Machine Vision: Navigation and Object Recognition in a Mobile Robot

Sri Kankanahalli

Atholton High School

Table of Contents

Abstract

The engineering goals of this project were to create a mobile robotic system capable of automated obstacle avoidance and realtime object recognition. Specifically, the robot should be able to construct a visual model of an object from a set of training images, detect and identify said object within a realtime environment, and navigate safely to the object's position. Several different algorithms, such as Simultaneous Localization and Mapping (SLAM), Point-Line Iterative Corresponding Point (PLICP), Speeded-Up Robust Features (SURF), and linear triangulation were used in order to create and iteratively refine a working prototype of the robot. The robot's navigation capabilities were tested by timing the robot as it planned and followed a path to five predefined goals around the lab area, and its object recognition capabilities were tested by requesting the robot to identify several types of items in five different images. The project is not currently complete, but the robot has at least partially met all of its engineering goals; it is able to consistently and efficiently identify, locate, and navigate to a given object after viewing images of it. This is in stark contrast to the first iteration of the system's first iteration, where none of the goal points were reached due to map corruption.

Machine Vision: Navigation and Object Recognition in a Mobile Robot

For the past half century, mankind has envisioned a future built on the foundations of robotics, and one of the most fundamental areas in this quickly expanding field has been machine vision. The central task of a machine vision system is twofold: first, it receives a description of the sensory world, and second, it attempts to interpret and describe the world's properties algorithmically (Szeliski, 2010). Machine vision has applications in a variety of fields, from medical imaging to space exploration; if realtime robot navigation and vision could be carried out accurately, then many tasks which currently require human supervision could be automated by robots instead. (Palmisano, n.d).

The end goal of this project is to create a machine vision system capable of automated obstacle avoidance and realtime object recognition. Specifically, the robot should be able to construct a visual model of an object from a set of training images, detect and identify said object within a realtime environment, and navigate to the object's position. The robot's setup is relatively limited – its only sensors are a two-dimensional laser scanner and a grayscale 768p camera, and it lacks the three-dimensional mapping, infrared sensors, high camera resolution, and processing power of higher-end robot projects. However, despite these restrictions, significant advancements have been made in the robot's capabilities.

In order to successfully navigate and avoid obstacles in realtime, a robot must have accurate knowledge of where it is relative to the world around it; it must use the data received from its sensors to continuously update an estimate of its position. This process is known as robot odometry. A common approach is to detect changes in the robot's wheel motors and use these to determine its velocity over time; however, this method is imprecise and can lead to large inaccuracies as the robot moves further from its starting point (Anderson, 2010). Any error in odometry also leads to errors in environment mapping; these can stack up and essentially corrupt the robot's map, making navigation impossible.

A more stable and accurate approach to position estimation is provided by an algorithm known as Point-Line Iterative Corresponding Point (PLICP.) PLICP is a general algorithm to find a transformation minimizing the distance between two sets of data points. Using PLICP, the translation and rotation between subsequent sets of the robot's laser scan data can be found in realtime (Censi, 2008). This results in many small changes of the robot's position and angle being computed quickly and iteratively, creating a much more accurate odometry estimate than that of the wheel motors. The implementation of the PLICP algorithm is provided by the Robot Operating System (ROS), an advanced open-source software framework which aids in the development of robotic applications (“ROS wiki,” 2011).

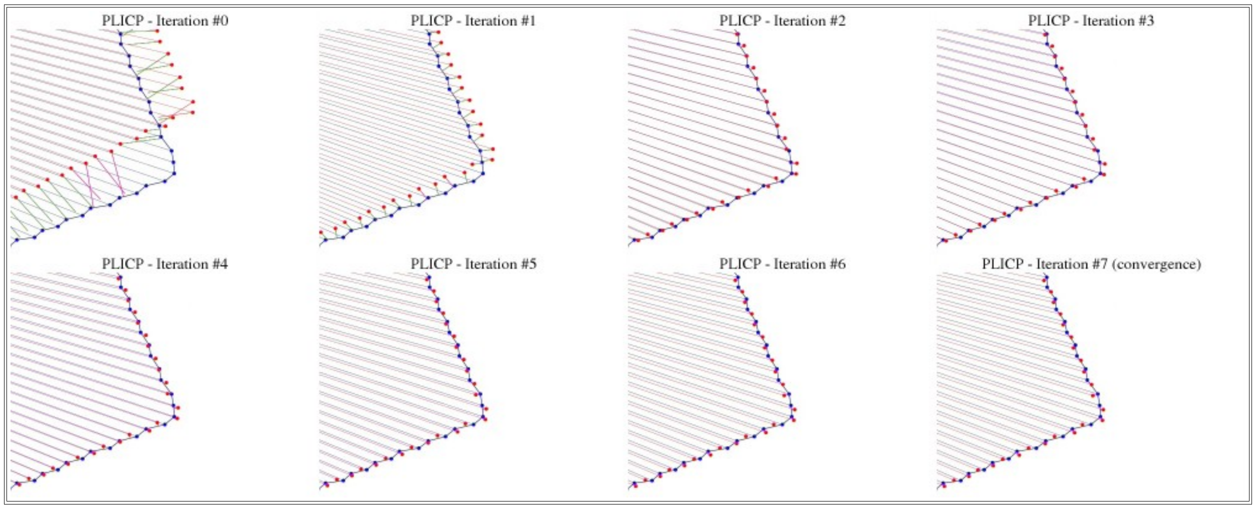

Figure 1: PLICP algorithm finding a near-optimal transformation between the red and blue data sets in 7 iterations (Censi, 2008).

Once it has obtained an accurate position estimate, the robot uses an algorithm known as Simultaneous Localization And Mapping (SLAM) to create and update a map of its surroundings. SLAM is defined as “the problem of building a map while at the same time localizing the robot within that map," and these two goals are deeply dependent on each other; the robot needs to know its own location in order to build an accurate map from laser scans, but at the same time a map is also extremely helpful in adjusting the robot's position (Stachniss, Frese, & Grisetti, 2011). The specific SLAM variant used is GMapping, a sophisticated approach which provides an extremely accurate estimation of the robot's surroundings based on the robot's current odometry, past positions, and laser scan data received over time. GMapping uses a technique known as particle filtering to compute probabilities and sequentially improve its map estimate, adapting over time in order to increase speed and keep computational requirements as low as possible (Grisetti, Stachniss, & Burgard, 2006). The implementation of the GMapping algorithm used in this project is provided by the ROS codebase as well.

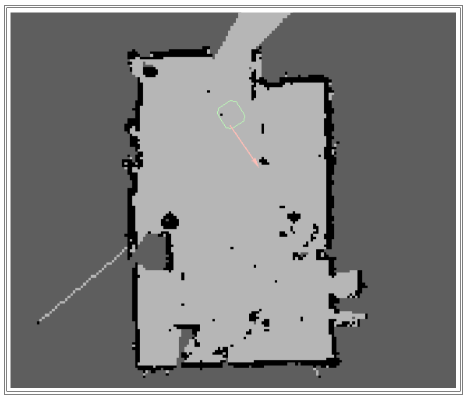

Figure 2. Sample GMapping output of a cluttered lab area. Light gray areas indicate unoccupied space, black areas indicate occupied space, and dark gray areas indicate unexplored space.

Another helpful feature ROS provides is a path planner which can calculate a route between any two positions on the map. An ROS path is stored as a simple sequence of positions ("nav_msgs/Path documentation", 2012), and the robot can follow this planned path easily just by navigating in a straight line from one given position to the next. This also means that paths can easily be recalculated at any time; if the robot strays too far off or encounters an obstacle in its way then it can simply request a new path and change course immediately. This results in a dynamic system which can navigate to any preset goal.

However, a truly autonomous robot acts without human input; the ideal is that there is no need for preset goals at all. In other words, the robot should be able to explore its environment and fill gaps in its knowledge without any human guidance. This can be accomplished using a relatively simple but efficient technique known as frontier-based exploration. Boundaries on the map between unoccupied space and unknown space are detected, and large boundary areas are marked as high priority "frontiers" for the robot to visit (Yamauchi, 1997). Once the robot reaches a frontier area, it performs a 360 degree sweep with the laser scanner and adds the information to its map. (If the robot finds that it has no way of navigating to a particular frontier, then the space is simply marked as occupied.) This results in an effective navigation and exploration system which constantly searches out and integrates new information about the world into its knowledge base without any need for external input – the robot is not only able to map its environment but also use that map to explore it.

The current approach to object recognition involves a two-step process of image filtering followed by feature detection and matching. First, a bilateral filter is applied to the image in order to reduce noise and only retain the most relevant features possible (Fisher, 2011). Then, simple contrast adjustments are made in order to make the image's features appear stronger.

Feature detection and matching are carried out using a popular algorithm known as Speeded-Up Robust Features (SURF.) SURF is designed to find distinctive feature points within an image that do not vary with scale or rotation (corners and edges, for example), and it uses these points to compute robust sets of feature descriptors that can be matched with relative ease (Bay, Ess, Tuytelaars, & Gool, 2008). The Erratic robot can use SURF to find an object within an image by comparing the object's feature descriptors to the image's descriptors; the accuracy of this match depends on the certainty of the matches and the total number of points found.

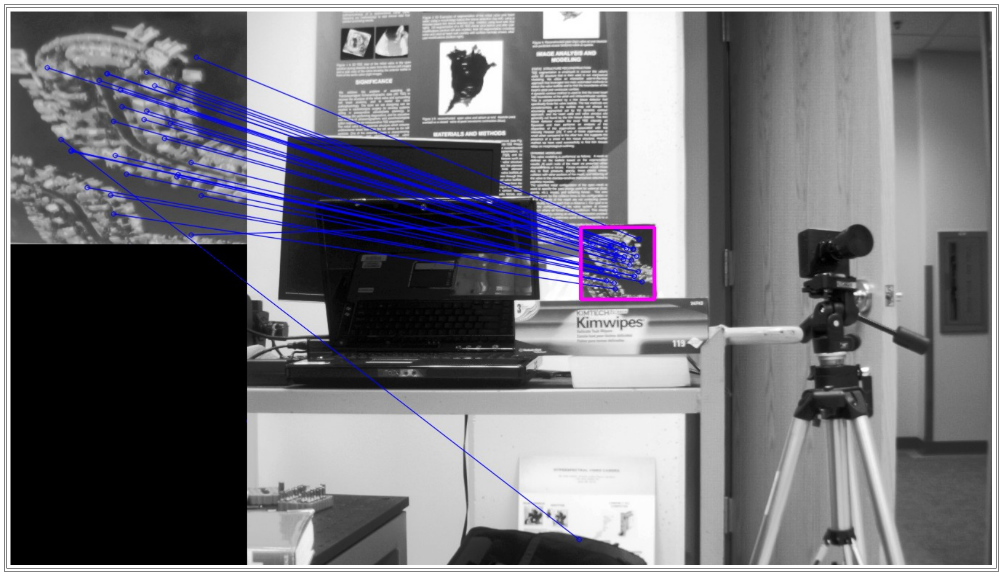

Figure 3. Sample SURF feature matching. Blue lines indicate match points, and the magenta rectangle indicates the approximate rectangular region of the object within the image.

After an object has been identified using SURF and its identity has been verified over several different frames, there still remains the matter of locating the object's position in 3D space. This is where linear triangulation and camera geometry come in. The fundamental concept is that any given camera has a 3x4 projection matrix P, containing both intrinsic parameters such as focal length or center point and extrinsic parameters such as camera translation or rotation, which relates the projected 2D point x and the real 3D point X in the following manner: (Hartley & Zissermann, 2004)

$$\mathbf{x} = \mathbf{PX}$$

Using basic cross product rules, this equation becomes $\mathbf{x} \times (\mathbf{PX}) = \mathbf{0}$. Evaluating the cross product yields the following 3 equations, where P_i indicates row i of the projection matrix P:

$$\mathbf{y(P_3X) - P_2X = 0}$$

$$\mathbf{P_1X - x(P_3X) = 0}$$

$$\mathbf{x(P_2X) - y(P_1X) = 0}$$

These can be rewritten as:

$$\mathbf{X(yP_3 - P_2) = 0}$$

$$\mathbf{X(xP_2 - P_1) = 0}$$

$$\mathbf{X(xP_2 - yP_1) = 0}$$

Combining the top two equations for both points yields a 4x4 matrix A, which can be used in the equation **AX = 0** to solve for X. This is the basic linear triangulation method; though more advanced ones exist, it is still fairly effective for most scenarios.

Materials and Methods

The materials needed for this project are a Videre Erratic-brand mobile robot with a 768p grayscale camera and a Hokuyo URG-04LX-UG01 laser scanner attached. In addition, the project requires a working computer (preferably running Ubuntu Linux) with the following software fully installed and configured:

- OpenCV 2.3.1 (used for image processing and object recognition)
- Boost 1.48 (used by OpenCV)
- Robot Operating System (ROS), ver. Electric (used for robot navigation and interfacing – not needed if running object recognition system in simulation)

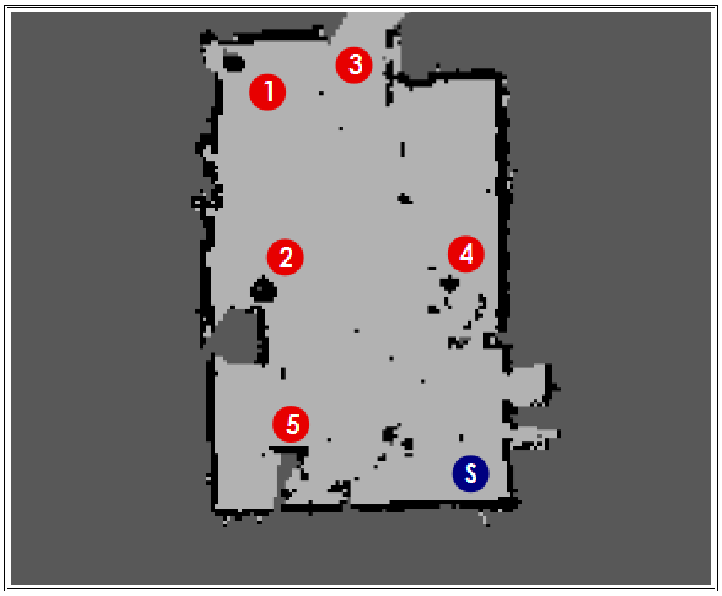

Figure 4. Map of testing process for the robot's navigation system. Red circles indicate predefined goal points, blue circle indicates start point.

The robot's navigation system was tested as follows:

1. A map of the lab area was created, either by driving the robot around manually or letting it explore autonomously.
2. The robot was instructed to find and follow a path to several predefined goals around the lab area.
3. The robot's behavior was observed and recorded.
4. Steps 2 and 3 were repeated for several trials.
5. Based on the observations recorded, adjustments and changes were made to the robot's codebase.

The robot's object recognition system was tested as follows, within a simulation mode (almost no experimental testing has been done within the real lab setting as of this point:)

1. Photo images were downloaded from the 2009 Semantic Robot Vision Challenge

database (DeMenthon & Rybski, n.d). 5 images were picked out, each corresponding to one of the following five objects: Goldfish crackers, bag of Lays potato chips, “I Am A Strange Loop” book by Douglas Hofstader, “Photoshop in a Nutshell” book, pumpkin.

2. For each object, a training image was found on the Internet manually.
3. The robot was given the task of detecting whether each object was visible within its corresponding image, and drawing a rectangle around the object (if detected.)
4. The robot's accuracy in identifying the object was recorded, as well as the quality of the rectangle it drew. Quality was separated into three categories: “good,” “okay,” and “bad” where a good rectangle covers over 75% of the object, an okay rectangle covers 75% or less of it, and a bad rectangle covers 25% or less.
5. Step 3 was repeated for 3 trials.
6. Based on the observations recorded, adjustments and changes were made to the robot's codebase.

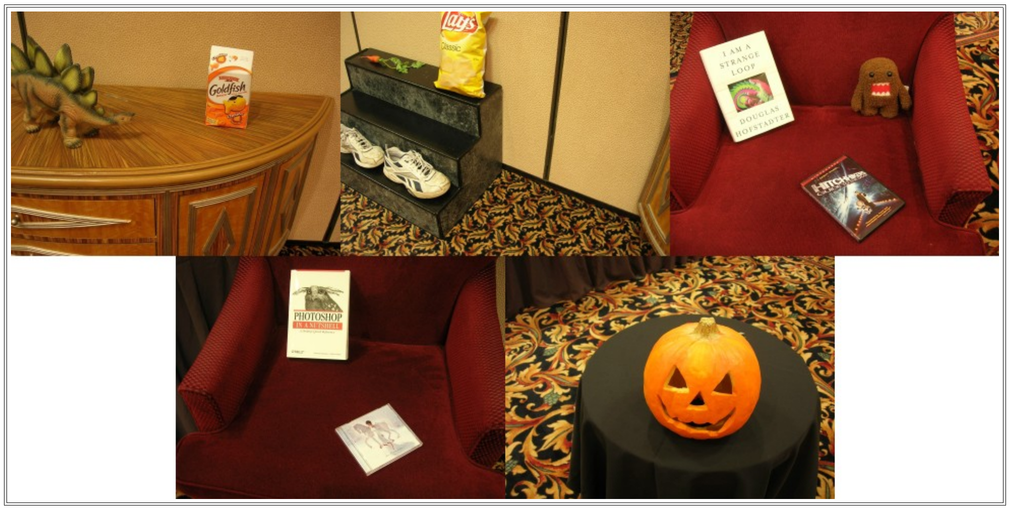

Figure 5. The 5 images used for testing the robot's object recognition capabilities. One item to identify per image. From top-left to bottom-right: Goldfish crackers, bag of Lays potato chips, *I*

Am A Strange Loop book, *Photoshop in a Nutshell* book, pumpkin.

Results

Table 1

Erratic Robot's Navigation System over Several Iterations

Trial	Iteration	Point 1	Point 2	Point 3	Point 4	Point 5
Trial 1	**Iteration 1 (original)**	Off track; map corruption	Off track; map corruption	Off track; map corruption	Off track; map corruption	Off track; map corruption
	Iteration 2 (PLICP)	Off track; path drift	Off track; path drift	Off track; path drift	Off track; path drift	Reached goal in 1:47
	Iteration 3 (path f.)	Reached goal in 0:43	Reached goal in 0:34	Reached goal in 0:49	Reached goal in 0:48	Reached goal in 0:21
Trial 2	**Iteration 1 (original)**	Off track; map corruption	Off track; map corruption	Off track; map corruption	Off track; map corruption	Off track; map corruption
	Iteration 2 (PLICP)	Off track; path drift	Reached goal in 2:11	Off track; path drift	Off track; path drift	Reached goal in 1:20
	Iteration 3 (path f.)	Reached goal in 0:41	Reached goal in 0:31	Reached goal in 0:52	Reached goal in 0:41	Reached goal in 0:26
Trial 3	**Iteration 1 (original)**	Off track; map corruption	Off track; map corruption	Off track; map corruption	Off track; map corruption	Off track; map corruption
	Iteration 2 (PLICP)	Off track; path drift	Off track; path drift	Off track; path drift	Off track; path drift	Reached goal in 2:31
	Iteration 3 (path f.)	Reached goal in 0:38	Reached goal in 0:48	Reached goal in 0:44	Reached goal in 0:46	Reached goal in 0:28

Table 2

Erratic Robot's Machine Vision System over Several Iterations

Trial	Iteration	Goldfish crackers	Lays chips bag	"Strange Loop" book	"Nutshell" book	Pumpkin
Trial 1	**Iteration 1 (original)**	Recognized; bad rect	Not recognized	Recognized; okay rect	Recognized; good rect	Not recognized

	Iteration 2 (filtering)	Recognized; good rect	Recognized; bad rect	Recognized; okay rect	Recognized; good rect	Not recognized
Trial 2	**Iteration 1 (original)**	Recognized; bad rect	Not recognized	Recognized; okay rect	Recognized; okay rect	Not recognized
	Iteration 2 (filtering)	Recognized; good rect	Recognized; bad rect	Recognized; good rect	Recognized; good rect	Not recognized
Trial 3	**Iteration 1 (original)**	Recognized; bad rect	Not recognized	Recognized; okay rect	Recognized; okay rect	Not recognized
	Iteration 2 (filtering)	Recognized; good rect	Recognized; bad rect	Recognized; good rect	Recognized; okay rect	Not recognized

Discussion

During the first two iterations of the robot navigation system, many of the trials failed due to corrupted map data and path drift. The former problem was due to inaccurate robot odometry – as the robot moved around, the error between its estimated position and its actual one increased greatly over time. This resulted in map corruption, rendering the robot unable to navigate properly – it did not reach any of the preset goals within the first iteration of the navigation system. This disappeared completely once the PLICP algorithm was introduced. The path drift within the next iteration was due to the default ROS path follower not controlling the Erratic robot properly; this resulted in the robot either not reaching its goal at all or taking an extremely roundabout and inefficient way to its goal. This was solved by writing original path following code, containing failsafes so that the robot would automatically recalculate its trajectory if it detected that it was too far off the path (or if it encountered an obstacle in the middle of its path.) At this point, the robot's navigation capabilities improved dramatically, and it was able to get to all of the given preset goals in a reasonable amount of time (less than a minute for every goal.) There may still be some room for improvement regarding path optimization, but this is currently lower priority.

Between the first two iterations of the image recognition system, the robot showed the most improvement in recognizing the Goldfish crackers (from a “bad” rectangle to a “good” rectangle.) The system also showed some minor improvement in recognizing the “Photoshop in a Nutshell” and “I Am A Strange Loop” books. The least improvement was seen with the pumpkin and the Lays potato chips; the former was not recognized even once in either the first or second iteration of the system, and the latter was only identified with a “bad” rectangle at most.

The two books and the Goldfish crackers bag were probably recognized relatively easily due to their concentration of interesting feature points, as well as their lack of a noisy background. For example, both books have detailed designs on their cover (including fairly large serif text), and the Goldfish bag has many changes in regional color intensity (e.g. the borders between the orange and white areas, as well as the “Goldfish” font.) In contrast, the only detailed design on the Lays bag is the logo itself (a relatively small portion of the whole object), and the noisy patterns on the room's floor serve as a large distractor to the feature detection algorithm.

Part of the failure to recognize the pumpkin may also be attributed to the room's noisy floor designs (which are even more prominent in its image than in the Lays image), but it also points to a larger problem with the image recognition system – it only works with specific objects, not general categories. That is, asking the system to recognize a specific book or brand logo will generally yield positive results, but asking it to recognize a general class of object (such as a “pumpkin” or a “laptop”) will not. This is because the training set for each object is currently extremely narrow; only one image is being used to train the robot, which is not nearly enough to create a generalized visual model of an object class.

This is one area where further advancements for this project lie; an general object model could be constructed by clustering together common features from many individual training images. If done correctly, this would not only lead to improvements in the general case but also the specific one, since the object recognition system would have an even more nuanced set of features. Another important further advancement would be connecting the navigation and object recognition systems together, so the robot is able to determine the position of an object and navigate to it once it identifies one. Yet another future goal would be to fully automate the robot's training; instead of using carefully human-selected training images, the robot could search and filter images from the Internet on its own without any external assistance. Finally, if speed considerations allow it, a mixed approach could be applied to the object recognition; in addition to SURF, other types of object detection algorithms could be used in order to provide verification and narrow down possible locations for the object.

Acknowledgments

The author wishes to thank his mentor, Dr. Daniel DeMenthon, and his internship site, the Johns Hopkins University Applied Physics Laboratory, for providing him with this fantastic opportunity and making this all possible. In addition, the author would also like to thank his teacher Mrs. Natalie Kelly for all the support, encouragement, and advice that she has given him.

References

Anderson, D. (2010, August 21). *IMU odometry*. Retrieved from http://www.geology.smu.edu/dpa-www/robo/Encoder/imu_odo/

Bay, H., Ess, A., Tuytelaars, T., & Gool, L. V. (2008). SURF: Speeded up robust features. *Computer Vision and Image Understanding, 110*(3), 346-359. Retrieved from http://www.vision.ee.ethz.ch/~surf/eccv06.pdf

Censi, A. (2008, May 23). An ICP variant using a point-to-line metric. In *2008 IEEE International Conference on Robotics and Automation*. Retrieved from http://www.cds.caltech.edu/~ender/pub/research/2008-icra-plicp.pdf

DeMenthon, D., & Rybski, P. (n.d.). *The semantic robot vision challenge*. Retrieved from http://www.semantic-robot-vision-challenge.org/

Fisher, R. (2011). *Bilateral filtering*. Retrieved from School of Informatics, University of Edinburgh website: http://homepages.inf.ed.ac.uk/rbf/CVonlineLOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html

Grisetti, G., Stachniss, C., & Burgard, W. (2006). Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*. Retrieved from http://www.informatik.uni-freiburg.de/~stachnis/pdf/grisetti06tro.pdf

Hartley, R., & Zissermann, A. (2004). *Multiple View Geometry in Computer Vision* (2nd ed.).

nav_msgs/Path documentation [Online documentation]. (2012, January 12). Retrieved from Willow Garage website: http://www.ros.org/doc/api/nav_msgs/html/msg/Path.html

Palmisano, J. S. (n.d.). *Programming - computer vision tutorial*. Retrieved from Society of Robots website:

http://www.societyofrobots.com/programming_computer_vision_tutorial.shtml

ROS wiki [Wiki documentation]. (n.d.). Retrieved from Willow Garage website: http://www.ros.org/wiki/

Stachniss, C., Frese, U., & Grisetti, G. (2011, December). *OpenSLAM*. Retrieved from http://openslam.org/

Szeliski, R. (2010). *Computer vision: algorithms and applications*. Retrieved from http://szeliski.org/Book/

Yamauchi, B. (1997, July 11). A frontier-based approach for autonomous exploration. *IEEE International Symposium on Computational Intelligence in Robotics and Automation*. Retrieved from http://www.robotfrontier.com/papers/cira97.pdf