Optimized Heuristic Based Path Planning within a Complex Environment

Mark S. Robson [1]

University of New South Wales at the Australian Defence Force Academy

The aim of the project has been to develop a path mapping algorithm suitable for future implementation into the Seekur Jr, a versatile robotics platform produced by Adept Mobile Robots. The path mapping algorithm is designed with the Seekur Jr's intended operating environment and use case scenarios considered, namely, a hostile environment populated with enemy assets and varying terrain. To produce this path mapping algorithm the suitability and performance of several existing path mapping schemes were compared and in doing so, another more suitable solution was designed. The developed path mapping algorithm was enhanced with the addition of variable travel costs and an enemy line-of-sight algorithm based on an enemy asset's effective range. These capabilities allow the planned path to give movement preference to areas that are outside the enemy's line of sight/effective range and where environmental effects are minimized to allow for the quickest and safest real-time traversal. To test these capabilities, environmental changes were simulated and applied to the map representation provided to the path mapping algorithm. This paper contains important path mapping theory, development considerations, results of the initial path mapping comparison, testing procedures and an outline of potential future development of the path mapping capability for the Seekur Jr.

Contents

[1] PLTOFF, School of Engineering & Information Technology. ZEIT4500

Nomenclature

BFS	=	A best-first search
GDF	=	A greedy-depth-first search algorithm
GDF Split	=	A modified version of the greedy-depth-first search algorithm
A*	=	A modern path mapping algorithm
f(n)	=	cost function value at node *n*
c(n)	=	travel cost value at node *n*
h(n)	=	heuristic estimate at node *n*
LoS	=	Line of Sight
UGV	=	Unmanned Ground Vehicle
AOO	=	Area of operation

I. Introduction

The development of automated systems and the improvement in their capability and complexity has been progressing for centuries. As early as the 3rd century [1] simple mechanical systems were developed to use the force of fast flowing rivers and streams to grind wheat and to chop wood (mills), in the late 1700's invention of the steam engine revolutionized transportation [2] and in the 1900's the introduction of automated factory equipment such as driven assembly lines/conveyor networks revolutionized mass production [3]. While the process of automation in these applications was largely focused on increasing productivity, reducing operational requirements and reducing production/transport time automation can also be an effective way of removing human risk from a system. For example, this can be seen in the mars rover where an automated system has been used to perform a task (geographical/geological surveying) that would be too dangerous for a human to perform. Another example is in the use of bomb disposal drones that distance their human operators from any risks that they may be exposed to as part of the bomb disposal process.

While these examples represent significant steps forward, they don't represent fully automated systems. These systems still require direction from an operator. For example the human bomb drone operator must navigate the drone to the bomb then operate the drone's tools to disarm the bomb. A more complete form of automation would be for the drone to be alerted to the location of the bomb by the operator, be able to navigate itself to the bombs location and disarm the bomb without further input from the operator. It is this level of automation that is the top level motivation behind this project.

The motivation behind this project is to contribute to the design and implementation of a robot system/robot system software that will be able to direct ammunition via unmanned ground vehicle (UGV) to resupply an operator out on the field autonomously. Given its starting and target end destinations the UGV will plan its own path to the target operator based on prior knowledge of its operating area, prioritizing the use of cover from enemy exposure and speed of delivery.

The contribution of this project within this field is in developing the path planning component of the movement control system with consideration for environmental/terrain conditions and the utilization of cover based on enemy asset(s) line of sight (LoS). This component of automation requires several levels of assumption. Firstly, for the UGV to utilize a movement planning algorithm(s), it first must know its area of operation (AOO) and be able to accurately and regularly be able to assess its position within this AOO. Secondly, the UGV must have the ability to move around the AOO. Both of these assumptions have been fulfilled by the Seekur Jr robot, the mobile robot platform to be used in this project.

This paper outlines the operation of each of the compared algorithms, including the proposed optimal solution, GDF split, in Section II. It then discusses important implementation issues and proposed solutions to these issues in Section III. Section IV contains a comparison between algorithms and how the results draw the conclusion that, while the proposed optimal algorithm is not the optimal solution for the application described in the motivation of this project, the proposed optimal path planning algorithm does offer optimal solve time for path planning in most of the four tested cases. Section V proposes a novel way of representing a complex environment that takes consideration for enemy assets lines of sight and environmental/terrain considerations.

II. Path Planning

In the world of automation and machine vision the term path planning refers to the ability of a control system to plan a path, usually the shortest path, between a set of points within its operating space. This path mapping can be undertaken with (informed) or without (uninformed) prior knowledge of the operating environment. The path mapping algorithms discussed in this paper and the algorithms developed as part of this project all operate on the pretence of an informed environment. This decision is based on intended future integration of this path mapping solution with the Seekur Jr robot platform. This platform already possesses the tools required to

produce an informed operating environment in the form of a ground plane [4] which is easily converted into an occupancy grid.

For the development of the path mapping algorithms compared in this project, the informed environment was initially represented in the form of an occupancy grid. An occupancy grid is an *xsize*ysize* array (where *xsize* is the width of the map array and *ysize* is the height of the map array) comprised of cells, called nodes, containing either a one or a zero. A node value of one indicates an non-traversable obstacle node and a node value of zero indicates a traversable node [5]. In a spatial sense, all compared algorithms assume that nodes are positioned in the centre of their relevant grid cells and that there are eight degrees of movement from each node within the map, discounting nodes on the edges of the map where the degree of movement is restricted to ensure mapped paths stay within the defined map. The eight degrees of movement permissible from each node are shown in Fig. 1.

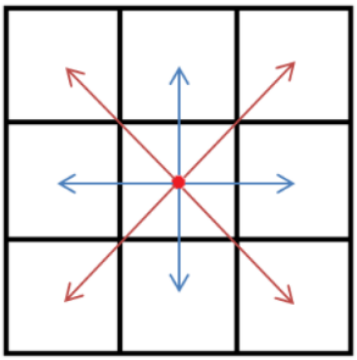

Figure. 1 – Movement. This figure shows the different movements available from a node. The blue arrows indicate movement allowed in a four movement scheme and the red arrows indicate the additional movements allowed in an eight

A. Dijkstra

Dijkstra's algorithm is a path planning algorithm that was developed in 1959 [6] as a solution to single-source shortest path problems on weighted, directed graphs where all travel costs between nodes are non-negative [7]. It does so by initially conducting a breadth-first search of every node within a weighted graph (or in the case of this project, within the occupancy grid) and recording each nodes parent node until the destination node is queried. Once the algorithm has run, the shortest path from any node to the source can be calculated by following the parent nodes back to the source node. While this method produces the shortest path from the source node to any queried node within an equal cost map, it is not the most efficient in terms of computational time. This is due to the initial breadth-first expansion of the querying node which results in the maximum number of queried nodes at any time being $4*N^2$, where N is the number of movements taken to reach the destination. This means that for increasingly large occupancy grids with increasingly large path lengths, the maximum number of queried nodes increases on a quadratic scale.

B. A* Standard

The A* algorithm is a complete shortest path algorithm originally proposed by Hart, Nilsson and Rafeal in 1968 [8] and is considered as the primary basis of modern node based path mapping algorithms. The A* algorithm uses a customized cost function to determine which node is queried and then which nodes are added to the path. The cost function for each of the nodes, f(n), is comprised of the addition of the heuristic cost and the travel cost. The travel cost is the cost associated with moving between the start node and the current node based on previously acknowledged map obstacles and restrictions on movement. The heuristic cost is based on a selected estimation scheme that provides an estimation of the distance to the destination from the current node, ignoring obstacles and environmental factors.

The A* algorithm operates by first setting the source node as the querying node. The querying node then queries all nodes around it within its eight degrees of movement, adds them to a queried list and the node with the smallest f(n) value is recorded. If the smallest f(n) value is equal or smaller than the f(n) value of the current querying node then it becomes the new querying node. If the smallest surrounding f(n) value is larger than the f(n) value of the current querying node then the node from the queried list with the smallest f(n) value that has not been the querying node becomes the querying node. This means that based on the heuristic, the querying node will move towards the destination node until it moves 'away' from the destination node. At this point, it will move to a previously queried node that the algorithm believes will offer the most direct route toward the destination node.

The A* algorithm was implemented in this project in a way such this it received the same initialised map representation as the Dijkstra algorithm. This was to ensure the same testing conditions were maintained between the algorithms. The data structure used within the A* algorithm was more comprehensive as it required multiple additional variables to account for the f(n) and h(n) costs and the queried and querying check flags. This resulted in an M*8 array where M was the number of nodes within the map array.

C. Best-first Search (BFS)

The BFS algorithm is a combination of the Dijkstra algorithm and a heuristic estimate. The BFS implemented as part of this project operates in a similar way to the A* algorithm except that it checks for the node within its queried list with the smallest f(n) value at every change in querying node rather than only when an increase in the f(n) value was registered [9]. Like in the Dijkstra algorithm, all surrounding nodes of the querying node are added to the queried list. The next node to become the querying node is the node within the queried list with the smallest f(n) value that hasn't yet become the querying node [10]. This continues until the destination node is queried.

Like the A* Standard algorithm the BFS algorithm produced as part of this project was implemented in such a way as to receive the same initialised map representation as the Dijkstra algorithm to ensure the same testing conditions were maintained between the algorithms. The data structure used within the BFS algorithm was more comprehensive than the data structure required for the Dijkstra algorithm as each node needed to store additional variables to account for the nodes f(n) and h(n) values.

D. Greedy-Depth First Search (GDF)

The GDF algorithm implemented as part of this project operates in the same way as the conventional GDF algorithm [11], except it also includes additional functionality to prevent the development of no-paths between nodes caused by the self-induced dead-end flaw through. The conventional GDF algorithm operates in the same way as the A* algorithm does before the smallest f(n) value from a querying node is less than the querying nodes fn. This means that the next querying node will always become the node within one permissible move from the current querying node with the smallest f(n) value until the destination node is queried. Because of the node eligibility condition that a node may only be the querying node once, certain environmental conditions can result in a phenomenon called a self-induced dead end. This is where a querying node may register no valid moves within its allowed range of movement resulting in the algorithm incorrectly declaring that there is no-path between the source node and the desired destination node. This phenomenon and the solution proposed in this project are further outlined in Section III part C.

Like the A* and BFS algorithms, the GDF split algorithm produced as part of this project was implemented in such a way as to receive the same initialised map representation as the Dijkstra algorithm to ensure the same testing conditions were maintained between the algorithms. The data structure used within the GDF algorithm was more comprehensive than the data structure required for the Dijkstra algorithm as each node needed to store additional variables to account for the nodes f(n) and h(n) values as well as the master parent variable. The purpose of this master parent variable and its implications for the improved performance of the standard GDF algorithm are further explained in Section III part C.

E. Greedy-Depth First Split Search (GDF Split)

GDF Split is a modification of the GDF algorithm and the proposed optimal solution to the optimal planning problem in this project. The GDF split algorithm deviates from the standard GDF algorithm as it generates an additional querying node when two (or more) nodes queried from the querying node share the lowest f(n) values. This occurs when the querying node comes into contact with an obstacle that is perpendicular to the vector formed between the current node and the destination node but also, as covered in Section III part C, under some specific environmental conditions. In both of these cases, additional querying nodes were placed into the nodes of equal f(n) value forming separate search 'tendrils'. Each tendril then functions in the same way as the initial tendril and may split itself into further tendrils as required.

This modification was introduced as a solution to solve the shortest path problem faster through a map by allowing one or more of the 'tendrils' to continue searching for the destination node even if one or more of the other tendrils are forced into self-induced dead-ends. The issue of self-induced dead ends, how they occur and the solutions that were implemented to solve them are covered more extensively in Section III part C of this paper.

The GDF split algorithm is similar to its predecessor, the GDF algorithm, in that it was implemented in such a way as to receive the same initialised map representation as the Dijkstra algorithm. This ensured the same testing conditions were maintained between tested algorithms. The data structure used within the GDF split algorithm was the same as the structure required for the GDF algorithm.

III. Algorithm Implementation

The following section outlines various design decisions made during the implementation of each of the aforementioned algorithms. Many of these design decisions were required as the search algorithms implemented

and compared in this project are optimized for searches through spanning trees rather than representations of real world scenarios in the form of occupancy grids and weighted graphs.

A. Heuristic

The estimation of the heuristic is a very important aspect of the overall generated cost function [12]. If the heuristic estimate scheme underestimates the actual distance of a node from the destination, then the algorithm may degrade to Dijkstra's algorithm. However, if the selected heuristic estimation scheme overestimates the distance of a node from the destination, then the algorithm will degrade to a standard GDF algorithm (without self-induced dead-end resolution). This is because the overestimated heuristic means that an emphasis is put on the heuristic and the significance of the cost of moving to a destination gets overlooked.

In this project he heuristic cost estimate is calculated, during map initialization, by one of two basic schemes; Euclidian and Manhattan Distance. The Euclidian method of estimation calculates the direct distance to the destination from the current node using basic trigonometry [13]. This method of estimation if useful for maps populated with objects oriented randomly compared to the grid lines. The Manhattan method of estimation is the total x-axis and y-axis movement required to move to the destination from the current node [14]. This method is well suited for environments such as cities where the edges of obstacles like buildings are usually square shaped and oriented parallel to grid lines. The Manhattan heuristic estimation is also preferred over the Euclidian estimation as the required multiplication and square root operations used in the Euclidian estimations are more computationally expensive than the simple subtraction and addition operations required in the Manhattan estimation scheme. This difference in performance was tested by calculating the heuristic estimates for each node for an empty array of various resolutions using each method. The results of these tests are shown in Table 1 below.

	50x50 nodes	500x500 nodes	5000x5000 nodes
Euclidian Estimate	0.000064 seconds	0.0064 seconds	0.6448 seconds
Manhattan estimate	0.000031 seconds	0.0028 seconds	0.2776 seconds

Table. 1 – Heuristic Scheme Calculation Times. The average time taken to calculate the heuristic estimate for every node within 3 arrays of various resolutions for each estimation scheme.

As observed from the results displayed in Table 1, the Manhattan estimate times, in all tested cases, were less than half of the Euclidian estimates. This difference in performance becomes more noticeable for higher resolution images. While neither scheme produced results that indicated that their produced estimates were overestimated or underestimated, the decision was made to test the performance of each algorithm with a Manhattan heuristic estimator to ensure initialisation time of the map structure was minimised.

B. Parent Optimization

All of the implemented algorithms are graph search algorithms that search for the destination node starting from the start node. As each of these algorithms search through the array of nodes it has a querying node which queries the nodes around it, searching, based off a series of algorithm specific requirements, for the optimal node to be the next querying node. As a node is queried its parent node is updated with the unique identifier of the node that it was queried from. Once the destination node becomes the querying node the search algorithm ends and the parent node variable is followed back from the destination to the start node and the resulting node list is reversed, producing a path from start node to end node. However, due to the operation of the Dijkstra, BFS, GDF and GDF Split algorithms, it was found that while the path produced by following the parent node variable back from the destination node produced a valid path through the map to the source node, the path was not optimal. This was caused by the fact that the initial specified parent of any particular node within the map may not be the optimal parent, where a nodes optimal parent node is defined as the immediate preceding node in the shortest path from the source to the node (given the movement restrictions). These non-optimal parents result in a non-optimal path between the source and the destination and this can be observed from the results of the initial test of the Dijkstra implementation. In the initial test case the uniform travel cost was set to be one travel unit for a single dimension movement (straight movement) and sqrt(2) travel units for two dimensional movement (diagonal movement) with eight degrees of movement. To test the algorithm's operation several paths were generated through a map with no obstacles. The results of these paths are shown in Fig. 2.

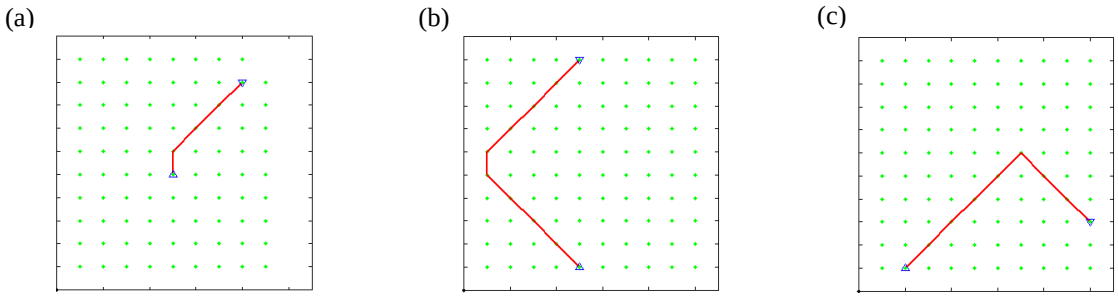

Figure. 2 – Pre-Parent Optimization Planned Paths. This figure depicts several paths generated from the Dijkstra algorithm, without parent optimization, from a source node (upward pointing triangle) to a destination node (downward pointing triangle) through an unpopulated 10x10 occupancy grid. The planned path is shown in red. Queried nodes are shown in green.

While a path between the start and destination has been generated in each of the above cases, it is obvious from the plotted paths depicted in Fig. 2 (b) and (c) that the algorithm in its current form did not produce the shortest path. This is because, as outlined above, the initial produced parent nodes of each node are not always a nodes optimal parents. To solve this issue, the eligibility of a nodes current parent as the optimal parent was tested at each query. The eligibility of the current parent was tested by comparing the travel cost from the source node to the node through the current querying node with the travel cost from the node to the source through the existing parent node. If the former was found to be shorter, the querying node becomes the queried nodes new parent, and if not, the existing parent node was maintained. This alteration of the algorithm produced the paths depicted in Fig. 3 for the same start and destination nodes as in Fig. 2.

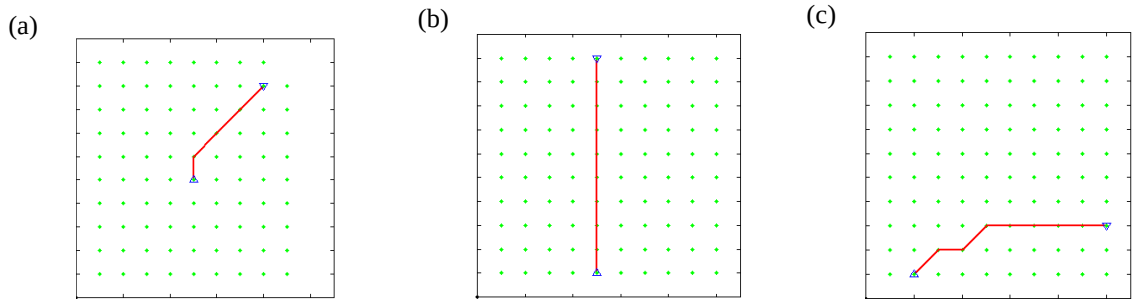

Figure. 3 – Post Parent Optimization Planned Paths. This figure depicts several paths generated by the Dijkstra algorithm, with parent optimization, from a source node (upward pointing triangle) to a destination node (downward pointing triangle) through an unpopulated 10x10 occupancy grid. The planned path is shown in red. Queried nodes are shown in green.

C. Self-induced Dead Ends

As previously mentioned, self-induced dead ends were an issue experienced during the implementation of the GDF algorithm. To explain the issue the basic operation of the implementation needs to be understood. The standard GDF algorithm registers that there was no valid path between a source and destination node if it reached a point in its operation where there were no available traversable nodes from the current querying node. This situation can occur in two cases; when the destination node is in fact unreachable (all traversable nodes have been searched) or when the querying node is entirely surrounded by non-traversable nodes, where a non-traversable node in defined as either an obstacle (occupied node) or a node that has previously been a querying node. This second situation meant that the algorithm could in some cases 'trap' the querying node from finding a path to the destination node by moving it through an obstacle bottleneck that it couldn't find its way out of.

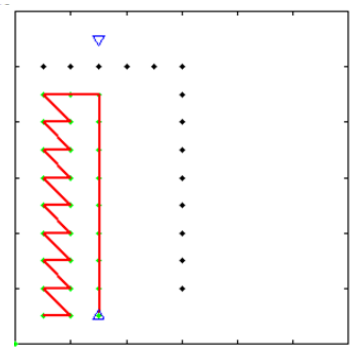

Figure. 4 – Self-Induced Dead End Flaw. This figure illustrates the self-induced dead end flaw within the GDF algorithm. Notice how in the planned path does not reach the destination node.

To solve this issue a solution was proposed that set the next querying node to be the current querying node's parent until the querying node was no longer surrounded by non-traversable nodes. When tested on the original A* algorithm, this provided a suitable solution to the self-induced dead end problem; however, when implemented into the A* algorithm with optimal parent modifications, the algorithm would occasionally fail to find a valid path to the destination node when there was a valid path. On analysis of the situations that produced this error the problem was found to be that by updating the parents of the queried nodes, in certain situations, every updated parent node along the updated parent path would be in a self-induced dead end again resulting in an incorrect no-path declaration. To solve this problem an additional variable was added for each node called the 'master parent node'. This master parent node variable stores the unique identifier the querying node that first queried the current node. When the algorithm registered no valid moves from the current querying node, instead of backtracking to its parent node it would backtrack to its master parent node preventing self-induced no-paths. Fig. 5 demonstrates an instance where following the parent node path (red) would result in a no-path declaration, but where following the master parent path (purple) would allow for the correct path to be produced.

The trace back through the master parent variable continues until a traversable node is queried or until the start node becomes the querying node and no valid nodes are queried. This indicates that there is no solution for a path between the selected source and destination nodes and all traversable nodes within the map have been checked.

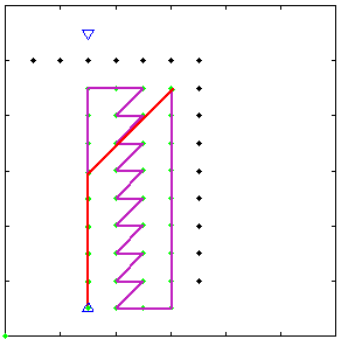

Figure. 5 – Master Parent and Updated Parent Paths. This figure depicts an instance where the combination of parent optimization and backtracking does not provide a solution to path planning problem. The current parent optimized path from the trapped node is depicted in red and the non-parent optimized path is depicted in purple. While tracing the optimized parent path will result in another self-induced dead end, tracing the non-parent optimized path back will allow for the production of a complete path.

IV. Algorithm Comparison

To determine which algorithm offered the best solution to the shortest path problem for the desired application, the performance of each algorithm was compared to the benchmark (Dijikstra) in terms of three main criteria; average time, average path length and average percentage of nodes queried. These criteria were chosen for various reasons; average time serves an indicator of overall performance, path length is used to test the effectiveness of the algorithm at fulfilling its purpose, and average percentage of nodes queried has implications for future development of the algorithms.

Data used to calculate the above criteria was collected over 500 randomly selected traversable paths through four different maps. The random number generator used to generate the random traversable paths used the same seed for each map to ensure the same random paths were selected.

A. Map Selection

The four tested maps were selected because they each offer an entirely different environment that the algorithms way be required to plan a path through. By selecting maps that offer extreme examples of possible operating scenarios the best and worst case scenarios can be more effectively realised. The four maps used are shown in Fig. 6.

The first map Fig.6 (a), 'squares', is comprised of an offset grid of square obstacles oriented with the grid. This map is similar to the basic layout of buildings in a suburban area and offers multiple incidents of obstacle edges that are parallel to the desired direction of travel without any dead-ends. The second map Fig. 6 (b), 'pacman' is comprised of a number of cut circular objects. Objects of this shape produce more dead ends and effectively simulate the shape of many

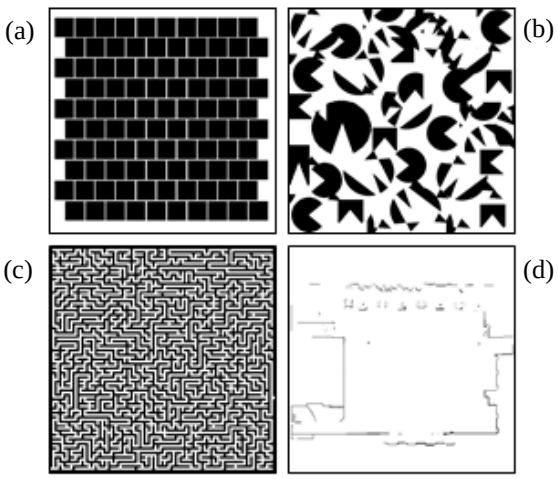

Figure. 6 – Tested Maps. This figure depicts the 4 maps used to compare the performance of each of the developed algorithms; (a) 'squares', (b) 'pacman', (c) 'maze' and (d) 'campbell'. Each maps resolution is 500x500 pixels where each pixel was recognized as a node.

'no-travel zones' produced by enemy LoS capability discussed in Section V. The third map Fig.6 (c), 'maze', is a complex maze where every traversable node can be mapped to any other traversable node *i.e.* there are no dead areas. This map was selected as it offers an environment where the benefits of a heuristic estimate are minimised, thus producing close to worst case performance from the heuristic based algorithms. The fourth map Fig. (d), 'campbell', is a ground plane of Campbell Primary School, Campbell, ACT, Australia that was generated using the Seekur Jr's laser rangefinder. This map was selected as it depicts a realistic potential operating environment that the map mapping algorithms can be applied too.

B. Results

The results of the comparison testing between the algorithms are shown in Table 2. In Table 2, for each map, the average value of each algorithm for the relevant criteria was divided by the smallest achieved average. This was done to more easily highlight the optimal algorithm for each of the comparison criteria.

	Average time					Average Path Length					Average Perc. nodes Queried				
	Dij	A*	BFS	GDF	GDF Split	Dij	A*	BFS	GDF	GDF Split	Dij	A*	BFS	GDF	GDF Split
squares	20.46	10.75	33.93	**1**	3.99	**1**	1.0061	1.0039	1.0705	1.0411	54.21	5.82	3.41	**1**	20.10
pacman	4.37	99.23	83.54	2.58	**1**	**1**	1.0078	1.0198	1.1481	1.1349	5.80	2.33	1.22	**1**	4.51
campbell	30.32	82.83	104.69	2.93	**1**	**1**	1.0045	1.0010	1.0985	1.0542	26.93	2.69	1.41	**1**	5.38
maze	**1**	215.81	132.71	3.47	17.79	**1**	1.0012	**1**	1.0336	1.0344	1.19	2.21	1.10	**1**	2.44

Table. 2 – Testing Results. This table shows the results of the testing conducted as outlined above. Values contained within this table are the corresponding average for each algorithm, for each variable, for each map, divided by the smallest average across all algorithms for that variable within that map. A value of 1 indicates that the corresponding algorithm, for the corresponding map recorded the lowest average for that variable.

C. Discussion of Results

Based on knowledge of the operation of each of the algorithms outlined in Section 2 and observation of the defining features of each of the maps, the results depicted in Table 2 can be effectively explained and understood. The 'squares' map encourages the splitting behaviour of the GDF split algorithm due to the number of instances within the map that result in multiple adjacent nodes of equal f(n) value. However, with the absence of dead ends, which would normally filter out unneeded tendrils produced as a result of this splitting behaviour, this has resulted in the suboptimal performance of this algorithm on this map in terms of average calculation time. Conversely, the GDF algorithm performed best in this map in terms of average calculation. This was due to the lack of dead-ends within the 'squares' map which also accounts for the minimisation of the number of queried nodes.

The splitting behaviour of the proposed GDF split algorithm allowed it to achieve the fastest average path calculation time on the 'pacman' map. This is because the splitting allowed for the GDF split algorithm to locate the goal more quickly than the GDF algorithm as it was able to move toward the goal even if one of the tendrils was forced into a dead-end. This did come at the cost of, on average, an additional ~13% path length.

The performance of all four of the heuristic based algorithms (A*, BFS, GDF and GDF split) was reduced for the tests conducted through the 'maze' map. This is to be expected as in this map, neither of the heuristic schemes covered in this project would provide an accurate estimate of the distance from a node to the destination and would, in most cases, provide an underestimate. This underestimate results in Dijikstra's algorithm offering the optimal performance.

The average calculation time of the GDF split time was optimised for all of the tested maps, other than its identified worst case scenario, 'squares' and the heuristic worst case scenario 'maze'. While this indicates that it reaches a solution to the path planning problem posed in each of the random test cases first, the average path length compared to the other compared algorithms is non-optimal. The difference between the performances of the optimal path length, generated by the Dijikstra algorithm, and the GDF split algorithms average path was ~13% higher for the worst case scenario, this trade-off between calculation time and path length has significant implications for the intended application of this path planning algorithm. For example, the 'campbell' map depicts an area within the Campbell Primary School that is approximately 100 m*100 m. This can be translated to give the distance between each node, each 'travel unit', as ~20 cm. Raw data from the tests conducted shows that the shortest average path length in travel units, where one travel unit is defined as the distance between two horizontally or vertically adjacent nodes, was calculated by the Dijkstra algorithm at 289.75 travel units. The average path length of the GDF split algorithm on this map was measured to be 305.44 travel units. This means

a difference of ~15.7 travel units or, given the maps scale, ~3.14 m. While the calculation time of the Dijkstra algorithm was ~3000% higher than the calculation time for the GDF split algorithm this only translates to an increase in calculation time of ~0.45s. Given that the intended UGV platform for this path planning algorithm is the Seekur Jr, which is capable of maximum speeds of ~1.2 m/s [4], the trade-off between calculation time and increased path length will not result in the optimal time taken for calculation time plus real-time path traversal time in this case.

While the results gathered through this testing process can provide a brief insight into the performance of the operation of each of the compared algorithms on the tested maps, the sampled random routes do not offer a complete representation of each algorithm's performance for every route through the tested maps and thus the significance on the quantitative results cannot be asserted. To obtain a set of results that represent the complete overall performance of each of the algorithms with a suitable confidence, a suitable sampling scheme would need to be developed for each of the tested maps. This sampling scheme would need to define a suitable random sample size or a suitable informed sampling method for the (n-1)! possible routes through a map containing n traversable nodes. The results collected above and shown in Table 2, do however allow for qualitative assessment of the performance of each of the compared algorithms.

V. Environmental Considerations

As previously discussed, the motivation behind this project is to contribute to the development of a movement planning system that can be used within a UGV operating within a hostile environment. It can be assumed that a hostile environment will contain enemy assets that may threaten the safety of the UGV. To minimize the risks to the UGV the planned path must take into consideration these enemy assets. Similarly the planned path must also take into consideration the environmental/terrain effects that may affect real-time planned path traversal costs.

A. Basic Enemy Consideration

Initially hostile areas were marked by designating nodes within each enemy assets effective range and line of sight as obstacle nodes on the occupancy grid. Each enemy assets line of sight was calculated by tracing a number of vectors or 'rays' away from the enemy assets position in a 360 degree arc. Rays were traced outward from the node where the enemy asset was located using Bresenham's line trace function [15] until the length of the traced vector was the size of the enemy assets declared effective range or until the ray collided with a permanent obstacle. Bresenham's line tracing function works by incrementing the current inspected nodes one position along the primary node axis (e.g. x-axis) and calculating the corresponding change in the secondary axis (e.g. y-axis) at each increment [15]. Once the accumulated change in the secondary axis exceeds 0.5, the currently inspected node increments its position along the secondary axis by one node and the secondary axis change accumulator variable is reset. Every node that was checked during the trace was marked as a temporary obstacle. The number of rays was based on the circumference of the circle whose radius was the effective range of the enemy asset. This was to ensure that all nodes within an enemy assets line of sight and within its effective range were marked as temporary obstacles. While based on the circumference of this circle, the number of rays was over-estimated by multiplying the circumference by a constant to ensure full coverage and to prevent the effects of 'cartwheeling', where nodes within the effective range and line of sight were missed due to an insufficiently detailed vector gradient or a lack of rays (see Fig. 7). There was an identified tradeoff between minimization of cartwheeling and the time taken to trace all of the rays; however, the minimization of cartwheeling was prioritized due to the implications it had on the quality of the produced path i.e. if cartwheeling was present within an enemy assets line of sight the planned path would not effectively avoid it which could result in an unsafe path.

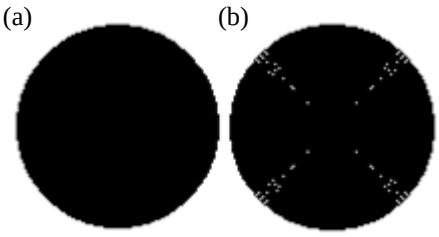

Figure. 7 – 'Cartwheeling' Effects. This figure depicts the effects of 'cartwheeling' caused by an inadequate number of rays traced from an enemy assets position. (a) shows the desired environmental effect profile of an enemy asset with an unobstructed LoS within its effective range where (b) shows an environmental effect profile for the same enemy asset with the effects of cartwheeling. Notice in (a) how there are several nodes within the profile that have not been checked (white pixels). This can result in incorrect paths being planned within an enemy assets effective range and LoS.

B. Complex Enemy Consideration

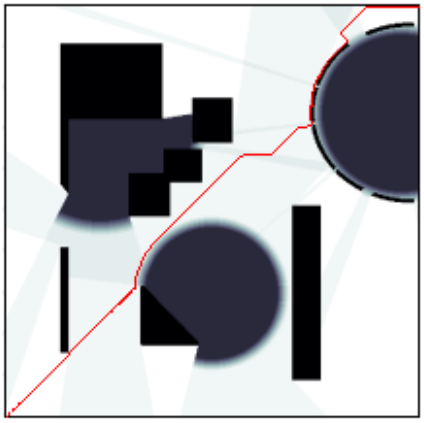

Figure. 8 – Map Representation with complex Enemy Considerations. This figure depicts a map with the complex enemy consideration capability added. Obstacles within the map are shown in black while areas with a darker color indicate nodes with a higher environmental multiplier indicating non-ideal terrain caused by enemy LoS. Notice how the planned path seeks to minimize travel cost with consideration for the environmental multipliers.

While the basic method of enemy avoidance, outlined above, does effectively prevent the path from being planned through areas within the effective range of an enemy asset, it does not result in the production of a path that maximizes the safety of the UGV. To minimize the risk to the UGV caused by enemy assets, the planned path needs to consider line of sight past an enemy assets effective range.

This has been achieved in this project by altering the map representation used by the path mapping algorithm. Rather than representing the operating environment in the form of an occupancy grid as before, the operating environment was represented in the form of a weighted graph or an array of 'environmental multipliers'. In this way, paths planned through areas with lower multipliers were produced as these paths resulted in the smallest projected real time traversal cost. Obstacles were given the highest multiplier, with a tolerance, employed to ensure nodes with a multiplier higher than this tolerance were not considered during the path planning progress. This idea of a travel multiplier is a simplified version of the multiple node variables assigned by Jönsson in his application of the principle of a weighted graph to represent environmental effects in a 3D environment [16]. An enemy assets effect in a nodes multiplier was determined during the initialization stage. If a node was within the enemy assets line of sight and effective range, *r*, its multiplier was set above the allowable movement tolerance, *ot*, to ensure that a path was not planned through these areas. If a node was within the line of sight of an enemy asset but outside its effective range, its multiplier, *m*, was based of the value of the exponential decay function shown in (1) where *a* is defined as the Euclidian distance of the node from the enemy asset less *r* (enemy asset's effective range).

$$m = (ot) * \left(\frac{1}{ot}\right)^{\frac{a}{r}} \quad (1)$$

The effects of multiple enemy assets were additive until the maximum environmental multiplier value was reached. An example of a complex environment populated with enemy assets of various effective ranges generated using the methods and formulas is shown in Fig. 8.

C. Environmental Considerations

There may be areas within the proposed UGV's operating environment that are hazardous to the operation of the UGV or may not provide optimal movement conditions. If a path is planned through these areas the resulting real-time travel cost and the safety of the UGV as it traverses the planned path may not be optimized. For example muddy, sandy areas or areas with dense vegetation may take longer to traverse than a bitumen road. To allow for the path planning algorithm to account for these non-optimal nodes, the effects of these environmental factors has been incorporated into the variable travel multiplier applied as part of the complex enemy consideration. As can be observed in Fig. 9, the variable travel modifier serves as an effective way of encouraging path planning through more desirable environmental conditions where proximity to enemy assets is minimized. This differs from the scheme proposed by Jönsson [16] as it combines terrain and enemy multipliers to reduce the dimensions of the required weighted graph data structure.

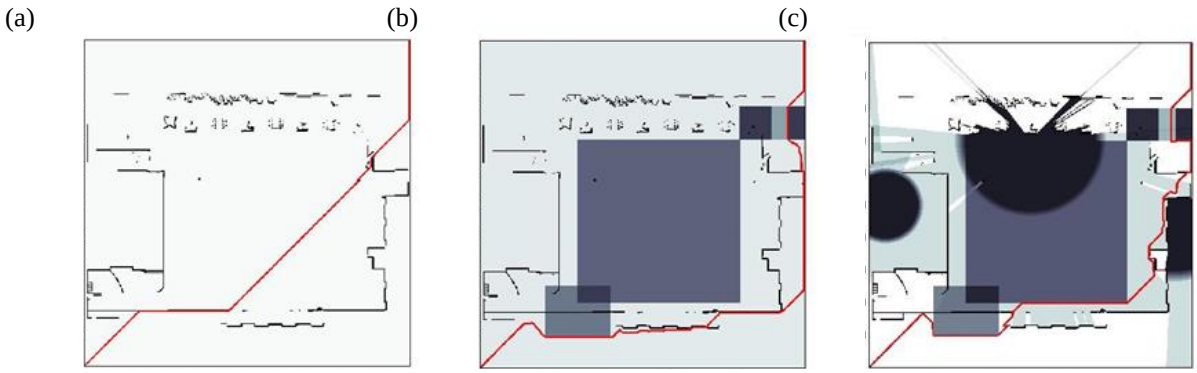

Figure. 9 – Complete Complex Map Representations w/ Environmental and Complex Enemy LoS Considerations. This figure shows three different scenarios; (a) no environmental or enemy effects, (b) environmental effects only and (c) both environmental and complex enemy LoS effects. Each path (shown in red) is planned from a source node at (1,1) (bottom left corner) to a destination node at (500,500) (top right). Notice how the paths change to compensate for the varying environmental factors.

VI. Future Work

As outlined in Section IV part C, for the quantitative comparison of the compared algorithms performance to be confidently assessed and for the significance of any quantitative conclusions to be considered an appropriate informed sampling method needs to be developed. This sampling method will need to produce a sample of routes that can be used to accurately model the performance of each algorithm within a 95%+ confidence level.

While the comparison of algorithm performance offered in this project was conducted on maps represented in the form of occupancy grids, a similar comparison needs to be completed for the same algorithms for the map representation scheme proposed in Section V that takes into consideration environmental and enemy LoS factors.

VII. Conclusions

This project has compared the performance of several search algorithms adapted for application to path planning across four selected maps and has proposed a more optimized solution. It was identified that to best decide which algorithm is most suitable for any given application the trade-off between computation time, and the real-time taken to traverse the longer produced path needs to be considered. By considering this trade-off, the intended operating environment and the results collected as part of this project, it can be concluded that, while the GDF split algorithm does, in most cases, provide optimized calculation time, it is not the optimal algorithm for implementation into a UGV inspired by this projects motivation. This is due to the decreased path calculation time not efficiently compensating for the additional projected time taken to travel a longer planned path. While the trade-off is this application is non-optimal, if the algorithm was implemented into a system with a higher real-time movement speed or where the real-time path traversal time was less significant than the path calculation time then the proposed algorithm will be the most optimal solution to the path planning problem.

This project has also offered a solution for efficiently adding additional enemy asset and environmental information to a map representation so that it can be effectively utilized by the compared path mapping algorithms.

Acknowledgements

Firstly I'd like to thank my supervisors, Valeri Ougrinovski and Charles Harb, for their guidance throughout the progression of this project. I'd also like to acknowledge Randy Orton for the motivation he has provided me throughout the year, 'from outa nowhere'.

References

[1] Oleson, John Peter. Greek and Roman Mechanical Water-Lifting Devices: The History of a Technology, University of Toronto Press, 1984.

[2] Hills, Richard L. Power from Steam: A history of the stationary steam engine, Cambridge, Cambridge University Press, 1989.

[3] Michigan Yesterday & Today. Voyageur Press. 2009-10-01.

[4] Adept MobileRobots. Seekur Jr SKR0100, SKR0200 Manual, Revision 4, 25 March 2014.

[5] Koefoed-Hansen, Andreas. "Representations for Path Finding in Planar Environments", Aarhus University, February 29, 2012.

[6] Dijkstra, E.W. "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik 1, pg 269-271, 1959

[7] Cormen, Thomas H. "Introduction to Algorithms" (1st ed.), MIT Press and McGraw-Hill, 1990

[8] Hart, P.E., Nilsson, N.J., Raphael, B. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", Systems Science and Cybernetics, IEEE Volume 4 Issue 2, July 1968

[9] Vempaty, N.R., Kumar, V., Korf, R.E. "Depth-First vs Best-First Search", Dept. of Computer Sciences, University of Florida, 1991

[10] Korf, R.E."Linear-space best-first search", Computer Science Department, University of California, Los Angeles, USA, 2003

[11] Tarjan, R. "Depth-First Search and Linear Graph Algorithms", Society for Industrial and Applied Mathematics Comput Vol.1 Issue. 2, 1972

[12] Thayer, Jordan T., Ruml, Wheeler., Kreis, Jeff. "Using Distance Estimates in Heuristic Search: A Re-evaluation", Department of Computer Science, University og New Hampshire, Durham, NH USA, 2009

[13] Teitz, M.B., Bart, P. "Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph", University of California, Berkeley, California, USA, 1968

[14] McDermott, D. "A Heuristic Estimator for Means-Ends Analysis in Planning", Yale Computer Science Department, New Haven, USA, 1996

[15] Wright, W.E. "Parallelization of Bresenham's line and circle algorithms", Computer Graphics and Applications, IEEE, Volume 10, Issue 5, 2002

[16] Jönsson, F. Markus. "An optimal pathfinder for vehicles in real-world digital terrain maps", Royal Institute of Science, Stockholm, Sweden, Department of Numerical Analysis and Computing Science, 1997.