Path-Planning Software for Unmanned Ground Vehicles Using Airborne LiDAR Data

ECE4012 Senior Design Project

Team NavX

Project Advisor: Professor Erik Verriest

Project Sponsor: Harris Corporation

Kartik Sastry, ksastry3@gatech.edu, Team Leader

Jacob Jeong, jacobsh.jeong@gmail.com

Alvin O‘Garro, aogarro3@gatech.edu

Antony Samuel, antonysam2000@gmail.com

Submitted

2018 December 8

Executive Summary

A major challenge for unmanned ground vehicles is path planning. Harris Corporation, an American technology company and defense contractor, sponsored the development of a 3D path planning tool for routing towards a potentially moving target, given 3D point cloud data obtained via airborne LiDAR. Harris is interested in mission planning, Intelligence, Surveillance, Target Acquisition, and Reconnaissance (ISTAR), area scanning, emergency situation response, and warehouse/material handling applications of this technology. The cost of this project was $26,348.80, solely based on an entry-level software engineering salary for each member of NavX's team.

NavX has developed software that computes the shortest navigable path through austere terrain for an unmanned ground vehicle. A given ground vehicle is characterized by its physical constraints such as maximum sustainable tilt, and restrictions on the types of terrain on which it can drive. These constraints are then used by NavX's software to determine the navigability of the terrain in question. The path of shortest distance is then computed between specified start and end points, under the constraint that only navigable terrain is traversed. NavX's software first transforms airborne LiDAR data into a directed graph, which models a segment of terrain. NavX then modifies the graph to account for physical limitations of the ground vehicle and a safety factor. Finally, the software executes the A* algorithm, a variant on Dijkstra's Algorithm, to find the shortest, navigable path through the terrain. The software outputs a set of waypoints that may be used to command an unmanned ground vehicle.

During field operation, the waypoints computed by NavX's software will serve as guidance for an unmanned ground vehicle. Another design team was sponsored by Harris Corporation to follow a set of waypoints, such as those produced by NavX. This team's challenge is to handle run-time issues and other unforeseen circumstances using tools from computer vision and control theory.

Table of Contents

Path-Planning Software for Unmanned Ground Vehicles using Airborne LiDAR Data

1. Introduction

NavX has developed software that computes the shortest navigable path through austere terrain for an unmanned ground vehicle. A given ground vehicle is characterized by its physical constraints, which are then used by NavX's software to determine the navigability of the terrain in question. The team requested $26,348.80 to fund the development of this software.

1.1 Objective

Path planning for unmanned ground vehicles is a broad and well-studied problem. NavX focused on the specific aspects of world representation and shortest path routing under vehicle performance limitations. NavX's sponsor, Harris Corporation, provided NavX with high-resolution LiDAR point cloud data with point classification (which determines whether a surface point is water, foliage, or ground). Using this data, NavX first constructed a graph representation of the environment within the host computer. Following graph generation, the A* graph search algorithm was employed to compute the least-cost path through the terrain. The generated path was finally translated into a sequence of coordinates, called waypoints, for commanding a ground vehicle. NavX consulted with another design team, Harris Team B, to develop a suitable format for the waypoints. Harris Team B's work focuses on real-time considerations when following the produced waypoints.

1.2 Motivation

One of the major challenges for unmanned ground vehicles is path planning. Harris Corporation, an American technology company and defense contractor, has sponsored NavX's development of a three-dimensional path planning tool for navigating towards a potentially moving target. The primary input to NavX's path planning software is a three-dimensional point cloud with point classification, defining the terrain over which an unmanned ground vehicle is to navigate. The software also accepts desired starting and endpoints within the terrain, as well as parameters describing the ground vehicle's physical limitations, the desired level of connectivity in the graph model, and a safety factor. Harris Corporation is interested in mission planning, Intelligence, Surveillance, Target Acquisition, and Reconnaissance (ISTAR), area scanning, emergency situation response, and warehousing/material handling applications of this path planning technology [1]. The use of airborne LiDAR to map environments is not new work. Some recent examples can be found in [2], [3], and [4]. However, as advances in sensor technology increase the information content and resolution of LiDAR point cloud data, path planning based on this data is still a desirable venture.

1.3 Background

Motion planning has been a long-standing area of research in the robotics and control communities, with seminal work beginning perhaps as early as 1957 [5], [6], [7]. The fundamental building blocks utilized in NavX's project can be categorized as mapping/representation techniques and planning algorithms. A wide variety of mapping/representation and planning techniques have been established over time. [6] provides an exhaustive classification of mapping schemes, as well as associated algorithms. Based on the nature of the LiDAR data given to NavX, a spatially discrete world representation was chosen. Such discrete representations lend themselves to graph-based models, which allow for the use of graph search algorithms for path planning [8], [9]. For contrast, [10] takes an analytic approach to path planning based on a continuous-space model of the environment.

Initially, NavX's software was to be developed for the existing Tactical Unmanned Ground Vehicle (TUGV) that was produced by a previous ECE 4011/4012 team [11]. Due to battery complications with the TUGV and a lack of documentation, Harris Team B was forced to acquire a different robot platform. At this juncture, NavX made the decision to parameterize their solution with the physical limitations of a given ground vehicle, thereby allowing the solution to be extended to other vehicles besides the TUGV.

2. Project Description and Goals

NavX has developed software that computes the shortest navigable path through austere terrain for an unmanned ground vehicle. A given ground vehicle is characterized by its physical constraints, which are then used by NavX's software to determine the navigability of the terrain in question. The path of shortest distance is then computed between specified start and end points, under the constraint that only navigable terrain is traversed. This path is not guaranteed to exist. NavX's software also includes functionality to rapidly compute paths on existing graphs, hence allowing for the tracking of moving targets. NavX's sponsor, Harris Corporation, anticipates deployment of this technology in austere terrain conditions. A list of the capabilities of NavX's software is provided below:

- Extracts a 3D point cloud, point classification information, and file metadata from LiDAR data given as an .las file compliant with versions 1.0 – 1.4 of the LAS Standard [12].
- Constructs a graph model of the terrain described by the input .las file, with user-configurable connectivity and safety factors.
- Employs the A* graph search algorithm to compute the single-source, least-cost path between user-specified start and end locations in the terrain.
- Translates the result of the graph search into a sequence of waypoints that can be issued as reference commands to a ground vehicle.

Figure 1 illustrates how an end-user would integrate NavX's software into a path planning mission.

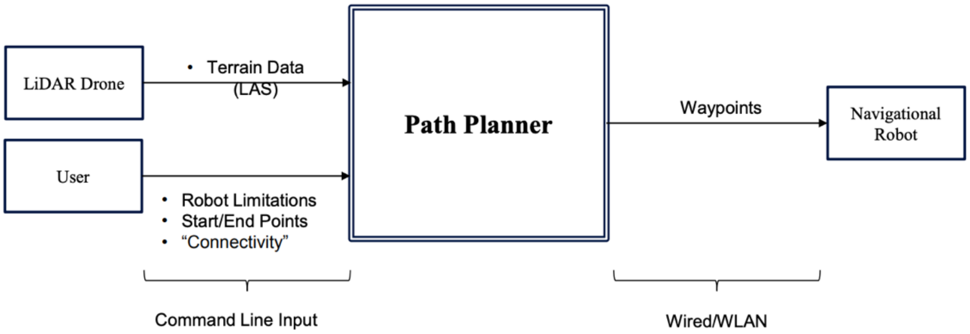


Figure 1. High-level block diagram illustrating integration of NavX's software into a path planning mission.

3. Technical Specifications

NavX's directives from Harris Corporation were to develop a path planning tool that could be deployed on a typical laptop. Table 1 contains specifications for a typical laptop base station, as well as the specifications for the laptop that NavX used during testing.

Table 1: Summary of Base Station Requirements

Base Station Parameter	Minimum Requirement	Specification of Test Laptop
Clock Speed	2.0 GHz	2.7 GHz
RAM	8 GB	8 GB
Disk/Flash Memory	256 GB	256 GB
I/O Capability	IEEE 802.11n compliant WLAN USB Interface	IEEE 802.11n compliant WLAN USB Interfaces

NavX successfully demonstrated their software running on the laptop with the specifications described above. NavX also achieved an internal goal of computing paths on the order of seconds in order to account for potentially moving target nodes. NavX demonstrated a path being computed in approximately three to four seconds on a dataset of approximately 38,000 points, after graph generation.

4. Design Approach and Details

4.1 Design Approach

NavX's software design is summarized in Figure 2. Each of the individual modules shown is encapsulated in software by a class or function. The entire program executes ably on a typical laptop computer meeting the specifications listed in Table 1.

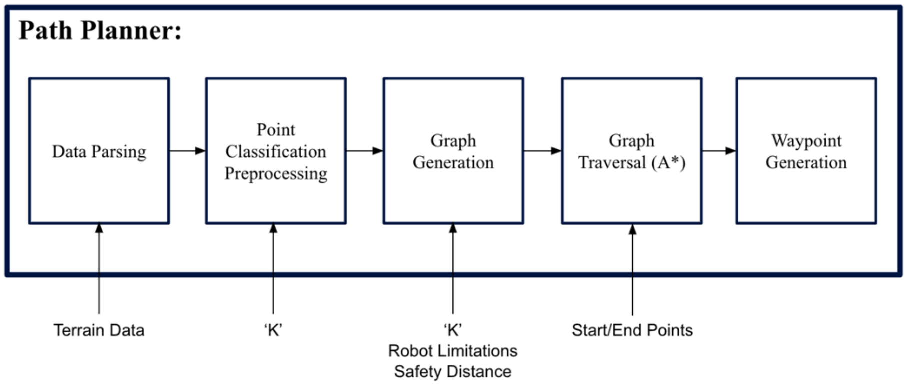


Figure 2. Block diagram illustrating the task flow of NavX's Path Planner software.

4.1.1 Data Parsing

The 'Data Parsing' module is responsible for converting LiDAR data given in the form of an .las file into a usable set of 3D points, labelled with point classification information. Sub-tasks that occur in this module are file parsing, determination of units, and re-scaling of coordinates to the specified units. The code to parse and rescale the data as per the LAS Standard is NavX's original content [12]. The particular segment of an .las file containing units of measure is encoded according to a different standard, the GeoTiff Standard. To parse this information, NavX utilized an existing tool, developed by a PhD student at the University of North Carolina, Chapel Hill [13].

4.1.2 Point Classification Preprocessing

The purpose of the 'Point Classification Preprocessing' module is to reduce noise in the point classification information by exploiting spatial locality – the idea that points in the terrain that are close together are likely to have the same terrain classification. NavX has employed the *K*-Nearest Neighbors classification technique to accomplish this. The LAS Standard specifies 64 unique terrain classification labels, with support for an additional 192 user-defined labels [12]. The ‘unclassified’ label typically corresponds to noise. Thus, in this module, the *K* nearest neighbors of each initially ‘unclassified’ point are identified using a Euclidean distance metric. Each ‘unclassified’ point is then assigned a new classification by taking a vote of its *K* nearest neighbors. Each neighboring point votes with its own classification label, and majority wins. Here, *K* is an integer and is typically odd to avoid ties. In employing this method, certain classes of outliers in the data are accounted for - such as stray ‘unclassified’ points, surrounded by points classified as ‘ground’. Such ‘unclassified’ points are reclassified as ‘ground’ by majority vote.

Due to the binary nature of classification information in the test data that NavX was given, it could not be safely assumed that points labelled as ‘unclassified’ were noise, as illustrated in Figure 3. Thus, the ‘Point Classification Preprocessing’ module is unused in NavX’s present solution. Nonetheless, NavX has included working Python code for this module with the final deliverables, in the hope that it may prove to be useful when performing path planning using a dataset which is richer in classification information.

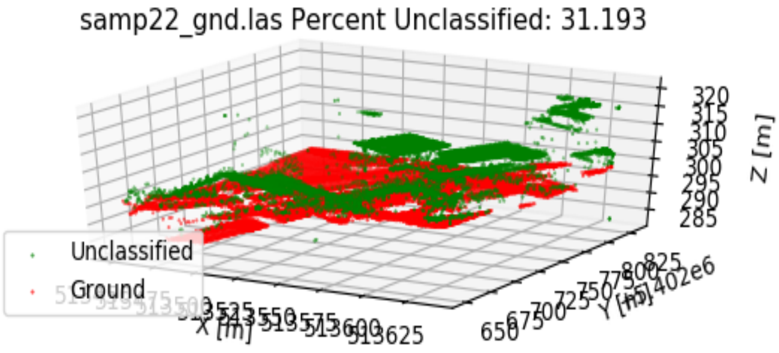


Figure 3. Visualization of test data given to NavX. Clearly, ‘unclassified’ points (green) are not all noise, as vegetation and buildings can be discerned by visual inspection.

4.1.3 Graph Generation

The ‘Graph Generation’ module transforms the point cloud parsed from the LiDAR data file into a directed graph – a set of nodes connected by directed edges. This graph models how a ground vehicle may traverse the terrain. Each point in the three-dimensional point cloud is represented with a node in the graph. An edge represents a straight-line path between the two nodes that it connects. Thus, edge weights are assigned by computing the Euclidean distance between the two endpoints of each edge. It should be noted that any obstacles or undesirable terrain conditions encountered while travelling along these straight-line paths must be handled at run-time by the user.

Initially, the graph is connected by forming a directed edge from each node to each of its *K* nearest neighbors, where *K* is a user-configurable, integer parameter. As in the ‘Point Classification Preprocessing’ module, Euclidean distance is used as the distance metric. That is, for points represented as the tuple:

$$p_i = (x_i, y_i, z_i, Classification_i)$$

the distance between any two points is computed as:

$$\mathrm{dist}(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Once the initial connections have been made, NavX eliminates edges according to several user-provided parameters, as detailed below. After the elimination procedure, the edges that remain are deemed navigable.

Safety Distance

As noted above, each edge in NavX's graph model constitutes a straight-line path between two points. Qualitatively, the risk of encountering undesirable terrain conditions or obstacles while travelling along an edge increases with the length of an edge. NavX defined a safety distance parameter to give the user a degree of control over this risk. This distance must be specified in the same units as the input LiDAR data. NavX's software then eliminates all edges in the graph that have edge weights (path lengths) greater than or equal to the safety distance.

It has been NavX's experience that due to the non-uniform spatial distribution of points in the point cloud data, it is often necessary to increase the connectivity parameter, *K*, to yield a graph in which a path exists between a given start and end node. Since the points in a given dataset are fixed, increasing *K* also has the effect of forming edges with larger weights. In this case, the interplay between the safety distance parameter and *K* is exploited to form a graph in which a path may be found, while placing a (qualitative) bound on the risk posed by travelling along the longest edge.

Forward and Lateral Tilt Limits

NavX's software also accepts two inputs defining the maximum tilt that a ground vehicle can experience. One of these parameters defines the 'forward tilt limit' – the maximum incline, up or down, that the ground vehicle can sustain in the direction of its travel. The second parameter is the 'lateral tilt limit' – the maximum incline that the ground vehicle can sustain in the direction perpendicular to its motion. Both limits are to be specified in degrees. NavX has developed a method for estimating the forward and lateral tilt experienced by a ground vehicle while traveling along a given edge in the graph. If either tilt estimate exceeds the user-provided limits, the edge in question is removed from the graph. NavX's method for estimating the tilt experienced by a ground vehicle while travelling along an edge is outlined below.

For each edge in the graph, NavX first extracts the coordinates of the start node, as well as those of its K nearest neighbors. The plane tangent to the terrain at the start node is estimated by fitting a plane to this set of $K + 1$ points such that the sum of squared errors in the z direction is minimized. The analytical solution to the regression problem is given in Appendix A. The solution is then programmed using matrix operations from the open-source Eigen library, "a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms" [14].

From the defining equation for the tangent plane, which has the general form $ax + by + cz + d = 0,\ a, b, c, d \in \mathbb{R}$, an angle, α, is computed, defining the incline of the plane with respect to flat ground. Next, the start and end points of the edge in question are projected onto the tangent plane, as they do not lie in the plane in general. An angle, θ, is then computed, which defines the heading of the ground vehicle with respect to the horizontal, as viewed in a reference frame attached to the tangent plane. Concise formulas for the estimated forward tilt, $\hat{\theta}_F$, and estimated lateral tilt, $\hat{\theta}_L$, were found to be:

$$\hat{\theta}_F = \sin^{-1}(\sin\theta \sin\alpha)$$

$$\hat{\theta}_L = \sin^{-1}(\cos\theta \sin\alpha)$$

The details of these calculations are given in Appendix A. The estimated forward tilt, $\hat{\theta}_F$, and estimated lateral tilt, $\hat{\theta}_L$ (for each edge) are then compared against the user-specified tilt limits, and any edge found to present too much of an incline to the ground vehicle in either the forward or lateral directions is removed from the graph.

Terrain Limitations

The final criterion that NavX uses to ensure the navigability of terrain is terrain classification. Since the classification information in NavX's test data is binary with each point being assigned either 'ground' or 'unclassified', NavX's software eliminates all edges for which the start node and end node have different classification labels. Thus, if the specified start and end nodes have desirable classification labels like 'ground', and a path between them exists, every node along the path will have the same classification label. A point of future work could be to extend this feature to specifically include or exclude particular combinations of terrain classifications – for example, in the case of amphibious vehicles which can travel on both land and water.

After edges have been eliminated as described above, one final step is required to complete the graph generation. It should be noted that the relationship of two points being nearest neighbors is not a symmetric one. That is, if point p_1 is in the set of nearest neighbors of point p_2, it is not necessarily true that point p_2 is in the set of nearest neighbors of point p_1. Therefore, up to this point in graph generation, the graph does not faithfully model the connectivity between two points in the real world, as there is generally no reason to connect two points in space in a one-way manner. Since at this point, all edges present in the graph are navigable, NavX completes graph generation by 'mirroring' each edge. That is, if, after elimination, an edge is present connecting point p_1 to point p_2, another edge from point p_2 to point p_1, is introduced – effectively converting the graph from directed to undirected.

Visualizations of the edge elimination and path output are available for a variety of intuitive, artificial test cases on NavX's website under 'Deliverables', 'Intuitive Test Case Results': (http://ece4012y201808.ece.gatech.edu/fall/sd18f04/testPseudoData.html).

4.1.4 Graph Traversal and Waypoint Generation

The ‘Graph Traversal’ module implements the A* graph search algorithm. The inputs to this module are a graph containing only navigable edges, and the user-defined start and end nodes. A* is a single-source, shortest-path algorithm which improves upon the runtime of the more commonly known Dijkstra’s Algorithm through the use of a heuristic, which is an estimation of the path cost to the end node. The run time complexity of A* depends heavily on the heuristic used, but it has an average case complexity of $O(b^d)$, where b is the *effective branching factor* and d is the number of nodes in the path [15]. The effective branching factor relates the number of nodes explored during search to the number of nodes in the path. In general, b is very close to 1 if the error of the estimated cost is very low. That is the case for NavX’s Euclidean heuristic, so the performance of A* is generally trivial for the range of expected input sizes.

The path found by A* is proven to minimize distance if the heuristic used is both *admissible* and *consistent* [15]. The heuristic and edge weights have the same units – units of distance in NavX’s application. The heuristic said to be *admissible* if, for each node in the graph, it is less than or equal to the actual distance from the node to the desired end node. The heuristic is said to be *consistent* if it obeys *triangle inequality*; that is to say that the estimated cost from a node *n* to the end node is less than the sum of the cost to a neighboring node *n’* and the estimated cost from *n’* to the end node. NavX computes the heuristic at each node in the graph as the Euclidean distance between that node and the desired end node. This heuristic, like all distances in Euclidean space, obeys triangle inequality, hence the heuristic is *consistent*. It is also clear that the heuristic is *admissible*, since straight line distance is the shortest possible path connecting two points in Euclidean space and therefore cannot overestimate true path cost. Because NavX's heuristic maintains these two properties, the generated path is guaranteed to minimize distance with respect to the generated graph.

After the least-cost path has been determined by the A* algorithm, 'Waypoint Generation' is performed by simply building a list of the coordinates of each node along the path. The coordinates are given with reference to the origin implicitly defined in the input .las file, and in the same units of measure as the coordinates in the original .las file. An end user performing localization in another coordinate system (perhaps using GPS) will need to transform between the two coordinate systems as necessary.

A noteworthy feature of the 'Graph Traversal' module is the ability to compute a path on an existing graph. Given a new .las file, NavX's software produces a .csv file containing the associated graph model during graph generation. In NavX's testing, the generation of a graph took between 5 and 8 minutes for a dataset of approximately 38,000 data points. Once this graph is generated, the actual path computation via A* completed in 4 to 5 seconds. In order to account for the scenario of a moving endpoint, NavX developed the ability to check for an existing graph file (as a .csv file) before starting graph generation. If the graph already exists, the graph generation phase is skipped over entirely, and a path is rapidly produced on the order of seconds. This allows NavX's solution to be applied in the case of a moving endpoint, with the constraint that the endpoint coordinates must be updated at a rate lower than the time to compute a path (4-5 seconds).

4.2 <u>Codes and Standards</u>

A significant standard for NavX's design is the LAS Standard, which defines the format for LiDAR data. The LAS Standard is a public, binary file format for storage and exchange of 3D point cloud data [12]. LiDAR .las files contain a variety of data including (terrain) point classification, pulse intensity, GPS time stamp, number of return pulses. These data are used to determine if the ground vehicle can physically pass through points in its environment. The first module in NavX's software, 'Data Parsing', primarily consists of a parser which was written with the LAS Standard in hand. The

LAS parser developed by NavX is fully backwards-compatible with versions 1.0 through 1.4 (current) of the LAS Standard.

Though NavX has not explicitly addressed this in their design, the IEEE 802.11 Standard for wireless communication and the RS-232 standard for wired, serial communication may be of importance if the waypoints need to be sent between computers [16], [17]. NavX anticipates that waypoint computation will occur on a host computer onboard the ground vehicle, given that their software runs ably on a civilian computer with specifications outlined in Table 1.

4.3 Constraints, Alternatives, and Tradeoffs

4.3.1 Constraints

One constraint on NavX's design is the spatial resolution and distribution of points in the given LiDAR data. This limits the 'optimality' of the path generated in the sense that a finer spatial resolution may yield different results. Furthermore, increased spatial resolution will yield edges with shorter distances (lower weights), on average. This also reduces the risk associated with assuming that a safe, straight-line path exists between every pair of connected nodes, as discussed above.

The amount of RAM available on the host computer also constrains NavX's software to operate on datasets of limited size. To understand the memory requirements of NavX's software, let *N* be the number of data points in the given LiDAR dataset, and *K* be the connectivity parameter, as usual. As noted above, points are represented as tuples of the form $p_i = (x_i, y_i, z_i, Classification_i)$. NavX's software stores points as C structures with four fields - three fields of floating-point type to store *x*, *y*, and *z*, and one field of integer type for encoding the classification label. Per the C99 Standard, floating point numbers require a minimum of four bytes of storage, while integers require two bytes, for a total of at least 14 bytes required per point. Thus, for a dataset of *N* points, 14*N* bytes of memory will be required to store the point cloud alone. Since the number of data points is unknown at compile-time, memory for storage of the points must be allocated dynamically on the heap.

In order to reduce to memory footprint of the graph generated, NavX constructs graph nodes using C pointers. C pointers require either 32 or 64 bits (4 or 8 bytes) of storage, depending on the underlying architecture of the host computer. On a host computer with a 64-bit architecture, storing an 8-byte pointer instead of 14 bytes per point yields a significant savings of 6 bytes per point. A node is stored as a C structure with *K+1* fields – one field containing a pointer to the point represented by the node, and *K* fields containing pointers to the node's *K* nearest neighbors. Each node in the graph is initially is connected to *K* other nodes. After the elimination procedure, *K* gives an upper bound on the number of edges extending from a given node. Thus, an upper bound for storage requirements of the graph is 8**N*(K+1)* bytes for a 64-bit architecture. For the datasets that NavX was given, *N* is on the order of 100,000, and *K* can be as high as 10. Thus, the graph and point cloud together require storage on the order of tens of megabytes. Since programs typically do not have access to the entire RAM of the host computer, the storage requirements become significant as *N* and *K* are increased. While RAM is not a significant constraint on a laptop computer, NavX's software is also able to be deployed on an embedded platform, in which case this becomes more of a limiting factor.

Perhaps more restrictive than memory limitations are the effects of increasing *N* and *K* on the runtime of the program. During NavX's testing, it was determined that the graph generation procedure accounted for the majority of the program's execution time. In the present implementation, forming the initial graph is an $O(N^2)$ procedure, as it involves a comparison between every pair of points in the dataset. The edge elimination procedure is carried out on each of the *K*N* edges in the initial graph, making this an $O(N)$ procedure, as $K << N$. Clearly, as *N* grows, the $O(N^2)$ graph formation procedure dominates the runtime. In NavX's testing, graph generation alone on a dataset of approximately 38,000 points ran for 5 to 8 minutes. If the end user had any restrictions on the runtime of graph generation, the $O(N^2)$ nature of graph generation could prove to be restrictive, as the number of data points in the input data is increased (for instance, due to a desire for increased spatial resolution).

4.3.2 Alternatives and Tradeoffs

As mentioned earlier, based on the nature of the LiDAR data given to NavX, a spatially discrete world representation was chosen. Such discrete representations lend themselves to graph-based models, which allow for the use of graph search algorithms for path planning [8], [9]. Additionally, an established history in the robotics literature and ease of implementation make the pairing of a discrete-space map representation and a graph-based planning algorithm an attractive option. As an alternative, NavX could have pursued a continuous-space world representation and planning algorithm, as in [10]. However, this would require some form of interpolation or otherwise forming a continuous model of the terrain from the sampled LiDAR data. While the path could then be found analytically, the discrete-to-continuous modeling would be subject to evaluation, and likely be the limiting factor in the quality of the path produced.

Another alternative that NavX could have pursued is to employ Dijkstra's Algorithm in lieu of A*. While Dijkstra's Algorithm is more interpretable and simpler to implement, A* produces the same result in significantly less time. The run time complexity of Dijkstra's Algorithm when an end node is specified also has the form $O(b^d)$, where *b* is the *effective branching factor* and *d* is the number of nodes in the path [15]. As mentioned above, the effective branching factor relates the number of nodes explored during search to the number of nodes in the path. In general, b is higher for Dijkstra's Algorithm than A*, because there is no heuristic to limit the number of nodes explored per iteration. Due to the timing advantage of A* over Dijkstra's Algorithm, NavX elected to implement the former as the graph traversal algorithm.

Being a purely software-based project, another significant trade-off in NavX's design was between memory footprint and code interpretability/development effort. As noted above, NavX conserved memory by using pointers to define graph nodes. This gave rise to significant development effort in the debugging phase.

4.4 Implementation Details

Please see Appendix B for a listing of code and results available on NavX's website. Code is well documented, and usage is made explicit through header files and/or comments.

5. Schedule, Tasks, and Milestones

The Gantt chart provided in Appendix C lists all major tasks that NavX anticipated as part of this project, the person(s) assigned to the tasks, and their respective completion times in number of days. The tasks that are on the critical path of this project in red, and tasks off the critical path are indicated in blue. Orange bars are also shown to visualize the effect of uncertainty in task completion times. The PERT chart provided in Appendix D shows a task-flow diagram with all tasks and their completion times, and accounts for uncertainty in task completion times.

The tasks on the critical path, namely developing a map representation and implementing the path planning algorithm, required the most effort and time due to lack of prior experience by team members, as well as time required for testing and documentation. Additionally, for tasks that involved a refinement phase, a buffer period of one week was allocated to cope with uncertainty.

Developing a map representation and implementing the path planning algorithm, each required an additional week for completion and troubleshooting, beyond what is shown in the Gantt Chart in Appendix C. However, NavX was able to complete their work before the Capstone Design Expo by troubleshooting multiple tasks in parallel and eliminating dependencies where possible through the use of simulated results.

6. Project Demonstration and Final Presentation

NavX's final presentation discussed the functionalities of the software, as well as the entire process development process and special considerations. A live demonstration was deemed most effective in presenting each key component of the path-planning software listed in Section 2, as well as for showcasing overall computation time. The presentation began by introducing the project's motivation and objectives, as well as specifications and requirements provided by Harris Corporation. Afterwards, a brief explanation of the order and contents of the demonstration was followed by an explanation of the components of the software, primarily using a block diagram to portray the individual elements of the software.

Afterwards, a real-time demonstration of graph generation and path planning on a variety of intuitive test cases was conducted. The timing for both graph generation and path computation was negligible in all of these test cases. The output of NavX's software was visualized using MATLAB. The results of this demonstration can be found on NavX's website at http://ece4012y201808.ece.gatech.edu/fall/sd18f04/testPseudoData.html. A second demonstration using LiDAR data given by Harris Corporation was conducted next. A 3D visualization of the LiDAR data and path output were generated in MATLAB. NavX also obtained satellite imagery of the terrain using latitude and longitude information in the .las file for a visual comparison. A visualization of the results for one dataset are given in Appendix E. This dataset contained approximately 38,000 data points. The graph model was generated in 5-8 minutes, and the shortest navigable path between specified start and end points was computed in 4-5 seconds following graph generation. A second run of NavX's software was also conducted to demonstrate the ability to compute paths from existing graphs with significantly less total runtime (4-5 seconds versus 5-8 minutes).

7. Marketing and Cost Analysis

7.1 Marketing Analysis

The project's sponsor, Harris Corporation, is the sole customer for this project and defined the system requirements and specifications. While other LiDAR based navigational systems already exist, this project has the marketing advantage of being designed solely to Harris Corporation's design criteria [2], [3], [4].

7.2 Cost Analysis

NavX's product is purely software-based. As such, there were no physical parts to purchase. Furthermore, Harris Corporation has provided pre-processed LiDAR data for testing, so there is no need for NavX to purchase any software. Thus, prototyping and project costs consisted solely of an engineering salary for each of NavX's four members.

Each of the NavX's four members worked an average of ten working hours per week. With a $41.17/hour software engineering salary, each of NavX's four members required $6,587.20 in compensation, yielding a total cost estimate of $26,348.8 [18]. The suggested selling price for NavX's product is equal to the total cost estimate since the sole customer of the product is Harris Corporation.

8. Conclusion

NavX's path planning software successfully computes the shortest navigable path through a given segment of terrain, described by an .las file. NavX's solution accounts for limitations of a given ground vehicle, such as maximum sustainable tilt in the forward and lateral directions, and terrain type restrictions. NavX's software meets the design requirements set by the sponsor. The path planning software runs ably on a typical laptop computer with specifications outlined in Table 1. For a terrain data set consisting of approximately 38,000 points, a graph model is generated in 5-8 minutes. This graph model is user-configurable with variable connectivity and a safety distance. The graph model needs to be generated only once per dataset, unless the user's configuration preferences change. Following graph generation, the shortest navigable path between specified start and end points is computed in 4-5 seconds. The start and end points may be changed both within, and across invocations of NavX's software, allowing for tracking of a moving endpoint – a 'stretch goal' set by NavX's sponsor. NavX's software finally outputs a set of waypoints between a given start and end point for an unmanned ground vehicle to follow. These waypoints are given with reference to the same coordinate system as the input data.

For future work with respect to the modelling side of the problem, NavX suggests accounting for the physical dimensions of a ground vehicle, and a more detailed treatment of terrain restrictions. The safety distance parameter could also be made more rigorous with a probabilistic model for risk of undesirable terrain conditions as a function of path length. On the algorithm design and performance optimization side of the problem, multithreading and batch computation of tilt estimates could be explored. With more aggressive performance restrictions, different world representation techniques and advanced data structures could be considered to optimize for either runtime or memory footprint.

9. Leadership Roles

Leadership roles are defined by the requirements of ECE 4011 and ECE 4012. Alvin O'Garro was NavX's webmaster and led the research and development of the path planning algorithm. Jacob (Shin Hyun) Jeong led data preprocessing and oversaw task scheduling and design expo coordination. Antony Samuel was the liaison to the team's corporate sponsor, Harris Corporation, and oversaw data acquisition. Antony also contributed to website design and led presentations. Kartik Sastry was NavX's Team Leader. Kartik worked on data parsing, world representation, and tilt estimation. He also served as the team's documentation coordinator and the liaison to the team's project advisor, Professor Erik Verriest.

10. Acknowledgements

The NavX team would like to thank Professor Erik Verriest, our Faculty Advisor; Mr. Michael McGonagle, our sponsor at Harris Corporation; as well as Professor Bruno Frazier, Professor Whit Smith, and Mr. James Steinberg for their support, feedback, and mentoring throughout the two-semester design process in ECE 4011 and ECE 4012.

11. References

[1] McGonagle, M. (2014). University Capstone: Research robotic path planning with moving end point or target.

[2] Y. Egi, C. Otero, M. Ridley and E. Eyceyurt, "An Efficient Architecture for Modeling Path Loss on Forest Canopy Using LiDAR and Wireless Sensor Networks Fusion," *European Wireless 2017; 23th European Wireless Conference*, Dresden, Germany, 2017, pp. 1-6.

[3] C. Zhang, J. Wang, J. Li and M. Yan, "2D Map Building and Path Planning Based on LiDAR," *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, Changsha, 2017, pp. 783-787.

[4] L. Wang, J. Wang, X. Wang and Y. Zhang, "3D-LIDAR based branch estimation and intersection location for autonomous vehicles," *2017 IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, CA, 2017, pp. 1440-1445.

[5] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991

[6] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006

[7] L. E. Dubins, On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents, *Amer. J. Math*., vol. 79, no. 3, pp. 497-516, 1957

[8] A. V. Aho and J. D. Ullman, Foundations of Computer Science: C Edition. New York, NY: W. H. Freeman, 1994.

[9] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, Y. Xia, “Survey of Robot 3D Path Planning Algorithms,” Journal of Control Science and Engineering, vol. 2016, April, 2016. [Online serial]. Available: https://www.hindawi.com/journals/jcse/2016/7426913/. [Accessed Mar. 4, 2018].

[10] N. s. P. Hyun, E. I. Verriest and P. A. Vela, "Optimal obstacle avoidance trajectory generation using the root locus principle," In Proc. 2015 54th IEEE Conference on Decision and Control (CDC), 2015, pp. 626-631.

[11] A. Beaty, J. Freeman, R. Jones, “Tactical Unmanned Ground Vehicle,” Georgia Tech School of Electrical and Computer Engineering, Atlanta, Georgia. Tech. Report. 16 Dec. 2016.

[12] The American Society for Photogrammetry & Remote Sensing. (2011). LAS SPECIFICATION VERSION 1.4 – R13.

[13] M. Isenberg, “LAStools: converting, filtering, viewing, processing and compressing LiDAR data in LAS format,” LAStools: award-winning software for rapid LiDAR processing. [Online]. Available: http://cs.unc.edu/~isenburg/lastools/. [Accessed: 11-Dec-2018].

[14] B. Jacob and G. Guennebaud, “Eigen,” *Eigen*. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page. [Accessed: 11-Dec-2018].

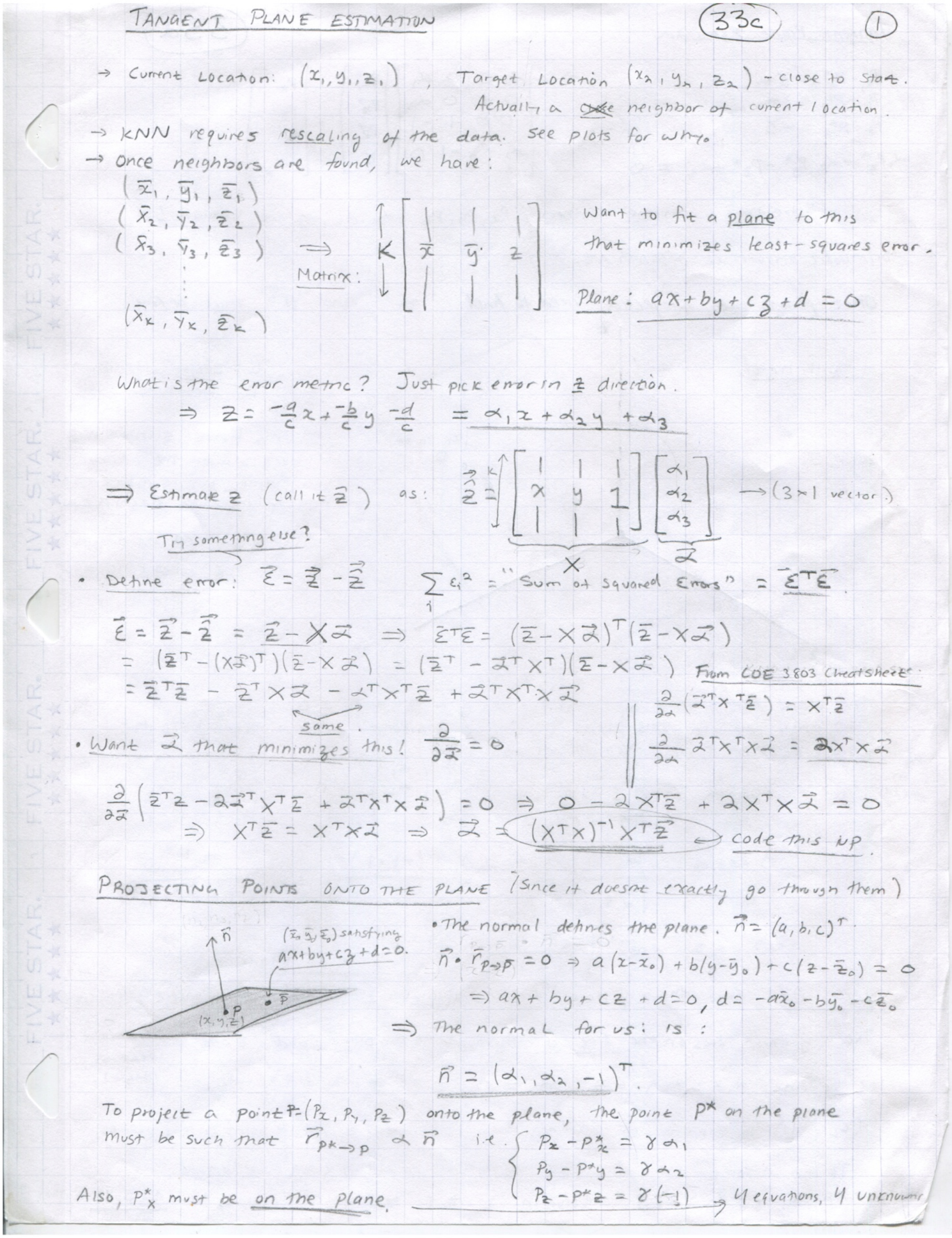
[15] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*. Upper Saddle River: Prentice Hall, 2010. p 95-103

[16] IEEE802.org. (2018). IEEE 802.11, The Working Group Setting the Standards for Wireless LANs.

[17] Omega, “The RS232 Standard” (2018). OMEGA Web Technical Reference

[18] Glassdoor.com. (2018). Salary: Entry Level Software Engineer | Glassdoor.co.uk.

Appendix A (Tangent Plane Calculations, Page 1)



TANGENT PLANE ESTIMATION

33c

1

→ Current Location: (x_1, y_1, z_1) , Target Location (x_2, y_2, z_2) - close to start. Actually a ~~close~~ neighbor of current location.

→ KNN requires rescaling of the data. See plots for why.

→ Once neighbors are found, we have:

$(\bar{x}_1, \bar{y}_1, \bar{z}_1)$
$(\bar{x}_2, \bar{y}_2, \bar{z}_2)$
$(\bar{x}_3, \bar{y}_3, \bar{z}_3)$
$\vdots$
$(\bar{x}_K, \bar{y}_K, \bar{z}_K)$

$\Longrightarrow$ Matrix: $K \updownarrow \begin{bmatrix} | & | & | \\ \bar{x} & \bar{y} & z \\ | & | & | \end{bmatrix}$

Want to fit a plane to this that minimizes least-squares error.

Plane: $ax + by + cz + d = 0$

What is the error metric? Just pick error in z direction.

$$\Rightarrow z = \frac{-a}{c}x + \frac{-b}{c}y \frac{-d}{c} = \alpha_1 x + \alpha_2 y + \alpha_3$$

$\Longrightarrow$ Estimate z (call it $\hat{z}$) as: $\vec{\hat{z}} = K\updownarrow \underbrace{\begin{bmatrix} | & | & | \\ x & y & 1 \\ | & | & | \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}}_{\vec{\alpha}} \longrightarrow$ (3×1 vector.)

Try something else?

- Define error: $\vec{\varepsilon} = \vec{z} - \vec{\hat{z}}$ $\quad \sum_i \varepsilon_i^2 =$ "Sum of squared Errors" $= \vec{\varepsilon}^T \vec{\varepsilon}$.

$$\vec{\varepsilon} = \vec{z} - \vec{\hat{z}} = \vec{z} - X\vec{\alpha} \Rightarrow \vec{\varepsilon}^T\vec{\varepsilon} = (\vec{z} - X\vec{\alpha})^T(\vec{z} - X\vec{\alpha})$$
$$= (\vec{z}^T - (X\vec{\alpha})^T)(\vec{z} - X\vec{\alpha}) = (\vec{z}^T - \vec{\alpha}^T X^T)(\vec{z} - X\vec{\alpha})$$
$$= \vec{z}^T\vec{z} - \underbrace{\vec{z}^T X \vec{\alpha} - \alpha^T X^T \vec{z}}_{\text{same}} + \vec{\alpha}^T X^T X \vec{\alpha}$$

From COE 3803 Cheatsheet:

$$\frac{\partial}{\partial \alpha}(\vec{\alpha}^T X^T \vec{z}) = X^T \vec{z}$$
$$\frac{\partial}{\partial \alpha}\vec{\alpha}^T X^T X \vec{\alpha} = 2X^T X \vec{\alpha}$$

- Want $\vec{\alpha}$ that minimizes this! $\frac{\partial}{\partial \vec{\alpha}} = 0$

$$\frac{\partial}{\partial \vec{\alpha}}\left(\vec{z}^T z - 2\vec{\alpha}^T X^T \vec{z} + \vec{\alpha}^T X^T X \vec{\alpha}\right) = 0 \Rightarrow 0 - 2X^T\vec{z} + 2X^T X\vec{\alpha} = 0$$
$$\Rightarrow X^T\vec{z} = X^T X \vec{\alpha} \Rightarrow \vec{\alpha} = (X^T X)^{-1} X^T \vec{z}$$

code this up.

PROJECTING POINTS ONTO THE PLANE (Since it doesn't exactly go through them)

$\vec{n}$
$(\bar{x}_0, \bar{y}_0, \bar{z}_0)$ satisfying $ax + by + cz + d = 0$.
$\bar{P}$
P
(x, y, z)

- The normal defines the plane. $\vec{n} = (a, b, c)^T$

$$\vec{n} \cdot \vec{r}_{P \to \bar{P}} = 0 \Rightarrow a(x - \bar{x}_0) + b(y - \bar{y}_0) + c(z - \bar{z}_0) = 0$$
$$\Rightarrow ax + by + cz + d = 0, \quad d = -a\bar{x}_0 - b\bar{y}_0 - c\bar{z}_0$$

$\Longrightarrow$ The normal for us: is:

$$\vec{n} = (\alpha_1, \alpha_2, -1)^T.$$

To project a point $P = (P_x, P_y, P_z)$ onto the plane, the point P^* on the plane must be such that $\vec{r}_{P^* \to P} \propto \vec{n}$ i.e. $\begin{cases} P_x - P^*_x = \gamma \alpha_1 \\ P_y - P^*y = \gamma \alpha_2 \\ P_z - P^*z = \gamma(-1) \end{cases}$

Also, P^*_x must be on the plane. $\longrightarrow$ 4 equations, 4 unknowns

Appendix A (Tangent Plane Calculations, Page 2)

Tangent Plane Estimation

33d

②

$$P_x - P_x^* = \gamma\alpha_1$$
$$P_y - P_y^* = \gamma\alpha_2$$
$$P_z - P_z^* = -\gamma$$
$$\alpha_1 P_x^* + \alpha_2 P_y^* - P_z^* + \alpha_3 = 0$$

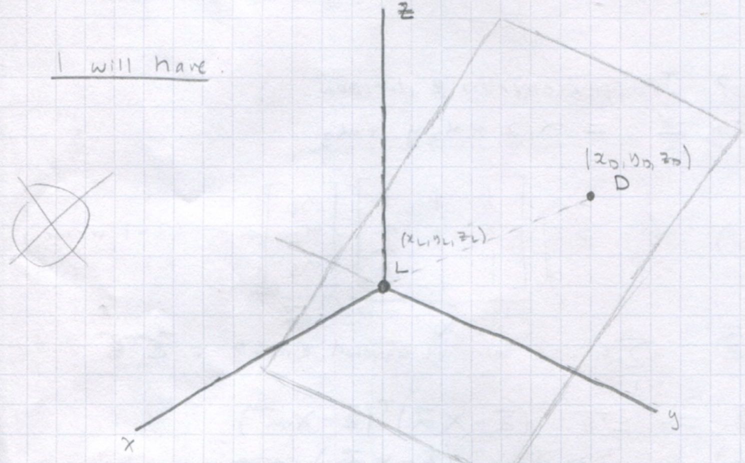
$$\Rightarrow \begin{bmatrix} 1 & 0 & 0 & \alpha_1 \\ 0 & 1 & 0 & \alpha_2 \\ 0 & 0 & 1 & -1 \\ \alpha_1 & \alpha_2 & -1 & 0 \end{bmatrix} \begin{bmatrix} P_x^* \\ P_y^* \\ P_z^* \\ \gamma \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \\ -\alpha_3 \end{bmatrix}$$

Solve this with the numbers. $P_x, P_y, P_z, \alpha_1, \alpha_2, \alpha_3$ are known!

Symbolic answer is in MATLAB.

Once points are projected, need to find "α" and "θ" from before.

I will have:



L = Location (current)
D = Destination

This frame shares a z axis with the Earth frame, but is rotated so its y axis lies in the plane, and its origin is at the robot's location.

The plane is defined by $\alpha_1 x + \alpha_2 y - z + \alpha_3 = 0$.

The angle between this plane and the "ground" (x y plane) is the angle between their normal vectors.

"α" from earlier.

i.e. $\cos\alpha = \dfrac{\vec{n}_1 \cdot \vec{n}_2}{\|\bar{n}_1\| \|\bar{n}_2\|}$ $\quad \bar{n}_1 = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ -1 \end{pmatrix} \quad \bar{n}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$

$$\Rightarrow \cos\alpha = \frac{(\alpha_1 \cdot 0) + (\alpha_2 \cdot 0) + (-1 \cdot 1)}{\sqrt{\alpha_1^2 + \alpha_2^2 + 1}\ (1)} = \frac{-1}{\sqrt{1 + \alpha_1^2 + \alpha_2^2}}$$

$$\Longrightarrow \alpha = \cos^{-1}\left(\frac{-1}{\sqrt{1 + \alpha_1^2 + \alpha_2^2}}\right)$$: Tilt of Plane

- Now, "θ" Since the x y plane drawn is parallel (rotation does not change this (about z)),

The "y axis" lies on the line of intersection of the plane and $z = z_L$ ← The z-value of location. (origin).

i.e. the line $\alpha_1 x + \alpha_2 y - \boxed{z_L} + \alpha_3 = 0$. $= y = \frac{1}{\alpha_2}(z_L - \alpha_1 x - \alpha_3)$

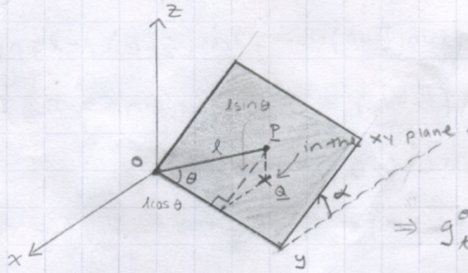
(x_L, y_L) is a point on this line, since (x_L, y_L, z_L) is a point on the plane.

To find a point on the line s.t. the dot product will give an acute angle, check if $\{x_D > x_L\}$, and plug in $x_L \pm 1$ to line accordingly. Now, dot product gives θ

Appendix A (Tilt Estimation Calculations, Page 1)

FORWARD TILT

33a



- $\mathcal{O}$ is the frame drawn.
- A is a frame with its xy plane on the shaded plane, z axis pointing in $+x_{\mathcal{O}}$ direction.
- B is a frame obtained by rotating frame A about its z axis by θ.

$$\Rightarrow g^{\mathcal{O}}_{A} = \left(\vec{0},\ R_y(\alpha)\right),\quad g^{B}_{A} = \left(\vec{0},\ R_z(\theta)\right)$$

- Point $\underline{P}^{B}$ is $(0,1,0)^T$. $\Rightarrow$ $\underline{P}^{\mathcal{O}}$ (changing frames) $= (-l\sin\theta\cos\alpha,\ l\cos\theta,\ l\sin\theta\sin\alpha)^T$
- Point Q is the "shadow" of point P. $\Rightarrow Q = (-l\sin\theta\cos\alpha,\ l\cos\theta,\ 0)$.

- $\bar{r}_{O\to P}$ represents the direction of the robot's motion. We are interested in the angle $\hat{\theta}_f$ between $\bar{r}_{O\to P}$ and $\bar{r}_{O\to Q}$, which I call the forward tilt.

- $\cos\hat{\theta} = \dfrac{\bar{r}_{O\to P}\cdot\bar{r}_{O\to Q}}{\|\bar{r}_{O\to P}\|\,\|\bar{r}_{O\to Q}\|}$ $\Big\|$ $\cdot\ \underline{\bar{r}_{O\to P}\cdot\bar{r}_{O\to Q}} = (-l\sin\theta\cos\alpha)^2 + (l\cos\theta)^2 + \cancelto{0}{(l\sin\theta\sin\alpha)(0)}$
$= l^2\sin^2\theta\cos^2\alpha + l^2\cos^2\theta = l^2(\sin^2\theta\cos^2\alpha + \cos^2\theta)$

Since $\cos^2\alpha + \sin^2\alpha = 1$: $\bar{r}_{O\to P}\cdot\bar{r}_{O\to Q} = l^2\left(\overbrace{\sin^2\theta(1-\sin^2\alpha)+\cos^2\theta}^{(1)}\right) = \underline{l^2(1-\sin^2\alpha\sin^2\theta)}$

- $\underline{\underline{\|\bar{r}_{O\to P}\|}} \equiv l$. $\underline{\text{Verifying}}$: $= \left((-l\sin\theta\cos\alpha)^2 + (l\cos\theta)^2 + (l\sin\theta\sin\alpha)^2\right)^{1/2}$
$= \left(l^2\sin^2\theta\cos^2\alpha + l^2\cos^2\theta + l^2\sin^2\theta\sin^2\alpha\right)^{1/2}$
$= l\left(\sin^2\theta\cos^2\alpha + \cos^2\theta + \sin^2\theta\sin^2\alpha\right)^{1/2} = l\left(\sin^2\theta\cancelto{1}{(\cos^2\alpha+\sin^2\alpha)} + \cos^2\theta\right)^{1/2}$
$= \underline{l}$

- $\underline{\underline{\|\bar{r}_{O\to Q}\|}} = \left((-l\sin\theta\cos\alpha)^2 + (l\cos\theta)^2\right)^{1/2} = \left(l^2\sin^2\theta\cos^2\alpha + l^2\cos^2\theta\right)^{1/2}$
$= l\left(\sin^2\theta\cos^2\alpha + \cos^2\theta\right)^{1/2} = l\left(\sin^2\theta(1-\sin^2\alpha) + \cos^2\theta\right)^{1/2}$
$= \underline{l(1-\sin^2\theta\sin^2\alpha)^{1/2}}$

$\Rightarrow \cos\hat{\theta}_f = \dfrac{\bar{r}_{O\to P}\cdot\bar{r}_{O\to Q}}{\|\bar{r}_{O\to P}\|\,\|\bar{r}_{O\to Q}\|} = \dfrac{\cancel{l^2}(1-\sin^2\alpha\sin^2\theta)}{\cancel{(l)}\left(\cancel{l}(1-\sin^2\theta\sin^2\alpha)^{1/2}\right)}$ (Independent of l ∴ $\underline{\text{sanity check}}$.)

$= \sqrt{1-\sin^2\theta\sin^2\alpha} \Rightarrow \hat{\theta}_f = \cos^{-1}\left(\underbrace{\sqrt{1-\sin^2\theta\sin^2\alpha}}_{\text{"}x\text{"}}\right)$

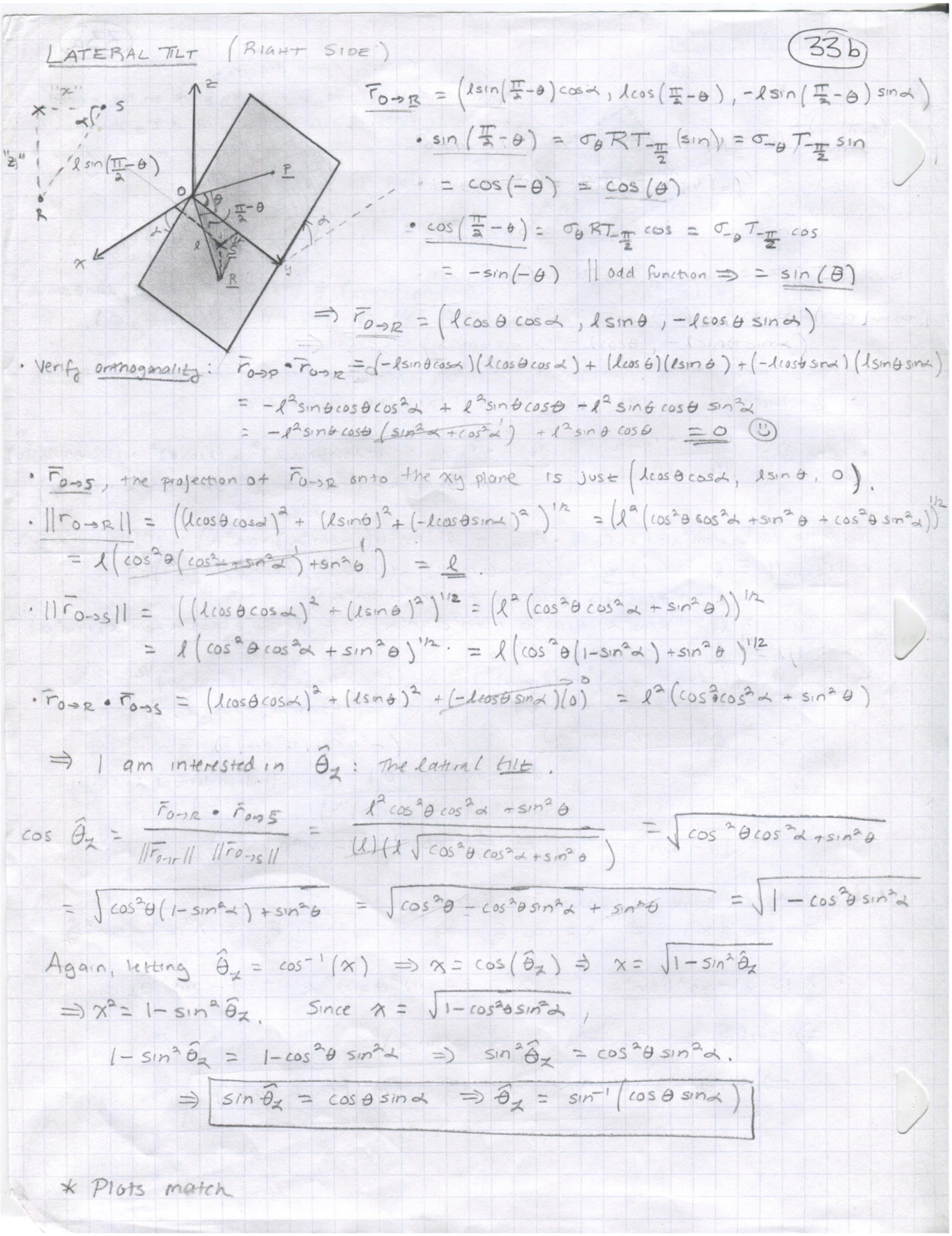
- Using the same substitution as Prof. Verriest:
$\hat{\theta}_f = \cos^{-1}(x) \Rightarrow x = \cos(\hat{\theta}_f) \Rightarrow x = \sqrt{1-\sin^2\hat{\theta}_f} \Rightarrow x^2 = 1 - \sin^2\hat{\theta}_f$.

- $\underline{x \text{ is } \sqrt{1-\sin^2\theta\sin^2\alpha}} \Rightarrow x^2 = 1-\sin^2\theta\sin^2\alpha = 1-\sin^2\hat{\theta}_f$

$\Rightarrow \sin^2\hat{\theta}_f = \sin^2\theta\sin^2\alpha = (\sin\theta\sin\alpha)^2$

$$\Rightarrow \boxed{\sin\hat{\theta}_f = \sin\theta\sin\alpha \Rightarrow \hat{\theta}_f = \sin^{-1}(\sin\theta\sin\alpha)}$$

Appendix A (Tilt Estimation Calculations, Page 2)


"x"
S
α
"z"
$l\sin(\frac{\pi}{2}-\theta)$
R
z
O
P
θ
$\frac{\pi}{2}-\theta$
α
l
S
R
x
y
α

LATERAL TILT (RIGHT SIDE)

33b

$$\bar{r}_{O\to R} = \left(l\sin(\tfrac{\pi}{2}-\theta)\cos\alpha,\ l\cos(\tfrac{\pi}{2}-\theta),\ -l\sin(\tfrac{\pi}{2}-\theta)\sin\alpha\right)$$

- $\underline{\sin(\frac{\pi}{2}-\theta)} = \sigma_\theta R T_{-\frac{\pi}{2}}(\sin) = \sigma_{-\theta} T_{-\frac{\pi}{2}} \sin$
  $= \cos(-\theta) = \underline{\cos(\theta)}$
- $\underline{\cos(\frac{\pi}{2}-\theta)} = \sigma_\theta R T_{-\frac{\pi}{2}} \cos = \sigma_{-\theta} T_{-\frac{\pi}{2}} \cos$
  $= -\sin(-\theta)$ || odd function $\Rightarrow = \underline{\underline{\sin(\theta)}}$

$$\Rightarrow \underline{\bar{r}_{O\to R}} = \left(l\cos\theta\cos\alpha,\ l\sin\theta,\ -l\cos\theta\sin\alpha\right)$$

- Verify orthogonality: $\bar{r}_{O\to P}\cdot\bar{r}_{O\to R} = (-l\sin\theta\cos\alpha)(l\cos\theta\cos\alpha) + (l\cos\theta)(l\sin\theta) + (-l\cos\theta\sin\alpha)(l\sin\theta\sin\alpha)$
  $= -l^2\sin\theta\cos\theta\cos^2\alpha + l^2\sin\theta\cos\theta - l^2\sin\theta\cos\theta\sin^2\alpha$
  $= -l^2\sin\theta\cos\theta\,\overset{1}{\cancel{(\sin^2\alpha+\cos^2\alpha)}} + l^2\sin\theta\cos\theta \ \underline{\underline{=0}}$ ☺

- $\underline{\bar{r}_{O\to S}}$, the projection of $\bar{r}_{O\to R}$ onto the xy plane is just $(l\cos\theta\cos\alpha,\ l\sin\theta,\ 0)$.
- $\underline{\|r_{O\to R}\|} = \left((l\cos\theta\cos\alpha)^2 + (l\sin\theta)^2 + (-l\cos\theta\sin\alpha)^2\right)^{1/2} = \left(l^2(\cos^2\theta\cos^2\alpha + \sin^2\theta + \cos^2\theta\sin^2\alpha)\right)^{1/2}$
  $= l\left(\cos^2\theta\,\overset{1}{\cancel{(\cos^2\alpha+\sin^2\alpha)}} + \sin^2\theta\right)^{\!1} = \underline{l}$.
- $\|\bar{r}_{O\to S}\| = \left((l\cos\theta\cos\alpha)^2 + (l\sin\theta)^2\right)^{1/2} = \left(l^2(\cos^2\theta\cos^2\alpha + \sin^2\theta)\right)^{1/2}$
  $= l(\cos^2\theta\cos^2\alpha + \sin^2\theta)^{1/2} = l\left(\cos^2\theta(1-\sin^2\alpha) + \sin^2\theta\right)^{1/2}$
- $\bar{r}_{O\to R}\cdot\bar{r}_{O\to S} = (l\cos\theta\cos\alpha)^2 + (l\sin\theta)^2 + \overset{0}{\cancel{(-l\cos\theta\sin\alpha)(0)}} = l^2(\cos^2\theta\cos^2\alpha + \sin^2\theta)$

$\Rightarrow$ I am interested in $\hat{\theta}_x$: The lateral tilt.

$$\cos\hat{\theta}_x = \frac{\bar{r}_{O\to R}\cdot\bar{r}_{O\to S}}{\|\bar{r}_{O\to R}\|\ \|\bar{r}_{O\to S}\|} = \frac{l^2\cos^2\theta\cos^2\alpha + \sin^2\theta}{(l)\left(l\sqrt{\cos^2\theta\cos^2\alpha+\sin^2\theta}\right)} = \sqrt{\cos^2\theta\cos^2\alpha + \sin^2\theta}$$

$$= \sqrt{\cos^2\theta(1-\sin^2\alpha)+\sin^2\theta} = \sqrt{\cos^2\theta - \cos^2\theta\sin^2\alpha + \sin^2\theta} = \sqrt{1-\cos^2\theta\sin^2\alpha}$$

Again, letting $\hat{\theta}_x = \cos^{-1}(x) \Rightarrow x = \cos(\hat{\theta}_x) \Rightarrow x = \sqrt{1-\sin^2\hat{\theta}_x}$

$\Rightarrow x^2 = 1-\sin^2\hat{\theta}_x$. Since $x = \sqrt{1-\cos^2\theta\sin^2\alpha}$,

$1-\sin^2\hat{\theta}_x = 1-\cos^2\theta\sin^2\alpha \Rightarrow \sin^2\hat{\theta}_x = \cos^2\theta\sin^2\alpha$.

$$\Rightarrow \boxed{\sin\hat{\theta}_x = \cos\theta\sin\alpha \Rightarrow \hat{\theta}_x = \sin^{-1}(\cos\theta\sin\alpha)}$$

* Plots match

Appendix B (Code Pointers and Implementation Details)

Please note that in order to run NavX's software, the following are required:

- C++ 11 or greater
- Bash shell
- Python 3
- MATLAB R2017a or greater
- g++

Please Note: NavX's code has been tested only on the datasets included in this package. Furthermore, NavX's code has only been tested on macOS and Linux platforms.

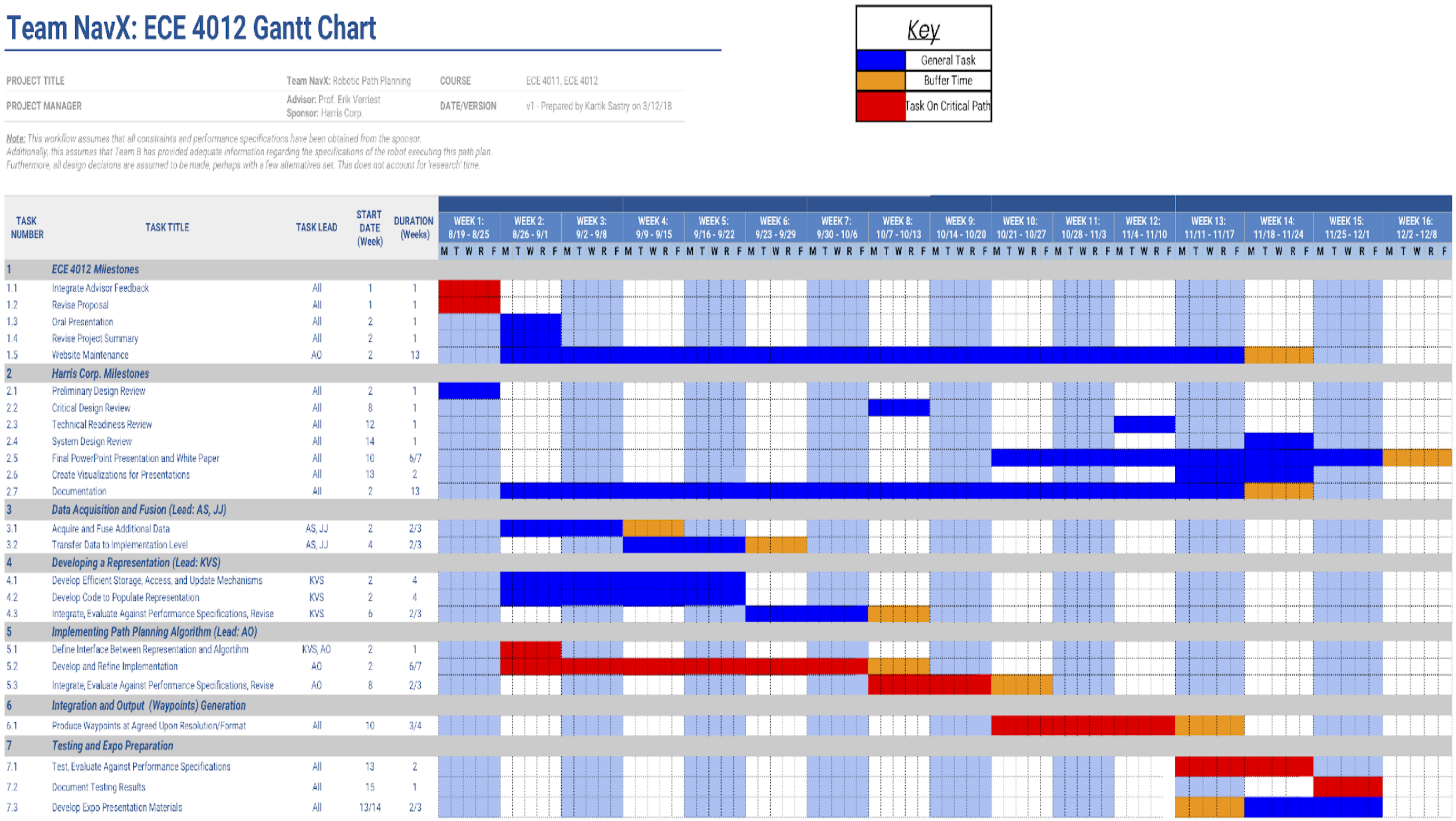
All of NavX's code can be found in the public repository: https://github.gatech.edu/aogarro3/NavX. A map of the repository is given below:

- **src:**
    - **Tilt**:
        - Contains tilt estimation code and MATLAB code for visualizing tangent plane fitting.
    - **Eigen**:
        - A publicly available library for performing Linear Algebra in C++. A subset of this library is utilized by NavX for performing matrix operations
    - **pseudoTestData**:
        - *.csv - CSV files containing artificial point cloud data (with classification).
        - *.fig - MATLAB-generated figures visualizing the output of NavX's path planning software for each of the artificial datasets above.
        - *.html - A compilation of plots and visualization source code.

    - testPseudoData.m – Well-commented source code for visualizing results of NavX's path planning software. Written in MATLAB.
  - **Classifier** - Exploratory work on kNN classification of point terrain type
    - Cython: A library for executing compiled C code in Python
    - *This code is unused in NavX's final solution.*
  - **run.sh** – Well-documented Bash script illustrating how to compile and invoke NavX's software with a target .las file and user parameters.
  - **runToy.sh** - Well-documented Bash script illustrating how to compile and invoke NavX's software with a target .csv file and user parameters.
  - **LAS.h, LAS.cpp -** Class for representing LAS files in C++. Parser code is contained.
  - **Graph.h, Graph.cpp:** Class for representing directed graphs in C++. Representation of points is split between LAS.h and Graph.h. Graph.cpp contains NavX's implementation of the A* search algorithm.
  - **main.cpp –** Well-commented template code demonstrating the necessary sequence of function calls to transform an .las file and user parameters into a sequence of waypoints. Clearly indicates where waypoints can be extracted programmatically. Invoked by run.sh.

- **Data**: LAS files given to NavX by Harris Corporation
  - **isprs-gnd:** contains
    - *.las - Files used as input
    - *.png - Results from kNN classification on each file.
    - *.csv - Path and Graph files produced by NavX's path planning software
    - visRealResults.m - MATLAB code to visualize data and paths generated.

Appendix C (Gantt Chart)



Team NavX: ECE 4012 Gantt Chart

PROJECT TITLE	**Team NavX:** Robotic Path Planning	COURSE	ECE 4011, ECE 4012
PROJECT MANAGER	**Advisor:** Prof. Erik Verriest **Sponsor:** Harris Corp.	DATE/VERSION	v1 - Prepared by Kartik Sastry on 3/12/18

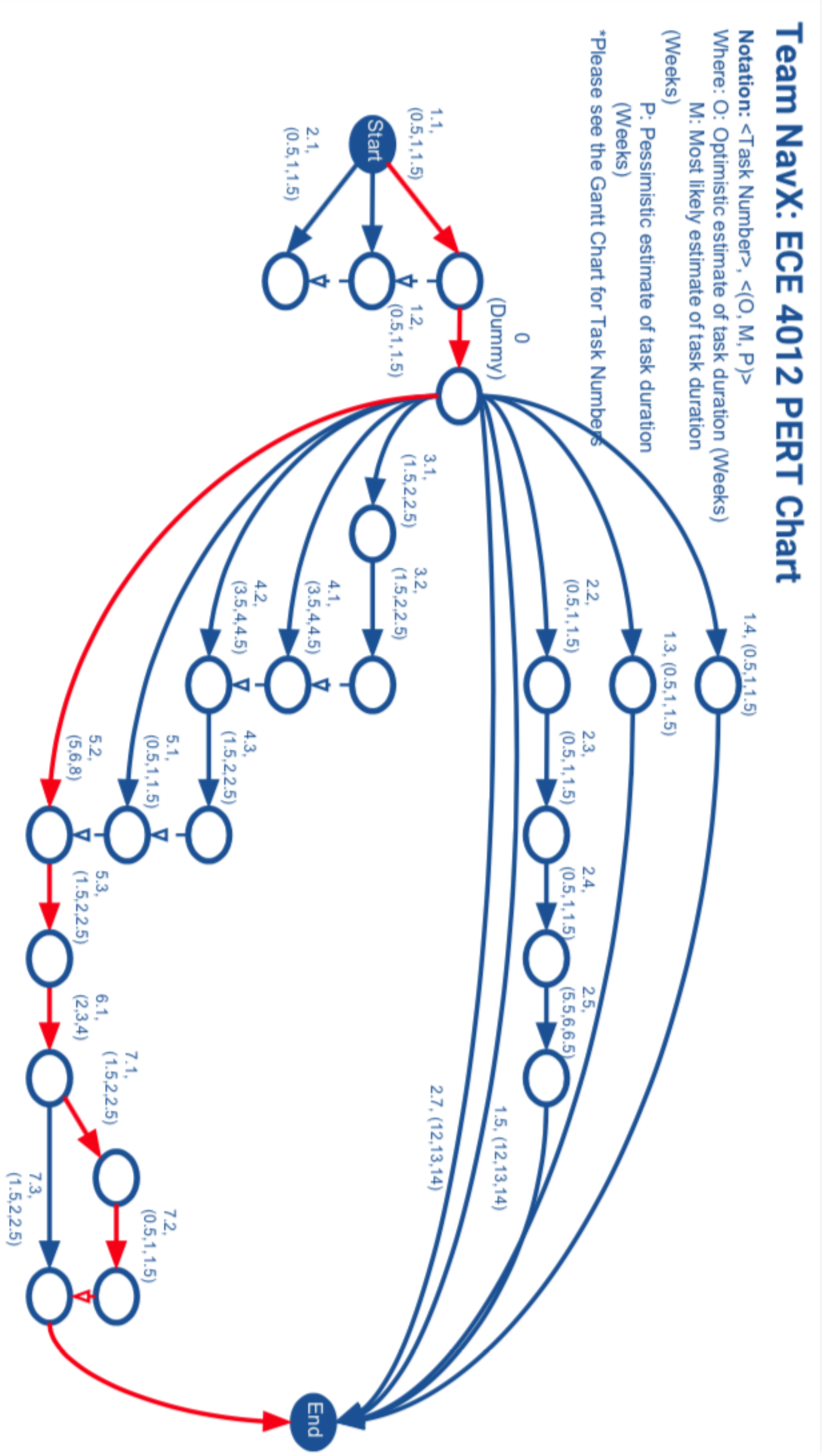
<u>Note:</u> This workflow assumes that all constraints and performance specifications have been obtained from the sponsor. Additionally, this assumes that Team B has provided adequate information regarding the specifications of the robot executing this path plan. Furthermore, all design decisions are assumed to be made, perhaps with a few alternatives set. This does not account for 'research' time.

Key

🟦	General Task
🟧	Buffer Time
🟥	Task On Critical Path

TASK NUMBER	TASK TITLE	TASK LEAD	START DATE (Week)	DURATION (Weeks)
1	*ECE 4012 Milestones*			
1.1	Integrate Advisor Feedback	All	1	1
1.2	Revise Proposal	All	1	1
1.3	Oral Presentation	All	2	1
1.4	Revise Project Summary	All	2	1
1.5	Website Maintenance	AO	2	13
2	*Harris Corp. Milestones*			
2.1	Preliminary Design Review	All	2	1
2.2	Critical Design Review	All	8	1
2.3	Technical Readiness Review	All	12	1
2.4	System Design Review	All	14	1
2.5	Final PowerPoint Presentation and White Paper	All	10	6/7
2.6	Create Visualizations for Presentations	All	13	2
2.7	Documentation	All	2	13
3	*Data Acquisition and Fusion (Lead: AS, JJ)*			
3.1	Acquire and Fuse Additional Data	AS, JJ	2	2/3
3.2	Transfer Data to Implementation Level	AS, JJ	4	2/3
4	*Developing a Representation (Lead: KVS)*			
4.1	Develop Efficient Storage, Access, and Update Mechanisms	KVS	2	4
4.2	Develop Code to Populate Representation	KVS	2	4
4.3	Integrate, Evaluate Against Performance Specifications, Revise	KVS	6	2/3
5	*Implementing Path Planning Algorithm (Lead: AO)*			
5.1	Define Interface Between Representation and Algorithm	KVS, AO	2	1
5.2	Develop and Refine Implementation	AO	2	6/7
5.3	Integrate, Evaluate Against Performance Specifications, Revise	AO	8	2/3
6	*Integration and Output (Waypoints) Generation*			
6.1	Produce Waypoints at Agreed Upon Resolution/Format	All	10	3/4
7	*Testing and Expo Preparation*			
7.1	Test, Evaluate Against Performance Specifications	All	13	2
7.2	Document Testing Results	All	15	1
7.3	Develop Expo Presentation Materials	All	13/14	2/3

Appendix D (PERT Chart)

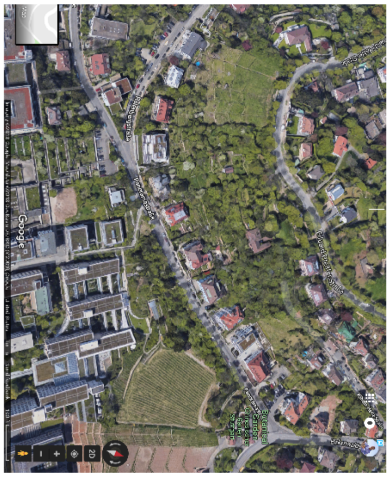


Team NavX: ECE 4012 PERT Chart

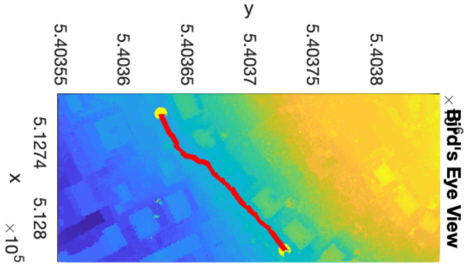
Notation: <Task Number>, <(O, M, P)>
Where: O: Optimistic estimate of task duration (Weeks)
M: Most likely estimate of task duration (Weeks)
P: Pessimistic estimate of task duration (Weeks)

*Please see the Gantt Chart for Task Numbers

Appendix E (Visualization of Results For samp11_gnd.las)

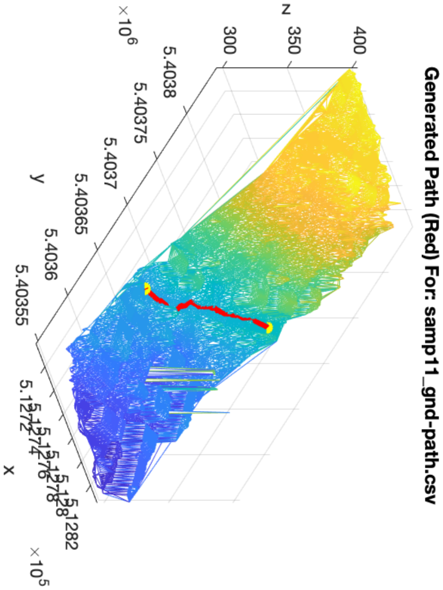


Real-World Location: Stuttgart, Germany - Bird's Eye



Bird's Eye View

Real-World Location: Stuttgart, Germany



Generated Path (Red) For: samp11_gnd-path.csv