I confirm that this fyp interim report is my own work and I have documented all sources and material used.

, April 14, 2018 Anushka Vashishtha (3035299404)
Acknowledgments

I would like to thank my final year project’s supervisor Dr. C. Wu for her advice. Next, I want to thank Dr. Y. C. Wu my advisor who always steered me in the right direction whenever I needed it. I would also like to thank Dr. Shuai Wong, post-doctoral candidate at the Department of Electrical and Electronic Engineering. The door to his office was always open whenever I ran into trouble or had questions. He specially provided me assistance related to parts where knowledge in electrical engineering was crucial.
Abstract

In the past few decades, there has been an influx in the number of internet of thing devices being used worldwide, and the amount of data which they are producing is estimated to be 100s of trillion gigabytes per year [1]. This tremendous reliance on IoT devices, generates a situation where we have to find efficient ways to communicate with them as well as charge them, specifically in the case of tiny IoT devices like an RFID or Bluetooth. One one hand, using a traditional method like battery is not a viable option for miniscule sized IoT devices. On the other hand, charging cables are not suitable, as it is not only expensive to purchase them in abundance considering each device, but also not practical for inaccessible areas. Henceforth, this project proposes the deployment of an unmanned ground vehicle in designated areas to wirelessly charge [2] and collect data from clusters of tiny IoT devices.

The objective of this report is to explore different methods like Multi Integer Non-Linear Programming, Reinforcement Learning and Deep Reinforcement Learning in order to plan the path of an unmanned ground vehicle so that it can charge the devices, meanwhile optimising both the energy consumed by it and the total path taken. Results from the above methods have been included and compared. All of the above methods are compared extensively on the basis of their efficiency and speed, and ultimately the one which gives the best result in a real world environment is chosen. Additionally, steps are listed throughout this report, on how to adapt the above methods for a practical scenario.

This report demonstrates that if the performance of the chosen method is promising, then such a vehicle can actually be deployed and can help in charging and gathering data in real life, for example from packages kept in a warehouse and marked by an RFID. Moreover, it leads to reduction in charging cable usage which can help the environment.
## Abbreviations

Here are some abbreviations used in this report:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>UGV</td>
<td>Unmanned ground vehicle or robot</td>
</tr>
<tr>
<td>MINLP</td>
<td>Mixed Integer Non-Linear Programming</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>MTC</td>
<td>Machine-Type Communications</td>
</tr>
<tr>
<td>UWB</td>
<td>Ultra-Wide Band</td>
</tr>
</tbody>
</table>
Contents

Acknowledgments ii
Abstract iii
Abbreviations iv
List of Figures 1
List of Tables 2

1. Introduction 3
   1.1. Project Background 3
   1.2. Objective 4
   1.3. Outline 4

2. Previous Work 5

3. Methodology 6
   Charging Region Model 6
   3.1. Phase I 7
      Background 7
      Mathematical Formulations 7
   3.2. Phase II 9
      Background 10
      Procedure 11
      Practical Application 11
   3.3. Phase III 12
      Background 13
      Neural Network Architecture 14

4. Results 16
   4.1. Phase I 16
   4.2. Phase II 18
   4.3. Phase III 20
Contents

4.4. Comparison of Results .............................................. 23
    MINLP vs Q-learning .............................................. 23
    MINLP vs deep Q-learning ....................................... 25
    MINLP vs Q-learning vs deep Q-learning ...................... 27

5. Difficulties Encountered ............................................. 29

6. Conclusion ..................................................................... 30

7. Future Planning .......................................................... 31

Bibliography ..................................................................... 33

A. Appendix ...................................................................... 35
    A.1. Charging Model .................................................. 35
List of Figures

3.1. Charging Region Model ............................................. 6
3.2. Q-Learning ............................................................. 10
3.3. Q Learning model .................................................... 12
3.4. Neural Network Architecture .................................... 14
3.5. Neural Network Model .............................................. 15

4.1. Shortest Path by MINLP ............................................ 16
4.2. No Sub-tour Elimination in MINLP ............................... 17
4.3. Optimal Solution by MINLP ....................................... 18
4.4. High Reward for Goal Q-learning ................................. 19
4.5. Final Solution by Q Learning ...................................... 20
4.6. 8x8 Deep Q-Learning ................................................. 21
4.7. Epochs Required for Deep Q-Learning Training ............... 22
4.8. 40x40 Grid Deep Q-Learning ..................................... 23
4.9. Moving Energy Comparison 8x8 Grid: MINLP and Q-Learning ................................................................. 24
4.10. Moving Energy Comparison 8x8 Grid: MINLP and Q-Learning(all devices charged) ................................................................. 25
4.11. Moving Energy Comparison 8x8 grid: MINLP and Deep Q-Learning ............................................................... 26
4.12. Moving Energy Comparison 40x40 Grid: MINLP and Deep Q-Learning ............................................................... 27
4.13. Moving Energy comparison 8x8 grid: MINLP, Q-Learning and Deep Q-Learning ............................................................... 28

7.1. Continuous Charging Model ....................................... 31
# List of Tables

1. Abbreviation table .................................................... iv
2. Parameter table ....................................................... 8
3. Schedule table ......................................................... 32
1. Introduction

This section embarks with a detailed description on the general project background, followed by a problem statement and objective. In addition, this section also provides with an outline for the remaining sections of the report.

1.1. Project Background

In the last years, internet of things have taken the centre stage in the technology world by creating one of the fastest growing markets and it has been predicted by Forbes that more than 30% of the companies in manufacture, internet of vehicle as well as retail, have already adopted IoT devices in 2015. Notably, IoT devices will soon be worth 1700 billion U.S. dollars, as they are expected to outnumber by 20 billions in 2020 [3]. In addition to that, IoTs are already estimated to be generating 100s of trillion gigabytes of data per year and this figure is only increasing [1]. In the next decade, almost every device will be connected to the internet, ranging from sensors, vehicles, wearable electronics to other embedded systems like refrigerators [1].

To prepare for the future, design engineers are working on finding efficient solutions, specifically in order to power as well as to communicate with billions of tiny IoT devices, since providing sufficient energy to them particularly, is quite a difficult task. Relying on traditional resources like batteries cannot meet the requirement due to the miniscule size of such a device. Additionally, using charging cables can suffice the requirements but the downfall here is that they must be bought in abundance and then, repaired and disposed sustainably. Moreover, maintenance of such a cable becomes a challenge, when IoT devices work in inaccessible areas.

To solve this problem, an unmanned ground vehicle is used to wirelessly charge [4] (and communicate with) a cluster of small scaled IoT devices like RFIDs, Bluetooth and UWB as they not only require short distance MTC (distance ≤ 10m) but also do not need enough power for charging (typically 1µW ~1mW )[3]. As a result, cables will become an obsolete solution for powering the IoT devices and this techniques will also resolve the difficulties involved in purchasing and maintenance. Therefore, rather than buying say 100 different cables for 100 different IoT devices, there will now be just one UGV to power all of the devices and also to collect data from them, if required.
1.2. Objective

From a Computer Science perspective, this problem can be abstracted to be a path planning problem in a graph. The UGV will be capable of interacting with the devices and charging the devices present at different locations in a graph, one by one. To achieve this, the given report will present different approaches such as MINLP, Reinforcement Learning and Deep Reinforcement Learning. After individually receiving the results for the optimum paths from the various approaches mentioned above, the report will depict a systematic comparison of their performances to conclude which has a promising solution theoretically and if it is practically viable to apply any of the approaches in a real world environment. Furthermore, steps are listed throughout this report, on how to adapt the above-mentioned approaches for a practical scenario.

1.3. Outline

The remainder of this report will proceed as follows. First, there is a detailed description about the prior work which is done in the field of robot wireless charging. Next, the report explores different methodologies in detail on how to plan the path of the UGV. Here, the relevance of each method, technical side and step by step procedure involved in execution of each algorithm is explained in depth. Following, the report discusses the results and covers the main difficulties encountered till now. Adding on to this, conclusion for this project is discussed. Lastly, the report is supplemented with a section which provides information on how the project can be expanded in the future.
2. Previous Work

Through this project, reinforcement learning is being implemented for the first time in order to plan the path of a UGV (also referred to as robot) thereupon which, it can charge IoT devices. Nevertheless, there has been extensive research done on various robot wireless charging approaches which are explained below.

A traditional scheme for the robot wireless charging is to deploy multiple static transmitters. However, this is cost expensive and not adaptive to network changes. Additionally, there are two other methods which can help in robot wireless charging.

1. Wirelessly powered two way communication with non-linear energy harvesting model: Rate regions under fixed and mobile relay [5].


Recently, the above two references use mobile robot for charging, but they adopted fixed paths and assumed complete knowledge of user channels [5][6]. This can lead to excessive energy usage. To this end, path planning with or without channel knowledge is needed as in real life situations, it’s not practical to know the complete picture of where each IoT device is located.

Therefore, this project advocates the use of reinforcement learning so that the robot can learn from the environment and can explore where each IoT device is located. At the same time the robot is also charging the devices and aiming to minimise energy consumption. This is quite cost effective as only the robot is required for learning and charging without any additional hardware. Moreover, while planning the path, the robot also has the ability to communicate with IoT devices.
3. Methodology

This section includes description of the model of the environment being used, as well as different phases involved in the development of the project.

Charging Region Model

The abstraction of the robot path planning problem gives the following model:

There is a set of IoT devices \( D = \{d_1, \ldots, d_k\} \) which are positioned on vertices \( V = \{v_1, \ldots, v_M\} \) of an equal-distance-grid-shaped-graph representing the operation area. The UGV is located on one of the vertex \( v_U \) and a goal is present at another
3. Methodology

vertex \( v_G \). This goal provides a location where UGV can get charging and maintenance services but ideally, the UGV should reach the goal only after charging all the IoT devices present in the operation area. Therefore, in each time-step, UGV moves one step along the grid. UGV has to interact (i.e. provide power or communicate) with the devices by entering the charging region [appendix A.1] of each device one by one and ultimately reach the goal. A certain amount of energy is required by UGV in moving and that is the energy which is reduced through path planning.

3.1. Phase I

Background

The first phase to obtain a solution for the path planning problem, involved the use of Multi Integer Non-Linear Programming which uses branch and bound approach for solving problems [7]. It is used to solve convex or non-convex optimisation problems with discrete variables and non-linear functions which can be placed as either an objective function or as a constraint [7].

These properties of MINLP made it particularly useful in deciphering the robot path planning problem because of two reasons. First, the constraint on path planning, which is sub-tour elimination, is of non-linear nature and second, the location of each IoT device on the grid satisfies the requirement for the presence of a discrete variable [7].

A CVX optimisation solver Mosek was used to handle combinatorial difficulty of optimising over discrete variable set together with the issue of handling a non-linear function in order to solve MINLP.

Mathematical Formulations

This table presents the parameters used in the MINLP problem [8].

In this problem, the moving time from the vertex \( m \) to the vertex \( j \) is \( D_{m,j}/a \), hence the total moving time along the path is

\[
\sum_{m=1}^{M} \sum_{j=1}^{M} \frac{W_{m,j} D_{m,j}}{a} = \text{Tr}(D^T W) \tag{3.1}
\]

Furthermore, since the total motion energy \( E_M \) of the UGV is proportional to the total motion time, the motion energy can be expressed in the form of

\[
E_M = (\frac{\alpha_1}{a} + \alpha_2) \text{Tr}(D^T W) \tag{3.2}
\]
3. Methodology

Table 3.1.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v )</td>
<td>Visit a point in the grid or not (Boolean variable)</td>
</tr>
<tr>
<td>( X )</td>
<td>Link between two points in the grid or not (Boolean matrix)</td>
</tr>
<tr>
<td>( \alpha_1, \alpha_2 )</td>
<td>Toning parameter: pioneer’s 3DX robot experiment result at MIT (constant)</td>
</tr>
<tr>
<td>( a )</td>
<td>Velocity of UGV (constant)</td>
</tr>
<tr>
<td>( D )</td>
<td>Distance between two points in the grid or not (Boolean matrix)</td>
</tr>
<tr>
<td>( W )</td>
<td>Summation of ( X ) values (matrix)</td>
</tr>
<tr>
<td>( M )</td>
<td>Length &amp; width of the grid (variable)</td>
</tr>
<tr>
<td>( K )</td>
<td>Total number of IoT devices (variable)</td>
</tr>
</tbody>
</table>

where \( \alpha_1 \) and \( \alpha_2 \) equal 0.29 and 7.4 respectively [8].

Notice that the variables \( v \) and \( W \) are dependent since \( v_m = 0 \) implies \( W_{m,j} = W_{j,m} = 0 \) for any \( j \in V \). On the other hand, the UGV would visit the vertex with \( v_m = 1 \), making \( \sum_{j=1}^{M} W_{m,j} = \sum_{j=1}^{M} W_{j,m} = 1 \). Combining the above two cases, we have

\[
\sum_{j=1}^{M} W_{m,j} = v_m, \sum_{j=1}^{M} W_{j,m} = v_m, \quad \forall m = 2, \ldots, M \tag{3.3}
\]

Furthermore, since the path must be connected, the following subtour elimination constraints are required to eliminate disjointed sub-tours [8].

\[
\lambda_m - \lambda_j + \left( \sum_{l=1}^{M-1} v_l - 1 \right) W_{m,j} + \left( \sum_{l=1}^{M-1} v_l - 3 \right) W_{j,m} \\
\leq \sum_{l=1}^{M-1} v_l - 2 + J \left( 2 - v_m - v_j \right), \quad \forall 2 \leq m, j \leq M - 1, \; m \neq j, \\
v_m \leq \lambda_m \leq \left( \sum_{l=1}^{M-1} v_l - 1 \right) v_m, \quad \forall m \geq 2, \tag{3.4}
\]

where \( \{\lambda_m\} \) are slack variables to guarantee a connected path, and \( \sum_{l=1}^{M} v_l \) is the number of vertices involved in the path. The constant \( J = 106 \) is large enough such that the first line of constraint is always satisfied when \( v_m = 0 \) or \( v_j = 0 \). In this way, the vertices not to be visited would not participate in subtour elimination constraints [8].

Finally, the mathematical description of objective function (represents the energy lost by the UGV due to movement in joule) and constraints involved in MINLP are as follows:
3. Methodology

\[
\min_{\mathbf{v}, \mathbf{X}, \{\lambda_m\}} \left( \frac{\alpha_1}{\alpha} + \alpha_2 \right) \text{Tr}(\mathbf{D}^T\mathbf{W})
\]

s.t. \( v_1 = v_M = 1, \) (select starting and end points)

\[ v_m \in \{0, 1\}, \ \forall 2 \leq m \leq M - 1, \] (selection is binary)

\[ \sum_{m \in \mathcal{C}_i} v_m \geq 1, \ \forall i = 1, \cdots, K, \] (charge all IoT users)

\[ W_{m,j} \in \{0, 1\}, \ \forall m, j, \ W_{m,m} = 0, \ \forall m, \] (flow selection is binary)

\[ \sum_{j=1}^{M} W_{i,j} = 1, \ \sum_{j=1}^{M} W_{j,1} = 0, \] (flow from starting point)

\[ \sum_{j=1}^{M} W_{M,j} = 0, \ \sum_{j=1}^{M} W_{j,M} = 1, \] (flow to end point)

\[ \sum_{j=1}^{M} W_{m,j} = v_m, \ \sum_{j=1}^{M} W_{j,m} = v_m, \ \forall m = 2, \cdots, M, \] (flow passing selected points; no flow passing abandoned points)

\[ \lambda_m - \lambda_j + \left( \sum_{l=1}^{M-1} v_l - 1 \right) W_{m,j} + \left( \sum_{l=1}^{M-1} v_l - 3 \right) W_{j,m} \]

\[ \leq \sum_{l=1}^{M-1} v_l - 2 + J (2 - v_m - v_j), \ \forall 2 \leq m, j \leq M - 1, \ m \neq j, \]

\[ v_m \leq \lambda_m \leq \left( \sum_{l=1}^{M-1} v_l - 1 \right) v_m, \ \forall m \geq 2. \] (guarantee flow connected)

In Phase I when there is complete knowledge about the environment, then MINLP gives the most optimal solution and when there is incomplete knowledge then it helps by giving the lower bound.

3.2. Phase II

Thereafter, the model is put in a Reinforcement Learning setting and Q-learning is applied on it. The goal of Q-learning is to learn from the environment, and to tell what action should be taken under which state. It does not require a model of the environment and can handle problems with stochastic transitions and rewards. In the
3. Methodology

next paragraphs, this report defines Q-learning in detail and talks about the components of Q-learning for this particular problem.

![Figure 3.2. Q-Learning](image)

**Background**

In basic terms, Q-learning is defined to be goal directed learning, which can be achieved through interactions with an environment. An episode of robot path planning can be visualised as a sequence of states, actions and rewards as shown in fig. 3.2. In each state (position on the grid) say $S_t$ at time $t$, a robot takes actions $A_t$ and as a consequence, is presented with reward $R_{t+1}$ and a new state $S_{t+1}$. This reward is the sum of reward received on transitioning from state $S_t$ to another state $S_{t+1}$, plus the future discounted rewards which the robot can get by being in this new state $S_{t+1}$. This reward can be shown mathematically as,

$$R_{t+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+2} + \cdots + \gamma^{n-t-1} r_n$$ (3.5)

where $0 \leq \gamma \leq 1$, is the discount factor and as the rewards very far in the future hold less value as compared with immediate rewards that is why it is less than 1.

The objective of Q-learning is to maximise the expected value of total reward over all the successive steps by finding an optimal policy. This optimal policy has an optimal action-value function (also known as Q-function) which is represented as,

$$q^*(s,a) = \max_\pi q_\pi(s,a)$$ (3.6)

for all $s \in S$ and $a \in A(s)$. In other words, $q_\pi$ gives the largest expected return achievable by any policy $\pi$ for each possible state-action pair. All the state action pairs for this problem are stored in a Q table.

One fundamental property of $q_\pi$ is that it must satisfy the following equation.

---

**Figure 3.2.: Q-Learning**
3. Methodology

\[ q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s',a')] \]  (3.7)

This is called the Bellman optimality equation[9]. It states that, for any state-action pair \((s,a)\) at time \(t\), the expected return from starting in state \(s\), selecting action \(a\) and following the optimal policy (also known as the Q-value of this pair), is going to be the expected reward on taking action \(a\) in state \(s\), which is \(R_{t+1}\), plus the maximum expected discounted return that can be achieved from any possible next state-action pair \((s',a')\). Since, the UGV is following an optimal policy, the following state \(s'\) will be the state from which the best possible next action \(a'\) can be taken at time \(t+1\).

To conclude, the Q-learning algorithm uses value iteration by iteratively updating the Q-values for each state-action pair using the Bellman equation until the Q-function converges to the optimal Q-function.

Procedure

Now, for the path planning scenario, the state space, the actions and the rewards discussed above for Q-learning algorithm, are proposed to be defined in the following way:

State \(S = \{(x,y)|x,y \in [M]\}\) (location of each point on the grid)
Action \(A = \{"move left UGV","move up UGV","move down UGV","move right UGV"\}\)

\[ R(x_i,y_i) = \begin{cases} 
10 + (5 \times x) & \text{if } (x_i,y_i) = v_G, x = \text{ no. of IoT devices charged} \\
10 & \text{if } (x_i,y_i) \text{ in charging region of a particular IoT for the first time} \\
-1 & \text{otherwise.} 
\end{cases} \]

The policy which we want to optimise in the path planning scenario for both Q-learning and deep Q-learning is called as epsilon-greedy policy. This policy dictates the UGV to choose an action by either exploring that is by choosing a random action or by exploiting that is by choosing an action associated for a particular state having the maximum Q-value for that state, which can be obtained by looking up at the Q-table in case of Q-learning.

Practical Application

Now before elaborating on Phase III, this report examines on the Dyna-Q algorithm [10] which is extremely crucial for practical situations where nothing about the model...
3. Methodology

Figure 3.3.: Q Learning model

of an environment is known. Since, in a real world scene, it is more expensive to interact with the environment, it is preferred if the UGV can train in a simulated environment. That is when Dyna-Q comes into play as Dyna-Q algorithm integrates both direct reinforcement learning and model learning. Therefore, planning is one-step tabular Q-planning, and learning is one-step tabular Q-learning. Real or simulated experience improves both the model via model learning and value function via direct RL. Model contains state, action, next state and reward tuples. Thus, the model can be both improved and queried to get to the next state in planning part. In conclusion, Dyna-Q can really prove useful when data available is insufficient to provide good models.

3.3. Phase III

Q-learning algorithm is capable of planning the robot’s path for a small operation area but unfortunately, when the grid size of the operation area increases, Q-learning becomes inefficient as it takes a long time to update the Q-table. This occurs due to the presence of a Q-table with huge dimensions and therefore, the computer might not have enough memory to increment values of such a table.

Therefore, in this project Deep Reinforcement Learning is used to approach this
3. Methodology

problem with the same action and rewards values as Q-learning. Deep Reinforcement Learning uses neural networks to estimate the Q-values for each state-action pair (as described in the previous section) in a given operation area, and in turn, the neural network approximates the optimal Q-function. The act of combining Q-learning with a deep neural network is called deep Q-learning, and a deep neural network that approximates a Q-function is called a deep Q-Network, or DQN. In the next paragraphs, this report defines and talks about the components of deep Q-learning for robot path planning in depth.

Background

First step in executing deep Q Learning for path planning, involved the use of experience replay. With experience replay, the UGV’s experiences are stored at each time step in a data set \( D \) called the replay memory having capacity \( N \) [11]. The UGV’s experience at time \( t \) is represented as

\[
et = (s_t, a_t, r_{t+1}, s_{t+1}) \tag{3.8}
\]

This tuple contains the state of the environment \( s_t \), the action \( a_t \) taken from state \( s_t \), the reward \( r_{t+1} \) given to the UGV at time \( t+1 \), as a result of the previous state-action pair \( (s_t, a_t) \), and the next state of the environment \( s_{t+1} \). All the experiences are obtained with the help of an emulator and due to sampling, are not correlated which enhances learning process.

After obtaining the replay memory dataset, sampling is done and served as an input to the first network called policy network and in path planning case, it is represented by \( \text{build}_\text{net} \). The output in the form of Q-values called \( q(s, a) \) is obtained via forward propagation from \( \text{build}_\text{net} \) for every action \( a \). A loss is then calculated by comparing \( q(s, a) \) for the action in the experienced tuple and the corresponding optimal Q-value \( q_*(s, a) \), for the same action.

\[
q_*(s, a) - q(s, a) = \text{loss} \tag{3.9}
\]

However, to calculate the optimal Q-value \( q_*(s, a) \), a second pass is done to a new neural network called \( \text{target}_\text{net} \) with the next state \( s_{t+1} \) or \( s' \). From this second pass, the maximum Q-value among the possible actions from \( s' \) is obtained, and put that into the Bellman equation to calculate the optimal Q-value \( q_*(s, a) \) for the action \( a_t \) or \( a \) from the first pass.

\[
q_*(s, a) = E[R_{t+1} + \gamma \max_{a'} q_*(s', a')] \tag{3.10}
\]
3. Methodology

After the loss is calculated, the weights within the build_net are updated via Rectified linear unit activation function and backpropagation. The deep Q-Network is set to use a learning rate of 0.01. This learning rate is an inherent hyper-parameter of the network and is fine tuned to yield good training results. RMSProp is used as the optimization function and offers improved stability and convergence over regular gradient descent. Moreover, in case of target_net, weights are updated after every certain amount of time steps. This certain amount of time steps is again a hyperparameter which is fine tuned. This feature of deep Q-Learning is called fixed Q-targets and is very essential as it prevents \( q(s,a) \) from moving close to \( q^*(s,a) \) after each iteration.

This process is done over and over again for each state in the environment until we sufficiently minimize the loss and get an approximate optimal Q-function.

**Neural Network Architecture**

Next, the report describes the architecture of deep Q-Networks (see fig. 3.4, fig. 3.5) as well as how the loss is minimised. To start off, neural networks are defined to be a computing system and they consist of a collection of connected nodes. They are used
to work together and process complex data as inputs and therefore are able to take in a model with a very large state space. This project uses two neural networks. One is called *build_net* and the other is called *target_net*. Both, are identical to each other in terms of architecture.

The input to the neural network is the state set of the environment representing if the state has been finished or not. Then, this input is passed to the two hidden layers of the neural network. In this project, the hidden layers have 10 neurons each. They have weights \( w_1 \) and \( w_2 \) having random value between 0 and 0.3 whereas bias \( b \) is 0.1. The first hidden layer takes the input described above and gives the corresponding output to the second hidden layer. The second hidden layer takes that and gives out Q values corresponding to all the actions possible by the UGV for a given state which is the final output.

Subsequently, the activation of the hidden layers’ used is rectified linear unit. Rectified linear unit is a non-linear function \( f(z) = \max(0, z) \) and due to faster training, it is quite popular in deep neural networks. It is suggested that increasing the amount of hidden nodes leads to increased representational capacity of the deep Q-Network. However, increasing the amount of hidden nodes also increases the time it takes to train one session of the operation area[11]. That is why, the number of nodes are kept at a minimum.

![Figure 3.5.: Neural Network Model](image)
4. Results

This section presents the results from different methods, and comparisons between the methods.

4.1. Phase I

In this section, all the results and inferences obtained after the application of MINLP are discussed. Implementation is carried out in MATLAB using CVX solver Mosek.

In fig. 4.1 we do not consider the charging case (section 3.1) which means that UGV will not consider charging IoT devices as a constraint while planning its path.

![Figure 4.1.: Shortest Path by MINLP](image)

Next fig. 4.2 shows the path taken by the UGV where sub-tour eliminations (sec-
tion 3.1) is not involved as one of the constraints[12]. Therefore, there will be various small tours in the grid apart from the path between UGV’s initial position and goal(terminal), which are highly undesirable.

Finally, we have the optimal solution after considering all the constraints from section 3.1. Here, the energy value obtained is 241 Joule by taking the path shown in fig. 4.3.
Finally, the energy value obtained by MINLP for a larger grid size of 80x80 is 345J. The graph is not presented due to its large size.

4.2. Phase II

In this section, all the results and inferences obtained after the application of Q-Learning are discussed. Implementation is carried out in python using numpy. The user interface for the current and the next phase has been made using Tkinter.

The fig. 4.4 shows that when a high reward is given to the UGV (say 100 + 5*each device charged) when it reaches the goal, then the UGV has a very low tendency to charge all IoT devices and therefore we will not get the desired path.
Finally after implementing all the reward values from section 3.2, we get the result by Q Learning in fig. 4.5. The lowest energy value consumed is 252 Joule after training for 100 epochs.
4. Results

4.3. Phase III

In this section, all the results and inferences obtained after the application of Deep Q-Learning are discussed. Implementation is carried out in python using numpy and tensorflow. The environment used is same as that in the case of Q-learning.

Below in the fig. 4.6, is the result obtained after applying the deep Q-learning algorithm on an 8x8 dimension maze. The lowest energy consumed is 246 Joule after training for 600 epochs.
4. Results

Figure 4.6.: 8x8 Deep Q-Learning

It was observed that deep Q-learning requires more number of episodes for efficient training as compared to Q-learning which was able to find an efficient path with just 100 epochs (fig. 4.5). Nevertheless, the maximum number of epochs required are around 800 (fig. 4.7) as after that the moving energy of UGV does not change irrespective of the increment in the epochs. Moreover after tuning the hyper-parameters for deep Q-Learning, the $\epsilon - greedy$ values used is 0.6 as compared to 0.9 for Q-learning. Additionally, the $\alpha$ and $\gamma$ values are 0.01 and 0.9 which is contrary to 0.5 and 0.5 values obtained from Q-learning respectively.
4. Results

Figure 4.7.: Epochs Required for Deep Q-Learning Training

The next major step was to increase the state space to an operation area having grid of the dimension 40x40 (fig. 4.8 where the blue function represents deep Q-Learning and red function represents MINLP and serves as lower bound). The lowest energy consumed is 356 Joule after training for 300 epochs.

The grid dimensions was not increased further, as then the environment would not have been able to fit in the tkinter canvas software which is used to build the operation area, and therefore, analysis of UGV’s movement would not be efficient.
4. Results

4.4. Comparison of Results

MINLP vs Q-learning

This section compares results obtained from MINLP and Q-Learning.

In fig. 4.9 a graph is plotted between energy consumption by UGV in moving and number of epochs. Here as the UGV gets trained for more epochs, the energy consumed by it in moving decreases drastically. The Blue line shows the energy consumption for Q-Learning and the red line serves as a lower bound and shows the energy consumption for MINLP (energy is calculated in section 3.1). This comparison is not fair as it is not certain that every path obtained by Q-Learning after a certain number of epochs, involves all the IoT devices to be charged.
Therefore, we only take those energy values from Q-Learning where path involves charging all the IoT devices. The final result obtained is shown in fig. 4.10.

Figure 4.9.: Moving Energy Comparison 8x8 Grid: MINLP and Q-Learning
4. Results

Figure 4.10.: Moving Energy Comparison 8x8 Grid: MINLP and Q-Learning (all devices charged)

**MINLP vs deep Q-learning**

This section compares results obtained from MINLP and deep Q-Learning.

In fig. 4.11 a graph is plotted between energy consumption by UGV in moving in the standard grid size of 8x8 and the number of epochs. Here as the UGV gets trained for more epochs, the energy consumed by it first increases and reaches a maximum value at epoch number 260. Thereafter, it decreases drastically. This is because of a high $\epsilon$ — greedy value and therefore, UGV explores extensive initially. The Blue line shows the energy consumption for deep Q-Learning and the red line serves as a lower bound and shows the energy consumption for MINLP (energy is calculated in section 3.1). This comparison is fair as every path obtained by deep Q-Learning and included in this graph, involves all the IoT devices to be charged.
4. Results

Figure 4.11.: Moving Energy Comparison 8x8 grid: MINLP and Deep Q-Learning

Next we have fig. 4.12 which compares energy consumption by UGV in moving in the standard grid size of 40x40. Here as the UGV gets trained for more epochs, the energy consumed by it decreases drastically as it is able to comprehend the large grid efficiently. This is because due to the large size but the same number of IoT regions, UGV is not presented with a lot of paths which yield similar rewards. Therefore, the UGV learns the local/global optimum path with the maximum number of rewards, faster and ultimately, converges.
4. Results

MINLP vs Q-learning vs deep Q-learning

This section compares results obtained from MINLP, Q-Learning and deep Q-learning. Finally, fig. 4.13 a graph is plotted between energy consumption by UGV in moving and the number of epochs. Here as the UGV gets trained for more epochs, the energy consumed by it in moving decreases drastically. The blue line shows the energy consumption for Q-Learning, red line shows the energy consumption for deep Q-Learning and the green line serves as a lower bound and shows the energy consumption for MINLP (energy is calculated in section 3.1).

The inferences drawn from the figure are as follows:

1. The moving energy of UGV decreases significantly for Q-learning whereas for deep Q-learning it increases by a huge amount and then decreases. This is because the deep neural network takes much longer to train and to find the optimum path in a smaller grid.

2. On observing later epochs, it is estimated that after sufficient training the moving energy required by UGV in case of deep Q-learning is lower than the moving
4. Results

energy required by UGV in case of Q-learning. Reasons that can justify this behaviour are unnecessary training of Q-learning algorithm which leads to unnecessary exploration. The neural networks involved in case of deep Q-learning are much more efficient and optimise to a better path as due to experienced replay there is no correlation in the present and the next state, unlike in the case of Q-learning.

Figure 4.13.: Moving Energy comparison 8x8 grid: MINLP, Q-Learning and Deep Q-Learning

Q-learning was observed not to be able to function in an operation area with 80x80 grid because the UGV was getting stuck at particular locations again and again as well as it was taking a very long time for Q-learning to converge. Therefore, only results from deep Q-learning and MINLP have been included (section 4.4).
5. Difficulties Encountered

The main difficulties encountered in the development of this project are explained as follows:

1. Formulating the model for the project initially appeared as a difficult task, since there was no prior work available in this area. Therefore, different action sets and reward functions were considered before choosing the most suitable one for Phase I modelling.

2. Understanding the electrical engineering concepts involved in robot wireless charging was a challenging work as it required quite high level knowledge in that field.

3. Limited experience in TensorFlow and MATLAB was one of the biggest hurdle in project development as all the implementation is carried out with these software.

4. Most importantly all the main methods like Reinforcement Learning, Deep Reinforcement Learning and Multi-armed Bandit Optimisation are quite new for me, therefore considerable amount of time was spent to get a basic foundation in order to understand the mechanism of each of them.

5. Problems related to the installation of CVX software due to licensing issues, delayed the progress of the project significantly.
6. Conclusion

The main aim of this project is comparison of different approaches so as to choose the most efficient path planning method for the deployment of a UGV. In order to achieve this, the report presented results obtained from MINLP, Q-Learning and deep Q-learning respectively. Comparison of outcome obtained from above mentioned methods, showed that Q-learning should be used for robot path planning in case the operation area is small and when there is not enough time for training the robot. Moreover, dyna-Q should be incorporated with Q-learning in cases when no information about the model is present in order to save the cost as dyna-Q uses simulation for training.

In all the other cases, deep Q-learning should be used because it supports large grid size and is more efficient in achieving an optimum path with low energy consumption. All the above comparisons have been made by keeping result obtained from MINLP as a lower bound.

Finally, after the comparison of different methods, the results from deep Q-learning are optimum for real world application as they are quite close to the lower bound provided by MINLP. Therefore, the deployment of such a UGV for wirelessly charging (and communication with) IoT devices is feasible. Moreover, the algorithm presented in this report can also be used to create a simulated environment so that the UGV can train for large epochs without additional costs of actually interacting with the environment. After sufficient training, the UGV will have required knowledge beforehand.

This will not only make cables obsolete but also will play a big role in data collection and charging in various sectors ranging from manufacturing to retail.
7. Future Planning

This section describes the remaining milestones as well as some of the future research work which can be done in this project if time permits.

1. Tackling the situation when variable power is given by UGV to an IoT device according to how far away it is from an IoT device

2. Limited energy present in UGV

3. Multi armed bandit application in Continuous charging model (fig. 7.1)

4. Federated Deep Reinforcement Learning with multiple agents

---

**Continuous Charging model**

- **Low probability for nearby arms**
- **If chosen get random observation (How many users active?)**
  - Change in loc of IoT
  - Disturbances

**Figure 7.1.: Continuous Charging Model**
7. Future Planning

The table 7.1 illustrates already achieved milestones and a few future work extensions.

<table>
<thead>
<tr>
<th>Dates</th>
<th>Milestones</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>September 30</td>
<td>Deliverable 1</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>1. Project Plan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Project Website</td>
<td></td>
</tr>
<tr>
<td>October</td>
<td>Working into MATLAB, Python and TensorFlow. Reading up on MINLP, Reinforcement Learning and Deep Reinforcement Learning.</td>
<td>Completed</td>
</tr>
<tr>
<td>Nov - Dec</td>
<td>Development of demo application. Creating simulated environment for MINLP application and applying it.Applying Reinforcement Learning and comparing them</td>
<td>Completed</td>
</tr>
<tr>
<td>Jan</td>
<td>Deliverable 2</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>1. Demo Application</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Interim Report</td>
<td></td>
</tr>
<tr>
<td>Dec - Feb</td>
<td>Applying Deep Reinforcement Learning with function approximation method.</td>
<td>Completed</td>
</tr>
<tr>
<td>Mar - Apr</td>
<td>Comparing the results obtained from the different approaches. If time permits considering variable power given by UGV, limited energy present in UGV as further extensions to the problem. Then working into Continuous charging model (Multi armed Bandit Optimization)</td>
<td>Completed &amp; future work</td>
</tr>
<tr>
<td>April</td>
<td>Deliverable 3</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td>1. Finalized Implementation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Finalized Report</td>
<td></td>
</tr>
</tbody>
</table>
Bibliography


A. Appendix

A.1. Charging Model

If the transmit power at UGV is $P$, then the harvested power at IoT user $k$ is $\Upsilon \left( |g_k|^2 \cdot P \right)$, where $g_k$ is the wireless channel from UGV to user $k$, and $\Upsilon$ is the function representing the energy conversion process and is given by

$$\Upsilon(P_{in}) = \left[ \frac{P_{\text{max}}}{\exp(-\tau P_0 + \nu)} \left( \frac{1 + \exp(-\tau P_0 + \nu)}{1 + \exp(-\tau P_{in} + \nu)} - 1 \right) \right]^+, \quad (A.1)$$

where the parameter $P_0$ denotes the harvester’s sensitivity threshold and $P_{\text{max}}$ refers to the maximum harvested power when the energy harvesting circuit is saturated. The parameters $\tau$ and $\nu$ are used to capture the nonlinear dynamics of energy harvesting circuits. For the Powercast energy harvester P2110, we have $\tau = 274$, $\nu = 0.29$, $P_{\text{max}} = 0.004927$ W and $P_0 = 0.000064$ W.