Dynamic Programming Method for Energy-Aware Path Planning

Randall Schur, Conor Lyman, and Adam M. Wickenheiser

***Abstract*—This paper presents Energy Aware Path Planning (EAPP), a path planning algorithm for mobile robots deployed in extended missions. Battery constraints are often the limiting factor for the range and length of deployment in ground vehicles. In order to extend the lifetime of a mobile robot, solar panels or other energy transducers can be attached to the robot to harvest energy from the environment. The environment may not be well defined; this approach assumes the availability of satellite images which can be segmented into grids, but requires no additional information about obstacles. Our approach defines a novel description of the environment consisting of two costs associated with each grid cell: an energy-based cost of traverse and a collision-based probability of traverse calculated from the model of the robot and the environment. The proposed algorithm takes a dynamic programming approach to finding the least-energy path through the environment without violating a constraint on probability of traverse. Simulations and experiments show more efficient paths for robots in target environments.**

I. Introduction

NAVIGATION algorithms for mobile robots in realistic scenarios present multi-objective optimization problems. The goal is to minimize some cost function while maximizing the chances of success. In a real-world problem, this means synthesizing incomplete and potentially incorrect information about the environment, while minimizing the cost of executing the final path. The cost function may have competing goals; decreasing probability of collision (defined in this study as any obstacle that can damage the robot or stop it from driving), optimizing distance traveled or information gained, minimizing energy usage, or minimizing the time of traverse to a desired region are all potential objectives of the algorithm. As these are often competing interests, the cost function must weight each objective according to the application, along with the uncertainty associated with each piece of information.

These competing costs are often difficult to sum into a scalar cost with a single function. The approach presented in this paper, called Energy-Aware Path Planning (EAPP), presents a novel context for characterizing the environment by introducing multiple metrics to describe cost and uncertainty. EAPP, considers the case of extended deployment in a partially unknown environment. The scenario targeted by this approach is a robot that is deployed on a continuous, extended, and autonomous basis. Using robots for applications such as inspection of infrastructure (especially pipelines, bridges, or roads), environmental monitoring and data collection in remote areas, and surveillance[?] all require this type of long-term autonomy. First and foremost for these applications the robot must continue running and must approach a goal location, and any other goal is secondary. This means that two objectives must be considered: minimizing collisions and minimizing energy usage. Energy minimization here is important for extended duration missions, as on-board battery storage is often a limiting factor for mobile robot deployment. More energy-dense solutions such as fuel cells or radioactive power sources are often impractical due to safety and cost concerns. This approach assumes a battery powered vehicle with the addition of energy-harvesting equipment such as a solar panel. EAPP segments the map into grid cells and evaluates the competing objectives using both a cost function based on estimated energy usage and a penalty function based on estimated probability of a successful traverse. This approach calculates the most efficient path through a cell which does not violate the defined threshold on the likelihood of collision.

Rather than attempting to define a cell by its coverage of obstacles or by the uncertainty in the position of obstacles as in cell decomposition methods,[?] probability of traverse ($P_{tr}(x)$) is defined as the estimate of the likelihood that the robot will be able to traverse through the map segment. The presence of any obstacle that impedes the robot's motion means that there is some probability the robot will fail to traverse the cell. Probability of traverse is a pre-calculated metric related to chance constraints,[?] although it is not an exact parallel. An obstacle density ($\mu(x)$) is also estimated for each map segment, which is used in the calculation of the cost function. Separating these metrics allows the explicit calculation of both the cost function (which is additive) and probability of traverse (which is multiplicative) for a path.

II. Related Work

In this paper we focus on go-to-goal behavior for navigation, in particular moving a wheeled robot through an environment from its present location to some target. The many approaches to this problem are typically categorized as either probabilistic or deterministic.[?]

Probabilistic navigation algorithms, such as Rapidly Exploring Random Trees (RRT),[?] Probabilistic Roadmaps (PRM),[?] and their extensions depend on randomly chosen points in the environment. They deal well with high dimensional configuration spaces, do not require an explicit map of the environment, and are probabilistically complete.[?] Both RRT and PRM are easily extendible, meaning it is often possible to adapt the navigation method to a particular type of situation by adjusting the algorithm's steps. RRT* and PRM* are asymptotically optimal,[?] meaning they converge to the optimal solution given an infinite number of iterations. Other extensions,[??] decrease running time, to the point that probabilistic navigation can be used in real time for mobile robots. Authors in [?] show that a predetermined choice of points in the environment such that the points minimize some metric can in some cases outperform the typically used randomly generated points.

Deterministic Methods such as A*[?] and D*[?] are also successfully implemented as motion planning algorithms for mobile robots. One advantage of these is that as deterministic methods, they will give the same answer for the same input, which may be a desirable property for an autonomous system. These methods are resolution complete, meaning that they are guaranteed to find a solution if the grid resolution is fine enough. The method presented in this paper shares this property. Many related algorithms, including EAPP, give the optimal path down to the grid resolution.

An important consideration for an algorithm interacting in the physical world is how it deals with uncertainty.[?] Uncertainty in mobile robots can originate from motion uncertainty, sensing uncertainty, or environmental uncertainty.[?] Each of these can be addressed in the motion planning phase, and extensive work exists addressing one or more types of uncertainty. Among many examples, work by Bry and Roy[?] considers motion uncertainty, or uncertainty in the state estimate, during path planning. Sensing uncertainty, generated by the uncertainty of measurements, is addressed by work such as[?] or.[?] Environmental uncertainty is defined as the uncertainty in the placement of obstacles and is addressed by[?], [?? in the context of probabilistic navigation algorithms. This work addresses environmental uncertainty using a novel metric, and then uses a deterministic path planning algorithm. One further source of uncertainty arises from a dynamic environment, which is not addressed in this paper. For methods dealing with a dynamic environment, see for example[?] or.[?]

Recent work examines navigation algorithms which consider energy usage of the robot. For mobile robots,[?], [? addresses the deployment of multiple robots which collectively accomplish some exploration task in the most energy efficient manner.[?] Work in[?] presents a navigation strategy which estimates when a robot must return to some charging base. Authors in[?] use optimal motion planning to minimize energy consumption. Each of these papers consider energy usage of mobile robots in the path planning stage, but are not targeted towards extending the duration of a deployment using energy-harvesting, or are focused on exploration rather than go-to-goal behavior. Work in[?] examines recharging capabilities of a robot, but completely ignores the issue of uncertainty and uses a grid-based approach that characterizes each cell as a binary obstacle or not obstacle. The following work aims to address a realistic scenario while incorporating energy usage and recharging into the planning stage.

III. Problem Definition

The problem statement can be defined as finding the path which minimizes the cost function according to the constraint on probability of traverse.

$$\min_{x \in W} \; J(x) = \sum_{i=1}^{k} H_i \tag{1}$$

$$subject\ to\ P_{tr}(x) < \epsilon_c \tag{2}$$

where W is the workspace, n is the number of grid cells in a path, J_i is the cost function evaluated at each grid cell, x is a complete and valid path consisting of waypoints which ends at the target node, $P_{tr}(x)$ is the probability of traverse over the entire path, and ϵ_c is the threshold on probability of traverse. Workspace is defined as set of all possible paths which end at the target node. A path in the workspace is composed of waypoints with position information, as opposed to Configuration space, $\mathcal{C}$, which describes the full state of the robot.[?] Probability of traverse for a path is calculated as the product of that value for each segment in the path:

$$P_{tr}(x) = \prod_{i=1}^{n} P_{tr}(x_i) \tag{3}$$

In any real scenario, each of these quantities is an estimate, as robot environments are unpredictable and models are imprecise.[?] Thus the constraint in equation 1 means that the *estimate* of $P_{tr}(x)$ must not violate the user-defined threshold, ϵ_c. Both the method of estimation and the level of threshold depend on the scenario involved and more importantly on the mobile robot used. In this paper a simple robotic platform demonstrates the navigation algorithm, but any parameters used will vary with a different platform. Specifically, the cost function must take into account how much energy the robot uses, how much energy it can pull from the environment, and the ability of the robot to traverse different types of terrain in order to accurately evaluate the constraint.

Parameter estimates for the robot and the environment are generated before the algorithm runs. In order to increase the usability of the algorithm, the input is limited to information which is realistically available in the targeted scenario. The only required inputs are the robot parameters and estimates of $P_{tr}(x)$, $\mu(x)$, and cost of driving on a surface. All of this information can be generated strictly from a satellite image and information about the robot. The focus of this paper is on navigation, whereas the identification and classification of the environment from the satellite images is out of scope.

A. Global vs. Local Algorithm

This work focuses on the global navigation algorithm, and assumes the availability of a local navigation strategy which takes inputs from onboard sensors and avoids obstacles. It must attempt drive the robot to each successive waypoint, and it must use sensors to avoid obstacles. Extensive work[?] has been done on this topic. The implementation of this local algorithm is interchangeable as long as it meets these requirements. Some options include potential field methods, an optimal tracking controller such as an LQR for a system with a linear model, or a real time extension of a separate navigation algorithm.[?]

EAPP is a global navigation algorithm, meaning it generates waypoints through an environment from a current position to a target position in the workspace. Velocity is not considered by the global algorithm, as this is something best handled by the local algorithm. For estimation of energy usage, this work assumes that the robot travels at an efficient constant velocity. See [?], [?] for a discussion of how velocity affects energy efficiency in mobile robots.

B. Energy Harvesting

The simplest way to extend the range of a vehicle is to carry additional energy storage on board. In any mobile robot, the storage capacity is limited by the size of the platform used. Often this size is limited by external factors such as transportation of the platform or size of passages in the environment, or is limited so that that the robot will be unobtrusive or undetected. The next obvious choice to extend range of a mobile robot is refueling, either by swapping batteries or using a refueling station that can recharge a battery or dispense additional fuel. This approach makes sense when navigating within a predefined network such as a system of roads, but may be infeasible in an unstructured environment. Our approach is to collect energy from the environment using renewable or man-made sources of energy. Transducers for vibration energy (generated by machinery or movement of the vehicle),[?] solar energy, and thermal energy exist and are commercially available. The choice and sizing of these methods of energy collection should be entirely dependent on the availability of energy sources within the environment that the robot will be navigating. While the power density of these sources is low,[?] the availability of these resources in natural environments still makes them an attractive choice for small mobile robots.

The effectiveness of these methods for energy collection is expected to improve as the technology continues to develop. The most widespread energy harvesters are solar panels, which when deployed over a larger area are used to generate a significant amount of power.[?] There are some commercial products that take advantage of other energy sources. These products are mainly focused on personal electronics, gathering energy from sources such as the kinetic energy of the user (vibration energy) using inductors or harvesting RF energy in the form of excess wireless signals. EAPP does not specify what sources of energy to use, but the experimental platform in this work uses a solar panel attached to the robot.

IV. Navigation Algorithm

This section describes the EAPP algorithm and the information necessary to implement it. The cost function presented here is the same as is used in the theoretical and experimental results sections.

A. Algorithm Description (EAPP)

The algorithm is a dynamic programming strategy which takes as input a graph describing the connections between map cells and outputs the optimal path from each grid cell to some target node. Each vertex in the graph represents the averaged cost over one grid cell, and the graph is formed in an eight-connected pattern. In practice, the constraint on probability of traverse is implemented as a penalty function. This allows the algorithm to save a path even if the constraint is violated; the only case in which a path like this is returned is when all possible paths from one node to the goal violate this constraint. The penalty function allows the algorithm to return this option, and it is up to the user (or some pre-determination by the user) to reject this path if desired. Separate from this constraint, the cost of a path provides an estimate of energy usage. If this estimate is larger than the state of charge (SOC) on the battery, the robot should not move, and instead should report failure and request a new target destination. This behavior may be different from that enforced on the violation of the Ptr threshold.

The algorithm begins by sorting the nodes in topological order . In order to transform the eight-connected grid into a directionally acyclic graph (DAG), the graph is 'unwrapped' (see IV-A). Each node is listed in the first row, representing a path starting at any given node. Successive rows in the matrix also list each node, so that the nth row represents the nth step in a given path. If a graph can be topologically ordered, then it can be transformed into a DAG.[?] For example, the cyclic graph in IV-A is turned into a DAG.

Once the nodes are sorted, the next step is to build an adjacency matrix describing the connections present in the input graph. The adjacency matrix, adj is of size $n \times n$, where n is the number of nodes in the graph (equivalently the number of grid cells in the map). The number of rows in the adjacency matrix is thus the maximum length of a path produced by the algorithm. Based on the assumption that there are no negative weight cycles in the graph, the optimal path will visit each node at most one time. Therefore, the optimal path will have at most n steps, which is the number of rows in the adjacency matrix.

There is a separate entry in the matrix for each possibility of traveling from one cell to another. A separate cost may be assigned for moving in different directions between two adjacent points. This allows the algorithm to account for hills or other asymmetrical obstacles, an advantage over many existing algorithms. If the robot can move from cell i to cell j, the entry in column j row i is the estimated cost for this move.

$$\begin{aligned} adj(i,j) &= cost(i \rightarrow j) && (4)\\ &= \frac{1}{2}(H_i + H_j) && (5) \end{aligned}$$

where H_i is the cost function evaluated at node i. If i and j are diagonally connected, the cost should be additionally be scaled by $sqrt(2)$. The cost is a function of the incremental energy required to drive over the surface, which accounts for coefficent of friction and incline. This scheme stores both the connections in the graph and the memoization of costs in the adjacency matrix. Sparse matrix storage and operations are used to drastically reduces running time. The non-zero entries should always be in the same pattern, depicted for a 2x2 grid and a 4x4 grid in IV-A.

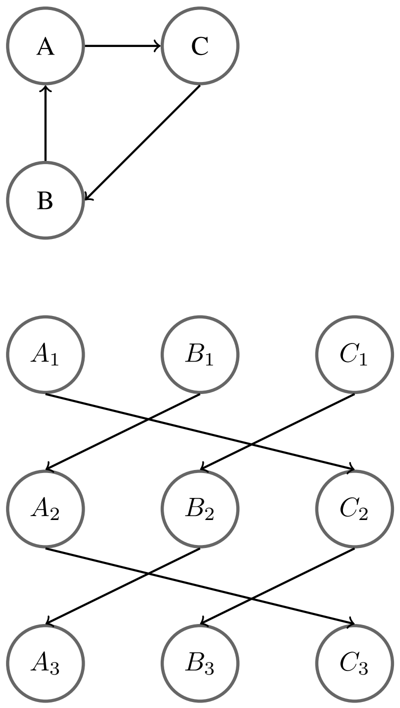

Fig. 1. Conversion of a cyclic graph into a DAG. A similar process will convert the eight-connected grid into a DAG.

Topological order of the nodes is critical for dynamic programming,? and it needs to be in the reverse of the eventual path taken. So, the ordering of nodes begins with the target in the lower right, then moves radially out from this corner. This scheme will be affected by the connections between grid cells. Choosing an eight-connected grid allows for a relatively simple ordering scheme. The order needs to be such that as the algorithm progresses from target node to start node, the cost at each successive node depends only on nodes that have already been completely considered.

The algorithm moves through the adjacency matrix one column at a time, in topological order. Each column represents a map region and a path step number. The unwrapping procedure means that the first N columns represent each node at the first entry in any path, the next N columns represent the nodes at step two in any path, etc. where N is the number of map regions. At each column, cost and probability of traverse for all connections going to that column are calculated. These values are concatenated with the existing list of paths at that node (if any) and the Pareto front of non-dominated paths is saved. The front consists of the paths with the lowest cost and highest $P_{tr}(x)$. Once the algorithm has completed, there is a saved list of paths on the Pareto front from each node to the target node. After the algorithm completes, the output will need to be 're-wrapped.' This process will take each path in the adjacency matrix and associate it with a specific map region. This is the opposite of the un-wrapping procedure shown in figure IV-A. Some heuristic or metric is then used to select a single best path from this front at each node. The output of the algorithm is a list of paths from each node to the target node. Each path is a set of waypoints located at the center of map regions.

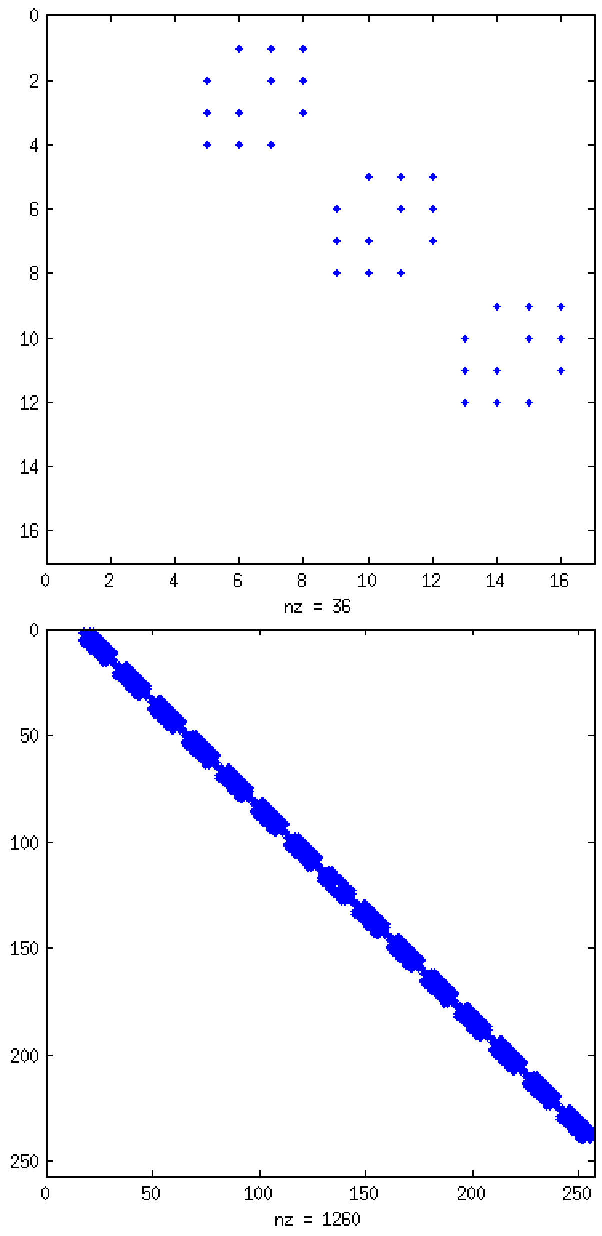

Fig. 2. Placement of non-zero entries for a 2x2 and a 4x4 adjacency matrix. NZ is the number of non-zero entries in the adjacency matrix

B. Environmental Metrics

Each grid segment has several metrics associated with it. These metrics are averaged over an entire grid cell, so they are each a function of position in the map and area covered by a cell.

These are probability of traverse (P_{tr}), obstacle density (μ), and harvested energy (E_h). Probability of traverse reflects the dispersion of obstacles throughout a segment. Dispersion is defined as in the Euclidean dispersion discussed in,? which is the radius of the largest ball that fits into the segment without touching an obstacle. A larger value of dispersion indicates that the robot is more likely to successfully navigate through

25	24	22	19	15
23	21	18	14	10
20	17	13	9	6
16	12	8	5	3
11	7	4	2	1

Fig. 3. Ordering of nodes for an adjacency matrix with the starting node in the upper left and ending node in the lower right. The topological order of nodes should expand outward starting with the target node.

a grid segment. This probability of traverse can be multiplied among sequential map segments to calculate the probability that the robot will get stuck while moving from one segment to another, as in equation 3. A segment covered entirely by a known obstacle, for example a river if the vehicle is a wheeled robot, would have $P_{tr} = 0$.

Obstacle density is defined as the ratio of area covered by obstacles in a map segment to the total area of that segment. This metric is used to calculate the cost function, and reflects the likelihood that the actual path will need to deviate from a straight line through the map segment.

Harvested energy is a function of the available energy in a grid segment. Both the intensity and the type of energy are reflected in this measure; the availability of geothermal energy is not factored into the cost function for a robot which does not carry a transducer to harvest that energy.

C. Cost Function

The cost function is used to give an estimate of total energy usage between two points. Therefore, it is entirely dependent on the vehicle model used. Dong and Sun[?] propose the following cost function based on power usage for movement of wheeled vehicles:

$$J_i = J_{i-1} + 2\mu_{i-1,i} mg s_{i-1,i} \tag{6}$$

where μ denotes the frictional coefficent, i is the node index, m is mass, g is the gravitational acceleration, and s is the distance between points. Additionally, they use a penalty function which affects the cost function as the path moves near an obstacle. Mei et al.[?] note that in addition to motion there are energy costs associated with sensing and computing power, which are relatively constant over time (provided that the usage rate is constant). They also build the energy model for each vehicle empirically.

To account for the possibility that the ground path of the robot (the actual path taken) may not be a straight line from the center of one map region to the next, we introduce an obstacle density $\mu(x_i)$. A higher density of obstacles within a region increases the likelihood that the robot will need to deviate from a straight line path. In building the cost function we assume that the length ground path increases proportionally to the obstacle density.

For a generic wheeled vehicle, the cost function used to evaluate each cell in EAPP is:

$$H_i = (1 + \mu(x_i))(\delta_i s_i + E_c + E_s - E_{h_i}) \tag{7}$$

where l_i is the width of the cell, $\mu(x)_i$ is the obstacle density in a grid cell, δ_i is the estimated incremental cost of driving on the the surface of the cell, E_c and E_s are the estimated computing and sensing costs respectively, and E_h is the estimated energy harvested by the robot from the environment. The cost δ_i is built up empirically, by driving the vehicle on multiple surfaces with different coefficients of friction and inclines. Estimated δ_i in a grid cell comes from the satellite image, which is used to determine type of surface. The robot should have an internal model of types of surfaces to use, and by comparing the surface type with this look-up table will give the incremental cost. Some modification may be appropriate for different vehicle types, if specific motions require a higher energy. For example, treaded vehicles require more power for turning than for going straight, due to the additional sliding friction. The total cost of a path is the summation of the cost of each included cell:

$$J(x) = \sum_{i=1}^{k} H_i \tag{8}$$

where in this case k is the number of cells traversed by a given path x.

D. Assumptions

1) Availability of local navigation algorithm: The algorithm presented in this paper assumes the availability of a local algorithm, as discussed in the algorithm description. The local algorithm must act while the robot is moving to follow waypoints provided by the global algorithm, and it must avoid obstacles. However, in the case that the robot's actual path deviates from the waypoints due to obstacles, a new path is easily generated from the current node to the target. In fact, the algorithm returns the Pareto front of non-dominated paths from each node to the target node, and these are easily stored in a lookup table.

2) No negative weight cycles are present in graph: There must not be any negative weight cycles present in the original graph, although zero-weight cycles are allowed. A cycle is defined as any path segment containing more than one node which begins and ends in the same location. So for all path segments describing a cycle,

$$\sum_{i=1}^{n} J_i(x) > 0, \; n > 1 \tag{9}$$

for any path segment that starts and ends at the same node.

We assert that this is a realistic assumption for any map. Any on-board battery has a set capacity, and once SOC reaches 100% no additional energy can be harvested until some is expended. As harvested energy (E_h) is the only negative term in the cost function, this means that there can be no infinite negative weight cycles. The assumption of no negative weight cycles is then reasonable. The case of a zero weight cycle can be handled by adding a small penalty each time a path returns to a grid segment that it has already visited. In our test environment, there are no zero-weight cycles and therefore no penalty due to the relatively low value of harvested energy compared to the cost of movement.

3) Availability of map data: Map data must be available as input to the algorithm. This means both a satellite image and the image processing capability to estimate environmental parameters. While the design of the algorithm attempts to limit the required information to that which is typically available, there is no guarantee that this information is always available or that the image processing is always accurate. The lack of representative data will affect the performance of the algorithm.

E. Pseudocode

The pseudocode for the algorithm is given in this section. The subfunctions used in the algorithm are described below. Note that the loop beginning in line 3 must run in reverse, beginning with the column containing the goal node.

Inputs to the algorithm are: $\mathcal{G}$, the graph of connections between grid cells; v_{target}, the index of the target cell; and ϵ_c, the minimum acceptable value of P_{tr} for a path. The variable definitions are: v, b, and $path$ refer to a single path; D is a list storing the pareto front of best paths leading to each node (including the path v and the values for cost and P_{tr} associated with it), and the pareto front for a single node is referred to as D_{col}.

```
EAPP($\mathcal{G}, v_{target}, \epsilon_c$)
1: $D := \{\}$ // the pareto front of paths leading to each node
2: A ← BUILD_ADJACENCY($\mathcal{G}$)
3: for all $col$ in adj do
4:     $C$ = FIND_CONNECTIONS($col$)
5:     for all $v$ in $C$ do
6:         $v \leftarrow v \cup col$
7:         $new\_c \leftarrow$ COST($v$)
8:         if $new\_c.P_{tr} > \epsilon_c$ then
9:             $new\_c.cost+ = penalty$
10:        end if
11:        $C \leftarrow C \cup new\_c$
12:        $D_{col}$ =PARETO($C$)
13:    end for
14: end for
15: $paths \leftarrow$ EXTRACT($D, v_{target}$)
16: for all paths do
17:     if $path.SOC \leq 0$ then
18:         $path = \emptyset$
19:     end if
20: end for
21: $b \leftarrow$ BEST($paths$)
22: return $b$
```

1) BUILD_ADJACENCY($\mathcal{G}$) creates the adjacency matrix A from the connected graph by the unwrapping procedure described in a previous section.

2) FIND_CONNECTIONS(col) returns the list of nodes that connect to col. This means any non-zero entry in the column of the adjacency matrix corresponding to the current node, col.

3) COST(v) calculates and returns the total cost, $P_{tr}(x)$, and estimated remaining SOC for the node v. The cost here relates the ending node of a given path and the current node v. As it is built in reverse, the estimate of SOC is only valid once the path reaches the first node of the adjacency matrix (meaning it contains one possible starting node).

4) PARETO(C) builds the pareto front of non-dominated paths from all of the paths passed to it. If the nodes are in the front they are returned, otherwise they are removed.

5) EXTRACT(D, v_{target}) returns all viable paths from the list D for the given node v_{target} These consist of any path which begins with a node in the first row of the adjacency matrix and also contains v_{target}. To speed execution, an additional check can be added in the main algorithm which removes a node from the list C once it contains the target node. If this check is not present, then any nodes in a path which come after v_{target} are ignored in this function.

6) BEST(paths) returns the single best path from the pareto front for each node. 'Best' is a user-defined metric, and may be some combination of cost and probability of traverse. For this paper it is simply the lowest-cost path in each Pareto front.

F. Dynamic programming:

The primary contribution of this work is in the problem definition; the use of $P_{tr}(x)$ and μ to define the environment is novel, and it is possible to design many types of navigation algorithm which use these measures. The proposed dynamic programming algorithm presents several advantages over competing approaches. The primary advantage is that the proposed algorithm considers every possible path of length n or less, meaning that the results are globally optimal according to the cost function and within the constraint. Additionally, the results of the algorithm are a choice of paths from each node to the target node, greatly reducing the burden on the local navigation algorithm. The segmented grid approach is flexible, and estimates of environmental parameters can be improved by refining individual sections of the grid. There is no need for square or homogenously sized grid sections, it is merely a convenience. The algorithm is deterministic, so the same results are returned regardless of whether the algorithm is run on-board a robot or remotely with the resulting waypoints transmitted to the robot. Conversely, the proposed algorithm must be re-run if the threshold on P_tr or the environmental costs change. The algorithm is also not anytime,[?] and its running time may preclude it from use with fast-moving vehicles such as Unmanned Arial Vehicles.

V. Simulation Results

In this section we test the algorithm on a sample environment and discuss the results. The environment used is a section of Rock Creek Park in Washington, D.C. at coordinates: **coords** A satellite image is pulled from (cite google maps?), and the map is broken up into equal area regions. Each region is assigned a cost and a probability of traverse.

figure of costs and probabilities of traverse.

The algorithm is programmed into a MATLAB script file. The robot begins in the bottom left corner of the map and the target node is the top right corner. The resulting best path is shown in figure [fig no].

figure of best path through environment.

VI. Experimental Results

In this section we present results using a mobile robotic platform in a simple environment. The environment consists of three different surfaces, each of which has an experimentally deternmined incremental cost.

A. Robot Parameters

Conor's section

B. Implementation

Conor's section

C. Experimental Environment

The test environment is represented in figure [fig no] by the cost and probability of traverse for each map region. The three different surfaces have different coefficients of friction, and are made of variously squishy? material which changes the cost associated with each region. In order to simulate a probability of traverse, obstacles placed in the environment were shifted slightly between runs. The probability of traverse is an estimate, as it would be for an environment only observed through satellite images.

D. Results

Using an identical map, we tested various starting and target positions. For each set of start and goal positions we tested a straight line path, a path which operates only in the lowest-cost region, and EAPP. The paths are shown here, along with the normalized energy usage for each path.

VII. Conclusion

This paper presents a global navigation algorithm that can extend the duration of deployment for wheeled mobile robots. Our approach targets applications which require long-term autonomy and repetitive tasks, and it uses only information which may realistically be available in those scenarios. We consider energy constraints on the robot, and account for harvesting energy from the environment in order to increase battery life. By introducing a recharging behavior where the robot seeks out a high-energy region and sits until its battery is full, it may be possible to extend the duration of deployment indefinitely. Our method is tested in simulated and experimental environments, with results showing a significant improvement over more naive path choices. EAPP separates probability of traverse from the energy-based cost function, allowing the algorithm to throw out paths which may be efficient but risk the loss of the robot.

Acknowledgment

The authors would like to thank...