies, instead of values, are evaluated and ‘evolve’ to better policies (XIE Li-juan, 2009). To use the knowledge learned from experience, as well as gather new knowledge, a trade-off between exploration and exploitation must be established. In practice, often some percentage of the actions is chosen random and the remaining actions are determined by the value function or policy learned so far (Veen, M. van der. 2011)**.** Markov Decision Process (MDP) models are the mathematical foundation for RL in a single agent environment (Yang, E., & Gu, D. 2004). Commonly a MDP is characterized as the mathematical formulation for sequential decision-making problems defined as a 4-tuple {S, A, R, T,*y*} (Sutton&Barto, 1998) of which sequences of MDP tupples describe the agents experience in the learning process.

2.4 Taxonomy of RL Frameworks

Being a very flexible learning technique, there are several methods and algorithms that have been developed. Each of these can be classified under one or more categories according to different criteria. There are mainly three basic criteria to classify reinforcement learning methods:

- By the presence of a system model.
- By the way they learn and select a correct policy.
- By its structure.

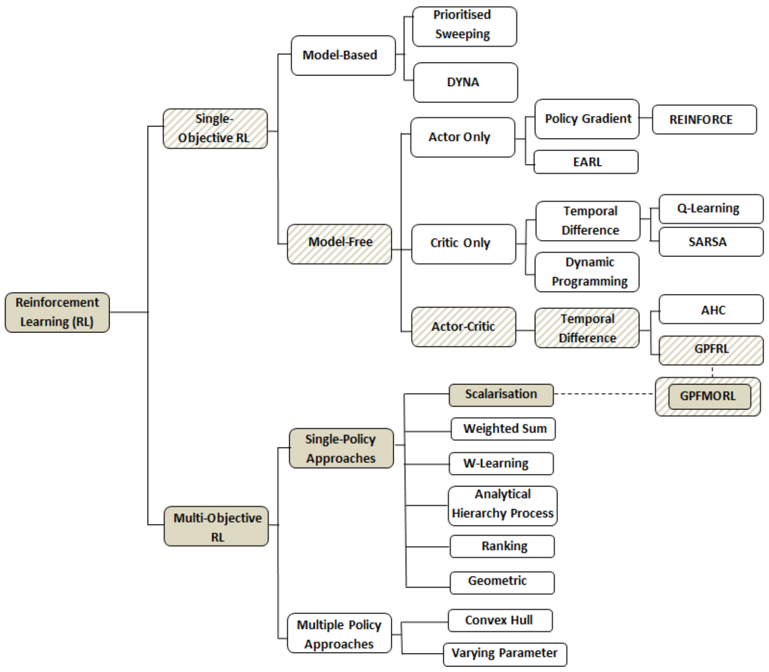


Figure 2.2: Reinforcement Learning Taxonomy

Figure 2.2 presents a flow chart illustrating a reinforcement learning taxonomy, along with some well-known algorithm examples. Each of the reinforcement leaning methods or algorithms are classified into one or more categories according to different criteria. The hashed blocks show the structure of a recently developed GPFRL algorithm (Hinojosa, W. 2008), where the shaded blocks show the model that forms the principal focus of this thesis which primarily concerns extending the GPRFL algorithm to the multi-objective domain for complex dynamic unknown stochastic environments.

A. Model-Based & Model-Free Single Objective

Model-based or indirect methods act in two phases: first, they learn the transition probability and reward functions and second, they make use of those transition probabilities in order to compute the Q function by means of, for example, the Bellman equations (Bellman, 1957). Some of the best well known algorithms in this category are Dyna (Roy et al., 2005) and prioritised sweeping (Demspster, 1969, Moore and Atkeson, 1993). Model free or direct methods learn the policy by directly approximating the Q-function with updates from direct experience. These methods are sometimes referred to as the Q-learning family of algorithms (Sutton, 1988), (Watkins, 1989). Some of the best well known algorithms in this category are adaptive heuristic critic (AHC) (Barto et al., 1983), Q-learning (Watkins, 1989) and SARSA (Rummery and Niranjan, 1994). Model free methods can be further classified in three sub groups depending on whether the algorithm focuses on learning the policy or the value function, critic only, actor only, and actor-critic (Barto et al., 1983).

B. Actor, Critic, Actor-Critic

Critic only methods are called value function-based methods and they attempt to find the optimal value function from which an optimal policy is derived. Some of the most important reinforcement learning algorithms are critic-only methods, such as Dynamic programming (Bellman, 1957) and Temporal Difference (Sutton and Barto, 1987, Sutton, 1988). In contrast with the critic only method, actor only methods learn the policy, which is a function that depends only on the current state and therefore a value function is never defined. In actor only methods, the learning agent uses an explicit representation of its behaviour with the goal of improving it by searching the space of possible policies P. Therefore, an actor only method will be feasible, if its search space is restricted to a subset of P. Some important actor only algorithms include: Associative Reinforcement Learning Algorithms (Barto and Anandan, 1985). Policy gradient algorithms (Cheeseman, 1985). REINFORCE (Williams, 1992). EARL, evolutionary algorithm RL (Moriarty et al., 1999). The best well known example of the actor-critic algorithm is the Adaptive Heuristic Critic Algorithm (AHC) (Barto et al,1983) where this novel concept was introduced for the very first time. Actor-critic (AC) methods are TD methods that have a separate memory structures to explicitly represent the policy independent of the value function. The policy structure is known as the actor, because it is used to select actions, and the estimated value function is known as the critic, because it criticizes the actions made by the actor. When combined into an AC structure, the learning is always on-policy: the critic must learn about and critique whatever policy is currently being followed by the actor. The critique takes the form of a TD error (a scalar signal) and represents the sole output of the critic which drives all learning in both actor and critic. It would seem that an actor-critic system with a lookup table is guaranteed to converge to optimality no matter what. Surprisingly, that is not the case. Although it always converges for y>0.5 (Williams & Baird 1993), it does not always converge for larger y as shown by Baird (1999).

C. ON-Policy / OFF-Policy

In both model free and model based methods there are two basic strategies to update the policies value function which are known as on-policy and off-policy learning. Off-policy methods are able to update the estimated value functions by using hypothetical actions, those that may never actually be tried. The behaviour policy of the agent is usually “soft” and therefore includes some element of exploration. An advantage of off-policy algorithms is that they can separate exploration from control, whilst on-policy algorithms cannot. One of the most well-known on-policy learning algorithms is called Q-Learning (Watkins, 1989) and is reviewed in more detail in section 4A.

D. Single-Policy Multi-Objective

MORL approaches may be divided into two categories based on the number of policies to be learned: single-policy approaches and multiple-policy approaches. Single-policy approaches aim to obtain the best single policy that satisfies the preferences among the multiple objectives, as specified by a user or derived from the problem domain (Vamplew, P et al. 2011). The major difference between the single-policy approaches is the way in which these preferences are expressed.

Weighted Sum Approach

The weighted sum approach is a naturally extended version of Greatest Mass approaches such as (GM-Q) and (GM-Sarsa). GM-Q suffers from the problem of positive bias where as GM-Sarsa does not due to being an on-policy method. Using

GM-Sarsa's approach the updates were based upon actual actions taken rather than the best possible action, the algorithm discovers Q-Values that are closer to the true expected return. This weighted sum approach has the advantage of allowing the user to have some control over the nature of the solution by placing more or less emphasis on each of the several objectives. Assuming there are two objectives to be completed and 5 possible candidate actions as illustrated in (Fig 2.3), you notice that actions a2, a3 and a4 lie within the concave regions of the pareto front and will therefore never be selected, instead either a1 or a5 will be selected according to their pre-defined weights. This disadvantage can be overcome by using less frequent non linear functions as demonstrated in Tesauro, G. et al (2007)

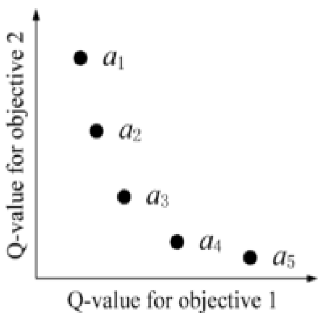


Figure 2.3: Weighted Sum Concave Drawback (*Source*: Liu et al, 2013)

<u>W-Learning Approach</u>

In the paper by Humphrys, M (1996), Humphrys demonstrates how 'W' values can be generated using Top-Q algorithm which generates the action according to the objective with the highest Q-value in the current state. The advantage of this is that the Optimal action is chosen for at least one objective. However the disadvantage is that the objective with the highest Q-Value may have no preference over what action is chosen, while another objective stands to lose a great deal if its action is not selected. A much better approach is to learn the W values based upon the update rule for a process known as W learning (Liu, C. et al 2013). The reverse of this process is called Negotiated W-Learning which results from Liu, C. et al's (2013) statement that there is no need to learn the W values, instead they can be computed directly from the Q-Values.

A common approach in complex reinforcement learning tasks is to divide the problem into functional parts, or behaviours, and then to assign a sub-agent to solve each task. The action selection problem then becomes to negotiate between sub-agents with conflicting desires. W-learning is a method whereby agents build up W-values in each state that indicate how important that state is for that agent. These values are then used as basis for selecting agents.

Analytic Hierarchy Process (AHP) Approach

In most cases, knowledge about the optimisation problem does not exists explicitly therefore we often define the objectives in a qualitative manor such as “It is more important that the UAV does not fly into a wall rather than landing precisely on the helipad.” However this presents a problem for stochastic/deterministic systems as there is no precise mathematical description to quantify this information. Using fuzzy subsets and inference rules, the Importance factor Ii and the value of improvement Di(ap,aq)= Qi(s, ap) − Qi(s, aq) are used as inputs to construct the action selection mechanism (Humphrys, M. 1996). Another example using the AHP algorithm is the Greenhouse Parameter Control Strategy demonstrated in Qian and Wang (2013) where the target for the RL agent is to produce as high yield of crops as possible by taking into account several environmental factors such as light intensity, Co2 level, humidity, and temperature. Various equipment such as heaters, ventilation fans, irrigation equipment and LED lights are controlled by the AHP algorithm to provide a more scientific and reasonable sequence in selecting control measures. Providing the ideal growth values for any particular crop are known, this method may be used to automatically collect and control crop growth parameters to be used as an automatic control algorithm for maximising any crops production yield.

Ranking Approach

Mitten (1964) and Sobel (1975) expressed an objective function TQ(s,a) in terms of partial policies which is also known as the sequential or threshold approach. In this approach one objective is maximized whilst satisfying the constraints on other objectives (such as maximizing factory production while maintaining a required safety level). In the paper by Zheng et al (2012), the two metrics of transmission delay and packet loss rate are considered as an optimisation problem of two choices.

- Minimize transmission delay under desired constraint of packet loss rate, which would be suitable for the scenario of a best effort application.
- Minimize packet loss rate under desired constraint of transmission delay, which would be proper for the scenario of a real-time application.

Two Q-Tables are used in this approach to address the challenges of randomness, uncertainty, and multiple metrics in the field of cognitive radio networks. However

Since they only consider the packet loss caused by the link condition, the comparison between MORL based routing and shortest path routing is actually unfair.

Geometric Approach

This algorithm proposed by Mannor and Shimkin (2004) focuses on the long term average reward vector in a dynamic, unknown environment. Interestingly this algorithm can also deal with environments that may be altered by the actions of other agents providing pre-existing knowledge of the problem domain is provided to define the target set. Importantly a disadvantage of this system is that although the changes to the environments state can be observed, they cannot be predicted before hand. The geometric algorithm learns to approach a prescribed target set in multidimensional objective space shown in fig 2.4b below.

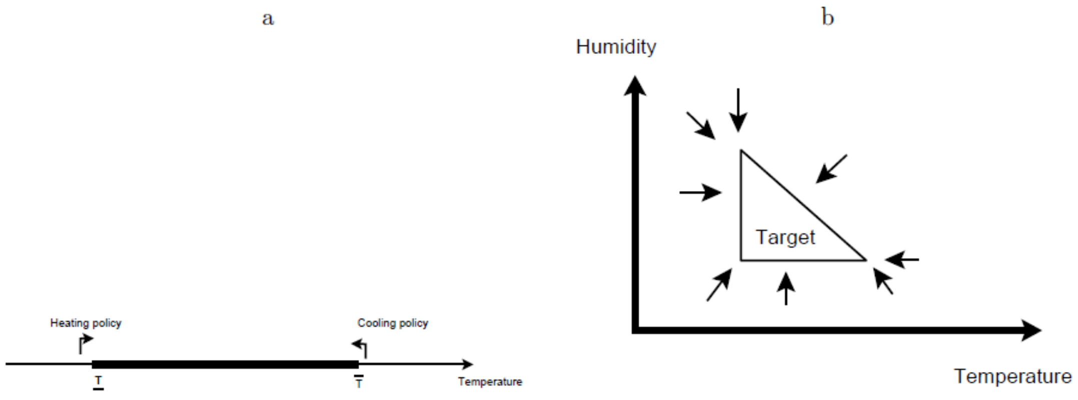


Figure 2.4a & 2.4b : Reward Vectors (*Source*: Mannor and Shimkin, 2001)

The above learning algorithm illustrated in Figure 2.4a only needs to determine which element to use at each of the two extreme regions. EG: If the temperature is higher than T, the cooling fan policy shall be executed, where as if the temperature is lower than T then the heater will be activated. Now consider a more complex multi-objective version of this example where the objective is still to maintain a specific temperature although now other parameters such as “frequency of switching between policies, average energy consumption, average humidity are to be added as extra objectives. The target set shown in Figure 2.4b demonstrates how the steering policy for the controller is now much more complex than figure 2.4a’s left or right directions, now there is a continuum of possible directions, each associated with a possible different steering policy.

E. Multiple-Policy Multi-Objective

Multiple- policy approaches aim to find a set of policies that approximate the Pareto front. The fundamental difference between multiple policy approaches is the manner in which the Pareto front is approximated (Chunming et al 2003).

Convex Hull Approach

Until now, only single policy approaches to multi objective reinforcement learning have been discussed however there have been efforts to develop algorithms that are multiple policy approaches by nature. A multiple policy approach known as the convex hull approach is described in Li-juan et al (2009) which can learn optimal policies for all linear preference assignments over the objective space at once. Conventional RL algorithms repeatedly back up maximal expected rewards whereas this type of RL backs up the set of expected rewards that are maximal for some set of linear preferences. Barret and Narayanan (2008) proved that their solution gives the optimal policy for any linear preference function and the solution is reduced to the standard value iteration algorithm for specific weight vector. Crucially this convex hull technique can be used in combination with other RL algorithms, because multiple policies are learned at once, these RL algorithms must be off-policy such as Q-Learning or Dynamic Programming Techniques.

Varying Parameter Approach

Given a set of parameters, the optimal policy with respect to those parameters could be learned. As demonstrated in Castelletti (2002), scalarised Q-learning can be applied in a multiple-policy context by performing repeated runs of a single-policy algorithm using different parameter values. Similarly, a multiple-policy approach can be implemented by performing multiple runs with different parameters, objective threshold values, and orderings in any single-policy algorithm. Shelton (2001) applied policy gradient methods and the idea of varying parameters to the MORL domain. Gradients in the parameter space of the policy were computed individually for each objective and then combined to form a weighted gradient. By varying the weighting of the objective gradients, a range of policies can be discovered.

2.5 Historical Progress

A. Most significant RL frameworks

Q-Learning aims to learn and estimate Q values in order to construct optimal control strategies (policy) from delayed rewards, even when the agent has no prior knowledge of the effects of its own actions on the environment. The RL agent uses its past experience to improve its estimate by blending new information into its prior experience. More than one optimal policy can be generated however the Q values remain unique (Yang and Gu, 2004). Although Q-learning is a very effective framework for individual robot learning (Gherega et al 2012), alone, it does not appear to lend itself to Multi-robot systems where other robots may alter the state of the environment, rather than a single agent learning from the new state of the environment changed only by actions performed by themselves. In the paper by Mei et al (2007) the blackboard architecture is adopted to realise cooperative Q learning in multi-robot system. Each individual robot first queries the blackboard according to its current state, then executes allocated action and sends the obtained reward back to the blackboard. The blackboard carries out learning using received state- action pairs and rewards. Q-learning is applied as a coordination mechanism for multiple robot teams.. A common hypothesis for MRS is that the more robots participating in completing this task, the better the overall performance. However Mataric's (1994) analysis of interference suggests that there is a limit in multi-agent systems to the level of improvement gained by adding additional robots. What was demonstrated was that while the size of the team causes an exponential drop-off in the number of iterations required to complete the map, the change in the reward function only makes local changes about this curve. More important to the performance of the overall team than the reward values are the learning rate and exploration properties. The algorithm in Castelletti et al (2012) proposed an extension of fitted Q-Iteration (FQI) enables to learn the control policies for all the linear combinations of preferences that are assigned to the objectives in a single training process. In Liu and Wu (2010), visited states are assigned with different immediate rewards by comparing the objective vector of current state with those of the Pareto optimal solutions found previously. These Pareto optimal solutions are stored in an elite list, which keeps track of the non-dominated solutions found so far and is used to construct the Pareto front at the end of the optimization process. Gherega et al (2012) use a Q-Learning Approach to

Decision Problems in Image Processing by using reinforcement learning in image processing and computer vision extending as a way of replacing human assistance with intelligent agents.

B. Most Significant Hybrid Algorithms

In recent years, many researchers have attempted to implement new RL schemes either by developing new hybrid architectures or designing new learning methods from scratch. Many new hybrid architectures are being attempted that combine reinforcement learning with other methods like fuzzy logic and neural networks such as Castelletti et al (2012) and Liu and Wu (2010). The Generalised Probabilistic Fuzzy Reinforcement Learning Algorithm (GPFRL) proposed by Hinojosa (2010) demonstrates how the combination of RL, fuzzy logic, and probabilistic theory have proven to be an excellent way of managing information in the presence of different kinds of uncertainty. When combined with fuzzy logic, the RL task of fine tuning controllers can be achieved by one of two ways, either by performing structure identification as studied by Lin and Xu (2006), Lin and Lee (1993), Wang et al (2007), or as parameter identification like in Berenji and Khedkar (1992), Lin and Lee (1993), Wang et al (2007). A similar study to GPFRL is by Strens (2000) who uses the idea that uncertainty in the underlying MDP model can be encoded with a probability distribution. An early approach known as Adaptive Heuristic Critic (Barto et al, 1983) where the system state is described based upon the input variables in what is called a “box system.” The agent then uses these discrete input variables as a way of deciding the most appropriate action to take and then represents the system as a numerical value that corresponds to the input state. A better approach considers a continuous system characterisation, like in Anderson (1989), whose algorithm allows the addition of continuous inputs by using a two-layer neural network. The work in Anderson (1989) was an improvement over Barto et al (1983) in the fact that continuous inputs were used however the learning speed was still relatively poor. An improvement to the learning speed was suggested by Berenji and Khedkar (1992) whose work of the (GARIC) algorithm introduces generalised approximate reasoning and structure learning into its architecture therefore further reducing the learning time. Another interesting hybrid approach that uses approximate reasoning theory and neural networks was proposed by Lee (1991) however, his approach has the disadvantage of not being capable of being used as a stand-alone controller without the learning structure. Lin and Lee (1993) developed two approaches: A reinforcement neural fuzzy control network (RNFCN) (Lin, 1995) and a reinforcement neural network based fuzzy logic control system (RNN-FLCS). Both algorithms were given the ability to learn using structure and parameter learning to achieve good overall performance compared with other hybrid methods. A generalised RL fuzzy controller was proposed by Zarandi et al (2008) which is capable of handling vagueness of inputs but overlooks the handling of ambiguity which is crucial when dealing with uncertainty. Furthermore the structure of the algorithm became very complex by the use of two independent fuzzy inference systems. Several other fuzzy RL algorithms, primarily model free, have been implemented based upon Q-learning (Lin and Lin, 1996), (Almeida and Kaymak, 2009) or AC techniques such as those by Jouffe (1998) and Lin (2003). Lin and Lin (1996) developed RL strategy based on fuzzy-adaptive-learning control network (FALCON- RL) method. Fuzzy-AC-learning (FACL) method (Jouffe, 1998), Lin's RL-adaptive fuzzy-controller (RLAFC) method (Lin, 2003), and Wang's fuzzy AC RL network (FACRLN) method (Wang et al, 2007) However, most of these algorithms fail to provide a way to handle real-world uncertainty which is a greater issue in such a large complex stochastic environments with continuous inputs.

2.6 Prospect Theory

Psychologists and economists have recently paid closer attention to modelling how decisions are made under conditions of risk, where options are characterised by a known probability distribution over possible outcomes. Decision making under risk was first introduced by a mathematician named Daniel Bernoulli who published a paper in 1738 entitled "Exposition of a new theory on the measurement of risk, as cited in Hannson (1994). His paper was based upon the economic theory of risk aversion, The St Petersburg paradox, risk premiums, and utility. Bernoulli also proposed a function called the "Utility Function" to explain how people behave when making choices. Bernoulli took the assumption that people try to maximise their utility rather than their expected value and that people are typically risk averse. This model was the beginning of utility theory which combines descriptive and normative elements at the same time.

Based upon Bernoulli's principle, later in 1944, John von Neumann and Oskar Morgenstern proposed a reinterpreted formal model called expected utility theory (EUT). Since then, it has been used as a model of rational choice under risk based upon the assumption that people behave as rational agents and that they adhere to EUT in decision making behaviour under risk (Andriottyi, 2009). However EUT cannot capture the human decision making attitudes towards risk i.e.: risk aversion and risk seeking, because it does not model the domain of human intuition, thoughts or preferences that come to mind quickly without much reflection (Coleman et al, 2007). This drawback is also confirmed in Allais's and Ellsberg's paradoxes (Zhang et al 2007). A group of psychologists called Kahneman & Tversky (1979) addressed this limitation and developed an alternative theory of choice based upon Prospect Theory (PT) that can accurately describe how people actually make their decisions. They claimed that PT captures common human decision making attitudes towards risk that cannot be captured by EUT ie: risk aversion and risk seeking. While risk aversion is generally assumed in any economic analysis under uncertainty, risk seeking is generally not. PT goes towards explain why people often make decisions that seem irrational. If people always made decisions on a utilitarian bases there would be little need for insurance and few, if any, people would gamble. Moreover, if utility were the basis for all decision making there would be little need for marketing. The list of the topics that PT could apply to is long and still growing, some of these topics are: Investment banking (Willman et al. 2002), commercial banking (e.g. Godlewski, 2004), finance (Han& Hsu, 2004), analyst behaviour (e.g. Ding et al, 2004), hedge funds (e.g. Siegmann & Lucas, 2002) and political science (Mercer, 2005). Andriotti (2009) proposed a new Q-learning algorithm that reproduced human non-rational behavior by using PT as its basis instead of the EUT. Andriotti's new model has been applied in urban traffic modelling.

2.7 Discussion

Despite the recent developments in Reinforcement Learning, the challenge remains to scale up solutions to larger and more complex problems. "Recently, RL has been applied in many fields however MORL algorithm research is a relatively new field of study, therefore inherently there are few real-world applications developed so far (Liu et al, 2013). Traditionally RL works particularly well in finite discretised environments such as the deep see treasure benchmark (Vamplew et al, 2008) but not so well in larger continuous environments. Additionally due to the multi objective nature of many real world practical optimisation and control problems, there is a need for multi objective reinforcement learning algorithms to be further developed to accommodate such applications. The difference between single-agent and multi-agent system exists in the environments. In multi-agent systems other adapting agents make the environment no longer stationary, violating the Markov property that traditional single agent behaviour learning relies upon (Yanjg and Gu, 2004).

Many alternate approaches to solving MORL are actively being researched to work around this Markov violation and some algorithms are reviewed below. However, it appears that, should the environment remain stationary, this violation can be avoided. Forcing the robots to move in a synchronised manor could allow Multi Robot Systems (MRS) to utilise well understood Markovian principles, which have proven to work well in single robot systems. Providing all robots are synchronised during the learning process the only changes to the environment are the ones that we are actually trying to understand. This novel methodology could effectively scale up the MORL problem to be able to deal with more complex dynamic environments without other robots altering the state of the environment and causing misinterpretations of the agents effects of its own actions upon the environment. “Unlike single-objective optimisation that returns a best solution to the problem, multi-objective problems have usually no unique, perfect solution, but a set of equally efficient, or non-inferior, alternative solutions, known as the Pareto optimal set, which represents the possible trade-off among conflicting objectives” (Liao, 2010).

While conventional single objective algorithms such as the Q-Learning and SARSA may vary in terms of the internal mechanisms, they all share the same aim of maximising the scalar reward received, which is achieved by identifying a single optimal policy. For multi objective tasks there is no longer a single optimal policy, as many policies may in fact be Pareto optimal. Therefore variations can exist between MORL algorithms in terms of the number and nature of policies that they aim to discover. Within MORL research the algorithms developed belong to one of two classes based upon the number of policies to be learned. Single-policy algorithms aim to learn a single policy that best satisfies a set of preferences between objectives as specified by the user or derived from the problem domain. Alternatively multiple policy approaches aim to find a set of policies which approximate the Pareto front. As with any MORL algorithm, the aim is to intrinsically identify a single or multiple policies that produces suitable compromises between the multiple objectives. A good compromise can be expressed in terms of Pareto dominance (Pareto 1896) which allows a comparison of a pair of solutions to a multi objective problem. The main disadvantage of generating multiple policies rather than a single policy is the increased computation cost and the increased time spent interacting with the environment. The single policy approach is therefore more suited to on-line learning tasks within a real environment such as the UAV visual navigation task described in section 8. Using a UAV to learn from its own interactions is constrained by the fact that it has limited power where the additional cost of searching for multiple policies may be impractical due to the maximum learning flight time of approximately 15 minutes. The earliest example of a single policy algorithm was provided by Gabor et al (1998) where a similar approach is designed for problems where constraints apply to some of the objectives. For example in our experimentation the UAV must carry out a navigation task of maintaining a relative position above the helipad in the presence of stochastic uncertainty whilst maintaining a battery level above zero.

2.8 Conclusion

In order to identify the structure of the field, a representative set of approaches have been reviewed that provide insights into some of the latest state of the art algorithms used actively by researchers within the past decade.

In this review the background and basic architecture of RL and its methods were introduced first; then several representative MORL approaches were discussed. It has been shown that MORL is particularly successful in the following areas:

1) Improving the performance of the traditional single-objective RL.
2) Generating highly diverse multiple Pareto-optimal models for constructing ensembles.
3) Achieving a desired trade-off between accuracy and interpretability of neural networks or fuzzy systems.

Reinforcement learning provides a set of very useful methods for learning in cases where a model of the environment or system is not available. The most remarkable advantage in the use of reinforcement learning methods is in its use on systems where complete feedback information is not available, as RL only requires information about

the success or failure of an action. It was seen that the SARSA and Q-learning algorithms are very similar, while SARSA updates for the policy it is actually executing, Q-learning updates for a greedy policy. It can be said that Q-learning will learn the "true" optimal policy, but SARSA will learn about what its actually doing.
Hinojosa (2010) proposed an approach that is robust under probabilistic uncertainty while also being capable of dealing with continuous states and actions as is prominent in complex real world scenarios particularly when using computer vision as the primary observation method for sensing changes in the environment. The two main fields of application of reinforcement learning are decision making and control problems. So far reinforcement learning has been used successfully in several different applications under both fields, yet some more practical and real-world applications need to be tested particularly for scenarios involving multiple conflicting objectives. For single-policy MORL approaches, the weighted-sum and W-learning approaches are simple to implement, but they cannot express exactly the preferences of the designer. The AHP, ranking, and geometric approaches may express the preferences more exactly, but they need more prior knowledge of the problem domain. For multiple-policy approaches, the convex hull algorithm can learn optimal policies for all linear preference assignments over the objective space at once. The varying parameter approach can be easily implemented by performing multiple runs with different parameters, objective threshold values, and orderings in any single-policy algorithm. MORL approaches have been improved recently in three important directions:

- Enhancing their qualities
- Adapting dynamic preferences
- Constructing evaluation systems

Except for these three important improvements, the main challenges and open problems still remain in MORL which include Value Function Approximation (VFA), feature representation, convergence analysis of algorithms, and its applications to multi-agent and MRS systems for real-world problems. It is to the best of my knowledge that no other algorithm exists that is capable of dealing with multiple objectives in the presence of risk and uncertainty by using generalised probabilistic fuzzy logic in the area of decision making for UAV's, Healthcare & Agriculture.

Chapter 3

GPFMORL Methodology

3.1 Introduction

This chapter explains the principal components of the GPFMORL architecture for solving sequential decision making problems in uncertain stochastic environments. The mathematical formulisation of a sequential decision making process is explained along with the agent, policy, environment and reward structure. We then analyse the importance of uncertainty handling; review the concept of fuzzy inference systems, and then introduce the reader to probabilistic theory and its application to fuzzy logic. The proposed GPFMORL algorithm extends an existing single objective GPFRL algorithm (Hinojosa, 2011) which itself merged 3 different paradigms in order to deal with their individual limitations. In order to improve the work by Hinojosa (2011) a further two methodologies were introduced namely linear scalarisation and prospect theory in order to ensure the convergence of a multi objective sequential decision making problem. MORL has proven to be a valuable learning method when a system requires a selection of control actions associated with each objective, whose consequences emerge over long periods for which input or output data are not available (Hinojosa, 2011). In particular UAV visual navigation for automatically landing a UAV on a helipad in uncertain stochastic conditions. Finally, conclusions are made and rationale behind extending the GPFRL single objective RL algorithm into an improved multi objective GPFMORL algorithms are justified.

3.2 Multi Objective Reinforcement Learning Framework

The following subsections describe the most important terms used in both single and multi objective reinforcement learning that shall be used through the entire thesis.

<u>Markov Decision Process</u>

MDP models are for RL in single agent environments (Yang and Gu, 2004). Commonly an MDP is characterized by the mathematical formulation of a sequential decision-making problem defined as a 4-tuple {S, A, R, T, y} (Sutton&Barto, 1998) of which sequences of MDP tupples describe the agents experience in the learning process.

- **S** = State Space – A finite discrete set of environment states (5x5 Grid = 25 states)
- **A** = Action Space – A finite discrete set of actions available to the agent (Left, Right, Forward, Backwards)
- **R** = Reward Function giving the expected immediate reward by the agent under each action in each state.
- **T** = Transition Function T: $S \times R \rightarrow \Pi(S)$ that provides a probability distribution matrix for each state and action
- **y** = discount factor which controls how much effect future rewards have on the decisions at each moment

Every MDP has a deterministic stationary optimal policy which consists of a stationary policy (a probability distribution over actions to be taken for each state) and a deterministic policy (one with probability 1 to some action in each state).

<u>The Agent</u>

A system such as a UAV moves and interacts with the environment through a collection of both straight and diagonal actions such as Forward, Backward, Left Right and combinations of these such as FL, FR, BL, BR. The agent observes the state of the environment and uses a learned policy in order to select an appropriate action or controller output. The agent must also have a goal or goals relating to the state of the environment. For example, an agent could be a quadcopter UAV that is attempting to stabilise itself above a helipad whilst perceiving the state of the environment in which it is navigating.

<u>The Policy</u>

The policy (π) is a decision rule that dictates what action to take at every possible state. The goal of the learning agent is to find a policy that maximizes the total expected reward (or the total discounted rewards) that it will receive over time, known as “the expected return”. For example: for the present thesis, given any particular state, the use of the policy will determine a probability distribution for the agent’s future rewards, or as we will call it, a probability of success.

The Environment

The environment is the external and immediate world in which the agent acts in. The environment interacts with the agent, dynamically affecting its perceived state. The environment can be represented in many ways. For example: the environment for a robot could be composed of walls to be avoided, or the friction of its wheels on the floor, or aerodynamic wind forces from external sources or created itself by a UAV's propeller blades. Some well known multi and single objective RL benchmark environments are Buridians Ass Grid World (a foraging task) and the MO Puddleworld problem which composes of a goal and puddles to avoid (Boyan and Moore, 1995). The environment that is currently perceived by the UAV in this thesis is a downwards facing view of the area surrounding the helipad which has been cellular decomposed into a 5x5 grid. Each of the 25 rectangles represents a unique state in which the UAV may perform one of 8 distinct actions as previously described.

Agent environments are classified based on different properties that can affect the complexity of the agents decision-making process according to Russell and Norvig (1994).

- Accessible or inaccessible: An accessible environment is one in which the agent can obtain complete, timely and accurate information about the state of the environment.
- Deterministic or non-deterministic: In a non-deterministic environment, the state that will result from an action is not guaranteed even when the system is in a similar state. This uncertainty presents a greater challenge to the agent designer than deterministic systems.
- Static or dynamic: Static environments remain unchanged except for the results produced by the actions of the agent. Whilst dynamic environments are created when other processes operate in them, thereby changing the environment outside the control of the agent.
- Discrete or continuous: An environment is discrete if there are a fixed, finite number of actions and percepts in it.

The proposed three dimensional environment is a challenging one for any robot to operate within due to the highlighted factors mentioned above. It is inaccessible at times when the agent can not obtain complete visual state information in the unlikely event that the UAV flies outside the boundaries of the localisation grid. Any of the 8 possible actions that may be taken could actually lead to a non-deterministic state transition ie: if there is a stochastic cross wind or other aerodynamic affects present then the expected state after taking an action is likely to change and not be predictable prior to moving.

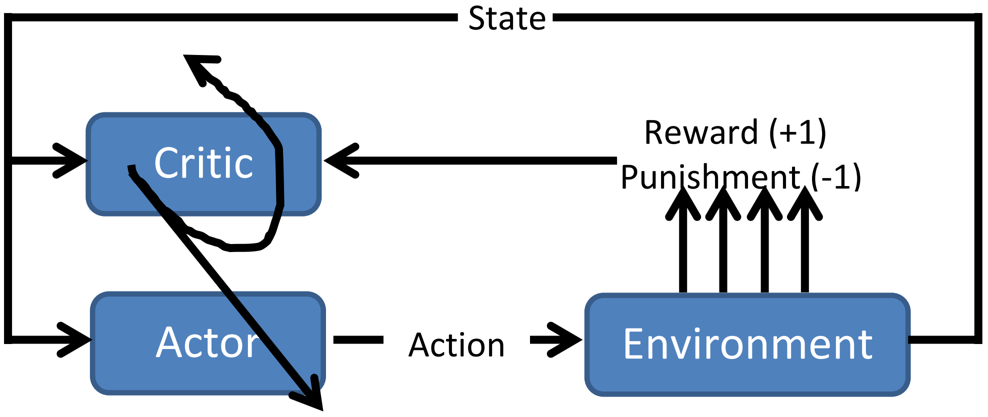
The Reward

The reward function is a signal that expresses the failure or success of performing a specific action and it is extracted from a state observation; in other words the reward maps each perceived state (or a state-action pair) of the environment to a single number. The reward indicates the intrinsic desirability of the observed state; therefore, if a policy selected action returns a negative reward, then the policy may be changed in order to select a different action for that given state in the future. For example, a simple system like the cart-pole example by Hinojosa (2011) can be credited with a reward of zero for every action that keeps it in balance and punished with a reward of -1 as soon as the controller takes an action that leads the pole out of balance. In the case of the grid world problem, where we want to use a value based reinforcement learning algorithm to learn how to reach the goal state as quickly as possible, a natural way to model goal states in a reward function is to give a positive reward on reaching the goal state and zero reward on all other steps. In the grid world this would translate to a positive reward for reaching the goal state and zero reward on each other step except boundary and hill states which are negative. The reward function would then also require a discount factor strictly lower than one. Otherwise, any action that eventually exits in the grid world is optimal and there is no incentive to reach the goal quickly. An alternative method would be to set the reward for each transition to some negative value and to use this as an incentive to find the goal quickly.

3.3 Multi Objective Actor-Critic Architecture

As discussed in chapter 2 of this thesis both actor only and critic only methods have strong theoretical foundations. However, their performance cannot be assured for the cases when the agent has to cope with deficient function approximation and partial observability. Critic only methods have a superior sample complexity and asymptotic performance when provided complete state information; yet, critic only methods are robust to inadequate function approximation and noisy state information. Although value function-based critic methods and actor-only methods are contrasting approaches to solve reinforcement learning tasks, it is possible to combine their advantages into an Actor-Critic (AC) architecture. AC methods are also temporal difference methods but are unique in the fact that they have two memory structures to represent the policy structure independently to the value function. The actor refers to the policy structure and is responsible for selecting actions whereas the critic is so called because it criticises the actions made by the actor. When combined into an AC structure, the learning is always on-policy: the critic must learn about and critique whatever policy is currently being followed by the actor. The critique takes the form of a TD error (a scalar signal) and represents the sole output of the critic which drives all learning in both actor and critic, as seen in

Figure 3.1 Multi Objective Actor-Critic Architecture



In the Actor-Critic architecture illustrated previously, at any given time, the critic is learning the values for the Markov chain that comes from following the current policy of the actor. The actor is constantly learning the policy that is greedy with the respect to the critic's current values. AC methods were among the first reinforcement learning algorithms to use temporal-difference learning. These methods were first studied in the context of a classical conditioning model in animal learning by Sutton and Barto (1981). Later, Barto et al, (1983) successfully applied AC methods to the cart-pole balancing problem, where they defined for the first time the terms actor and critic. In the simplest case of finite-state and action spaces, the following AC algorithm has been suggested by Sutton and Barto (1998). After choosing the action a_t in the state s_t and receiving the reward r_t, the critic evaluates the new state and computes the temporal-difference (TD) error δ_t according to (3.1):

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3.1}$$

where γ is the discounting rate and V is the current value function implemented by the critic.

The next step that proceeds is to update the critics value function:

$$V(s_t) \leftarrow V(s_t) + \alpha_t \delta_t \tag{3.2}$$

Where α_t represents the critic's learning rate at time t. The key step in this algorithm is the update of the actor's parameters. If TD error is positive, the probability of selecting that action in that state in the future should be increased since action a has resulted in a better than expected state value. By reverse logic, the probability of selecting $a = a_t$ in the state $s = s_t$ in the future should be decreased if the TD error is negative. Suppose the actor chooses actions stochastically using the Gibbs softmax method:

$$\Pr\{a_t = a | s_t = s\} = \frac{e^{\theta(s,a)}}{\sum_b e^{\theta(s,b)}} \tag{3.3}$$

where $\theta(s, a)$ is the value of the actor's parameter indicating the tendency of choosing action in state. Then, these parameters are updated as follows:

$$\theta(s_t, a_t) \leftarrow \theta(s_t, a_t) + \beta_t \delta_t \tag{3.4}$$

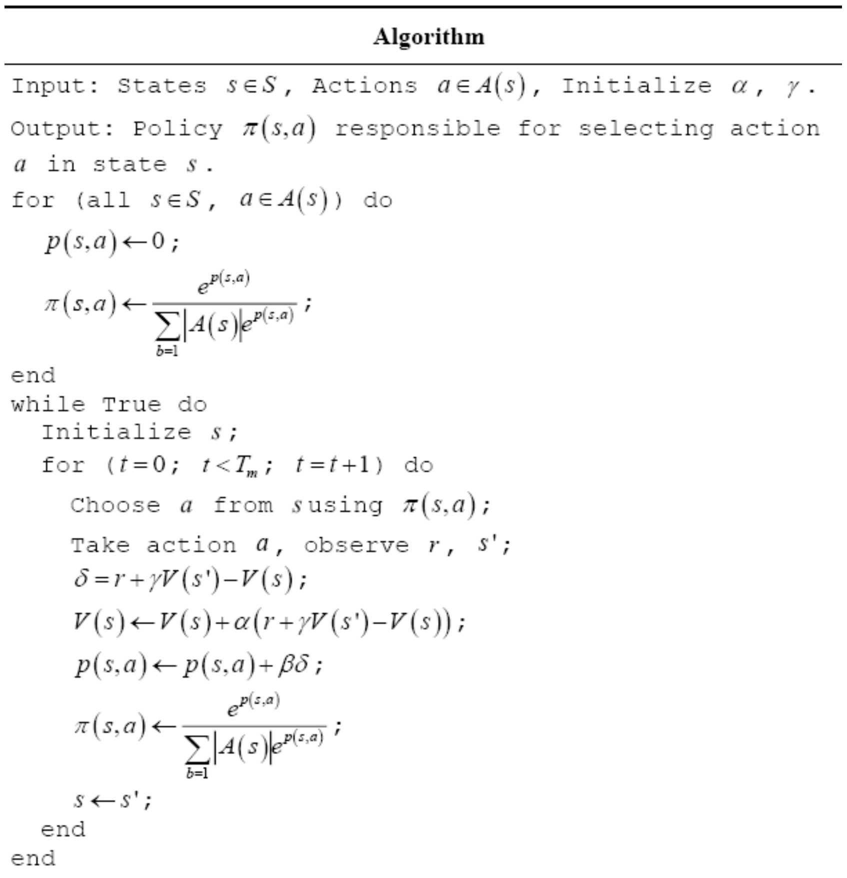
where β_t is the actor's learning rate at time t.

An AC system first updates the value in every state once, then it updates the policy in every state once, finally the process is repeated, this occurs in the form of incremental value iterations. This process is a form of dynamic programming, which is guaranteed to converge to the optimal policy. If it instead updates all the values repeatedly in all the states until the values converge, then updates all the policies once, then repeats, then it reduces to policy iteration, another form of dynamic programming with guaranteed convergence. If it updates all the values N times between updating the policies, then it reduces to modified policy iteration, which is also guaranteed to converge to optimality.

In conclusion we list some of the most important advantages of using actor-critic methods below which is followed by a pseudo code description of the crucial steps to be executed:

- Explicit representation of policy as well as value function.
- Minimal computation to select actions.
- Can learn an explicit stochastic policy.
- Can put constraints on policies.
- Appealing as psychological and neural models.

Figure 3.2: Actor Critic Pseudo Code (*Source*: Hinojosa et al, 2011)



Algorithm

```
Input: States $s \in S$, Actions $a \in A(s)$, Initialize $\alpha$, $\gamma$.
Output: Policy $\pi(s,a)$ responsible for selecting action
$a$ in state $s$.
for (all $s \in S$, $a \in A(s)$) do
  $p(s,a) \leftarrow 0$;
  $\pi(s,a) \leftarrow \frac{e^{p(s,a)}}{\sum_{b=1} |A(s)| e^{p(s,a)}}$;
end
while True do
  Initialize $s$;
  for ($t=0$; $t<T_m$; $t=t+1$) do
    Choose $a$ from $s$ using $\pi(s,a)$;
    Take action $a$, observe $r$, $s'$;
    $\delta = r + \gamma V(s') - V(s)$;
    $V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$;
    $p(s,a) \leftarrow p(s,a) + \beta\delta$;
    $\pi(s,a) \leftarrow \frac{e^{p(s,a)}}{\sum_{b=1} |A(s)| e^{p(s,a)}}$;
    $s \leftarrow s'$;
  end
end
```

The learning process of a GPFMORL is based on an Actor-Critic reinforcement learning scheme, where the actor learns the policy function and the Critic learns the value function using the TD method simultaneously. This makes it possible to focus on on-line performance, which involves finding a balance between exploration of uncharted territory and exploitation of current knowledge.

RL typically requires an unambiguous representation of states and actions and the existence of a scalar reward function. For a given state, the most traditional of these implementations would take an action, observe a reward, update the value function and select, as the new control output, the action with the highest expected value (probability) in each state (for a greedy policy evaluation). The updating of the value function is repeated until convergence is achieved. This procedure is usually summarized under policy improvement iterations.

The parameter learning of the GPFMORL system includes two parts: the Actor parameter learning and the Critic parameter learning. One feature of the Actor–Critic learning is that the learning of these two parameters is executed simultaneously.

Given a performance measurement and a minimum desirable performance we define the external reinforcement signal r as:

$$r = \begin{cases} 0, & \forall \quad Q(t) \geq Qmin > 0 \\ -1 & \forall \quad 0 \leq Q(t) < Qmin \end{cases} \tag{3.5}$$

The internal reinforcement $\bar{r}$ expressed in (3.6), is calculated using the temporal difference of the value function between successive time steps and the external reinforcement.

$$\bar{r}_k(t) = r(t) + \gamma p_k(t) - p_k(t-1) \tag{3.6}$$

Where γ is the discount factor used to determine the proportion of the delay to the future rewards and the value function $p_k(t)$ is the prediction of eventual reinforcement for action a_k

The goal of reinforcement learning is to adjust correlated parameters in order to maximize the cumulative sum of the future rewards. The main goal of the Actor is to find a mapping between the input and the output of the system that maximizes the performance of the system by maximizing the total expected reward. The role of the Critic is to estimate the value function of the policy followed by the Actor. The TD error is the temporal difference of the value function between successive states. The goal of the learning agent is to train the Critic to minimize the squared TD error.

3.4 Fuzzy Logic and Fuzzy Inference Systems

Fuzzy logic is a technique originally developed by Zedeh (1973) as a way of processing data by allowing partial set membership rather than crisp set membership or non membership. In a similar way that this thesis' contributions stem from the extension of a previously developed GPFRL theory, Fuzzy Logic (FL) is an extension of the conventional Boolean Logic theory. It is based upon fuzziness and uncertainty of non crisp logic and the main concept concerns the degree of truth in the sense that the degree of truth is no longer limited to the distinct values zero or one. For example: Belonging to a set is not a binary rough criteria, instead it is described with some uncertainty inside the range of zero to one that allows a partial membership.

Due to these fascinating characterisations of fuzzy logic, many interesting advantages are presented. Firstly FL systems have demonstrated an outstanding capability for mapping non linear relationships of input and output models without the need for precise mathematical formulation. Pinder (2013) demonstrates the use of a fuzzy altitude controller to control hovering height of a helicopter equipped with a camera and angled mirror. Image processing was used to estimate the non linear area to distance relationship in order to be fuzified using linguistic variables represented by natural language. Descriptive linguistic variables then allow the fuzzy if-then rules to be constructed to eventually control output actions to alter the throttle depicting the helicopters vertical motion in three dimensional space.

In addition to the previously mentioned advantages one of the most interesting is that FL systems are capable of handling and representing non statistical uncertainty, distinguishing traits of which probability theory cannot. The fuzzy altitude controller also provides a convenient way of mapping input space to output space whilst being roust to data input errors ie: poor estimates of the helipads area possibly due to illumination factors, shadows or simply a noisy video stream.

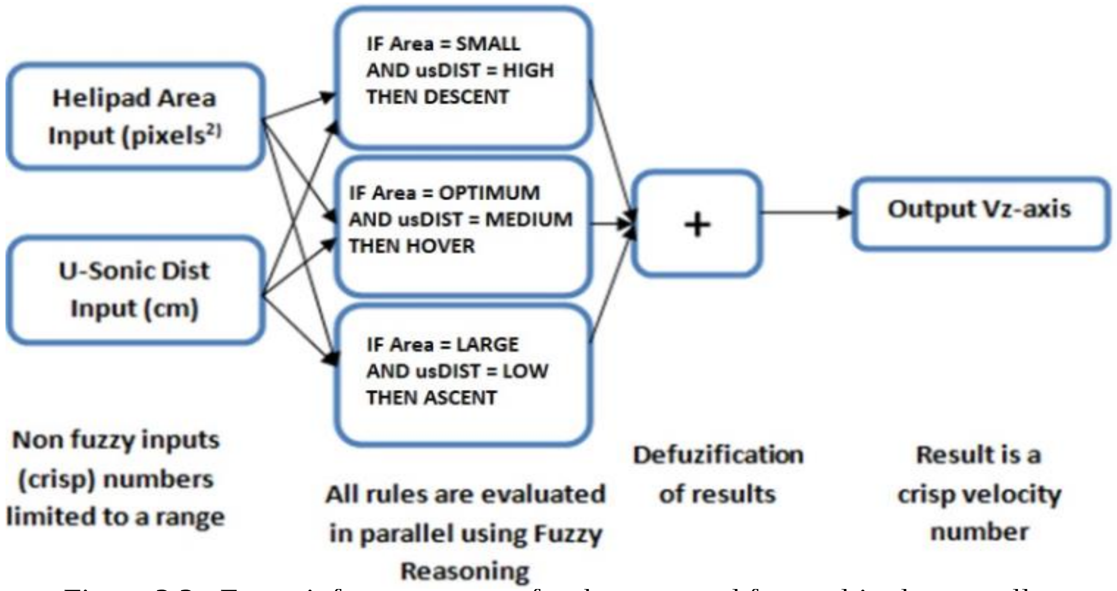

Helipad Area Input (pixels²)
U-Sonic Dist Input (cm)
IF Area = SMALL AND usDIST = HIGH THEN DESCENT
IF Area = OPTIMUM AND usDIST = MEDIUM THEN HOVER
IF Area = LARGE AND usDIST = LOW THEN ASCENT
+
Output Vz-axis
Non fuzzy inputs (crisp) numbers limited to a range
All rules are evaluated in parallel using Fuzzy Reasoning
Defuzification of results
Result is a crisp velocity number

Figure 3.3: Fuzzy inference system for the proposed fuzzy altitude controller

FIS have been successfully applied in many different fields ranging from automatic control and expert systems to data classification, decision analysis and computer vision. Due to this multidisciplinary nature, FIS became associated with a rather large number of methods such as fuzzy-rule-based systems, fuzzy expert systems (Siler and Buckley, 2004), fuzzy modelling, fuzzy associative memory (Kosko, 1991), and the one of especial concern for this work, fuzzy logic controllers.

When a FIS is used for controlling systems it is called fuzzy logic controller (FLC). FLC are especially advisable for cases where the mathematical model of the system to be controlled is unavailable, and the system is known to be significantly nonlinear, time varying, or to have a time delay. Within the area of automatic control, FIS have had great success in the field of robotics. FLC are particularly suitable for implementing systems with stimulus-response behaviour, since fuzzy rules provide a natural framework to describe the way the system "should react" whilst providing human reasoning capabilities in order to capture uncertainties (Jang et al., 1997).

One of the first control systems built using fuzzy set theory (and one of the most commonly used) is the Mamdani's fuzzy inference method (Mamdani and Assilian, 1975). Mamdani's effort was based on Zadeh's paper on fuzzy algorithms (Zadeh, 1973) for complex systems and decision processes. Another important method is the Sugeno inference method. In general, Sugeno systems can be used to model any inference system in which the output membership functions are either linear or constant.

3.5 Probabilistic Theory and Uncertainty

From all the methods for dealing with uncertainty mentioned earlier in this chapter, probabilistic theory is the oldest (can be back traced to the early 1960s) and the best understood of all. As a contrast with FL, probability theory concerns with the concept of “probability of truth” and gives information about the likelihood of an event in the future by representing information through probability densities. Therefore, probability and fuzziness are concepts that represent two different kinds of uncertainty, statistical and non-statistical, respectively.

In recent years this probabilistic approach has become the dominant paradigm in a wide array of problems, ranging from financial (Berg et al., 2004, Almeida and Kaymak, 2009), control (Liu and Li, 2005, Blackmore, 2006, Hinojosa et al., 2008), robotics (Thrun, 2000, Valavanis and Saridis, 1991, Park et al., 2007, Thrun et al., 2000, Jaulmes et al., 2005), and for representing uncertainty in mathematical models (Ross, 2004). Some research work, as in Cheeseman (1985), supports the idea that all the numerous schemes for representing and reasoning about uncertainty featured in the AI literature are unnecessary as probability theory can easily and effectively deal with this issue.

Probability theory attempts to quantify the notion of probable. The general idea is divided into two concepts:

- Aleatory (objective) probability, which represents the likelihood of future events whose occurrence, is governed by some random phenomena.

- Epistemic (subjective) probability, which expresses the uncertainty about the outcome of some event, in the lack of knowledge or causes.

If we apply the concepts of probabilistic theory to the particular case of an embodied agent, this approach becomes divided in two fields, (Thrun, 2000):

- Probabilistic perception: Deals with the uncertainty about the external world, as captured by the sensors of the embodied agent by using a probability distribution of the captured data instead of discreet values.

- Probabilistic control: Given the uncertainty about the environment, a learning agent faces the task of taking a decision about its next action which consequences might be uncertain, especially over the long-term. In this same particular case, a probabilistic controller will need to anticipate various contingencies that might arise in uncertain worlds, by blending information gathering (exploration) with robust performance-oriented control (exploitation). In this case, the uncertainty of the state is propagated forward in order to obtain a probabilistic representation of a long-term behaviour. There are two broad views on probability theory for representing uncertainty: the frequentist and the subjective or Bayesian view.

- The Frequentist View, sometimes also referred as “empirical” or “a posteriori” view of probability, relates to the situation where an experiment can be repeated indefinitely under identical conditions, but the observed outcome is random. Empirical evidence suggests that the relative occurrence of any particular event, i.e. its relative frequency, converges to a limit as the number of repetitions of the experiment increases. Therefore, probabilities are defined in the limit of an infinite number of trials.

- The Subjective View, was originally introduced by Pearl (1982, 1988) and further developed later by Lauritzen and Spiegelhalter (1988). The subjective or Bayesian view of probability is used as a belief where the basic idea in the application of this approach is to assign a probability to any event based on the current state of knowledge and to update it in the light of the new information. The conventional procedure for updating a prior probability in the light of new information is by using Bayesian theorem where subjective probabilities quantify degrees of belief.

Other less common views can include the classical view, sometimes referred as “a priori” or “theoretical” and the axiomatic view, which is a unifying perspective aimed to provide a satisfactory formal structure for the development of a rigid theory by focusing on the question “How does probability work?” rather than trying to define what probability is.

As probabilistic methods and fuzzy techniques are good for processing uncertainties (Zadeh, 1995, Laviolette and Seaman, 1994), it would be beneficial to endow FLS with probabilistic features. The integration of probability theory and fuzzy logic has been the subject of many studies as in Liang and Song (1996).

3.6 GPFMORL Algorithm Flowchart

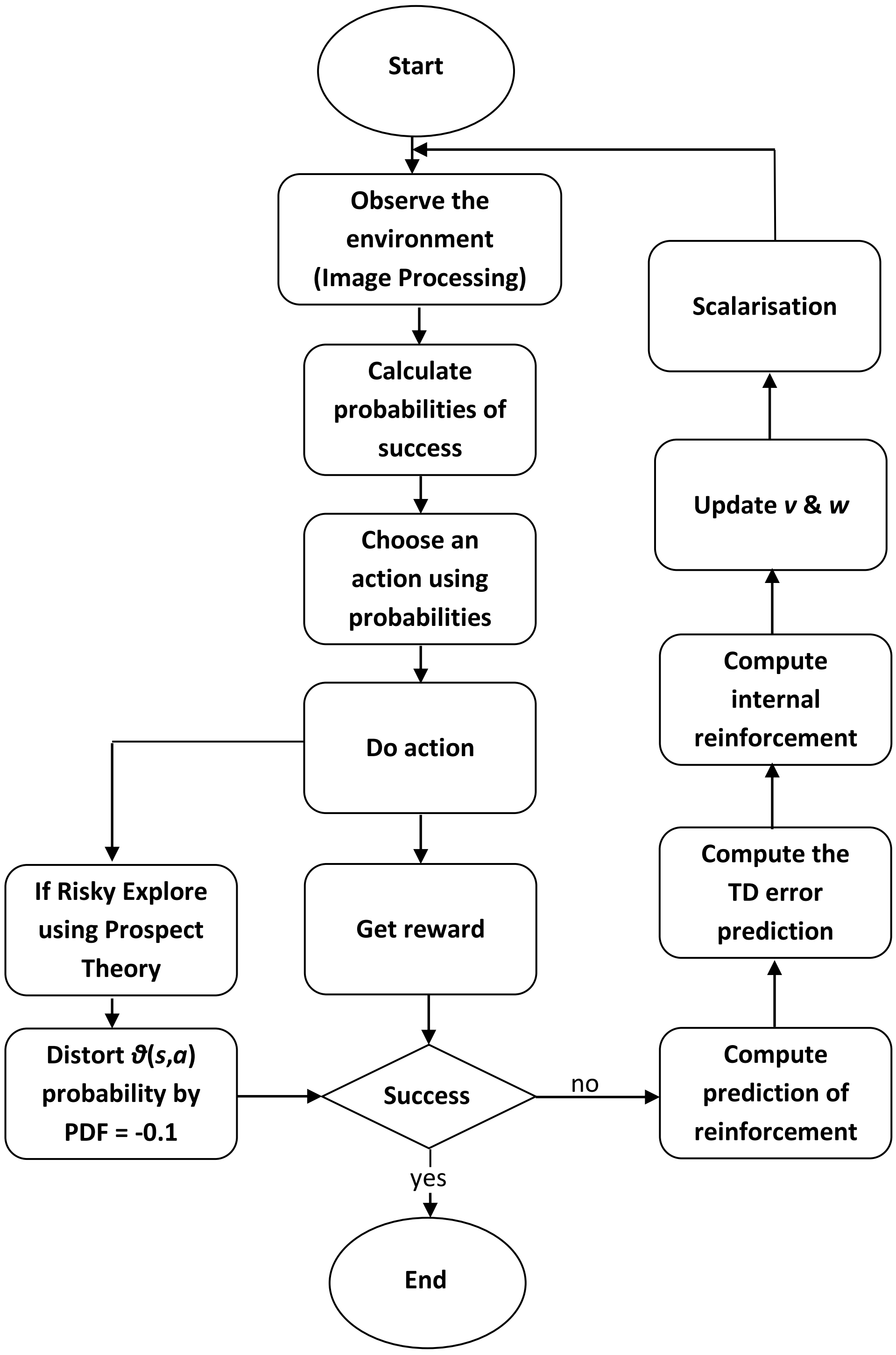

Start
Observe the environment (Image Processing)
Calculate probabilities of success
Choose an action using probabilities
Do action
If Risky Explore using Prospect Theory
Distort $\vartheta(s,a)$ probability by PDF = -0.1
Get reward
Success
no
yes
End
Compute prediction of reinforcement
Compute the TD error prediction
Compute internal reinforcement
Update v & w
Scalarisation

Figure 3.4: Proposed GPFMORL Algorithm Flowchart

3.7 Rationale behind extending GPFRL with a Multi Objective Methodology

This chapter has explored the different advantages and limitations of three different popular paradigms, fuzzy logic systems, probabilistic theory and reinforcement learning separately. It was concluded that each of this paradigms could very well complement the drawbacks of the others and seamlessly work together in a cooperative rather than competitive way.

Fuzzy logic systems are good at generalizations, and due to its distinctive characteristics, it is able to handle non-statistical uncertainties, and fuzziness; however under certain conditions, the design and development of rather large or more complex systems can be too complicated for human operators.

Reinforcement learning methods have been an intense focus of research in the last decade. Research has proven that reinforcement learning can be successfully used in many different areas, such as decision making or control. A remarkable characteristic is that RL methods do not require input-output pairs for training or previous knowledge of the environment model. RL only uses sparse signal information in order to reach to optimal conclusions. Therefore using RL for automatic tuning of fuzzy logic parameters and even image processing parameterisation is the focus of recent research.

Probabilistic theory is still one of the most effectives way (and most explored) to deal with uncertainties, especially stochastic uncertainty. The fusion of probabilistic theory with fuzzy logic controllers have shown to be a powerful tool for practical areas such as finance and weather forecasting. Both paradigms can work in collaboration, in order to complement each other. As a result, probabilistic fuzzy logic system can handle a very large range of uncertainties.

The present work combines these three paradigms into a novel multi objective method, able to learn optimal policies for control or decision making whilst being resistant to stochastic, non-stochastic, uncertainties, randomness and fuzziness.

3.8 Conclusion

Fuzzy logic is an effective tool for solving many nonlinear control problems, where the nonlinear behaviour of the system makes it difficult, if not impossible, to build an analytical model of the system. Additionally, fuzzy set theory provides a mathematical framework for modelling vagueness and imprecision. However, building a fuzzy controller has its own difficulties.

The process of designing a FLC has two challenging tasks: defining the controller structure and, second, finding the numerical values for the controller parameters. These challenges arise due to a lack of a well-established theoretical approach; rather, they are entirely based on the empirical experience of a human operator, which is transferred into the FLC. However, the extraction of the expert's knowledge is not always an easy task; decision rules with complex structures and an excessively large number of variables necessary to solve the control task introduce difficulty in performing the knowledge extraction.

A direct solution to these problems is to use learning algorithms in order to replace or enhance the human operator "a priori" knowledge. Fuzzy logic learning can be used to automatically provide a solution for these issues, thereby removing human input from the design.

Several techniques reported in recent literature to create such intelligent controllers include the use of algorithms such as neural networks, genetic algorithms, and more recently reinforcement learning, in order to learn and optimize a fuzzy logic controller.

Design & Implementation

4.1 Introduction

This chapter explains the implementation of a number of components necessary for our proposed reinforcement learning approach. In sub section 4.2, we consider a Helipad Visual Localisation method using Cellular Decomposition (CD) in order to allow the UAV to approximate its relative location and therefore its current state. Subsection 4.3 illustrates the potential field methodology (PFM) used for assigning reactive behaviours when in danger of flying outside the boundaries of the grid. Section 4.4 conceptualises the development of a fuzzy altitude controller for quadcopters using only the perceived area of the helipad combined with the ultrasonic distance estimates to provide a more robust multi-modal altitude estimation algorithm. Section 4.5 explains the simulation experiments of using GPFMORL for a locating goal location in the fewest number of steps starting from both static and random starting positions within the grid. The UAV positional controller's architecture is illustrated in subsection 4.6 which is used by the MORL agent in order to facilitate as a real world controller in addition to the decision making capabilities demonstrated in the simulation results. Subsection 4.7 details some representative MORL benchmarks and methods of empirical evaluation. Subsection 4.8 includes a brief analysis of preliminary test results achieved using both simulation and real world application of the UAV with appropriate performance measures to compare the performance of our proposed algorithm with other reinforcement learning and intelligent approaches. Finally the conclusion of this section can be found in subsection 4.9 which also summarises the implementation and overall design of the proposed system.

4.2 Multi Objective Linear Scalarisation

This approach to MORL translates a multi objective problem to a single objective task by applying a function to the reward vector in order to produce a single, scalar reward. The scalerisation function can be either a non-linear function tuned to the problem domain (Tesauro et al. 2007) or more commonly a linear weighted sum of the objective rewards. The use of weights allows the user to choose some control over the nature of the policy to be found by the system, by placing greater or lesser emphasis on each objective to be achieved. The fundamental limitation with linear scalarisation is that it cannot find policies which lie in non-convex regions of the pareto front.

When scalarisation is performed on Q-Learning, we actually scalarise the state value function which is a single value for each state representing the likelihood score of expected reward for transitioning to the reward location from any state. However, formally scalarisation is performed on state-action pairs representing the likelihood of reaching the goal/reward location from a specific state when taking any particular action available from the action set. After careful consideration it was decided that we should either re-implement q-learning to generate value functions for every state-action pair, or we should continue with my efforts of using GPFRL as a baseline algorithm which does in fact store state- action pairs through the use of a multi-agent approach.

Also we know that GPFRL outperforms other single objective algorithms such as Q-Learning & SARSA therefore it is highly unlikely that the multi objective version of Q-Learning would yield better performance results that GPFMORL. Therefore, the decision was made to focus on the multi-objective version of GPFRL which shall be referred to as GPFMORL from here on.

GPFMORL

The work of Hinojosa et al (2011) consisted of a multi agent algorithm (GPFRL) where each agent learns the probabilities of success upon choosing one of the four cardinal directions, forward, backward, left and right. For every step within the discretised grid world that results in the agent being a state closer to the goal state, it received a reward of +1. In contrast, for every step it takes in a direction that is not towards the goal, it receives a punishment of -1. Importantly, actions that would take the agent off the grid leave its location unchanged but also result in a punishment of -1, other actions result in a reward of 0.

The first modification of the GPFRL algorithm was derived from the realisation that -1 punishment for moving further from the goal was equal to -1 punishment generated from the much more serious action of exiting the grid world by crossing the boundaries. The "Over Boundary" penalty can be enhanced by additionally reducing the probability derived from the actor weights for a specific action taken when in a particular state. This additional probability reduction is performed by introducing a small weighting factor called the "Potential Field Penalty Factor" (PFPF = 0.01) which is multiplied by the probability value to reduce the chance of performing illogical actions such as exiting the grid boundary when in risky states. Modifying this PFPF value allows greater or less emphasis on the secondary objective of avoiding the grid boundaries.

The second modification attempted a linear scalarisation to the single objective (GPFRL) algorithm which stored the expected reward for a combination of state *j* and action *k* pairs in the Actors policy function. In the GPFMORL algorithm, the two-dimensional value function array *Wjk* is extended to incorporate multiple objectives *Wjko* by a three-dimensional array so that the expected rewards for each state *j*, action *k* and objective *o* can be stored, retrieved and updated separately. Therefore in the case of two objectives, the second objective is stored in the third dimension. However, this approach enables more than two objectives to be achieved by extending the array to n-dimensions depending upon the number of objectives to be achieved.

Finally the MDP for single objective RL is also extended to by replacing the single scalar reward signal by a vector of reward signals ie: $\vec{R}$(si, ai) = (R1(si, ai),…Rm(si, ai)), where *m* represents the number of objectives. Now that the reward vector consists of multiple components, each representing different objectives, it is likely that conflicts arise whilst trying to optimise one or more objectives. In such cases, a trade-off between these objectives has to be learned, resulting in a set of policies. As mentioned previously the set of optimal policies for each objective, or a combination of objectives is referred to as the Pareto Optimal Set. In addition to conflicting objectives there also may be multiple objectives which are complementary or independent by nature. For example the benchmark proposed in this paper involves complementary objectives of maximising stabilisation above the helipad in preparation for landing whilst minimizing the possibility of becoming lost by avoiding the grid boundaries.

By employing scalarisation functions as a scoring mechanism, a multi objective problem is transformed into a single objective by performing a function over the objectives to obtain a combined score for an action *k* for different objectives *o*. This single score can then be used to evaluate the particular action *k* by utilising the standard action selection strategies of single objective reinforcement learning such as E-Greedy to decide which action to select.

For a multi objective solution x, a weighted-sum is performed over each objective function (ie: f_o with o =1..m) and their corresponding weights to obtain the score of x, for example.

$$\mathrm{LS}(x) = \sum_{0=1}^{m} w_o \ . \ f_o(x) \qquad (4.1)$$

In the case of GPFMORL, the objective functions f are considered the actors policy structures Wjko values. As a result of applying the scalarisation, scalarised W-Values or SW-values are obtained.

$$\mathrm{SW}_{jk} = \sum_{0=1}^{m} w_o \ . \ W_{jko} \qquad (4.2)$$

The action corresponding to the largest weighted-sum or SW-Values is considered the greedy action in state *s*, formulated as the following:

$$\mathrm{greedy}_{k'}(s) \underset{k'}{=} \max \mathrm{SW}(j,k') \qquad (4.3)$$

Pseudo Code

```
Initialise W(j,k.o) arbitrarily
foreach episode T do
Initialise state s
repeat
        Choose action kfrom state j using policy derived from $W_{jko}$(e.g. scal-ε-greedy)
        Take action k and observe state j'ϵ J and reward vector $\vec{r}$ϵ $\mathbb{R}$
        $greedy_{k'}$(j'), ← Call scal. Greedy action selection
        foreach objective odo
                $W_{jko} \leftarrow W_{jko} + \alpha[\vec{r}_{jko} + \gamma W(j', greedy_{k'}(j'),o) - W_{jko})]$
                end for
                jj'←
until s is terminal
end for
```

In his approach there are four learning agents, where each one learns the probabilities of success of going in each one of the four (or 8) directions. So for every step that the agent takes that result with the agent being in a state closer to the goal state, it receives a reward of one, “1”; in the same way for every step it takes in a direction that is not towards the goal, it receives a punishment of “-1”. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions result in a reward of 0.

The -1 reward for leaving the grid can be improved by additionally reducing the probability of taking an OVER GRID action whenever in that particular state. This is done by introducing a small fraction PFM Factor which reduces the probability by 10% of that states actor value function according to the responsible agent for the non-logical move. No adjustment to the critics memory structure are performed due to the punishment being instant and not a temporal credit assignment reward function relating to the goal.

“For every step it takes in a direction that is not towards the goal, it receives a punishment of “-1”Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1.”

Creating a richer reward scheme appears to not only allow more than one objective to be specified by the designer but also improves performance. Confirming my hypothesis that in addition to the -1 for leaving the grid there should be a more severe punishment in addition to that of moving further away from the goal whilst still being in a safe zone state.

Therefore in addition to this punishment of -1, the probability value also needs to be decreased further using the proposed PFM-Factor by multiplying them together.

Preliminary results with the PFM factor <0.1 sometimes results in the GPFRL finding the goal in less steps than the GPFMORL & GPFRL sometimes even converges faster, however the GPFMORL always finds the optimal policy with fewer trials than the GPFRL.

4.3 Prospect Theory

In 1979, Daniel Kahneman and Amos Tversky, two researchers in behavioural psychology, developed prospect theory which aims to model human decision making in the context of risk. This model fits in the field of subjective probabilities and helps to make decisions depending on the context as well as balancing an aversion of loss based on the current state. The purpose of prospect theory allows the MORL agent to establish a dynamic compromise between exploration and exploitation during learning. Based on a weighting of probabilities, it allows us to solve muti objective problems with reasonable computational and adds a level of dynamism to the proposed scalarisation technique.

A review of existing work suggests that prospect theory is a descriptive theory to model the way in which human beings make decisions that involve risk. It is considered to be the model used by behavioural economics since 1979. According to prospect theory, decision-making under risk may be seen as a choice between several gambles or prospects. The basic equation of the theory that determines the value of utility $v(x, p)$ of a simple prospect that pays £x with probability p (and nothing otherwise)

$$v(x,p) = v(x)w(p) \tag{4.4}$$

where the value function $v(x)$, and the function for weighting the probability $w\ (p)$, are non-linear transformations for the outcomes (x), and the probabilities for each possible outcome (p), respectively, and they represent the participants' perception of both values.

The value function and the probability weighting function are shown in Fig4.1 A & B

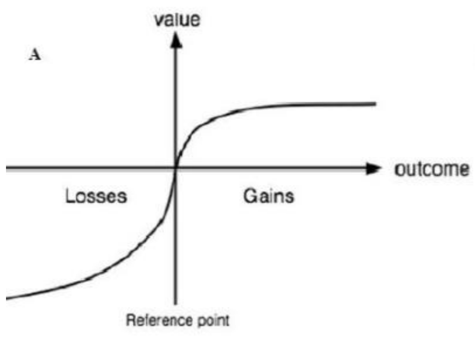


Figure 4.1A: The value function v(.) as a function of gains and losses

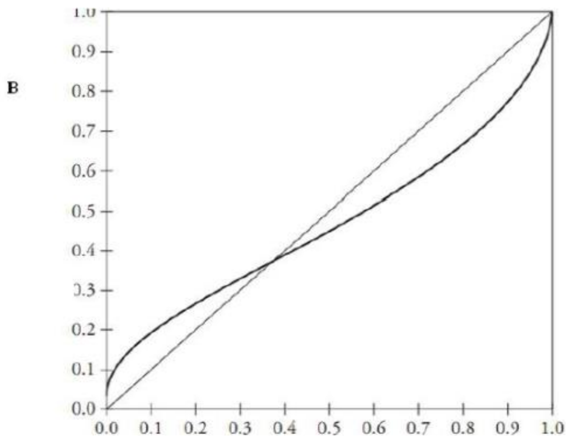


Figure 4.1B: Weighting function (w.) for gains as a function of the probability p of a chance event. (Kahneman and Tversky 1979)

Reference Point

Kahneman & Tversky (1979) defined the reference point (RP) which determines if the result is a gain or loss. The principle comes from the fact that, when confronted by a decision making problem, the decision-maker analyses the options gain or loss based on the current state; this is also referred to the status quo. This reference works as a behavioural trigger where, how risky someone behaves all depends upon the status quo. According to Kahneman & Tversky (1979) people behave more riskily when dealing with losses rather than dealing with gains. The utility value in Prospect theory depends upon the status quo or in our experiments the quadcopter's current state.

In Equation (3.2), it is assumed that the outcomes (the values) are already manipulated with the status quo set as zero. where sq is the value of the status quo.

$$v(x) = v(x - sq) \tag{4.5}$$

The Value Function

In PT, the outcomes are assigned to gains and losses rather than to final assets; in addition, the value function captures how much better one gain is than another gain and how much worse one loss is than another loss. Moreover, the value function is steeper for losses than for gains, a property known as loss aversion. Kahneman et al. (1997) observed that people typically require more compensation to give up a possession than they would have been willing to pay to obtain it in the first place.

Figure 4.1A shows that the three psychological principles—the RP, DS, and loss aversion–constrain the shape of the value function. The first principle, the RP, suggests that outcomes are viewed relative to a RP and, hence, coded as gains or losses. For example, a person is tall only in comparison to others who are shorter. This comparison in PT is performed with respect to the RP. This means that people might accept an option in one situation that they reject in another (Chiu & Wu, 2010).

In the second principle, DS, the value function is S-shaped and predicted to be concave for gains above the RP and convex for losses below the RP. This means that differences between small gains or losses close to the RP are assigned a high value, whereas differences further away from the RP are assigned smaller values. For example, there is a big difference between a £100 gain and a £200 gain, but a much smaller difference between gains of £1,100 gain and £1,200. Similarly, a loss of £100 seems quite distinct from a loss of \$200, but losses of \$1,100 and \$1,200 seem pretty similar.

In the third psychological principle, loss aversion forms the shape of the value function in the loss region, where the value function appears steeper for losses than for gains (Figure 4.1A). This means that a loss is assigned a greater value than a gain of an objectively identical amount. Thus, losses are given more significance than gains. To explain this principle, for example, a loss of £100 seems much more painful than a gain of £100 which seems pleasurable. Most people dislike a prospect that gives an equal chance of winning £1,000 or losing £1,000.

Tversky & Kahneman (1992) formulated the value function as a power function:

$$v(x) = \begin{cases} x^{a}, & x \geq 0 \\ -\lambda(-x^{\beta}), & x < 0 \end{cases} \tag{4.6}$$

Where α and β>0 measures the curvature of the value function for gains and losses respectively, and λ is the coefficient of loss aversion. Thus, the value function for gains (losses) is increasingly concave (convex) for smaller values of α (β) < 1, and loss aversion is more pronounced for larger values of λ > 1. The estimated values for the value function parameters as concluded from a study by Tversky & Kahneman (1992) of a sample of college students were: α= 0.88, β=0.88, and λ= 2.25.

<u>Probability Weighting Function</u>

PT assumes that individuals do not weight outcomes by their probability, as in EUT, but by some distortion of probabilities. This distortion of probability is captured by prospect theory's probability weighting function. Figure 4.1B implies an inverse-S-shaped probability weighting function, which is concave near zero and convex near one (multidimensional character).

In weighting functions, two of the three psychological principles, the RP and DS, govern the shape of the function. For the probability, there are two obvious RPs, certainty and impossibility, or a 100% chance and a 0% chance. The distortion of the probability shown in the probability weighting function captures the DS away from these two RPs. People are most sensitive to changes in probability when they are near 0% or 100% than when the change applies to intermediate probabilities. People will pay much less, for example, for a lottery in which they have a 99% chance of winning

£1,000 than they will for a lottery in which they have a 100% chance of winning £1,000, but there is little difference between the amount people would pay for a 50% versus a 51% chance of winning £1,000. The objective difference in the probabilities (1%) is identical, but its impact on one's decision is not.

The probability weighting function is exhibits DS also. The function is concave for small probabilities and convex for medium and large probabilities (See Figure 4.1B). Involves DS into probability weighting function leads to give more weight to low probability than they would receive using EUT. This overweighting is consistent with risk-seeking for low probability gains and risk-aversion for low probability losses. Thereby which explaining non-rational gambling behaviour and providing insurance against very low probability events. In contrast, medium to high probabilities are given less weight than they would receive using EUT. Such underweighting is consistent with risk-aversion for medium to high probability gains, and risk-seeking for medium to high probability losses.

Scholars and researchers such as Gonzalez & Wu (1999), Abdellaoui (2000), and Wakker (2001) have confirmed that the inverse-S-shaped weighting function seems to be consistent with a range of empirical findings. The weighting function can be parameterised in the following form according to the probability weighting function originally proposed by Tversky & Kahneman (1992):

$$w(p) = \frac{p^y}{(p^y+(1-p)^y)^{1/y}} \tag{4.7}$$

Where p is the weighting probability of the distribution of gains or losses and > 0 measures its degree of curvature..

For all simulations the parameter values shall be the estimated values by Tversky & kahneman (1992) n = 0.88, P = 0.88, y = 0.75 and A = 2.25

- Reference Point

  The reference point uses the current value of reward. Whenever one modifies the values of rewards, it accumulates for each objective, these values in the reference point. For the GPFMORLPT algorithm, it uses the values in the critic table.

- Probability weighting function

  Our simulation environment probability weighting function was not only stochastic, but also allowed the generation of probability distribution. It was then extracted from the values of reward which is then applied as a distortion independently for each of the criteria.

$$w(p) = \frac{p^y}{(p^y+(1-p)^y)^{1/y}} \tag{4.8}$$

- The value function computes the value function with the formula above, by adding the offset reference point:

$$v(x) = \begin{cases} (x - RP)^a, & x \geq 0 \\ -\lambda(RP - x)^\beta, & x < 0 \end{cases} \tag{4.9}$$

where *RP* is the current value of the reference point, and *x* is the current Q-value or the current critics value function for any particular state.

For each possible action, we associate the following results:

$$PT(x) = w_1(x).\, v_1(x) + \cdots + w_n(x).\, v_n(x) \tag{4.10}$$

where *PT* is the value from the perspective of the corresponding action x, $w_1(x)$ is the weighting for objective 1, $v_1(x)$ its value function for objective 1 and *n* represents the number of objectives.

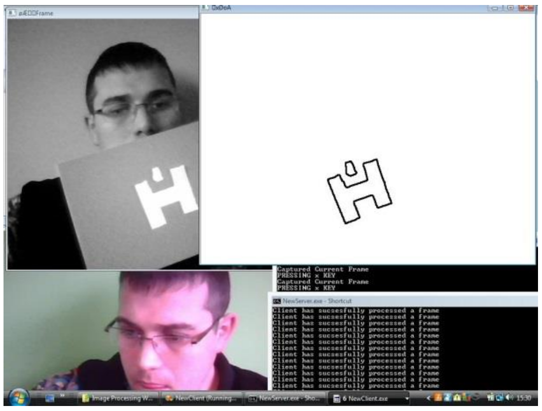
4.4 Case Study Research Prototypes

4.4.1 Unmanned Aerial Vehicle Visual Navigation Framework

4.4.1.1 Helipad Visual Localisation using Cellular Decomposition (CD)

Cellular decomposition (CD) is a simple yet effective technique for discretization of large continuous input spaces into smaller well defined regions each one being unique to the rest. The CD technique particularly lends itself well to our chosen application due to the fact that reinforcement learning determines what best actions to choose based upon probabilities derived from what particular state the agent in currently occupying. The UAV observes the environment through a downwards facing camera where real time video is streamed back to the workstation PC where it is then evenly divided up into cells or state space. In order to maintain the intuitive methodology, a relatively small number of 25 cells were used to describe the UAV's perception into discrete states forming a 5x5 grid. The 5x5 grid or "Cellular Decomposed State Matrix" (CDSM) as referred to throughout the rest of the thesis, is also commonly known as the grid world which can be altered to accommodate a variety of applications such as the introduction of stochastic cross winds to demonstrate the algorithms adaptive learning capabilities in the presence of risk and uncertainty. Providing the pixel screen resolution of the UAV's video stream is known, in addition to the Grid Size, we can calculate how many pixels each state should contain and more importantly which states are responsible for each pixel's location. Using this inferred information we can estimate the UAV's current position by using image processing to find the centroid of a helipad marker with respect to the position of the UAV in two dimentional space. Keeping the UAV stationary and moving the helipad marker on the floor would cause a change of state. Equally if the helipad marker was stationary and the UAV moved, then this would also cause a similar change in state. Taking advantage of this phenomenon we can implement visual localisation of the helipad for informing the MORL algorithm where the UAV is positioned above ie: what state the UAV is in. Now knowing where the UAV is positioned with respect to the helipad, further

Figure 4.2: OpenCV Visual Localisation



```
Captured Current Frame
PRESSING x KEY
Captured Current Frame
PRESSING x KEY
```

```
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
Client has sucsesfully processed a frame
```

information can be inferred such as which is the most appropriate action or action(s) to take when in this state.

Actions

The actions available to the agent are movements in the following directions: Forward, Backward, Left and Right. However to improve agility and take advantage of the UAV’s omnidirectional capabilities an additional 4 regions may be added for diagonal manoeuvres such as Forward-Left, Forward-Right, Backward-Left, and Backward-Right. The addition of fuzzy Near & Far membership functions allows the implementation of Multi Modal fuzzy translational velocity controller for more precise control of the UAV’s state transitions (Figure: 4.3).

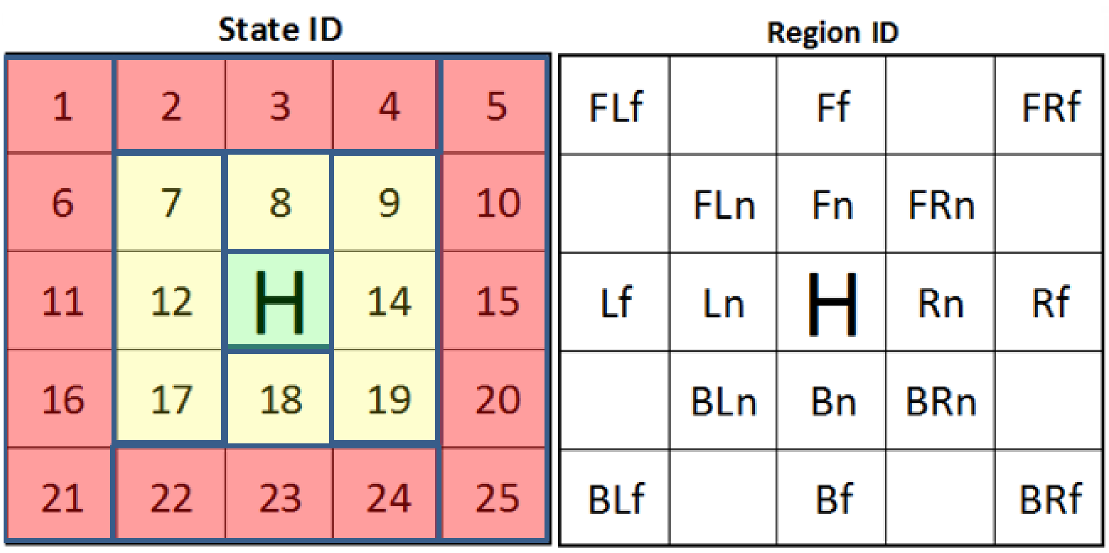


Figure 4.3: State ID and Region ID

Rewards/Punishments

- If the UAV is in the green state 13 then the agent receives a reward of +1 for achieving its objective of stabilizing itself directly above the helipad.
- If the UAV is in any yellow state 7,8,9,12,14,17,18,19 then the agent receives no reward or punishment 0 because this is still a safe zone
- If the UAV is in any red state 1,2,3,4,5,6,10,11,15,16,20, 21,22,23,24,25 the agent receives a punishment -1 due to moving outside the safe zone and into the risky danger zone.

In the unlikely event that the UAV loses visual localisation of the helipad by moving outside of the Cellular Decomposed State Matrix (CDSM), the UAV is forced to land as a safety precaution due to not knowing its current state (State 0) or action to take in order to continue learning. This counts as the end of an episode just like when the reward is found and is represented by a 3 on the Helipad Deviation Graph's Y-Axis signifying being lost. Section 4.6 Figure 4.41 shows preliminary results for the stabilisation performance above the goal position (Helipad). The x-axis represent the incrementing number of state transitions and the y-axis represent the Zone State ID. 0 = Green Target Zone, 1 = Yellow Safe Zone, 2 = Red Danger Zone, 3 = Lost

4.4.1.2 Potential Field Methodology (PFM) Reactive Behaviour

Due to the importance of acquiring consistent visual localisation information to inform the rest of the learning algorithms of their current state, it is crucial that the UAV maintains within visual perspective of the helipad. There are many different reactive architectures that can be applied to the control of robotics navigation, one in particular which immediately appears appropriate for use with UAV's flying in 3D space is the potential field methodology. The motor component of a reactive behaviour can be expressed with a potential field methodology whereby each field consists of both magnitude and direction components similar to vectors. The perpendicular Potential Field Methodology (PFM) shown in Figure 4.4b is implemented for the outer states near the boundaries known as the "danger zone". An attractive potential field (Figure 4.4c) could also be used for the inner states in the safe zone with equal directions to the adjacent outer states but with reduced magnitude to ensure a smooth control gradient preventing overshooting the helipad. However actions executed within the observable state space are to be better controlled using GPFMORL as this has never been attempted before. "*The motor commands of the robot at any position in a potential field correspond to the vector on which the robot is situated. Goals attract, and thus the goals will have vectors pointing towards them; obstacles repulse, and will be surrounded by vectors pointing away*." (Yang and Gu, 2004)

There are 5 primitive Pfields which all have different influences on reactive behaviours

(a) uniform; (b) perpendicular; (c) attractive; (d) repulsive; (e) tangential.

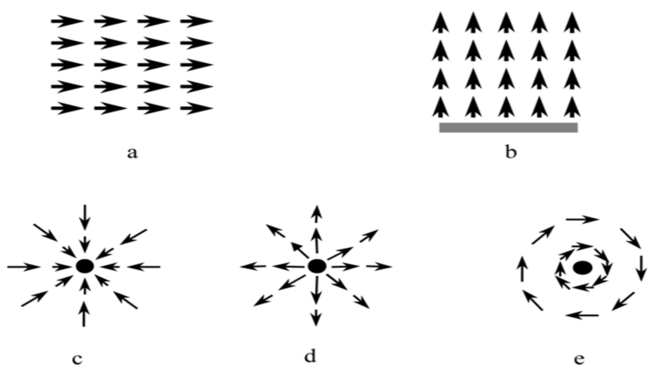


Figure 4.4a,b,c,d,e: Potential Field Representations (*Source*: Arkin, 1989)

a) Uniform: the robot feels the same force wherever it is. Thus, if it is placed in such a field it aligns itself with the "arrows" and moves at a velocity proportional to the length of the arrows. Often used to capture behaviour of go in direction of n-degrees.

b) Perpendicular: e.g. directs robot to or from a wall.

c) Attractive: useful for representing attraction of robot to a light or goal position.

d) Repulsive: opposite of attractive – represents obstacles the robot must avoid. The closer the robot is to the object, the stronger the repulsive force is.

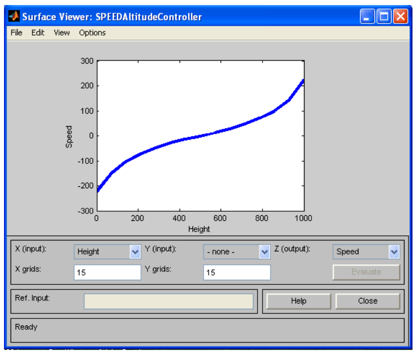
e) Tangential: the field is a tangent round the object. Useful for directing a robot around an obstacle, or having it investigate something.

The type of potential field which is most relevant for forcing the UAV into an observable state space is a combination of the attractive and perpendicular pfields shown as Figure4.4c and in Figure4.4b respectively, extracted from Yang and Gu (2004). This is due to the fact that all these potential fields attempt firstly direct the UAV away from the boundaries so that if the UAV overshoots its intended state, or if the GPFMORL algorithm attempts to try and explore outside its observable state space, this is instantly corrected by automatically forcing the UAV to fly in the opposite direction adjacent to that specific boundary. The UAV should compute the effect of the occupied pfield at every update, with no memory of where it was previously or where it has moved. Although no memory of the UAVs position is necessary for the PFM, it is necessary for the MORL agent to learn. Effectively the PFM reactive behaviour has been implemented to prevent illogical moves from being executed such as flying out of the boundaries of the CDSM. This allows the MORL agent to explore more rational/less risky moves without sacrificing future learning performance by becoming lost.

Equation 11 shown below is the most straight forward formula, where the potential function is constructed as the sum of attractive and repulsive potentials: To ensure a smooth gradient, Uatt(q) is often chosen so that the magnitude of the attractive gradient decreases as the robot approaches the goal location q goal

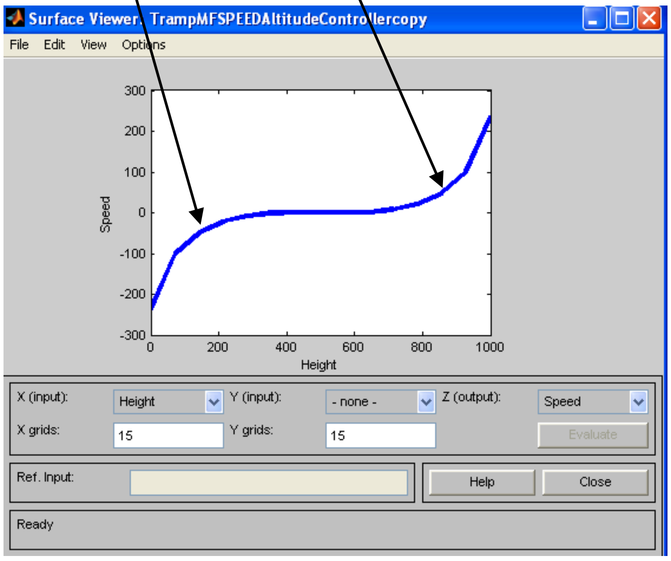
$$U(q) = U_{att}(q) + U_{rep}(q)$$

Equation 4.11 (*Source:* Li-juan, 2001)



X (input): Height
Y (input): - none -
Z (output): Speed
X grids: 15
Y grids: 15
Ref. Input:
Ready

Figure 4.5: Accuracy optimised surface view profile



X (input): Height
Y (input): - none -
Z (output): Speed
X grids: 15
Y grids: 15
Ref. Input:
Ready

Figure 4.6: Speed optimised surface view profile

Above you can see an accuracy surface view profile for creating a non linear gradient change when the UAV reaches critical heights. Using this, the UAV would be adjusting its altitude constantly therefore not allowing any other commands to be executed. The illustration to the right shows a similar technique applied to a fuzzy altitude controller where the constant magnitude of the *Urep* gradient is keeping the UAV away from the wall. Then in the event that the UAV does pass outside the observable state space into an unknown state (state 0) the PMF then changes to an attractive pfiled to pull the UAV back towards the direction of the last known state. Figure 4.6 shows more intense gradient change as the UAV hovers too high or too low. The advantage of such a profile allows delicate and proportional control of the UAVs throttle when the correct height is almost reached and more severe movement applied when the UAV is going to crash into the floor or the roof. The fact that there is a blue horizontal line across most of the graph also means that the correct hovering altitude range has been extended to only allow drastic altitude changes when on a vertical collision course. The advantage of this improved profile is that the altitude controller now does not create a bottleneck of an excessive number of correctional commands which the majority are not required.

4.4.1.3 UAV 1st Person Perspective Heads Up Display (HUD)

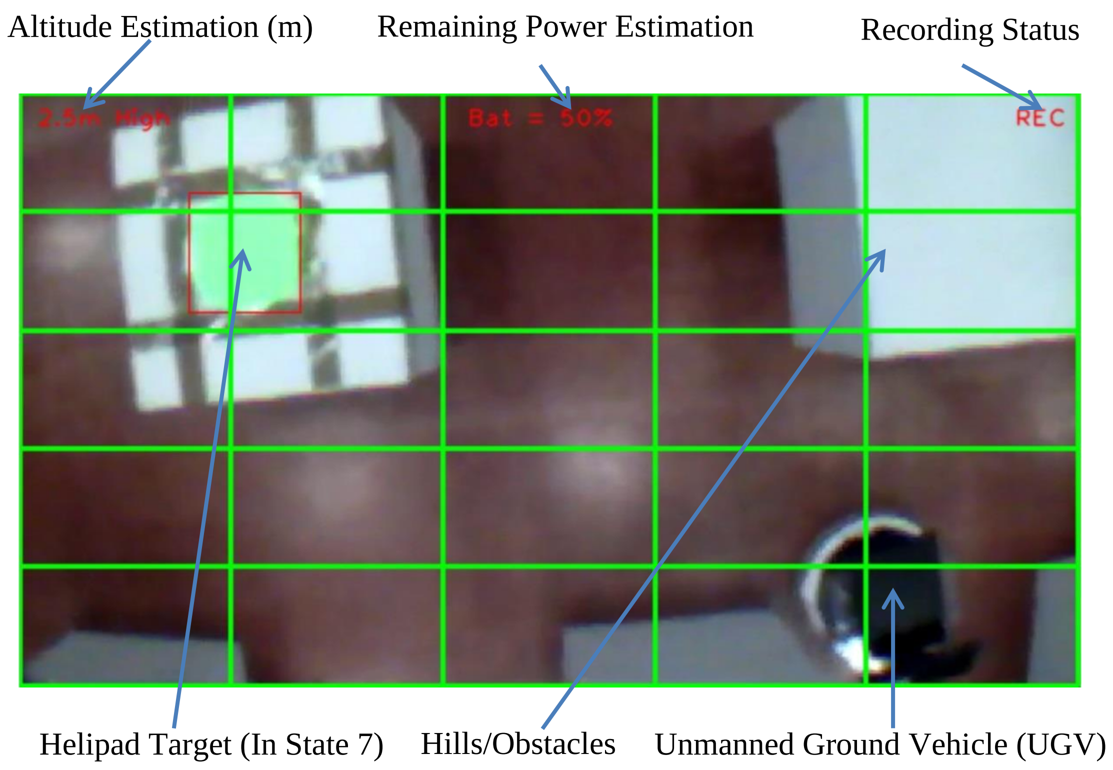

Altitude Estimation (m)
Remaining Power Estimation
Recording Status
2.5m High
Bat = 50%
REC
Helipad Target (In State 7)
Hills/Obstacles
Unmanned Ground Vehicle (UGV)

Figure 4.7: UAV's Heads Up Display (HUD)

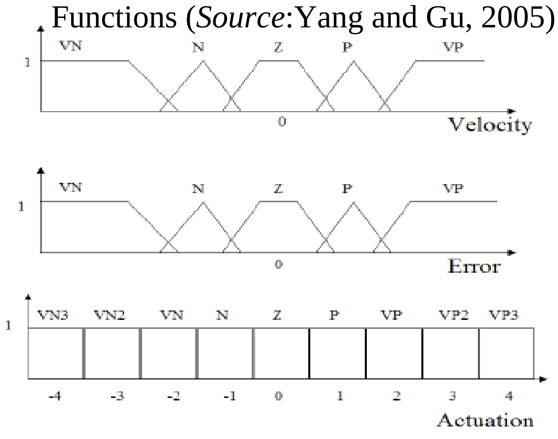
In Addition to the HUD the software writes the recorded video frames to .avi files as well as logging the entire state transition history for each step taken in every trial.

4.4.1.4 Fuzzy Altitude Controller

As the distance between the UAV and the helipad decreases the area of the helipad appears to increase. Likewise when the robot moves away from the helipad its size appears smaller. We intend on using this phenomenon to control the speed of the altitude correction using fuzzy logic. We aim to create an effective control system that gradually slows the UAVs reactions down the nearer it gets to the appropriate hovering height. If the UAV were to move sharply close to the helipad then the helipad would move out the cameras field of view easily. In fact the UAV should move sharply when it is too high in order to get there faster and more efficiently. In the event of the UAV being slightly too high the correction will be made much more slowly to prevent overshooting. In order to increase performance a non linear fuzzy altitude controller has been developed. Fuzzy sets are ideal for this situation because they can be used to model the non linear input area of the helipad when the UAVs altitude changes. A knowledge base can then be constructed consisting of rules determining when the UAV is at the correct hovering height. e.g. “Too Low, Low, ok, High, Too High”

*“This kind of controller has several advantages because it does not need to recalculate parameters when environmental conditions change”(*Yang and Gu, 2005).

Figure 4.8: Fuzzy Membership Functions (*Source*:Yang and Gu, 2005)



Fuzzy linguistic variables for the fuzzy altitude controller.

The inference engine should then generate a control action as a function of state variable values at any given time. This would theoretically ensure that the appropriate motor power and sign (+-) were immediately available to be executed by the UAV whenever required.

The illustration to the right extracted from Yang and Gu (2005) shows an attempt of creating the various fuzzy sets required ie: when it is hovering at the correct altitude (Z), or too high (P), or too low (N), or Very Positive/Negative for extreme situations. Assuming the correct hovering point of 0 on the vertical Z-Axis in 3D space, anything below that point will be negative and anything above will be a positive direction of corrective movements.

Yang and Gu (2005) inspired the logic behind using the non linear area of the helipad to be fuzzified in combination with raw ultra-sonic sensor data and then output as a number between -100 & +100. This output can then be used to directly control the altitudes speed or vertical distance that needs to be adjusted. The + & - values are used for determining the speed that the motors should move in order to move either up or down to control the amounts of thrust used to hover. This approach has advantages against noisy sensor data such as deficient light for the camera to see the helipad properly or if the UAV is landing on textured helipads whose distance may not be accurately estimated due to the sound waves not deflecting as normal.

"Control of a scale model autonomous helicopter during takeoff and landing manoeuvres has proven to be an extremely difficult problem." (Li-juan, 2001)

Li-juan (2001) suggests that the reason for take-off and landing manoeuvres being so difficult is a consequence of two main factors:

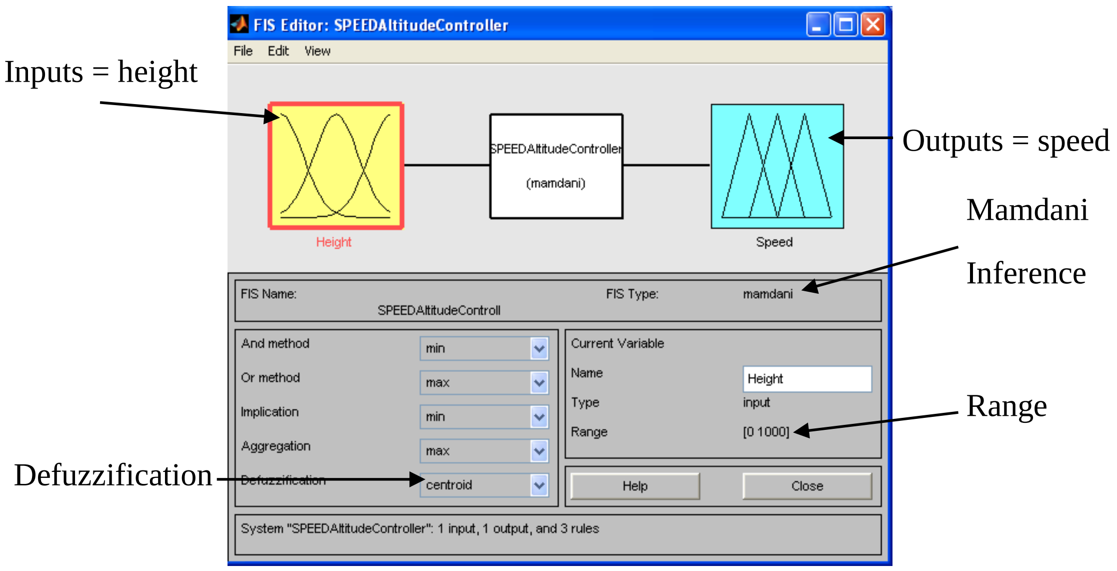
1. The slow time varying and environment dependant nature of the aerodynamic forces encountered during take-off and landing.
2. The high sensitivity of the UAV to collective pitch and roll changes during these manoeuvres.

Whilst the sensitivity of the controls can be manually altered by adjusting several parameters, the aerodynamic forces encountered are much more difficult to compensate for.

"It is no simple matter to model a priori of the aerodynamic effects that occur due to ground effects since they depend upon the nature of the environment in which the helicopter is flying as well as the ambient properties of the air." (Li-juan, 2001)

Due to flight testing being performed in different environments both at home and at universities testing lab, coupled with the factors mentioned above, it would have been very difficult if not impossible to develop my own standardised aerodynamic model of ground effect. Therefore the GPFMORL algorithm is proposed which is capable of learning and compensating for complex aerodynamic effects through trial and error interaction with the real environment. In order to further test the uncertainty handling capabilities we can introduce stochastic random cross winds either using manual tele-operation or via algorithmic approaches. Ie: Generating stochastic cross winds occurring with a probability of 0.2.

The fuzzy altitude controller is responsible for maintaining a steady hovering height roughly half way the height from the floor to the ceiling in the testing environment. The input to the controller is the area of the helipad which are inherently non linear. The non linear inputs are then fuzzified using the Mamdani method to output the speed of the motors and hence the direction of vertical movement as a range from -100 (Down Full Speed) to +100 (Upwards Full Speed. Defuzification method of centroid of area yields the best results obtained from preliminary tests.

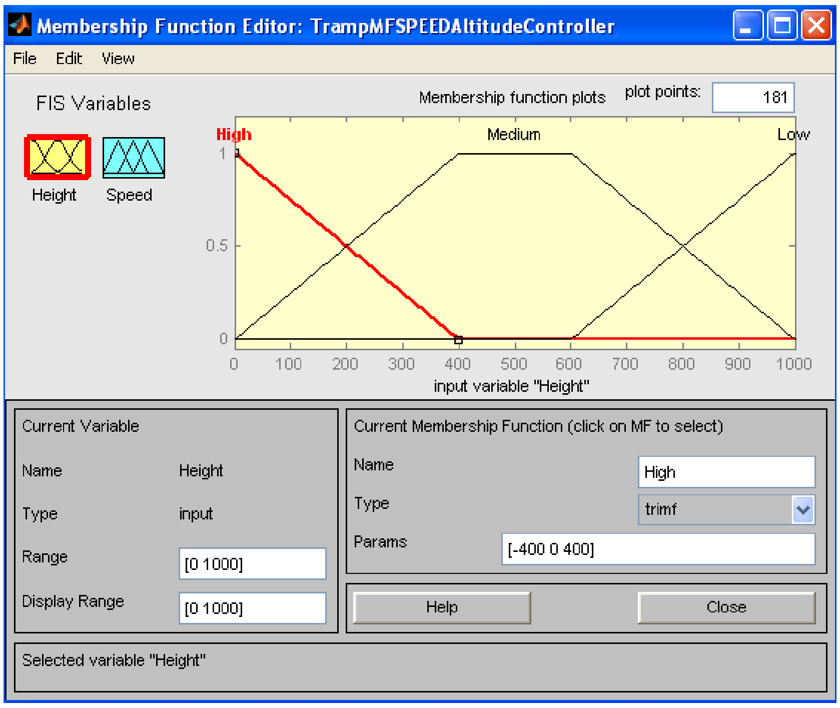


FIS Name: SPEEDAltitudeControll
FIS Type: mamdani
And method: min
Or method: max
Implication: min
Aggregation: max
Defuzzification: centroid
Current Variable
Name: Height
Type: input
Range: [0 1000]
System "SPEEDAltitudeController": 1 input, 1 output, and 3 rules

Figure 4.9: Fuzzy Altitude Controller Structure

Inputs

Fuzzification of the Helipad Area input is achieved by three membership functions, two of type triMF and one of type TrapMF. The first triangular membership function is for when the quadcopter is flying at a high altitude and has the range of (-400 0 400).



plot points: 181
Current Variable
Name: Height
Type: input
Range: [0 1000]
Display Range: [0 1000]
Current Membership Function (click on MF to select)
Name: High
Type: trimf
Params: [-400 0 400]
Selected variable "Height"

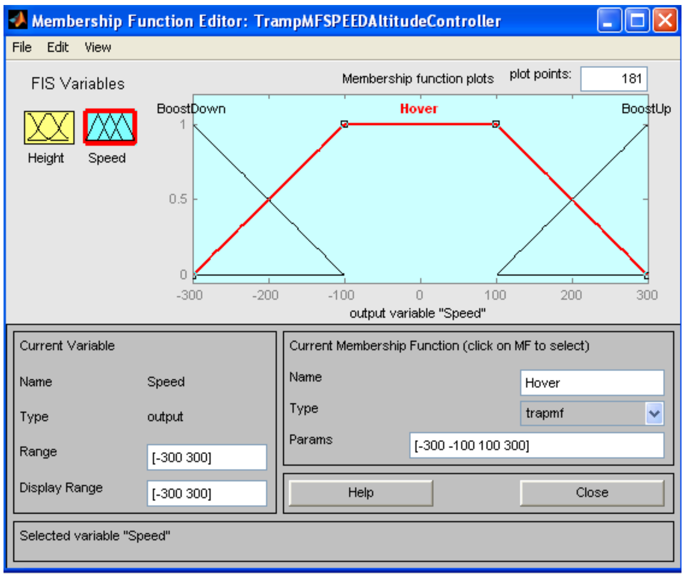
Figure 4.10: Fuzzy Altitude Controller Inputs

The second MF from the left is a trapezium membership function chosen so that the hovering height range is extended to reduce the number of altitude corrections necessary to maintain a reasonable hovering height. The range parameters which are currently set to (0 400 600 1000) can also be modified to allow shorter or longer ranges for the hovering condition to be satisfied depending upon the required flight characteristics. The third and final membership function is a triangularMF used for the scenario where the quadcopter is flying too low and must increase its throttle to achieve higher altitude. The membership function "Low" has the parameters (600 1000 1297) and has an identical shape to the "High" triangular membership function. Overlapping of membership functions is essential to ensure fuzzy logic inference is correctly applied.

<u>Outputs</u>

The range of the output is set to -300 300 although an output of -100 to +100 is only ever used. Extending the range to three times its limit resulted in more responsive control of the motors providing validation is used to prevent invalid commands being sent to the UAV's motors. The membership functions are arranged as before using a combination of TriMF and TrapMF functions with the parameters as described below.

Boost Down = -540 -300 -100	→TriMF
Hover = -300 -100 100 300	→TrapMF
Boost Up = 100 300 540	→TriMF


Membership Function Editor: TrampMFSPEEDAltitudeController

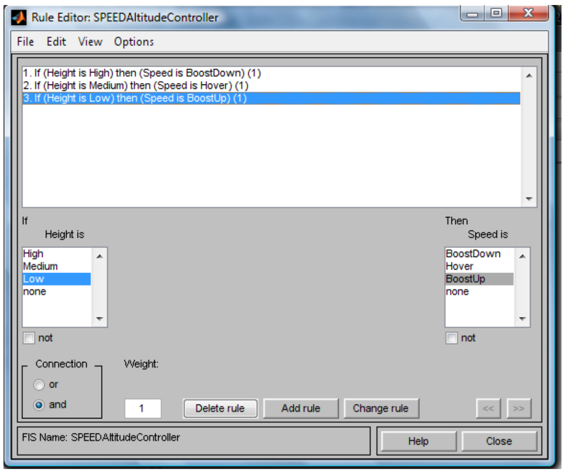
Current Variable
Name: Speed
Type: output
Range: [-300 300]
Display Range: [-300 300]

Current Membership Function (click on MF to select)
Name: Hover
Type: trapmf
Params: [-300 -100 100 300]

Selected variable "Speed"

Figure 4.11: Fuzzy Altitude Controller

Rule Base



1. If (Height is High) then (Speed is BoostDown) (1)
2. If (Height is Medium) then (Speed is Hover) (1)
3. If (Height is Low) then (Speed is BoostUp) (1)

If Height is: High, Medium, [✓ Low], none — not

Then Speed is: BoostDown, Hover, [✓ BoostUp], none — not

Connection: or [✓ and]

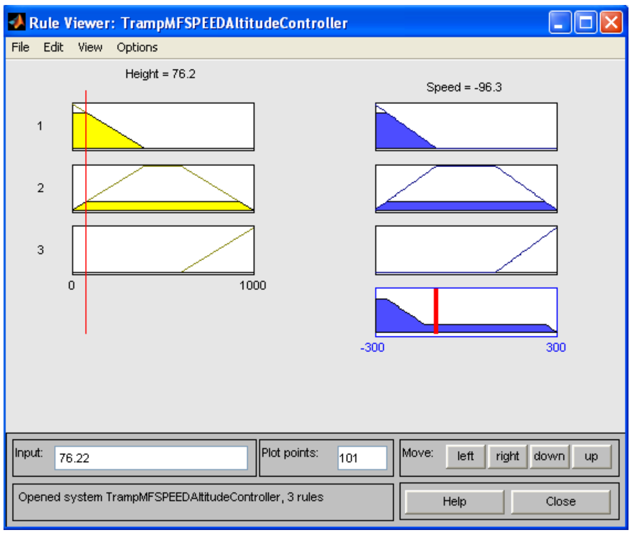
Weight: 1

FIS Name: SPEEDAltitudeController

Figure 4.12: Fuzzy Logic Rule Base Editor

The rule editor allows the altitude measurements to be translated into control actions to be performed by the various motors connected to the UAV. The altitude input is separated into three intuitive height categories of High, Medium and Low by selecting appropriate parameters for each membership function. Depending upon the current height category, the output action will be determined which allows the quadcopter to correct its altitude to remain at a constant hovering height. The output BoostUp will be initiated if the quadcopter flies too close to the floor or BoostDown when dangerously close to the ceiling. The output Hover prevents any altitude corrections from being made due to the inputs stating that it is currently flying at a medium height approximately chest height.

Rule View



Height = 76.2

Speed = -96.3

Input: 76.22

Plot points: 101

Opened system TrampMFSPEEDAltitudeController, 3 rules

Figure 4.13: Fuzzy Altitude Controller Inputs

The illustration to the left shows the effect on output speed as the height input is altered. The example shows a scenario where the quadcopter is flying dangerously high with an input of approximately 76. The controller responds by instructing the motors to rotate slower therefore moving in the negative direction at approximately- 96% full power.

Although the range of the output is set to -300 to +300, the actual output should never exceed +-100. This is ensured by validation however the input values should never be such that would cause invalid outputs to be generated. The example to the right shows the helicopter at slightly above hovering position therefore correction is a low 5%

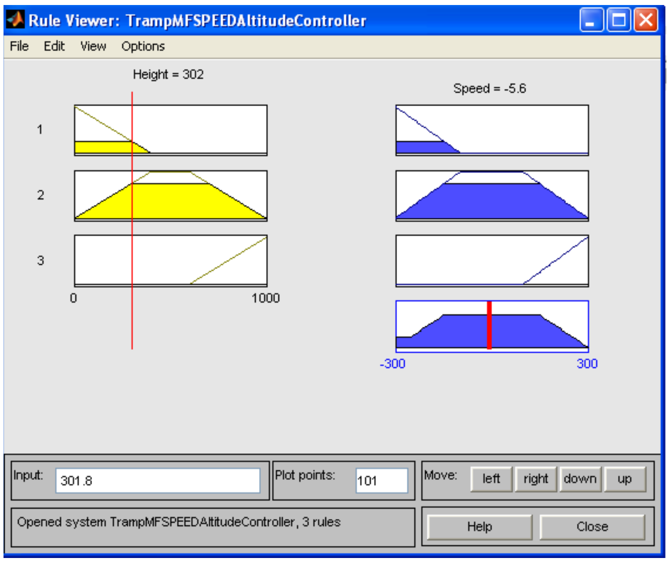


Figure 4.14: Fuzzy Rule View – Slightly too HIGH

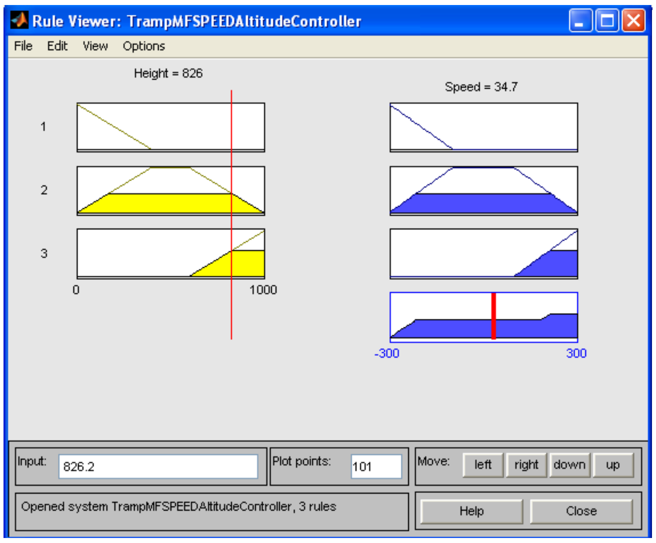


Figure 4.15: Fuzzy Rule View – Slightly too LOW

The Illustration to the left shows the effect of descending to an altitude that is considered too low ie: 826. The controller now decides that the direction the UAV must travel is now positive (UP) and that the speed of correction should be slightly faster than the previous occasion at approximately 35%

Reactive Behaviour (Fuzzy Logic Control)

Pre-processing

The helipad has a constant area when the quadcopters altitude remains constant; when the altitude changes so does the area perceived by the camera. For example when the quadcopter is very close to the floor the helipad will appear larger than if the quadcopter was flying at high altitude. Unfortunately the area returns inversely proportional values that appear to be opposite to what is expected. For example, when the quadcopter is flying low the values are small and when the quadcopter is flying higher the values become larger. In order to convert this raw data into more intuitive altitude knowledge we have subtracted the raw area value from 800 to allow a more realistic output to be displayed i.e. Height. This value of 800 was acquired experimentally and, if required, it could be modified to allow even more realistic measurement of actual altitude in centimetre units.

4.4.1.5 Multi-Modal Fuzzy Translational Velocity Controller

The translational velocities for each state transition were originally hard coded to all be of equal speed (+1.0). However, this proved not to be the most effective way of control as overshooting and undershooting of state transitions became problematic. In the situation where the UAV was far from the goal and HIGH up it would make more sense to increase the speed to reach the next state as there is more room for error at higher altitudes. However if the altitude is low then any slight movement could result in overshooting the state so its movements should be gentle by reducing the translational velocities accordingly. The design of a multi modal fuzzy controller allows a more smooth control of translational velocities from state to state by using the ultrasonic altitude estimation along with the visual perceived distance to the goal to output speeds from 0 to 1.6

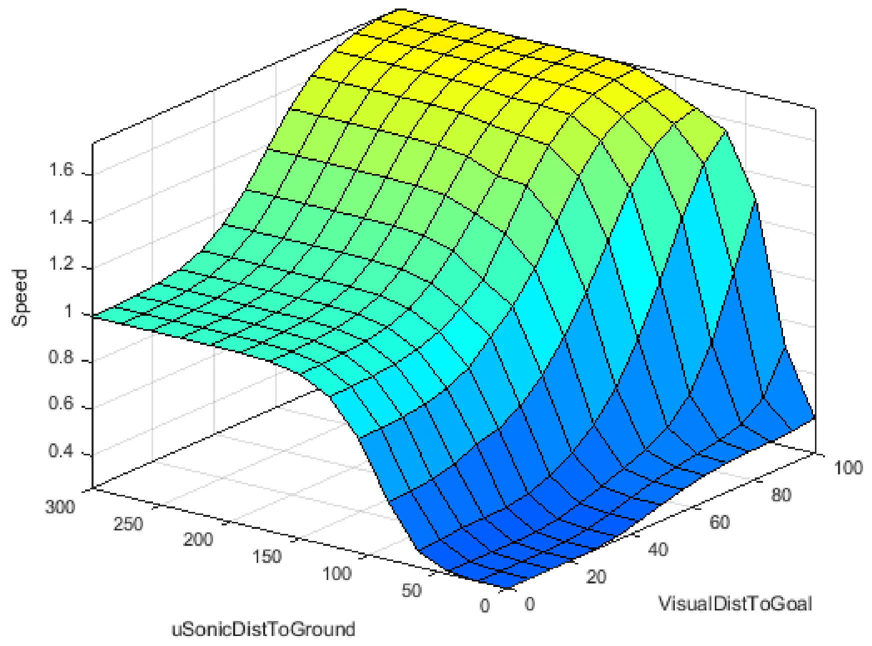
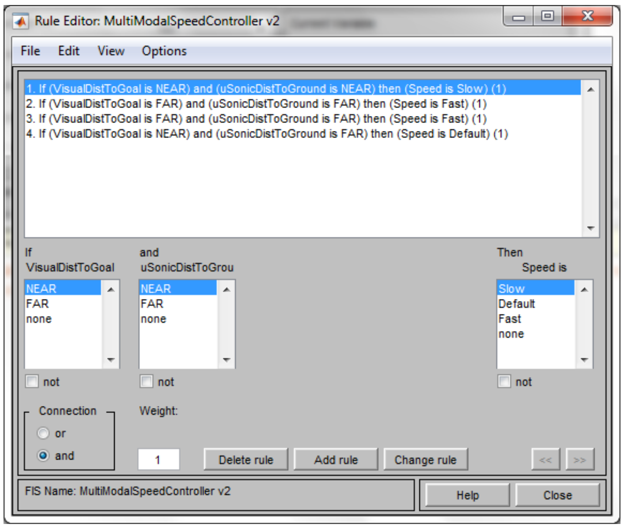


Figure 4.16: Multi-Modal Fuzzy Translational Velocity Controller Surface View



1. If (VisualDistToGoal is NEAR) and (uSonicDistToGround is NEAR) then (Speed is Slow) (1)
2. If (VisualDistToGoal is FAR) and (uSonicDistToGround is FAR) then (Speed is Fast) (1)
3. If (VisualDistToGoal is FAR) and (uSonicDistToGround is FAR) then (Speed is Fast) (1)
4. If (VisualDistToGoal is NEAR) and (uSonicDistToGround is FAR) then (Speed is Default) (1)

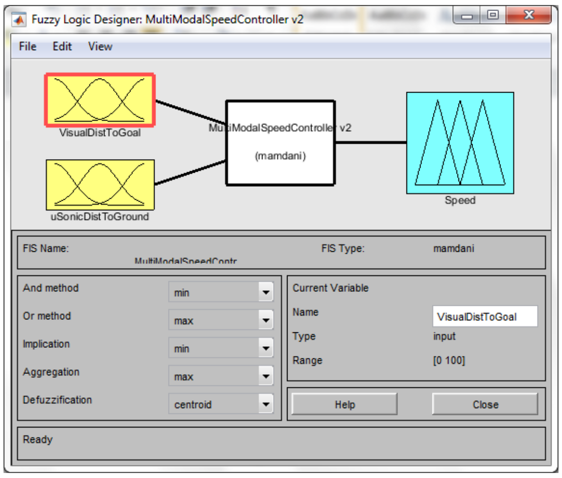
If VisualDistToGoal	and uSonicDistToGrou	Then Speed is
NEAR	NEAR	Slow
FAR	FAR	Default
none	none	Fast
		none
not	not	not

Connection: or [✓ and]

Weight: 1

FIS Name: MultiModalSpeedController v2

Figure 4.17: Translational Velocity Controller – Fuzzy Rules


Fuzzy Logic Designer: MultiModalSpeedController v2
File
Edit
View
VisualDistToGoal
uSonicDistToGround
MultiModalSpeedController v2
(mamdani)
Speed
Help
Close

FIS Name: MultiModalSpeedContr

FIS Type: mamdani

And method: min

Or method: max

Implication: min

Aggregation: max

Defuzzification: centroid

Current Variable

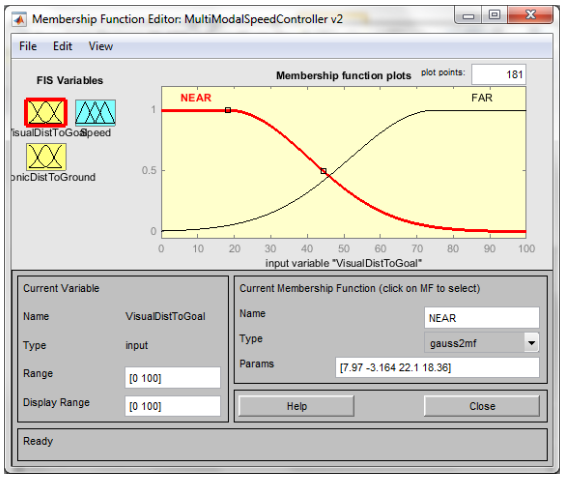
Name: VisualDistToGoal

Type: input

Range: [0 100]

Ready

Figure 4.18: Translational Velocity Controller – Input/Output Fuzzy Logic Design


Membership Function Editor: MultiModalSpeedController v2
File
Edit
View
FIS Variables
isualDistToGoal
Speed
onicDistToGround
Membership function plots
NEAR
FAR
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
input variable "VisualDistToGoal"
Help
Close

plot points: 181

Current Variable

Name: VisualDistToGoal

Type: input

Range: [0 100]

Display Range: [0 100]

Current Membership Function (click on MF to select)

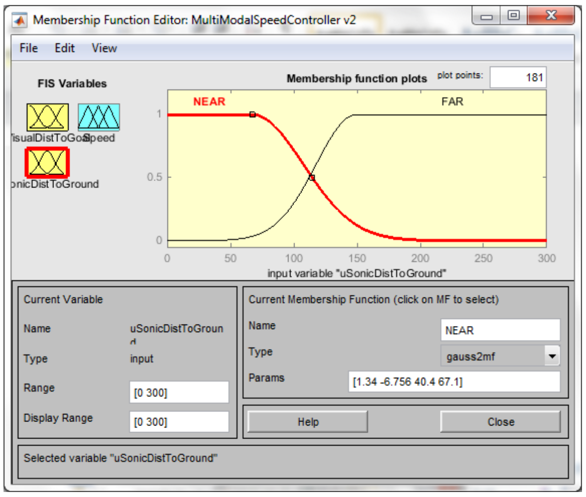
Name: NEAR

Type: gauss2mf

Params: [7.97 -3.164 22.1 18.36]

Ready

Figure 4.19: Translational Velocity Controller – DistanceToGoal Membership Functions


Membership Function Editor: MultiModalSpeedController v2
File
Edit
View
FIS Variables
isualDistToGoal
Speed
onicDistToGround
Membership function plots
plot points:
181
NEAR
FAR
1
0.5
0
0
50
100
150
200
250
300
input variable "uSonicDistToGround"
Help
Close

Current Variable

Name uSonicDistToGround

Type input

Range [0 300]

Display Range [0 300]

Current Membership Function (click on MF to select)

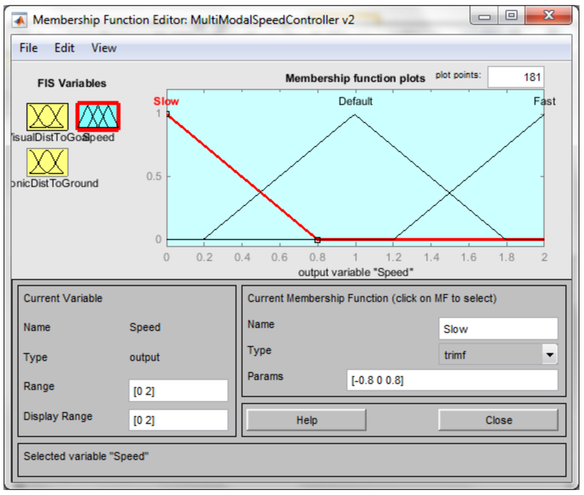
Name NEAR

Type gauss2mf

Params [1.34 -6.756 40.4 67.1]

Selected variable "uSonicDistToGround"

Figure 4.20: Translational Velocity Controller– DistanceToGround Membership Functions



Current Variable

Name Speed

Type output

Range [0 2]

Display Range [0 2]

Current Membership Function (click on MF to select)

Name Slow

Type trimf

Params [-0.8 0 0.8]

Selected variable "Speed"

Figure 4.21: Translational Velocity Controller – Output Speed Membership Functions

4.4.1.6 Discussion

The developed fuzzy controllers improved the learning speed of the algorithm by reducing the stochastic nature of the problem to deterministic state transitions. Rather than one state leading to another random state due to incorrect speed, the controlled speed ensures that the moving from one state to the next state is what is expected. Therefore only external stochastic influences are dealt with rather than internal influences caused by inadequate control.

4.4.2 Automated Solar Powered Environmental Controller

4.4.2.1 Motivation

As a society we find ourselves in a world where its becoming increasingly more difficult to obtain clean food. So much of what is available to is over processed and exposed to toxins. The leading causes of death are all tightly connected to diet and yet the types and qualities of food available to us have not changed. Over the recent years editing the DNA of food crops has become so prevalent that virtually every one of us eats genetically modified food every day, what that does to our planet and bodies is unknown…

In response to this situation there is a movement focused upon growing our own food, on our own terms, that means selecting varieties for taste and quality rather than durability. It means not subjecting our food to long transports, or soaking it in pesticides. It means not fiddling with genetic integrity of the plants we eat.

This clean food movement is demonstrated in several different ways such as as: Urban farming groups and community gardening clubs, Hobbyist Home Growers, Organisations promoting local food and farmers markets (Fig: 4.22)

Figure 4.22: Organisations promoting local food and Hobbyist Home Growers

The ASPEC system was created as a compliment to all of these as well as providing a feasible alternative for industrial/commercial suppliers who inevitably must allow us to have access to healthier hydroponic food. The proposed ASPEC system is automating food production techniques and optimising them by an advanced automation programme where we are developing the world's first solar powered multi-functional embedded system that will increases the efficiency of hydroponic installations without consuming expensive/non-renewable power from the grid. We are using open source hardware and software in conjunction with modern artificial intelligence approaches to produce an affordable smart management system for scalable food production.

for the past 4 years we have been documenting and designing basic structures for growing food. From the larger Twin Channel Independently Controlled ASPEC system to DIY miniature windowsill hydroponic gardens made from recycled water bottles, takeaway containers and jam jars.

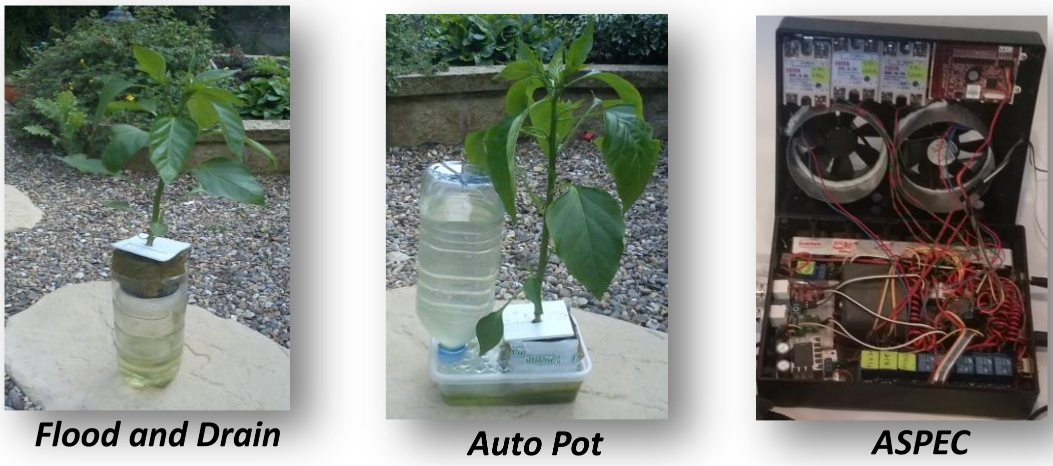


Figure 4.23: Design and Testing of cost effective and advanced hydroponic methods

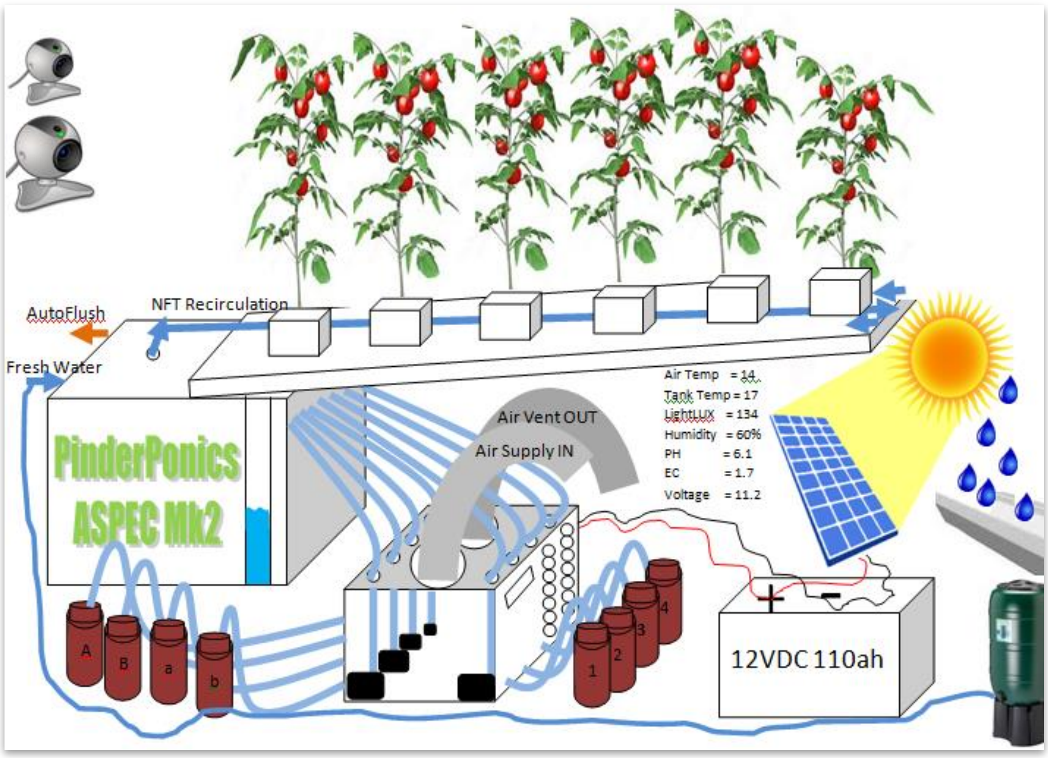


Figure 4.24: ASPEC Nutrient vessel arrangement and NFT crop layout

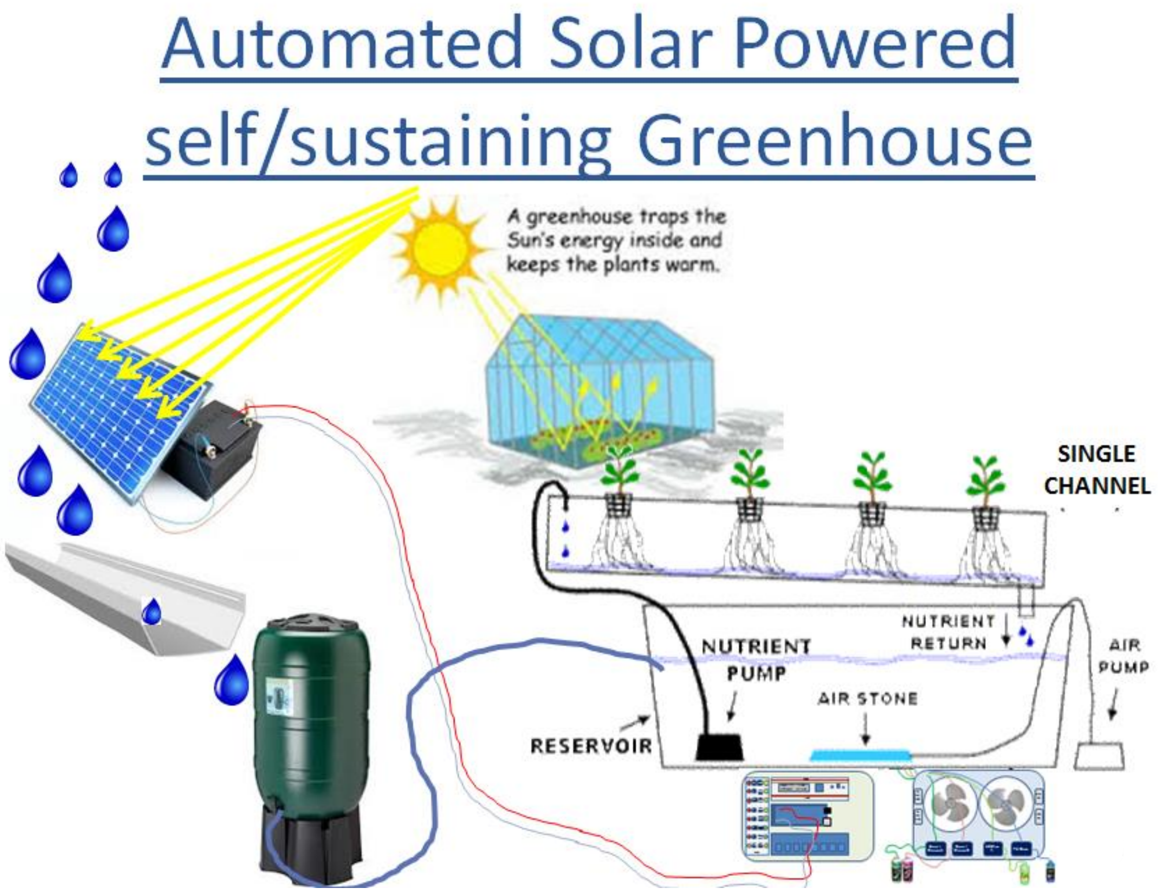

Automated Solar Powered self/sustaining Greenhouse
A greenhouse traps the Sun's energy inside and keeps the plants warm.
SINGLE CHANNEL
NUTRIENT RETURN
NUTRIENT PUMP
AIR PUMP
AIR STONE
RESERVOIR

Figure 4.25: Automated Solar Powered Self-Sustainable Greenhouse

The total world area of glasshouses is estimated to be over 41,000 ha with most of these found in north western Europe and only a small percentage of those are fully automated. Controlled environment agriculture (CEA) is highly productive, conservative of water and land and protective of the environment. There are many types of CEA systems however not every system is cost effective in each location. While the technique of hydroponic culture in the tropics may be quite similar to those in the desert and temperate regions of the world, the greenhouse structures and methods of environmental control can differ greatly.

In conventional soil growth, any excess water, nutrients, vitamins and minerals that haven't been absorbed by the plants roots eventually drain away and are wasted, potentially harming the environment through Eutrophication. Using NFT hydroponics there is a continuous recirculation of nutrient solution that constantly allows the roots to absorb whatever they want, whenever they want, this can lead to yields 3x higher than soil production with the advantage that:

1. Smaller volumes of nutrients can be completely absorbed, much less volume of water is required providing a system that is better for the environment and saves cost on otherwise wasted fertilisers.

2. In addition to the environmental & financial benefits of hydroponics, automation of climatic and nutrient conditions results in an easy to use, hassle free system that optimises the conditions for perfect plant growth.

3. If a solar panel is used to provide all the automation power required to maintain the optimal environmental conditions, then not even humans are required as a resource.

4. The key difference between the limited range of commercial controllers and the one being proposed is that the ASPEC is artificially intelligent and can operate using very low voltages/current e.g. 12VDC 3A. This allows the controllers auxiliary battery to recharge and function carbon free during the day while the plants use natural sunlight to grow during the day while the controller is charged enough to operate all night maintaining the optimum conditions.

Such a system allows the ASPEC controller to be commercially viable in third world countries with power deficiencies and even is a viable option for growing food on the international space station where resources are extremely costly.

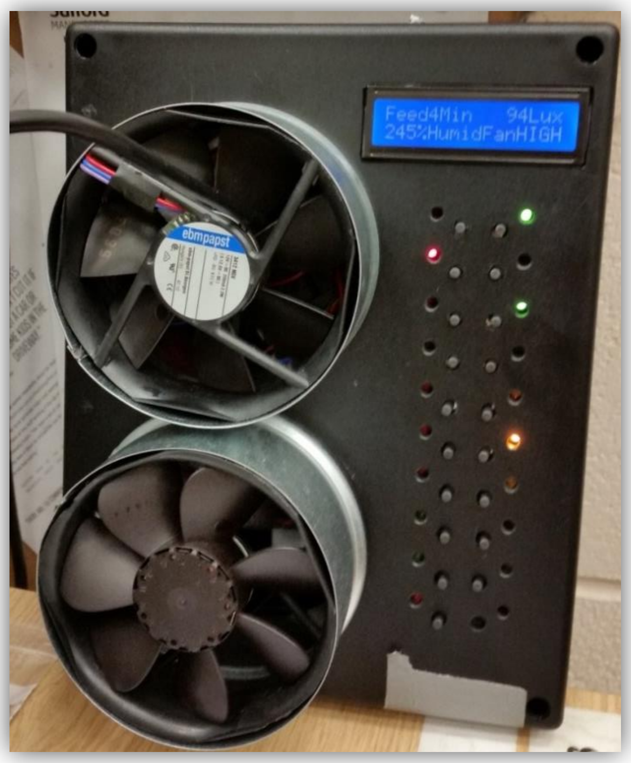


Figure 4.26: ASPEC mk2 with purpose built illuminated micro switch Human Machine Interface

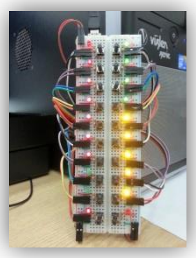


Figure 4.27: Custom HMI (Front View)

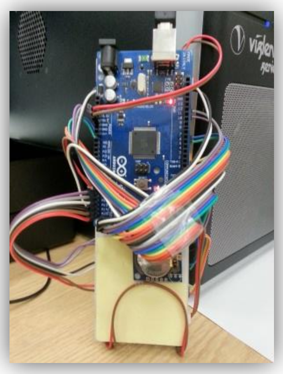
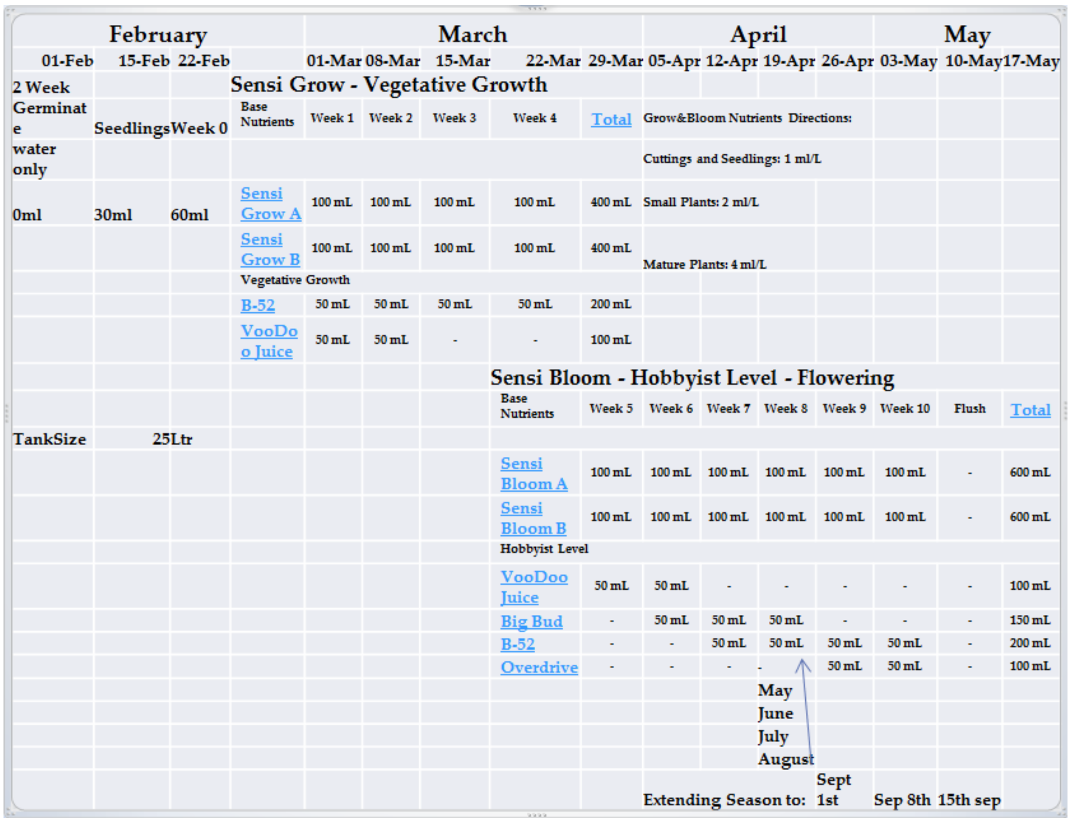


Figure 4.28: Custom HMI (Rear View)



February				March					April				May		
01-Feb	15-Feb	22-Feb		01-Mar	08-Mar	15-Mar	22-Mar	29-Mar	05-Apr	12-Apr	19-Apr	26-Apr	03-May	10-May	17-May
2 Week			Sensi Grow - Vegetative Growth												
Germinate	Seedlings	Week 0	Base Nutrients	Week 1	Week 2	Week 3	Week 4	Total	Grow&Bloom Nutrients Directions:						
water only									Cuttings and Seedlings: 1 ml/L						
0ml	30ml	60ml	Sensi Grow A	100 mL	100 mL	100 mL	100 mL	400 mL	Small Plants: 2 ml/L						
			Sensi Grow B	100 mL	100 mL	100 mL	100 mL	400 mL	Mature Plants: 4 ml/L						
			Vegetative Growth												
			B-52	50 mL	50 mL	50 mL	50 mL	200 mL							
			VooDoo Juice	50 mL	50 mL	-	-	100 mL							
							Sensi Bloom - Hobbyist Level - Flowering								
							Base Nutrients	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Flush	Total
TankSize		25Ltr													
							Sensi Bloom A	100 mL	100 mL	100 mL	100 mL	100 mL	100 mL	-	600 mL
							Sensi Bloom B	100 mL	100 mL	100 mL	100 mL	100 mL	100 mL	-	600 mL
							Hobbyist Level								
							VooDoo Juice	50 mL	50 mL	-	-	-	-	-	100 mL
							Big Bud	-	50 mL	50 mL	50 mL	-	-	-	150 mL
							B-52	-	-	50 mL	50 mL	50 mL	50 mL	-	200 mL
							Overdrive	-	-	-	-	50 mL	50 mL	-	100 mL
											May				
											June				
											July				
											August				
									Extending Season to:			Sept 1st	Sep 8th	15th sep	

Figure 4.29: ASPEC Grow Schedule (by Hesi)

An ASPEC system could be used in your garden, back yard, conservatory or even on your rooftop. Your community can have an even larger system on an unused plot of desolate land even if it does not have a mains supply of water or electricity available.

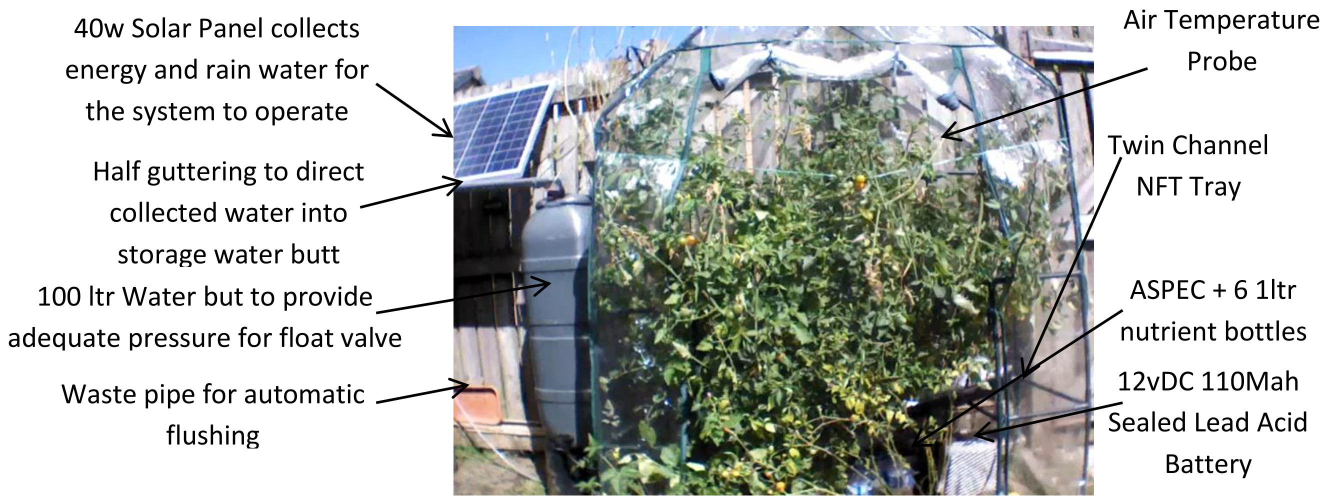

40w Solar Panel collects energy and rain water for the system to operate
Half guttering to direct collected water into storage water butt
100 ltr Water but to provide adequate pressure for float valve
Waste pipe for automatic flushing
Air Temperature Probe
Twin Channel NFT Tray
ASPEC + 6 1ltr nutrient bottles
12vDC 110Mah Sealed Lead Acid Battery

Figure 4.30: ASPEC self-sustaining greenhouse

Commercial application of the proposed system is also a viable option for much larger scale food production at reduced costs to both the environment and the consumer whilst improving the taste, quality, speed and quantity of the food produced. This is achieved by using a hydroponic nutrient film technique (NFT) that has been optimised by introducing a cleverly designed Automated Solar-Programmable Environmental Controller (ASPEC).

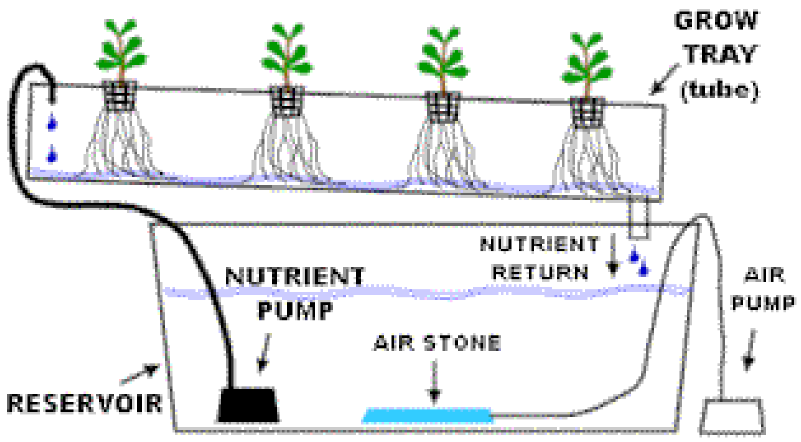


Figure 4.31: Conventional Hydroponic Nutrient Film Technique (NFT)

4.4.2.2 ASPEC Functions and Features

In manually controlled hydroponics, the following numbered tasks heavily rely on a human operator. The bulleted points below each number show how these processes have been automated and optimised resulting in more efficient and effective crop growth.

1. Ensuring the water level of the tank is constantly topped up to a specified level

- Float valve maintains water level at the maximum volume (25ltrs)

2. Turning the air pump on when necessary to cool, mix and oxygenate the nutrient solution.

- The air pump automatically switches on when the temperature of the nutrient solution exceeds 15^{o}c or when nutrients have just been added.

3. Opening greenhouse windows to reduce humidity or switching on a fan to cool the greenhouse air temperature down

- Humidity is maintained between the recommended 80%-95% for clones, 60%-70% for Vegetative and 40% - 60% for flowering plants. The integrated dual fans also ensure the air temperature never exceeds 29^{o}c. In very large greenhouses/polytunnels an external 230VAC more powerful fan can be easily connected to assist the integrated dual fans.

4. Turning on a heater during the early cold months and at night to prevent frostbite .

- An external 230VAC heater can be easily connected to the controller which ensures that temperatures never drop below 10^{o}c

5. Ensuring the NFT pump is always on and does not become blocked

- The filtered NFT pump is activated at 6am until 9pm each day where it then switches on and off intermittently to reduce power consumption at night when there is less chance of the root zone drying out.

6. Once a week a hand held Electrical Conductivity (EC) probe is used to test the nutrient solution concentration.

- The permanently submerged EC probe constantly checks how concentrated the nutrient solution is every day so that the next feed can be reduced, cancelled or increased depending upon the uncertainties of the weather.

7. If the nutrient concentration is too high, the human may drain some of the nutrient solution away to waste and then top up with fresh water to lower the concentration.

- If it has been a cold and cloudy week then the probability that the nutrient solution is too concentrated is very high therefore the scheduled feed may be reduced or even prohibited. In the event of a dangerously high nutrient concentration, 25% of the nutrient solution is automatically flushed away resulting in fresh water mixing to achieve a lower safe concentration of nutrient solution.

8. If the concentration is too low then more nutrients will be added little by little until the desired concentration is achieved for that specific type of plant.

- If the weather has been a very hot and sunny one week then nutrient uptake is increased hence the probability of increasing the recommended feed concentration is high therefore the exact volume of delivered nutrients is calculated based upon the current EC value and the volume of water present in the nutrient reservoir.

9. Approximately every month, the concentration of each individual micronutrient will become unbalanced so the entire contents of the tank should be flushed to

waste and fresh water and nutrients should replace the old unbalanced nutrients to maintain optimal growth conditions.

- Each month after 9pm the NFT pump stops recirculating the nutrient solution while the flush pump empties the contents of the nutrient reservoir. Once empty, the flush pump stops while the nutrient reservoir refills with fresh water. Finally the new nutrients are added to the water, followed by 10 minutes of mixing by the air stone pump, then the NFT pump re-activates to achieve balanced nutrients; mixed to the optimal concentration.

10. Once a week the potential of hydrogen (PH) within the nutrient solution is also tested with a hand held probe to check if the solution is too acidic or too alkaline.

- The permanently submerged PH probe constantly checks how acidic/alkaline the nutrient solution is every day so that it can be automatically altered using PH UP and PH Down correctors.

11. The human operator must painstakingly record a history of all environmental factors such as EC, PH, Air Temp, Tank Temp, Humidity, Date & Time, Dosage Volumes and battery levels in order to predict future discrepancies or potential problems.

- The ASPEC system automatically logs every value recorded so that graphs can be automatically generated at the end of each month which highlights any problems and most importantly at what date and time they occurred. Each grow season, the results from these graphs can be used to further optimise the feeding schedule in attempt to reduce the amount of waste flushed away each month.

12. The hand held probes used are difficult to read whilst partially submerging them in the nutrient solution and often NFT re-circulation must be stopped during probing to prevent nutrient solution from leaking away.

- The developed TFT touch screen Human Machine Interface (HMI) provides an intuitive way of configuring the system on the fly whilst also displaying real-time information, manual functions and system alarms. The data on the HMI can also be accessed wirelessly using radio frequency communication to a PC or tablet.

Alternatively a mobile smart phone can also be connected to monitor serial communication and even be used to re-programme the control software if desired.

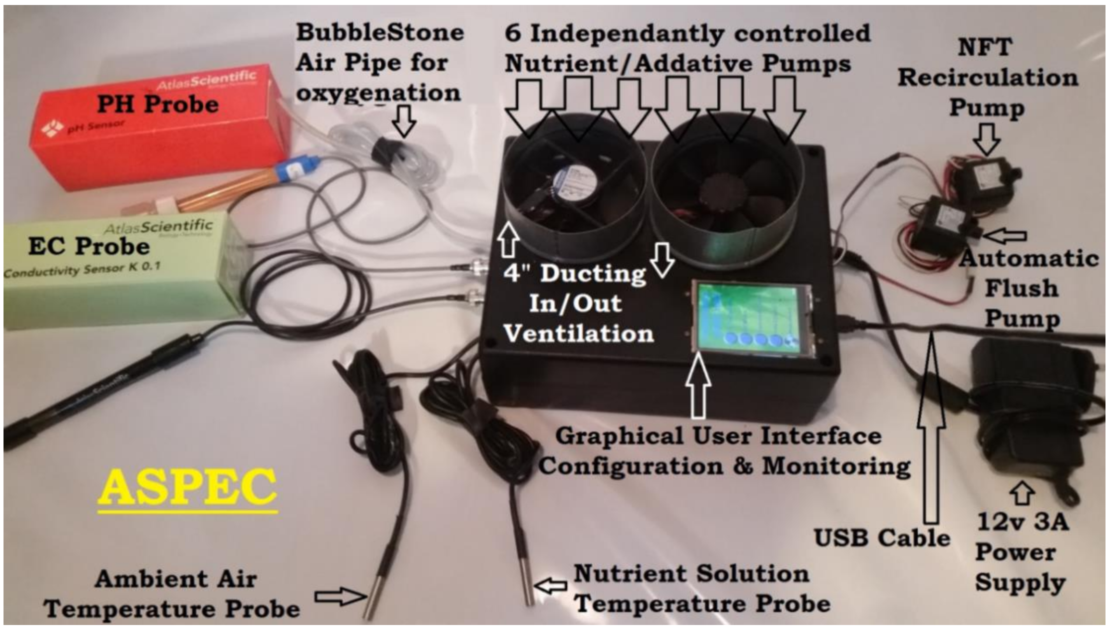

PH Probe
BubbleStone Air Pipe for oxygenation
6 Independantly controlled Nutrient/Addative Pumps
NFT Recirculation Pump
EC Probe
4" Ducting In/Out Ventilation
Automatic Flush Pump
Graphical User Interface Configuration & Monitoring
ASPEC
USB Cable
12v 3A Power Supply
Ambient Air Temperature Probe
Nutrient Solution Temperature Probe

Figure 4.32: ASPEC & Additional complementary hardware layout

It is suggested that this type of system could also benefit from the use of AI algorithms such as GPFMORL taking uncertain weather conditions into account when selecting the correct dosage of nutrients to inject into the nutrient reservoir to ensure optimal crop growth. Over time, the collected data from the probes can be logged by the system therefore allowing the system to also manage other objectives such as minimise number of automatic flushes to reduce wasted nutrients and cost. Alternatively a similar type of system could also be used for other types of medical applications such as medication administration where instead of nutrients being injected into the NFT solution tank, the various types of liquid medications can be accurately delivered into a drinking vessel for a patient to self-medicate whist ensuring the correct dosage and medication is delivered at multiple times of day.

4.4.2.3 Hardware Design

The hardware required to build the working prototype consists of the following components:

- 1 x Arduino Mega Atmel Micro Controller (Atmega2560 MCU)

- 1 x 8ch Relay Board + 1x 1ch Relay Board
- 6 x 12vDC Liquid Pumps
- 12 x 1meter lengths of 11mm1/13mm PVC Pipe
- 1 x 12vDC to 5v Regulator
- 1 x Micro SD Card Data Logging Module
- 1 x Real Time Clock Module with battery backup
- 1 x 3v Air Pump with pipe and air stone
- 1 x EC Sensor Circuit and BNC connector Probe
- 1 x PH Sensor Circuit and BNC connector Probe
- 2 x Waterproof Temperature Sensors
- 2 x 12v 100mm Computer Fans with 4"ducting connectors
- 1 x TFT Touch Screen + Standby Selector Switch
- 3 x 230v Solid State Relays for auxiliary equipment

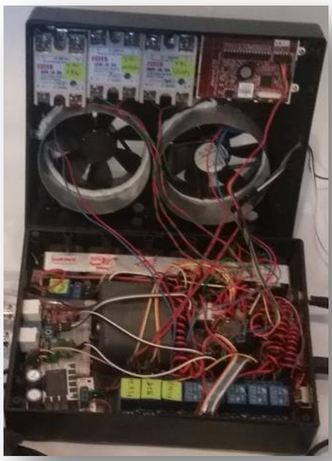


Figure 4.33: ASPEC (Inside View)

4.4.2.4 Touchscreen Human Machine Interface (HMI) Design

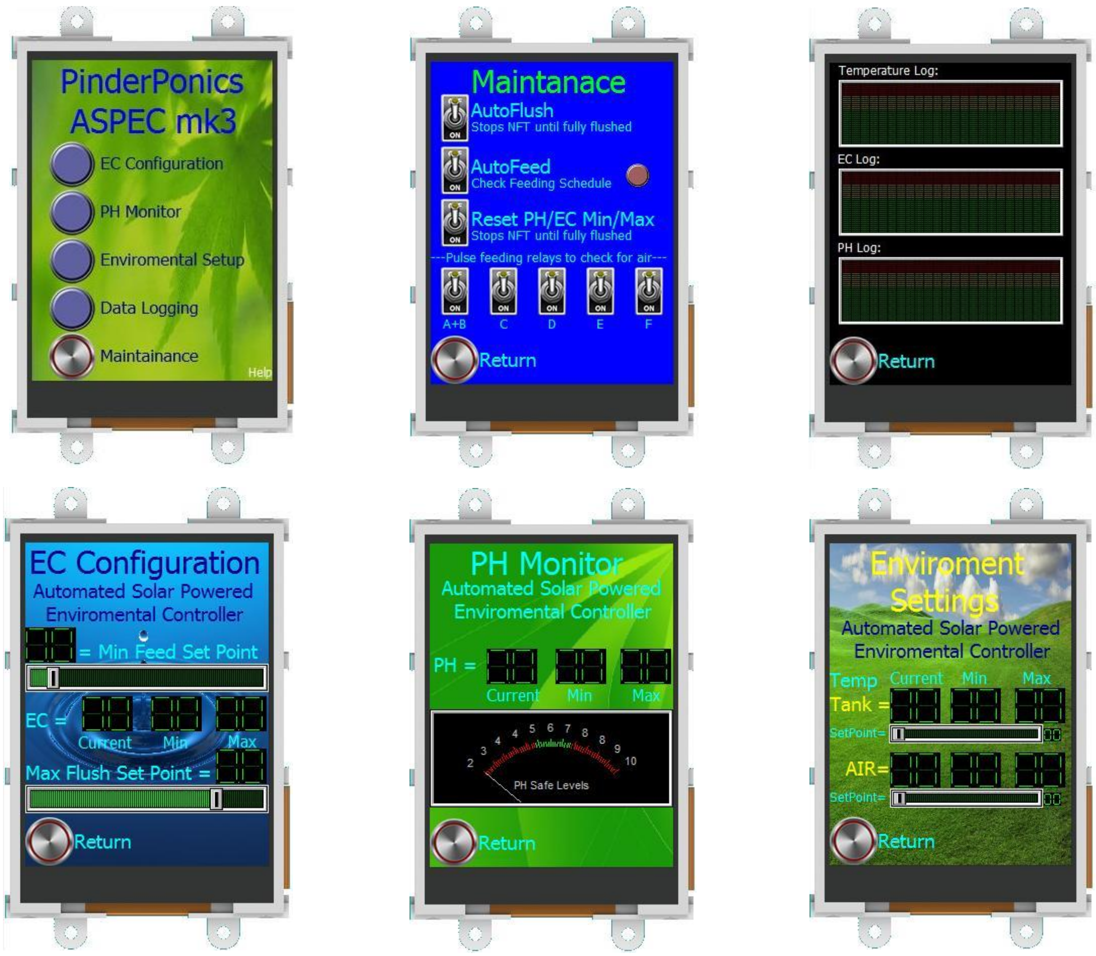


Figure 4.34: ASPEC Touchscreen Human Machine Interface screens

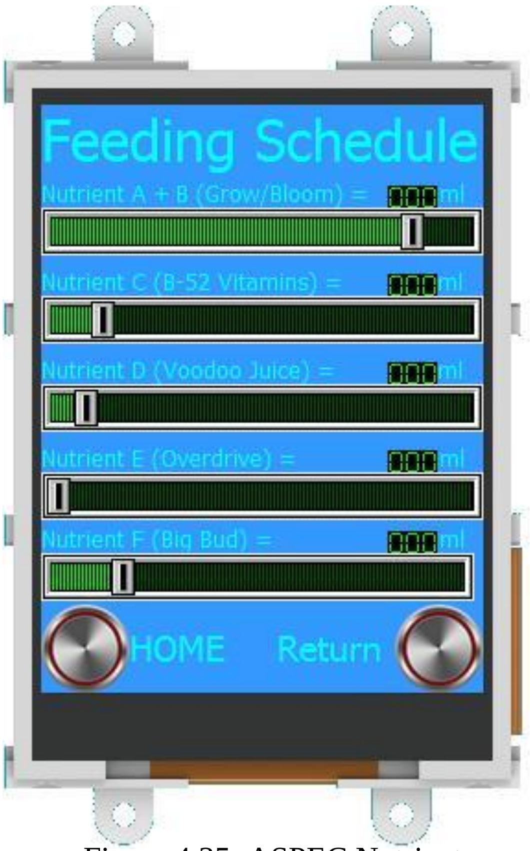

Feeding Schedule
Nutrient A + B (Grow/Bloom) = ml
Nutrient C (B-52 Vitamins) = ml
Nutrient D (Voodoo Juice) = ml
Nutrient E (Overdrive) = ml
Nutrient F (Big Bud) = ml
HOME
Return

Figure 4.35: ASPEC Nutrient Configuration Screen

In addition to the AI control, there also exists the possibility to manually control all aspects of the system.

This allows the systems value functions (learned knowledge) to be initialised at some initial pre-determined values. Although these values can be a good guide to start learning from, they are not optimal and can then be optimised using the proposed GPFMORL methodology should the user select to utilise this intelligent feature of the system.

4.4.2.5 ASPEC State Representation

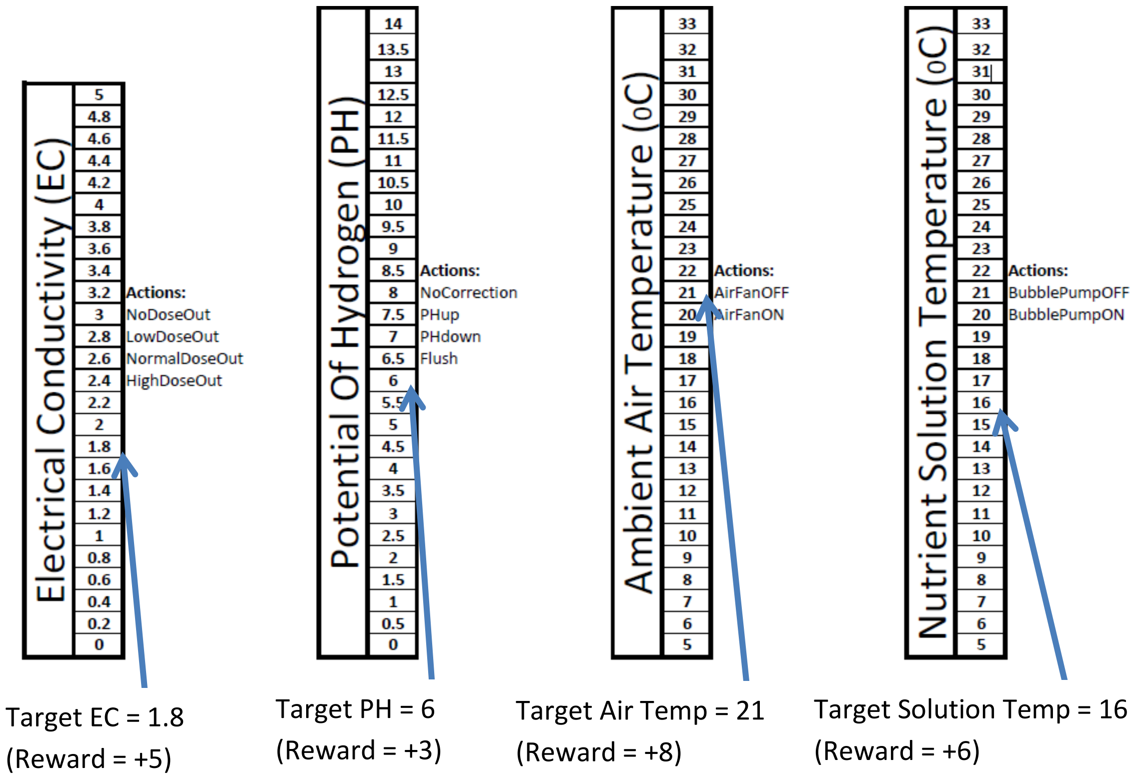


Figure 4.36: ASPEC State Representation

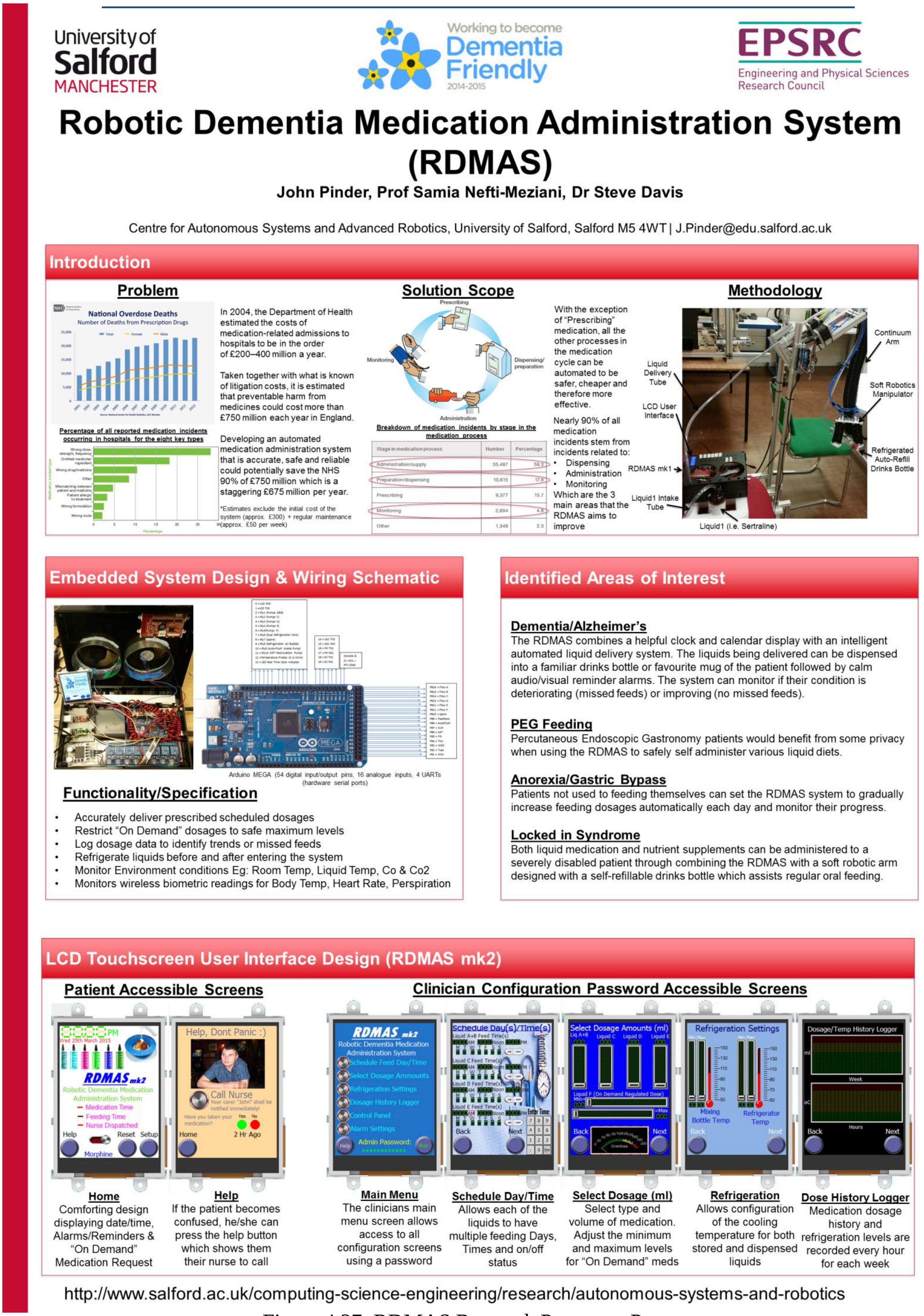
4.4.2.6 Discussion

The ASPEC controller appears to be a feasible solution for controlled environment agriculture (CEA) and improves upon existing technologies by automating the hydroponic process of crop growth. The proposed system design is unique in the fact that it has been designed to operate on very low voltages so can be powered by the sunlight entering your greenhouse. Unlike current CEA methods that use synthetic light to extend the grow period, we propose to optimise conditions based upon whatever the weather is like on that particular day. The cost advantages of not having to use artificial lighting are extremely appealing whilst also benefiting the environment by allowing people to easily grow their own hydroponic food at home.

4.4.3 Robotic Dementia Medication Administration System

4.4.3.1 RDMAS Case Study Poster

See next page P77 for poster illustration (Fig: 4.37 – RDMAS Research Prototype Poster)



University of Salford MANCHESTER

Working to become Dementia Friendly 2014-2015

EPSRC Engineering and Physical Sciences Research Council

Robotic Dementia Medication Administration System (RDMAS)

John Pinder, Prof Samia Nefti-Meziani, Dr Steve Davis

Centre for Autonomous Systems and Advanced Robotics, University of Salford, Salford M5 4WT | J.Pinder@edu.salford.ac.uk

Introduction

Problem

National Overdose Deaths
Number of Deaths from Prescription Drugs

Percentage of all reported medication incidents occurring in hospitals for the eight key types

In 2004, the Department of Health estimated the costs of medication-related admissions to hospitals to be in the order of £200–400 million a year.

Taken together with what is known of litigation costs, it is estimated that preventable harm from medicines could cost more than £750 million each year in England.

Developing an automated medication administration system that is accurate, safe and reliable could potentially save the NHS 90% of £750 million which is a staggering £675 million per year.

*Estimates exclude the initial cost of the system (approx. £300) + regular maintenance (approx. £50 per week)

Solution Scope

Breakdown of medication incidents by stage in the medication process

Stage in medication process	Number	Percentage
Administration/supply	35,487	59.3
Preparation/dispensing	10,615	17.8
Prescribing	9,377	15.7
Monitoring	2,894	4.8
Other	1,349	2.3

With the exception of "Prescribing" medication, all the other processes in the medication cycle can be automated to be safer, cheaper and therefore more effective.

Nearly 90% of all medication incidents stem from incidents related to:

- Dispensing
- Administration
- Monitoring

Which are the 3 main areas that the RDMAS aims to improve

Methodology

Continuum Arm
Liquid Delivery Tube
Soft Robotics Manipulator
LCD User Interface
Refrigerated Auto-Refill Drinks Bottle
RDMAS mk1
Liquid1 Intake Tube
Liquid1 (i.e. Sertraline)

Embedded System Design & Wiring Schematic

Arduino MEGA (54 digital input/output pins, 16 analogue inputs, 4 UARTs (hardware serial ports)

Functionality/Specification

- Accurately deliver prescribed scheduled dosages
- Restrict "On Demand" dosages to safe maximum levels
- Log dosage data to identify trends or missed feeds
- Refrigerate liquids before and after entering the system
- Monitor Environment conditions Eg: Room Temp, Liquid Temp, Co & Co2
- Monitors wireless biometric readings for Body Temp, Heart Rate, Perspiration

Identified Areas of Interest

Dementia/Alzheimer's
The RDMAS combines a helpful clock and calendar display with an intelligent automated liquid delivery system. The liquids being delivered can be dispensed into a familiar drinks bottle or favourite mug of the patient followed by calm audio/visual reminder alarms. The system can monitor if their condition is deteriorating (missed feeds) or improving (no missed feeds).

PEG Feeding
Percutaneous Endoscopic Gastronomy patients would benefit from some privacy when using the RDMAS to safely self administer various liquid diets.

Anorexia/Gastric Bypass
Patients not used to feeding themselves can set the RDMAS system to gradually increase feeding dosages automatically each day and monitor their progress.

Locked in Syndrome
Both liquid medication and nutrient supplements can be administered to a severely disabled patient through combining the RDMAS with a soft robotic arm designed with a self-refillable drinks bottle which assists regular oral feeding.

LCD Touchscreen User Interface Design (RDMAS mk2)

Patient Accessible Screens

Home
Comforting design displaying date/time, Alarms/Reminders & "On Demand" Medication Request

Help
If the patient becomes confused, he/she can press the help button which shows them their nurse to call

Clinician Configuration Password Accessible Screens

Main Menu
The clinicians main menu screen allows access to all configuration screens using a password

Schedule Day/Time
Allows each of the liquids to have multiple feeding Days, Times and on/off status

Select Dosage (ml)
Select type and volume of medication. Adjust the minimum and maximum levels for "On Demand" meds

Refrigeration
Allows configuration of the cooling temperature for both stored and dispensed liquids

Dose History Logger
Medication dosage history and refrigeration levels are recorded every hour for each week

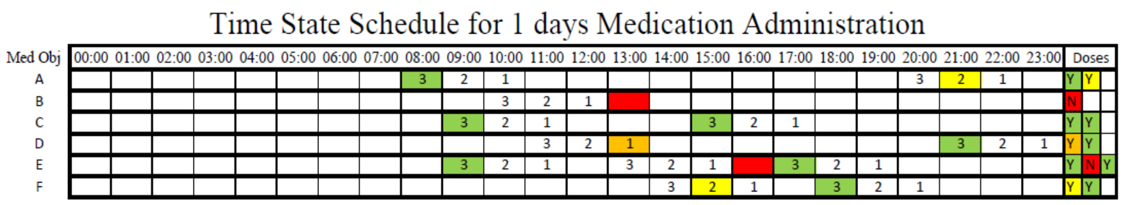
http://www.salford.ac.uk/computing-science-engineering/research/autonomous-systems-and-robotics

Figure 4.37: RDMAS Research Prototype Poster

4.4.3.2 Hardware Design

The hardware design for the proposed RDMAS is very similar in design to the ASPEC being that both applications facilitate the controlled dispensing of liquids either based upon an intelligent schedule or a closed loop feedback system. The RDMAS was integrated into a soft robotic arm capable of moving the dispensed medication vessel towards the patient’s mouth when required.

4.4.3.3 RDMAS State Representation



Time State Schedule for 1 days Medication Administration

Med Obj	00:00	01:00	02:00	03:00	04:00	05:00	06:00	07:00	08:00	09:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00	21:00	22:00	23:00	Doses		
A									3	2	1										3	2	1		Y	Y	
B											3	2	1	🟥											N		
C										3	2	1				3	2	1							Y	Y	
D												3	2	1								3	2	1	Y	Y	
E										3	2	1		3	2	1	🟥	3	2	1					Y	N	Y
F															3	2	1		3	2	1				Y	Y	

Figure 4.38: RDMAS State Representation

1 Day = 1 Simulation Trial

1 week = 7 trials

And each month its a new play so 4plays of 7 trials = 1 month of testing

4.4.3.4 Discussion

It is hoped that the proposed RDMAS system shall be tested with real dementia patients in a controlled environment to fully test its diagnosis capabilities on human subjects. The qualified nurse shall initially set up the system to deliver medications at the prescribed dosage and time of day and ensure that validation has been assigned for risky medications which may not mix well with others. The goal of the system is to deliver multiple medications at the suggested times, then monitor if the medications have actually been administered. This is done so that the system can re-assign medication delivery times to more appropriate times of day if the medication was missed for whatever reason by analysing their daily memory reactions times.

4.5 Empirical Evaluation methods for MORL

MO-Puddleworld (Boyan and Moore, 1995) is a two-dimensional environment, which has previously been used as a single-objective RL bench-mark. The agent starts each episode at a random, non-goal state and has to move to the goal in the top-right corner of the world, while avoiding the puddles. The agent receives its current coordinates as input, and at each step selects between four actions (left, right, up or down) which move it by 0.05 in the de- sired direction. At each step a small amount of gaussian noise (standard deviation 0.01) is also added. The agent's position is bounded by the limits of the world (0,...,1). The reward structure for Puddle world is interesting, as it is effectively a form of scalarisation with fixed weights for the two objectives of reaching the goal quickly and avoiding the puddles. On each step on which the goal is not reached, the agent receives a penalty of −1. An additional penalty is applied when the agent is within a puddle, equal to 400 multiplied by the distance to the nearest edge of the puddle. To convert this problem to a multiobjective task, the two penalties are presented as separate elements of a reward vector (omitting the multiplication by 400, as it is no longer relevant).

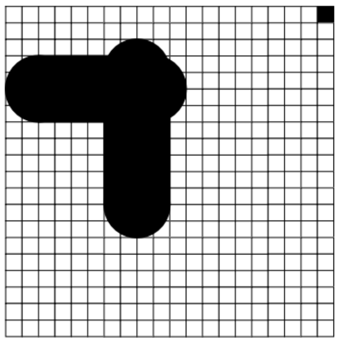


Figure 4.39: MO Puddleworld (*Source:* Boyan and Moore, 1995)

The Buridan's ass problem (Chen and Hu, 2010) with three objectives is shown in Figure 4.40 shows the donkey is in the centre square of the 3x3grid. There are food piles on the diagonally opposite squares. The food is visible only from the neighbouring squares in the eight directions. If the donkey moves away from the neighbouring square of a food piles, there is a certain probability Pstolen 0.9 with which the food is stolen. Food regenerates once every N appear 10 time steps. If the donkey chooses to stay at a square with food, then it eats food. Otherwise, if the time since the donkey last ate food is more than max 9 T, it will feel hungry. The donkey has to strike a compromise between minimizing the three different costs: hunger, food-stolen, and walking. The environment, actions and objectives are similar to the UAV scenario where food is replaced by the helipad goal.

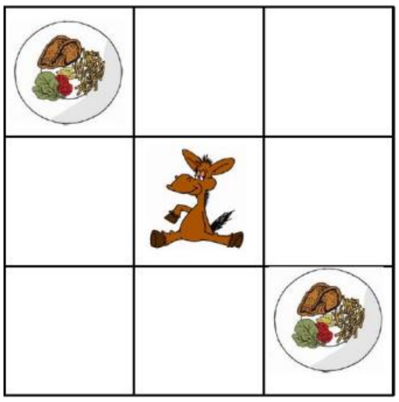


Figure 4.40: Buridan's Ass (*Source:* Chen and Hu, 2010)

4.6 PRELIMINARY RESULTS

The chosen methodology for maintaining visual localisation is potential field methodology (PFM) which has been selected due to its intuitive design and real time response for reactive behaviour. The PFM is used to keep the UAV away from the boundaries and within the observable state space. The magnitude of the PF can be automatically adjusted depending upon the output of the fuzzy translational velocity controller. Implementation of this methodology allows a baseline of results to be obtained that show how well the UAV is able to maintain its position within the desired state. Storing a history of previous states visited allows a graph to be plotted showing the deviation from the helipad in number of states over time ie: when UAV is in the safe zone the deviation shall be 1 and when in any danger zone the deviation shall be 2. This graph not only shows how long the UAV took to reach the helipad, but also how long it maintained its position there before naturally drifting away to another state, from this data we can determine a performance measure which can be compared with higher level control algorithms.

PFM is intended to create the action sets necessary for avoiding the situation where the UAV flies too far away from the helipad and looses visual localisation of the environment. It is therefore intended that the RL agent intelligently decides which actions to take when in the safe inner zone of the environment and the PFM controls the reactive movements in the danger zone to always keep the helipad in the UAV's field of view to allow learning to occur. Search for the Helipad (Moving towards the helipad using Forward Camera Improving upon the PFM, a similar reward scheme shall be implemented for a RL approach that is designed to keep the helipad's landing box in view by occupying the lower-middle state 23 in order to perceive the Landing box on the floor by flying low until it gets close enough to switch the camera to downwards perspective and then increase its altitude to acquire a better view of the helipad for localised landing or following. While the GPFRL algorithm is capable of dealing with discrete state and actions spaces, it reaches its full potential when using continuous states such as when dealing with the uncertainty of sensor data. For example the UAV must know how far it is away from the helipad on the floor in order to know when it should switch to the PFM, therefore the only available sensor is the camera which can ascertain the distance by calculating the area of any side of the landing box which contains the helipad marker. When detecting the area of the landing box from a forward view we are actually detecting the walls of the box due to being situated at the correct plane for successful image processing. The area of the landing box may be noisy due to ambient light levels or light reflections therefore the Probabilistic Fuzzy Inference System should be able to compensate for this and decide whether the UAV is either near or far from the landing box. Furthermore our hypothesis is that increasing the action set and state spaces to a 10x10 grid with 8 possible actions would result in a much more continuous movement of the UAV but would inherently increase computational time due to doubling the action agents from 4 to 8 and quadrupling the number of possible states encountered from 25 to 100.

Helipad Deviation Comparison Graph of Manual vs Automatic Control

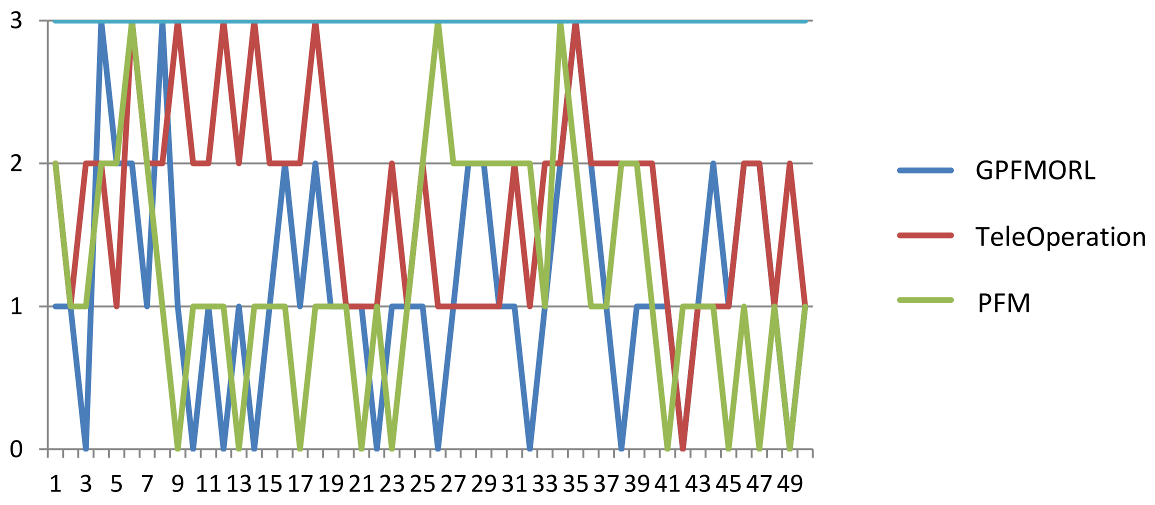


Figure 4.41: UAV Helipad Deviation Comparison Graph

Helipad Deviation Y-Axis: 0 = Helipad, 1 = Safe Zone, 2 = Danger Zone, 3 = Lost

Step Numbers X-Axis = Each time the UAV transitions/moves into a new state

Asomptotes (Flat Ridges) = Where the UAV moved states but not zones. Caused by human error (Teleoperation) or aerodynamic affects (PFM) that cannot be predicted due to no automatic learning occurring. Using GPFMORL we hypothesise that these “Flat Ridges” should reduce over time as the agents learn better policies to execute in similar states.

4.7 Discussion

From the graph in figure 4.41 you can see the red line which represents teleoperated control by a human operator to attempt to fly above the helipad in preparation for landing. In this test the quadcopter started in the safe zone (1) then quickly became lost many times by flying outside the perceivable state space. After many transitions between the safe zone and the danger zone, the quadcopter finally finds the helipad (0) for a brief second before flying back into the safe zone. This is considered to be the worst performing solution to the problem mainly due to delayed reaction times from the human operator. The green line on the graph represents Assistive teleoperation which uses the PFM to try and prevent the quadcopter losing visual localisation and becoming lost. This improved method only allows the quadcopter to get lost 3 times and the helipad goal is found many more times than teleoperation alone at 10 times. Finally the GPFMORL approach gets lost at the start of the trial a few times just as much as assisted teleoperation, but then after becoming lost twice, it learns to never get lost again and reaches the goal just one time fewer than the assistive teleoperated control.

4.8 Conclusion

The process of localisation is no simple task for a robot and is a prerequisite for most navigation tasks. It is possible to achieve localisation even when there may only be one landmark visible in the captured image providing the hardware and software capabilities allow this. Range finders and gyroscopes alone are not enough to solve this complicated task due to their accuracy limitations therefore other sensors such as a camera must be used. Localisation is necessary for correcting the robots inaccurate positioning incurred through drift. The camera equipped on the quadcopter, whilst the information is rich it is still limited mainly due to its field of view and the absence of other on-board sensors. The quadcopter is incapable of carrying any other relatively heavy sensors however a more suitable wide-angle lens could be retro fitted the existing lens. Although the use of more sophisticated sensors would improve the performance of the system the added weight would require a much more expensive and powerfull hexcopter or octocopter. Using such a large and expensive quadcopter would go against the expectations of this thesis which is to alow cheap, non intrusive drones to be used in everyday applications.

The reactive paradigm of potential fields can be interpreted for both positional corrections to stabilise over a central (H) goal or for vector summation to achieve both the direction and magnitude of the altitude potential fields used for the fuzzy translational velocity controller. Potential fields provide a comprehendible visualisation of an abstract theory allowing the development stage to be successfully completed methodically. The Helipad deviation graphs undoubtedly show that although the PFM is suitable for preventing out of bounds moves, sometimes its actions are not optimal when there are stochastic environmental factors such as ground affect and unpredictable cross winds. While the repulsive force from the boundaries is sufficient to allow the RL to observe its current state by ensuring visual localisation, a simple attractive force around the helipad would often lead to overshooting or even never coming close to the helipad due to inconsistent winds. This is one of the reasons for combining these methodologies in conjunction with the GPFMORL approach to multi objective optimisation for sequential decision making processes.

Chapter 5

Experiments

5.1 Introduction

This chapter considers a number of example algorithms related to the proposed multi objective reinforcement learning approach. In section two, I demonstrate how the random walk grid world is used to evaluate the convergence of the multi objective learning algorithms while comparing its performance with two classic temporal difference algorithms Q-Learning and SARSA and their multi objective extensions referred to as MOQ and MOSARSA. In addition, a third objective of hills/obstacles is added to a highly stochastic version of the grid world called Windy Hill World. The purpose of this is to evaluate the influence of environment stochasticity on the performance of our proposed GPFMORL algorithm in hope that it shall be robust against external aerodynamic forces in real world testing using a quadcopter. In section 3 we test our optimised GPFMORL algorithm on a real AR-Drone 2.0 quadcopter and compare classical approaches with artificially intelligent approaches for real world applications such as autonomous landing and search and rescue. Quadcopter modifications and related experimental results are shown along with results produced from the multi modal fuzzy altitude controller verifying its capabilities in the presence of uncertainty. Section four describes two similar prototypes that have been created for different real world applications where the proposed algorithm could be used to improve the effectiveness of each system beyond the capabilities of any conventional human operator. Finally section five summarises the contents of this chapter of the thesis.

5.2 Decision Making Simulated Experiments

5.2.1 Random Walk Problem (1st Objective Helipad)

The random walk problem is a mathematical formulisation of a trajectory which consists of taking random successive steps. The random walk analysis provides results that have been applied to computer science, physics, ecology, psychology, and several other fields as a fundamental model for random processes in time.

The purpose of this experiment is to provide a mean by which to evaluate the update rule, the stability and the convergence of our proposed GPFMORL algorithm, and compare it with two other multi objective extensions of classical reinforcement learning algorithms, MOSARSA and MOQ-Learning. Figure 5.1 Shows an example of a 10x10 grid world testing environment in which the learning agents act.

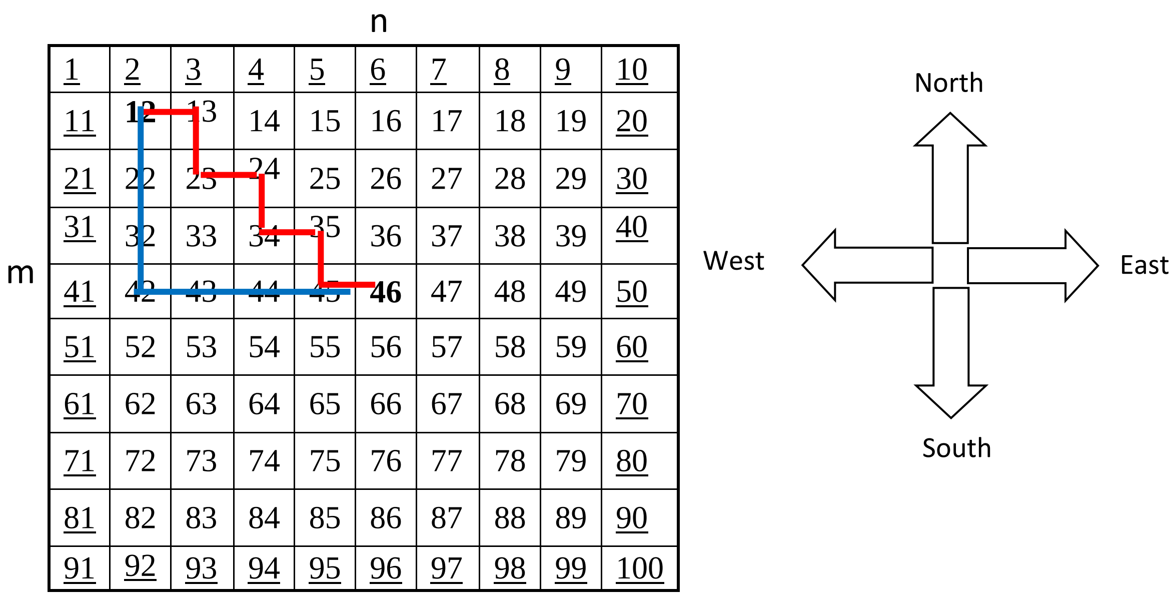


Figure 5.1: Random Walk Grid World Environment and Action Set

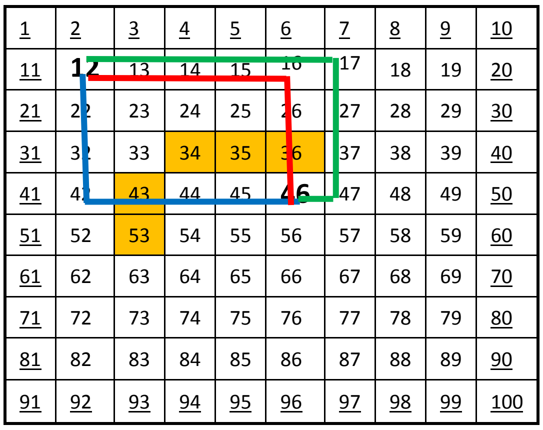
In figure 5.1 the learning agents start from a non-boundary state “12” at the upper left corner and the task of the learning agents is to find the shortest possible path to the goal state “46” (helipad) in the shortest possible time. The goal described above is often referred to as the first Helipad Goal objective, or Objective 1. In the above example there are many possible solutions to this first objective, two of them are marked with a red and blue line in Fig5.1. In either solution, the shortest possible number of steps taken to reach the goal is 7 steps.

5.2.2 Obstacle Avoidance (2nd Objective Boundary)

In order to maintain visual localisation allowing the agent to identify the current perceived state, it is necessary that the agent does not deviate too far away from the helipad visual marker. The previously described second objective of not becoming lost off the edges of the grid world environment can be implemented through a scalarisation approach to compute boundary punishments (-1) independently of helipad rewards (+1) using a vector of rewards. The secondary objective of staying within the confines of the grid world environment is often referred to as the second boundary objective and is complementary to the first objective providing the helipad goal is not situated on a boundary state. The location of the boundary punishments is emphasised in Figure 5.1 with the boundary states underlined and the start and goal stated in bold.

5.2.3 3D Hill Traversal (3rd Objective Hills)

To further test the multi objective capabilities of the proposed GPFMORL algorithm, a third and potentially conflicting objective is considered during the learning process. The third objective, also referred to as the Hill objective, allows three dimensional obstacles such as buildings or hills to be either avoided or traversed depending upon the cost of flying over or around the hill. In the below example there is a total of five hills, each with their own punishment value ranging from -0.1 to -0.8. For example the first hill in state 43 may be -0.1, the second hill in state 53 may be -0.2, the third hill occupying state 34 may be -0.4, the fourth hill occupying state 35 may be -0.6 and the final hill occupying state 36 may be -0.8. No hill punishment value should be equal to or exceed the boundary punishments due to due to avoiding or traversing the hills always being preferred compared to exiting the grid world by entering a boundary state described by the secondary objective



Blue Policy – Minimum of seven steps to goal crossing -0.1 Small Hill

Red Policy – Minimum of seven steps to goal crossing -0.8 Tall Hill

Green Policy – Minimum of nine steps to goal crossing No Hills

Figure 5.2: Multi Objective Solutions

The blue line in Figure 5.2 represents the optimal policy in terms of the shortest number of steps to reach the goal and only having to traverse a small hill. The red line shows the agent traversing a very large hill which consumes more energy so in fact, a better solution would be the green line going around the hill at the cost of taking more steps to reach the goal.

5.2.4 Grid World Exploration

In the following experiments, the agent learns which direction to select for every state in order to reach the goal in as few steps as possible whilst satisfying several other objectives. To accomplish this task the learning agent explores each option for every state, evaluates the long term outcome and updates the value functions of every state, so that the next time that state is visited, the agent can select the action with the highest probability of success. ie: reaching the goal in as few steps as possible whilst minimising the number of boundaries and tall hills encountered.

Inevitably, the probabilities of success for each action taken in every state will diverge from their initial values towards their real values at a rate that is directly proportional to the number of times that the state is visited. It becomes clear that in order to generate accurate probabilities for all actions in every state, all states must be visited as many times as possible. Obviously this poses a problem in terms of computational time for large state spaces due to the explorative behaviour attempting every possible route. As a solution, the explorative behaviour could be significantly reduced making the agent "greedy" however then the agent is less likely to find new more optimal solutions due to many states being not being visited therefore their probabilities of success are unknown.

In most reinforcement learning implementation, there is an issue concerning the trade-off between "exploration" and "exploitation" (Sutton and Barto, 1998). It is the balance between trusting that the information obtained so far is sufficient to make the right decision (exploitation) and trying to get more information so that better decisions can be made (exploration). For example, when a robot faces an unknown environment, it has to first explore the environment to acquire some knowledge about it. The experience acquired must also be used (exploited) for action selection to maximize the rewards (Kantardzie, 2002). Choosing an action merely considering the best actions will lead to a exploiting behaviour. In order to create a balance, Barto et al (1983) suggested the addition of a noise signal with mean zero and a Gaussian distribution. The use of this signal will force the system into an explorative behaviour where different than optimum actions are selected for all states; thus a more accurate input-output mapping is created at the cost of learning speed. In order to maximize both, accuracy and learning speed an enhanced noise signal is proposed. This new signal is generated by a stochastic noise generator defined in (5.1).

$$n_k = N(0, \sigma_k) \tag{5.1}$$

Where N is a random number generator function with a Gaussian distribution, mean zero and a standard deviation $\sigma_k$which is defined as

$$\sigma_k = \frac{1}{1+e^{[-2p_k(t)]}} \tag{5.2}$$

The stochastic noise generator uses the prediction of eventual reinforcement, $p_k(t)$ shown in (5.2), as a damping factor in order to compute a new standard deviation. The result is a noise signal which is more influential at the beginning of the runs, boosting exploration, but quickly becomes less influential as the agent learns, leaving the system with its default exploitation behaviour. Considering there is only one learning agent for every action and every learning agent assigns a probability value to every action according to Equation 5.3.

$$P'_k = P_k + n_k \tag{5.3}$$

Where $n_k$was defined in Hinojosa (2011). If only one action exists with a probability value equal to max P'_k and this probability value is greater than or equal to the predefined epsilon threshold value of $\varepsilon | \varepsilon \in \{0,1\}$, then the action will be selected. Alternatively an action $a \in s$ is selected at random. The previously mentioned epsilon value ε can be defined within the reinforcement learning algorithms parameters in order to bias the behaviour towards exploratory or exploitative tendencies.

5.2.5 Multi Objective Deterministic Environment

5.2.5.1 System Description

This simulation uses a 10x10 square grid to illustrate the value functions for a simple finite MDP. Each of 100 cells within the grid represents a different state of the environment in which the agent can move in any one of the four possible directions: North, South, East or West. These actions deterministically cause the agent to move one cell in the respective direction within the grid world environment with a probability of 1.

All episodes start from state 12 which is at the upper left corner of the grid world adjacent to the North and West boundaries with coordinates S(2,2). The agent proceeds North, South, East or West by one state in every step with equal probability. The primary objective is to find an optimal policy for moving from the starting S(2,2) state to the goal state $G(m_{g,}n_g)$ with minimised cost i.e. minimum number of moving steps. The secondary objective avoids traveling towards the outer boundary states in order to minimise the risk of becoming lost outside of the grid world environment i.e.: minimum number of boundary steps. The third and final objective is to either circumnavigate an obstacle or traverse over the top of the obstacle depending upon its size and energy cost involved i.e.: minimum number of High Hill steps. The overall objective is to find an optimal policy which satisfies all three objectives by using a scalarisation approach described in section 4.2.

For this series of simulation experiments, the initial starting location of the agent is set to S(2,2) or state 12 and the goal location is fixed at G(5,6) or state 46. The state number can be derived according to equation 5.4 where n_{max} is the total number of grid world rows and *m* and *n* are the vertical and horizontal axis positions of the grid.

$$n_{max} \times (m - 1) + n \tag{5.4}$$

The rewards and punishments associated to different states remain constant for each algorithm and for all experiments as shown in table 5.1

TABLE 5.1 REWARD AND PUNISHMENT SCHEME FOR THREE OBJECTIVES IN THE RANDOM WALK

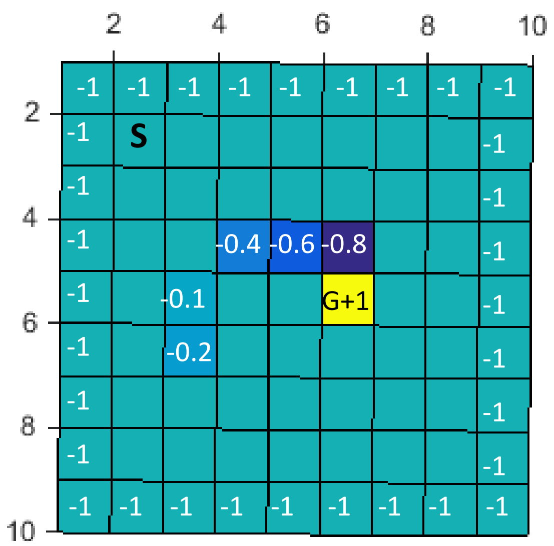
Reward/Punishment	GPFMORL	MOSARSA	MOQ-Learning
Goal State 46	1	1	1
Boundaries	-1	-1	-1
Hill State 34	-0.4	-0.4	-0.4
Hill State 35	-0.6	-0.6	-0.6
Hill State 36	-0.8	-0.8	-0.8
Hill State 43	-0.1	-0.1	-0.1
Hill State 53	-0.2	-0.2	-0.2

Any over boundary actions result in a negative reward or punishment of -1 as the table shows however, the location of the agent will not change in this circumstance to allow the learning process to continue using observable states. Any other action that has not been described results in a reward of zero “0”. The MORL parameters and objective weightings for each algorithm are shown in table 5.2

TABLE 5.2 MORL PARAMETERS AND OBJECTIVE WEIGHTINGS

Parameter/Weights	GPFMORL	MOSARSA	MOQ-Learning
α	0.003	0.4	0.4
β	0.005	-	-
γ	0.95	0.95	0.95
ε	0.01	0.01	0.01
Objective 1	0.4	0.4	0.4
Objective 2	0.3	0.3	0.3
Objective 2	0.3	0.3	0.3

For this experiment the discount factor γ is set to 0.95 and the action selection strategy is ε-greedy which is determined by setting the epsilon value to $\varepsilon = 0.01$. The parameters shown above in table 5.2 are the values used in this series of simulations and the corresponding code developed and evaluated using MATLAB 2014® can be found in Appendix A.



+1 = 1st Goal Objective (Yellow S46)

-1's = Boundary's (outer perimeter)

-0.1 to -0.8 = Hills/Buildings The darker the blue, the taller the building

Figure 5.3: MORL Punishment Locations

5.2.5.2 Fixed Start State

For this test, the simulation was set to execute 40 trials of starting from the start state S(2,2) and attempting to reach the Goal G(5,6) whilst satisfying the other objectives as shown in figure 5.3. These results are then averaged over a total of 100 plays. Figure 5.4 Shows the average number of steps the agent takes to reach the goal state for the 1st objective, the average number of boundary steps taken for the 2nd objective and the average number of steps traversing over a hill for the final 3rd objective.

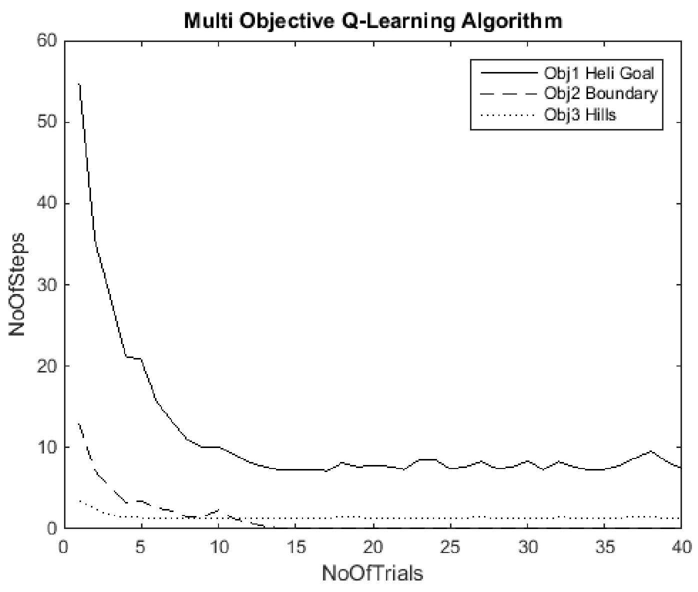


Figure 5.4: Static Starting position MOQ learning rates 40trials & 100 Plays (5.470s)

The graph shown above in figure 5.4 illustrates that in the very first trial it took approximately 55 steps to reach the helipad goal, 13 of those steps were boundary crossing steps and 4 of them were hill traversal steps. This is clearly not the optimal policy at the beginning of the trials but as the learning process continues, by the 10th trial, a more optimal solution has been discovered which satisfies all the objectives as described by the reward structure in figure 5.3.

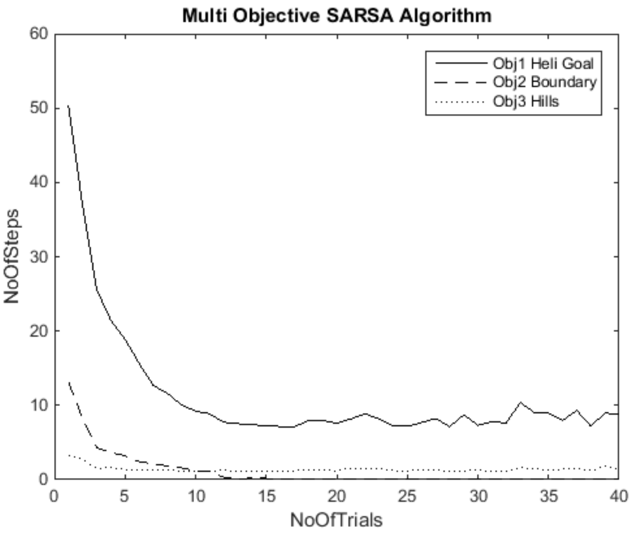


Figure 5.5: Static Starting position MOSARSA learning rates 40trials 100 Plays

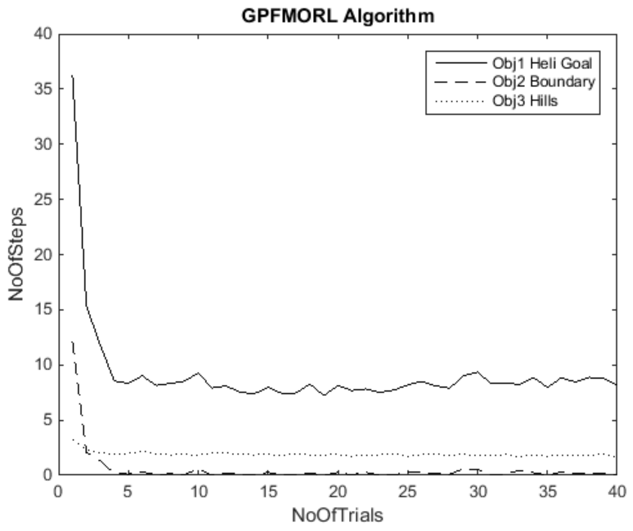


Figure 5.6: Static Starting position GPFMORL learning rates 40trials 100 Plays

It can be seen that the GPFMORL algorithm converged the optimal policy much faster in only 4 trials compared with MOQ and MOSARSA that took approximately 15 trials. The GPFMORL algorithm also took only 37 steps in the very first trial compared with 55 and 50 for MOQ and MOSARSA respectively. Finally the GPFMMORL algorithm also demonstrated more stability than the other algorithms showing only small deviations from the optimal policy once found.

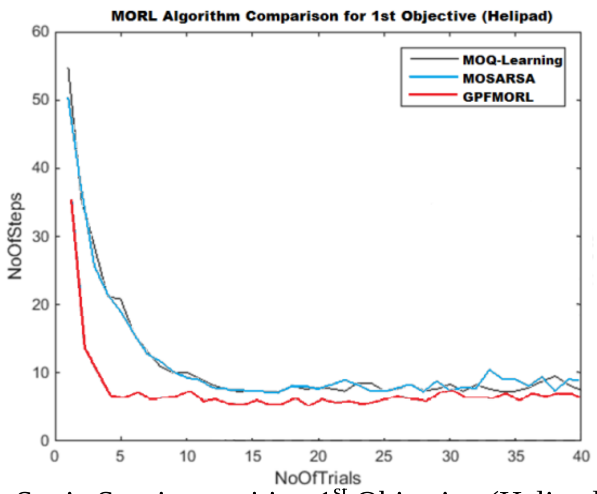


Figure 5.6.1: Static Starting position 1st Objective (Helipad) Comparison

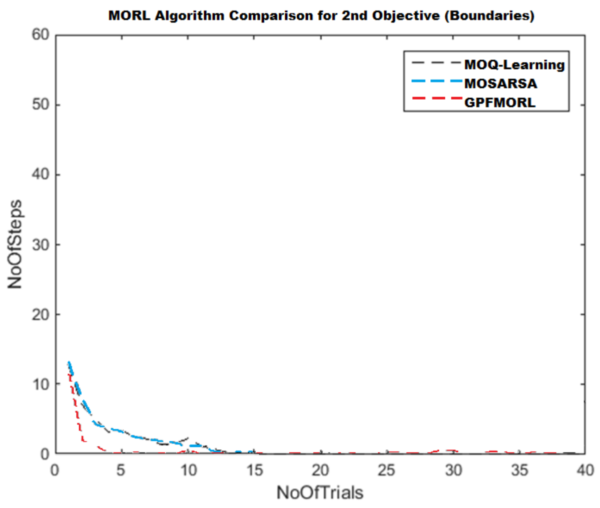


Figure 5.6.2: Static Starting position 2nd Objective (Boundaries) Comparison

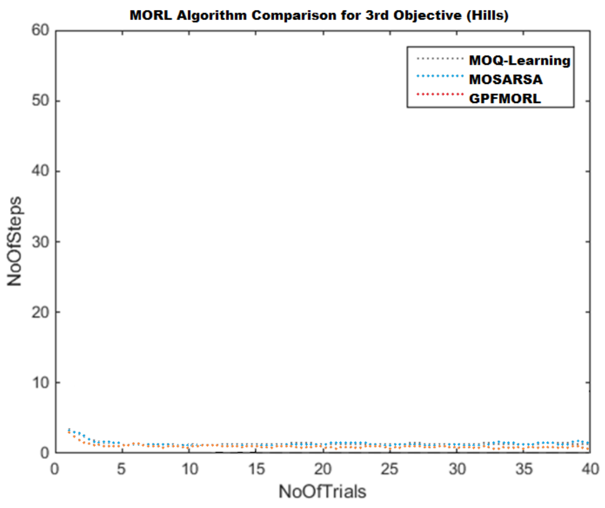


Figure 5.6.3: Static Starting position 3rd Objective (Hills) Comparison

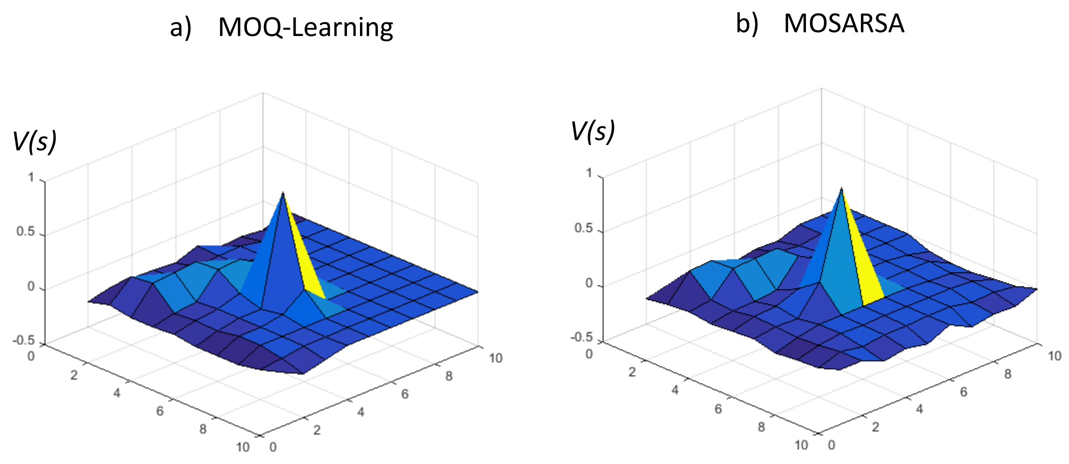


Figure 5.7: Fixed Start State Utility Value Distributions for a) MOQ-Learning and b) MOSARSA

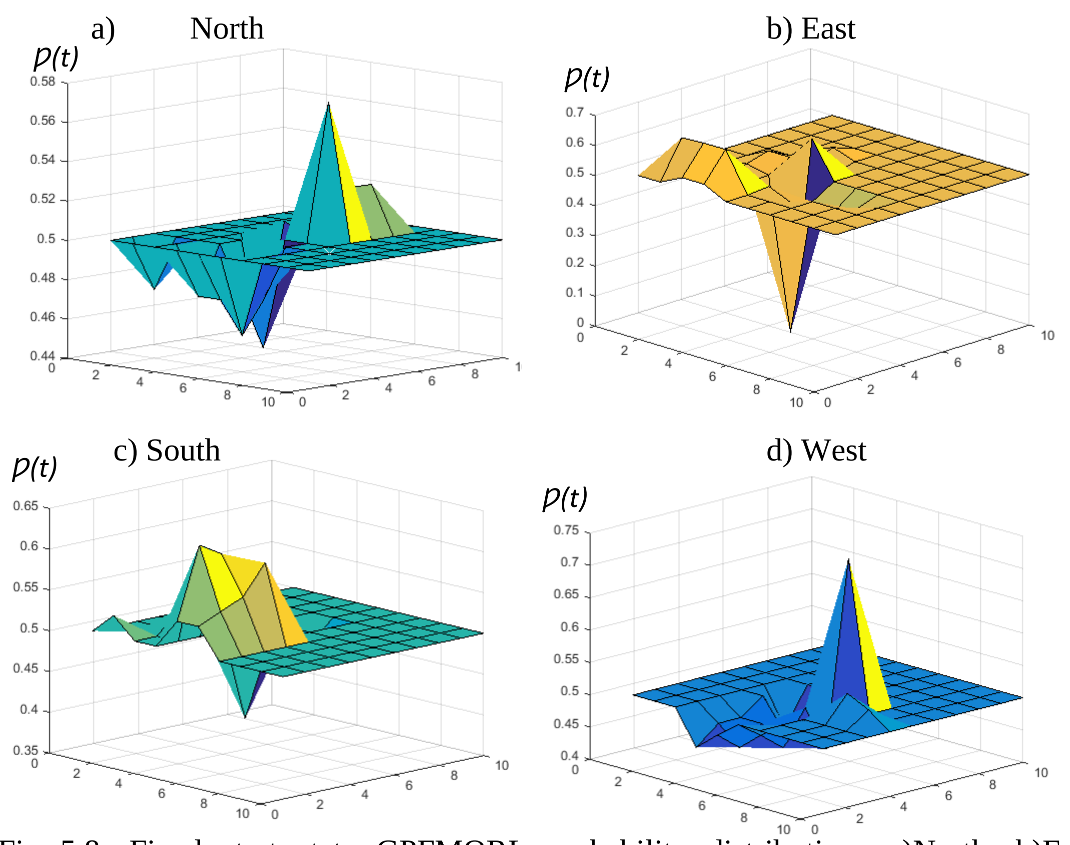


Fig 5.8: Fixed start state GPFMORL probability distributions a)North, b)East, c)South, d)West

Figure 5 .7 Illustrates in a) The scalarised utility value distribution of the MOQ-Learning algorithm and b) The scalarised utility value distribution of the MOSARSA algorithm generated after 40 trials averaged over 100 plays of the random walk experiment starting from a static location moving towards a static goal.

Fig 5.8 shows the GPFMORL algorithms scalarised probabilities of success after 40 trials averaged over 100 plays when choosing any of the four available actions a)North, b)East, c)South and d)West, starting from a static location moving towards a static goal location.

MOQ-Learning and MOSARSA's value assignments for each of the 100 states are clearly visualised in figure 5.7 Where state "1" is situated at the upper left of the lower grid axis. The tallest point on each of the surface graphs is where the goal reward of "+1" is located at state "46" or (m=5 & n=6). All other hills indicate the relative value of each corresponding state in relation with the value assigned to the goal state. In this case the learning agent follows the path with the highest steep due to following an ε-greedy policy tuned for maximum exploitation. It can be observed that the flat surrounding area near the opposite side of the goal is mainly unchanged from its initialised values, this is due to following an ε-greedy policy and not completely exploring every state within the grid world environment as a result of primarily exploiting its successful behaviour. The other reason for unchanged state values is because that particular state may not have led to a state with an assigned value. Should the ε-greedy value be increased to promote a more explorative behaviour then it is much more likely that every state is visited and a value assigned for each state, however this increases computational requirements which can significantly reduce the learning time and produce less stable convergence results with a much higher standard deviation in all cases.

Figure 5.8 shows the surface areas generated for each of the four learning agents. In each sub-graph there are 100 possible states each with their own associated probability for moving either North, South, East or West. In these graphs it can be noticed that there are a series of hills and dips in contrast with MOQ-Learning and SARSA where only hills are present. For the GPFMORL algorithms, the dips can be understood as representing a low probability of taking that choice of action when in that particular state. The addition of low probabilities in the GPFMORL algorithm as well as high probabilities is likely to contribute to a much richer source of options available to the decision making process therefore improving the learning performance. It can also be observed in Fig 5.8 showing the generated probability values for each state only deviate slightly from their initial probability of 0.5, this is due to following an ε-greedy policy therefore the learning agent only requires a small difference in each of the directional probabilities to determine the appropriate action to take. Altering the ε-greedy parameter can force the agent to *randomly* select actions with small probability values therefore resulting in a more exploratory behaviour where more of the state space is explored generating higher variations in probability values.

5.2.5.3 Random Start State

For this second series of experiments within the grid world we shall randomly initialise the starting position of the agent to any of the 100 possible states. The number of trials has been extended to 100 and these have been averaged over 500 plays in order to compare the original single objective GPFRL's performance with the newly proposed GPFMORL approach. Figure 5.9 shows the average number of steps for each of the three objectives for MOQ-Learning. Figure 5.10 shows the average number of steps for each of the three objectives for MOSARSA. Figure5.11 shows the average number of steps for each of the three objectives for the newly proposed GPFMORL algorithm. In figure 5.12 we show the utility value distributions for the MOQ-Learning algorithm a), and the MOSARSA algorithm b). Additionally figure 5.13 shows the probability distributions for each of the 4 agents responsible for moving either North, South, East or West.

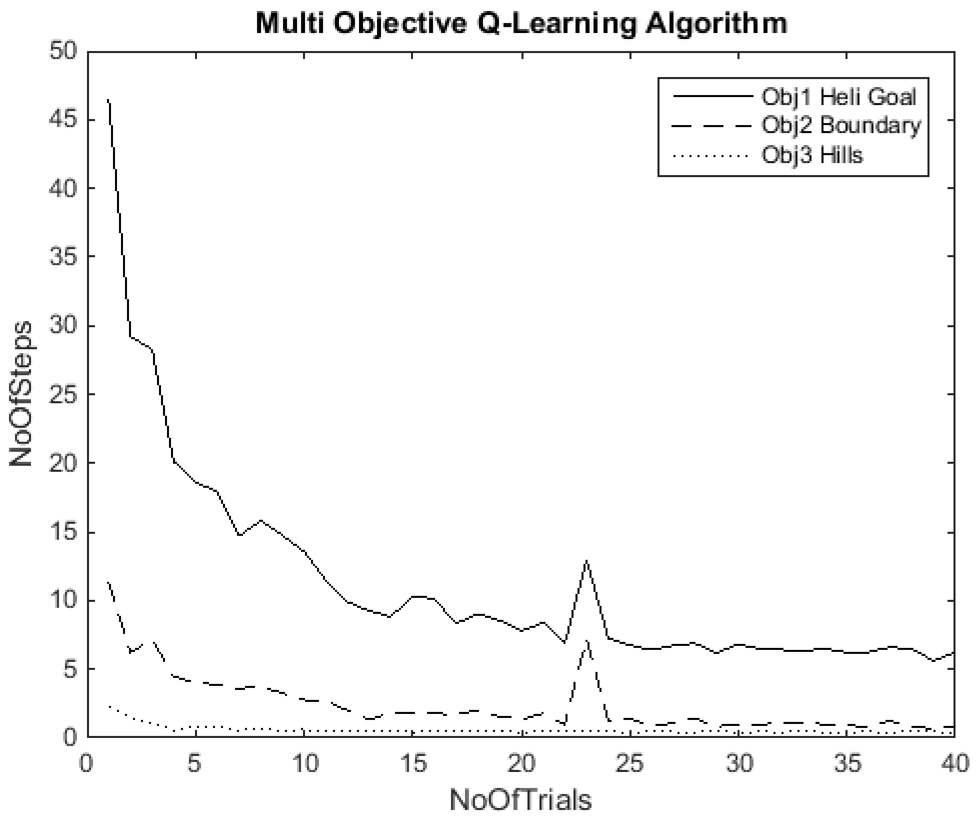


Figure 5.9: Random Starting Location MOQ learning rates 100trials & 500 Plays

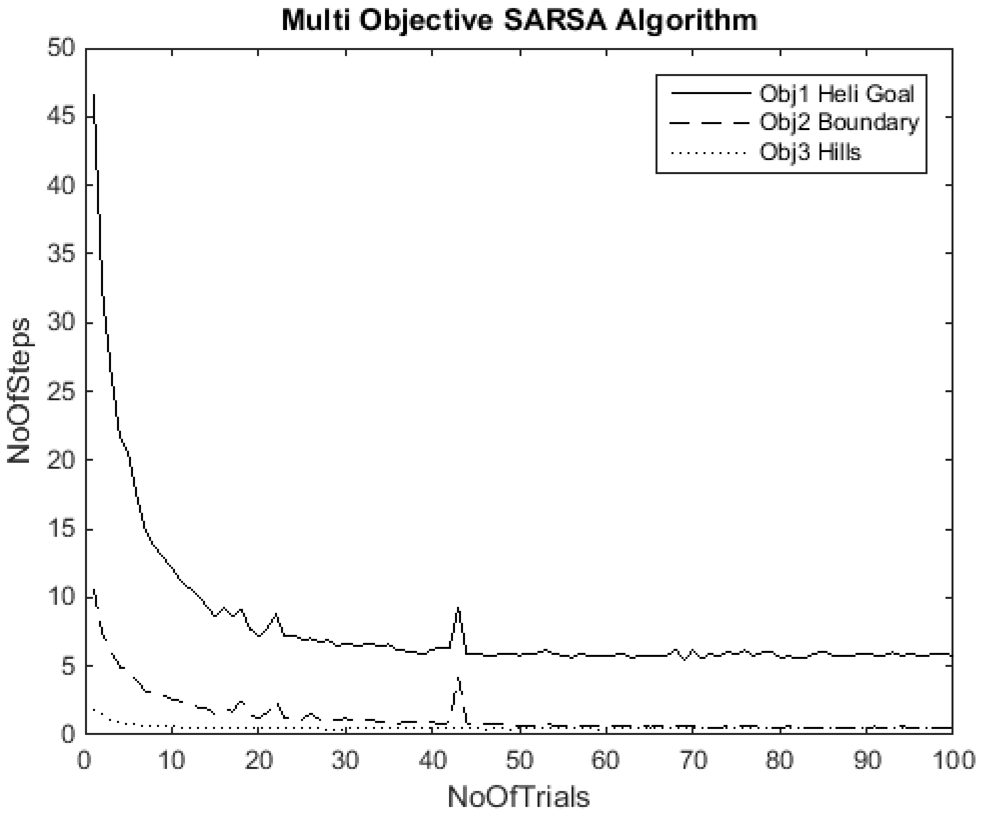


Figure 5.10: Random Start Location MOSARSA learning rates 100trials & 500 Plays

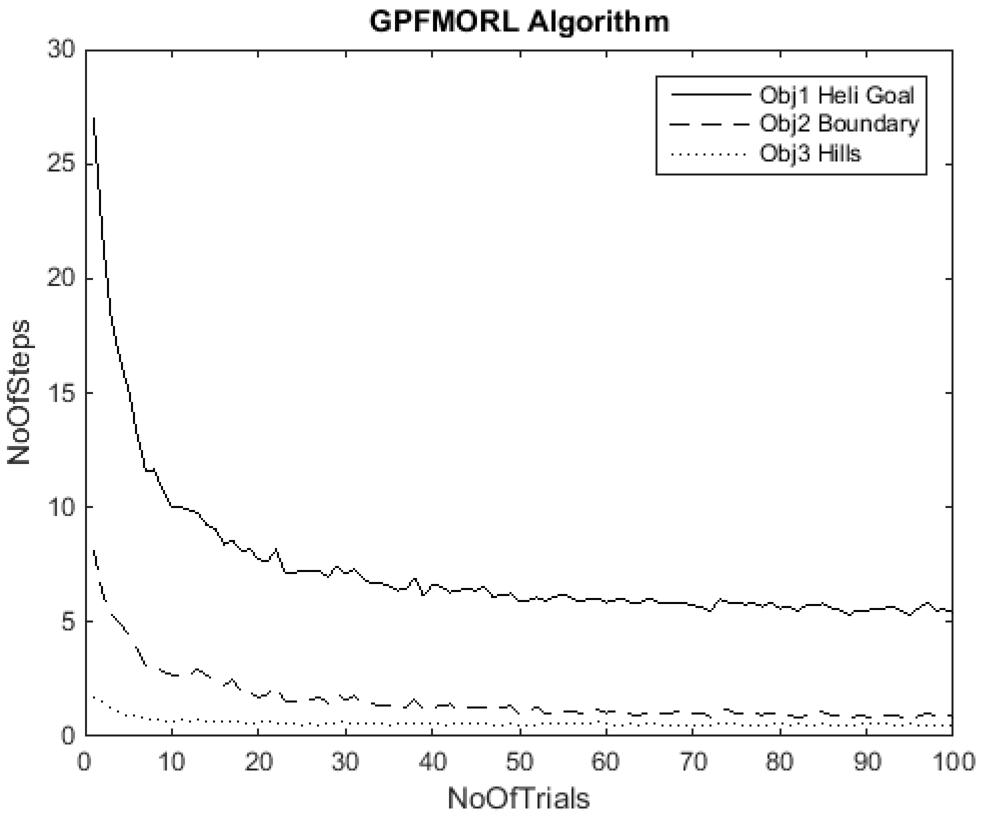


Figure 5.11: Random Start Location GPFMORL learning rates 100trials & 500 Plays

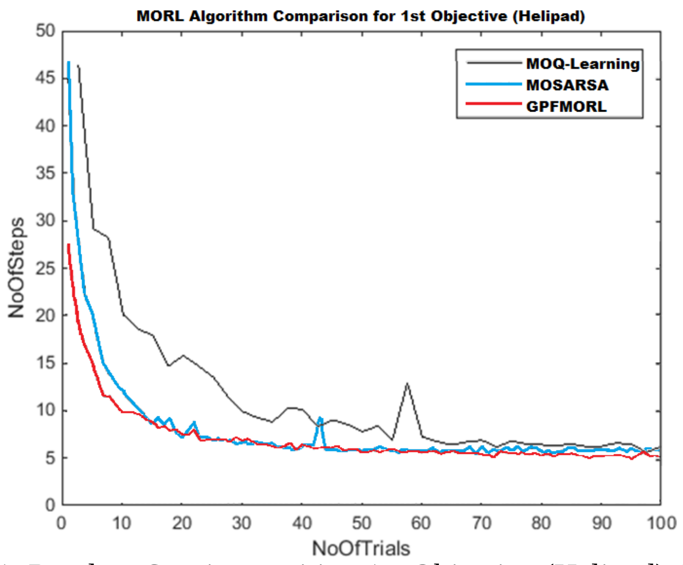


Figure 5.11.1: Random Starting position 1st Objective (Helipad) Comparison

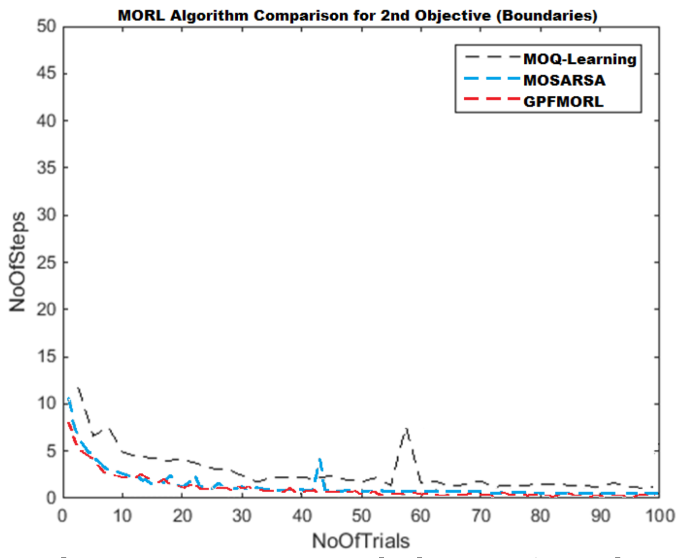


Figure 5.11.2: Random Starting position 2nd Objective (Boundaries) Comparison

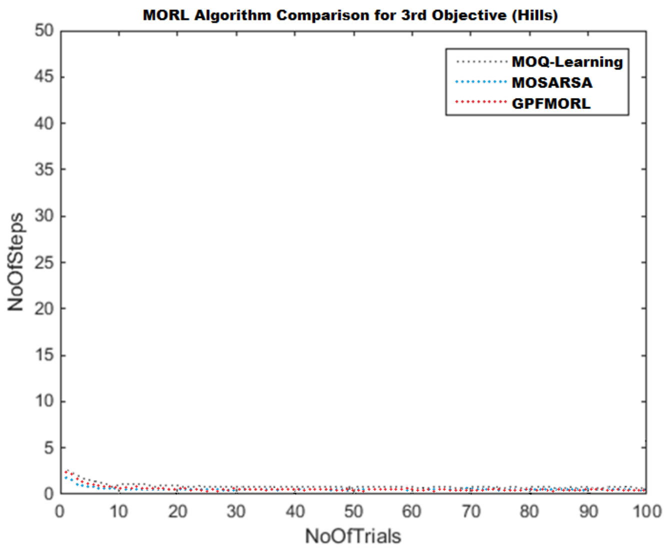


Figure 5.11.3: Random Starting position 3rd Objective (Hills) Comparison

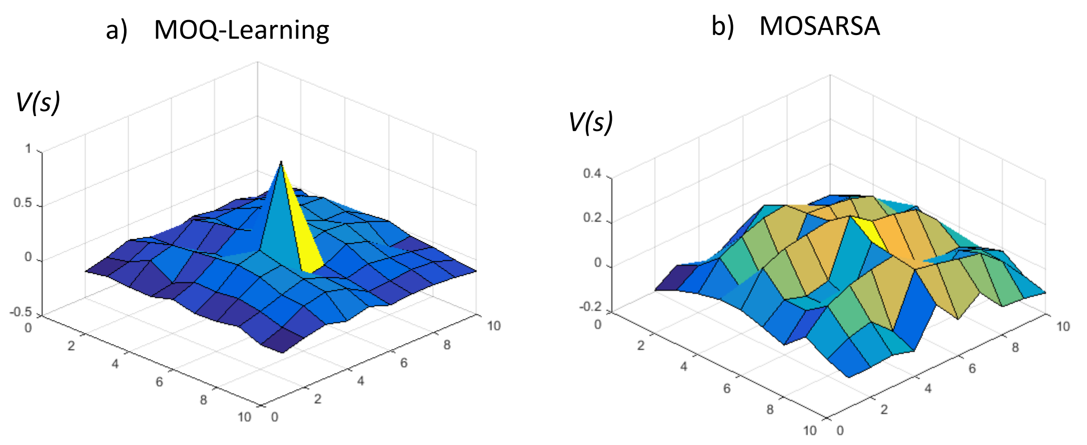


Figure 5.12: Random Start State Utility Value Distributions for a) MOQ and b) MOSARSA

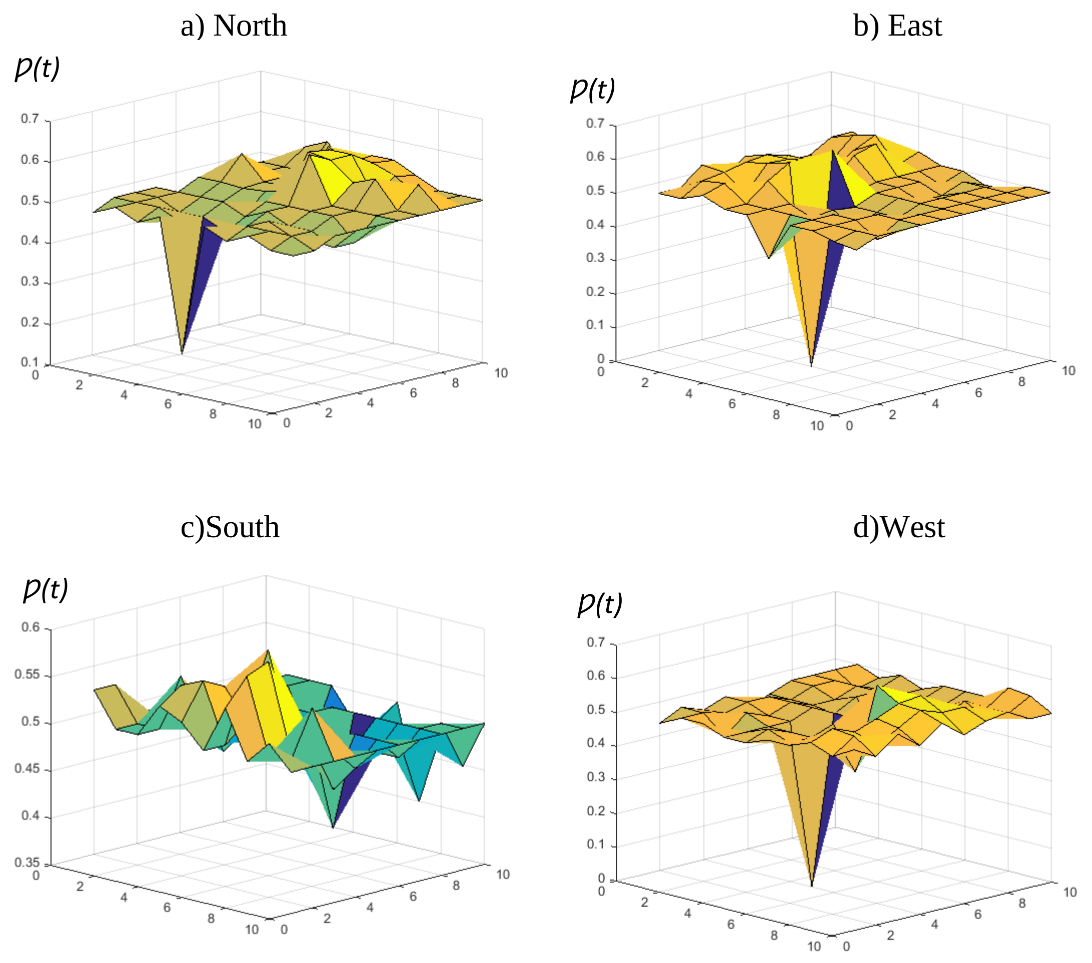


Fig 5.13: Random start state GPFMORL probability distributions a)North, b)East, c)South, d)West

Due to the random starting locations, more of the states are visited therefore more knowledge is acquired about the environment which improves its ability to find alternate optimal policies by behaving in a more explorative way.

5.2.5.4 Discussion

The fixed starting location experiment is an example of a completely deterministic environment without stochasticity present in either the inputs or outputs ie: The stochastic signal generator is set to $\varepsilon = 0.001$). This results in the agent being certain that the selected action taken will always result in moving one step to the adjacent state in that corresponding direction. The fixed start state ensures that the relative position between the starting position and goal state remain the same for every trial. It can be observed that the proposed GPFMORL algorithm exhibits a faster convergence towards the optimal policy whilst MOQ and MOSARSA also learned the optimal policy but in a slightly slower time and was less stable than the GPFMORL algorithm. This is likely to be due to the temporal difference error updating the value functions in every time step where as MOQ and MOSARA must wait until the goal is reached before the value functions can then be updated.

Another important observation is that the optimal value for the learning rate α of GPFMORL is comparatively smaller at (0.003) compared to MOQ and MOSARA which is much higher at (0.4). The learning rate of GPFMORL can be much smaller due to only one value function update being necessary for the learning agent to differentiate between states and take an appropriate action for every state, therefore the learning agents has a rough idea of the optimal policy even after the very first trial. In comparison with MOQ and MOSARA, these algorithms do not have any idea of the optimal policy until a state has been visited where the value function has been updated previously. Regarding MOQ and MOSARSA, it becomes clear that the addition of any stochastic signal will confuse the learning process in the first few trials as any information it has learned will be modified randomly forcing the learner to revisit the same state several times in order to differentiate the decisions to make in every state.

5.2.6 Multi Objective Stochastic Environment

5.2.6.1 System Description

This section of the results presents a different version of the grid world environment called the Windy Hill World in which stochasticity has been added to in order to evaluate the influence on the performance of our proposed GPFMORL algorithm and compare it to that of MOQ and SARSA algorithms. In this experiment, there are two types of stochasticity to be introduced to verify the advantages of the GPFMORL in a broader setting which include:

1. Policy Stochasticity – Altering the policy from e-greedy to a more explorative behaviour by changing the epsilon value from ε = 0.01 to ε = 0.2
2. Environment Stochasticity – Generating varied strength of upwards wind gust depending upon geographical location with a probability of 0.2

Under these stochastic conditions, uncertainties in the input states become inherent which allows us to evaluate the uncertainty handling capabilities of our proposed method. The windy hill world is therefore a more realistic simulation of what decisions may need to be made in the event of aerodynamic forces or windy weather conditions forecasted by the met office.

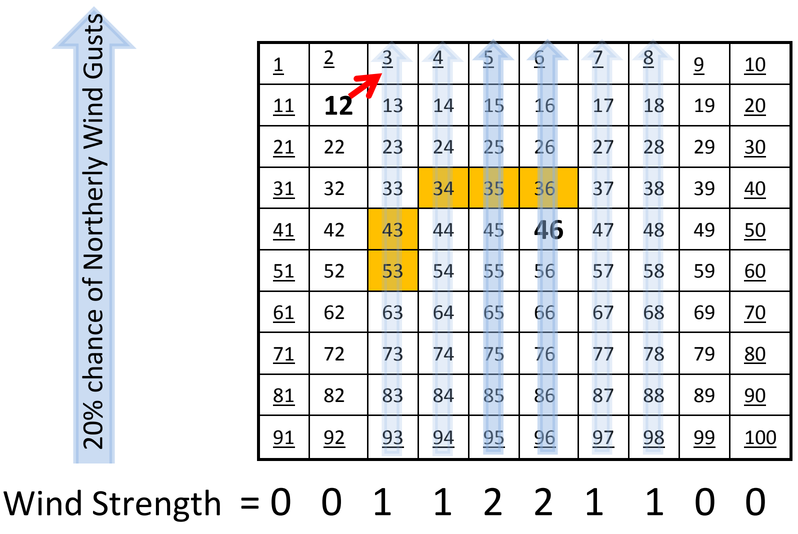

20% chance of Northerly Wind Gusts
12
46
Wind Strength = 0 0 1 1 2 2 1 1 0 0

Figure 5.14 Windy Hill World

In the same way as previous experiments, the agent must move from the starting location (S:12) towards the goal location (G:46) whilst avoiding the boundaries of the grid world and choosing to either go around or traverse the hills/buildings. However in the windy hill world there is a crosswind that travels upwards and its strength varies from column to column. Hypothetically, if this entire environment was also on a large hill that was steeper in the middle and lower at the sides then you could expect there to be a much stronger wind in the middle of the hill at its highest altitude. To really push the limits of the algorithm, the goal location (G:46) is in a column where the maximum wind strength of 2 is blowing. The values of each columns wind strength along with the direction of the wind is shown in Fig 5.14. The wind strength values are the number of cells that the agent will shift north when the wind blows with a probability of 0.2. The red arrow in Fig 5.14 shows an example of the agent trying to move one cell to the right (east) from cell 12 towards cell 13 however due to the moderate northerly stochastic crosswind, its final position actually ends up being 3, one step north of where the agent intended on being located.

5.2.6.2 Multi Objective Stochastic Windy Hill World

Table 5.3 shows the parameter values for all three learning algorithms with added stochasticity.

TABLE 5.3 WINDY MORL PARAMETERS AND OBJECTIVE WEIGHTINGS

Param/Weights	GPFMORL	MOSARSA	MOQ-Learning
α	0.003	0.4	0.4
β	0.005	-	-
γ	0.95	0.95	0.95
ε	0.2	0.2	0.2
Objective 1	0.4	0.4	0.4
Objective 2	0.3	0.3	0.3
Objective 2	0.3	0.3	0.3

For this experiment, the simulation was set to perform 40 trials which are averaged over 100 plays. Figure 5.15 shows the average number of steps taken for all 3 objectives when starting from a static start position moving towards a static goal with the addition of stochasticity.

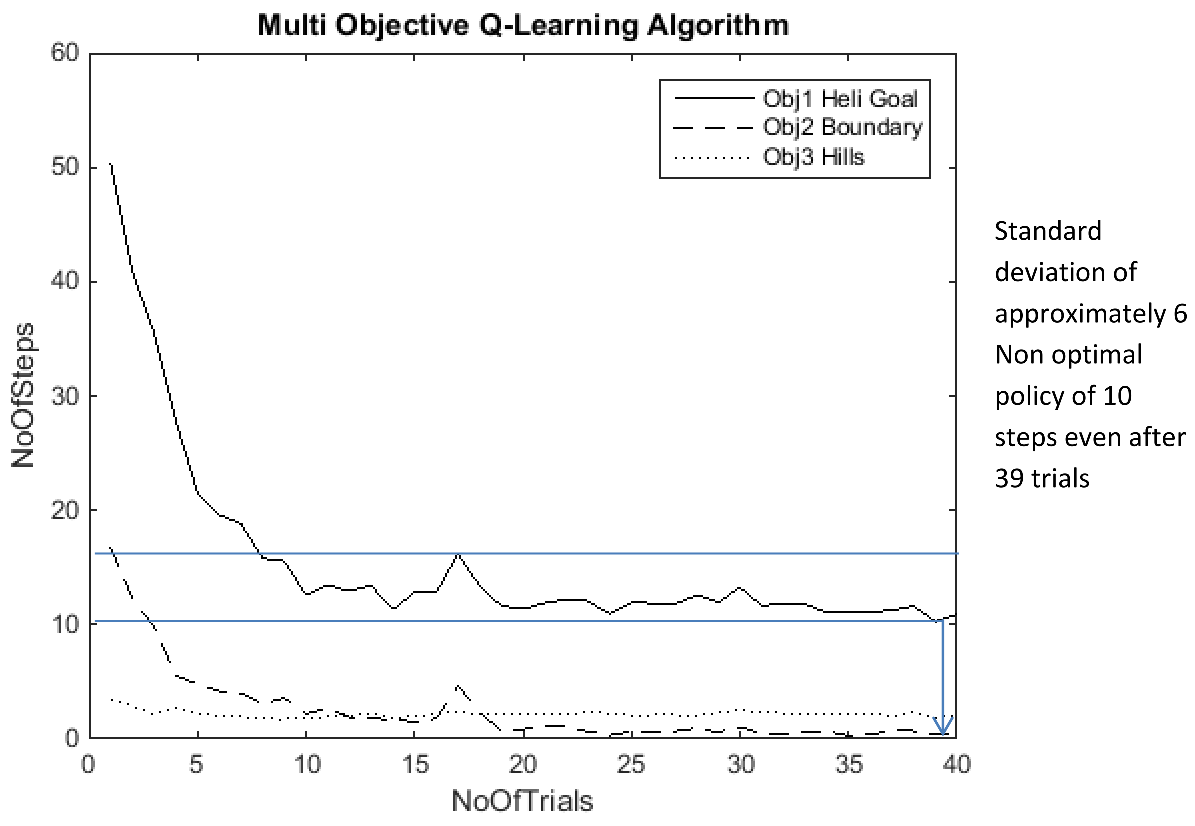


Figure 5.15: Static Start Location Windy MOQ learning rates 40 trials & 100 Plays

It can be observed that the single objective Q-learning algorithm fails to converge under these highly stochastic conditions however the multi objective version of the algorithm copes much better and has a low standard deviation from the optimal policy once reached. The minimum number of steps taken to reach the goal is more than the optimal policy without the presence of the stochastic winds. However, the stochastic winds inevitably cause the agent to steer off course therefore having to take extra steps to compensate and reach the goal. The number of boundary steps taken is also increased due to the stochastic winds often forcing the agent into the northern boundary. The number of hill encounters appears to also have increased to approximately 3 also due to the random gusts of wind forcing the agent into a more explorative behaviour rather than exploitative with a probability of 0.2.

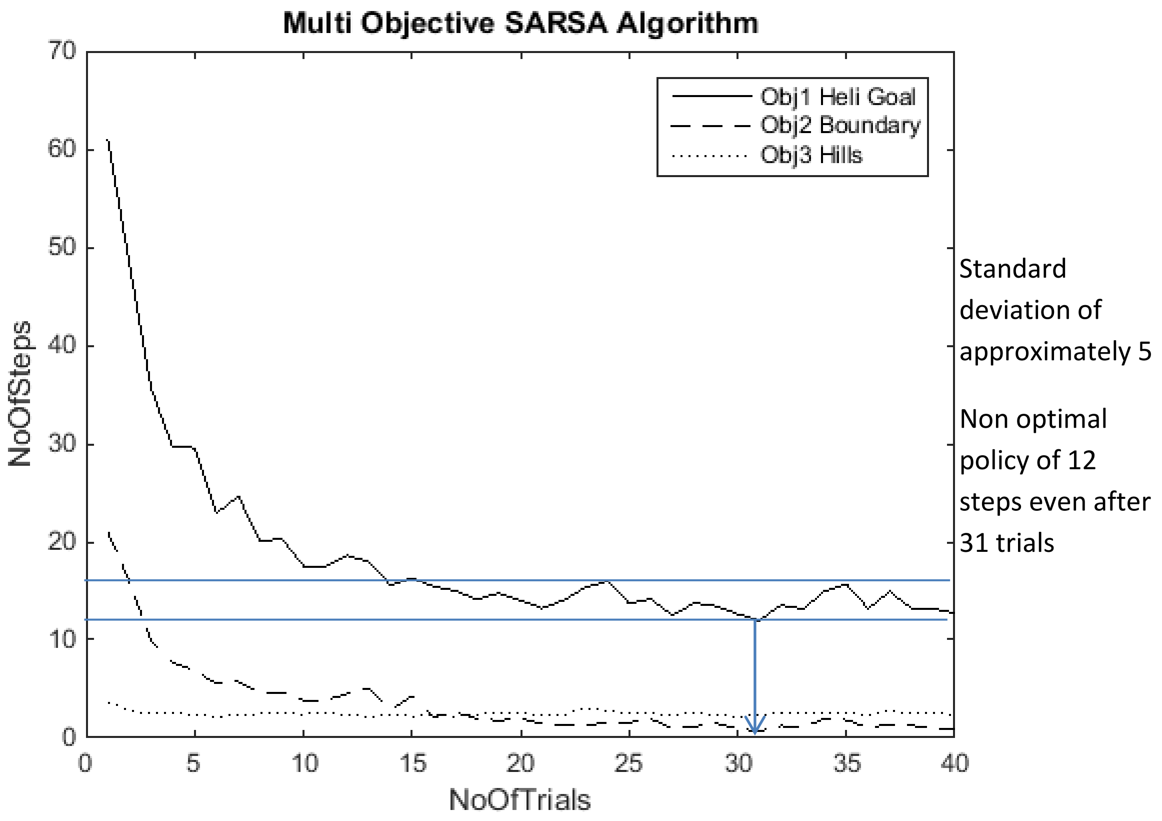


Figure 5.16: Windy Static Start Location MOSARSA learning rates 40 trials & 100 Plays

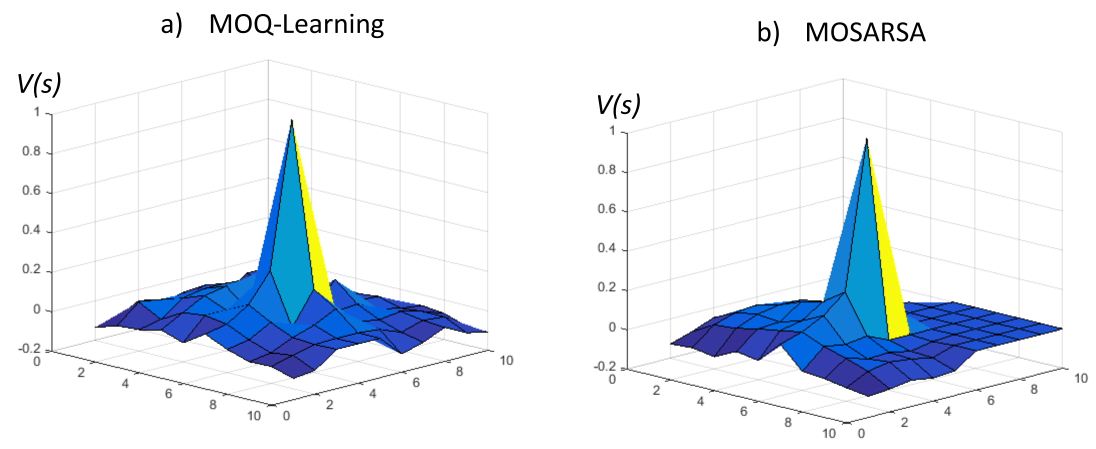


Figure 5.17: Windy Static Start State Utility Value Distributions for a) MOQ and b) MOSARSA

Figure 5.16 shows the learning performance of MOSARSA by storing the average number of steps taken for all 3 objectives when starting from a static start position moving towards a static goal with the addition of stochasticity. Fig 5.17 a) and b) show the utility value distributions for MOQ and MOSARSA algorithms respectively.

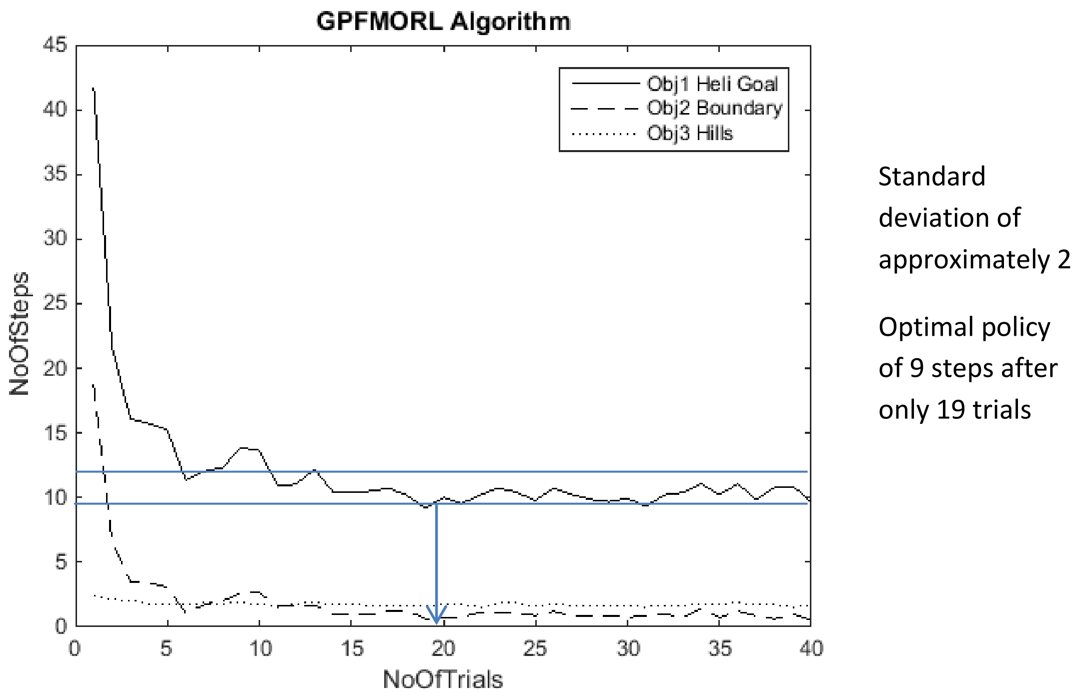


Figure 5.18: Static Start Location Windy GPFMORL learning rates 40 trials & 100 Plays

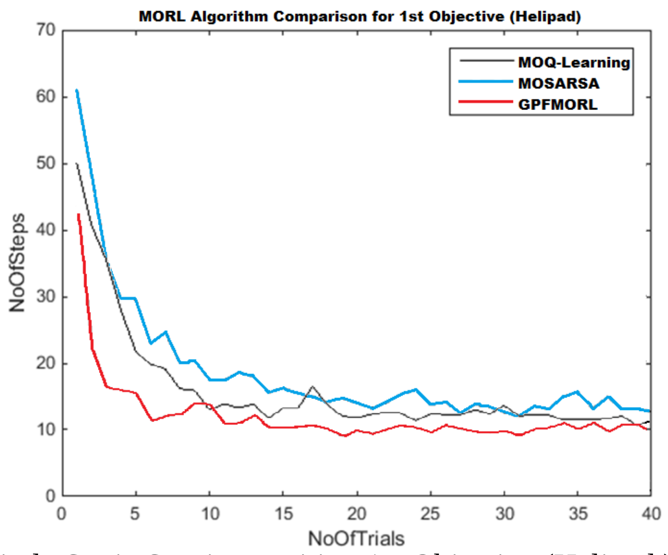


Figure 5.18.1: Windy Static Starting position 1st Objective (Helipad) Comparison

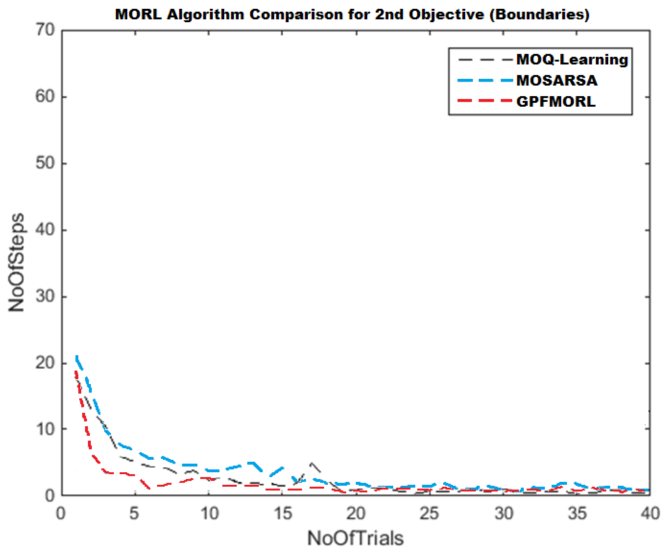


Figure 5.18.2: Windy Static Starting position 2nd Objective (Boundaries) Comparison

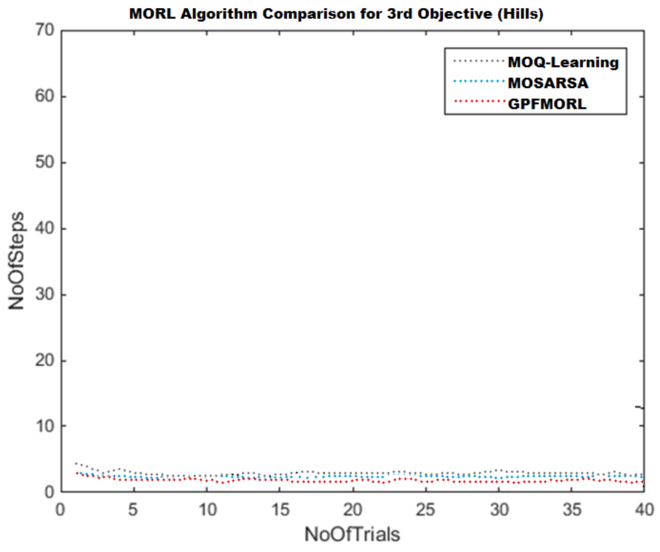


Figure 5.18.3: Windy Static Starting position 3rd Objective (Hills) Comparison

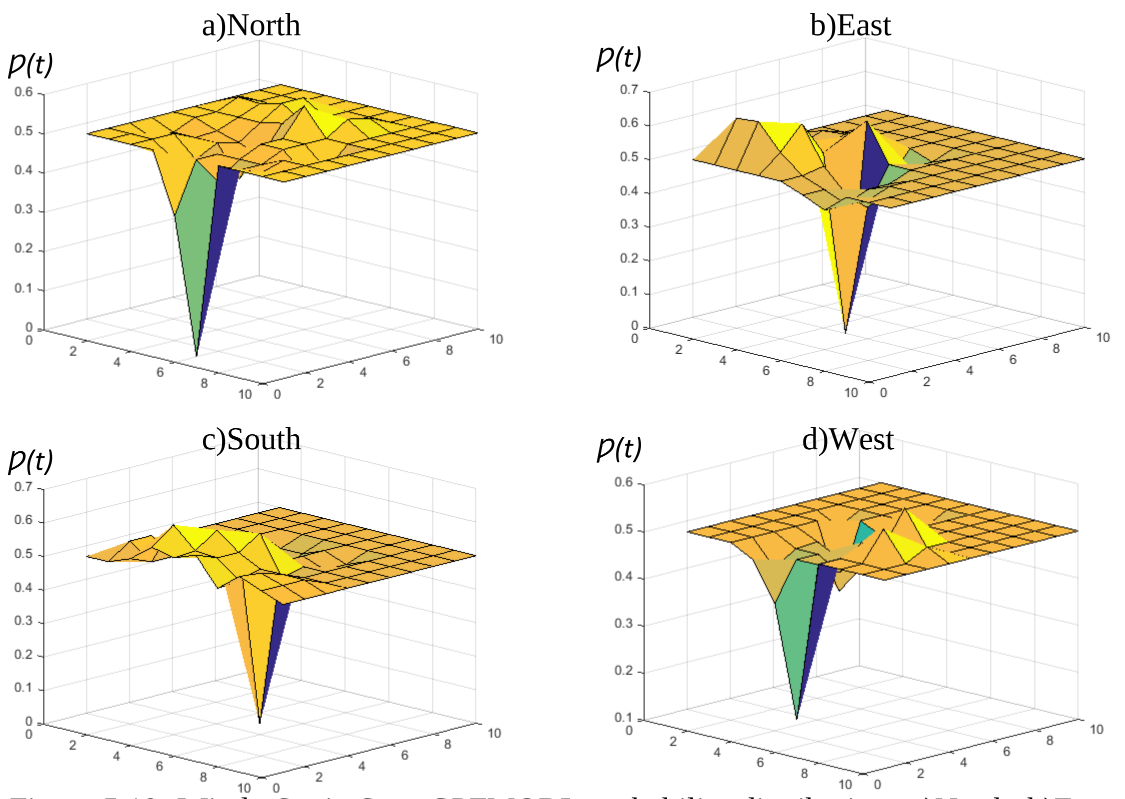


Figure 5.19: Windy Static Start GPFMORL probability distributions a)North, b)East, c)South, d)West

It can be observed that “why does MOQ have less flat surrounding states which contain more information than MOSARSA, could it just be chance, the wind has helped in this particular run of the experiment.

In figure 5.17 we show the utility value distributions for the MOQ-Learning algorithm a), and the MOSARSA algorithm b). In contrast fig 5.19 shows the probability distributions for each of the 4 agents responsible for moving either North, South, East or West.

It can be seen that the GPFMORL algorithm clearly converges to an optimum faster than both the MOQ and MOSARSA algorithms taking only half as many trials to find the optimum policy and with considerably fewer steps in the first trial. The GPFMORL algorithm demonstrates that it is capable of handling the added stochasticity whilst still discovering the optimal policy and exploiting the learned information from then on.

5.2.6.3 Discussion

With the added stochasticity, both MOQ and MOSARSA did find a suitable path to the goal however both failed in all trials to learn the optimal solution. MOQ was 2 steps away from determining the optimal solution whereas MOSARSA was even worse taking 4 steps more than the optimal solution for reaching the goal from a static starting point. These non-optimal solutions found for both these algorithms also took considerably longer to find with almost twice the number of trials than the GPFMORL algorithm took and which actually found the optimal solution, this proving the uncertainty robustness of the proposed algorithm.

5.2.7 Prospect Theory Results

Notice that with prospect theory, number of steps taken to reach the goal in the very first trial is only 15 steps. The agent then quickly begins to improve performance until it experiences a boundary around trial 5. This confuses the system and forces the agent to explore a different path without encountering the boundary. By trial 15, the algorithm converges to an optimal goal step count of 7 whilst traversing two hills/obstacles without ever coming into contact with the boundary. Due to this being a deterministic experiment, when the agent decides upon the optimal policy, there are no factors altering this optimal policy therefore it keeps to this policy thereafter without any standard deviation.

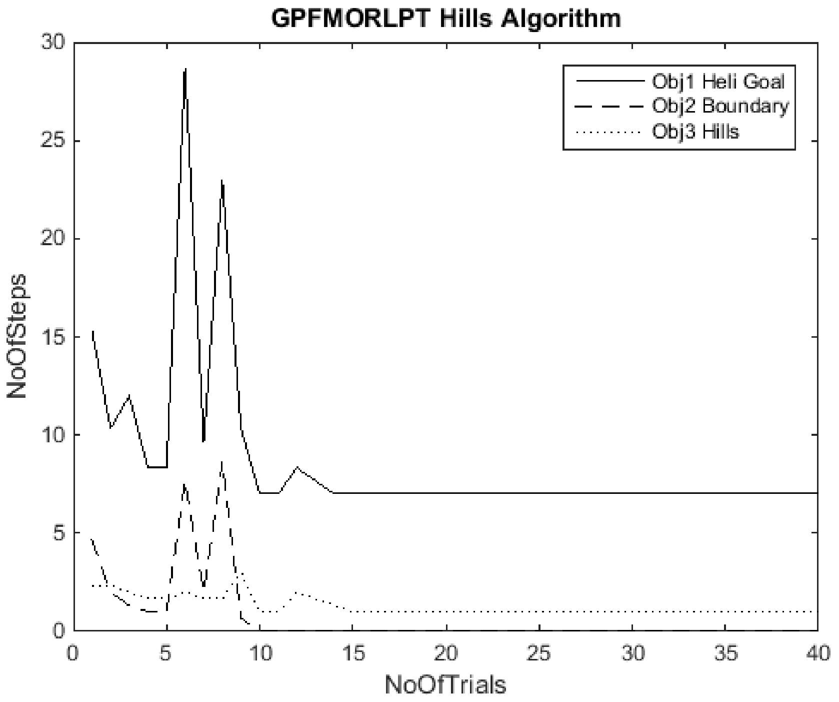


Figure 5.20: Deterministic Experiment with Prospect Theory (40 trials of 3 plays)

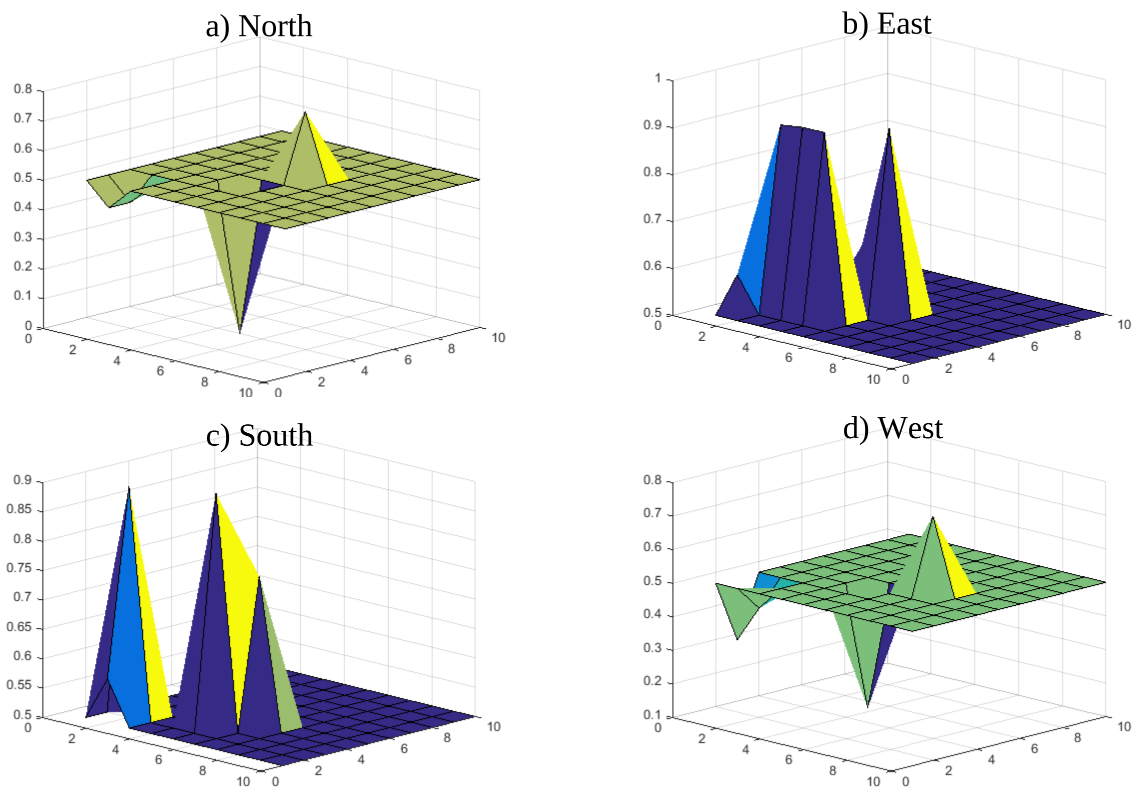


Figure 5.21: Deterministic GPFMORL probability distributions a)North, b)East, c)South, d)West

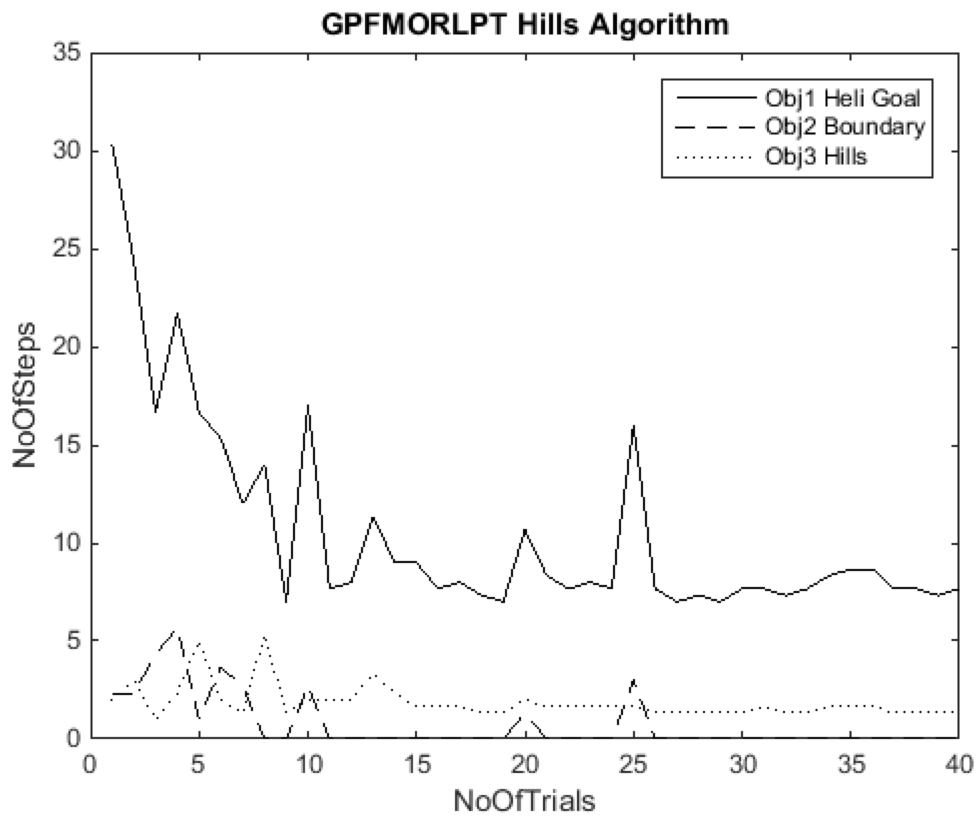


Figure 5.22: Stochastic Experiment with Prospect Theory (40 trials of 3 plays)

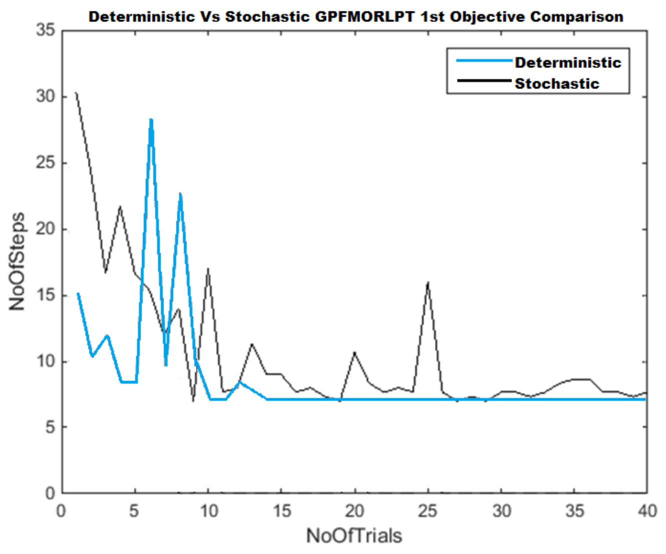


Figure 5.22.1: Deterministic Vs Stochastic 1st Objective (Helipad) Comparison

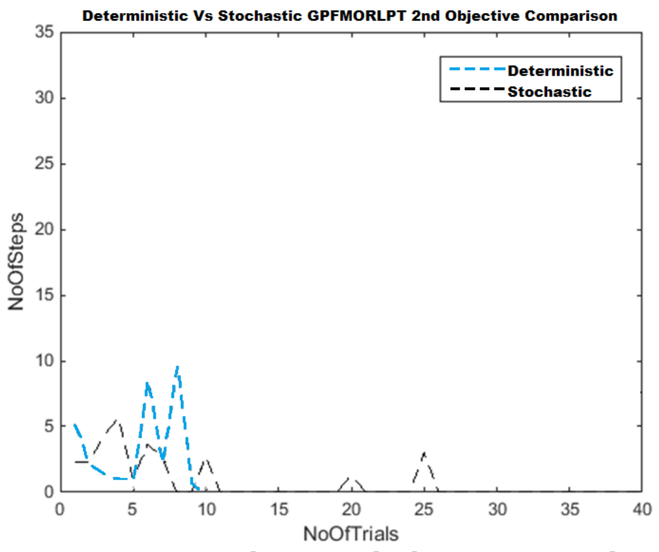


Figure 5.22.2: Deterministic Vs Stochastic 2nd Objective (Boundaries) Comparison

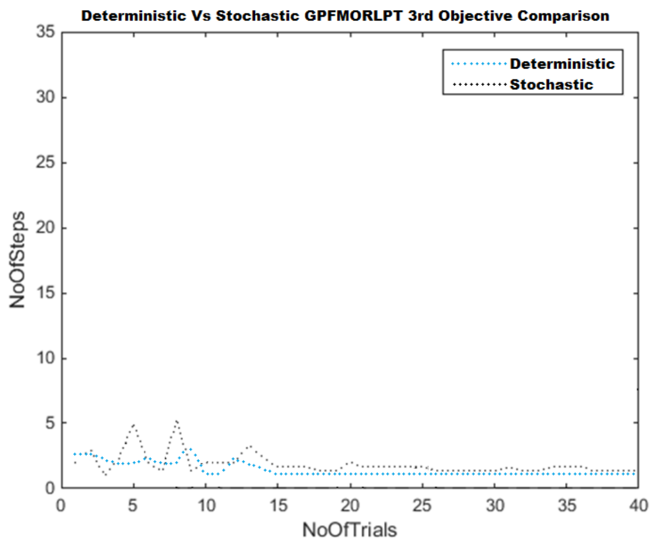


Figure 5.22.3: Deterministic Vs Stochastic 3rd Objective (Hills) Comparison

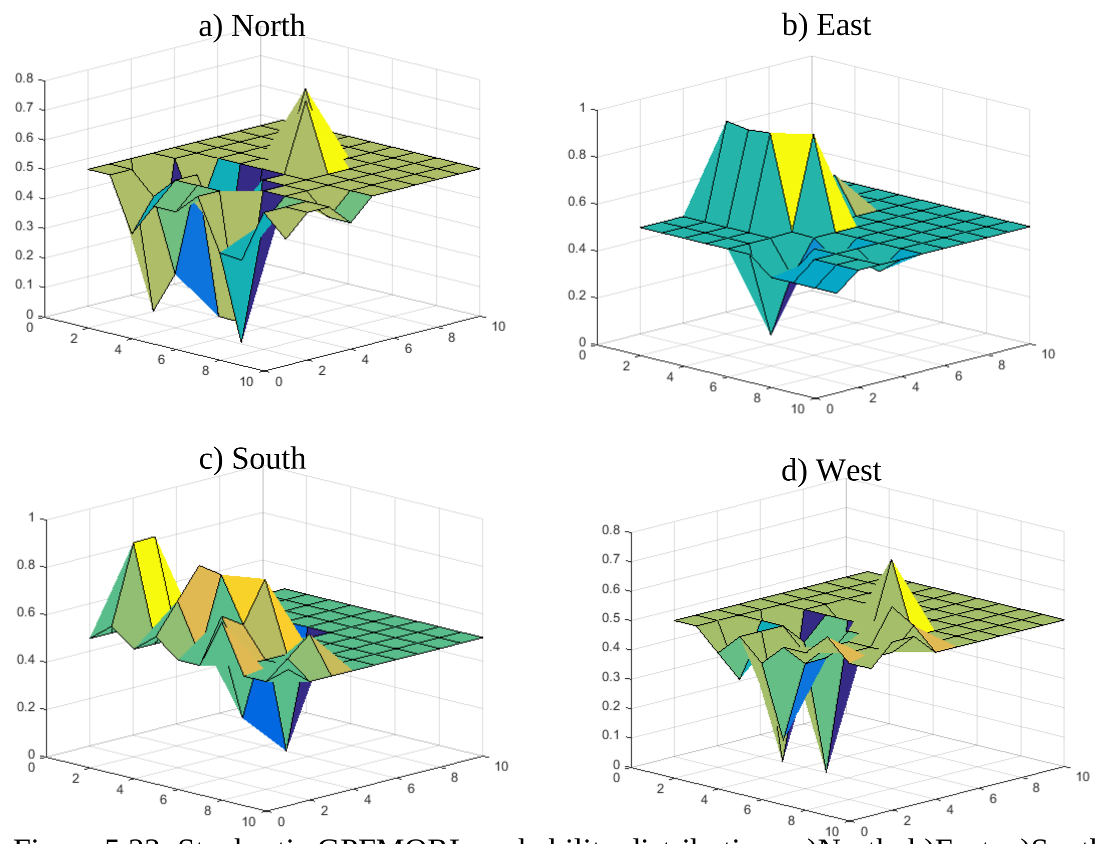


Figure 5.23: Stochastic GPFMORL probability distributions a)North, b)East, c)South, d)West

Experimentation with the stochastic windy hill world environment presents some interesting results. Similar to the deterministic experiment, the algorithm takes around 30 steps to reach the goal and then quickly learns a more optimal route. The optimal policy in this case was actualy reached by the 9th trial which is slightly better than the deterministic case possibly due to the exploration characteristics of the stochastic environment forcing the agent to explore policies which may never have been attempted. Another interesting result is the fact that although this algorithm still converges to the optimal policy faster, it never remains at the optimal policy and shows some standard deviation. This can be attributed to the stochastic nature of the testing environment which causes the actual path to deviate from the optimal policy with a probability of 0.2. Therefore at around trial 20, you can see the effect of this stochastic crosswind on all 3 objectives, such that at this point in time, a cross wind blew the agent upwards to a state where the boundary was encountered. Similary at trial 25, the stochastic cross wind blew the agent twice in concession which resulted in twice as many boundary steps, and therefore twice as many steps required to reach the goal however the number of hills/obstacles remains the same.

5.2.8 Conclusions

The GPFMORL algorithm is conceptually simple and relies on well-established theory's, such as Bayesian statistics. A clear difference between the proposed GPFMORL algorithm and more common algorithms such as MOQ and MOSARA, is the use of a probabilistic model for the transition dynamics which mimics two important features of biological learners.

1. The ability to generalise
2. The ability to explicitly incorporate uncertainty into the decision making process

These model uncertainties have to be taken into account during long-term planning to reduce model bias. Beyond specifying a reasonable cost function, GPFMORL does not require expert knowledge to learn the task. This characteristic is ideal because it would be extremely difficult if not impossible to model all the aerodynamic forces that the UAV would encounter, even with a good model, this could not be relied upon due to the random nature of the stochastic elements.

In the presented experiments, the GPFMORL algorithm was tested in a grid world environment with 3 objectives under different conditions and the results compared to those of MOQ and MOSARSA. First, a standard deterministic version of the grid world was used to test MOQ, MOSARSA and GPFMORL algorithms in a random walk using both fixed and then random starting locations. The purpose of these experiments was to test the ability of the algorithms to handle randomness an also to test the exploration/exploitation characteristics. The results can be observed in figure 5.6 and Figure 5.11 for the fixed starting location and random starting location respectively. The obtained results were conclusive; the GPFMORL algorithm outperformed the classic MOQ and MOSARSA algorithms in terms of learning rate. The GPFMORL algorithm converged to the optimum solution in a shorter time and showed good stability in both experiments. In the random starting location experiment the results were even more remarkable. The GPFMORL algorithm found its way to the goal in the very first trial in fewer than 27 steps whilst MOQ and

MOSARSA took more than 46 steps to reach the goal on their very first attempt with very unstable initial choices.

In the second set of experiments, the GPFMORL was tested in a highly stochastic version of the grid world, the windy hill grid world. In this case, “wind” was added to the standard grid world, in order to add environment stochasticity. Also the greedy term was increased from 0.01 (greedy behaviour) to 0.2 (more exploration) in order to add randomness. Under these conditions both MOSARSA and MOQ-learning failed to converge to an optimal policy. Again the GPFRL showed a strong uncertainty resistance, by converging to the optimal policy in 19 trials and showed a very stable behaviour thereafter.

In all the experiments described, an important factor that contributed to the fast learning speed is the use of a richer reward scheme, where an internal reinforcement signal is provided. Whilst in MOSARA and MOQ-Learning the agent receives a positive reward every time the agent reaches the goal state, in the GPFRL method, the agent receives a reward in every state. This reward is calculated by shaping the external reward (given at the goal state) by using temporal difference as described in (3.3). This internal reinforcement or reward signal represents an estimation of the reward for the current state as a difference between the prediction of eventual reinforcement of the current state and that of the previous state, as it was first described by Barto et al (1983).

Optimal design of reward functions has been studied before (Laud and DeJong, 2003, Mataric, 1994) where different experiments showed faster learning rates. In 1998, Dorigo and Colombetti (1998), suggested the use of reward shaping which import behaviourist concepts and methodology into RL, and discussed a model for automatic training of a RL agent. In the scheme they consider, the automatic trainer has an abstracted specification of the task, and it automatically rewards the agent whenever its behaviour is a better approximation of the desired behaviour. However, it has been suggested by Marthi (2007) that this notion of shaping goes well beyond of using a rich reward scheme.

It appears that by adding more objectives the problem is actually solved much faster due to both complementary and conflicting objectives both being satisfied as one whole problem.

It is clear that a well-designed reward function may facilitate learning, promote faster convergence, and prevent aimless wandering. Also, as pointed out by Laud and DeJong (2003), if the optimal value can be provided as a reward, the RL task successfully collapses to greedy action selection.

The extended MOQPT-learning and MOSARSA algorithms for multiple objectives caused erratic results and therefore have not been included. The present results for the GPFMORLPT (Prospect Theory extended) algorithms were tested in both deterministic and stochastic environments giving satisfactory results leaving more room for exploration before converging to on an optimal path in a similar number of trials to that of the standard GPFMORL algorithm.

In these simulated and practical experiments the GPFMORL algorithm has been exploited to facilitate the real time dynamic learning and control of a UAV. The GPFMORL algorithm generalises the continuous input space with fuzzy rules and has the ability to respond to varying states with the appropriate smooth translational velocities using fuzzy reasoning. Additionally it is also possible to embed prior knowledge into the fuzzy rules, enabling the UAV to explore interesting environments and reducing the training time significantly.

Beyond this specific application, however, the larger and more important issue is whether learning from experience can be useful and practical for more general complex problems. Certainly the quality of results obtained in this study suggests that the approach may work well in practice, and may work better than we have a right to expect theoretically.

5.3 Unmanned Aerial Vehicle Practical Experiments

5.3.1 Introduction

The two most important topics in mobile robot design are planning and control. Both of them can be considered a utility optimization problem, in which a robot seeks to maximize the expected utility (performance) under uncertainty (Thrun, 2000). In order to increase the flexibility of robots, facing unforeseen changes such as changes in their environments or sensor failure, the amount of predefined knowledge used in the control strategy has to be kept as low as possible. The present experiment, presents an automatic learning algorithm that uses reinforcement learning in order to find sensor-motor couplings through the robot's interaction with the environment.

Although the results obtained in many different test and simulated domains look promising, RL techniques have rarely been implemented in application requiring real robots. Robotic applications present difficult challenges to RL methods, nevertheless its experience based motivation, high reactivity and effectively layered structure are still of great potential.

While recent techniques have been successfully applied to the problem of robot control under uncertainty (La, 2003, Pineau et al., 2003, Poupart and Boutilier, 2004, Roy et al., 2005), they typically assume a known (and stationary) model of the environment. This dissertation, investigated the problem of finding an optimal policy for controlling a robot in a partially observable domain, where the model is not perfectly known, and may change over time; whilst proposing that a probabilistic approach is a strong solution not only to the navigation problem, but also to a large range of robot problems that involves sensing and interacting with the real world. However, few control algorithms make use of full probabilistic solutions and as a consequence; robot control can become increasingly fragile as the system's perceptual and state uncertainty, increase.

Reinforcement learning enables an autonomous mobile robot to sense and act in its environment to select the optimal actions based on its self-learning mechanism. Two credit assignment problems should be addressed at the same time in reinforcement learning algorithms, i.e., structural and temporal assignment problems. The autonomous mobile robot should explore various combinations of state-action patterns to resolve these problems.

5.3.2 Autonomous Quadcopter Landing

In our third experiment, the GPFMORL algorithm was implemented on an Parrot AR-Drone 2.0 quadcopter mobile robot (Figure 5.24) in order to learn a mapping that will enable itself to avoid obstacles (hills) and boundaries whilst finding the shortest route to the helipad goal. For this case, four active learning agents were implemented to read four clusters of information gathered from a single Ultrasonic (US) distance sensor and learn the probabilities of success for the preselected actions. The US distance sensor first checks the current altitude of the quadcopter when in any state; this provides a baseline altitude value to which other measurements can then be compared to. The other measurements consist of the drone changing its pose/attitude to angle the proximity detection field to approximately 30 degrees in front, behind, to the left and to the right of the quadcopter all in one manoeuvre, without changing the current position of the quadcopter. On completion of the detection manoeuvre, there shall be five unique US distance measurements for example, The floor direction (altitude) may be 1.5m, The right, front and back distances may all be approximately 2m due to the hypotenuse of a triangle being a longer distance as shown in fig 5.24 a). Finally if there was an obstacle/hill to the left of the drone, then rather than 2m like the rest of the directional proximity values, the left distance may be 1.3 indicating that there is an obstacle or hill present in the state to the left of the drone as shown in figure 5.24b).

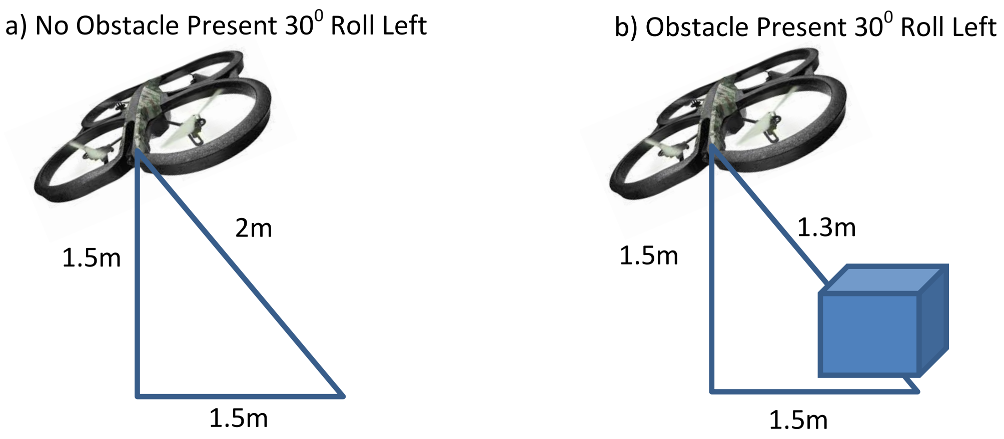

a) No Obstacle Present 30^0 Roll Left
2m
1.5m
1.5m
b) Obstacle Present 30^0 Roll Left
1.3m
1.5m
1.5m

Figure 5.24a,b ¼ of the Detection manoeuvre (Left) for detecting obstacles/hills before being there

In this experiment the GPFMORL algorithm is implemented on a personal computer with a wireless link to an AR-Drone 2.0 quadcopter equipped with forward and down facing cameras. All the computations were performed offline and the data was sent and received via a Wi-Fi link between the PC and the quadcopter. The software was developed in C++ using Microsoft Visual Studio and uses OpenCV computer vision library.

The algorithm consists of two software threads, where one thread manages the communication and the second executes the GPFRL algorithm which decides how to control the quadcopter. Thread one, requests data from the quadcopter such as video streams, Ultra sonic data, inertial measurements etc. which are then sent by the quadcopter and then processed via the C++ programme on the PC. Processing Includes:

- Discretising the perceived field of view into a finite state space of equal proportions.
- Segment helipad from background and calculate the area and position of the helipad
- Potential fields created around the perimeter of the environment to ensure localisation
- Determine the current state of the quadcopter with relative referencing to the helipad.
- Multi-Modal Fuzzy speed controller to determine optimal translational velocities.
- Functions: Record live video streams to file, record drone state history log, HUD, Automatic or Manual control toggle keys (A & Z), Auto-Land toggle key (L).

The second thread reads the processed information from the blackboard, and selects an appropriate action using GPFMORL and velocity using a probabilistic fuzzy logic controller where the rules are updated every cycle using the proposed GPFRL algorithm. The selected action is then executed with the appropriate velocity and the state history is updated accordingly.

5.3.3 Search & Rescue Application

In our fourth experiment we investigate if the proposed framework and algorithm is suitable for performing the much more complex task of search and rescue. The Rescue part of this application is very similar to experiment 3 in the fact that once it has found the goal/Helipad/Collapsed patient it will centralise itself over the goal and then come to a controlled landing as close to the goal as close as possible. The Search part of this application also uses a similar method for moving closer to the goal but instead of a using a ground facing camera, we first use the forward facing camera with a higher resolution. The appropriate actions to take in each state primarily depend upon the learning algorithm however it becomes clear that in order to keep the goal centralised the forward version of the algorithm should increase or decrease altitude along with strafing left or right, alternatively the downwards camera algorithm is more focused on moving forward backward left or right to keep the goal centralised and only really uses the altitude control to acquire a better field of view or when finally landing once centralised.

For this experiment the field of view was slightly increased by mounting a lightweight wide angle lens onto the quadcopter’s existing lens in order to improve perception especially at lower altitudes.

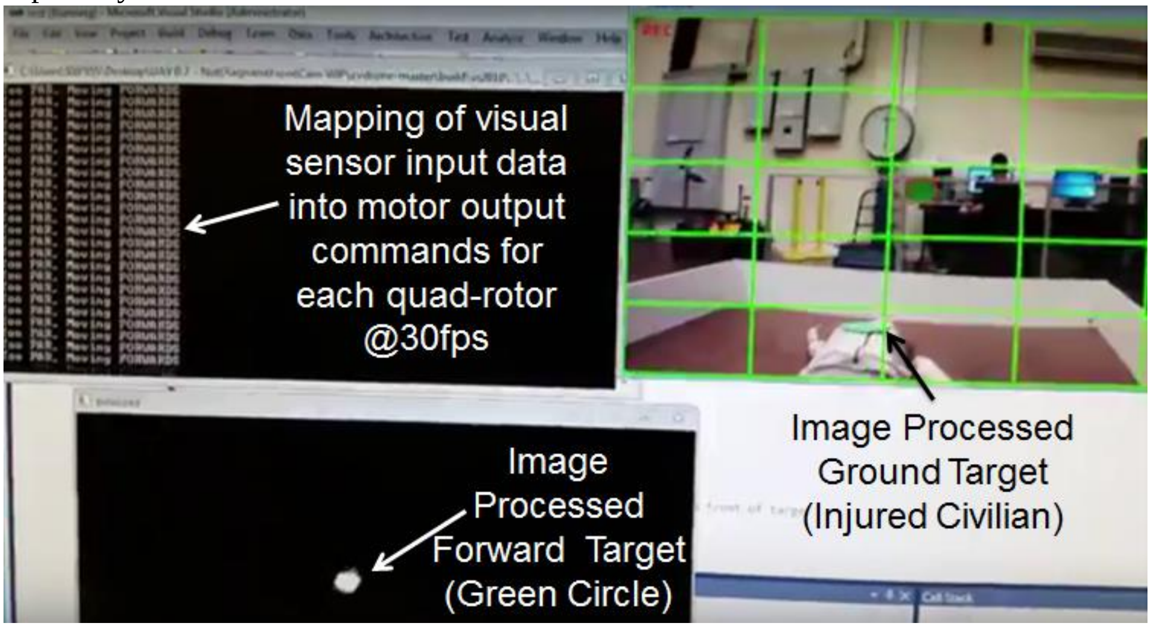

Mapping of visual sensor input data into motor output commands for each quad-rotor @30fps
Image Processed Ground Target (Injured Civilian)
Image Processed Forward Target (Green Circle)

Figure 5.25: C++ Search and rescue application using AR-Drone 2.0 and OpenCV

Conventional RL algorithms aim to maximise performance on a single objective. In the case of Williams recently proposed GPFRL algorithm the objective is to find the goal in as few steps as possible. However many real world problems exhibit multiple objectives such as the UAV must avoid flying outside of the visual localisation grid whilst attempting to land efficiently on the helipad.

The perceived field of view from an AR-Drone 2.0 is discretised into a 5x5 grid world in attempt to reduce the curse of dimensionality in large continuous state spaces such as that of real time video streams. The goal attempting to be reached is the helipad marker which is segmented from the background image using OpenCV image processing in order for the MORL algorithm to know a common frame of reference for determining the agents relative position. A reward of +1 is received whenever the Unmanned Arial Vehicle (UAV) reaches the goal by directly stabilising itself above the helipad goal marker. Conversely a punishment or negative reward -1 is received whenever the agent takes each step that does not lead to one of the terminating goal states.

In the event of the agents trying explorative behaviour to find a more optimal route to the helipad they could actually make a very bad move by attempting to fly outside of the grid world environment. Such “Boundary Crossing” actions should be punished due to resulting in the UAV becoming lost in an unknown state, also potentially encountering unknown obstacles. When hypothetically flying out of the danger zone in simulation, the position of the UAV does not change, however in real flight the UAV must move into an undesired state before being punished, and then move back to a known state. Whilst this is easy to simulate by ignoring the physical change of position when instructed to move outside of the grid world, there must be some other methodology to maintain the agents within the boundaries of the learning environment. To solve this issue of maintaining the agent within the observable MDP, a potential field methodology has been implemented with attractive forces assigned from outer states towards the central goal state thus enabling the MORL agents to explore/exploit within the confines of the grid world environment.

5.3.4 Multi-Modal Fuzzy Translational Velocity Controller

In preliminary heuristic based experiments the speeds for each individual state transition were hard coded however as mentioned previously it is not wise to code the system with too much expert knowledge as this can be uncertain in these highly stochastic conditions. It is therefore proposed that the translational velocity of the state transitions is determined by a fuzzy logic controller. The two inputs to this controller are “Distance to Goal” and “Distance to Ground” due to these factors being responsible for how fast or slow the quadcopter should travel in order to get to its desired state. The inputs are fuzzified using Near and Far Gaussian membership functions in order to generate a crisp value for the velocity to travel in the direction of whatever action is selected The outputs are fuzzified using three membership functions Slow, Default and Fast. The rule base consists of just four simple rules:

IF Distance to Goal = NEAR and Distance to Ground = NEAR then SPEED = Low

IF Distance to Goal = FAR and Distance to Ground = FAR then SPEED = High

IF Distance to Goal = NEAR and Distance to Ground = FAR then SPEED = Medium

IF Distance to Goal = FAR and Distance to Ground = NEAR then SPEED = Low

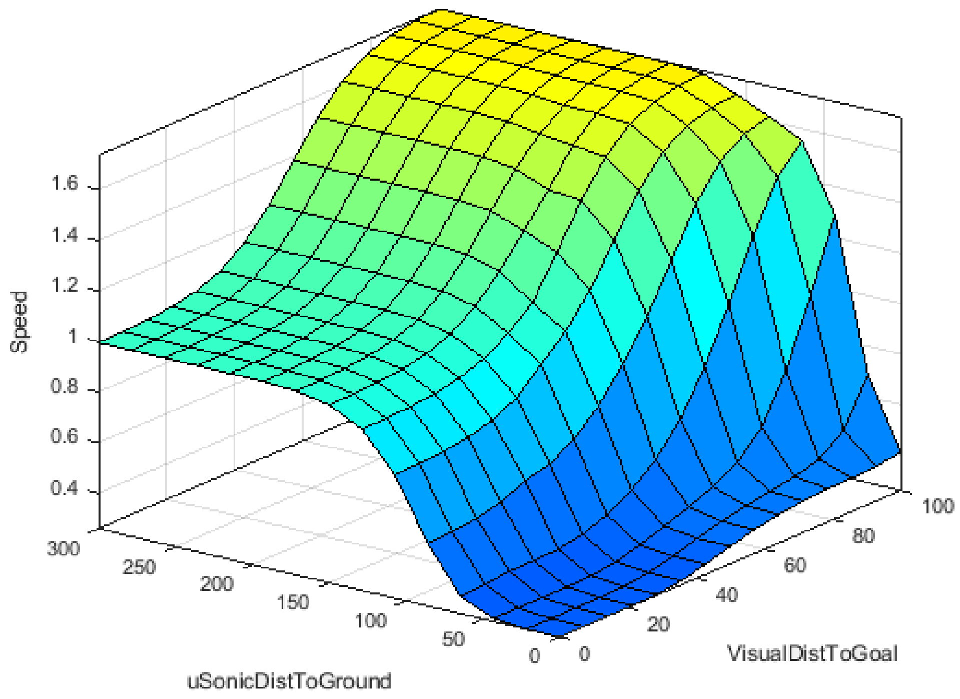


Figure 5.26: Multi-Modal Fuzzy Speed Controller outputs for translational velocity

5.3.5 Discussion

Unlike in a simulation where you may begin each iteration of the program initialised to any starting position, in real life the UAV needs to move randomly after it finds the central goal but just before it lands in order to replicate this action. The next learning episode now begins from a random physical location anywhere except directly over the helipad. Otherwise the UAV would find take off and land straight away because it goal has been found in the first perceived state that was obviously revisited without any searching necessary. The proposed GPFMORL algorithm is considered multi objective due to the fact that not only does it need to find the goal, but it should also avoid flying out of the visual localisation grid using as little power as possible. The afore mentioned application is similar to the MO Puddleworld benchmark where instead of avoiding the boundaries of the grid, the secondary objective is to avoid entering the puddles.

While the UAV may choose to explore by trying to find another way to the helipad by attempting to fly outside of the grid, this action should in fact be heavily punished due to potentially causing the UAV to waste power and possibly become lost or fly into unknown obstacles outside of the danger zone. When flying out of the danger zone the position of the UAV shall not change in simulation, however in real flight the UAV must move in order to reach the undesired state then move back whilst being punished. Whilst this is easy to achieve using simulation by simply ignoring the change of position when instructed to move outside of the grid, there must be some method for keeping the UAV within the bounds of the learning environment. To solve this issue a Potential Field Methodology has been implemented as previously described.

5.4 Automated Solar Powered Environmental Controller (ASPEC)

5.4.1 Introduction

To demonstrate the theoretical research outlined in this thesis, it was decided that an additional case study of CEA (Controlled Environment Agriculture) using an experimental ASPEC (Automated Solar Powered Environmental Controller) shall be developed and used for experimentation. The Experimental setup for this novel Embedded system is described in section 5.4.2 followed by the experimental results described in section 5.4.3. Finally section 5.4.4 concludes with a short discussion of the outcomes of the experiment and suggested future improvements for further development.

5.4.2 Experimental Setup

In order to facilitate the accurate perception of the nutrient solutions state on a real time basis, it was necessary to utilise both a PH Probe measuring the Potential of Hydrogen and an EC probe to measure the electrical conductivity of the nutrient solution. In addition to the internal sensors, the system also comprised of two temperature sensors to measure the external ambient air temperature and the internal nutrient reservoir solution temperature. The ambient air temperature was controlled by activating cooling fans and the nutrient solution temperature was controlled by a submersible air bubble stone, both regulated to optimal conditions via a closed loop system. The chosen crops to optimise their growth were 4 Sungold F1 Tomato plants using a hydroponic Nutrient Film Technique controlled by a bespoke embedded system called the ASPEC (Automated Solar Powered Environmental Controller). The experiment was carried out over a course of an extended grow cycle of 6 month (24 weeks) and with the exception of ensuring nutrient bottles were replenished, there was no manual intervention required throughout the whole growing cycle. The available actions available to the learning agent were to add nutrients to the solution, dilute the solution by adding water, increasing the PH with PH Up corrector (Strong Alkali) and finally decreasing the PH with PH down corrector (Strong Acid). The infinite supply of water was provided by a rainwater collection barrel stored under gravitational pressure and all the power needs were achieved by storing solar power from a small 40w solar panel into a 12v Lead acid battery.

5.4.3 Results

Presented in (Figure 5.27) are the graphical results for the ASPEC optimisation of EC & PH using Advanced Controlled Environment Agriculture of Hydroponic Tomato plants (Sungold F1). The two most important hydroponic parameters of EC & PH are optimised via a novel MORL algorithm implemented into an embedded system capable of performing a certain action when in specific states based upon an iterative learning process.

The X-Axis represent days of the growing cycle over an extended growing period of 6month, whilst the Y-axis represent both PH values, and EC values in the unit of micro–siemens (μS) . The two measurements of state information and the relevant actions are represented by the following coloured lines:

Electrical Conductivity (EC) - Blue Line

Potential of Hydrogen (PH) - Red Line

Addition of nutrient - Green Line

Dilution of nutrients - Purple Line

PH UP Corrector - Turquoise Line

PH Down Corrector - Orange Line

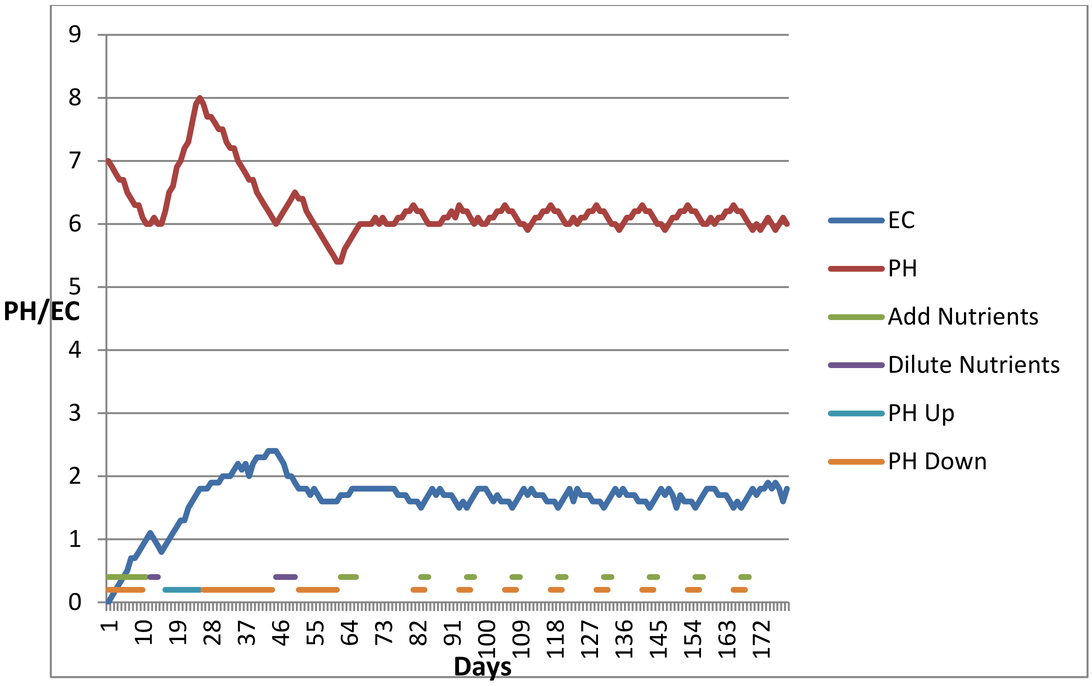


Figure 5.27: ASPEC Experimental Results Graph

The system begins by filling up the hydroponic reservoir with fresh water up to a volume of 25ltrs which shall be maintained throughout the entire grow cycle. During the first few weeks the plants are susceptible to high EC concentrations which can damage the plants and prevent growth. It is therefore it is recommended by the grow schedule that only half the dosage of nutrients are administered during the first two weeks to allow the plants to get acclimatise to a more concentrated dose later on. It can be observed on the graph (Figure 5.27) that during the first week, the EC state (Blue) gradually increases from 0 to 1.1 by choosing the action of Adding nutrients (Green). This overfeeding of nutrients is quickly reduced back down to the optimal recommended EC value of 0.8 by stopping the addition of nutrients and choosing to dilute the solution slightly. This dilution of nutrients (Purple) is performed by pumping some of the nutrient solution out of the reservoir therefore allowing a small amount of fresh water back in to maintain a constant volume of 25ltrs hence diluting the nutrient concentration. The PH of the fresh water entering the system initially has a natural value of 7 (PH neutral) and combined with the nutrients, should be reduced down to a slightly acidic value of PH 6 (Red). The PH state of the solution is corrected to PH 6 by first attempting the optimal action of adding PH down corrector to the solution.

After 2 weeks the optimal conditions are reached PH 6 and EC 0.8. As the nutrient dose is increased to full strength (EC1.8), which generally causes an increase in PH, the learning agent explores a non-optimal action of PH Up corrector which is administered causing the PH to dramatically increase to PH 8. The high alkalinity of the solution is now in danger of preventing optimal plant growth so the corrective measure is decided to try reducing the PH of the solution by executing the PH down corrector action. The PH is gradually adjusted to a more optimal solution of PH 6 but with adverse effects to the EC concentration. In adding large amounts of PH corrector, the undesired effect of the EC concentration is increased. To correct the EC the action of diluting the nutrient solution is decided which does in fact lower the EC but also has the effect of increasing the PH slightly due to the fresh water being neutral PH 7. Once again the action of PH down is used to reduce the value of the PH, however on this occasion the action is repeated for too long which results in a lower than optimal PH of 5.4. Now rather than using PH correctors as before, the learning agent decides to just increase the nutrient dosage action in hope that as a result of increasing the EC slightly, the PH will resolve to an optimal value of PH6 whilst also reaching an optimal EC. Once the majority of good actions in certain states have been repeated and the bad actions have learned to be avoided the graph begins to stabilise. The optimal policy that has been determined ensures the most suitable PH and EC by adding appropriate amount of nutrients and the PH down corrector whenever the state of the solution requires slight corrections.

5.4.4 Discussion

The proposed ASPEC case study confirms what was previously learned from manual control of hydroponic agriculture by converging to an almost identical method of optimisation learned entirely from its own interaction with the environment. Although the learning algorithm took several weeks to reach an optimal policy, it did so entirely from scratch without no prior knowledge of hydroponic agriculture. With respect to the several years it would take a human gardener to learn how to effectively grow hydroponic crops, this system did so in considerably less time. The learning time could possibly be increased by monitoring state information on an hourly basis rather than on a daily one at the cost of having to process much larger amounts of data over the course of a 6 month period. Such an approach would require the hardware to be designed with increased non-volatile memory capacity and a more powerful processor to deal with the increased amount of data and computation required for such a complex task. It is hoped that such a system shall become more accepted into the ever expanding general population to aid controlled environment agriculture of hydroponic fruit and vegetables for everyone to enjoy at a lower cost to both the consumer and environment.

5.5 Summary

This chapter presented four experiments used to test different aspects of the proposed GPFMORL method. The experiments performed include: Deterministic random walk, Stochastic random walk, UAV navigation for autonomous landing, Search and rescue application. Results and conclusion have been detailed for each experiment. Two other applications were considered for use with the GPFMORL algorithm and the hardware prototypes were developed such as the Autonomous Solar Programmable Environmental Controller (ASPEC) and Robotic Dementia Medication Administration System (RDMAS)

The next chapter provides a final conclusion and observations and then proposes some topics for future research.

Chapter 6

CONCLUDING DISCUSSION

6.1 Introduction

This work has explored the different advantages and limitations of three different popular paradigms, fuzzy logic systems, probabilistic theory, prospect theory and reinforcement learning separately. It was concluded that each of this paradigms could very well complement the drawbacks of the others and seamlessly work together in a cooperative rather than competitive way.

Fuzzy logic systems are good at generalizations, and due to its distinctive characteristics, it is able to handle non-statistical uncertainties, and fuzziness; however under certain conditions, the design and development of rather large or more complex systems can be too complicated for human operators.

Reinforcement learning methods have been an intense focus of research in the last decade. Research has proven that reinforcement learning can be successfully used in many different areas, such as decision making or control. A remarkable characteristic is that RL methods do not require input-output pairs for training or previous knowledge of the environment model. RL only uses sparse signal information in order to reach to optimal conclusions. Therefore using RL for automatic tuning of fuzzy logic parameters have also being the focus of recent research.

Probabilistic theory is still one of the most effectives way (and most explored) to deal with uncertainties, especially stochastic uncertainty. The fusion of probabilistic theory with fuzzy logic controllers have shown to be a powerful tool for practical areas such as finance and weather forecasting. Both paradigms can work in collaboration, in order to complement each other. As a result, probabilistic fuzzy logic system can handle a very large range of uncertainties.

The present work combines these three paradigms into a novel method, able to learn optimal policies for control or decision making whilst being resistant to stochastic, non-stochastic, uncertainties, randomness and fuzziness. Four different experiments were carried on to ratify our claims; a random walk within a fixed start and random start grid world and a windy hill world, and finally control of a wi-fi camera equipped quadcopter mobile robot.

The rest of this chapter is organized as follows; Section two highlights the contributions of this dissertation. In section three some important observations are described. Section four proposes some future work and in section five the final conclusions are given.

6.2 Contributions Of this Work

The contributions of this work can be summarised in the following list:

- This work proposed a novel modification to the GPFRL algorithm where, the implementation of a scalarisation methodology provided a way to find optimal policies for multiple objectives under different kinds of uncertainties in an automatic way.

- Through several different experiments of grid world variations, the proposed method has demonstrated high performance under situations of decision making taking into consideration risk.

- The observed results of this work showed that the GPFMORL method can find its major application in the un-calibrated control of non-linear, multiple inputs, and multiple output systems, especially in situations with high uncertainty.

- GPFMORL algorithm was compared in a grid world, with the multi objective extensions of well-established RL methods, MOSARSA and MOQ-learning. In this experiment the GPFMORL method showed not only a strong convergence but also a much faster one.

- PROSPECT THEORY was investigated as recommended by hinojosa and proved to be good at preventing the agent from risky situations and actualy improved the GPFMORL algortighms performance.

- The GPFMORL method was implemented on an AR-Drone 2.0 Quadcopter robot with the task of landing on the helipad, avoiding boundaries and traversing hills if necessary. The preliminary results of this heuristic based experiment were promising showing that the performance of the developed system is likely to be able to work well with real-time applications..

6.3 Observations

The following section presents some observations that were made during the development of this work.

- Due to wi-fi interference the characteristics of the AR-Drone 2.0 could often be random and erratic even when reducing network traffic.
- Due to the inherent limitations of ultrasonic proximity sensors, the processed value of altitude is often inaccurate and can lead to sudden increases or decreasing in altitude without prior warning.

- The frame rate of developed the image processing framework is more than sufficient for detecting the helipad even with erratic high speed manoeuvres.

6.4 Future Work

6.4.1 Theoretical Suggestions

Learning still is a black box in most of the cases. The presence of uncertainty and the concept of knowledge itself make it intrinsically difficult if not impossible to model and analyze the final behaviour of a learning system; therefore no system with unsupervised learning should be unsupervised. It is suggested the research of methods combining the probability estimation of reinforcement learning, as the one proposed in this work, with other methods that can incorporate the knowledge of human operators; especially for situation where the result of selecting actions or behaviours with a relatively "high" probability of success can have considerable consequences. As an example, it can be suggested the use of ***prospect theory****, which is a method of calculating decisions not only based on probabilities of success but also based on a risk evaluation.* " W. Hinojosa, "Probabilistic fuzzy logic framework in reinforcement learning for decision making," September, 2010.

Originally we began by creating an multi modal fuzzy altitude controller in order to deal with the non-statistical uncertainties present in detecting the environment. However since the vertical axis “Z” (altitude) of the UAV does not contribute to whatever positional state the UAV is in, it seemed necessary to alter the FIS. Instead of using ultra-sonic and area measurements of the helipad as inputs, now Distance to helipad and ultra-sonic measurements are used to create a fuzzy translational velocity controller. This facilitates smooth transition between states at the appropriate speed to reduce overshooting from one state to the next. Performing the obstacle detection manoeuvre combined with prospect theory, it is possible to distort the probabilities of success regarding risky situations. For example if the UAV detects a potential obstacle/hill to the left then the probability associated with moving left when in that state shall be reduced forcing the agent into a more explorative behaviour to take an more optimal alternate route in future.

A further improvement would be a fuzzy controller that uses 2 inputs of Roll & Pitch to determine how level the UAV was when it acquired the current state using visual detection. If the Roll and pitch is 0 in both cases then the probability of the UAV flying directly over the detected state is 1. However if either the roll or pitch is more than 0 then the probability of being in the detected state is reduced due to UAV possibly perceiving a differnet state from the perceived viewing angle. If the UAV is at its limits of 30^0 incline in either axis then the probability of the perceived state being correct is almost 0.

6.4.2 Practical Suggestions

Although mathematical models of dynamical systems can be accurate to a high degree, it doesn’t come close to real systems in terms of uncertainty. Therefore, it is highly suggested a more rigorous testing of the proposed GPFRL method on systems interacting with real environments, such as with mobile robots navigating on uncharted environments, highly nonlinear manipulators, etc.

Moreover, it is suggested the exploration on how this proposed algorithm performs on more challenging tasks, especially in systems with higher dimensional states. [1] W. Hinojosa, “Probabilistic fuzzy logic framework in reinforcement learning for decision making, ” September, 2010.

- Adding a omnidirectional camera to the UAV would improve perception
- Using a purpose built professional wide angle lens instead of retro fit lenses designed for smartphones.
- The translational velocities should be dramatically increased during landing in order to quickly react to the fast changing state information perceived from the environment.
- Optical flow algorithm to detect adjacent obstacles/hills by turning on the spot (yaw) to check the velocity of POI detected to determine obstructed regions.
- Improve resolution of downwards facing camera to improve helipad detection
- Either implement an digital image stabilisation algorithm or mount the camera on a 3 axis mechanical gimbal to reduce UAV in-place movements from affecting the state detection.

6.5 Final Conclusions

It is my conclusion after observing the results of the proposed algorithm, that the objectives of this dissertation have been successfully met. We can also anticipate the effective solution to more complex problems with more objectives using this algorithm. As any approach, there are limitations and shortcomings in the proposed algorithm, which should be improved further.

References

E. L. Thorndike, Animal Intelligence. Darien, CT, USA: Hafner, 1911.

A.M. Turing, "Computing machinery and intelligence," Mind, vol. 59, pp. 433–460, 1950

L. P.Kaelbling, M. L. Littman, and A.W. Moore, "Reinforcement learning: A survey," J. Artif. Intell.Res., vol. 4, pp. 237–285, 1996.

R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge,MA, USA: MIT Press, 1998

AirRobot. Quadrocopter UAV Model AR100. http:// www.airrobot-uk.com/index.htm.

Liu, C., Xu, X., & Hu, D. (2013). Multiobjective Reinforcement Learning: A Comprehensive Overview. ieeexplore.ieee.org, 1–13. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6509978

Gherega, A., Udrea, R. M., & Monica, R. (2012). A Q-Learning Approach to Decision Problems in Image Processing, (c), 60–66.

Veen, M. van der. (2011). Optimizing artificial force fields for autonomous drones in the pylon challenge using reinforcement learning.UvARescue and Amsterdam Oxford Joint Rescue Forces in the USARSim simulator. Retrieved from https://martijn-bsc-thesis.googlecode.com/files/Thesis.pdf

Yang, E., &Gu, D. (2005). A Survey on Multiagent Reinforcement Learning Towards Multi-Robot Systems.CIG, 2. Retrieved from http://cscourse.essex.ac.uk/cig/2005/papers/p1012.pdf

Yang, E., &Gu, D. (2004). Multiagent reinforcement learning for multi-robot systems: A survey. Department of Computer Science, Univeristy of …. Retrieved from http://computerscience.nl/docs/vakken/aibop/MAS-reinforcement-learning.pdf

XIE Li-juan, XIE Guang-rong, CHEN Huan-wen, and LI Xiao-li. A novel artificial potential field-based reinforcement learning for mobile

M. L. Littman, "Value-function reinforcement learning in markov games," Journal of Cognitive Systems Research, vol. 2, pp. 55–66, 2001.

Arkin, R. C., Endo, Y., Lee, B., Mackenzie, D., & Martinson, E. (n.d.). Multi Strategy Learning MethodsFor Multi robot Systems, (Lm).

Mataric, M. "Reward Functions for Accelerated Learning" in Machine Learning: Proc. Eleventh International Conference, San Francisco, CA, 1994, 181-189. 8.

Mitchell, T. (1990). *Machine Learning* (Vol. 4, pp. 417–433). doi:10.1146/annurev.cs.04.060190.002221

H. L. Liao, S. Member, Q. H. Wu, and S. Member, "Multi-Objective Optimisation by Reinforcement Learning," 2010.

E. Yang and D. Gu, “A Survey on Multiagent Reinforcement Learning Towards Multi-Robot Systems.,” CIG, vol. 2, 2005.

P. Vamplew, R. Dazeley, A. Berry, R.Issabekov, and E. Dekker, “Empirical evaluation methods for multiobjective reinforcement learning algorithms,” Mach. Learning, vol. 84, no. 1–2, pp. 51–80, 2011.

Y. Yang, Y. Tian, and H. Mei, “Cooperative Q Learning Based on Blackboard Architecture,” pp. 224–227, 2007.

G. Qian and X. Wang, “Greenhouse parameter control strategy based on the AHP(Analytical Hierarchy Process),” 2013 25th Chinese Control Decis. Conf., pp. 1589–1594, May 2013.

P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry, ”On the Limitations of Scalarisation for Multi-Objective Reinforcement Learning of Pareto Fronts”, Artificial Inteligence 2008, LNAI 5360, pp.372-378, 2008

Shaker, M., Smith, M. N. R., Yue, S., &Duckett, T. (2010). Vision-Based Landing of a Simulated Unmanned Aerial Vehicle with Fast Reinforcement Learning.2010 International Conference on Emerging Security Technologies, 183–188. doi:10.1109/EST.2010.14

Hinojosa, W. M., Nefti, S., &Kaymak, U. (2011). Systems Control With Generalized Probabilistic Fuzzy-Reinforcement Learning. IEEE Transactions on Fuzzy Systems, 19(1), 51–64. doi:10.1109/TFUZZ.2010.2081994

Kim, H., & Jordan, M. (2003). Autonomous helicopter flight via reinforcement learning. Advances in neural …. Retrieved from http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2003_CN07.pdf

Dijkshoorn, N. (2012). Simultaneous localization and mapping with the Ar. Drone (MASTERS THESIS). Amsterdam University. Retrieved from http://www.nickd.nl/dl/thesis_Nick_Dijkshoorn.pdf

Theodoridis, T., & Hu, H. (2012). Toward Intelligent Security Robots: A Survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(6), 1219–1230. doi:10.1109/TSMCC.2012.2198055

A. Castelletti, G. Corani, A. Rizzolli, R. Soncinie-Sessa, and E.Weber, “Reinforcement learning in the operational management of a water sys- tem,” in Proc. IFAC Workshop Model. Control Environ. Issues, Yoko- hama, Japan, 2002, pp. 325–330.

J. Karlsson, “Learning to solvemultiple goals,” Ph.D. dissertation, Dept. Comput.Sci., Univ. Rochester, Rochester, NY, USA, 1997.

D. C. K. Ngai and N. H. C. Yung, “A multiple goal reinforcement learn- ing method for complex vehicle overtaking maneuvers,” IEEE Trans. Intell. Trans. Syst., vol. 12, no. 2, pp. 509–522, Jun. 2011.

F. Sahba, H.R. Tizhoosh, and M. Salama, “Application of reinforcement learning for segmentation of transrectal ultrasound images”, Biomed Central Medical Imaging, 2008.

G. Tesauro, R. Das, H. Chan, J. O. Kephart, C. Lefurgy, D.W. Levine, and F. Rawson, “Managing power consumption and performance of computing systems using reinforcement learning,” in Advances in Neural Information Processing Systems. Cambridge, MA, USA: MIT Press, 2007.

M. Humphrys, “Action selection methods using reinforcement learning,” in From Animals to Animats 4, P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S.W.Wilson, Eds. Cambridge,MA, USA: MIT Press, 1996, pp. 134–144…

P. Abbeel, A. Coates, M. Quigley, and A. Ng.An application of reinforcement learning to aerobatic helicopter flight. In NIPS 19, 2007

Liu, Y., & Dai, Q. (2010). A survey of computer vision applied in aerial robotic vehicles. Optics Photonics and Energy Engineering (OPEE …, (201), 277–280. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5508131

L. G. Mitten, “Composition principles for synthesis of optimum multi- stage processes,” Oper. Res., vol. 12, pp. 610–619, 1964.

M. J. Sobel, “Ordinal dynamic programming,” Manage. Sci., vol. 21, pp. 967–975, 1975.

S. Mannor and N. Shimkin, “The steering approach for multi-criteria reinforcement learning,” in Proc. Adv. Neural Inf. Process. Syst., 2001, pp. 1563–1570.

Cherian, A., Andersh, J., Morellas, V., Papanikolopoulos, N., &Mettler, B. (2009).Autonomous altitude estimation of a UAV using a single onboard camera.2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 3900–3905. doi:10.1109/IROS.2009.5354307

K. Zheng, H. Li, R. C.Qiu, and S. Gong, “Multi-objective reinforcement learning based routing in cognitive radio networks:Walking in a random maze,” in Proc. Int. Conf. Comput. Netw.Commun., 2012, pp. 359–363.

S. Mannor and N. Shimkin, “A geometric approach to multi-criterion reinforcement learning,” J. Mach. Learning Res., vol. 5, pp. 325–360, 2004.

XIE Li-juan, XIE Guang-rong, CHEN Huan-wen, and LI Xiao-li. A novel artificial potential field-based reinforcement learning for mobile robotics in ambient intelligence. International Journal of Robotics and Automation, 2009.

L. Barrett and S.Narayanan, “Learning all optimal policies with multiple criteria,” in Proc. 25th Int. Conf.Mach. Learning, 2008, pp. 41–47.

C. R. Shelton, “Importance sampling for reinforcement learning with multiple objectives,” AI Laboratory,Massachusetts Institute of Technol- ogy, Cambridge,MA, USA, Tech. Rep. 2001-003, 2001.

A. Castelletti, F. Pianosi,&M.Restelli, “Tree-based fitted Q-iteration for multi-objective Markov decision problems,” in Proc. Int. Joint Conf. Neural Netw., 2012, pp. 1–8.

H. L. Liu and Q. H.Wu, “Multi-objective optimization by reinforcement learning,” in Proc. IEEE Congr. Evol.Comput., 2010, pp. 1–8.

S. Singh, T. Jaakkola, M. L. Littman et al, “Convergence results for single-step on-policy reinforcement-learning algorithms,” Machine Learning, vol. 39, pp. 287–308, 2000.

Matari, M. J., & Systems, C. (1997). Reinforcement Learning in the Multi-Robot Domain, 83, 73–83.

ROY, N., GORDON, G. & THRUN, S. 2005. Finding approximate POMDP solutions through belief compression. Journal of Artificial Intelligence Research, 23, 1-40.

DEMSPSTER, A. P. 1969. Upper and lower probability inferences for families of hypotheses with monotone density rations. Annals of Mathematical Statistics, 40, 953-969.

SUTTON, R. S. 1988. Learning to predict by the methods of temporal differences. Machine Learning, 3, 9-44.

RUMMERY, G. A. & NIRANJAN, M. 1994. On-line q-learning using connectionist systems. Cambridge: Cambridge University.

BARTO, A. G., SUTTON, R. S. & ANDERSON, C. W. 1983. Neuron-like adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics, 13, 835-846.

Watkins, C. J. C. H. 1989. Learning from delayed rewards.PhD, King's College.

BELLMAN, R. E. 1957. Dynamic programming, Princeton, New Jersey, Princeton University Press.

BARTO, A. G. & ANANDAN, P. 1985. Pattern recognizing stochastic learning automata. IEEE Transactions on Systems, Man and Cybernetics, SMC-15, 360-374.

CHEESEMAN, P. 1985. In defense of probability.9th International Joint Conference on Artificial Intelligence. Los Angeles, California: Morgan Kaufmann Publishers Inc.

WILLIAMS, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8, 229-256.

MORIARTY, D. E., SCHULTZ, A. C. & GREFENSTETTE, J. J. 1999. Evolutionary algorithms for reinforcement learning. Journal of Artificial Intelligence Research, 11, 241-276.

KALYANAKRISHNAN, S. & STONE, P. 2009. An empirical analysis of value function-based and policy search reinforcement learning. 8th International Conference on Autonomous Agents and Multiagent Systems. Budapest, Hungary: International Foundation for Autonomous Agents and Multiagent Systems.

BAIRD, L. C. 1999. Reinforcement learning through gradient descent.Doctor of Philosophy, Carnegie Mellon University.

HINOJOSA, W., NEFTI, S., GRAY, J. & KAYMAK, U. 2008. Reinforcement learning for probabilistic fuzzy controllers.International Conference on Control.Manchester, UK

LIN, C.-J. & XU, Y.-J. 2006. A novel genetic reinforcement learning for nonlinear fuzzy control problems. Neurocomputing, 69, 2078-2089

LIN, C.-T. & LEE, C. S. G. 1993.Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems.IEEE International Conference on Fuzzy Systems. San Francisco, CA, USA.

WANG, X.-S., CHENG, Y.-H. & YI, J.-Q. 2007. A fuzzy actor-critic reinforcement learning network. Information Sciences, 177, 3764-3781.

BARRICELLI, N. 1954. Esempinumerici di processi di evoluzione.Methodos, 45-68.

BERENJI, H. R. & KHEDKAR, P. 1992. Learning and tuning fuzzy logic controllers through reinforcements. IEEE Transactions on Neural Networks, 3, 724-740.

STRENS, M. 2000. A Bayesian framework for reinforcement learning.Proceeedings of the 17th International Conference on Machine Learning. Stanford University, California: ICML.

Watkins, C. J. C. H. & Dayan, P. 1992. Q-learning. Machine Learning, 8, 279-292.

SINGH, S., JAAKKOLA, T., LITTMAN, M. L. & ARI, C. S. 2000. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms.Machine Learning, 38.

A.G.Barto et al., “Neuron-like adaptive elements that can solve difficult learning control problems,” IEEE Trans. Syst,Man, Cybern.,vol.SMC-13,no.5,pp.835–846, 1983

C.W. Anderson, “Learning to control an inverted pendulum using neural networks,” IEEE Control Syst. Mag., vol. 9, no. 3, pp. 31–37, Apr. 1989.

H.R.Berenji and P.Khedkar, “Learning and tuning fuzzy logic controllers through reinforcements,” IEEETrans.NeuralNetw., vol. 3, no. 5, pp. 724– 740, Sep. 1992.

C.-C. Lee, “A self-learning rule-based controller employing approximate reasoning and neural net concept,” Int. J. Intell.Syst., vol. 6, no. 1, pp. 71–93, 1991.

C.-T. Lin and C. S. G. Lee, “Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems,” presented at the IEEE Int. Conf. Fuzzy Syst., San Francisco, CA, 1993.

C.-T. Lin, “A neural fuzzy control system with structure and parameter learning,” Fuzzy Sets Syst., vol. 70, no. 2/3, pp. 183–212, Mar. 1995.

C.-J. Lin and C.-T. Lin, “Reinforcement learning for an ART-based fuzzy adaptive learning control network,” IEEE Trans. Neural Netw.,vol.7, no. 3, pp. 709–731, May 1996.

R. J. Almeida and U. Kaymak, “Probabilistic fuzzy systems in value-at- risk estimation,” Int. J. Intell. Syst. Account., Finance Manag., vol. 16, no. 1/2, pp. 49–70, 2009.

L. Jouffe, “Fuzzy inference system learning by reinforcement methods,” IEEE Trans. Syst.,Man, Cybern., vol. 28, no. 3, pp. 238–255, Aug. 1998.

C.-K.Lin, “A reinforcement learning adaptive fuzzy controller for robots,” Fuzzy Sets Syst., vol. 137, no. 1, pp. 339–352, Aug. 2003.

X.-S.Wang et al., “A fuzzy actor–critic reinforcement learning network,” Inf. Sci., vol. 177, no. 18, pp. 3764–3781, Sep. 2007.

MOORE, A. W. & ATKESON, C. G. 1993. Prioritized sweeping: reinforcement learning with less data and less real time. Machine Learning, 13, 103-130.

SUTTON, R. S. & BARTO, A. G. 1987. A temporal-difference model of classical conditioning.9th Annual Conference of the Cognitive Science Society.Seattle,WA.

Morriss, F. H., Abramowitz, P. W., Nelson, S. P., Milavetz, G., Michael, S. L., Gordon, S. N., ... & Cook, E. F. (2009). Effectiveness of a barcode medication administration system in reducing preventable adverse drug events in a neonatal intensive care unit: a prospective cohort study. The Journal of pediatrics, 154(3), 363-368.

Danie Kahneman Amos Tversky: Prospect Theory: An Analysis of Decision under Risk;– Economica, Vol.47, No. 2 (Mar 1979), pp. 263-292

Thaler, R.H., Tversky, A., Kahneman, D., & Schwartz,A.(1997)The effect of myopia andloss aversion on risk taking: An experimental test. Quarterly Journal of Economics, 112, 647-661.

Andrew Chiu, George Wu (2010) Advances in Prospect Theory: Cumulative Representation of Uncertainty Amos Tversky, Daniel Kahneman – Journal Of Risk and Uncertainty 5:297-323 (1992)

Richard Gonzalez, George Wu. 1999. On the shape of the Probability Weighting Function Cognitive Psychology 38, 129-168 (1999)

Mohamed Abdellaoui 2000, Parameter-free Elicitation of Utilities and Probability Weitghting Function, Management Science 46, 1497-1512 (2000)

May Bunny (2012). Multi-criteria optimisation using Prospect Theory

Peter P.Wakker, Jose Luis Pinto, Han, Bleichrodt. Making Descriptive Use of Prospect Theory to improve the Perspective Use of Expected Utility– Management Science 47, 1498- 1514

Chris Warkins, P. Dayan. 1989. Q-Learning– Machine Learning 8 (3), 279-292(1989)

Frank MJ, Claus ED 2006. Anatomy of A Decision: striato-orbito frontal interactions In reinforcement learning, decision making and reversal– Psychol Rev 2006 Apr;113 :300-26

Chris Gaskett, Gordon Cheng. 2003. Online Learning of a motor map for humanoid robot reaching– Proceedings of theInternational l conference on computational intelligence, robotics and autionomous systems (2003)

Mohammad Hossein Fazel Zarandi, Javid Jouzdani, Maryam Fazel Zarandi, 2008. Reinforcement Learning for Fuzzy Control with Linguistic States. Journal of Uncertain Systems Vol.2, No.1, pp.54-66, 2008

Appendix

Appendix A

Multi Objective Random Walk Problem

MOQ-Learning

```
% -------------RL Parameters-------------
alpha = 0.4; % Step-size (learning rate)0.1 for windy
epsilon = 0.2; % e-Greedy behaviour for maximum stability
gamma = 0.95; % Discount Factor
totalPlays = 100; % Number of plays to average over
totalTrials = 40; % Trials per play

%-------------MORL Parameters-------------
NoOfObjectives = 3; % Specify any number of objectives
W1 = 0.4;
W2 = 0.3;
W3 = 0.3;

% ----------Enviroment Constants----------
gridSize = 10; % Size of the Grid
totalStates = gridSize*gridSize; % Total number of system states

% Starting Location (Fixed)
startM = 2;
startN = 2;

% Reward Location
rewardM = ceil(gridSize/2); %(center of the grid)y-axis
rewardN = ceil(gridSize/2)+1; %(center of the grid)x-axis

%Windy Grid World Enviroment Stochasticity
WindChance = 0.2; %20% chance of it being NotWindy, Windy or Very Windy

% ----------Variable Initialisation----------
totalNumSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all plays
totalNumBoundSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all
plays
totalNumHillSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all plays

% Out Of Boundary Counts
northCount = 0;
eastCount = 0;
southCount = 0;
westCount = 0;

S43hillCount = 0;
S53hillCount = 0;
S34hillCount = 0;
S35hillCount = 0;
S36hillCount = 0;

%-----------------------------------------------------------------
%Begin stepping through the grid world enviroment using MOQ-Learning
```

```
%-------------------------------------------------------------------
for j = 1 : totalPlays
% Store number of steps to reach reward for each trial
numSteps = zeros(totalTrials, 1);

% Store number of out of boundary steps (punishments)
numBoundSteps = zeros(totalTrials, 1);

% Store number of out of boundary steps (punishments)
   numHillSteps = zeros(totalTrials, 1);

% Initialize V(s) & Goal Reward
grid = zeros(gridSize, gridSize);
grid(rewardM, rewardN,1) = 1;
grid(rewardM, rewardN,2) = 1;
grid(rewardM, rewardN,3) = 1;

% Initialise Boundary Punishments -1 in all boundary states
for i=1 : gridSize
   grid(1,i,2)=-1;
   grid(gridSize,i,2)=-1;
   grid(i,1,2)=-1;
   grid(i,gridSize,2)=-1;
end

 %Initialise Hill Punishments
   grid(4,4,3) = -0.4; %State 34
   grid(4,5,3) = -0.6; %State 35
   grid(4,6,3) = -0.8; %State 36
   grid(5,3,3) = -0.1; %State 43
   grid(6,3,3) = -0.2; %State 53

   for i = 1 : totalTrials
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       % Random Starting Location Generator
%       stochGrid = rand(gridSize); %Generate 5x5 grid with random real numbers 0-1
%       vectorMax = max(stochGrid);% Return the maximum value of each column
%       big = max(vectorMax);%Determine maximum value of this vector of 5 values
%       %Delete semi colon after following line to display
%       %random start position
%       binaryMap = stochGrid==big; % Create a binary matrix indicating position of max value
%       [row_M,col_N] = find(stochGrid==big);
%       randomStartState = find(binaryMap'); % find the indice of the 5x5 grid (inverted to
comply with my notation)
%       randomStartState;
%
%       % Assign random co-ordinates to each trials starting position/state
%       startM = row_M;
%       startN = col_N;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Current position on grid is represented by m,n
    m = startM;
    n = startN;
    prevM = startM;
    prevN = startN;

    % Begin stepping through a path
    found = 0;
    while(found == 0)
      % increment # steps taken this trial
      numSteps(i) = numSteps(i) + 1;

      % pick an action and avoid going over grid borders
      N = m-1;
      if(N < 1)
        N = 1;
        northCount = northCount + 1;
        numBoundSteps(i) =  numBoundSteps(i) + 1;
      end
      E = n+1;
      if(E > gridSize)
        E = gridSize;
        eastCount = eastCount + 1;
        numBoundSteps(i) =  numBoundSteps(i) + 1;
      end
      S = m+1;
      if(S > gridSize)
        S = gridSize;
        southCount = southCount + 1;
        numBoundSteps(i) =  numBoundSteps(i) + 1;
      end
      W = n-1;
      if(W < 1)
        W = 1;
        westCount = westCount + 1;
        numBoundSteps(i) =  numBoundSteps(i) + 1;
      end


      % Perform an observation (using boxes system)
      stidx = ((m - 1) * gridSize) + n;
      X = zeros(totalStates, 1);
      X(stidx) = 1;
      disp(stidx);

      choices1 = [ grid(N,n,1), grid(m,E,1), grid(S,n,1), grid(m,W,1) ];
      choices2 = [ grid(N,n,2), grid(m,E,2), grid(S,n,2), grid(m,W,2) ];
```

```
choices3 = [ grid(N,n,3), grid(m,E,3), grid(S,n,3), grid(m,W,3) ];

indices1 = find(choices1 == max(choices1));
indices2 = find(choices2 == max(choices2));
indices3 = find(choices3 == max(choices3));

maximum1 = max(choices1);
maximum2 = max(choices2);
maximum3 = max(choices3);

%Detecting when agent has encountered an Obstacle/Hill
%State 43
if (m == 5 && n == 3)
     S43hillCount = S43hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 53
if (m == 6 && n == 3)
     S53hillCount = S53hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 34
if (m == 4 && n == 4)
     S34hillCount = S34hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 35
if (m == 4 && n == 5)
     S35hillCount = S35hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 36
if (m == 4 && n == 6)
     S36hillCount = S36hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

% choose the path with the maximum value, or explore
if((length(indices1) == 1) && (rand(1) - epsilon >= 0))
    index1 = indices1(1);
else
    index1 = ceil(length(choices1) * rand(1)); % randomly select direction
end

if((length(indices2) == 1) && (rand(1) - epsilon >= 0))
    index2 = indices2(1);
```

```
else
    index2 = ceil(length(choices2) * rand(1)); % randomly select direction
end

if((length(indices3) == 1) && (rand(1) - epsilon >= 0))
    index3 = indices3(1);
else
    index3 = ceil(length(choices3) * rand(1)); % randomly select direction
end


choice_rand = rand;
windy_rand = rand;

% disp(' ');
% disp(stidx);

%If random number is less than 0.2 (20% chance of shifting position)
if (windy_rand < WindChance)
    if (n == 1 && m>1)
        disp('N WAS 1, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end
    if (n == 2 && m>1)
        disp('N WAS 2, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end
    if (n == 3 && m>1)
        disp('N WAS 3, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 4 && m>1)
        disp('N WAS 4, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 5 && m>2)
        disp('N WAS 5, Wind Strength = 2');
        WindStrength = 2;
        m = m-WindStrength;
    end
    if (n == 6 && m>2)
        disp('N WAS 6, Wind Strength = 2');
        WindStrength = 2;
        m = m-WindStrength;
    end
    if (n == 7 && m>1)
```

```
        disp('N WAS 7, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 8 && m>1)
        disp('N WAS 8, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 9 && m>1)
        disp('N WAS 9, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end
    if (n == 10 && m>1)
        disp('N WAS 10, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end

     disp(' ');

else
    disp('NO WIND');

    %If random number is less than 0.4 (40% chance of choosing index 1)
    if(choice_rand < W1)
        index = index1;
    %If random number is less than 0.7 (30% chance of choosing index 2)
    elseif( choice_rand < W1+W2)
        index = index2;
    else
    %Otherwise there will be a (30% chance of choosing index 3)
        index = index3;
    end

    disp(' ');

end %end primary if

%Take action
if(index == 1) % grid(m-1,n)
    m = N;
elseif(index == 2) % grid(m,n+1)
    n = E;
elseif(index == 3) % grid(m+1,n)
    m = S;
elseif(index == 4) % grid(m,n-1)
    n = W;
else
```

```
            disp('error');
        end

        % stop after this step if the reward was found
        if(m == rewardM && n == rewardN)
            found = 1;
            disp(46);
        end

        %Expected discounted reward
        expDiscReward1 = gamma*grid(m,n,1);
        expDiscReward2 = gamma*grid(m,n,2);
        expDiscReward3 = gamma*grid(m,n,3);

        % calculate prediction error
        predErr1 = expDiscReward1 - grid(prevM,prevN,1);
        predErr2 = expDiscReward2 - grid(prevM,prevN,2);
        predErr3 = expDiscReward3 - grid(prevM,prevN,3);

        % V(s) = V(s) + alpha[r' + V(s') - V(s)]
        grid(prevM,prevN,1) = grid(prevM,prevN,1) + alpha*predErr1;
        grid(prevM,prevN,2) = grid(prevM,prevN,2) + alpha*predErr2;
        grid(prevM,prevN,3) = grid(prevM,prevN,3) + alpha*predErr3;

        grid(prevM,prevN,1) = (grid(prevM,prevN,1)*W1 + grid(prevM,prevN,2)*W2 +
grid(prevM,prevN,3)*W3)/NoOfObjectives;

        % update values for next step, and directional choices
        prevM = m;
        prevN = n;
      end
    end
    % Update storage matrices for across all plays
    totalNumSteps = totalNumSteps + numSteps;

     % Update storage matrices for across all plays
    totalNumBoundSteps =  totalNumBoundSteps + numBoundSteps;

    % Update storage matrices for across all plays
    totalNumHillSteps =  totalNumHillSteps + numHillSteps;

end

outOfBoundsCount = northCount + eastCount + southCount + westCount;
hillCount =  S43hillCount +  S53hillCount + S34hillCount +  S35hillCount +  S36hillCount;


%------------------------------------
% Plot 2D Graphs for both objectives
%------------------------------------
```

```
disp('Multi Objective Q-Learning Algorithm');
AverageBoundsCount = outOfBoundsCount/totalPlays
AverageHillCount = hillCount/totalPlays
disp(' ');

% Average number of steps across all plays
plot(totalNumSteps / totalPlays, 'color', 'k', 'LineStyle', '-');
title('Multi Objective Q-Learning Algorithm')
xlabel('NoOfTrials')
ylabel('NoOfSteps')
hold on

% Average number of boundary penaltys across all plays
plot(totalNumBoundSteps / totalPlays, 'color', 'k', 'LineStyle', '--');
hold on


% Average number of Hill penaltys across all plays
plot(totalNumHillSteps / totalPlays, 'color', 'k', 'LineStyle', ':');

legend('Obj1 Heli Goal','Obj2 Boundary','Obj3 Hills')
hold off
```

MOSARSA

```
% -------------RL Parameters-------------
alpha = 0.4; % Step-size
epsilon = 0.2; % e-Greedy behaviour for maximum stability
gamma = 0.95; % Discount
totalPlays = 100; % Number of plays to average over
totalTrials = 40; % Trials per play

%-------------MORL Parameters-------------
NoOfObjectives = 3; % Specify any number of objectives
W1 = 0.4;
W2 = 0.3;
W3 = 0.3;

% ----------Enviroment Constants----------
gridSize = 10;% Size of the Grid
totalStates = gridSize*gridSize; % Total number of system states

% Start Location (Fixed)
startM = 2;
startN = 2;

% Reward Location
rewardM = ceil(gridSize/2);%(center of the grid)y-axis
rewardN = rewardM+1;%(center of the grid)x-axis

%Windy Grid World Enviroment Stochasticity
WindChance = 0.2; %20% chance of it being NotWindy, Windy, Very Windy

% ----------Variable Initialisation----------
totalNumSteps = zeros(totalTrials, 1);% Total number of steps for each trial over all plays
totalNumBoundSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all
plays
totalNumHillSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all plays
%totalReward = zeros(totalTrials, 1); %not used

%Store the number of times that each boundary is crossed
northCount = 0;
eastCount = 0;
southCount = 0;
westCount = 0;

%Store the number of times that each Hill is crossed
S43hillCount = 0;
S53hillCount = 0;
S34hillCount = 0;
S35hillCount = 0;
S36hillCount = 0;
```

```
%-------------------------------------
% Start walking - SARSA
%-------------------------------------
for j = 1 : totalPlays
    % Store # steps for each trial
    numSteps = zeros(totalTrials, 1);

     % Store number of out of boundary steps
    numBoundSteps = zeros(totalTrials, 1);

     % Store number of out of hill steps
    numHillSteps = zeros(totalTrials, 1);

    % Initialize V(s)
    grid = zeros(gridSize, gridSize,3);
    grid(rewardM, rewardN,1) = 1;
    grid(rewardM, rewardN,2) = 1;
    grid(rewardM, rewardN,3) = 1;

    %bondaries punishement
        for i = 1 : gridSize
           % grid(1,i,1) = -1;
            grid(1,i,2) = -1;
            %grid(10,i,1) = -1;
            grid(10,i,2) = -1;
            %grid(i,1,1) = -1;
            grid(i,1,2) = -1;
            %grid(i,10,1) = -1;
            grid(i,10,2) = -1;
        end

    %obstacles

    %state 34 35 36 43 53
    grid(4,4,3) = -0.4; %34
    grid(4,5,3) = -0.6; %35
    grid(4,6,3) = -0.8; %36
    grid(5,3,3) = -0.1; %43
    grid(6,3,3) = -0.2; %53

    for i = 1 : totalTrials

%         % Random Starting Location
%                   stochGrid = rand(gridSize); %Generate 5x5 grid with random real numbers 0-1
%                   vectorMax = max(stochGrid);% Return the maximum value of each column
%                   big = max(vectorMax);%Determine maximum value of this vector of 5 values
%                  %Delete semi colon after following line to display
%                  %random start position
%                   binaryMap = stochGrid==big; % Create a binary matrix indicating position of
max value
```

```
%                  [row_M,col_N] = find(stochGrid==big);
%                  randomStartState = find(binaryMap'); % find the indice of the 5x5 grid (inverted
to comply with my notation)
%                  randomStartState;

%         %  end
%
%         %  r = randi([1 10],1,1)
%
%                  % Assign random co-ordinates to each trials starting position/state
%                  startM = row_M;
%                  startN = col_N;
%

      % Current position on grid is represented by m,n
      m = startM;
      n = startN;
      prevM = startM;
      prevN = startN;
      % Begin stepping through a path
      found = 0;
      while(found == 0)
         % increment # steps taken this trial
         numSteps(i) = numSteps(i) + 1;
         % pick an action and avoid going over grid borders
         N = m-1;
         if(N < 1)
            N = 1;
            northCount = northCount + 1;
            numBoundSteps(i) =  numBoundSteps(i) + 1;
         end
         E = n+1;
         if(E > gridSize)
            E = gridSize;
            eastCount = eastCount + 1;
            numBoundSteps(i) =  numBoundSteps(i) + 1;
         end
         S = m+1;
         if(S > gridSize)
            southCount = southCount + 1;
            numBoundSteps(i) =  numBoundSteps(i) + 1;
            S = gridSize;
         end
         W = n-1;
         if(W < 1)
            W = 1;
            westCount = westCount + 1;
            numBoundSteps(i) =  numBoundSteps(i) + 1;
         end
```

```
  % Perform an observation (using boxes system)
stidx = ((m - 1) * gridSize) + n;
X = zeros(totalStates, 1);
X(stidx) = 1;
disp(stidx);

choices1 = [ grid(N,n,1), grid(m,E,1), grid(S,n,1), grid(m,W,1) ];
choices2 = [ grid(N,n,2), grid(m,E,2), grid(S,n,2), grid(m,W,2) ];
choices3 = [ grid(N,n,3), grid(m,E,3), grid(S,n,3), grid(m,W,3) ];
% choose the path with the maximum value, or explore

indices1 = find(choices1 == max(choices1));
indices2 = find(choices2 == max(choices2));
indices3 = find(choices3 == max(choices3));


maximum1 = max(choices1);
maximum2 = max(choices2);
maximum3 = max(choices3);

%Detecting when agent has encountered an Obstacle/Hill
%State 43
if (m == 5 && n == 3)
    S43hillCount = S43hillCount + 1;
   numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 53
if (m == 6 && n == 3)
    S53hillCount = S53hillCount + 1;
   numHillSteps(i) =  numHillSteps(i) + 1;
end

  %State 34
if (m == 4 && n == 4)
    S34hillCount = S34hillCount + 1;
   numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 35
if (m == 4 && n == 5)
    S35hillCount = S35hillCount + 1;
   numHillSteps(i) =  numHillSteps(i) + 1;
end

  %State 36
if (m == 4 && n == 6)
    S36hillCount = S36hillCount + 1;
   numHillSteps(i) =  numHillSteps(i) + 1;
end
```

```
if((length(indices1) == 1) && (rand(1) - epsilon >= 0))
   index1 = indices1(1);
else
   index1 = ceil(length(choices1) * rand(1)); % randomly select direction
end

if((length(indices2) == 1) && (rand(1) - epsilon >= 0))
   index2 = indices2(1);
else
   index2 = ceil(length(choices2) * rand(1)); % randomly select direction
end

if((length(indices3) == 1) && (rand(1) - epsilon >= 0))
   index3 = indices3(1);
else
   index3 = ceil(length(choices3) * rand(1)); % randomly select direction
end

 %If random number is less than 0.5 (50% chance of choosing index 1)
%If random number is less than 0.2 (20% chance of choosing index 1)
choice_rand = rand;
windy_rand = rand;

disp(' ');
% disp(stidx);

if (windy_rand < WindChance)

   if (n == 1 && m>1)
      disp('N WAS 1, Wind Strength = 0');
      WindStrength = 0;
      m = m-WindStrength;
   end
   if (n == 2 && m>1)
      disp('N WAS 2, Wind Strength = 0');
      WindStrength = 0;
      m = m-WindStrength;
   end
   if (n == 3 && m>1)
      disp('N WAS 3, Wind Strength = 1');
      WindStrength = 1;
      m = m-WindStrength;
   end
   if (n == 4 && m>1)
      disp('N WAS 4, Wind Strength = 1');
      WindStrength = 1;
      m = m-WindStrength;
   end
   if (n == 5 && m>2)
```

```
        disp('N WAS 5, Wind Strength = 2');
        WindStrength = 2;
        m = m-WindStrength;
    end
    if (n == 6 && m>2)
        disp('N WAS 6, Wind Strength = 2');
        WindStrength = 2;
        m = m-WindStrength;
    end
    if (n == 7 && m>1)
        disp('N WAS 7, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 8 && m>1)
        disp('N WAS 8, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 9 && m>1)
        disp('N WAS 9, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end
    if (n == 10 && m>1)
        disp('N WAS 10, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end

    % disp(' ');

else
    disp('NO WIND');

    if(choice_rand < W1)
        index = index1;
    elseif( choice_rand < W1+W2)
        index = index2;
    else
        index = index3;
    end

     disp(' ');

end %end primary if

%Take action
if(index == 1) % grid(m-1,n)
    m = N;
```

```
         elseif(index == 2) % grid(m,n+1)
            n = E;
         elseif(index == 3) % grid(m+1,n)
            m = S;
         elseif(index == 4) % grid(m,n-1)
            n = W;
         else
            disp('error');
         end
         % stop after this step if the reward was found
         if(m == rewardM && n == rewardN)
            found = 1;
            disp(46);
         end
         %Expected discounted reward
         expDiscReward1 = gamma*grid(m,n,1);
         expDiscReward2 = gamma*grid(m,n,2);
         expDiscReward3 = gamma*grid(m,n,3);
         % calculate prediction error
         predErr1 = expDiscReward1 - grid(prevM,prevN,1);
         predErr2 = expDiscReward2 - grid(prevM,prevN,2);
         predErr3 = expDiscReward3 - grid(prevM,prevN,3);
         % V(s) = V(s) + alpha[r' + V(s') - V(s)]
         grid(prevM,prevN,1) = grid(prevM,prevN,1) + alpha*predErr1;
         grid(prevM,prevN,2) = grid(prevM,prevN,2) + alpha*predErr2;
         grid(prevM,prevN,3) = grid(prevM,prevN,3) + alpha*predErr3;

          grid(prevM,prevN,1) = (grid(prevM,prevN,1)*W1 + grid(prevM,prevN,2)*W2 +
grid(prevM,prevN,3)*W3)/NoOfObjectives;
         % update values for next step, and directional choices
         prevM = m;
         prevN = n;
      end
   end
   % Update storage matrices for across all plays
   totalNumSteps = totalNumSteps + numSteps;

    % Update storage matrices for across all plays
   totalNumBoundSteps =  totalNumBoundSteps + numBoundSteps;

   % Update storage matrices for across all plays
   totalNumHillSteps =  totalNumHillSteps + numHillSteps;

end

outOfBoundsCount = northCount + eastCount + southCount + westCount;
hillCount =  S43hillCount +  S53hillCount + S34hillCount +  S35hillCount +  S36hillCount;


%-----------------------------------
```

```
% Plot 2D Graphs for all objectives
%------------------------------------
disp('Multi Objective SARSA Algorithm');
AverageBoundsCount = outOfBoundsCount/totalPlays
AverageHillCount = hillCount/totalPlays
disp(' ');

% Average number of steps across all plays
plot(totalNumSteps / totalPlays, 'color', 'k', 'LineStyle', '-');
title('Multi Objective SARSA Algorithm')
xlabel('NoOfTrials')
ylabel('NoOfSteps')
hold on

% Average number of boundary penaltys across all plays
plot(totalNumBoundSteps / totalPlays, 'color', 'k', 'LineStyle', '--');
hold on


% Average number of Hill penaltys across all plays
plot(totalNumHillSteps / totalPlays, 'color', 'k', 'LineStyle', ':');

legend('Obj1 Heli Goal','Obj2 Boundary','Obj3 Hills')
hold off
```

GPFMORL

```
%GPFMORL Complex Enviroment with 3 Objectives
%7th August Start & Goal Env Test Exp1 (100plays)

%-------------RL Constants---------------
alpha = 0.003; % Actor learning rate
beta = 0.005; % Critic learning rate
epsilon = 0.2; % Exploration/explotation control parameter
gamma = 0.95; % Discount factor
totalActions = 4; % Number of possible actions
totalAgents = totalActions; % Total number of learning agents

totalPlays = 100; % Number of plays to average over (was 100) works at 1
totalTrials = 40; % Trials per play (was 40) works at 20

% ----------Enviroment Constants---------
gridSize = 10; % Size of the Grid (was 10)
totalStates = gridSize*gridSize; % Total number of system states
rewardM = ceil(gridSize/2); % Location of reward in M axe
rewardN = ceil(gridSize/2)+1; % Location of reward in N axe

startM = 2; % Location to start from in M axe
startN = 2; % Location to start from in N axe

% ---------Variable Initialisation--------
totalNumSteps = zeros(totalTrials, 1);% Total # steps for each trial over all plays
totalReward = zeros(totalTrials, totalAgents);
totalNumBoundSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all
plays
totalNumHillSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all plays

%-------------MORL Variables-------------
NoOfObjectives = 3; % Specify any number of objectives
W1 = 0.4; %Objective 1 weighting (reaching the helipad goal) Best@1
W2 = 0.3; %Objective 2 weighting (Avoiding the grid boundarys) Best@1
W3 = 0.3;

%Windy Grid World Enviroment Stochasticity
WindChance = 0.2; %20% chance of it being NotWindy, Windy, Very Windy


%Store the number of times that each boundary is crossed (for plot)
northCount = 0;
eastCount = 0;
southCount = 0;
westCount = 0;

S43hillCount = 0;
S53hillCount = 0;
```

```
S34hillCount = 0;
S35hillCount = 0;
S36hillCount = 0;

% RandomStartList= zeros(24, 1);
% RandomStartList(1, 1) = 13;
% RandomStartList(2, 1) = 14;
% RandomStartList(3, 1) = 15;
% RandomStartList(4, 1) = 16;
% RandomStartList(5, 1) = 17;
% RandomStartList(6, 1) = 18;
% RandomStartList(7, 1) = 19;
% RandomStartList(8, 1) = 23;
% RandomStartList(9, 1) = 29;
% RandomStartList(10, 1) = 33;
% RandomStartList(11, 1) = 39;
% RandomStartList(12, 1) = 43;
% RandomStartList(13, 1) = 49;
% RandomStartList(14, 1) = 53;
% RandomStartList(15, 1) = 59;
% RandomStartList(16, 1) = 63;
% RandomStartList(17, 1) = 69;
% RandomStartList(18, 1) = 73;
% RandomStartList(19, 1) = 74;
% RandomStartList(20, 1) = 75;
% RandomStartList(21, 1) = 76;
% RandomStartList(22, 1) = 77;
% RandomStartList(23, 1) = 78;
% RandomStartList(24, 1) = 79;
%
%
%  r = randi([1 26],1,1);
%  randomStartPoint = RandomStartList(r, 1);
% randomStartPoint

%---------------------------------------------------------------
% Begin stepping through the grid world enviroment using GPFMORL
%---------------------------------------------------------------
for j = 1 : totalPlays
   % Store # steps for each trial
   numSteps = zeros(totalTrials, 1);

   % Store number of out of boundary steps (punishments)
   numBoundSteps = zeros(totalTrials, 1);


       % Store number of out of boundary steps (punishments)
   numHillSteps = zeros(totalTrials, 1);

   % Initializing actor and critic weights
```

```
v = zeros(totalStates, totalAgents, 3);
w = zeros(totalStates, totalAgents,3);

%Northern & Eastern Boundary Penalty Association 10x10
for ii = 1 : gridSize
   v(ii,1,2) = -1;
   v(ii*10,2,2) = -1;
end

%Southern Boundary Penalty Association 10x10
for jj=91 : 100
   v(jj,3,2) = -1;
end

%Western Boundary Penalty Associationv 10x10
for hh=1 : gridSize
   v((hh*10)-9,4,2) = -1;
end




%reward at astate 46 CHANGED BECAUSE WAS WRONG
v(45,2,1) = 1;
v(45,2,2) = 1;
w(45,2,1) = 1;
w(45,2,2) = 1;
w(45,2,3) = 1;
v(45,2,3) = 1;

v(56,1,1) = 1;
v(56,1,2) = 1;
w(56,1,1) = 1;
w(56,1,2) = 1;
w(56,1,3) = 1;
v(56,1,3) = 1;

v(36,3,1) = 1;
v(36,3,2) = 1;
w(36,3,1) = 1;
w(36,3,2) = 1;
w(36,3,3) = 1;
v(36,3,3) = 1;

v(47,4,1) = 1;
v(47,4,2) = 1;
w(47,4,1) = 1;
w(47,4,2) = 1;
w(47,4,3) = 1;
v(47,4,3) = 1;
```

```
%building danger zone that would cost more to cross

%state 34

%comimg from west
v(33,2,3) =  -0.4;
w(33,2,3) =  -0.4;

%coming from south
v(44,1,3) =  -0.4;
w(44,1,3) =  -0.4;

%coming from north
v(24,3,3) =  -0.4;
w(24,3,3) = -0.4;

%coming from east
v(35,4,3) =  -0.4;
w(35,4,3) =  -0.4;

%state 35

%comimg from west
v(34,2,3) =  -0.6;
w(34,2,3) =  -0.6;

%coming from south
v(45,1,3) =  -0.6;
w(45,1,3) =  -0.6;

%coming from north
v(25,3,3) =  -0.6;
w(25,3,3) =  -0.6;

%coming from east
v(36,4,3) =  -0.6;
w(36,4,3) =  -0.6;

%state 36

%comimg from west
v(35,2,3) =  -0.8;
w(35,2,3) = -0.8;

%coming from south
v(46,1,3) =  -0.8;
w(46,1,3) =  -0.8;

%coming from north
```

```
v(26,3,3) =  -0.8;
w(26,3,3) =  -0.8;

%coming from east
v(37,4,3) =  -0.8;
w(37,4,3) = -0.8;

%state 43

%comimg from west
v(42,2,3) = -0.1;
w(42,2,3) = -0.1;

%coming from south
v(53,1,3) = -0.1;
w(53,1,3) = -0.1;

%coming from north
v(33,3,3) = -0.1;
w(33,3,3) = -0.1;

%coming from east
v(44,4,3) = -0.1;
w(44,4,3) = -0.1;

%state 53

%comimg from west
v(52,2,3) = -0.2;
w(52,2,3) = -0.2;

%coming from south
v(63,1,3) = -0.2;
w(63,1,3) = -0.2;

%coming from north
v(43,3,3) = -0.2;
w(43,3,3) = -0.2;

%coming from east
v(54,4,3) =  -0.2;
w(54,4,3) =  -0.2;


for i = 1 : totalTrials
    disp(j+(i/100))
    % while (randomStartState ~= 0 || 13 || 14 || 15 || 16 || 17 || 18 || 19 || 23 || 29 || 33 || 39 ||
43 || 49 || 53 || 59 || 63 || 69 || 73 || 74 || 75 || 76 || 77 || 78 || 79)


    %       % Random Starting Location
    %           stochGrid = rand(gridSize); %Generate 5x5 grid with random real numbers 0-1
    %           vectorMax = max(stochGrid);% Return the maximum value of each column
    %           big = max(vectorMax);%Determine maximum value of this vector of 5 values
    %           binaryMap = stochGrid==big % Create a binary matrix indicating position of max
value
    %           [row_M,col_N] = find(stochGrid==big);
    %           randomStartState = find(binaryMap'); % find the indice of the 5x5 grid (inverted
to comply with my notation)
    %           randomStartState



    %  end


    %  r = randi([1 10],1,1)

    %           % Assign random co-ordinates to each trials starting position/state
    %           startM = row_M;
    %           startN = col_N;

    % Current position on grid is represented by m,n
    m = startM;
    n = startN;
    prevM = startM;
    prevN = startN;

    % % Clean some variables
    % p = zeros(totalAgents,1);
    % oldp = p;

    % Clean some variables
    p = zeros(totalAgents,3);
    oldp(:,1) = p(:,1);
    oldp(:,2) = p(:,2);
    oldp(:,3) = p(:,3);

    % Begin stepping through a path
    found = 0;
    while(found == 0)

      % Increment # steps taken this trial
      numSteps(i) = numSteps(i) + 1;
```

```
% Find probabilities from actor weights
sumWeights = zeros(totalAgents,1);
sumWeights2 = zeros(totalAgents,1);
sumWeights3 = zeros(totalAgents,1);
ro = zeros(totalStates, totalAgents);
ro2 = zeros(totalStates, totalAgents);
ro3 = zeros(totalStates, totalAgents);
for agent = 1 : totalAgents
   for state = 1 : totalStates
      %ro(state, agent) = logsig(w(state, agent));
      ro(state, agent) = logsig(w(state, agent,1));
      ro2(state, agent) = logsig(w(state, agent, 2));
      ro3(state, agent) = logsig(w(state, agent, 3));
      sumWeights(agent) = sumWeights(agent) + ro(state, agent);
      sumWeights2(agent) = sumWeights2(agent) + ro2(state, agent);
      sumWeights3(agent) = sumWeights3(agent) + ro3(state, agent);
   end
end
% Perform an observation (using boxes system)
stidx = ((m - 1) * gridSize) + n;
X = zeros(totalStates, 1);
X(stidx) = 1;

% Finding final probability
P = zeros(totalAgents, 1);
P2 = zeros(totalAgents, 1);
P3 = zeros(totalAgents, 1);
for agent = 1 : totalAgents
   P(agent) = ro(stidx, agent);
   P2(agent) = ro2(stidx, agent);
   P3(agent) = ro3(stidx, agent);
end
% Creating action set avoiding grid borders
%N is the new m axis position after moving North
N = m - 1;
if(N < 1)
   N = 1;
   northCount = northCount + 1;
   numBoundSteps(i) =  numBoundSteps(i) + 1;
end

E = n + 1;
if(E > gridSize)
   E = gridSize;
   eastCount = eastCount + 1;
   numBoundSteps(i) =  numBoundSteps(i) + 1;
end

S = m + 1;
if(S > gridSize)
```

```
    southCount = southCount + 1;
    numBoundSteps(i) =  numBoundSteps(i) + 1;
    S = gridSize;
end

W = n - 1;
if(W < 1)
    W = 1;
    westCount = westCount + 1;
    numBoundSteps(i) =  numBoundSteps(i) + 1;
end



%Detecting when agent has encountered an Obstacle/Hill
%State 43
if (m == 5 && n == 3)
     S43hillCount = S43hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 53
if (m == 6 && n == 3)
     S53hillCount = S53hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

   %State 34
if (m == 4 && n == 4)
     S34hillCount = S34hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 35
if (m == 4 && n == 5)
     S35hillCount = S35hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

   %State 36
if (m == 4 && n == 6)
     S36hillCount = S36hillCount + 1;
    numHillSteps(i) =  numHillSteps(i) + 1;
end

% Initializing values
Nu = zeros(totalAgents, 1);
Nu2 = zeros(totalAgents, 1);
Nu3 = zeros(totalAgents, 1);
stdev = zeros(totalAgents, 1);
```

```
        stdev2 = zeros(totalAgents, 1);
        stdev3 = zeros(totalAgents, 1);

        % Stochastic random number generator
        for agent = 1 : totalAgents
            %stdev(agent) = 2*logsig(p(agent))-1;
            stdev(agent) = 2*logsig(p(agent,1))-1;
            stdev2(agent) = 2*logsig(p(agent,2))-1;
            stdev3(agent) = 2*logsig(p(agent,3))-1;
            Nu(agent) = stdev(agent)*randn;
            Nu2(agent) = stdev2(agent)*randn;
            Nu3(agent) = stdev3(agent)*randn;
            if Nu(agent) >= 1
                Nu(agent) = 1;
            elseif Nu(agent) <= -1
                Nu(agent) = -1;
            end

            if Nu2(agent) >= 1
                Nu2(agent) = 1;
            elseif Nu2(agent) <= -1
                Nu2(agent) = -1;
            end

            if Nu3(agent) >= 1
                Nu3(agent) = 1;
            elseif Nu3(agent) <= -1
                Nu3(agent) = -1;
            end

        end
        %troubles >> we need to evaluate both choices (obj1 & obl2) and decide with the
scalarized values
        % Finding choices (Probability + explotation/exploration) FOR OBJECTIVE 1
        choices = zeros(totalAgents, 1);
        for agent=1:totalAgents
            choices(agent) = P(agent)+Nu(agent);
        end

        % Finding choices (Probability + explotation/exploration) FOR OBJECTIVE 2
        choices2 = zeros(totalAgents, 1);
        for agent=1:totalAgents
            choices2(agent) = P2(agent)+Nu2(agent);
        end

        choices3 = zeros(totalAgents, 1);
        for agent=1:totalAgents
            choices3(agent) = P3(agent)+Nu3(agent);
        end
```

```
    %here, we need to evaluate both choice from the point of view of each
    %objective, basivally, what we do is applying twice a single objective
    %action selection. for the moment, we use a greedy selection

    %we choose an index for objective 1
    % Choose the path with the maximum probability, or explore
    indices = find(choices == max(choices));
    maximum = max(choices);
    if((length(indices) == 1) && (maximum >= epsilon))
       index1 = indices(1);
    else
       index1 = ceil(length(choices) * rand); % randomly select direction
    end

    %we choose an index for objective 2
    % Choose the path with the maximum probability, or explore
    indices2 = find(choices2 == max(choices2));
    maximum2 = max(choices2);
    if((length(indices2) == 1) && (maximum2 >= epsilon))
       index2 = indices2(1);
    else
       index2 = ceil(length(choices2) * rand); % randomly select direction
    end

    indices3 = find(choices3 == max(choices3));
    maximum3 = max(choices3);
    if((length(indices3) == 1) && (maximum3 >= epsilon))
       index3 = indices3(1);
    else
       index3 = ceil(length(choices3) * rand); % randomly select direction
    end


%If random number is less than 0.5 (50% chance of choosing index 1)
    %If random number is less than 0.2 (20% chance of choosing index 1)
    choice_rand = rand;
    windy_rand = rand;

    disp(' ');
    disp(stidx);

    if (windy_rand < WindChance)



       if (n == 1 && m>1)
          disp('N WAS 1, Wind Strength = 0');
          WindStrength = 0;
          m = m-WindStrength;
       end
```

```
        if (n == 2 && m>1)
            disp('N WAS 2, Wind Strength = 0');
            WindStrength = 0;
            m = m-WindStrength;
        end
        if (n == 3 && m>1)
            disp('N WAS 3, Wind Strength = 1');
            WindStrength = 1;
            m = m-WindStrength;
        end
        if (n == 4 && m>1)
            disp('N WAS 4, Wind Strength = 1');
            WindStrength = 1;
            m = m-WindStrength;
        end
        if (n == 5 && m>2)
            disp('N WAS 5, Wind Strength = 2');
            WindStrength = 2;
            m = m-WindStrength;
        end
        if (n == 6 && m>2)
            disp('N WAS 6, Wind Strength = 2');
            WindStrength = 2;
            m = m-WindStrength;
        end
        if (n == 7 && m>1)
            disp('N WAS 7, Wind Strength = 1');
            WindStrength = 1;
            m = m-WindStrength;
        end
        if (n == 8 && m>1)
            disp('N WAS 8, Wind Strength = 1');
            WindStrength = 1;
            m = m-WindStrength;
        end
        if (n == 9 && m>1)
            disp('N WAS 9, Wind Strength = 0');
            WindStrength = 0;
            m = m-WindStrength;
        end
        if (n == 10 && m>1)
            disp('N WAS 10, Wind Strength = 0');
            WindStrength = 0;
            m = m-WindStrength;
        end


        disp(' ');

    else
    disp('NO WIND');

    if(choice_rand < W1)
        index = index1;
    elseif( choice_rand < W1+W2)
        index = index2;
    else
        index = index3;
    end

     disp(' ');

end %end primary if

%Take action
r=zeros(totalAgents,1);
if(index == 1) % grid(m-1,n)
    m = N;
    r(index) = sign(m-rewardM);
elseif(index == 2) % grid(m,n+1)
    n = E;
    r(index) = sign(rewardN-n);
elseif(index == 3) % grid(m+1,n)
    m = S;
    r(index) = sign(rewardM-m);
elseif(index == 4) % grid(m,n-1)
    n = W;
    r(index) = sign(n-rewardN);
else
    disp('error random number too big');
end

% Stop after this step if the reward was found
if(m == rewardM && n == rewardN)
    found = 1;
    disp(46);
end

% Saving prediction information
oldp(:,1) = p(:,1);
oldp(:,2) = p(:,2);
oldp(:,3) = p(:,3);
p = zeros(totalAgents,3);


% % Computing the prediction of eventual reinforcement
% for agent = 1:totalAgents
% p(agent) = v(stidx,agent);
% end
```

```
        % Computing the prediction of eventual reinforcement
        for agent = 1:totalAgents
           p(agent,1) = v(stidx,agent,1);
           p(agent,2) = v(stidx,agent,2);
           p(agent,3) = v(stidx,agent,3);
        end
        % Computing the prediction error using temporal differences method
        rbar=zeros(totalAgents,1);
        rbar2=zeros(totalAgents,1);
        rbar3=zeros(totalAgents,1);
        for agent = 1:totalAgents
           %predErr = gamma*p(agent) - oldp(agent); %
           predErr = gamma*p(agent,1) - oldp(agent,1); %
           predErr2 = gamma*p(agent,2) - oldp(agent,2);
           predErr3 = gamma*p(agent,3) - oldp(agent,3);
           %rbar(agent) = r(agent) + predErr;
           rbar(agent) = r(agent) + predErr;
           rbar2(agent) = r(agent) + predErr2;
           rbar3(agent) = r(agent) + predErr3;
        end

        stateF = stidx;
        agentF = index;
        % Learning thev value functions
        %for agent = 1 : totalAgents
        %for state = 1 : totalStates

        %Actor Policy functions for 1st & Second dimention/objective of storage matrix
        w( stateF, agentF, 1) = w( stateF, agentF, 1 ) + alpha * rbar( agentF ) * X( stateF ) * ( 1
/ro ( stateF, agentF, 1 ) ) * exp( -w( stateF, agentF,1 ) ) * sumWeights( agentF );
        w( stateF, agentF, 2) = w( stateF, agentF, 2 ) + alpha * rbar2( agentF ) * X( stateF ) * ( 1
/ro2 ( stateF, agentF, 1 ) ) * exp( -w( stateF, agentF,2 ) ) * sumWeights2( agentF );
        w( stateF, agentF, 3) = w( stateF, agentF, 3 ) + alpha * rbar3( agentF ) * X( stateF ) * ( 1
/ro3 ( stateF, agentF, 1 ) ) * exp( -w( stateF, agentF,3 ) ) * sumWeights3( agentF );

        %Critic Value functions Value function for 1st and 2nd Objectives
        v(stateF,agentF,1)=v(stateF,agentF,1)-beta*gamma*rbar(agentF)*X(stateF); %Takes
the form of TD error scalar signal
        v(stateF,agentF,2)=v(stateF,agentF,2)-beta*gamma*rbar2(agentF)*X(stateF); %Takes
the form of TD error scalar signal
        v(stateF,agentF,3)=v(stateF,agentF,3)-beta*gamma*rbar3(agentF)*X(stateF); %Takes
the form of TD error scalar signal

        %SHOULD WE USE THESE WEIGHTS IF THE OTHER CODE CHOOSES INDEX1 or
        %INDEX2 BASED UPON THE WEIGHTs ANYWAY
        %Scalarized Value function for BOTH Objectives
        %v(stateF,agentF,3)=(v(stateF,agentF,1)*W1 +
v(stateF,agentF,2)*W2)/NoOfObjectives;
```

```
            v(stateF,agentF,1)=(v(stateF,agentF,1)*W1 + v(stateF,agentF,2)*W2 +
v(stateF,agentF,3)*W3)/NoOfObjectives;

            w(stateF,agentF,1)=(w(stateF,agentF,1)*W1 + w(stateF,agentF,2)*W2 +
w(stateF,agentF,3)*W3)/NoOfObjectives;

            %Store scalarized Value functon back into the storage matrix (this can slow
            %things down considerably)
            %v(stateF,agentF,1)=v(stateF,agentF,3);

            %end
            %end

            % update values for next step, and directional choices
            prevM = m;
            prevN = n;
        end % end while not found
    end %end totalTrials loop

    % Update storage matrices for across all plays
    totalNumSteps = totalNumSteps + numSteps;

    % Update storage matrices for across all plays
    totalNumBoundSteps =  totalNumBoundSteps + numBoundSteps;

          % Update storage matrices for across all plays
    totalNumHillSteps =  totalNumHillSteps + numHillSteps;


end %end totalPlays loop

outOfBoundsCount = northCount + eastCount + southCount + westCount;
hillCount =  S43hillCount +  S53hillCount + S34hillCount +  S35hillCount +  S36hillCount;

% %------------------------------------
% % Outputs
% %------------------------------------
% % Average number of steps across all plays
% plot(totalNumSteps / totalPlays, 'color', 'k', 'LineStyle', '-');

%------------------------------------
% Plot 2D Graphs for both objectives
%------------------------------------
disp('GPFMORL Algorithm');
AverageBoundsCount = outOfBoundsCount/totalPlays
AverageHillCount = hillCount/totalPlays
disp(' ');

% Average number of steps across all plays
plot(totalNumSteps / totalPlays, 'color', 'k', 'LineStyle', '-');
```

```
title('GPFMORL Algorithm')
xlabel('NoOfTrials')
ylabel('NoOfSteps')
hold on

% Average number of boundary penaltys across all plays
plot(totalNumBoundSteps / totalPlays, 'color', 'k', 'LineStyle', '--');
hold on


% Average number of boundary penaltys across all plays
plot(totalNumHillSteps / totalPlays, 'color', 'k', 'LineStyle', ':');

legend('Obj1 Heli Goal','Obj2 Boundary','Obj3 Hills')
hold off
```

GPFMORLPT

```
%GPFMORL-PT Complex Enviroment with 3 Objectives
%this 10x10 with ptmo0d
%------------RL Constants--------------
alpha = 0.003; % Actor learning rate
beta = 0.005; % Critic learning rate
epsilon = 0.01; % Exploration/explotation control parameter
gamma = 0.95; % Discount factor

totalActions = 4; % Number of possible actions
totalAgents = totalActions; % Total number of learning agents

totalPlays = 3; % Number of plays to average over (was 100) works at 1
totalTrials = 40; % Trials per play (was 40) works at 20


%------------PT Constants--------------

pt_alpha = 0.88;
pt_beta = 0.88;
pt_lambda = 2.25; % loss aversion discount
pt_gamma = 0.75; % probabiliy weighting coefficient

%pt_w contains the weights of probabilities
pt_w = zeros(4,3);

%pt_nu contains the subjective values of each prospect
pt_nu = zeros(4,3);

%PT will contain the prospect theory value for each possible action
PT = zeros(4,1);

%RP is the two dimensional reference point
RP = zeros(3,1);

%------------MORL Variables------------
NoOfObjectives = 3; % Specify any number of objectives
W1 = 0.4; %Objective 1 weighting (reaching the helipad goal) Best@1
W2 = 0.3; %Objective 2 weighting (Avoiding the grid boundarys) Best@
W3 = 0.3; %Objective 3 obstacle
objectives = 3;
%outOfBoundsCount = 0; //dont need to declare variables in matlab

%Windy Grid World Enviroment Stochasticity
WindChance = 0.2; %20% chance of it being NotWindy, Windy, Very Windy


PFMFactor = 0.1; %was o.1
```

```
% ----------Enviroment Constants---------
gridSize = 10; % Size of the Grid (was 10)
totalStates = gridSize*gridSize; % Total number of system states
rewardM = ceil(gridSize/2); % Location of reward in M axe
rewardN = ceil(gridSize/2)+1; % Location of reward in N axe

% ---------Variable Initialisation--------
totalNumSteps = zeros(totalTrials, 1);% Total # steps for each trial over all plays
totalReward = zeros(totalTrials, totalAgents);
totalNumBoundSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all
plays
totalNumHillSteps = zeros(totalTrials, 1); % Total number of steps for each trial over all plays


plotX = zeros(totalTrials,1);
plotY = zeros(totalTrials,1);

trialCount = 1;

startM = 2; % Location to start from in M axe
startN = 2; % Location to start from in N axe //



%Store the number of times that each boundary is crossed (for plot)
northCount = 0;
eastCount = 0;
southCount = 0;
westCount = 0;

S43hillCount = 0;
S53hillCount = 0;
S34hillCount = 0;
S35hillCount = 0;
S36hillCount = 0;

%--------------------------------------------------------------
% Begin stepping through the grid world enviroment using GPFMORL
%--------------------------------------------------------------
for j = 1 : totalPlays
   % Store # steps for each trial
   numSteps = zeros(totalTrials, 1);

   % Store number of out of boundary steps (punishments)
   numBoundSteps = zeros(totalTrials, 1);

       % Store number of out of boundary steps (punishments)
   numHillSteps = zeros(totalTrials, 1);

   % Initializing actor and critic weights
```

```
v = zeros(totalStates, totalAgents, 3);
w = zeros(totalStates, totalAgents,3);

%reward at astate 46 CHANGED BECAUSE WAS WRONG
v(45,2,1) = 1;
v(45,2,2) = 1;
w(45,2,1) = 1;
w(45,2,2) = 1;
w(45,2,3) = 1;
v(45,2,3) = 1;

v(56,1,1) = 1;
v(56,1,2) = 1;
w(56,1,1) = 1;
w(56,1,2) = 1;
w(56,1,3) = 1;
v(56,1,3) = 1;



v(36,3,1) = 1;
v(36,3,2) = 1;
w(36,3,1) = 1;
w(36,3,2) = 1;
w(36,3,3) = 1;
v(36,3,3) = 1;

v(47,4,1) = 1;
v(47,4,2) = 1;
w(47,4,1) = 1;
w(47,4,2) = 1;
w(47,4,3) = 1;
v(47,4,3) = 1;

%building danger zone that would cost more to cross

%state 34

%comimg from west
v(33,2,3) =  -0.4;
w(33,2,3) =  -0.4;

%coming from south
v(44,1,3) =  -0.4;
w(44,1,3) =  -0.4;

%coming from north
v(24,3,3) =  -0.4;
w(24,3,3) = -0.4;
```

```
%coming from east
v(35,4,3) =  -0.4;
w(35,4,3) =  -0.4;

%state 35

%comimg from west
v(34,2,3) =  -0.6;
w(34,2,3) =  -0.6;

%coming from south
v(45,1,3) =  -0.6;
w(45,1,3) =  -0.6;

%coming from north
v(25,3,3) =  -0.6;
w(25,3,3) =  -0.6;

%coming from east
v(36,4,3) =  -0.6;
w(36,4,3) =  -0.6;

%state 36

%comimg from west
v(35,2,3) =  -0.8;
w(35,2,3) = -0.8;

%coming from south
v(46,1,3) =  -0.8;
w(46,1,3) =  -0.8;

%coming from north
v(26,3,3) =  -0.8;
w(26,3,3) =  -0.8;

%coming from east
v(37,4,3) =  -0.8;
w(37,4,3) = -0.8;

%state 43

%comimg from west
v(42,2,3) = -0.1;
w(42,2,3) = -0.1;

%coming from south
v(53,1,3) = -0.1;
w(53,1,3) = -0.1;
```

```
%coming from north
v(33,3,3) = -0.1;
w(33,3,3) = -0.1;

%coming from east
v(44,4,3) = -0.1;
w(44,4,3) = -0.1;

%state 53

%comimg from west
v(52,2,3) = -0.2;
w(52,2,3) = -0.2;

%coming from south
v(63,1,3) = -0.2;
w(63,1,3) = -0.2;

%coming from north
v(43,3,3) = -0.2;
w(43,3,3) = -0.2;

%coming from east
v(54,4,3) =  -0.2;
w(54,4,3) =  -0.2;



plotX = zeros(totalTrials,1);
plotY = zeros(totalTrials,1);

trialCount = 1;

RP = zeros(3,1);

%State, Action(2), ObjDimention = punishment

%Northern & Eastern Boundary Penalty Association 10x10
for ii = 1 : gridSize
   v(ii,1,2) = -1;
   v(ii*10,2,2) = -1;
end



%Southern Boundary Penalty Association 10x10
for jj=91 : 100
   v(jj,3,2) = -1;
end
```

```
%Western Boundary Penalty Associationv 10x10
for hh=1 : gridSize
   v((hh*10)-9,4,2) = -1;
end




% %Row 1 Puinsihments
%  v(1,1,2) = -1;
%  v(1,4,2) = -1;
%  v(2,1,2) = -1;
%  v(3,1,2) = -1;
%  v(4,1,2) = -1;
%  v(5,1,2) = -1;
%  v(5,2,2) = -1;
%
% %Row 2 Puinsihments
%  v(6,4,2) = -1;
%  v(10,2,2) = -1;
%
% %Row 3 Puinsihments
%  v(11,4,2) = -1;
%  v(15,2,2) = -1;
%
% %Row 4 Puinsihments
%  v(16,4,2) = -1;
%  v(20,2,2) = -1;
%
% %Row 5 Puinsihments
%  v(21,3,2) = -1;
%  v(21,4,2) = -1;
%  v(22,3,2) = -1;
%  v(23,3,2) = -1;
%  v(24,3,2) = -1;
%  v(25,2,2) = -1;
%  v(25,3,2) = -1;




for i = 1 : totalTrials
   disp(j+(i/100))

   %       % Random Starting Location
   %       stochGrid = rand(gridSize); %Generate 5x5 grid with random real numbers 0-1
   %       vectorMax = max(stochGrid);% Return the maximum value of each column
   %       big = max(vectorMax);%Determine maximum value of this vector of 5 values
```

```
    %       binaryMap = stochGrid==big % Create a binary matrix indicating position of max
value
    %       [row_M,col_N] = find(stochGrid==big)
    %       randomStartState = find(binaryMap') % find the indice of the 5x5
    %       grid (inverted to comply with my notation)

    %       % Assign random co-ordinates to each trials starting position/state
    %       startM = row_M;
    %       startN = col_N;

    % Current position on grid is represented by m,n
    m = startM;
    n = startN;
    prevM = startM;
    prevN = startN;


    RP = zeros(3,1);

    % % Clean some variables
    % p = zeros(totalAgents,1);
    % oldp = p;


    % Clean some variables
    p = zeros(totalAgents,3);
    oldp(:,1) = p(:,1);
    oldp(:,2) = p(:,2);
    oldp(:,3) = p(:,3);


    % Begin stepping through a path
    found = 0;
    stepcount = 0;
    while(found == 0)
      stepcount = stepcount +1;

      % Increment # steps taken this trial
      numSteps(i) = numSteps(i) + 1;

      % Find probabilities from actor weights
      sumWeights = zeros(totalAgents,1);
      sumWeights2 = zeros(totalAgents,1);
      sumWeights3 = zeros(totalAgents,1);

      ro = zeros(totalStates, totalAgents);
      ro2 = zeros(totalStates, totalAgents);
      ro3 = zeros(totalStates, totalAgents);

      for agent = 1 : totalAgents
    for state = 1 : totalStates
        %ro(state, agent) = logsig(w(state, agent));
        ro(state, agent) = logsig(w(state, agent,1));
        ro2(state, agent) = logsig(w(state, agent, 2));
        ro3(state, agent) = logsig(w(state, agent, 3));

        sumWeights(agent) = sumWeights(agent) + ro(state, agent);
        sumWeights2(agent) = sumWeights2(agent) + ro2(state, agent);
        sumWeights3(agent) = sumWeights3(agent) + ro3(state, agent);

    end
end
% Perform an observation (using boxes system)
stidx = ((m - 1) * gridSize) + n;
X = zeros(totalStates, 1);
X(stidx) = 1;

% Finding final probability
P = zeros(totalAgents, 1);
P2 = zeros(totalAgents, 1);
P3 = zeros(totalAgents, 1);
for agent = 1 : totalAgents
    P(agent) = ro(stidx, agent);
    P2(agent) = ro2(stidx, agent);
    P3(agent) = ro3(stidx, agent);
end
% Creating action set avoiding grid borders
%N is the new m axis position after moving North
N = m - 1;
if(N < 1)
    %    %Here i should punish this agents action by reducing probability of
    %    %moving north in that state
    %    %Find what state we are currently in
    %    CurrentState = find(X == max(X));
    %
    %    %Reduce the North agents probability when in states near boundary
    %    ro(CurrentState,1) = ro(CurrentState,1)- PFMFactor;
    %
    %    %Reduce the North agents probability when in states near boundary
    %    v(CurrentState,1,2) = ro(CurrentState,1)- PFMFactor*northCount;
    %    northCount = northCount + 1;

    N = 1;

    northCount = northCount + 1;
    numBoundSteps(i) =  numBoundSteps(i) + 1;
end


E = n + 1;
```

```
if(E > gridSize)
   %    %Find what state we are currently in
   %    CurrentState = find(X == max(X));
   %
   %    %Reduce the East agents probability when in states near boundary
   %    ro(CurrentState,2) = ro(CurrentState,2)- PFMFactor;
   %
   %     %Reduce the Easts agents probability when in states near boundary
   %    v(CurrentState,2,2) = ro(CurrentState,2)- PFMFactor*eastCount;
   %    eastCount = eastCount + 1;
   %
   E = gridSize;

   eastCount = eastCount + 1;
   numBoundSteps(i) =  numBoundSteps(i) + 1;

end
S = m + 1;
if(S > gridSize)
   %     %Find what state we are currently in
   %    CurrentState = find(X == max(X));
   %
   %    %Reduce the South agents probability when crossing grid boundary
   %     ro(CurrentState,3) = ro(CurrentState,3)- PFMFactor;
   %
   %    %Reduce the South agents probability when in states near boundary
   %    v(CurrentState,3,2) = ro(CurrentState,3)- PFMFactor*southCount;
   %    southCount = southCount + 1;
   southCount = southCount + 1;
   numBoundSteps(i) =  numBoundSteps(i) + 1;

   S = gridSize;
end
W = n - 1;
if(W < 1)
   %     %Find what state we are currently in
   %    CurrentState = find(X == max(X));
   %
   %    %Reduce the West agents probability when in states near boundary
   %     ro(CurrentState,4) = ro(CurrentState,4)- PFMFactor;
   %
   %    %Reduce the West agents probability when in states near boundary
   %    v(CurrentState,4,2) = ro(CurrentState,4)- PFMFactor*westCount;
   %    westCount = westCount + 1;

   W = 1;
   westCount = westCount + 1;
   numBoundSteps(i) =  numBoundSteps(i) + 1;
end
```

```
% Initializing values
Nu = zeros(totalAgents, 1);
Nu2 = zeros(totalAgents, 1);
Nu3 = zeros(totalAgents, 1);

stdev = zeros(totalAgents, 1);
stdev2 = zeros(totalAgents, 1);
stdev3 = zeros(totalAgents, 1);

% Stochastic random number generator
for agent = 1 : totalAgents
   %stdev(agent) = 2*logsig(p(agent))-1;
   stdev(agent) = 2*logsig(p(agent,1))-1;
   stdev2(agent) = 2*logsig(p(agent,2))-1;
   stdev3(agent) = 2*logsig(p(agent,3))-1;

   Nu(agent) = stdev(agent)*randn;
   Nu2(agent) = stdev2(agent)*randn;
   Nu3(agent) = stdev3(agent)*randn;
   if Nu(agent) >= 1
      Nu(agent) = 1;
   elseif Nu(agent) <= -1
      Nu(agent) = -1;
   end

   if Nu2(agent) >= 1
      Nu2(agent) = 1;
   elseif Nu2(agent) <= -1
      Nu2(agent) = -1;
   end

   if Nu3(agent) >= 1
      Nu3(agent) = 1;
   elseif Nu3(agent) <= -1
      Nu3(agent) = -1;
   end

end


      %Detecting when agent has encountered an Obstacle/Hill
%State 43
if (m == 5 && n == 3)
    S43hillCount = S43hillCount + 1;
   numHillSteps(i) =  numHillSteps(i) + 1;
end

%State 53
if (m == 6 && n == 3)
```

```
                S53hillCount = S53hillCount + 1;
               numHillSteps(i) =  numHillSteps(i) + 1;
            end

              %State 34
            if (m == 4 && n == 4)
                S34hillCount = S34hillCount + 1;
               numHillSteps(i) =  numHillSteps(i) + 1;
            end

            %State 35
            if (m == 4 && n == 5)
                S35hillCount = S35hillCount + 1;
               numHillSteps(i) =  numHillSteps(i) + 1;
            end

              %State 36
            if (m == 4 && n == 6)
                S36hillCount = S36hillCount + 1;
               numHillSteps(i) =  numHillSteps(i) + 1;
            end


            %Prospect Theory values computation
            %w > probability weights

            for agent=1 : totalAgents
               pt_w(agent,1) = (P(agent)^pt_gamma) / ( P(agent)^pt_gamma +
((1+P(agent)^pt_gamma))^(1/pt_gamma) );
               pt_w(agent,2) = (P2(agent)^pt_gamma) / ( P2(agent)^pt_gamma +
((1+P2(agent)^pt_gamma))^(1/pt_gamma) );
               pt_w(agent,3) = (P3(agent)^pt_gamma) / ( P3(agent)^pt_gamma +
((1+P3(agent)^pt_gamma))^(1/pt_gamma) );

               if(w(stidx,agent,1) >= RP(1))
                  pt_nu(agent,1) = (w(stidx,agent,1) - RP(1))^pt_alpha;
               else
                  pt_nu(agent,1) = (-pt_lambda*(RP(1) - v(stidx,agent,1)))^pt_beta;
               end

               if(w(stidx,agent,2) >= RP(2))
                  pt_nu(agent,2) = (w(stidx,agent,2) - RP(2))^pt_alpha;
               else
                  pt_nu(agent,2) = (-pt_lambda*(RP(2) - v(stidx,agent,2)))^pt_beta;
               end

               if(w(stidx,agent,3) >= RP(3))
                  pt_nu(agent,3) = (w(stidx,agent,3) - RP(3))^pt_alpha;
               else
                  pt_nu(agent,3) = (-pt_lambda*(RP(3) - v(stidx,agent,3)))^pt_beta;
            end

            PT(agent) = pt_nu(agent,1)*pt_w(agent,1) + pt_nu(agent,2)*pt_w(agent,2) +
pt_nu(agent,3)*pt_w(agent,3);

        end



        % Finding choices (Probability + explotation/exploration) FOR OBJECTIVE 1
%           choices = zeros(totalAgents, 1);
%           for agent=1:totalAgents
%               choices(agent) = P(agent)+Nu(agent);
%           end
%
%           % Finding choices (Probability + explotation/exploration) FOR OBJECTIVE 2
%           choices2 = zeros(totalAgents, 1);
%           for agent=1:totalAgents
%               choices2(agent) = P2(agent)+Nu2(agent);
%           end
%

        %action selection




        %we choose an index
        % Choose the path with the maximum probability, or explore
        indices = find(PT == max(PT));
        maximum = max(PT);
        if((length(indices) == 1) && (maximum >= epsilon))
            index = indices(1);
        else
            index = ceil(length(PT) * rand); % randomly select direction
        end

        %we choose an index for objective 2
        % Choose the path with the maximum probability, or explore
        %           indices2 = find(choices2 == max(choices2));
        %           maximum2 = max(choices2);
        %           if((length(indices2) == 1) && (maximum2 >= epsilon))
        %               index2 = indices2(1);
        %           else
        %               index2 = ceil(length(choices2) * rand); % randomly select direction
        %           end
        %

        %now we have two choices, one for each objective
```

```
%we now need to choose one, we can't compute both of the final values, in a
%real environment, the response to the action would come up after taking
%action, so we will choose stochasticly, taking the objectives weights into
%account

%NB: this is where we are gonna integrate Prospect theory

%           choice_rand = rand;
%           if(choice_rand < W1)
%               index = index1;
%           else
%               index = index2;
%           end


 windy_rand = rand;

  disp(' ');
disp(stidx);

if (windy_rand < WindChance)



    if (n == 1 && m>1)
        disp('N WAS 1, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end
    if (n == 2 && m>1)
        disp('N WAS 2, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end
    if (n == 3 && m>1)
        disp('N WAS 3, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 4 && m>1)
        disp('N WAS 4, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 5 && m>2)
        disp('N WAS 5, Wind Strength = 2');
        WindStrength = 2;
        m = m-WindStrength;
    end
    if (n == 6 && m>2)
```

```
        disp('N WAS 6, Wind Strength = 2');
        WindStrength = 2;
        m = m-WindStrength;
    end
    if (n == 7 && m>1)
        disp('N WAS 7, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 8 && m>1)
        disp('N WAS 8, Wind Strength = 1');
        WindStrength = 1;
        m = m-WindStrength;
    end
    if (n == 9 && m>1)
        disp('N WAS 9, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end
    if (n == 10 && m>1)
        disp('N WAS 10, Wind Strength = 0');
        WindStrength = 0;
        m = m-WindStrength;
    end


     disp(' ');

else
    disp('NO WIND');

end %end primary if

%Take action
r=zeros(totalAgents,1);
if(index == 1) % grid(m-1,n)
    m = N;
    r(index) = sign(m-rewardM);
elseif(index == 2) % grid(m,n+1)
    n = E;
    r(index) = sign(rewardN-n);
elseif(index == 3) % grid(m+1,n)
    m = S;
    r(index) = sign(rewardM-m);
elseif(index == 4) % grid(m,n-1)
    n = W;
    r(index) = sign(n-rewardN);
else
    disp('error random number too big');
end
```

```
% Stop after this step if the reward was found
if(m == rewardM && n == rewardN)
   found = 1;
   disp(46);
end
% Saving prediction information
oldp(:,1) = p(:,1);
oldp(:,2) = p(:,2);
oldp(:,3) = p(:,3);
p = zeros(totalAgents,3);

% % Computing the prediction of eventual reinforcement
% for agent = 1:totalAgents
% p(agent) = v(stidx,agent);
% end


% Computing the prediction of eventual reinforcement
for agent = 1:totalAgents
   p(agent,1) = v(stidx,agent,1);
   p(agent,2) = v(stidx,agent,2);
   p(agent,3) = v(stidx,agent,3);
end
% Computing the prediction error using temporal differences method
rbar=zeros(totalAgents,1);
rbar2=zeros(totalAgents,1);
rbar3=zeros(totalAgents,1);
for agent = 1:totalAgents
   %predErr = gamma*p(agent) - oldp(agent); %
   predErr = gamma*p(agent,1) - oldp(agent,1); %
   predErr2 = gamma*p(agent,2) - oldp(agent,2);
   predErr3 = gamma*p(agent,3) - oldp(agent,3);
   %rbar(agent) = r(agent) + predErr;
   rbar(agent) = r(agent) + predErr;
   rbar2(agent) = r(agent) + predErr2;
   rbar3(agent) = r(agent) + predErr3;
end
rf_tmp = zeros(3,1);

% Learning the value functions
stateF = stidx;
agentF = index;


%Actor & Critic Value functions updating 1st dimention/objective of storage matrix
w( stateF, agentF, 1) = w( stateF, agentF, 1 ) + alpha * rbar( agentF ) * X( stateF ) * ( 1
/ro ( stateF, agentF, 1 ) ) * exp( -w( stateF, agentF,1 ) ) * sumWeights( agentF );
w( stateF, agentF, 2) = w( stateF, agentF, 2 ) + alpha * rbar2( agentF ) * X( stateF ) * ( 1
/ro2 ( stateF, agentF, 1 ) ) * exp( -w( stateF, agentF,2 ) ) * sumWeights2( agentF );
```

```
        w( stateF, agentF, 3) = w( stateF, agentF, 3 ) + alpha * rbar3( agentF ) * X( stateF ) * ( 1
/ro3 ( stateF, agentF, 1 ) ) * exp( -w( stateF, agentF,3 ) ) * sumWeights3( agentF );

        v(stateF,agentF,1)=v(stateF,agentF,1)-beta*gamma*rbar(agentF)*X(stateF); %Takes
the form of TD error scalar signal
        v(stateF,agentF,2)=v(stateF,agentF,2)-beta*gamma*rbar2(agentF)*X(stateF); %Takes
the form of TD error scalar signal
        v(stateF,agentF,3)=v(stateF,agentF,3)-beta*gamma*rbar3(agentF)*X(stateF); %Takes
the form of TD error scalar signal

        %Scalarized Value function for BOTH Objectives
       % v(stateF,agentF,3)=(v(stateF,agentF,1)*W1 +
v(stateF,agentF,2)*W2)/NoOfObjectives;

        %Store scalarized Value functon back into the storage matrix (this can slow
        %things down considerably)
        %v(stateF,agentF,1)=v(stateF,agentF,3);


        %update the reference point for prospect theory

        RP(1) =  RP(1) + v(stateF,agent,1);
        RP(2) =  RP(2) + v(stateF,agent,2);
        RP(3) =  RP(3) + v(stateF,agent,3);



%
%           plot(RP(1)*10, RP(2)*10,'o');
%           hold on
%
%           RP(1) = RP(1) / stepcount;
%          RP(2) = RP(2) / stepcount;

        % update values for next step, and directional choices
        prevM = m;
        prevN = n;
      end % end while not found
%
%       plotX(trialCount) = v(state,agent,1);
%       plotY(trialCount) = v(state,agent,2);

      trialCount = trialCount + 1;

%       plot(v(state,agent,1),v(state,agent,2),'o');
%       hold on



  end %end totalTrials loop
```

```
    %plot(plotX, plotY, 'o');
%     xlabel('1st objective');
%     ylabel('2nd objective');

    % Update storage matrices for across all plays
    totalNumSteps = totalNumSteps + numSteps;

    % Update storage matrices for across all plays
    totalNumBoundSteps =  totalNumBoundSteps + numBoundSteps;

        % Update storage matrices for across all plays
    totalNumHillSteps =  totalNumHillSteps + numHillSteps;

end %end totalPlays loop

outOfBoundsCount = northCount + eastCount + southCount + westCount;
hillCount =  S43hillCount +  S53hillCount + S34hillCount +  S35hillCount +  S36hillCount;

% %------------------------------------
% % Outputs
% %------------------------------------
% % Average number of steps across all plays
% plot(totalNumSteps / totalPlays, 'color', 'k', 'LineStyle', '-');

%------------------------------------
% Plot 2D Graphs for both objectives
%------------------------------------
disp('GPFMORLPTv5 Hills Algorithm');
AverageBoundsCount = outOfBoundsCount/totalPlays
AverageHillCount = hillCount/totalPlays
disp(' ');

% Average number of steps across all plays
plot(totalNumSteps / totalPlays, 'color', 'k', 'LineStyle', '-');
title('GPFMORLPT Hills Algorithm')
xlabel('NoOfTrials')
ylabel('NoOfSteps')
hold on

% Average number of boundary penaltys across all plays
plot(totalNumBoundSteps / totalPlays, 'color', 'k', 'LineStyle', '--');
hold on

% Average number of boundary penaltys across all plays
plot(totalNumHillSteps / totalPlays, 'color', 'k', 'LineStyle', ':');

legend('Obj1 Heli Goal','Obj2 Boundary','Obj3 Hills')
hold off
```

Appendix B

Unmanned Aerial Vehicle Practical

UAV Practical Test

```
//UAV v1
#include "ardrone/ardrone.h"
#include <fstream>

//-------------------Variables---------------------
double innerPower = 0.1;          // Safe zone potential field magnitude - set to 0 when RL
decides actions
double outerPower = 0.2;          // Danger zone potential field magnitude
double altitudePower = 0.1; // Altiitude velocity
double distancePower = 0.1;
bool AUTOfloor = false;           // Set Automatic Floor Reactive Behaviour ON or OFF
bool AUTOfront = false;           // Set Automatic Front Reactive Behaviour ON or OFF
bool AUTOland = false;            // Set Automatic Landing flag ON or OFF
bool rec = false;
const int gridSize = 5;
int DroneState = 0;                          // Current state that the UAV is hovering above or in
front of
int prevDroneState = 0;           // Previous state that the UAV was at before moving to its
current state
int targetArea;                              // Used for moving forward and backward when using
forward camera
bool targetReached = false; // Flag for when front camera detects centralised target at
appropiate distance
int XpixelSegments;
int YpixelSegments;
int m; //not used atm
int n; //not used atm

bool UpANDSwitch = false; //set to true for automatic increase of altitude and switch cameras

//-----------------Initialisation-------------------
//File Writing of DroneState History
std::ofstream outputFile; //Global create output file for storing what states have been visited

// GUI HSV Threshold slider bar limits 0 -> 255 (Fixed Forward Target)
//int minH = 90, maxH = 229;
//int minS = 187, maxS = 255;
//int minV = 39, maxV = 166;

//Improved for Salford Royal Visit
int minH = 0, maxH = 128; //max H best at 155 for ocluded corner (Fixed Forward Target)
int minS = 136, maxS = 235;
int minV = 0, maxV = 103;

//Create video object
CvVideoWriter *video;

int main(int argc, char **argv)
{
    // AR.Drone class
    ARDrone ardrone;

    // Initialize
    if (!ardrone.open()) {
        printf("Failed to initialize.\n");
        return -1;}

            // Image of AR.Drone's camera
    IplImage *image = ardrone.getImage();

             //// Orientation
  //      double roll  = ardrone.getRoll();
  //      double pitch = ardrone.getPitch();
  //      double yaw   = ardrone.getYaw();
  //      printf("ardrone.roll  = %3.2f [deg]\n", roll  * RAD_TO_DEG);
  //      printf("ardrone.pitch = %3.2f [deg]\n", pitch * RAD_TO_DEG);
  //      printf("ardrone.yaw   = %3.2f [deg]\n", yaw   * RAD_TO_DEG);

        //// Altitude
        double altitude = ardrone.getAltitude();
        printf("ardrone.altitude = %3.2f [m]\n", altitude);

        //// Velocity
        //double vx, vy, vz;
        //double velocity = ardrone.getVelocity(&vx, &vy, &vz);
        //printf("ardrone.vx = %3.2f [m/s]\n", vx);
        //printf("ardrone.vy = %3.2f [m/s]\n", vy);
        //printf("ardrone.vz = %3.2f [m/s]\n", vz);

            // Name of video
    char filename[256];
    SYSTEMTIME st;
    GetLocalTime(&st);
            //when compiled and run in VisualStudio, check build2010 folder, or if running
executable check bin2010
    //sprintf(filename, "cam%d%02d%02d%02d%02d%02d.avi", st.wYear, st.wMonth, st.wDay,
st.wHour, st.wMinute, st.wSecond);
            sprintf(filename, "vid%02d-%02d  %02d-%02d.avi", st.wDay, st.wMonth, st.wHour,
st.wMinute);

            //Open text file
            outputFile.open("DroneStateLog.txt"); //Open output file to be written to

    // Operational Instructions
    printf("***************************************\n");
    printf("*       John Pinder's MORL-UAV        *\n");
    printf("***************************************\n");
    printf("* - Controls -                        *\n");
    printf("*    'Space' -- Takeoff/Landing       *\n");
    printf("*    'Up'    -- Move Forward          *\n");
    printf("*    'Down'  -- Move Backward         *\n");
    printf("*    'Left'  -- Straife Left          *\n");
    printf("*    'Right' -- Straife Right         *\n");
    printf("*    'R'     -- Raise Altitude        *\n");
    printf("*    'F'     -- Fall Altitude         *\n");
            printf("*    'Q'     -- Turn Left             *\n");
            printf("*    'E'     -- Turn Right            *\n");
    printf("*                                     *\n");
    printf("* - Automatic Control Toggles -       *\n");
            printf("*    'A'     -- Auto Floor            *\n");
    printf("*    'Z'     -- Auto Front            *\n");
            printf("*    'L'     -- Auto Land             *\n");
            printf("*                                     *\n");
    printf("*  - Others -                         *\n");
            printf("*    'V'     -- Image Process Config  *\n");
            printf("*    'C'     -- Change Camera         *\n");
            printf("*    'O'     -- Start/Stop Recoring   *\n");
    printf("*    'Esc'   -- Exit                  *\n");
            printf("*    'M'     -- Flight Animations     *\n");
    printf("***************************************\n");

            // Battery
    printf("Battery = %d%%\n", ardrone.getBatteryPercentage());

//////////////////////////////////////////////////////WHILE LOOP//////////////////////////////////////////////////

int lock = 0; //Used to prevent multiple video writer objects from being instantiated
            while (1) {
        // Key input
        int key = cvWaitKey(33);
        if (key == 0x1b) break;

                            //// Altitude
        double altitude = ardrone.getAltitude();
        //printf("ardrone.altitude = %3.2f [m]\n", altitude);

                                    int bat = ardrone.getBatteryPercentage();

                        // Create a video writer
                        if (rec && lock == 0){
                                    video = cvCreateVideoWriter(filename, CV_FOURCC('D','I','B',' '), 30,
cvGetSize(image));
                                    lock = 1;}

                        // Create a window with HSV slider selections for image processsing
segmentation thresholds
                        if (key == 'v') {
                                    cvNamedWindow("Image Procesing Configuration");
```

```
                    cvCreateTrackbar("H max", "Image Procesing Configuration", &maxH,
255);
                    cvCreateTrackbar("H min", "Image Procesing Configuration", &minH,
255);
                    cvCreateTrackbar("S max", "Image Procesing Configuration", &maxS,
255);
                    cvCreateTrackbar("S min", "Image Procesing Configuration", &minS,
255);
                    cvCreateTrackbar("V max", "Image Procesing Configuration", &maxV,
255);
                    cvCreateTrackbar("V min", "Image Procesing Configuration", &minV,
255);
                    cvResizeWindow("Image Procesing Configuration", 0, 0);}

             // Instructions for Flight Animations
             if (key == 'm') {
                    printf("  G - ARDRONE_ANIM_PHI_M30_DEG\n");
                    printf("  N - ARDRONE_ANIM_PHI_30_DEG\n");
                    printf("  K - ARDRONE_ANIM_THETA_M30_DEG\n");
                    printf("  W - ARDRONE_ANIM_THETA_30_DEG\n");
                    printf("  S - ARDRONE_ANIM_THETA_20DEG_YAW_200DEG\n");
                    printf("  4 - ARDRONE_ANIM_THETA_20DEG_YAW_M200DEG\n");
                    printf("  5 - ARDRONE_ANIM_TURNAROUND\n");
                    printf("  D - ARDRONE_ANIM_TURNAROUND_GODOWN\n");
                    printf("  2 - ARDRONE_ANIM_YAW_SHAKE\n");
                    printf("  I - ARDRONE_ANIM_YAW_DANCE\n");
                    printf("  3 - ARDRONE_ANIM_PHI_DANCE\n");
                    printf("  P - ARDRONE_ANIM_THETA_DANCE\n");
                    printf("  T - ARDRONE_ANIM_VZ_DANCE\n");
                    printf("  G - ARDRONE_ANIM_WAVE\n");
                    printf("  B - ARDRONE_ANIM_PHI_THETA_MIXED\n");
                    printf("  Y - ARDRONE_ANIM_DOUBLE_PHI_THETA_MIXED\n");
                    printf("  H - ARDRONE_ANIM_FLIP_AHEAD\n");
                    printf("  6 - ARDRONE_ANIM_FLIP_BEHIND\n");
                    printf("  U - ARDRONE_ANIM_FLIP_LEFT\n");
                    printf("  J - ARDRONE_ANIM_FLIP_RIGHT\n");}


             //// LED animations (cOMMENT lOUT USUALY)
 //    if (key == 'q') ardrone.setLED(ARDRONE_LED_ANIM_BLINK_GREEN_RED,
0.5, 5);
 //    if (key == 'a') ardrone.setLED(ARDRONE_LED_ANIM_BLINK_GREEN,            0.5,
5);
 //    if (key == 'z') ardrone.setLED(ARDRONE_LED_ANIM_BLINK_RED,             0.5, 5);
 //    if (key == 'w') ardrone.setLED(ARDRONE_LED_ANIM_BLINK_ORANGE,           0.5,
5);
 //    if (key == 's') ardrone.setLED(ARDRONE_LED_ANIM_SNAKE_GREEN_RED,
0.5, 5);
 //    if (key == 'x') ardrone.setLED(ARDRONE_LED_ANIM_FIRE,                 0.5, 5);
 //    if (key == 'e') ardrone.setLED(ARDRONE_LED_ANIM_STANDARD,              0.5, 5);
```

```
  //     if (key == 'd') ardrone.setLED(ARDRONE_LED_ANIM_RED,                       0.5, 5);
  //     if (key == 'c') ardrone.setLED(ARDRONE_LED_ANIM_GREEN,                      0.5, 5);
  //     if (key == 'r') ardrone.setLED(ARDRONE_LED_ANIM_RED_SNAKE,                  0.5, 5);
  //     if (key == 'f') ardrone.setLED(ARDRONE_LED_ANIM_BLANK,                      0.5, 5);
  //     if (key == 'v') ardrone.setLED(ARDRONE_LED_ANIM_RIGHT_MISSILE,              0.5,
5);
  //     if (key == 't') ardrone.setLED(ARDRONE_LED_ANIM_LEFT_MISSILE,               0.5, 5);
  //     if (key == 'g') ardrone.setLED(ARDRONE_LED_ANIM_DOUBLE_MISSILE,             0.5,
5);
  //     if (key == 'b')
ardrone.setLED(ARDRONE_LED_ANIM_FRONT_LEFT_GREEN_OTHERS_RED,  0.5, 5);
  //     if (key == 'y')
ardrone.setLED(ARDRONE_LED_ANIM_FRONT_RIGHT_GREEN_OTHERS_RED, 0.5, 5);
  //     if (key == 'h')
ardrone.setLED(ARDRONE_LED_ANIM_REAR_RIGHT_GREEN_OTHERS_RED,  0.5, 5);
  //     if (key == 'n')
ardrone.setLED(ARDRONE_LED_ANIM_REAR_LEFT_GREEN_OTHERS_RED,   0.5, 5);
  //     if (key == 'u') ardrone.setLED(ARDRONE_LED_ANIM_LEFT_GREEN_RIGHT_RED,
0.5, 5);
  //     if (key == 'j') ardrone.setLED(ARDRONE_LED_ANIM_LEFT_RED_RIGHT_GREEN,
0.5, 5);
  //     if (key == 'm') ardrone.setLED(ARDRONE_LED_ANIM_BLINK_STANDARD,
0.5, 5);

        // Update
        if (!ardrone.update()) break;

        // Get an image
        IplImage *image = ardrone.getImage(); //maybe not needed

                        // Write video frames when told to
                        //if (rec){
                        //          cvWriteFrame(video, image);}

/////////////////////////////////////IMAGE PROCCESING ////////////////////////////////////////////////////////
                        // HSV image
        IplImage *hsv = cvCloneImage(image);
        cvCvtColor(image, hsv, CV_RGB2HSV_FULL);

        // Binalized image
        IplImage *binalized = cvCreateImage(cvGetSize(image), IPL_DEPTH_8U, 1); // Create
black white image

        // Binalize
        CvScalar lower = cvScalar(minH, minS, minV); // Set HSV lower threshholds from
sliderbar
        CvScalar upper = cvScalar(maxH, maxS, maxV); // Set HSV upper threshholds from
sliderbar

                        // Feature Extraction
                        cvInRangeS(image, lower, upper, binalized); //Binalize helipad segmented
from background using HSV thresholds

        // Show result
        cvShowImage("binalized", binalized);

        // De-noising
        cvMorphologyEx(binalized, binalized, NULL, NULL, CV_MOP_CLOSE);

        // Detect contours
        CvSeq *contour = NULL, *maxContour = NULL;
        CvMemStorage *contourStorage = cvCreateMemStorage();
        cvFindContours(binalized, contourStorage, &contour, sizeof(CvContour),
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

        // Find largest contour
        double max_area = 0.0;
        while (contour) {
          double area = fabs(cvContourArea(contour));
          if (area > max_area) {
            maxContour = contour;
            max_area = area;}
                                  contour = contour->h_next;}

        // Object detected loop
        if (maxContour) {
          // Show result
          CvRect rect = cvBoundingRect(maxContour);
          CvPoint minPoint, maxPoint;
          minPoint.x = rect.x;
          minPoint.y = rect.y;
          maxPoint.x = rect.x + rect.width;
          maxPoint.y = rect.y + rect.height;
          cvRectangle(image, minPoint, maxPoint, CV_RGB(255,0,0));
                        //printf("MinX = %d ",minPoint.x);
                        //printf("MinY = %d\n ",minPoint.y);

                   int  xCentroid = (minPoint.x + maxPoint.x)/2;
                   int  yCentroid = (minPoint.y + maxPoint.y)/2;
                        //printf("X= %d ",xCentroid);
                        //printf("Y= %d\n ",yCentroid);
                        //printf("width = %d ",rect.width);
                        //printf("Height= %d\n ",rect.height);

                        //Calculate area of target for distance correction
                        targetArea = rect.height*rect.width;
                        //printf("Area = %d\n ",targetArea);

                        //Pixel calculation
                        int totalXPixels = 638;
```

```
                        int XpixelSegments = totalXPixels/gridSize;
                        int totalYPixels = 357;
                        int YpixelSegments = totalYPixels/gridSize;

                        //Assigning pixel coordinates to states
                                //First row of 5 states
                        if ((xCentroid >0*XpixelSegments && xCentroid<1*XpixelSegments)
&& (yCentroid >0*YpixelSegments && yCentroid<1*YpixelSegments)){
                                        DroneState = 1;
                                        m=1;
                                        n=1;                    }
                        if ((xCentroid >1*XpixelSegments && xCentroid<2*XpixelSegments)
&& (yCentroid >0*YpixelSegments && yCentroid<1*YpixelSegments)){
                                        DroneState = 2;
                                        m=1;
                                        n=2;                    }
                        if ((xCentroid >2*XpixelSegments && xCentroid<3*XpixelSegments)
&& (yCentroid >0*YpixelSegments && yCentroid<1*YpixelSegments)){
                                        DroneState = 3;
                                        m=1;
                                        n=3;                    }
                        if ((xCentroid >3*XpixelSegments && xCentroid<4*XpixelSegments)
&& (yCentroid >0*YpixelSegments && yCentroid<1*YpixelSegments)){
                                        DroneState = 4;
                                        m=1;
                                        n=4;                    }
                        if ((xCentroid >4*XpixelSegments && xCentroid<5*XpixelSegments)
&& (yCentroid >0*YpixelSegments && yCentroid<1*YpixelSegments)){
                                        DroneState = 5;
                                        m=1;
                                        n=5;                    }

                        //Second row of 5 states
                        if ((xCentroid >0*XpixelSegments && xCentroid<1*XpixelSegments)
&& (yCentroid >1*YpixelSegments && yCentroid<2*YpixelSegments)){
                                        DroneState = 6;
                                        m=2;
                                        n=1;                    }
                        if ((xCentroid >1*XpixelSegments && xCentroid<2*XpixelSegments)
&& (yCentroid >1*YpixelSegments && yCentroid<2*YpixelSegments)){
                                        DroneState = 7;
                                        m=2;
                                        n=2;                    }
                        if ((xCentroid >2*XpixelSegments && xCentroid<3*XpixelSegments)
&& (yCentroid >1*YpixelSegments && yCentroid<2*YpixelSegments)){
                                        DroneState = 8;
                                        m=2;
                                        n=3;                    }
                        if ((xCentroid >3*XpixelSegments && xCentroid<4*XpixelSegments)
&& (yCentroid >1*YpixelSegments && yCentroid<2*YpixelSegments)){
```

```
                                                       DroneState = 9;
                                                      m=2;
                                                      n=4;                        }
                                        if ((xCentroid >4*XpixelSegments && xCentroid<5*XpixelSegments)
&& (yCentroid >1*YpixelSegments && yCentroid<2*YpixelSegments)){
                                                       DroneState = 10;
                                                      m=2;
                                                      n=5;                        }

                                        //Third row of 5 states
                                        if ((xCentroid >0*XpixelSegments && xCentroid<1*XpixelSegments)
&& (yCentroid >2*YpixelSegments && yCentroid<3*YpixelSegments)){
                                                       DroneState = 11;
                                                      m=3;
                                                      n=1;                        }
                                        if ((xCentroid >1*XpixelSegments && xCentroid<2*XpixelSegments)
&& (yCentroid >2*YpixelSegments && yCentroid<3*YpixelSegments)){
                                                       DroneState = 12;
                                                      m=3;
                                                      n=2;                        }
                                        if ((xCentroid >2*XpixelSegments && xCentroid<3*XpixelSegments)
&& (yCentroid >2*YpixelSegments && yCentroid<3*YpixelSegments)){
                                                       DroneState = 13;
                                                      m=3;
                                                      n=3;                        }
                                        if ((xCentroid >3*XpixelSegments && xCentroid<4*XpixelSegments)
&& (yCentroid >2*YpixelSegments && yCentroid<3*YpixelSegments)){
                                                       DroneState = 14;
                                                      m=3;
                                                      n=4;                }
                                        if ((xCentroid >4*XpixelSegments && xCentroid<5*XpixelSegments)
&& (yCentroid >2*YpixelSegments && yCentroid<3*YpixelSegments)){
                                                       DroneState = 15;
                                                      m=3;
                                                      n=5;                        }

                                        //Fourth row of 5 states
                                        if ((xCentroid >0*XpixelSegments && xCentroid<1*XpixelSegments)
&& (yCentroid >3*YpixelSegments && yCentroid<4*YpixelSegments)){
                                                       DroneState = 16;
                                                      m=4;
                                                      n=1;                        }
                                        if ((xCentroid >1*XpixelSegments && xCentroid<2*XpixelSegments)
&& (yCentroid >3*YpixelSegments && yCentroid<4*YpixelSegments)){
                                                       DroneState = 17;
                                                      m=4;
                                                      n=2;                        }
                                        if ((xCentroid >2*XpixelSegments && xCentroid<3*XpixelSegments)
&& (yCentroid >3*YpixelSegments && yCentroid<4*YpixelSegments)){
                                                       DroneState = 18;
```

```
                                                m=4;
                                                n=3;                    }
                        if ((xCentroid >3*XpixelSegments && xCentroid<4*XpixelSegments)
&& (yCentroid >3*YpixelSegments && yCentroid<4*YpixelSegments)){
                                                 DroneState = 19;
                                                m=4;
                                                n=4;                    }
                        if ((xCentroid >4*XpixelSegments && xCentroid<5*XpixelSegments)
&& (yCentroid >3*YpixelSegments && yCentroid<4*YpixelSegments)){
                                                 DroneState = 20;
                                                m=4;
                                                n=5;                    }

                        //Fith row of 5 states
                        if ((xCentroid >0*XpixelSegments && xCentroid<1*XpixelSegments)
&& (yCentroid >4*YpixelSegments && yCentroid<5*YpixelSegments)){
                                                 DroneState = 21;
                                                m=5;
                                                n=1;                    }
                        if ((xCentroid >1*XpixelSegments && xCentroid<2*XpixelSegments)
&& (yCentroid >4*YpixelSegments && yCentroid<5*YpixelSegments)){
                                                 DroneState = 22;
                                                m=5;
                                                n=2;                    }
                        if ((xCentroid >2*XpixelSegments && xCentroid<3*XpixelSegments)
&& (yCentroid >4*YpixelSegments && yCentroid<5*YpixelSegments)){
                                                 DroneState = 23;
                                                m=5;
                                                n=3;                    }
                        if ((xCentroid >3*XpixelSegments && xCentroid<4*XpixelSegments)
&& (yCentroid >4*YpixelSegments && yCentroid<5*YpixelSegments)){
                                                 DroneState = 24;
                                                m=5;
                                                n=4;                    }
                        if ((xCentroid >4*XpixelSegments && xCentroid<5*XpixelSegments)
&& (yCentroid >4*YpixelSegments && yCentroid<5*YpixelSegments)){
                                                 DroneState = 25;
                                                m=5;
                                                n=5;                    }

                        // } //end if drone is flat
                } //end IF Object Detected
                else  {DroneState = 0; //Assign current state to 0 when the helipad localisation
is lost
                  } //do i realy need these braces as a single line

////////////////////////////////////////OUTPUTS//////////////////////////////////////////////////
                //Console ouput of changing Drone State when Automatic Reactive Behaviour
is ON
                if (AUTOfloor == true || AUTOfront == true){
```

```
			if(DroneState!=prevDroneState)std::cout << "STATE " << DroneState << " " <<
std::endl;}

			//Always output changing DroneState to DroneState.txt
			if(DroneState!=prevDroneState)outputFile << DroneState << " " << std::endl;

///////////////////////////////////////ACTIONS/////////////////////////////////////////////////
			// Take off / Landing
			if (key == ' ') {
				if (ardrone.onGround()) ardrone.takeoff();
				else			ardrone.landing();}

			// Automatic Landing when floor cam activated and above helipad
	if ((DroneState == 13)&&(AUTOland == true)&&(AUTOfloor == true)) {
	   if (ardrone.onGround()) ardrone.takeoff();
				else			{
					ardrone.landing();
					printf("FORCED LANDING\n");} }

//THIS WAS TAKEN FROM EXAMPLE CODE AND HAS NOT BEEN CHECKED OR TESTED
//	// Return true if drone altitude is under .5 m
//	bool IsTooLow() {
//	double altitude = ardrone.getAltitude();
//
//	if (altitude < 0.5) return true;
//	else return false;
//}
//
//	// Return true if drone altitude is over 1.5 m
//bool IsTooHigh() {
//	double altitude = ardrone.getAltitude();
//
//	if (altitude > 1.5) return true;
//	else return false;
//}
//
//	// Autonomous drone altitude control
//void KeepGoodAltitude() {
//	// Lower the drone
//	if (IsTooHigh()) LooseAltitude();
//
//	// Raise the drone
//	if (IsTooLow()) GainAltitude();
//}
			/*void GainAltitude() {
	vz = ALTITUDE_SPEED;
	cout << "gain alt" << endl;
}

void LooseAltitude() {
```

```
            vz = -ALTITUDE_SPEED;
            cout << "loose alt" << endl;
}*/


                        // Teleoperation Velocity Parameters
                        double vx = 0.0, vy = 0.0, vz = 0.0, vr = 0.0;
                        if (key == 0x260000) vx =  2.0; //Forward
                        if (key == 0x280000) vx = -2.0; //Backward
                        if (key == 'q')                  vr =  1.0; //Turn Left
                        if (key == 'e')                  vr = -1.0; //Turn Right
                        if (key == 'r')      vz =  1.0; //Raise Alititude
                        if (key == 'f')      vz = -1.0; //Lower Altitude
                        if (key == 0x250000) vy =  2.0; //Straife Left
                        if (key == 0x270000) vy = -2.0; //Straife Right

                        // Flight Animations Actions (commented out for saftey)
                        /* if (key == 'g') ardrone.setAnimation(ARDRONE_ANIM_PHI_M30_DEG,
1000);
                        if (key == 'n') ardrone.setAnimation(ARDRONE_ANIM_PHI_30_DEG,
1000);
                        if (key == 'k') ardrone.setAnimation(ARDRONE_ANIM_THETA_M30_DEG,
1000);
                        if (key == 'w') ardrone.setAnimation(ARDRONE_ANIM_THETA_30_DEG,
1000);
                        if (key == 's')
ardrone.setAnimation(ARDRONE_ANIM_THETA_20DEG_YAW_200DEG,  1000);
                        if (key == '4')
ardrone.setAnimation(ARDRONE_ANIM_THETA_20DEG_YAW_M200DEG, 1000);
                        if (key == '5') ardrone.setAnimation(ARDRONE_ANIM_TURNAROUND,
5000);
                        if (key == 'd')
ardrone.setAnimation(ARDRONE_ANIM_TURNAROUND_GODOWN,      5000);
                        if (key == '2') ardrone.setAnimation(ARDRONE_ANIM_YAW_SHAKE,
2000);
                        if (key == 'i') ardrone.setAnimation(ARDRONE_ANIM_YAW_DANCE,
5000);
                        if (key == '3') ardrone.setAnimation(ARDRONE_ANIM_PHI_DANCE,
5000);
                        if (key == 'p') ardrone.setAnimation(ARDRONE_ANIM_THETA_DANCE,
5000);
                        if (key == 't') ardrone.setAnimation(ARDRONE_ANIM_VZ_DANCE,
5000);
                        if (key == 'g') ardrone.setAnimation(ARDRONE_ANIM_WAVE,
5000);
                        if (key == 'b') ardrone.setAnimation(ARDRONE_ANIM_PHI_THETA_MIXED,
5000);
                        if (key == 'y')
ardrone.setAnimation(ARDRONE_ANIM_DOUBLE_PHI_THETA_MIXED,  5000);
```

```
			if (key == 'h') ardrone.setAnimation(ARDRONE_ANIM_FLIP_AHEAD,
15);
			if (key == '6') ardrone.setAnimation(ARDRONE_ANIM_FLIP_BEHIND,
15);
			if (key == 'u') ardrone.setAnimation(ARDRONE_ANIM_FLIP_LEFT,
15);
			if (key == 'j') ardrone.setAnimation(ARDRONE_ANIM_FLIP_RIGHT,
15);*/

			//DOWNWARDS (floor) FACING CAM
ACTIONS////////////////////////////////////////////////
			//Orthogonal Actions (Rooks Moves)
			//When helipad is detected slightly in front of the UAV, move forward slightly
			if ((DroneState == 8) && (AUTOfloor == true)) {
				vx = innerPower;
				printf("UAV MOVING forward*\n");}
			// When helipad is detected far away in front of the UAV, move forward rapidly
to avoid getting lost
			if ((DroneState == 3) && (AUTOfloor == true)){
				vx = outerPower;
				printf("UAV MOVING FORWARD*\n");}

			// When helipad is detected slightly behind the UAV, move backward slightly
			if ((DroneState == 18) && (AUTOfloor == true)) {
				vx = -innerPower;
				printf("UAV MOVING backward*\n");}
			// When helipad is detected far away behind the UAV, move backward rapidly
to avoid getting lost
			if ((DroneState == 23) && (AUTOfloor == true)) {
				vx = -outerPower;
				printf("UAV MOVING BACKWARD*\n");}

			// When helipad is detected slightly to the left of the UAV, move left slightly
			if ((DroneState == 12) && (AUTOfloor == true)) {
				vy = innerPower;
				printf("UAV MOVING left*\n");}
			// When helipad is detected far away to the left of the UAV, move left rapidly to
avoid getting lost
			if ((DroneState == 11) && (AUTOfloor == true)){
				vy = outerPower;
				printf("UAV MOVING LEFT*\n");}

			// When helipad is detected slightly to the right of the UAV, move right slightly
			if ((DroneState == 14) && (AUTOfloor == true)) {
				vy = -innerPower;
				printf("UAV MOVING right*\n");}
			// When helipad is detected far away to the right of the UAV, move right rapidly
to avoid getting lost
			if ((DroneState == 15) && (AUTOfloor == true)){
				vy = -outerPower;
```

```
            printf("UAV MOVING RIGHT*\n");}

    //Diagnal Manovoures (Kings Moves)
    //FRn = Front Right near
    if ((DroneState == 9) && (AUTOfloor == true)) {
                    vy = -innerPower;
                    printf("UAV MOVING right*\n");
                    vx = innerPower;
                    printf("UAV MOVING forward*\n");}
    //FRf = Front Right far
    if ((DroneState == 5) && (AUTOfloor == true)) {
                    vy = -outerPower;
                    printf("UAV MOVING RIGHT*\n");
                    vx = outerPower;
                    printf("UAV MOVING FORWARD*\n");}

    //BLn = Back Left near
    if ((DroneState == 17) && (AUTOfloor == true)) {
                    vy = innerPower;
                    printf("UAV MOVING left*\n");
                    vx = -innerPower;
                    printf("UAV MOVING backward*\n");}
    //BLf = Back Left far
    if ((DroneState == 21) && (AUTOfloor == true)) {
                    vy = outerPower;
                    printf("UAV MOVING LEFT*\n");
                    vx = -outerPower;
                    printf("UAV MOVING BACKWARD*\n");}

    //FLn = Front Left near
    if ((DroneState == 7) && (AUTOfloor == true)) {
                    vy = innerPower;
                    printf("UAV MOVING left*\n");
                    vx = innerPower;
                    printf("UAV MOVING forward*\n");}
    //FLf = Front Left far
    if ((DroneState == 1) && (AUTOfloor == true)) {
                    vy = outerPower;
                    printf("UAV MOVING LEFT*\n");
                    vx = outerPower;
                    printf("UAV MOVING FORWARD*\n");}

    //BRn = Back Right near
    if ((DroneState == 19) && (AUTOfloor == true)) {
                    vy = -innerPower;
                    printf("UAV MOVING right*\n");
                    vx = -innerPower;
                    printf("UAV MOVING backward*\n");}
    //BRf = Back Right far
    if ((DroneState == 25) && (AUTOfloor == true)) {
```

```
                    vy = -outerPower;
                    printf("UAV MOVING RIGHT*\n");
                    vx = -outerPower;
                    printf("UAV MOVING BACKWARD*\n");}


//Combination Manovoures (NotDiagnals)
//FRn = Front Right near
if ((DroneState == 4) && (AUTOfloor == true)) {
                    vy = -innerPower;
                    printf("UAV MOVING right*\n");
                    vx = outerPower;
                    printf("UAV MOVING FORWARD*\n");}
//FRf = Front Right far
if ((DroneState == 10) && (AUTOfloor == true)) {
                    vy = -outerPower;
                    printf("UAV MOVING RIGHT*\n");
                    vx = innerPower;
                    printf("UAV MOVING forward*\n");}

//BLn = Back Left near
if ((DroneState == 16) && (AUTOfloor == true)) {
                    vy = outerPower;
                    printf("UAV MOVING LEFT*\n");
                    vx = -innerPower;
                    printf("UAV MOVING backward*\n");}
//BLf = Back Left far
if ((DroneState == 22) && (AUTOfloor == true)) {
                    vy = innerPower;
                    printf("UAV MOVING left*\n");
                    vx = -outerPower;
                    printf("UAV MOVING BACKWARD*\n");}

//FLn = Front Left near
if ((DroneState == 2) && (AUTOfloor == true)) {
                    vy = innerPower;
                    printf("UAV MOVING left*\n");
                    vx = outerPower;
                    printf("UAV MOVING FORWARD*\n");}
//FLf = Front Left far
if ((DroneState == 6) && (AUTOfloor == true)) {
                    vy = outerPower;
                    printf("UAV MOVING LEFT*\n");
                    vx = innerPower;
                    printf("UAV MOVING forward*\n");}

//BRn = Back Right near
if ((DroneState == 20) && (AUTOfloor == true)) {
                    vy = -innerPower;
                    printf("UAV MOVING RIGHT*\n");
```

```
                        vx = -outerPower;
                        printf("UAV MOVING backward*\n");}
//BRf = Back Right far
if ((DroneState == 24) && (AUTOfloor == true)) {
                        vy = -innerPower;
                        printf("UAV MOVING right*\n");
                        vx = -outerPower;
                        printf("UAV MOVING BACKWARD*\n");}

//FORWARDS  (front) FACING CAM ACTIONS//////////////////////////////////////////////////
// When face is detected slightly above the UAV, move up slightly
 if ((DroneState == 8) && (AUTOfront == true)) {
            vz =  altitudePower;
            printf("UAV MOVING up\n");}

// When face is detected far above the UAV, move UP rapidly to stay at eye
level
if ((DroneState == 3) && (AUTOfront == true)){
            vz =  2*altitudePower;
            printf("UAV MOVING UP\n");}

// When face is detected slightly below the UAV, move down slightly
if ((DroneState == 18) && (AUTOfront == true)) {
            vz =  -altitudePower;
            printf("UAV MOVING down\n");}

// When helipad is detected far away behind the UAV, move backward rapidly
to avoid getting lost
if ((DroneState == 23) && (AUTOfront == true)) {
            vz =  2*-altitudePower;
            printf("UAV MOVING DOWN\n");}

// When helipad is detected slightly to the left of the UAV, move left slightly
if ((DroneState == 12) && (AUTOfront == true)) {
            vy = innerPower;
            printf("UAV MOVING left\n");}
// When helipad is detected far away to the left of the UAV, move left rapidly to
avoid getting lost
if ((DroneState == 11) && (AUTOfront == true)){
            vy = outerPower;
            printf("UAV MOVING LEFT\n");
//or
//          r =  1.0; //Turn Left to keep facing towards target
            //printf("UAV TURNING LEFT\n");
                                                                }

// When helipad is detected slightly to the right of the UAV, move right slightly
if ((DroneState == 14) && (AUTOfront == true)) {
            vy = -innerPower;
            printf("UAV MOVING right\n");}
```

```
// When helipad is detected far away to the right of the UAV, move right rapidly
to avoid getting lost
if ((DroneState == 15) && (AUTOfront == true)){
            vy = -outerPower;
            printf("UAV MOVING RIGHT\n");
//or
            //r =  -1.0; //Turn right to keep facing towards target
            //printf("UAV TURNING RIGHT\n");
                                                                                    }


//Diagnal Manovoures
//FRn = Front Right near
if ((DroneState == 9) && (AUTOfront == true)) {
                        vy = -innerPower;
                        printf("UAV MOVING right\n");
                        vz = altitudePower;
                        printf("UAV MOVING up\n");}
//FRf = Front Right far
if ((DroneState == 5) && (AUTOfront == true)) {
                        vy = -outerPower;
                        printf("UAV MOVING RIGHT\n");
                        vz = 1.5*altitudePower;
                        printf("UAV MOVING UP\n");}

//BLn = Back Left near
if ((DroneState == 17) && (AUTOfront == true)) {
                        vy = innerPower;
                        printf("UAV MOVING left\n");
                        vz = -altitudePower;
                        printf("UAV MOVING down\n");}
//BLf = Back Left far
if ((DroneState == 21) && (AUTOfront == true)) {
                        vy = outerPower;
                        printf("UAV MOVING LEFT\n");
                        vz = 1.5*-altitudePower;
                        printf("UAV MOVING DOWN\n");}

//FLn = Front Left near
if ((DroneState == 7) && (AUTOfront == true)) {
                        vy = innerPower;
                        printf("UAV MOVING left\n");
                        vz = altitudePower;
                        printf("UAV MOVING up\n");}
//FLf = Front Left far
if ((DroneState == 1) && (AUTOfront == true)) {
                        vy = outerPower;
                        printf("UAV MOVING LEFT\n");
                        vz = 1.5*altitudePower;
                        printf("UAV MOVING UP\n");}
```

```
//BRn = Back Right near
if ((DroneState == 19) && (AUTOfront == true)) {
                    vy = -innerPower;
                    printf("UAV MOVING right\n");
                    vz = -altitudePower;
                    printf("UAV MOVING down\n");}
//BRf = Back Right far
if ((DroneState == 25) && (AUTOfront == true)) {
                    vy = -outerPower;
                    printf("UAV MOVING RIGHT\n");
                    vz = 1.5*-altitudePower;
                    printf("UAV MOVING DOWN\n");}


//NEED TO DO THE "NOT" diagnals for the front camera just like i have done
for down camera
//Combination Manovoures (NotDiagnals)
//FRn = Front Right near
if ((DroneState == 4) && (AUTOfront == true)) {
                    vy = -innerPower;
                    printf("UAV MOVING right\n");
                    vz = 1.5*altitudePower;
                    printf("UAV MOVING UP\n");}


//FRf = Front Right far
if ((DroneState == 10) && (AUTOfront == true)) {
                    vy = -outerPower;
                    printf("UAV MOVING RIGHT\n");
                    vz = altitudePower;
                    printf("UAV MOVING up\n");}

//BLn = Back Left near
if ((DroneState == 16) && (AUTOfront == true)) {
                    vy = outerPower;
                    printf("UAV MOVING LEFT\n");
                    vz = -altitudePower;
                    printf("UAV MOVING down\n");}
//BLf = Back Left far
if ((DroneState == 22) && (AUTOfront == true)) {
                    vy = innerPower;
                    printf("UAV MOVING left\n");
                    vz = 1.5*-altitudePower;
                    printf("UAV MOVING DOWN\n");}

//FLn = Front Left near
if ((DroneState == 2) && (AUTOfront == true)) {
                    vy = innerPower;
                    printf("UAV MOVING left\n");
```

```
                        vz = 1.5*altitudePower;
                        printf("UAV MOVING UP\n");}
        //FLf = Front Left far
        if ((DroneState == 6) && (AUTOfront == true)) {
                        vy = outerPower;
                        printf("UAV MOVING LEFT\n");
                        vz = altitudePower;
                        printf("UAV MOVING up\n");}

        //BRn = Back Right near
        if ((DroneState == 20) && (AUTOfront == true)) {
                        vy = -innerPower;
                        printf("UAV MOVING RIGHT\n");
                        vz = -altitudePower;
                        printf("UAV MOVING down\n");}
        //BRf = Back Right far
        if ((DroneState == 24) && (AUTOfront == true)) {
                        vy = -innerPower;
                        printf("UAV MOVING right\n");
                        vz = 1.5*-altitudePower;
                        printf("UAV MOVING DOWN\n");}


                if (DroneState == 13){

ardrone.setLED(ARDRONE_LED_ANIM_GREEN,              0.5, 5);
                }
                else {
                        ardrone.setLED(ARDRONE_LED_ANIM_RED,
                        0.5, 5);

                }

        // printf("Area= %d\n ",targetArea/1000);
         //printf("Area= %d\n ",targetArea/500);

        //Forward & Backwards Traversing
        if ((DroneState >0) && (AUTOfront == true)){
                //If far away move closer
                if (targetArea/500 < 2){//was 10,000 //was 1
                                printf("Too FAR, Moving FORWARDS\n");
                                vx = distancePower;     }//move forwards
                //If too close move further away
                else if (targetArea/500 > 10){ //was 50,000 //was 16
                                printf("Too CLOSE, Moving BACKWARDS\n");
                                vx = 1.5*-distancePower;}//move backward


                //If distance is appropiate ie: 2->20 & drone is directly in front of target
                else if (DroneState == 13){
```

```
                                        targetReached = true;


                                        if (UpANDSwitch == true) {

                                        printf("*******TARGET CENTRALISED AT SAFE
DISTANCE*******\n");
                                        printf("***************SWITCHING
CAMERAS*****************\n");

                                        //Automaticaly changes cameras used & searching
mode
                                        static int mode = 1;
                                        ardrone.setCamera(++mode%4);
                                        ardrone.setCamera(++mode%4);//had to do it twice
for some reason
                                        AUTOfront = false; //Turn OFF automatic control for
FRONT camera
                                        AUTOfloor = true; //Turn ON automatic control for
FLOOR camera

                                        //Ground target HSV parameters
                                        minH = 85, maxH = 240;
                                        minS = 226, maxS = 255;
                                        minV = 39, maxV = 166;
                                        printf("Image Procssing Parameters Have Been
Changed For Ground Target\n");

                                        //Move up to aquire better helipad view
                                        for (int a = 1; a <10000 ; a++){
                                        vz = 12*altitudePower;
                                        ardrone.move3D(vx, vy, vz, vr);}
                                        printf("UAV MOVED UP for better landing view\n");

                                        }//end of upANDswitch

                        }//end IF drone is centrtalised at the appropiate distance to the target
                }//end IF drone state detected ie: not 0

                ardrone.move3D(vx, vy, vz, vr); //move the uav according to dynamic preset
variables

/////////////////////////////////////Configuration Options/////////////////////////////////////////
//THIS COULD BE REMOVED WHEN TESTED THAT THE HSV PARAMS ARE CHANGING
DEPENDANT UPON CAMERA USED
                //if (AUTOfloor == true){
                //      // GUI HSV Threshold slider bar limits 0 -> 255 (Fixed Ground Target)
                //      int minH = 85, maxH = 240;
                //      int minS = 226, maxS = 255;
                //      int minV = 39, maxV = 166;}
```

```
                    //          //printf("Image Procssing Parameters Have Been Changed For
Ground Target\n");}
                    //else{
                    //          // GUI HSV Threshold slider bar limits 0 -> 255 (Fixed Forward Target)
                    //          //int minH = 90, maxH = 229;
                    //          //int minS = 187, maxS = 255;
                    //          //int minV = 39, maxV = 166;

                    //          int minH = 0, maxH = 0;
                    //          int minS = 0, maxS = 0;
                    //          int minV = 0, maxV = 0;
                    //          }

      // Change camera
      static int mode = 0;
      if (key == 'c') ardrone.setCamera(++mode%4);

                    // Toggle Visual Stabilisation for DOWNWARDS Camera Reactive Behaviour
                    if  ((key == 'a') && (AUTOfloor == false)){ AUTOfloor =  true; //toggle ON
                              printf("AUTOfloor ON\n");
                                                  //Ground target HSV parameters
                                                  minH = 85, maxH = 240;
                                                  minS = 226, maxS = 255;
                                                  minV = 39, maxV = 166;
                                                  printf("Image Procssing Parameters Have Been
Changed For GROUND FACING Target\n");
                                                  //ardrone.setCamera(++mode%4);          //switch
camera when near target

          }

                    else if  ((key == 'a') && (AUTOfloor == true)){ AUTOfloor =  false; //toggle OFF
                              printf("AUTOfloor OFF\n");}
                              // ardrone.setCamera(++mode%4);




                    // Toggle Visual Stabilisation for FORWARDS Camera Reactive Behaviour
                    if  ((key == 'z') && (AUTOfront == false)){ AUTOfront =  true; //toggle ON
                              printf("AUTOfront ON\n");

                                                  minH = 0, maxH = 128; //max H best at 155 for
ocluded corner (Fixed Forward Target)
                                                  minS = 136, maxS = 235;
                                                  minV = 0, maxV = 103;
                                                  printf("Image Procssing Parameters Have Been
Changed For FORWARD FACING Target\n");
                                                                                    }
```

```
                else if  ((key == 'z') && (AUTOfront == true)){ AUTOfront =  false; //toggle OFF
                        printf("AUTOfront OFF\n");}

                        // Toggle Automatic Landing for Downwards Camera Reactive
Behaviour
                if  ((key == 'l') && (AUTOland == false)){ AUTOland =  true; //toggle ON
                        printf("AUTOland ON\n");}
                else if  ((key == 'l') && (AUTOland == true)){ AUTOland =  false; //toggle OFF
                        printf("AUTOland OFF\n");}

                // Toggle Visual Stabilisation for FORWARDS Camera Reactive Behaviour
                if  ((key == 'o') && (rec == false)){ rec =  true; //toggle ON
                        printf("Started Recoring!\n");}
                else if  ((key == 'o') && (rec == true)){ rec =  false; //toggle OFF
                        printf("Stopped Recording\n");}

                // Release memory
                cvReleaseMemStorage(&contourStorage);

    // Show recording state
    if (rec) {
       static CvFont font = cvFont(1.0);
       cvPutText(image, "REC", cvPoint(600, 20), &font, CV_RGB(255,0,0));}


        //Trying to display srings of data to HUD
                //          ostringstream str2; // string stream

        //          // Battery
        //str2 << ardrone.getBatteryPercentage();
        //putText(result, str2.str(), Point(180, 33), CV_FONT_HERSHEY_PLAIN, 2,
CV_RGB(0, 250, 0), 2);

                if (altitude >0 && altitude <= 0.5) {
                     static CvFont font2 = cvFont(1.0);
       cvPutText(image, "0.5m High", cvPoint(10, 20), &font2, CV_RGB(255,0,0));}

                                if (altitude > 0.5 && altitude <= 1) {
                     static CvFont font2 = cvFont(1.0);
       cvPutText(image, "1m High", cvPoint(10, 20), &font2, CV_RGB(255,0,0));}


                                if (altitude >1 && altitude <= 1.5) {
                     static CvFont font2 = cvFont(1.0);
       cvPutText(image, "1.5m High", cvPoint(10, 20), &font2, CV_RGB(255,0,0));}

                                        if (altitude >1.5 && altitude <= 2) {
                     static CvFont font2 = cvFont(1.0);
       cvPutText(image, "2m High", cvPoint(10, 20), &font2, CV_RGB(255,0,0));}
```

```
                                        if (altitude >2 && altitude <= 2.5) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "2.5m High", cvPoint(10, 20), &font2, CV_RGB(255,0,0));}

                                        //2.8 is maximum height of robotics lab
                                                if (altitude >2.5 && altitude <= 2.8) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "3m High", cvPoint(10, 20), &font2, CV_RGB(255,0,0));}

                                                        if (altitude >2.8) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "TOO High", cvPoint(10, 20), &font2, CV_RGB(255,0,0));}


                                //BATERY WARNING

                                                if (bat >=0 && bat < 5) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "LAND NOW", cvPoint(275, 20), &font2, CV_RGB(255,0,0));}

                                if (bat >=5 && bat <= 10) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "Bat = 10% ", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                                if (bat >10 && bat <= 20) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "Bat = 20%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}


                                if (bat >20 && bat <= 30) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "Bat = 30%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                                if (bat >30 && bat <= 40) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "Bat = 40%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                                if (bat >40 && bat <=50) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "Bat = 50%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                                if (bat >50 && bat <= 60) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "Bat = 60%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                                if (bat >60 && bat <= 70) {
                        static CvFont font2 = cvFont(1.0);
cvPutText(image, "Bat = 70%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}
```

```
                        if (bat >70 && bat <= 80) {
                static CvFont font2 = cvFont(1.0);
    cvPutText(image, "Bat = 80%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                        if (bat >80 && bat <= 90) {
                static CvFont font2 = cvFont(1.0);
    cvPutText(image, "Bat = 90%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                        if (bat >90 && bat < 100) {
                static CvFont font2 = cvFont(1.0);
    cvPutText(image, "Bat = 99%", cvPoint(270, 20), &font2, CV_RGB(255,0,0));}

                        if (bat == 100) {
                static CvFont font2 = cvFont(1.0);
    cvPutText(image, "Fully Charged", cvPoint(260, 20), &font2, CV_RGB(255,0,0));}


            //Verticle Lines superimposed on video feed
            cvLine( image, cvPoint( 0, 0 ), cvPoint( 0, 357), cvScalar( 0, 255, 0 ),  2, 8 );
            cvLine( image, cvPoint( 127, 0 ), cvPoint( 127, 357), cvScalar( 0, 255, 0 ),  2, 8
);
            cvLine( image, cvPoint( 255, 0 ), cvPoint( 255, 357), cvScalar( 0, 255, 0 ),  2, 8
);
            cvLine( image, cvPoint( 383, 0 ), cvPoint( 383, 357), cvScalar( 0, 255, 0 ),  2, 8
);
            cvLine( image, cvPoint( 510, 0 ), cvPoint( 510, 357), cvScalar( 0, 255, 0 ),  2, 8
);
            cvLine( image, cvPoint( 638, 0 ), cvPoint( 638, 357), cvScalar( 0, 255, 0 ),  2, 8
);

            //Horizontal Lines superimposed on video feed
            cvLine( image, cvPoint( 0, 0 ), cvPoint( 638, 0), cvScalar( 0, 255, 0 ),  2, 8 );
            cvLine( image, cvPoint( 0, 71 ), cvPoint( 638, 71), cvScalar( 0, 255, 0 ),  2, 8 );
            cvLine( image, cvPoint( 0, 143 ), cvPoint( 638, 143), cvScalar( 0, 255, 0 ),  2, 8
);
            cvLine( image, cvPoint( 0, 214 ), cvPoint( 638, 214), cvScalar( 0, 255, 0 ),  2, 8
);
            cvLine( image, cvPoint( 0, 285 ), cvPoint( 638, 285), cvScalar( 0, 255, 0 ),  2, 8
);
            cvLine( image, cvPoint( 0, 357 ), cvPoint( 638, 357), cvScalar( 0, 255, 0 ),  2, 8
);

    // Display the image
    cvShowImage("camera", image);

                // Write video frames when told to
            if (rec){
                cvWriteFrame(video, image);}

            // Release images
```

```
                        cvReleaseImage(&hsv);
                        cvReleaseImage(&binalized);

                        // Remember last state visited
                        prevDroneState = DroneState;

            }// End Main While Loop

            // Save the video
    cvReleaseVideoWriter(&video);

            // Close the DroneStateLog.txt file
            outputFile.close();

    // See you
    ardrone.close();

    return 0;
} //END MAIN PROG
```

Appendix C

Case Study Research Prototype

ASPEC

```
//ASPEC v2.9 - TODO - Feed Count & Only feed when bat>=12v potdivider
#include <RTCTimedEvent.h>
#include <Wire.h>
#include "RTClib.h"
#include <dht11.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <FuzzyRule.h>
#include <FuzzyComposition.h>
#include <Fuzzy.h>
#include <FuzzyRuleConsequent.h>
#include <FuzzyOutput.h>
#include <FuzzyInput.h>
#include <FuzzyIO.h>
#include <FuzzySet.h>
#include <FuzzyRuleAntecedent.h>
RTC_DS1307 rtc; //must be specified after the includes or causes compilation errors

#define ONE_WIRE_BUS 10 // Temp Data wire is plugged into pin 10 on the Arduinowas 3
// Setup a oneWire instance to communicate with any OneWire devices
OneWire oneWire(ONE_WIRE_BUS);
// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);
DeviceAddress insideThermometer = { 0x28, 0xF1, 0x61, 0xD2, 0x05, 0x00, 0x00, 0xB4 };
DeviceAddress outsideThermometer = { 0x28, 0x19, 0xFE, 0xD2, 0x05, 0x00, 0x00, 0xE4 };

//Set scheduled feeding times Hr,Min
int SetFeedTimeHrs  = 13;
int SetFeedTimeMins = 30;
int SetFeedDay      = 3; //Day: 1=Mon, 2=Tues, 3=Wed, 4=Thur, 5=Friday, 6=Sat, 7=Sun

int A_BML = 100;//ml       A&B Range = 10ML -> 200ML in multiples of 10 (Enter total weekly
feed ammount)
int B_52ML = 50;//ml       1st Addative Range = 10ML -> 200ML in multiples of 10
int VoodooJuiceML = 50;//ml  help roots
int BigBudML = 50;//ml  help roots
int OverDriveML = 50;//ml  help roots
//to do: CarboLoadML = 50
int FlushML = 50;//ml  Swap voodo juice with royal flush before septemeber

int numofA_B10mlDoses = A_BML/10; //had to declare this again because not in scope
int numofB_5210mlDoses = B_52ML/10; //had to declare this again because not in scope
int numofVoodooJuice10mlDoses = VoodooJuiceML/10; //had to declare this again because
not in scope
int numofBigBud10mlDoses = BigBudML/10; //had to declare this again because not in scope
int numofOverDrive10mlDoses = OverDriveML/10; //had to declare this again because not in
scope
int numofFlush10mlDoses = FlushML/10; //had to declare this again because not in scope
int Pump10mlTimeDelay = 341; //and this
int settleTime = 5000;
int FeedCount;

//Specify pin numbers of each connected component
int led = 13; // Pin 13 has an LED connected on most Arduino boards.
int RLY8 = 9; //Bloom-A + Bloom-B Pumps (Symultanious Activation)
int RLY7 = 8; //B-52 (Vitamin Pump)
int RLY6 = 7; //VoodoJuice / CarboLoad / Royal Flush
int RLY5 = 6; //Big-Bud
int RLY4 = 5; //Overdrive
int RLY3 = 4; //Dual Cooling Fans
int RLY2 = 3; //NFT Pump
int RLY1 = 2; //Flush Pump
int AirPumpRLY= 52;
#define humidityPin 53

// Instantiating an object of library
Fuzzy* fuzzy = new Fuzzy();

FuzzySet* concentrated = new FuzzySet(0, 20, 20, 40);
FuzzySet* safe = new FuzzySet(30, 50, 50, 70);
FuzzySet* dilute = new FuzzySet(60, 80, 100, 100);

FuzzySet* alkali = new FuzzySet(0, 0, 0, 0);
FuzzySet* neutral = new FuzzySet(1, 10, 10, 20);
FuzzySet* normal = new FuzzySet(15, 30, 30, 50);
FuzzySet* acidic = new FuzzySet(45, 60, 70, 70);

FuzzySet* cold = new FuzzySet(-30, -30, -20, -10);
FuzzySet* good = new FuzzySet(-15, 0, 0, 15);
FuzzySet* hot = new FuzzySet(10, 20, 30, 30);

//Declare Variables
int PrevHumidity, Humidity, hCheck;
//int MinHumidity, MaxHumidity;
dht11 DHT11;
float myTemp,myTemp2; //Changes when updateTemperature method is repetadly run
int Temp,Temp2;
int MinTemp = 100;
int MinTemp2 = 100;
int MaxTemp = myTemp;
int MaxTemp2 = myTemp2;
float PrevMinTemp, PrevMaxTemp,PrevMinTemp2, PrevMaxTemp2;
bool AirIsON, NFTIsON, FlushIsON;
//////////////////////////////////////////////////////////SETUP/////////////////////////////////////////////////////////////////////////////////
void setup() {
 Serial.begin(57600);
 // Start up the library
```

```
  sensors.begin();
  //set the resolution to 10 bit (good enough?)
  sensors.setResolution(insideThermometer, 10);
  sensors.setResolution(outsideThermometer, 10);

Serial.print("Sucsessful Feeds = ");
Serial.print(FeedCount);
Serial.print(". Feed set to: ");
Serial.print(SetFeedTimeHrs);
Serial.print(":");
Serial.print(SetFeedTimeMins);
Serial.print(" on ");

 if (SetFeedDay == 7)
    Serial.print("Sun");
  if (SetFeedDay == 1)
    Serial.print("Mon");
  if (SetFeedDay == 2)
    Serial.print("Tues");
  if (SetFeedDay == 3)
    Serial.print("Wednes");
  if (SetFeedDay == 4)
    Serial.print("Thurs");
  if (SetFeedDay == 5)
    Serial.print("Fri");
  if (SetFeedDay == 6)
    Serial.print("Satur");

  Serial.println("day ");

  //---------------Manualy set the time (KEEP COMMENTED)------------------- set at 25 seconds to
to compile & upload
  // ALWAYS REMEMBER TO RE-UPLOAD A COMMENTED OUT VERSION IMMEDIATLY
AFTER CHANGING UNCOMMENTED TIME
   //RTCTimedEvent.time.hour = 11;
   //RTCTimedEvent.time.minute = 25; //set at same as target time minute
   //RTCTimedEvent.time.second = 0; //Press upload at 10 sec to the hour
   //RTCTimedEvent.time.day = 30; //date
   //RTCTimedEvent.time.month = 10; //Jan=1, Feb=2, March=3
   //RTCTimedEvent.time.year = 2015;
   //RTCTimedEvent.time.dayOfWeek  = 5; //1=Monday, 2=Tues 3= Wednesday 4=Thursday
   //RTCTimedEvent.writeRTC();//never use this or wil reset time

  if (! rtc.isrunning()) {
    Serial.println("RTC is NOT running!");
    Serial.print("Please check time and reset using code above if nessecery");
    // following line sets the RTC to the date & time this sketch was compiled
   // rtc.adjust(DateTime(__DATE__, __TIME__));//NOTE:does not set day of week or date/time
in RTCTimedEvent Lib
  }
```

```
//following line sets the RTC to the date & time of PC when compiled (19sec behind)
//rtc.adjust(DateTime(__DATE__, __TIME__)); //KEEP COMMENTED

//initial buffer for 13 timers
RTCTimedEvent.initialCapacity = sizeof(RTCTimerInformation)*13;

//At Lunchtime on every Saturday of April, Call BloomFeed1Call
RTCTimedEvent.addTimer(SetFeedTimeMins,
                SetFeedTimeHrs,
                TIMER_ANY, //Date
                4,  //Month: 4=April
                SetFeedDay,
                BloomFeed1Call);

RTCTimedEvent.addTimer(SetFeedTimeMins,
                SetFeedTimeHrs,
                TIMER_ANY, //Date
                5,  //Month:5 =May
                SetFeedDay,
                BloomFeed2Call);

RTCTimedEvent.addTimer(SetFeedTimeMins,
                SetFeedTimeHrs,
                TIMER_ANY, //Date
                6,  //Month: 6=June
                SetFeedDay,
                BloomFeed3Call);

RTCTimedEvent.addTimer(SetFeedTimeMins,
                SetFeedTimeHrs,
                TIMER_ANY, //Date
                7,  //Month:7=July
                SetFeedDay,
                BloomFeed4Call);

RTCTimedEvent.addTimer(SetFeedTimeMins,
                SetFeedTimeHrs,
                TIMER_ANY,
                8,  //Month: 8=August
                SetFeedDay,
                BloomFeed5Call);

RTCTimedEvent.addTimer(SetFeedTimeMins,
                SetFeedTimeHrs,
                TIMER_ANY, //Date
                9,  //Month: 9=September
                SetFeedDay,
                BloomFeed6Call);
```

```
//Turn Air Pump ON at 9am
RTCTimedEvent.addTimer(00, //minute
            14, //hour
            TIMER_ANY, //Date
            TIMER_ANY,  //Month
            TIMER_ANY,  //DayOfWeek
            AirCallON);

//Turn Air Pump OFF at 10pm
RTCTimedEvent.addTimer(01, //minute
            14, //hour
            TIMER_ANY, //Date
            TIMER_ANY,  //Month
            TIMER_ANY,  //DayOfWeek
            AirCallOFF);

//Turn NFT Pump ON at 6am
RTCTimedEvent.addTimer(0, //minute
            6, //hour
            TIMER_ANY, //Date
            TIMER_ANY,  //Month
            TIMER_ANY,  //DayOfWeek
            NFTCallON);

//Turn NFT Pump OFF at 9pm
RTCTimedEvent.addTimer(34, //minute
            6, //hour
            TIMER_ANY, //Date
            TIMER_ANY,  //Month
            TIMER_ANY,  //DayOfWeek
            NFTCallOFF);
//*******************************************************
//Turn Flush Pump ON at 11am on the first day of each month (20minFlush)
RTCTimedEvent.addTimer(47, //minute
            14, //hour
            23, //Date //should be 1
            TIMER_ANY,  //Month
            TIMER_ANY,  //DayOfWeek
            FlushCallON);

//Turn Flush Pump OFF at 11:20am on the first day of each month (20minFlush)
RTCTimedEvent.addTimer(50, //minute
            14, //hour
            23, //Date
            TIMER_ANY,  //Month
            TIMER_ANY,  //DayOfWeek
            FlushCallOFF);

//Display date/time & Temperatures (used for data logging & Serial Monitoring)
RTCTimedEvent.addTimer(TIMER_ANY, //minute
                TIMER_ANY, //hour
                TIMER_ANY, //Date
                TIMER_ANY, //Month
                TIMER_ANY, //DayOfWeek
                minuteCall);

// FuzzyInput
FuzzyInput* inputEC = new FuzzyInput(1);
EC->addFuzzySet(concentrated);
EC->addFuzzySet(safe);
EC->addFuzzySet(dilute);

fuzzy->addFuzzyInput(EC);

// FuzzyInput
FuzzyInput* inputPH = new FuzzyInput(2);
PH->addFuzzySet(alkali);
PH->addFuzzySet(neutral);
PH->addFuzzySet(normal);
PH->addFuzzySet(acidic);

fuzzy->addFuzzyInput(PH);

// FuzzyInput
FuzzyInput* temperature = new FuzzyInput(3);
temperature->addFuzzySet(cold);
temperature->addFuzzySet(good);
temperature->addFuzzySet(hot);

fuzzy->addFuzzyInput(temperature);

// FuzzyOutput
FuzzyOutput* risk = new FuzzyOutput(1);

FuzzySet* minimum = new FuzzySet(0, 20, 20, 40);
risk->addFuzzySet(minimum);
FuzzySet* average = new FuzzySet(30, 50, 50, 70);
risk->addFuzzySet(average);
FuzzySet* maximum = new FuzzySet(60, 80, 80, 100);
risk->addFuzzySet(maximum);

fuzzy->addFuzzyOutput(risk);

// FuzzyOutput
// adding nutrient dosage as output too
FuzzyOutput* outputDosage = new FuzzyOutput(2);

FuzzySet* stopedOut = new FuzzySet(0, 0, 0, 0);
outputDosage->addFuzzySet(noDoseOut);
FuzzySet* slowOut = new FuzzySet(1, 10, 10, 20);
```

```
  outputDosage->addFuzzySet(lowDoseOut);
  FuzzySet* normalOut = new FuzzySet(15, 30, 30, 50);
  outputDosage->addFuzzySet(normalDoseOut);
  FuzzySet* quickOut = new FuzzySet(45, 60, 70, 70);
  outputDosage->addFuzzySet(highDoseOut);

  fuzzy->addFuzzyOutput(outputDose);

  // Building FuzzyRule
  FuzzyRuleAntecedent* ECConcentratedAndPHAlkali = new FuzzyRuleAntecedent();
  ecConcentratedAndPHAlkali ->joinWithAND(concentrated, highDose);

  FuzzyRuleAntecedent* temperatureCold = new FuzzyRuleAntecedent();
  temperatureCold->joinSingle(cold);

  FuzzyRuleAntecedent* ifPHAlkaliAndECHighDoseOrTemperatureCold = new
FuzzyRuleAntecedent();
  ifECConcentratedAndPHAlkaliOrTemperatureCold-
>joinWithOR(ECConcentratedAndPHAlkali, temperatureCold);

  FuzzyRuleConsequent* thenRisMaximumAndPHAcidic = new FuzzyRuleConsequent();
  thenRisMaximumAndPHAcidic->addOutput(maximum);
  thenRisMaximumAndPHAcidic->addOutput(lowDoseOut);

  FuzzyRule* fuzzyRule1 = new FuzzyRule(1,
ifECConcentratedAndDosageHighDoseOrTemperatureCold,
thenRisMaximumAndDosageHighDoseOut);
  fuzzy->addFuzzyRule(fuzzyRule1);

  // Building FuzzyRule
  FuzzyRuleAntecedent* ECConcentratedAndPHNeutral = new FuzzyRuleAntecedent();
  ECSafeAndPHNormal->joinWithAND(safe, normal);
  FuzzyRuleAntecedent* ifECSafeAndPHNormalOrTemperatureGood = new
FuzzyRuleAntecedent();
  ifECSafeAndPHNormalOrTemperatureGood->joinWithOR(ECSafeAndPHSpeedNormal,
good);

  FuzzyRuleConsequent* thenRiskAverageAndECNormal = new FuzzyRuleConsequent();
  thenRiskAverageAndPHNormal->addOutput(average);
  thenRiskAverageAndPHNormal->addOutput(normalOut);

  FuzzyRule* fuzzyRule2 = new FuzzyRule(2, ifECSafeAndPHNormalOrTemperatureGood,
thenRiskAverageAndPHNormal);
  fuzzy->addFuzzyRule(fuzzyRule2);

  // Building FuzzyRule
  FuzzyRuleAntecedent* ECDiluteAndPHNeutral = new FuzzyRuleAntecedent();
  ECConcentratedAndPHNeutral->joinWithAND(concentrated, neutral);
  FuzzyRuleAntecedent* ifECDiluteAndPHNeutralOrTemperatureHot = new
FuzzyRuleAntecedent();
```

```
  ifECDiluteAndPHNeutralOrTemperatureHot->joinWithOR(ECDiluteAndPHNeutral, hot);

  FuzzyRuleConsequent* thenRiskMinimumDoseHigh = new FuzzyRuleConsequent();
  thenRiskMinimumDoseHigh->addOutput(minimum);
  thenRiskMinimumDoseHigh->addOutput(HighDoseOut);

  FuzzyRule* fuzzyRule3 = new FuzzyRule(3, ifECDiluteAndPHNeutralOrTemperatureHot,
thenRiskMinimumDoseHigh);
  fuzzy->addFuzzyRule(fuzzyRule3);

  // Initialize digital pins as an output
  pinMode(led, OUTPUT);
  pinMode(RLY8, OUTPUT); //Bloom-A + Bloom-B Pumps (Symultanious Activation)
  pinMode(RLY7, OUTPUT); //B-52 (Vitamin Pump)
  pinMode(RLY6, OUTPUT); //VoodoJuice (Root-Regen Pump)
  pinMode(RLY5, OUTPUT); //Big-Bud
  pinMode(RLY4, OUTPUT); //Overdrive
  pinMode(RLY3, OUTPUT); //Dual Cooling Fans
  pinMode(RLY2, OUTPUT); //NFT Pump
  pinMode(RLY1, OUTPUT); //Flush Pump
  pinMode(AirPumpRLY, OUTPUT); //3.3v Air Bubblestone Pump (Cools nutrient solution)
  //pinMode(humidityPin, INPUT); //Dont think this is needed, maybe pins default to INPUT
mode

  // Initialise relays to HIGH to prevent them from activating pumps
  digitalWrite(RLY8, HIGH); //Bloom-A + Bloom-B Pumps (Symultanious Activation)
  digitalWrite(RLY7, HIGH); //B-52 (Vitamin Pump)
  digitalWrite(RLY6, HIGH); //VoodoJuice (Root-Regen Pump)
  digitalWrite(RLY5, HIGH); //Big-Bud
  digitalWrite(RLY4, HIGH); //Overdrive
  digitalWrite(RLY3, HIGH); //Dual Cooling Fans
  digitalWrite(RLY2, LOW);  //NFT Pump (Activated on startup)
  digitalWrite(RLY1, HIGH);  //Flush Pump
  digitalWrite(AirPumpRLY, LOW); //3.3v Air Cooling Bubblestone Pump (This causes air pump
to run when arduino is powered up or reset)
}//end of SETUP
///////////////////////////////////////////////////////LOOP///////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////

// loop routine runs over and over again forever:
void loop() {
  RTCTimedEvent.loop();
  delay(1000); //Was 2000, may caus instability not sure

 // RELAYTEST();

//Dont pump NFT when flushing the system
if (FlushIsON == 1){
  digitalWrite(RLY2, HIGH);  //NFT Pump OFF
    digitalWrite(RLY1, LOW);  //Flush Pump ON
    }
    else
    {
      digitalWrite(RLY2, LOW);  //NFT Pump ON
    digitalWrite(RLY1, HIGH);  //Flush Pump OFF
    }

  //for constant monitoring of temperature rather than timer
  updateTemperature(insideThermometer);
    updateTemperature2(outsideThermometer);

  Temp = myTemp;
  if ((myTemp < MinTemp) && (myTemp != 0)) {
    // Serial.print("MyTemp < MinTemp\n\r");
    PrevMinTemp = myTemp;
    MinTemp = PrevMinTemp;
    //   Serial.print(MinTemp);
    //  Serial.print("HAS JUST BEEN SET FOR MIN TEMP EVER RECORED\n\r");
  }

  Temp2 = myTemp2;
  if ((myTemp2 < MinTemp2) && (myTemp2 != 0)) {
    // Serial.print("MyTemp < MinTemp\n\r");
    PrevMinTemp2 = myTemp2;
    MinTemp2 = PrevMinTemp2;
    //   Serial.print(MinTemp);
    //  Serial.print("HAS JUST BEEN SET FOR MIN TEMP EVER RECORED\n\r");
  }

  //  Serial.print(MinTemp);
  //  Serial.print(" MIN\n\r");
  //  Serial.print(MaxTemp);
  //  Serial.print(" MAX\n\r");

  if (myTemp > MaxTemp) {
    // Serial.print("MyTemp2 > MaxTemp2\n\r");
    PrevMaxTemp = myTemp;
    MaxTemp = PrevMaxTemp;
    //  Serial.print(MaxTemp2);
    //   Serial.print("HAS JUST BEEN SET FOR MAX TEMP2 EVER RECORED\n\r");
  }
  if (myTemp2 > MaxTemp2) {
    // Serial.print("MyTemp2 > MaxTemp2\n\r");
    PrevMaxTemp2 = myTemp2;
    MaxTemp2 = PrevMaxTemp2;
    //  Serial.print(MaxTemp2);
    //   Serial.print("HAS JUST BEEN SET FOR MAX TEMP2 EVER RECORED\n\r");
  }

 // Serial.print("Getting temperatures...\n\r");
```

```
  sensors.requestTemperatures();

 // if (insideThermometer != previnsideThermometer){
//  Serial.print("Inside temperature is: ");
//  printTemperature(insideThermometer);
//  Serial.print("\n\r");

//  Serial.print("Outside temperature is: ");
//  printTemperature(outsideThermometer);
//  Serial.print("\n\r");
 // Serial.print("\n\r\n\r");

   //RELAYTEST(); //Execute Relay Test Function

//----------------Humidity Output------------------
int HumidityCallibration = -10;

  hCheck = DHT11.read(humidityPin);
  if(hCheck != 0)
    Humidity = 255; //Must be an error
  else
    Humidity = DHT11.humidity;
    Humidity = Humidity + HumidityCallibration; //my calibration

  String s1;
  if(Humidity == 255 + HumidityCallibration)
    s1 = "HUMIDITY ERROR - Emergency DualFans ON";
  else
    s1 = String(Humidity) + "%Humidity";

  if(PrevHumidity!=Humidity){
  Serial.println(s1);
  digitalWrite(led, HIGH);}    // Initialise LED to high to turn on the LED

//When too humid inside the PinderPonics, Begin ventilation
  //Reccomeded humidity for flowering is 40% - 60%
  //Therefor this value should be 60
  if (Humidity > 60 || myTemp2 > 29)
      digitalWrite(RLY3, LOW);   // Dual Ventilation Fans ON
  else  digitalWrite(RLY3, HIGH);  // Dual Ventilation Fans OFF
      delay(200);   // wait for a second otherwise very buggy

  //Reccomended temp for nutrient solution tank is 15-24 oC
  if (AirIsON == 1 || myTemp > 24)//&& Inside temperature > 18oC
    digitalWrite(AirPumpRLY, HIGH);   // Air Stone Pump
  else digitalWrite(AirPumpRLY, LOW);  // Dual Ventilation Fans OFF
      delay(200);   // wait for a second otherwise very buggy

fuzzy->setInput(1, 10);
  fuzzy->setInput(2, 30);
  fuzzy->setInput(3, -15);

  fuzzy->fuzzify();

  Serial.print("EC: ");
  Serial.print(concentrated->getPertinence());
  Serial.print(", ");
  Serial.print(safe->getPertinence());
  Serial.print(", ");
  Serial.println(dilute->getPertinence());

  Serial.print("Dose: ");
  Serial.print(noDose->getPertinence());
  Serial.print(", ");
  Serial.print(lowDose->getPertinence());
  Serial.print(", ");
  Serial.print(normalDose->getPertinence());
  Serial.print(", ");
  Serial.println(highDose->getPertinence());

  Serial.print("Temperature: ");
  Serial.print(cold->getPertinence());
  Serial.print(", ");
  Serial.print(good->getPertinence());
  Serial.print(", ");
  Serial.println(hot->getPertinence());

  float output1 = fuzzy->defuzzify(1);
  float output2 = fuzzy->defuzzify(2);

  Serial.print("Risk output: ");
  Serial.print(output1);
  Serial.print(", Dose output: ");
  Serial.println(output2);

      //Activate NFT Pump
 //if (NFTIsON ==1){
  //    digitalWrite(RLY2, LOW);   // Dual Ventilation Fans ON
  //    Serial.print("NFT IS ON");}
  //  else { digitalWrite(RLY2, HIGH);  // Dual Ventilation Fans OFF
   //       Serial.print("NFT IS OFF");
    //     delay(200);}   // wait for a second otherwise very buggy
//
// //Activate Flush Pump
// if (Humidity > 27)
//      digitalWrite(RLY1, LOW);   // Dual Ventilation Fans ON
//    else digitalWrite(RLY1, HIGH);  // Dual Ventilation Fans OFF
//        delay(200);   // wait for a second otherwise very buggy

//-----------------------------------------------------------
```

```
  //Show time every update
//DateTime now = rtc.now();
//
//    Serial.print(now.year(), DEC);
//    Serial.print('/');
//    Serial.print(now.month(), DEC);
//    Serial.print('/');
//    Serial.print(now.day(), DEC);
//    Serial.print(' ');
//    Serial.print(now.hour(), DEC);
//    Serial.print(':');
//    Serial.print(now.minute(), DEC);
//    Serial.print(':');
//    Serial.print(now.second(), DEC);
//    Serial.println();

//at the end of the loop store current humiudity to old humidity
PrevHumidity = Humidity;
digitalWrite(led, LOW);    // Turn LED off after being on from humidity change
}

//-------------------------Timer Functions--------------------------------
void minuteCall(RTCTimerInformation* Sender) {
  Serial.print("Time: ");
  Serial.print(RTCTimedEvent.time.hour, DEC);
  Serial.print(":");
  Serial.print(RTCTimedEvent.time.minute, DEC);
  Serial.print(":");
  Serial.print(RTCTimedEvent.time.second, DEC);
  Serial.print(" Date: ");
  Serial.print(RTCTimedEvent.time.day, DEC);
  Serial.print("/");
  Serial.print(RTCTimedEvent.time.month, DEC);
  Serial.print("/");
  Serial.println(RTCTimedEvent.time.year, DEC);

  Serial.print("Greenhouse temperature is: ");
  printTemperature2(outsideThermometer);
  Serial.print("\n\r");

  Serial.print("Tank temperature is: ");
  printTemperature(insideThermometer);
  Serial.print("\n\r");

Serial.print("C:");
 Serial.print(MinTemp);
    Serial.print(" Tank MIN Temp  ");
    Serial.print("C:");
    Serial.print(MaxTemp);
    Serial.print(" Tank MAX Temp\n\r");

    Serial.print("C:");
  Serial.print(MinTemp2);
    Serial.print(" Greenhouse MIN Temp  ");
    Serial.print("C:");
    Serial.print(MaxTemp2);
    Serial.print(" Greenhouse MAX Temp\n\n\r");
}

//April Bloom Feed
void BloomFeed1Call(RTCTimerInformation* Sender) {
Serial.print("Feeding ");
  Serial.print(A_BML);
  Serial.print("ml Bloom A&B");
  for (int doseCount = 1; doseCount <= numofA_B10mlDoses; doseCount++) {
      Serial.print(".");
    digitalWrite(RLY8, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*1.3); //200 = MINml
    digitalWrite(RLY8, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");


  Serial.print("Feeding ");
  Serial.print(VoodooJuiceML);
  Serial.print("ml VoodooJuice");
  for (int doseCount = 1; doseCount <= numofVoodooJuice10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY6, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY6, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!\n");
FeedCount = FeedCount +1;
} //END April BloomFeed2Cal

//May Bloom Feed
void BloomFeed2Call(RTCTimerInformation* Sender) {
  Serial.print("Feeding ");
  Serial.print(A_BML);
  Serial.print("ml Bloom A&B");
  for (int doseCount = 1; doseCount <= numofA_B10mlDoses; doseCount++) {
      Serial.print(".");
    digitalWrite(RLY8, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*1.3); //200 = MINml
    digitalWrite(RLY8, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");


Serial.print("Feeding ");
  Serial.print(VoodooJuiceML);
  Serial.print("ml VoodooJuice");
  for (int doseCount = 1; doseCount <= numofVoodooJuice10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY6, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //NEEDS ALTERING FOR JUNE USING SLUGISH
CARBOLOAD
    digitalWrite(RLY6, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");


Serial.print("Feeding ");
  Serial.print(BigBudML);
  Serial.print("ml BigBud");
  for (int doseCount = 1; doseCount <= numofBigBud10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY5, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY5, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!\n");
   FeedCount = FeedCount +1;
  }//end BloomFeed2Call


//June Bloom Feed
void BloomFeed3Call(RTCTimerInformation* Sender) {
 Serial.println(" DONT FORGET TO SWITCH VOODO JUICE WITH CARBOLOAD");
  Serial.print("Feeding ");
  Serial.print(A_BML);
  Serial.print("ml Bloom A&B");
  for (int doseCount = 1; doseCount <= numofA_B10mlDoses; doseCount++) {
      Serial.print(".");
    digitalWrite(RLY8, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*1.3); //200 = MINml
    digitalWrite(RLY8, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

 Serial.print("Feeding ");
  Serial.print(BigBudML);
  Serial.print("ml BigBud");
  for (int doseCount = 1; doseCount <= numofBigBud10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY5, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY5, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

   Serial.print("Feeding ");
  Serial.print(B_52ML);
  Serial.print("ml B-52");
  for (int doseCount = 1; doseCount <= numofB_5210mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY7, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY7, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

  //NEED FEED CARBOLOAD  should i add ML VARIABLES for carbo load aand flush or use
same as voodujuice YES
Serial.print("Feeding ");
  Serial.print(VoodooJuiceML);
  Serial.print("ml CarboLoad");
  for (int doseCount = 1; doseCount <= numofVoodooJuice10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY6, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*4); //200 = MINml
    digitalWrite(RLY6, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!\n");
FeedCount = FeedCount +1;

}//end BloomFeed3Call


//July Bloom Feed
void BloomFeed4Call(RTCTimerInformation* Sender) {
  Serial.println(" CHECK VOODOO JUICE HAS BEEN SWITCHED WITH CARBOLOAD");
  Serial.print("Feeding ");
  Serial.print(A_BML);
  Serial.print("ml Bloom A&B");
  for (int doseCount = 1; doseCount <= numofA_B10mlDoses; doseCount++) {
      Serial.print(".");
    digitalWrite(RLY8, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*1.3); //200 = MINml
    digitalWrite(RLY8, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

 Serial.print("Feeding ");
  Serial.print(BigBudML);
  Serial.print("ml BigBud");
  for (int doseCount = 1; doseCount <= numofBigBud10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY5, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY5, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

   Serial.print("Feeding ");
  Serial.print(B_52ML);
  Serial.print("ml B-52");
  for (int doseCount = 1; doseCount <= numofB_5210mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY7, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY7, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");


//NEED FEED CARBOLOAD  should i add ML VARIABLES for carbo load aand flush or use
same as voodujuice YES
Serial.print("Feeding ");
  Serial.print(VoodooJuiceML);
  Serial.print("ml CarboLoad");
  for (int doseCount = 1; doseCount <= numofVoodooJuice10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY6, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*4); //200 = MINml
    digitalWrite(RLY6, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!\n");
FeedCount = FeedCount +1;
}//end BloomFeed4Call

//August Bloom Feed
void BloomFeed5Call(RTCTimerInformation* Sender) {
  Serial.println(" CHECK VOODOO JUICE HAS BEEN SWITCHED WITH CARBOLOAD");
  Serial.println(" CHECK OVERDRIVE HAS BEEN FILLED UP FOR 1st EVER USE");
  Serial.print("Feeding ");
  Serial.print(A_BML);
  Serial.print("ml Bloom A&B");
  for (int doseCount = 1; doseCount <= numofA_B10mlDoses; doseCount++) {
     Serial.print(".");
    digitalWrite(RLY8, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*1.3); //200 = MINml
    digitalWrite(RLY8, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

  Serial.print("Feeding ");
  Serial.print(B_52ML);
  Serial.print("ml B-52");
  for (int doseCount = 1; doseCount <= numofB_5210mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY7, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY7, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

Serial.print("Feeding ");
  Serial.print(OverDriveML);
  Serial.print("ml OverDrive");
  for (int doseCount = 1; doseCount <= numofOverDrive10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY4, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay); //200 = MINml
    digitalWrite(RLY4, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!");

  //NEED FEED CARBOLOAD  should i add ML VARIABLES for carbo load aand flush or use
same as voodujuice YES
Serial.print("Feeding ");
  Serial.print(VoodooJuiceML);
  Serial.print("ml CarboLoad");
  for (int doseCount = 1; doseCount <= numofVoodooJuice10mlDoses; doseCount++) {
    Serial.print(".");
    digitalWrite(RLY6, LOW); //BloomA &BloomB symultaniously
    delay(Pump10mlTimeDelay*4); //200 = MINml
    digitalWrite(RLY6, HIGH);
    delay(settleTime);//Let nutrient tubes settle
   }//end for
   Serial.println(" Done!\n");
  FeedCount = FeedCount +1;
```

```
 }//end BloomFeed5Call

 //September Bloom Feed
 void BloomFeed6Call(RTCTimerInformation* Sender) {

  //NEED FEED CARBOLOAD  should i add ML VARIABLES for carbo load aand flush or use
same as voodujuice YES
Serial.println(" DONT FORGET TO SWITCH CARBOLOAD WITH ROYAL FLUSH");
Serial.print("Feeding ");
 Serial.print(VoodooJuiceML);//or TODO FlushML
 Serial.print("ml FLUSH");
 for (int doseCount = 1; doseCount <= numofVoodooJuice10mlDoses; doseCount++) {
   Serial.print(".");
   digitalWrite(RLY6, LOW); //BloomA &BloomB symultaniously
   delay(Pump10mlTimeDelay); //200 = MINml
   digitalWrite(RLY6, HIGH);
   delay(settleTime);//Let nutrient tubes settle
  }//end for
  Serial.println(" Done!\n");
  FeedCount = FeedCount +1;
}//end BloomFeed6Call


void AirCallON(RTCTimerInformation* Sender) {
 AirIsON = 1;
 Serial.println("Air Pump is ON");
}

void AirCallOFF(RTCTimerInformation* Sender) {
 AirIsON = 0;
 Serial.println("Air Pump is OFF");
}

void NFTCallON(RTCTimerInformation* Sender) {
 NFTIsON = 1;
 Serial.println("NFT Pump is ON");
}

void NFTCallOFF(RTCTimerInformation* Sender) {
 NFTIsON = 0;
 Serial.println("NFT Pump is OFF");
}

void FlushCallON(RTCTimerInformation* Sender) {
 FlushIsON = 1;

 Serial.println("Flush Pump is ON");
}

void FlushCallOFF(RTCTimerInformation* Sender) {
 FlushIsON = 0;
 Serial.println("Flush Pump is OFF");
}

void updateTemperature(DeviceAddress deviceAddress)
{
 float tempC = sensors.getTempC(deviceAddress);
 if (tempC == -127.00) {
   Serial.print("Error getting temperature");
 } else {
  // Serial.print("C: ");
  // Serial.print(tempC);
  // Serial.print(" F: ");
   //Serial.print(DallasTemperature::toFahrenheit(tempC));
   myTemp = tempC;

 }
}

void updateTemperature2(DeviceAddress deviceAddress)
{
 float tempC2 = sensors.getTempC(deviceAddress);
 if (tempC2 == -127.00) {
   Serial.print("Error getting temperature");
 } else {
  // Serial.print("C: ");
  // Serial.print(tempC);
  // Serial.print(" F: ");
   //Serial.print(DallasTemperature::toFahrenheit(tempC));
   myTemp2 = tempC2;

 }
}

void printTemperature(DeviceAddress deviceAddress)
{
 float tempC = sensors.getTempC(deviceAddress);
 if (tempC == -127.00) {
   Serial.print("Error getting temperature");
 } else {
   Serial.print("C: ");
   Serial.print(tempC);
   Serial.print(" F: ");
   Serial.print(DallasTemperature::toFahrenheit(tempC));
   myTemp = tempC;
 }
}
void printTemperature2(DeviceAddress deviceAddress)
{
 float tempC2 = sensors.getTempC(deviceAddress);
 if (tempC2 == -127.00) {
   Serial.print("Error getting temperature");
 } else {
   Serial.print("C: ");
   Serial.print(tempC2);
   Serial.print(" F: ");
   Serial.print(DallasTemperature::toFahrenheit(tempC2));
   myTemp2 = tempC2;
 }
}

//----------------RELAY Testing Output------------------
void RELAYTEST()
 {
 digitalWrite(RLY8, LOW); //BloomA &BloomB symultaniously
 delay(500);
 digitalWrite(RLY8, HIGH);

 digitalWrite(RLY7, LOW); //B-52 (Vitamin Pump)
 delay(500);
 digitalWrite(RLY7, HIGH);

 digitalWrite(RLY6, LOW); //Voodo Juice (Healthy Bacteria Pump)
 delay(500);
 digitalWrite(RLY6, HIGH);

 digitalWrite(RLY5, LOW); //Big Bud or Final Phase (Big Yeilds or Flushing Solution)
 delay(500);
 digitalWrite(RLY5, HIGH);

 digitalWrite(RLY4, LOW); //OverDrive (LastBoost)
 delay(500);
 digitalWrite(RLY4, HIGH);

 digitalWrite(RLY3, LOW); //Dual Fans
 delay(500);
 digitalWrite(RLY3, HIGH);

 digitalWrite(RLY2, LOW); // NFT Recirculation Pump
 delay(500);
 digitalWrite(RLY2, HIGH);

 digitalWrite(RLY1, LOW); // Flushing/Waste Pump
 delay(500);
 digitalWrite(RLY1, HIGH);

 digitalWrite(AirPumpRLY, HIGH); // Flushing/Waste Pump
 delay(500);
 digitalWrite(AirPumpRLY, LOW);
}
```