Autonomous Pathfinding in Dynamic Environments Via Graph Cuts

Finding an Optimal Navigation Solution Under Multiple Constraints

David Days[1]

Independent Researcher, South Webster, OH 45682, USA

Abstract. As autonomous vehicles become more sophisticated, ever increasing focus is placed on meeting requirements for operating in dynamic and non-sterile environments. These environments are commonly subject to a variety of vehicular, environmental, and regulatory bounds that become difficult to encode and test. To overcome the difficulties and meet the overhead requirements to function independently, many research efforts have chosen one of two general but often diametrically opposing approaches: Increasing the mapping resolution and decreasing path computation time. This paper introduces a novel approach to pathfinding, using a simple graph structure to represent an n-dimensional environment. After operating constraints and various sensor and non-sensor inputs are applied, the optimal path solution can be directly calculated using Min-Cut/Max-Flow techniques. Experimental results suggest that many of the optimizations and techniques from standard graph cut image segmentation can be directly applied to this method.

Keywords: autonomous navigation, pathfinding, grap cuts, environment modeling

1 Introduction

The magnitude of the potential benefits from autonomous vehicles continues to drive commercial and research efforts towards the creation of semi- and fully-autonomous vehicles for general public use. According to the US National Highway Traffic Safety Administration, obstacle detection and avoidance technologies have the potential to reduce yearly average of 27,000 fatalities and $7.3 billion in property damage that arises from light motor vehicle crashes alone[20]. Internationally, the number of pedestrian fatalities is estimated to be as high as 500,000[33] and is a factor in motivating pedestrian detection and avoidance aspects of research[3]. In agriculture, the efficiencies from autonomous agricultural vehicles can help meet food requirements for the growing human population[5] with a wide variety of tasks being turned over to autonomously-operating equipment[29]. Primary and secondary economic benefits with regards to traffic are also significant; a 5% reduction in traffic congestion alone could save over \$6 billion in wasted fuel and lost productivity[4][41]. Additionally, the use of Unmanned Surface Vehicles (USVs) has been proposed to mitigate the human and economic cost of floods throughout the world[40], which account for 45% of all natural disaster fatalities and 30% of economic damage[19],

Despite these benefits, as well as numerous others, the underlying complexities that are associated with navigation and obstacle avoidance continues to prevent the general deployment of all but the most specialized or restricted public environments. For example, iRobot was granted a US patent for an autonomous lawnmower in February 2015[39], and FCC approval for public sales was granted in August[18]. In contrast, Unmanned Ground Vehicles (UGVs), such as autonomous passenger vehicles, are still undergoing testing in a variety of countries with no scheduled public availability determined[15]. Unmanned Aerial Vehicles (UAVs), which are another field of intense focus, are not allowed to be operated for commercial or research purposes without strict human oversight[45].

For many of these benefits to be realized, autonomous vehicles must be able to operate independently in dynamic, and often chaotic, environments. In addition to detecting obstacles, the planned path must also account for environmental restrictions (such as poor road conditions for automobiles, shallow depths for USVs, or inclement weather for long-range UAVs and USVs), and regulatory restrictions or requirements particular to their operating environment. These non-environmental restrictions can take the form of local traffic patterns or laws, aircraft right-of-way regulations ("see and avoid"), and international laws for maritime collision prevention regulations (COLREGS). Methods for increasing the resolution of the navigation model primarily focus on extracting more precise or more nuanced data from available sensors; however, this often has the tendency to significantly increase the computational requirements for the final path-finding implementation. This, in turn, can result in the vehicle failing to meet real time (RT) requirements that factor into safety and operational requirements for general commercial availability. Alternatively, efforts to solve the navigation problem in a smaller timeframe usually result in simplification of the environment model, effectively discarding available data in order to meet RT restrictions. The cumulative effect of these approaches is that, in general, implementations are often mission- and platform-specific, requiring significant rework to apply in another domain.

In this paper, a novel method for path-finding is presented, based on using existing Min-Cut/Max-Flow algorithms, commonly known as graph cuts. Section 2 describes related work in the field of autonomous navigation. In Section 3, we review the fundamentals of graph cuts and their current application. Section 4 lays out the theoretical model and basic implementation of the approach. Section 5 describes the experimental setup and results, and Section 6 concludes the paper with a review of the results and a discussion of implementing the method for further development.

2 Related Work

2.1 Sensors

For applications that have limited weight or power allowances, such as small UAVs or inexpensive ground vehicles, a common approach to collision avoidance is to use one or more inexpensive and light-weight cameras, which provide data about the surrounding environment[31][21][10][38].Image-only systems have proven capable of operating in a variety of situations. Selby et al. published an experiment in which a small quadrotor UAV was able to use a single camera and onboard target recognition to recognize and track simulated marine mammals[42]. The system developed at the Jet Propulsion Laboratory in [10] was able to navigation a visually noisy environment to a designated landing point with only a single camera. Vision-based Simultaneous Location and Mapping (SLAM) platforms are able to use visual cues, at times augmented by using RGB-D cameras, to successfully navigate ground paths[31][38].

Despite these advancements, many of the implementations are mission-specific. The algorithms in [42] seek targets based on deviations in the relatively sparse environment of the open ocean. SLAM systems generally require, or acquire during operation, highly detailed images of the environment in order to sucessfully navigation; [31] utilized 7.72 GB of storage for a limited operational environment during the experiment. In [38], the vision-only nature of the system led to the research team terminating a number of trials before completion due to collision hazards.

In environments with few or distant tracking detail (large or open areas) or visually poor conditions (at night, or during foggy or inclement weather), visual navigation can perform much more poorly than expected[21]. Radar, sonar, and LIDAR sensors can be used in place of camera sensors, but they, too, are limited to line-of-sight obstacle and target recognition[3]. LIDAR itself is able to produce highly-detailed input, but the production of quality data often entails significant processing and down-sampling of the data stream before usage[47][14].

Combining sensors data, a process generally termed "sensor fusion", is a popular method of overcoming the limitations of individual sensors[47][11]. Typically, it is the output of the sensor fusion process that is pushed to the environment model, discarding the original information as it pertains to navigation[13]. Sensor fusion is not only used to detect obstacles, but can also aid in defining the allowable maneuvering space for a vehicle, as Li et al. did for roads in [28]. In both [13] and [28], multiple LIDAR sensors, aided by one or more radar or camera sensors, were key to performance.

However, the cost of the sensors themselves, as well as the power, weight, and computational requirements, makes their application prohibitive for many other applications[40][11]. [36] provides a detailed model for UAVs that applies uses a variety of available high- and low-resolution sensors, as well as tracking errors; unfortunately, the solution process assumes a large maneuvering space that is not directly applicable to most commercial UGV applications or USVs in constrained spaces. Again, as with single-sensor processing, sensor fusion target identification and modeling is often mission and sensor suite specific.

2.2 Path Finding

Path finding and path planning for UGVs, USVs, and UAVs have also undergone extensive research. [48] studies using a combination of satellite imagery and a version of the A* algorithm as a means of guiding USVs in and out of ports, applying USV dimensions and maneuverablility to improve the solution. Similar to [11] in requiring global knowledge of the maneuvering domain, their system has limited utility where global tracking is unavailable.

The A* algorithm and its succesors are commonly used in research implementations[48][25], but as Shelton describes in [43], the basic A* algorithm must regenerate full paths when recalculation is necessary, while the D* Lite could encounter significant increases in solution times as the environment model becomes more complex. Other modifications to the A* algorithm carry their own costs in complexity or accuracy.

Because environmental complexity becomes a factor in many autonomous navigation problems, there is a body of work on reducing the workload for the pathfinding solution. In some cases, the environment model itself is reduced to a subset of the sensor data; for example, Zivkovic et al. [49] segment the maneuvering environment using graph cut processes to create a smaller search space for the path finding implementation, while Konolige et al. maintain a low-resolution map of the entire maneuvering space, only adding detail to the next immediate path finding problem[25]. As noted earlier, [14] uses highly-detailed input from a LIDAR sensor, but the data is simplified by segmenting terrain data into concave or convex osbstructions; it is these grouped points that are used in the environment model. The large volume of data in [31] is actually a subset of imagery called key frames, which are used to identify particular known positions within the navigation environment. Though these methods reduce the computation time, there is potential for important sensor data to be discarded or subsumed in the process.

Other proposed methods attempt to improve the search for an optimum path. In [37], Rivera et al. maintain a constant set of ideal paths towards a goal, with recalculation only occuring when an obstacle encountered. The paper describes the conditions and processes when a single or continuous object is encountered in 2D maneuvering space, but does not deal with rapidly successive obstacles in 2D or 3D space, as would be encountered by automobiles in heavy pedestrian areas or UAVs in crowded environments.

2.3 Non-Sensory Restrictions

In addition to the basic obstacle avoidance requirements, many of the applications of UGVs, USVs, and UAVs must also account for regulatory restrictions and customary traffic patterns. Li et al. [28] present a method that uses conditional lane detection after both a drivable area has been detected and it has been classified as a road. Although the choice of left or right hand lane is not specifically described, it appears that such a system would require reimplementation and retesting for different countries' motor traffic laws. Bibuli et al. take a mathematical approach by restricting the chosen path of a USV to a particular side of the obstacle, achieving partial compliance with COLREGS governing maritime traffic[6]; further modification of the path finding solution risks interfering with basic functionality. [36] effectively pads aerial obstacles with navigation and tracking errors, and it is conceivable that governing flight avoidance restrictions could be added directly to the padding; however, this method does not account for more general maneuvering restrictions, such as aircraft approach requirements, restricted areas, or weather and Notice To Airmen (NOTAMS), and does not give guidance on how this may be done.

3 Max Flow/Min Cut Segmentation Review

This section of the paper describes the basic usages and operations of Max Flow/Min Cut segmentation, commonly knows as graph cuts. For a more detailed description, see [24], [7], or [23].

3.1 Graph Cuts as Energy Minimization

Segmentation of sensor data is commonly used to support autonomous vehicles. Some implementations use graph cut algorithms to partition the environment model prior to applying the path finding algorithm[49], while most apply graph cut segmentation to the sensor data itself, typically images[34][30][26] or LIDAR[14].

Graph cut segmentation is a combinatorial means of minimizing the labeling cost of nodes within a graph. Although there are many forms of the cost function to be minimized, the general energy function from [24] is commonly used, as shown in Equation 1.

$$E(\mathcal{L}) = \sum D_p(\mathcal{L}_p) + \sum V_{p,q}(\mathcal{L}_p, \mathcal{L}_q) \tag{1}$$

where $\mathcal{L} = \{\mathcal{L}_p | p \in P\}$ is the final labeling of the image represented by P; $D_p(.)$ is the data penalty function, or cost of the labeling assignment; and $V_{p,q}$ is an interaction penalty for neighboring pixels having different labels[24].

In both 2D and 3D image segmentation, Equation 1 is minimized by converting the image into a graph model. Graph nodes represent the individual pixels or voxels within the image, and the edges connecting these nodes are call *n-links*. Classification of structures within the image is typically accomplished by adding two virtual nodes, or *terminal nodes*, and edges connecting each terminal to each graph node, called *t-links*. A two label, 2D image graph model is illustrated in Figure 1.

Capacities of n-links and t-links are usually calculated by one or more forms of similarity measurement, such as Gaussian Mixture Models (GMM) or shape

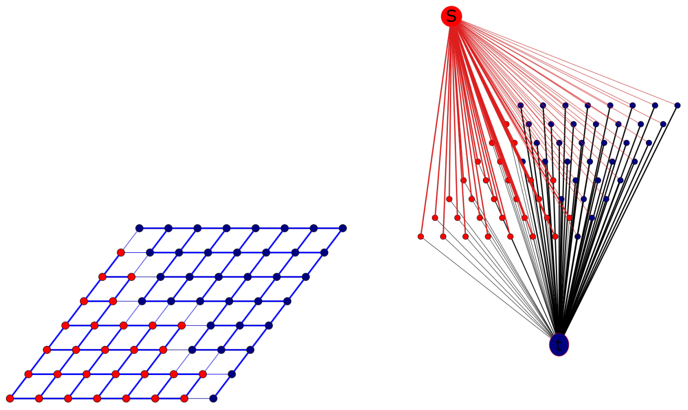

Fig. 1: Graph n-links connecting neighboring pixels and t-links connecting classifier nodes to pixels. Relative capacity of both n-links and t-links are indicated by thickness

priors[17].Generally, graph cut algorithms calculate a *virtual flow* from the designated *source* terminal to the designated *sink* terminal, causing the available capacity of weaker (lower-capacity) edges to fall to zero. When no more flow between the source and sink terminals can be pushed through the graph model, the graph model has been partitioned into two regions: Those that have a path with excess capacity leading to the source terminal, and those that have a path with excess capacity leading to the sink terminal. The n-links that once connected the two regions but now have no excess capacity define the *cut path* separating the two regions, as illustrated in Figure 2.

At the individual pixel/voxel level, the graph cut process allows for labeling based not only on similarities to the classifier, but also to neighboring nodes[16]. Fig. 2 illustrates this in that nodes adjacent to the cut path do not have a direct path to the source or sink node; rather, their label determination is based on an existing n-link to a neighboring node with remaining excess capacity. This allows the cut path to be determined by multiple factors when there are ill-defined borders between label regions while also following a consistent methodology. At the macro level for an entire input data set, graph cuts find the globally optimal or known-optimality labeling based on multiple internal (nodes and n-link capacities) and external (classifiers and t-link capacities) constraints[24][16]. Practical application of this process is explained in the next section.

3.2 Graph Cuts in Autonomous Vehicles

The utility of graph cuts in feature recognition and obstacle detection has been long recognized[44]. As noted previously, [49] segments the 2D map of the local environment to simplify the path finding problem, while [14] processes 3D LIDAR data to extract regions of excessive terrain change which can inhibit

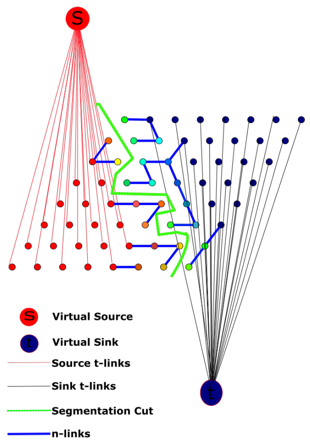

Fig. 2: Segmented 2D image model, showing the remaining t-links and n-links. The virtual cut path is the green boundary separating the two label regions

the UGV's movement. Other implementations use the segmentation process, as previously described, to extract and recognize possible obstacles or landmarks from the image background[26][38].

Figure 3 illustrates this process in the 2D case when the input data is an image frame. The data is modeled similarly to Figure 1, and the initial capacity for the n-links and t-links is set by an implementation-specific calculation. A graph cut implementation is applied, and the final graph is used to determine regions of interest (foreground and background). In non-navigation applications, such as medical image processing and map parsing, the labeled pixels or voxels are of primary interest[1][12][34][35]. In navigation problems, foreground-labeled regions are then passed as inputs to the navigation problem [14][26][38].

3.3 Graph Cut Products

Equation 1 generally defines graph cuts as a process to minimize an energy function. In computer vision applications, the primary purpose is to extract label groups. Inherent in that labeling process, however, is a cut path. In Equation 1, $\sum D_p(\mathcal{L}_p)$ denotes the set of t-links that are "cut" to minimize energy flow between the classifer nodes, and $\sum V_{p,q}(\mathcal{L}_p, \mathcal{L}_q)$ denotes the set of n-links that are cut. For a given graph cut problem, the magnitude of the maximum flow fixed, and the cut path is, if not unique, equal to or better than any other cut path chosen[7].

For computer vision and obstacle detection applications, the precise value for the maximum flow and the set of cut paths are essentially opaque and unusable. Because the cut paths are unique to the image presented, analysis of the cut path generally yields little more information than directly using the label

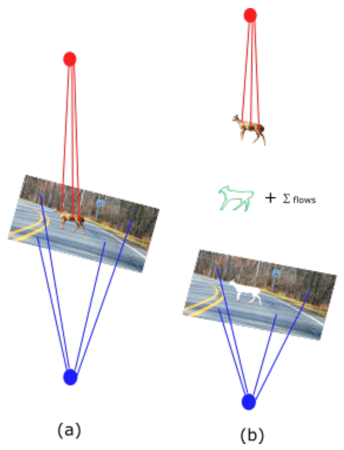

Fig. 3: Illustration of processing an onboard camera frame to detect an obstacle. To the left is the input image and initial modeling. To the right are the products of the graph cut process: Foreground region (connected to the source in red), background region (connected to the sink in blue), the cut path (green line), and the scalar sum of the flows

groups. The maximum flow value, too, is unique to that particular image, and is the product of the various similarity measurements that are implemented and applied. Although rarely used, these two products are readily available results of all graph cut processes.

3.4 Cut Paths As Hyperplanes

Although the most common current usage of graph cuts is foreground/background label extraction, Boykov and Veksler provided an alternate interpretation of the graph cut process as defining a hypersurface through the graph structure[9]. In this interpretation, the n-link set of cuts, $\sum V_{p,q}(\mathcal{L}_p, \mathcal{L}_q)$, are defined by where the optimal hypersurface intersects the model of the input data set, as shown in Figure 4.

Moreover, this hyperplane is not limited to binary cuts of 2D images. The dimensionality of the cut path and the hyperplane are a direct result of the graph structure used, the number of classifier nodes, and the energy functions that are applied[9].

This hyperplane concept allows the model to be partitioned according to a wide range of criteria, and is the basis for graph cut navigation.

4 Graph Cut Navigation

This section begins by building the graph model that will be used to solve a 2D navigation problem. Theoretical and practical considerations will be discussed as the model is built.

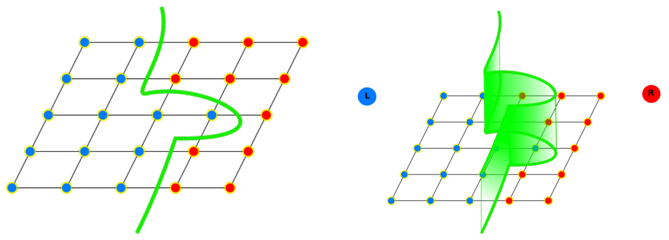

Fig. 4: The left image illustrates the simple cut path, or set of n-link cuts, that is the result of a graph cut process on a 2D image. The right image illustrates the same path interpreted as a hyperplane that intersects the image along the simple cut path

4.1 Overall Structure

The navigation graph for a two-dimensional navigation problem is illustrated in Figure 5. The structure consists of the following major groups:

1. A regularly-connected field of nodes and edges representing the 2D maneuvering plane
2. A terminal node denoting restrictions or costs associated with leftward movement of the vehicle
3. A terminal node denoting restrictions or costs associated with rightward movement of the vehicle in the navigation plane

In this proposal, the model in Figure 5 is intentionally similar to the model shown in Figure 1, as the purpose is to use basic graph cut processes to solve a navigation problem. Throughout the remainder of this paper, design choices and alternative approaches will be discussed as they are encountered.

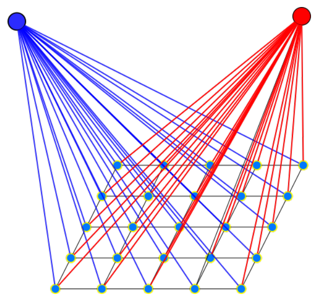

Fig. 5: The proposed navigation structure

4.2 Node and N-Link Field

This section covers the basic description of the local environment model, as well as how sensor data is applied.

Sensor Domain to Graph Domain To represent the region of interest surrounding an autonomous vehicle, a simple mapping of the vehicle- and sensor-centric polar domain to a regular Cartesian coordinate domain is chosen. Graph structures are, essentially, "dimensionless" in that the graph cut processes only consider the nodes and edges internal to the graph. The interpretation of nodes as positions is left to external processes. This allows a very straightforward transposition of coordinates and data, as shown in Figure 6.

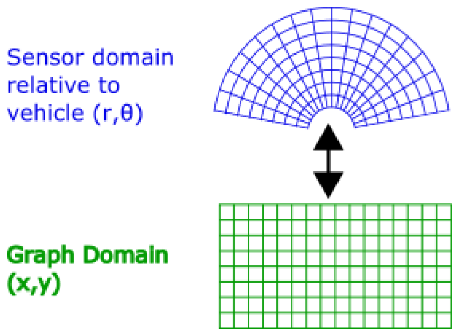

Fig. 6: Coordinate transformation of the sensor domain (r, θ) to the navigation graph domain (x, y)

By choosing a Cartesian coordinate system, this implementation is able to be built upon an undirected graph structure that represents a 4-edge neighborhood for internal graph nodes, with nodes on each side having a degree of 3 and corners a degree of 2.

The regularity of this layout also provides an advantage with regards to computational complexity. As illustrated in Figure 7, the result for an undirected graph is a regular layout of nodes and edges, with each undirected edge based out of the lowest-value node position and connecting it to the higher-valued neighbors, if they exist.

Nodes at the end of the row "lose" a connection, and nodes on the last row also "lose" a connection. The corner node opposite the origin has no additional connections. This arrangement simplifies the storage requirements, and leads to a direct relationship between the resolution of the graph and the size.

$$|E| = 4n - (w + h) \tag{2}$$

where $|E|$ is the number of edges, n is the number of nodes in the graph, w is the number of nodes per row, and h is the number of rows.

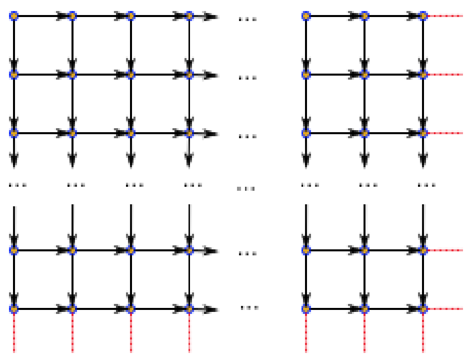
Fig. 7: Construction of the nodes and n-links for the undirected navigation graph. Edges in red are exceptions to the pattern and are not implemented in the navigation graph

Equation 2 indicates that the regularity of the layout makes storage requirements completely predictable.Because $|E| \propto |V|$, where $|V|$ is the node count, the memory used to store the navigation graph is $O(n)$. This memory can be set at design time by the implementation team or predictably allocated according to dynamic requirements. This also removes the necessity of applying complexity reduction techniques to the navigation graph, as in [14], [27], or [49].

At this point, it should be noted that the term “resolution” is being substituted for terms denoting width, range, area, or volume of the maneuvering space. Because the transformation to the graph domain disassociates the nodes locations from real world coordinates, the same graph structure can represent a large sensor area surrounding the vehicle, such as $180°$of arc out to 100 meters, or a much smaller region, such as $1°$of arc with a range of 1 meter.

This degree of freedom with regards to coordinate mapping implies that an optimized graph cut implementation can be used to find an optimal path for a small local sensor with a high degree of spatial resolution in one processing cycle and can then be used to plan a long-term path across a much larger area during the next processing cycle. The change from one form of path finding to another could be accomplished by simply altering the mapping parameters between the domains $(r, \theta) \leftrightarrow (x, y)$.

Additionally, the choice of a Cartesian grid to represent the graph domain was purely for simplicity. In [32], Moran et al. describe the limitations and effects of fusing multiple sensors, and found that changing from a Cartesian representation to a hexagonal coordinate system allows for simplified analysis. There are no technical limitations to utilizing such a structure for the n-link field, and the benefits of the additional analysis laid out in [32] could make such an arrangement worthwhile.

Applying Sensor Data to the N-Link Field Continuing with the hypersurface concept from [9], the optimal path through the n-link field crosses a set of edges rather than traveling from node to node. As a path finding construct, the cut path defines a series of “gates” through which the autonomous vehicles must pass. The overall goal of an optimal path solution for an autonomous vehicle is to maximize the probability of success, which equates to minimizing the probability of failure. It is this minimization of failure that allows the application of graph cuts to the navigation problem.

Graph cut processes are used to find the optimal solution by minimizing Equation 1. As Boykov et al. note in [8], many of the energy function constraints that are solved by graph cuts include log-likelihood models as their basis for labeling. Within the n-link field of the graph cut model, however, there are multiple sensors, sensor-fusion products, and external/internal data models to be applied. The log-likelihood function for data applied to the n-link field is implemented in the following manner. Starting with simple probabilities,

$$p_{success} = 1 - p_{obstacle} \tag{3}$$

where $p_{success}$ is the probability of successful transition across an edge, and $p_{obstacle}$ is the probability that an obstacle, if it exists, will halt passage of the vehicle across the given edge[1]

Using Equation 3 as a basis, the simple probability of successfully transiting an edge between two nodes in the 2D case, p and q, is

$$\begin{aligned} L_{p,q} &= \prod_{k=1}^{S} p_{pk} \prod_{k=1}^{S} p_{qk} \\ &= \prod_{p,q} \prod_{k=1}^{S} p_k \end{aligned} \tag{4}$$

where $L_{p,q}$ is the likelihood of successful transition across the graph edge connecting p and q, S is the set of sensors, p_{pk} is the individual sensor probability of an obstacle located at p, p_{qk} is the individual sensor probability of an obstacle located at q, and p_k is the individual sensor probability at a node[2].

Transforming Equation 4 into a log-likelihood form produces Equation 5:

$$ln\left(L_{p,q}\right) = \sum_{p,q} \sum_{k=1}^{S} ln\left(p_k\right) \tag{5}$$

[1] The term “obstacle” is used loosely in this context. Beyond the conventional definition of an object, such as other vehicles, pedestrians, or trees, or structures, such as buildings and rocks, the term also applies to anything that could prevent movements. Examples include shallows depths for a USV, a storm system for a long-range UAV, steep terrain for cross-country UGVs, or traffic-lane markers that require compliance for autonomous cars.

[2] As with the term “obstacle”, “sensor” is also more general than onboard sensors

Graph cuts generally operate on positive capacities only, so transformation into a negative log-likelihood both meets this requirement and allows us apply it to Equation 1 in the second term, as shown in Equation 6.

$$\sum V_{p,q}\left(\mathcal{L}_p, \mathcal{L}_q\right) = -\sum_{p,q}\sum_{k=1}^{S} ln\left(p_k\right) \tag{6}$$

Practically applied, Equation 6 is implemented by transforming the signal from one or more individual sensors, remote data sources, and applied non-sensor restrictions into probabilities. For the 2D implementation described here, those probabilities are associated with nodes of the navigation graph. Each of the connecting edges to that node have their initial capacity increased by the value $-ln\left(p_s\right)$, resulting in an appropriate increase in the initial capacity of the graph edge for graph cut processing, illustrated in Figure 8.

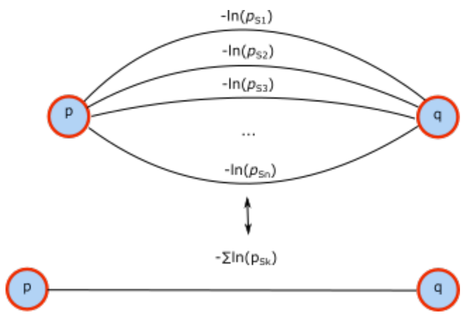

Fig. 8: The upper part of the figure shows the capacity values between nodes p and q as individual inputs from each sensor or subsystem. The data is summed via Equation 6, resulting in a final capacity value for graph cut processing

A common means of adapting relative influence of sensors is to apply individual weights to the individual sensor inputs. This can also be readily implemented in graph cut navigation. The use of weights is shown in Equation 7.

$$ln\left(L_{p,q}\right) = \sum_{p,q}\sum_{k=1}^{S} w_k \cdot ln\left(p_k\right) \tag{7}$$

where w_k is the weight value associated with sensor s_k. The use of such weight values is fairly common; Jacobson et al. applied a similar mechanism to their UGVs, though the weights used were binary 1 or 0 values to individually turn sensors on or off in the system[22].

In a more dynamic environment, it would be possible to adjust the negative log likelihood value for an individual sensor using real values based on current mission, subsystem status, relative capabilities, or environmental changes. For example, IR sensors can experience a phenomenon known as *thermal crossover*, where the environmental temperatures match that of targets of interest. At the same time, camera-based sensors can have difficulty during periods of low visibility or low light conditions, such as fog or night time, respectively. An autonomous vehicle could use environmental factors to adjust the relative weights of these two sensors accordingly, lowering the weight value for the IR sensor during periods of thermal crossover and lowering the weight of camera-based systems during heavy fog or night time. Similarly, sensor fusion processes such as those in [13] and [47], which can produce high quality data, can have higher weight values concomitant with their relative capabilities. While not investigated in this paper, using weights in this manner is a logical step that follows.

Transformation According to Sensor Capabilities Not all sensor inputs would be applied to the n-link field in the same manner; individual sensor capabilities can be taken into account. At the broadest scope, sensors generally fall into *directional* sensors and *point* sensors. Directional sensors are those that, in this environment model, only provide an azimuth of detection. The most common of these are basic image sensors, such as thermal cameras or edge- or object-detection outputs from image-based processing. Point sensors provide both azimuth and range of the detected target; examples include simple fixed radar and sonar sensors, stereo camera processing results, and processed LIDAR data. Map and remote sensor data from other autonomous vehicles could also be applied as point data, once it was translated into the local vehicle-based frame of reference.

Directional sensors can be modeleted as a probability of collision distributed across the area covered by the individual sensors field of view (FOV). Point data would be sampled to match the resolution of the graph domain and applied to the specific nodes of the n-link field. This process is illustrated in Figure 9.

4.3 T-Link Capacity Values

As noted earlier, standard graph cut applications use combinations of similarity metrics to define both n-link and t-link capacities. For binary segmentation of objects from a general background, the labels are often defined by contrasting median values for the regions of interest. This often takes the form of multivariate Gaussian distributions or the application of shapre or texture priors that can select target information out of complex images.

The formulation of Equation 1 is much broader. To produce a simple and regular path, the t-link formulation must simply follow the requirement that it is *graph-representable*, in that the energy function for pair-wise labeling must satisy the inequality in Equation 8 from [9].

$$V_{pq}(0,0) + V_{pq}(1,1) \leq V_{pq}(0,1) + V_{pq}(1,0) \tag{8}$$

where V_{pq} is the "pairwise interaction potential" regarding label values. This is also called the "regularity condition"[9].

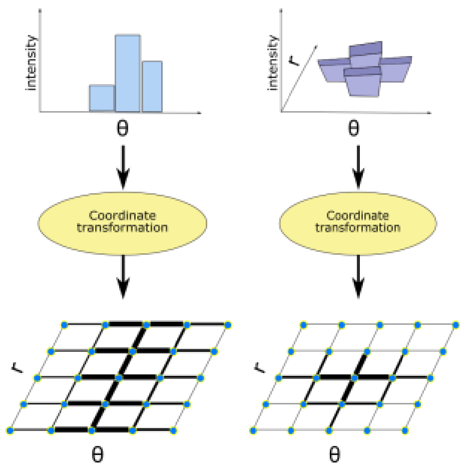

Fig. 9: Sensor data overlay on the n-link field. Transformation on the left is an example of direction-only sensor data. The example on the right is a 2D point sensor that has been transformed into appropriate r, θ values

T-link capacity distributions that satisfy Equation 8 allow for a globally optimal result from a graph cut process. In this application, the goal is to find a globally optimal cut path, so t-link capacities should be set accordingly.

Rather than use a similarity metric, the directional constraints in the 2D navigation problem are on movement left and right of the vehicle's current direction of travel. In essence, the competing constraints apply increasing cost as the vehicle deviates from the desired direction of travel. Simple cost equations, such as linearly-increasing or inverse-distance cost functions, can be applied to the t-links between the appropriate classifiers and the positional nodes in the n-link field.

Note that "direction of travel" is *desired* rather than *constant*. Although a fixed direction of travel could be preferred in some applications, the use of paramterized cost equations for t-links allows the live adjustment of the goal direction. If, for example, the autonomous vehicle is currently trying to move toward a defined waypoint, the relative direction of the waypoint to the vehicle's current direction could be applied directly, with the updated values satisfying the constraints of Equation 8, the hypersurface concept of [9], and the immediate navigation problem of moving toward the waypoint while avoiding obstacles.

Figure 10 illustrates how varying the parameters of a cost function can influence the final labeling, and thus the final cut path. For a no-obstacle navigation graph, the hyperplane from Figure 4 and the corresponding cut path will always go through the set of n-link edges that lie along the equality of the two constraints. By varying the equations, the navigation process sets the desired direction of travel, and the cut path will maneuver around the obstacles accordingly. This is illustrated in Figure 11.

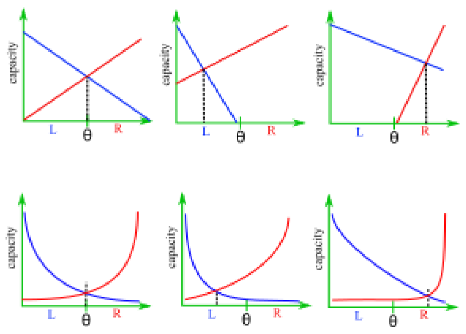

Fig. 10: Examples of potential t-link capacity distributions. The top row illustrates the use of a simple linear cost model. The bottom row illustrates the use of an inverse-distance cost model. The vertical green tick on the θ-azimuth marks the direction of travel. In all cases, the parameters are adjusted so that $L(\theta) = R(\theta)$ at the desired azimuth of travel

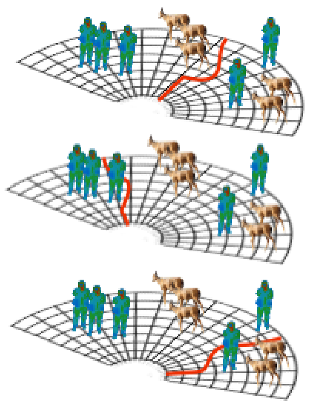

Fig. 11: Examples of the effect of varying parameters for $L(\theta) = R(\theta)$ on the same navigation problem. The top image represents a result for a desired path straight ahead of the vehicle. The center image is the expected result for a desired direction of travel to the left of center. The bottom image is the same navigation problem, but the desired direction of travel is to the right of center

4.4 Computational Complexity and Solution Times

As previously noted, the memory required for the navigation structure is $O(n)$, with the number of edges directly related to the resolution of the graph via Equation 2.

Once the graph cut process is completed, finding the cut path itself is possible in $O(n)$ time. The Boykov and Kolmogorov (BK) algorithm from [7] is particularly attractive for this purpose. The BK algorithm utilizes membership trees to maintain label assignments between iterations of the process, and these tre-based label groups persist after the solution is found. By querying the tree structure for neighboring nodes belonging to a different tree, the cut path can be found directly.

The computational complexity for the BK algorithm is $O\left(EV^2\left|C\right|\right)$, with E being the number of edges, V is the number of nodes in the graph, and $|C|$ is the maximum flow value of the solution[7]. Although the worst-case complexity is dependent upon the final maximum flow value of the solution, the BK algorithm has been found to outperform other algorithms in the binary labeling process[2].

5 Experimental Simulations

5.1 Software Implementation

The initial simulation was written in MATLAB, which provides a standard set of file management utilities, plotting functions, and analytical tools. Though the software itself is not noted for speed or efficiency, the ability to debug code, compartmentalize functions, and graphically present the results made this a reasonable choice.

Graph Cut Algorithm As noted in the previous section, the BK algorithm from [7] was selected because of it is both well-studied and capable of persisting the final graph structure in a manner that is easily analyzed. Slight modifications were introducted in order to extract both cut path results and process information for analysis. The use of a "parentage" array, maintained by the core BK algorithm, was added to the process structure; this allowed for the $O(n)$ recovery of the cut path.

Besides these minor alterations, the BK algorithm itself was left as a naive implementation; no optimizations with regard to the navigation problem were added.

Navigation Graph Structure The navigation graph was implemented as an undirected graph. The edge structure was maintained as an adjacency list, with all n-links located on the lower-valued node of the edge endpoints. T-links were also maintained in the adjacency list, with edge values located on the node position of the classifier. This facilitated determination of the tree-structure within the BK algorithm, as well as allowed for fast determination of the cut path.

Edge capacities were maintained in a separate memory structure. When applying sensor data, the sensor value was added to the existing capacity via graph maintenance functions; successive sensor values were added in precisely the same manner. T-link values were calculated according to a linear constraint model, with the desired direction of travel set for directly ahead of the simulated model except where noted.

Sensor Input Files For each simulated sensor, a MATLAB matrix file was created. The input matrix from the sensor contained the negative log likelihood estimations that reflected the nature of the sensor (directional or point) and the distribution of that target's sensor values across the sensor domain. These files represented the post-transformation inputs of the sensor, mapped onto the graph domain. All areas outside the individual sensor's FOV were set to zero.

For each sensor run, new graph structures were created, and the individual files representing the sensor array and target designations were loaded and applied to the graph.

5.2 Simulation Process

The MATLAB scripts were executed using Octave 4.0.0, running on a 64-bit installation of Windows 7 with an AMD FX-4100 Quad-Core processor and 8.0 GB of memory.

Upon execution, the initial MATLAB script loaded a list of simulation "scenes" to be interpreted by the graph cut implementation. Within the primary loop, a navigation graph was created in accordance with the parameters of the scene, and the n-link and t-link values were set appropriately. Scenes with no input sensors were mixed into the simulation runs to check for large changes in solution times.

A timestamp, using standard MATLAB functionality, was recorded by the system both immediately before and immediately after the graph cut process was called. The difference in these times, the resolution of the graph (as the number of nodes), and the maximum flow value were all added to a running list of process metrics. For each simulation scene, the state of the navigation array was taken before and after graph cut processing. These results were stored within the scene's subdirectory, along with individual process metrics.

Each simulation was run five times successively, and the process metrics for each individual run were recorded to the list. Because the state of each simulation was constant for this experiment, the individual scene results (cut path, maximum flow, and state of the pre- and post-processed navigation graph) were recorded a single time within the scene directory.

5.3 Results

Computation Time Figure 12 shows a scatter plot of the solution times against the n-field node count for each simulation. Each particular scenario is differentiated by marker and color combination. The trend of the solution times appears to follow a polynomial relationship with the graph resolution, as predicted. Also as expected, the maximum flow value for each scenario remained constant; this is in keeping with the concept that a graph cut solution is solely dependent upon the initial state of the graph.

Although the solution times for a particular scenario are generally clustered together, the differences between scenarios of equal graph resolution indicate a higher sensitivity to the maximum flow value than expected. The extent to which this depends on the implementation code and runtime environment will need to be investigated further.

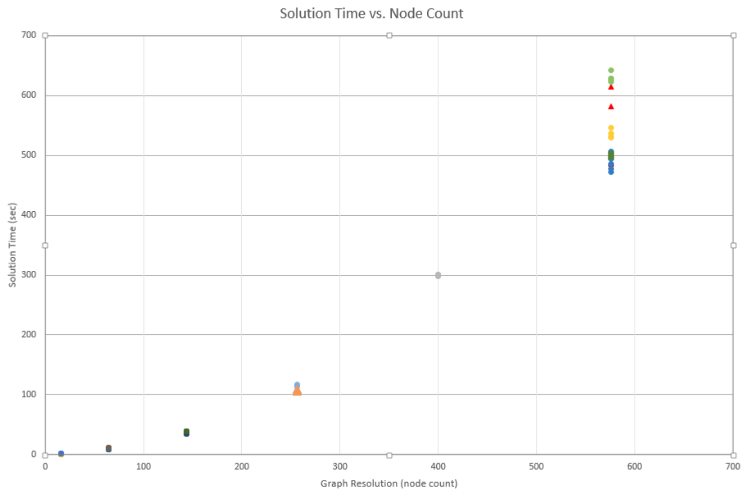

Fig. 12: Plot of graph cut solution time against the graph node count. Colors distinguish the individual scenarios

Sensor Inputs and Path Solutions Figure 13 illustrates a “real world” view of the simulated environment. This particular representation is from the scenario wherein map-based elevation changes for the entire field are combined with another point sensor, such as LIDAR. The map inputs are the red region, representing a steep portion of ground, such as a cliff or a deep drainage ditch; and the circular region in the upper right of the field, which is intended to represent a mound. The other objects on the image are point target detections, such as from a LIDAR or an image analysis system.

The left image in Figure 14 is a surface plot of the same scene, after translation to the two-dimensional navigation graph domain. Regions that have a higher z-value indicate regions with a higher likelihood of failure.

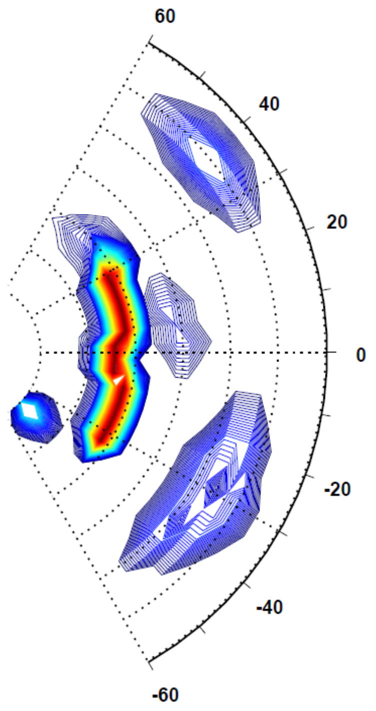

Fig. 13: Polar representation of a simulation scenario. The image represents an example of multiple inputs in the sensor domain

The right image of Figure 14 is an overlay of the path solution on the same scene after processing. . The path solution, shown as a series of diamond-shaped markers, tends to return to the centerline whenever allowed; this is an expected effect of the linear t-link capacity settings. The effects of other guidance cost equations, such as the inverse cost curves previously discussed, should give different results. This, too, is a subject for further study.

For most simulations, the resulting paths were generally smooth and regular, as expected. The irregularities in the path solution for Figure 14 are unique to that simulated scenario. The results have not been discounted, however, because the overall findings were well in line with expectations, and such artifacts merit close attention to determine if they are merely implementation issues or represent boundary cases that require further investigation.

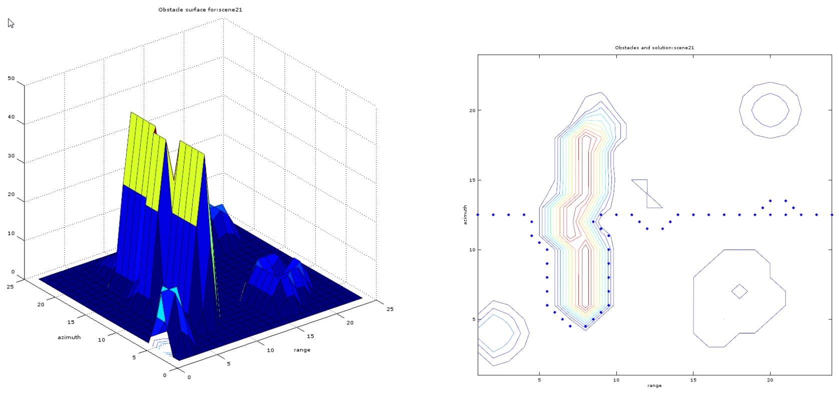

Fig. 14: On the left, a surface plot of the sensor inputs in the graph domain. Z-values indicate the total capacity of the corresponding n-links in accordance with Equation 6 and Figure 8. On the right is the overlay of the path solution on the graph domain. The blue diamonds indicate the crossing of the n-link edges by the solution hyperplane

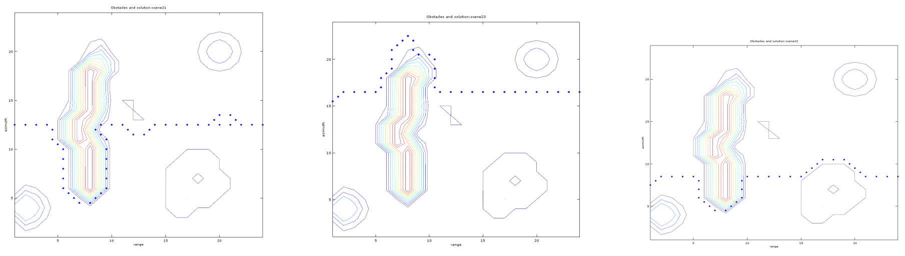

Fig. 15: Results of changing desired direction of travel. For all three solutions, the sensor inputs were the same. The left image had $L(\theta) = R(\theta)$ directly ahead of the simulated vehicle. The center image had $L(\theta) = R(\theta)$ adjusted for a desired path to the left, and the right most image had the desired path to the right

Directional Influence To test the effects of altering the desired direction of travel, a set of scenarios were constructed in which the inputs were identical, but the direction of desired movement $(L(\theta) = R(\theta))$ was altered. The series of results are displayed in Figure 15. As expected, the path solution moves through the obstacle field in a manner consistent with desired direction of travel. While not utterly unique to graph cut navigation, this simple method of adjusting the solution path should provide a straightforward means of handling real world mission or waypoint changes without significant recalculation.

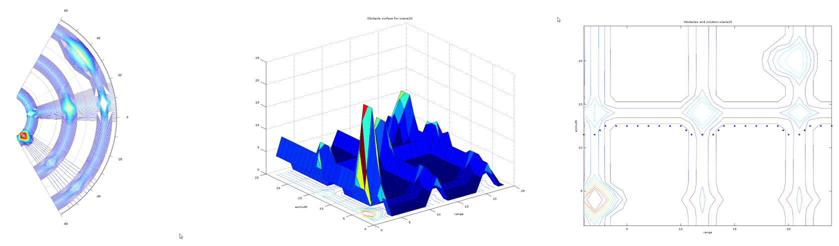

Fig. 16: Scenario for combination of range-only, azimuth-only, and point sensors. The left image shows the sensor domain. The middle image is the graph domain surface plot. The right image shows the path solution superimposed on the solved navigation graph

Low Resolution Sensors Beyond general testing of the pathfinding functions, scenarios were also constructed to simulate combinations of low-resolution sensor data. For example, it is common in naval operations to deal with various forms of sonar that are direction-only (passive sonar), range-only (omnidirectional active sonar), or relatively low resolution point data (long range or remote sensor data). The scenario presented in Figure 16 was constructed to test the applicability of graph cut navigation to such problems. Despite the low level of specific information, the graph cut navigation process selects a path that avoids these higher risk areas, while at the same time following the desired direction of travel.

6 Conclusions and Future Work

6.1 Viability of Graph Cut Navigation

At the most basic level, path finding via graph cuts appears to be a viable means of autonomous navigation. Using a naive implementation of a graph cut algorithm, the ability to extract solutions from various high and low resolution sensor combinations was simulated and successfully demonstrated for 2D navigation.

The experiment also simulated 2D navigation scenarios for common UGV and USV problems, and reasonable solutions were found by the same implementation. This reinforces the concept that the same implementation of graph cut navigation could be applied to most 2D navigation problems without major rework.

6.2 Current Projects and Future Work

Currently, an implementation of the navigation is being built on the basis of commercial off-the-shelf (COTS) sensors and commodity computers in the \$35US-\$50US price range. This hardware and software module is intended for testing in various environments and multiple vehicle types.

Future research into graph cut navigation includes testing the applicability of other graph cut algorithms to the navigation problem, as well as cross-comparison with standard algorithms like A* and D* Lite. Many of the optimizations that have been found for graph cut *segmentation*, the common usage of graph cuts, may be directly applicable to graph cut *navigation*. These optimizations include the use of disparate subgraphs to allow parallel processing and GPU processing to solve much larger segmentation problems in less than 40 milliseconds[46].

Creation of a 3D implementation is also planned; graph cut segmentation is commonly applied to 3D data, so it is logical to expect compatible performance for a 3D navigation problem.

References

1. Ali, A.M., Aslan, M.S., Farag, A.A.: Vertebral body segmentation with prior shape constraints for accurate bmd measurements. Computerized Medical Imaging and Graphics 38(7), 586–595 (2014)
2. Arora, C., Banerjee, S., Kalra, P., Maheshwari, S.: An efficient graph cut algorithm for computer vision problems. In: Computer Vision–ECCV 2010, pp. 552–565. Springer (2010)
3. Becker, D., Schaufele, B., Einsiedler, J., Sawade, O., Radusch, I.: Vehicle and pedestrian collision prevention system based on smart video surveillance and C2I communication. In: Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on. pp. 3088–3093. IEEE (2014)
4. Beiker, S.A.: Legal aspects of autonomous driving. Santa Clara L. Rev. 52, 1145 (2012)
5. Bergerman, M., Maeta, S.M., Zhang, J., Freitas, G.M., Hamner, B., Singh, S., Kantor, G.: Robot farmers: Autonomous orchard vehicles help tree fruit production. Robotics & Automation Magazine, IEEE 22(1), 54–63 (2015)
6. Bibuli, M., Bruzzone, G., Caccia, M., Lapierre, L., Zereik, E.: A collision avoidance algorithm based on the virtual target approach for cooperative unmanned surface vehicles. In: Control and Automation (MED), 2014 22nd Mediterranean Conference of. pp. 746–751. IEEE (2014)

7. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Transactions on Pattern Analysis and Machine Intelligence 26(9), 1124–1137 (Sep 2004), `http://dx.doi.org/10.1109/TPAMI.2004.60`
8. Boykov, Y., Isack, H., Olsson, C., Ayed, I.B.: Volumetric bias in segmentation and reconstruction: Secrets and solutions. arXiv preprint arXiv:1505.00218 (2015)
9. Boykov, Y., Veksler, O.: Graph cuts in vision and graphics: Theories and applications. In: Handbook of mathematical models in computer vision, pp. 79–96. Springer (2006)
10. Brockers, R., Susca, S., Zhu, D., Matthies, L.: Fully self-contained vision-aided navigation and landing of a micro air vehicle independent from external sensor inputs. In: SPIE Defense, Security, and Sensing. pp. 83870Q–83870Q. International Society for Optics and Photonics (2012)
11. Cardarelli, E., Sabattini, L., Secchi, C., Fantuzzi, C.: Multisensor data fusion for obstacle detection in automated factory logistics. In: Intelligent Computer Communication and Processing (ICCP), 2014 IEEE International Conference on. pp. 221–226. IEEE (2014)
12. Chen, X., Summers, R.M., Cho, M., Bagci, U., Yao, J.: An automatic method for renal cortex segmentation on ct images: evaluation on kidney donors. Academic radiology 19(5), 562–570 (2012)
13. Cho, H., Seo, Y.W., Vijaya Kumar, B., Rajkumar, R.R.: A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on. pp. 1836–1843. IEEE (2014)
14. Ding, F., Zhao, Y., Guo, L., Zhang, M., Li, L.: Obstacle detection in hybrid cross-country environment based on markov random field for unmanned ground vehicle. Discrete Dynamics in Nature and Society 2015 (2015)
15. Ensor, J.: Roadtesting Google's new driverless car (2015), `http://www.telegraph.co.uk/motoring/11382073/Roadtesting-Googles-new-driverless-car.html`
16. Freedman, D., Drineas, P.: Energy minimization via graph cuts: Settling what is possible. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. vol. 2, pp. 939–946. IEEE (2005)
17. Freedman, D., Zhang, T.: Interactive graph cut based segmentation with shape priors. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. vol. 1, pp. 755–762. IEEE (2005)
18. Gibbs, A.C.: 'Roomba' Maker, iRobot, gets green light for robot lawn mower (2015), `http://www.cnbc.com/2015/08/13/roomba-maker-irobot-corporation-gets-green-light-for-robot-lawn-mower.html`
19. Güneralp, B., Güneralp, İ., Liu, Y.: Changing global patterns of urban exposure to flood and drought hazards. Global Environmental Change 31, 217–225 (2015)
20. Harding, J., Powell, G., Yoon, R., Fikentscher, J., Doyle, C., Sade, D., Lukuc, M., Simons, J., Wang, J.: Vehicle-to-vehicle communications: Readiness of v2v technology for application. Tech. rep. (2014)
21. Huang, A.S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., Roy, N.: Visual odometry and mapping for autonomous flight using an rgb-d camera. In: International Symposium on Robotics Research (ISRR). pp. 1–16 (2011)
22. Jacobson, A., Chen, Z., Milford, M.: Autonomous multisensor calibration and closed-loop fusion for slam. Journal of Field Robotics 32(1), 85–122 (2015)

23. Juan, O., Boykov, Y.: Active graph cuts. In: Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on. vol. 1, pp. 1023–1029. IEEE (2006)
24. Kolmogorov, V., Zabin, R.: What energy functions can be minimized via graph cuts? Pattern Analysis and Machine Intelligence, IEEE Transactions on 26(2), 147–159 (2004)
25. Konolige, K., Marder-Eppstein, E., Marthi, B.: Navigation in hybrid metric-topological maps. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on. pp. 3041–3047. IEEE (2011)
26. Kristan, M., Sulic Kenk, V., Kovacic, S., Pers, J.: Fast image-based obstacle detection from unmanned surface vehicles (2015)
27. Labatut, P., Pons, J.P., Keriven, R.: Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on. pp. 1–8. IEEE (2007)
28. Li, Q., Chen, L., Li, M., Shaw, S.L., Nuchter, A.: A sensor-fusion drivable-region and lane-detection system for autonomous vehicle navigation in challenging road scenarios. Vehicular Technology, IEEE Transactions on 63(2), 540–555 (2014)
29. Liu, J.J., Wu, L.: The study on autonomous agricultural machinery modeling and control method. Sensors & Transducers (1726-5479) 182(11) (2014)
30. Maggiori, E., Tarabalka, Y., Charpiat, G.: Improved partition trees for multi-class segmentation of remote sensing images. In: 2015 IEEE International Geoscience and Remote Sensing Symposium (2015)
31. Meilland, M., Comport, A.I., Rives, P.: Dense omnidirectional rgb-d mapping of large-scale outdoor environments for real-time localization and autonomous navigation. Journal of Field Robotics 32(4), 474–503 (2015)
32. Moran, B., Cohen, F., Wang, Z., Suvorova, S., Cochran, D., Taylor, T., Farrell, P., Howard, S.: Bounds on multiple sensor fusion. arXiv preprint arXiv:1410.3083 (2014)
33. Naci, H., Chisholm, D., Baker, T.: Distribution of road traffic deaths by road user
34. Newman, P., Chandran-Ramesh, M., Cole, D., Cummins, M., Harrison, A., Posner, I., Schroeter, D.: Describing, navigating and recognising urban spaces-building an end-to-end slam system. In: Robotics Research, pp. 237–253. Springer (2011)
35. Qin, R., Gruen, A.: 3d change detection at street level using mobile laser scanning point clouds and terrestrial images. ISPRS Journal of Photogrammetry and Remote Sensing 90, 23–35 (2014)
36. Ramasamy, S., Sabatini, R., Gardi, A.: Avionics sensor fusion for small size unmanned aircraft sense-and-avoid. In: Metrology for Aerospace (MetroAeroSpace), 2014 IEEE. pp. 271–276. IEEE (2014)
37. Rivera, N., Illanes, L., Baier, J.A.: Real-time pathfinding in unknown terrain via reconnection with an ideal tree. In: Advances in Artificial Intelligence–IBERAMIA 2014, pp. 69–80. Springer (2014)
38. Sadat, S.A., Chutskoff, K., Jungic, D., Wawerla, J., Vaughan, R.: Feature-rich path planning for robust navigation of mavs with mono-slam. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on. pp. 3870–3875. IEEE (2014)
39. Sandin, P.E., Jones, J.L., Ozick, D.N., Cohen, D.A., Lewis Jr, D.M., Vu, C., Dubrovsky, Z.A., Preneta, J.B., Mammen, J.W., Gilbert, D.L., et al.: Lawn care robot (Feb 10 2015), US Patent 8,954,193

40. Scerri, P., Kannan, B., Velagapudi, P., Macarthur, K., Stone, P., Taylor, M., Dolan, J., Farinelli, A., Chapman, A., Dias, B., et al.: Flood disaster mitigation: A real-world challenge problem for multi-agent unmanned surface vehicles. In: Advanced Agent Technology, pp. 252–269. Springer (2012)
41. Schrank, D., Eisele, B., Lomax, T.: Ttis 2012 urban mobility report. Texas A&M Transportation Institute. The Texas A&M University System (2012)
42. Selby, W., Corke, P., Rus, D.: Autonomous aerial navigation and tracking of marine animals. In: Proc. of the Australian Conference on Robotics and Automation (ACRA) (2011)
43. Shelton, L.: Optimizing modern pathfinding methods in imperfect 2d environments. Game Behaviour 1(1) (2014)
44. Sinha, S.N.: Graph cut algorithms in vision, graphics and machine learning–an integrative paper. UNC Chapel Hill (2004)
45. Vincent, J.B., Werden, L.K., Ditmer, M.A.: Barriers to adding uavs to the ecologist's toolbox: Peer-reviewed letter. Frontiers in Ecology and the Environment 13(2), 74–75 (2015)
46. Vineet, V., Narayanan, P.: Cuda cuts: Fast graph cuts on the gpu. In: Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on. pp. 1–8. IEEE (2008)
47. Xu, J., Kim, K., Zhang, Z., Chen, H.w., Owechko, Y.: 2d/3d sensor exploitation and fusion for enhanced object detection. In: Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on. pp. 778–784. IEEE (2014)
48. Yang, J.M., Tseng, C.M., Tseng, P.: Path planning on satellite images for unmanned surface vehicles. International Journal of Naval Architecture and Ocean Engineering 7(1), 87–99 (2015)
49. Zivkovic, Z., Bakker, B., Krose, B.: Hierarchical map building and planning based on graph partitioning. In: Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on. pp. 803–809. IEEE (2006)