Dynamic Programming Method for Energy-Aware Path Planning

Randall Schur, Conor Lyman, and Adam 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 based on the model of the robot and the environment. The proposed algorithm takes a dynamic programming approach to finding the least cost 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 ~~a~~ realistic scenario present ~~a~~ multi-objective optimization problem. 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 ~~for this paper~~ as any obstacle that can damage the robot or stop it from driving), optimizing distance travelled or information gained, minimizing energy usage, or minimizing the time ~~to a goal~~ are all potential goals of the algorithm. As these are often competing interests, the cost function must weight each goal according to the application, along with the uncertainty associated with each piece of information.

The approach in this paper ~~(EAPP)~~ presents a novel context for characterizing uncertainty in the environment by introducing a metric called probability of traverse ($P_{tr}(x)$), which is a function of position. The environment is first segmented into grid cells. Rather than attempting to define a cell by its coverage of obstacles or by its uncertainty as in cell decomposition methods [?], probability of traverse is defined as the estimate of the likelihood that the robot will be able to traverse through the grid square. Any obstacle that the robot cant drive through means that there is some probability the robot will fail to traverse the cell.

Probability of traverse reflects the dispersion of obstacles through 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. 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. A segment covered entirely by a known obstacle, for example a river if the vehicle is a wheeled robot, would have Ptr(x) = 0. Probability of traverse is a pre-calculated metric related to chance constraints (see Blackmore - Robust Path Planning and Feedback Design under stochastic uncertainty) [?], although it is not an exact parallel.

In addition an obstacle density is estimated for each map segment, which 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. The obstacle density can be used to estimate actual path length through a segment.

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. Applications such as inspection of infrastructure such as pipelines or bridges, 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 the limiting factor for mobile robots. 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 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, allowing the most efficient path which does not violate the defined threshold on the likelihood of collision.

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 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, as well as 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 during path planning. Sensing uncertainty is addressed by work such as [?] or [?]. Environmental uncertainty 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.

$$\min_{x\in W}\sum_{i=1}^{n} J_i(x, SOC) \mid P_{tr}(x)_{total} < \epsilon_c \qquad (1)$$

where W is the workspace, n is the number of grid cells, J is the cost function, x is position, SOC is the battery state of charge, $P_{tr}(x)$ is the probability of traverse over the entire path, and ϵ_c is the threshold on probability of traverse.

In any real scenario, each of these quantities is an estimate, as robot environments are unpredictable and models are imprecise [?]. Thus the constraint 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 beyond robot parameters is limited to information which is realistically available in the targeted scenario. The only required input is the estimate of $P_{tr}(x)$, $\mu(x)$, and cost of driving on a surface, which can be generated strictly from a satellite image and information about the robot. The focus of this paper is on the navigation, ~~and we leave~~ the identification and classification of the environment from the satellite images ~~for future work~~.

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. The method used in the results section relies on potential fields. Other options are 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. Workspace is defined as the position of the robot on the surface of the ground, as opposed to Configuration space, curly C, which frequently describes the full state of the robot (cite some configuration space paper). 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 a constant velocity which is the optimally efficient rate. 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) (cite Wickenheiser), 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 ~~energy~~ 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 are solar panels, which when deployed over a larger area are used to generate a significant amount of power (DoE citation here). 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 ~~implementation~~ in this work uses a solar panel attached to the robot.

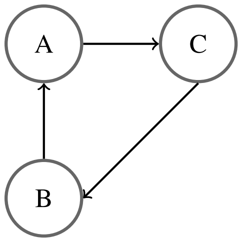
IV. Navigation Algorithm

A. Algorithm Description (EAPP)

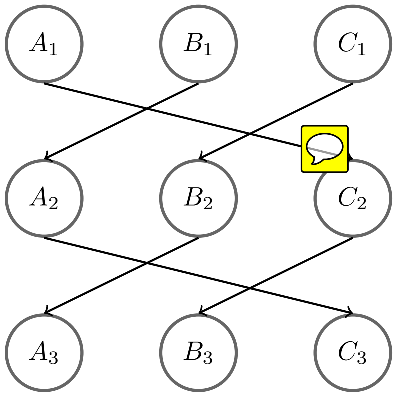
The algorithm is a dynamic programming strategy which takes as input a graph describing the connections between grid cells and outputs the optimal path from each grid cell to some target node. Each vertex in the graph represents the averaged cost over one grid cell, and the graph is formed in an eight-connected pattern. In practice, the constraint on ~~$P_{tr}(x)$~~ 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. This order is critical for dynamic programming, and it needs to be in the reverse of the eventual path taken. In the case of the examples in this paper, the target node is the bottom right corner, and the current position is at the node in the upper left corner. 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. See [?] for details.

Once the nodes are sorted, the next step is to build an adjacency matrix defining the connections present in the input graph. The adjacency matrix is of size n x n, where n is the number of nodes in the graph (equivalently the number of grid cells in the map). In order to transform the eight-connected grid into a directionally acyclic graph (DAG), the adjacency matrix is 'unwrapped'. 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 [?]. So the graph:



becomes:



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. This scheme has the advantage of creating a square adjacency matrix.

There is a separate entry in the matrix for each possibility of traveling from one cell to another. A separate cost 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 A to cell B, the entry in column B row A is the estimated cost for this move. The cost is a function of the incremental driving cost 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 can be used, which drastically reduces running time. The non-zero entries should always be in the same pattern, depicted for a 2x2 grid and a 6x6 grid below.

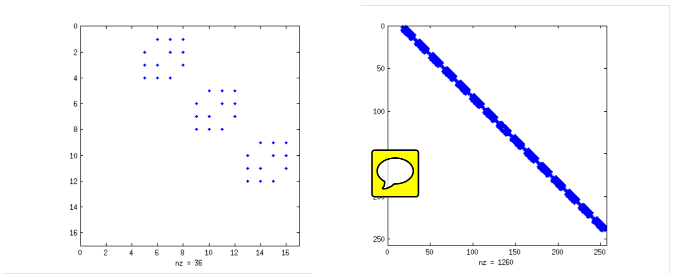


Fig. 1. Placement of non-zero entries for a 2x2 and a 4x4 adjacency matrix

The output of the algorithm is a list of paths from each node to the target node. This list is the Pareto front of non-dominated paths from each node. The front consists of the paths with the lowest cost and highest $P_{tr}(x)_{total}$.

B. 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:

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

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.

For a generic wheeled vehicle, the cost function used in EAPP is:

$$J_i = J_{i-1} + \mu(x)(\delta_s * s_{i-1,i} + E_c + E_s - E_h) \tag{3}$$

where $\mu(x$ is the obstacle density in a grid cell, δ_s 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_s is built up empirically, by driving the vehicle on multiple surfaces with different coefficients of friction and inclines. Estimated δ_s 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.

C. Pseudocode

The pseudocode for the algorithm is given in this section. The subfunctions **BUILD_ADJACENCY**, **FIND_CONNECTIONS**, **COST** are described below. Note that the loop beginning in line 3 must run in reverse, beginning with the column containing the goal node and expanding outward.

```
EAPP($\mathcal{G}, v_{target}, \epsilon_c$)
 1: $D := \{\}$
 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:         $v.path \leftarrow [v.path; col]$
 7:         $new\_c \leftarrow$ COST($v$)
 8:         if $new\_c.P_{tr} > \epsilon_c$ then
 9:             $new\_c.cost += penalty$
10:         end if
11:         $C$ =PARETO(C, new_c)
12:     end for
13: end for
14: $P \leftarrow$ EXTRACT($D, v_{target}$)
15: if $P.SOC >= 0$ then
16:     $path = P$
17: else
18:     path $= \emptyset$
19: end if
20: return $path$, $P_{tr_{total}}$
```

1) BUILD_ADJACENCY creates the adjacency matrix from the connected graph by the unwrapping procedure described in a previous section.

2) FIND_CONNECTIONS pulls the list of nodes that connect to the node given as an argument. This means any non-zero entry in the column of the adjacency matrix corresponding to the current node, col.

3) COST calculates and returns the total cost, $P_{tr}(x)$, and estimated remaining SOC for the node passed as an argument. 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 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 pulls all viable paths from the list given as an argument. 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.

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 graph, although zero-weight cycles are allowed. 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 is the only negative term in the cost function, this means that there can be to 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 may affect the performance of the algorithm.

E. 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 $P_t r$ 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

VI. Experimental Results

A. Robot Parameters

B. Experimental Environment

C. Results

VII. Conclusion

The conclusion goes here.

Appendix A
Proof of the First Zonklar Equation

Appendix one text goes here.

Appendix B

Appendix two text goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LATEX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.