Quercus: Linfield Journal of Undergraduate Research

Volume 1 Article 1

2012

Simultaneous Localization and Mapping Using Stereoscopic Computer Vision for Autonomous Vehicles

Robert Ferrese
Linfield College

Simultaneous Localization and Mapping Using Stereoscopic Computer Vision for Autonomous Vehicles

Acknowledgements

- Professor Dan Ford - Computer Science Department - Physics Department - Professor Crosser - Linfield Student Faculty Collaborative Grant

ABSTRACT

Simultaneous Localization and Mapping Using
Stereoscopic Computer Vision for Autonomous Vehicles

Creating a robotic system capable of autonomously making decisions based on the environment must first be capable of visualizing and recording its surroundings. This paper focuses on creating a robotic system that makes use of stereoscopic imaging from two offset camera feeds to create a 3D image. The robot is able to generate a 3D image from the cameras and convert the 3D images into a digital map. Furthermore, the robot is able use this model to detect its location within the digital map for navigational and mapping purposes. This process of simultaneously localizing and mapping (SLAM) is a revolutionary procedure used to generate maps in real time. By combining these three aspects, the robotic system can generate an accurate 3D map for the region of operation. Stereoscopic imaging provides many benefits over conventional mapping methods which allow for cheap, rapid, and detailed autonomous mapping. This paper demonstrates how a low cost robotic system can independently generate a 3D map in real time.

Introduction

To successfully map an area using a robot, the robot must be able to measure distances, create a digital map, and localize within the map. To accomplish this, the first task for the robot is to be able to detect its surroundings. Without a way of measuring distances, the robot would be unable to create a map. The second task for the robot is to be able to record this map digitally so that it can be referred to later and updated as new data is collected. Finally, the robot must be able to find its location within the map it has created. The process of localization is vital to the the success of the project because the robot must know where it is before it can know what to do with the distance measurements it collects. After the robot determines its location within its map, the robot can update the map, add new data, and correct mistakes made within the digital map. The mapping and localizing can be combined into Simultaneous Localization and Mapping (SLAM) where the robot will create a map as it navigates an unknown region. While this procedure could be replaced by a human operator, there are many applications which would make human operation infeasible such as unmanned missions to space, search and rescue operations, or future robotics applications where the robot would need to act on its own. For the robot to be successful, it should be able to map in real-time, be cheap, and be modular. Furthermore, the robot should rely primarily on its vision sensor readings for all operations, with basic wheel rotation readings to measure movement feedback. By creating an autonomous robot that is able to map out its surroundings with vision sensors alone, this thesis illustrates the power of stereo vision in robotics.

Technology

The robotic system used to autonomously map a region consists of the vision sensors, the processing unit, and the locomotion system. These components work together with the custom algorithms to allow the robot to autonomously map, localize, and navigate its surroundings. Figure 1 shows the robot and a matching diagram of the components. The processing unit used for this project is a small onboard computer that processes the imaging data and controls the wheel system.

There are several different data collection methods available for sampling distances to walls and obstructions for robotic systems. The method that is used in this project is stereo vision because of its potential for data collection. Stereo vision uses the difference in perspectives between two cameras separated by a fixed distance to resolve objects, similar to human sight. Examples of artificial stereoscopy can be seen as far back as 1840, when two slightly different images were created and viewed with glasses to create the illusion of a three dimensional image[1]. More recently, NASA has created a STEREO project[2] which consists of two satellites that orbit around the earth to create 3D maps of the sun and other solar phenomena. Stereo vision works by taking pictures simultaneously with two offset cameras, detecting the same object in the two separated cameras, and measuring the pixel offset. The pixel offset (disparity) can be converted into a distance map where each pixel will represent the distance to the object which that pixel depicts. Stereo vision encompasses both vertical and horizontal resolution, which allows for objects above and below the robot to be mapped. Another positive aspect of this project is the use of small cameras: small cameras are less costly than other technologies. By mounting two cameras with a known separation the distance to features of the images can be determined. Thus, using two images, the distance to each pixel can be calculated simultaneously. However an alternative to using cameras are laser rangefinders.

Infrared laser rangefinders shine a laser at the target and measure the reflection to determine the distance. They have an accuracy in the centimeter range for distances approx-

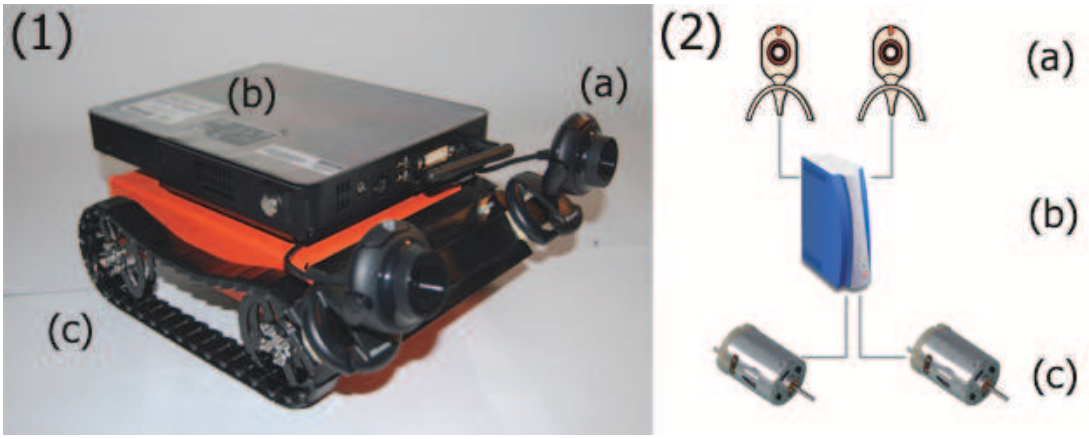

Figure 1: 1) Photograph and 2) diagram of the robotic setup. The webcams (a) connect to a small computer (b). The computer processes the image data and generated motion control signals. These signals drive the motors wheel encoders (c)

imately up to 20 meters but do not have a large field of view. The measurement is made by measuring the reflection of a laser, so the sensor will only be able to detect an object directly in the path of the laser. By combining many infrared sensors, a detailed map could be generated. However, combining sensors is costly and size restrictive. A large number of sensors would be needed to create a detailed description of the robot's surroundings. For our project, using laser rangefinders would be the best alternative to using stereo vision if stereo vision were not available. In recent years, this technology has been advanced into what is called a scanning laser rangefinder.

Scanning laser rangefinders use a single laser rangefinder with a rotating mirror to generate a scanning line that samples every half a degree to make a map. Unfortunately, scanning laser rangefinders only generate distances in the plane of the laser scan, so there is no vertical resolution, which is a limiting factor for this project. Furthermore, scanning laser rangefinders are expensive (starting at over five thousand dollars for a basic unit), making them cost-prohibitive for our project, as well. Scanning rangefinders have been used in many autonomous vehicle projects, including the DARPA car challenges[3] where autonomous cars were programmed to navigate urban environments.

Lastly, ultrasonic rangefinders send pulses of ultrasonic sound out and listen for the

reflection. Although they provide broad coverage, they are ineffective for distances over a few meters. Also, they have a wide angle of "vision" per measurement meaning that they cannot be combined to create a high-resolution map. For this reason, they are the least suited for this mapping application.

Implementation

The robotic system consists of three main components. First, the vision sensors (Figure 1 (a)), collect all of the imaging data used. Second, there is the data processing system, which is responsible for analyzing the image data and for generating motion control data. Finally, the robot must have a motor controller to translate the motion control data into motor movement.

For this project, webcams were used for the image collection because they are affordable, robust, and easily acquired. The webcams connect directly to the onboard computer via USB, and are mounted 10 inches apart to create a usable depth resolution from 3 to 30 feet.

The webcams and the motor controller are controlled by the onboard computer system (Figure 1 (b)). This computer is responsible for converting the 2D imaging data into 3D images and implementing the SLAM algorithm using this vision data. By combining the 3D images into a 3D map, the robot can then generate a desired navigational trajectory for the robot. Finally, this motion data must then be sent to the motor controller.

The motor controller uses digital signals corresponding to desired velocities, and translates them into analog signals that drive the motors. The robot is placed upon caterpillar tracks, where the motion is controlled by the motors. In this project, caterpillar tracks (Figure 1 (b)) are used because they provide unpredictable movement due to the treads slipping. This aspect is useful because the goal of the project is to navigate using vision alone. If accurate wheels were used, the algorithm would rely on that accuracy in movement, and thus, it would not test the limits of the SLAM algorithm. The motors are also connected to wheel encoders which measure the wheel rotations. This data is used for speed control and

in the SLAM algorithm make the robots guess of its location more accurate. The motors will output different speeds given different loads, so a Proportional, Integra, and Differential (PID) controller is created to account for any errors between actual motor speeds and the desired speeds.

Background

Stereoscopy

To create a 3D image, the robot takes advantage of the parallax effect. The parallax effect, depicted in Figure 2, shows that two offset vision sensors, such as our eyes, see things with slightly different perspectives. This change in perspective means objects will appear in relatively different locations for each camera depending on the locations each camera's location. The robot uses parallax and combines the images from two cameras seperated by a known distance into a single map of their differences, called a disparity map. Figure 3 shows how two images can be combined into a single disparity map. This map can then be converted into a distance map by knowing the specifications of the camera lens used to take the picture (like angle of view and angle per pixel across the viewable area).

When two cameras are offset from one another and then used to take a picture of the same subject, objects closer to the cameras will be seen in comparatively different locations on the image, while objects farther away will be closer to the same place. This is similar to observing the landscape when driving a car. The trees that are close appear to be going by much faster than the mountain in the background. Similarly, with two offset cameras the difference in the object's position will depend on its distance from the camera. The distance between the same objects in two images is called disparity. An object that is very close to the cameras will have a large disparity. The goal of stereoscopy is to create this disparity map. From the map, the data can be converted into actual distances with simple calibrations. Four

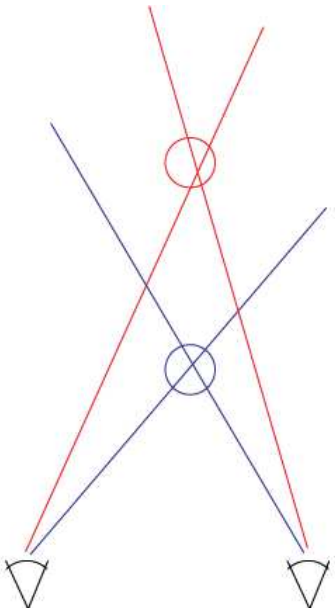

Figure 2: The parallax effect. The left and right eye look at two objects (red and blue) which appear in different relative locations for each eye.

stereoscopic methods were tested to pick the best candidate for autonomous mapping. In this project, the cameras will be offset horizontally, however, using vertically offset cameras would require very little modification to the algorithms.

The first method tested, Bitwise Image Comparison, simply compares the color of each pixel. Each pixel was compared to corresponding pixels in the same row to see where they matched best. The location where they matched was the disparity for that pixel. The second method, Edge Map Comparison, expanded on the first and decreased the amount of errors by only comparing the edges. By using a Canny Filter, the edges the only pixels with color, and thus there are fewer possibilities for matching. The third method, Vector Edge Map Comparison, builds upon the first two methods by converting the Canny Filtered image into a list of line vectors using a Hough Filter. This allows even faster processing because it again decreases the amount of data. The final method, OpenCV, takes the procedures from all of the other options but also makes educated guesses of the disparity where there are no well defined edges.

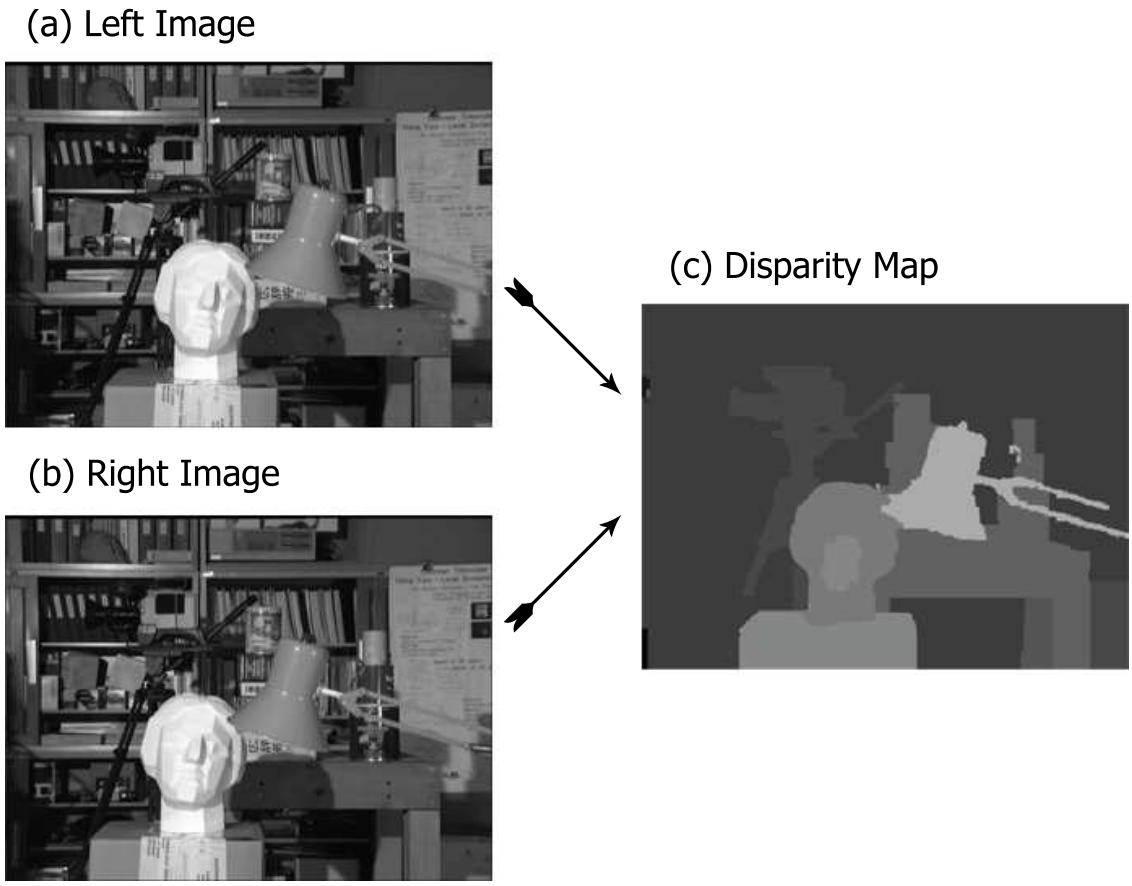

Figure 3: Stereoscopic vision and the creation of the disparity map. (a) and (b) show images from two horizontally offset cameras. Disparity map (c) displays computed differences in an object from two images. Light areas represent large pixel offsets, while dark areas represent regions which have very little offset. The sample images were provided by OpenCV. The disparity map was created using the FindStereoCorrespondence method from OpenCV[4].

SLAM

To map an area, a robot must be able to do two things: determine its location within the existing map and add to the map with newly collected data. For example, if a robot knows all about the room it is in, it must be able to go into a hallway and extend its map to include the hallway. By adding onto the map in this fashion, the robot is able to create a detailed map of a large area without any knowledge of its overall structure. This process, called Simultaneous Localization And Mapping (SLAM), is used by many robotic systems to build up maps of their surroundings without needing any initial hard-coded maps built

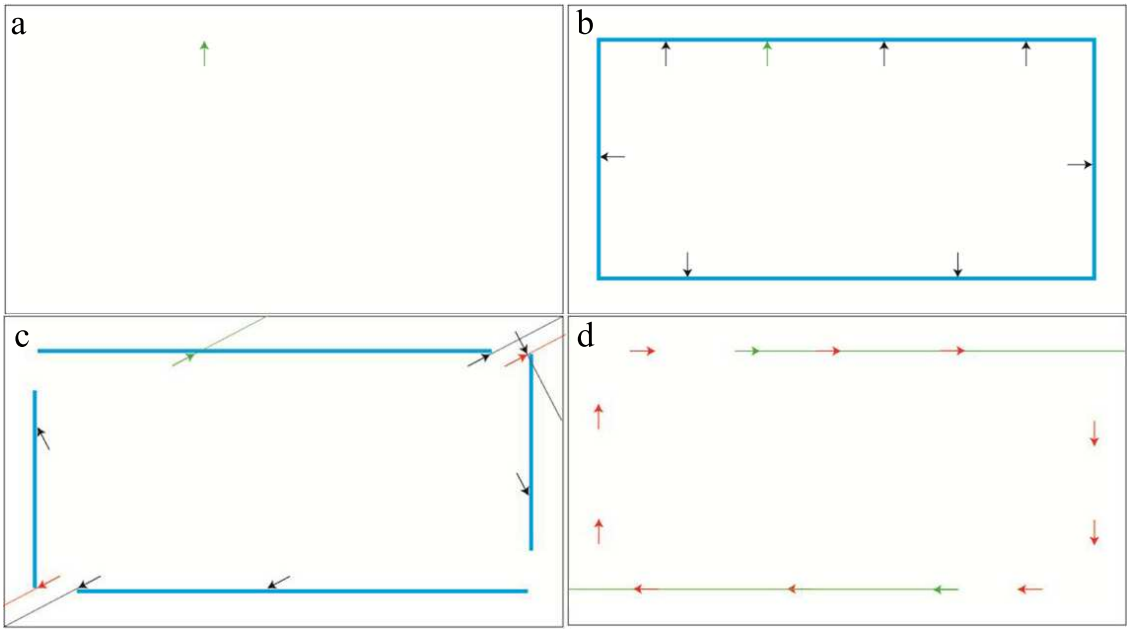

Figure 4: An example of localization. The robot (the green arrow) cannot differentiate between possible locations (black arrows) until it starts to rotate and get different perspectives on the room. As the robot rotates, the guesses of the robots location conflict with sensor data and are removed (red arrows) until the robot is left with two possible locations (green arrows in frame d).

in.

The first step to the SLAM algorithm is localization. It is important for the robot to know exactly where it is so it knows where in the map to place the new data. For example, if the robot sees a table, but is unsure of where in the room the robot is, then the robot cannot add the location of the table to the map with any accuracy. There are two main ways that the robot can find its location. It can either search the map systematically or make many guesses and statistically narrow them down.

The statistical method used in this project follows the general procedure shown in the example in figure 4 where a robot attempts to localize in a symmetric rectangular room. The robot is initially one meter away from the wall in Figure 4(a). In this example, the robot has one sensor which measures the distance directly in front of the robot. Thus, the robot in this example would see a reading of one meter. Because the robot knows at this point only that there is a wall 1 meter from itself, the robot could be in any position along the blue

line in 4(b). However, as the robot turns, some of the guesses of the robot's position (black arrows) would show different measurements if the robot were actually there. These guesses would have a low confidence and are thus denoted as a red arrow because the robot cannot be in that position and orientation. As the robot turns farther (from frames c to d) all but two of the guesses are removed because the robot is confident it cannot be in those locations. Because the room is symmetrical, the robot cannot actually determine its position further so this is as close as the algorithm can get to determining the robots actual location.

Experiment

Stereoscopy

Several methods are available to combine two images into a 3D image. The algorithms aim to identify similar objects in the two cameras and measure the distance between them. Each algorithm takes a different approach to identifying similar objects. Once the objects are identified, the distance between the objects can be converted into a distance measurement from the robot to the pixels comprising that object.

Bitwise Image Comparison

The first method of distance map creation is a simple bitwise matching of images. The images slide over each other one pixel at a time, and each pixel is compared with its corresponding pixel in the other image. Then, by comparing how well each pixel matched with the other tests, the best match can be recorded for that pixel's disparity.

The bitwise image comparison method is the simplest to implement because each pixel is evaluated independently based on its color. However, because two neighboring pixels of the same object could match to two completely different objects this procedure contains a high potential for error due to the independent comparisons. Because no data has been removed, each pixel must be tested against many other pixels, creating a very time-consuming process. Furthermore, this procedure has a difficult time detecting matches unless the colors of the pixels are identical which is not always the case due to dynamic white balance and exposure

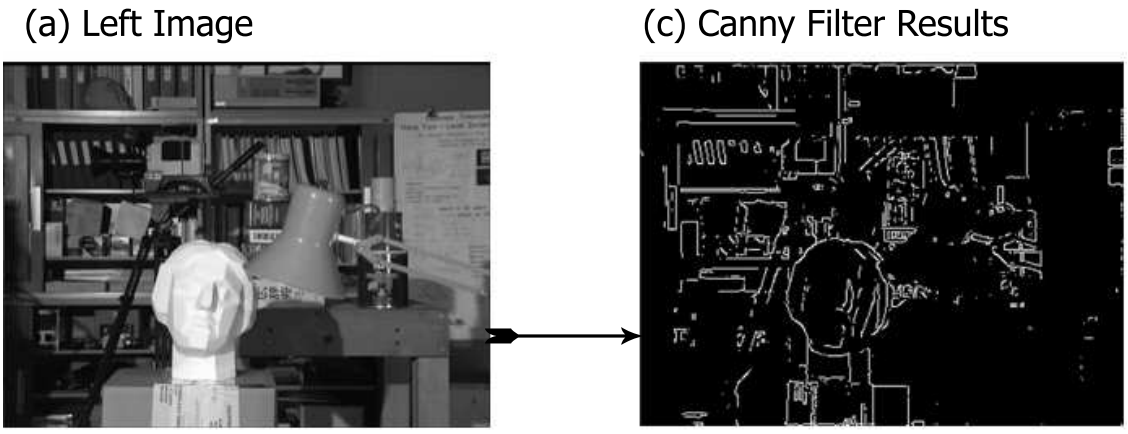

Figure 5: Using the Canny Filter detects edges in images. This illustration shows two images before and after a Canny Filter was applied. The output of the Canny Filter is either a black or white pixel. White pixels represent an edge.

settings as well as differences in perspective. In addition, large blocks of the same color cannot be differentiated because each pixel will match with the whole block.

Edge Map Comparison

The second method is to convert the image into an "edge map" using a Canny Filter (Shown in Figure 5). The Canny Filter can be applied to an image in order to create a second image of the same size. With this method, the second image will have a white pixel if there is an edge and a black pixel if there is no edge. By varying the parameters the Canny Filter will output a different number of edges. Then by using a similar algorithm to the bitwise image comparison to compare the edge maps, a more accurate disparity map can be created for the edges of objects. If an image has well-defined edges, then the lines will show up in the edge map. Furthermore, if there are less lines, it is less likely to mistake one line for another and create an incorrect disparity.

Bitwise edge mapping has similar speed problems as the bitwise image matching because the same number of comparisons are made. Even though edge maps only have two colors, unlike the millions of colors found in the original images, the process is still slow. This method

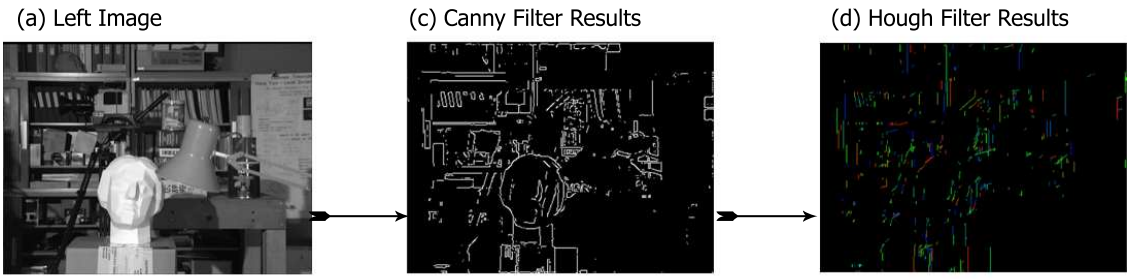

Figure 6: Using the Hough filter detects edges in an image and converts them into vector line segments. The Hough Filter takes an image, applys a Canny Filter on the image to find the edges and outputs a list of line segments as shows in frame (d). Each line segment is assigned a different random color to differentiate between different segments.

does, however, solve the problem of having a high probability of a mismatch because only the most essential data is preserved - the edges. Because the rest of the data does not provide good points of comparison, keeping only the edges is a logical way to solve the problem of false positives. This problem is not completely alleviated though because if multiple edges exist close together, then there is no way to determine the best match. Because the edges are points of comparison in this method, preserving the colors at the edges would be useless. Edges are detected based on high contrast regions, and thus picking colors to match edges with would be impossible. Furthermore, the images only have disparities for the edges, so there is only a partial disparity map.

Vector Edge Map Comparison

A third method is to find the lines by employing a Canny Filter and then converting the lines into a list of line segments. This technique is useful because it will be valid only to compare lines with similar slopes together for the disparity map. Thus, the list of comparisons will be shorter because only similar line segments need to be compared. The Hough Filter is used to convert the output of the Canny Filter to a list of line segments as shown in Figure 6.

The vector edge map is able to quickly generate a list of line segments from the Canny Filter's output, however it can only detect perfectly straight lines. If a line is slightly curved,

it will detect many short line segments which make up the curve. In real life, the lens of the cameras is not perfect, causing warping which will make lines appear slightly curved. Furthermore, if the robot were to map an area which is not composed of square walls and obstacles, it would be difficult for the robot to perform stereoscopy. Thus Vector Edge Map Comparison does not provide a broad enough application for this thesis project[5].

OpenCV

The last stereoscopic method tested was a function provided with the "EMGU" distribution of OpenCV[6]. This function, called FindStereoCorrespondence, took two grayscale images and created a disparity map[4] by detecting similar objects in each picture. The OpenCV method was slow for large images, but it sped up significantly when smaller image sizes were used. The function used the two images and a Canny Filter on each image to generate an edge map for each image. The algorithm then used a procedure similar to the bitwise edge comparison described; however, it took into account all neighboring pixels as well. The two reasons this was possible while running as fast as the other methods was because its algorithm was more efficently designed and because the library was written in a much faster programming language. After finding matching points, the algorithm interpolated the data to generate regions of each distance. This allowed not only point distances to be calculated, but it also generated distances automatically where there was no usable data. The output of the FindStereoCorrespondence function is shown in Figure 3.

The function not only not only finds the pixel offset at the edges of each object (similar to the edge matching or Hough Filter), but also able to detect objects inside the images and interpolate the disparity. This process works by using the offset at the edges surrounding the blank region to make an educated guess about the offset in the blank regions of the image. The ability to interpolate the disparity in regions with little discernable data is an improvement over the previous two methods because those methods simply found the disparity at the edges. Since the distances are known all over the image, it will be simple to

simulate this action for testing and to simulate what a robot would see if it were at any given position in the map. Unfortunately, a negative aspect is that this process is slower than the other methods outlined. However, it provides excellent data interpolation and is able to provide the highest fault tolerance for images that are blurry and distorted. Furthermore, the output at very low resolutions is useable, so faster processing times can be achieved.

SLAM

Localization

To determine the location of the robot, the program implemented a statistical model which allowed the robot to increase the accuracty of its location over time as opposed to brute force search for its location which is a time-consuming process. The statistical method for finding a robot's position is called Particle Filtering which is derived from the Monte Carlo method. There are other methods for determining a robot's location (like the Kalman Filters[7]), but the Particle Filter allows multiple guesses of the robot's location to be remembered while the robot finds more details about its surroundings to keep or eliminate the guesses[8]. For example, if the robot's map contains two identical rooms and the robot is in one of them, there is no way to determine which room it is in. If the robot leaves the room and notices the hallway is different on each, it can then remove one of the guesses. In a particle filter, hundreds of guesses are made as to the robot's location. These guesses are called "Particles".

Even the best guess for the robot's location will not be good unless it accounts for the robot's movements. This application will have a moving robot and thus, each particle is moved the same amount that the robot moved in each iteration of the algorithm. Each particle is moved forward and turned the same amount that the robots wheel sensors measure. However, to account for some error that is caused by wheel slipping and sensor error into the program, a small amount of random error is added to each particles movement. This way slight drift in sensor data can be overcome by causing the particles to drift along with

the errors.

Next, the particles are ranked based on the likelihood that their location is actually the real robot's location. Since each particle has a location and a heading, the map the robot has in memory can be used to simulate what the robot would see if it were at the particle's location. As an example, a particle facing a wall would see a wall directly in front of it. If the robot has a window in front of it, this guess would not be good because the real sensor data and what the particle "sees" are different.

Now that the robot knows which of the guesses are good and which are bad, the robot can then start to refine where it is located. Because only a couple hundred particles are ever used, the initial random distribution of particles will not provide a location with enough accuracy. Therefore, the particles must be redistributed and retested. The methods with which these particles are redistributed determines how the particle filter behaves and if it will converge on the actual robot's location. As the robot refines its actual location in the map, the particles begin to cluster around likely locations of the robot in the map. Eventually, all guess clusters but one will be removed out due to conflicting data. At this point, the robot knows where it is and needs to use localization only to correct for wheel errors and to check periodically if the robot is actually wrong about its location. Because the particle filter is slow and processing time is valuable, decreasing the number of particles once the robot has found its location allows the program to run at a higher frame rate, and thus run more smoothly, during normal operation[9].

Table 1: Particle filter procedure

1	Randomly distribute particles and rate the particles
2	Redistribute particles based on last rating
3	Move particles based on robot's actual movement
4	Rate particles
5	Repeat from step 2

The step that makes each particle filter unique is the way the particles are redistributed. There are three different methods that were used in this program to redistribute the particles

and refines the robot's location. All of the particles were first sorted by their order through a coarse sorting which introduces errors through the random number generator used. This allows the general order of the sorting to be from low confidence to high confidence particles, but allows particles to be shifted out of their exact place in the list and to perform different redistribution methods. Then, the algorithm went through each particle choose between the three methods of redistribution. If the particle was in the bottom 40% of the list, the particle was removed and placed back into the map based on the positioning of other particles. Particles of low confidence were cloned from particles of high confidence. The high confidence particles were selected from the top 20% of the list. After cloning the particle, some error was intentionally added based on the confidence of the original cloned particle. When the cloned particle was very confident, the randomness was small, but when there was less confidence, then a larger amount of error was added to the new particle. If the particles were in the next 10% of the list, it was picked up and replaced randomly. The remainder of particles were left where they were. This was important if the particle's confidence was not high enough to be cloned, but not low enough to be redistributed. Leaving particles allowed the particle to move into better alignment through future iterations, which could create a better guess later in the mapping. By using these three techniques, the particles are able to generate better robot location guesses, refine their location once a probable location is found, and account for error in the robot's movement.

Mapping

To map out an area, the data was stored in a data structure. The way that the data was stored dictated the performance of the program. In this project, the data was stored in an array. This split space into small blocks of a certain size where the block can either be an object or be empty. For this project, a block size 1cm $\times$ 1cm $\times$ 1cm was found to be the limit for processing and memory. By changing the size of each discrete block, the resolution of the map was changed and the performance followed. When there were small blocks, a lot

of data was processed, but large blocks caused the resolution of the map to be poor.

In each block, the number of times something was seen in that block was recorded as well as the number of times something was not seen in that block as a confidence that the block was filled or not filled. That is, if the robot was looking at an area and nothing was seen there, then the "nothing" confidence increased. This allowed a dynamic environment such as an infrequently used room to be mapped accurately. For example, sometimes the robot would see a person standing in the room. However, the majority of times the robot would see no one. Thus, the number of times the robot saw no one in the room overpowered the infrequent visitor. This choice also allowed errors to be recorded into the map without causing difficulties. By allowing errors to be corrected, the map was more robust and allowed the map to change with a changing environment. Another addition was to allow older memories to fade from the map. When a new addition was made to the map, what was there before could be faded such that if the area suddenly had a new object in it, the memory of not having that object before faded with time and was replaced with the new data.[10].

Results

3D Stereoscopy

As explained in the Exploration of 3D Methods section, there are four methods that were tested to generate disparity mappings: bitwise matching, bitwise edge matching, vector Hough Filter matching, and lastly, an OpenCV algorithm. While each method has its own advantages, one of the most critical factors is the speed with which the algorithm can process the data and the quality of the output. This section evaluates the speed and scalability of the algorithms as well as the output quality.

The first two algorithms tested were a bitwise comparison of images and edge maps. One of the most detrimental factors for these methods was that they needed to compare each pixel in the images with every pixel in the corresponding row to find a match. With image sizes of approximately 0.3 mega pixels and 70 levels of disparity on the full size images, the output of the function required over 24 million pixel comparisons. These tests were both unusably slow and provided poor data output for full resolution images. Low resolution images were faster, but had even worse output. Because edge and image matching are the same general procedure, they have been grouped together into a general "bitwise comparison" category. Table 2 shows the data from bitwise comparisons. Each smaller image is a reduced copy of the original. The rescaling processes took a 4 pixel square and averaged the pixels into one pixel thus creating an image with half the vertical and horizontal resolution. Each test was with a set number of disparity tests. Each disparity test offests the images by that that

number of pixels and tests which parts (if any) match. For example, if an algorithm was tested for only 2 disparities, it would test a 0 pixel offset and a 1 pixel offset and determine the disparity for each pixel within the set of tests.

Table 2: Bitwise comparison data. The test was conducted using 3 different parameter combinations as well as three images sizes. The number of disparities is the number of image offsets tested for matching objects. The original image was taken directly from the Web Cams. The results are averaged over 10 trials.

Test	Disparities	Average Time	Standard Deviation
Image Size: 640 x 480	70	8,669 ms	327 ms
Image Size: 320 x 240	35	1,091 ms	184 ms
Image Size: 160 x 120	17	141 ms	31 ms

In the edge comparison, the images must also undergo a Canny Filter to get from the raw images to the edge images. The Canny Filter is fast in comparison to the bitwise comparisons (around 200 times faster) so it can be ignored. Furthermore, the two input parameters for the Canny Filter can be varied with little change in processing time. Table 3 shows the processing time data for the Canny Filters, averaged over 10 samples, is shown.

Table 3: Canny Filter data. The results are averaged over 10 trials.

Test	Input Parameters	Avg. Time	Std. Deviation
Image Size: 640 x 480	Gaussian Size: 0, Threshold: 0	25.8 ms	6.9 ms
Image Size: 640 x 480	Gaussian Size: 100, Threshold: 100	22.7 ms	4.5 ms
Image Size: 640 x 480	Gaussian Size: 255, Threshold: 255	17.9 ms	5.1 ms
Image Size: 320 x 240	Gaussian Size: 0, Threshold: 0	5.5 ms	0.8 ms
Image Size: 320 x 240	Gaussian Size: 100, Threshold: 100	3.1 ms	0.5 ms
Image Size: 320 x 240	Gaussian Size: 255, Threshold: 255	5.5 ms	0.7 ms
Image Size: 160 x 120	Gaussian Size: 0, Threshold: 0	1.6 ms	0.1 ms
Image Size: 160 x 120	Gaussian Size: 100, Threshold: 100	0.8 ms	0.2 ms
Image Size: 160 x 120	Gaussian Size: 255, Threshold: 255	0.8 ms	0.1 ms

To reduce this lengthy processing time, the images can have a Hough Filter applied to them to create vector lines. These vector lines can be compared with each other to generate a similar output to the bitwise edge comparison, but the comparisons are much faster. By finding corresponding edge lines, the offset of the lines will be the disparity of all edge pixels. In bitwise comparison the disparity was determined by discovering which pixels matched.

Therefore, the output of the two should be similar. However, because edge vectors are represented as two end points (and inherently a slope) the slopes just have to be sorted and compared, which quickly generates a set of matching edge candidates. The limiting factor of this procedure is the Hough Filter itself. The Hough Filter is very fast but is unable to generate useful data output unless the lines of the image are perfectly straight. Table 4 shows the timing results for the Hough Filter (Figure 6).

Table 4: Hough Filter data with Canny parameters of 100 and 100. The results are averaged over 10 trials.

Test	Average Time	Standard Deviation
Image Size: 640 x 480	868 ms	112 ms
Image Size: 320 x 240	365 ms	56 ms
Image Size: 160 x 120	163 ms	12 ms

The imaging library OpenCV contains a function that takes two images and converts them into a disparity map. This function is the slowest of all the functions. However, because it incorporates image grouping, data interpolation, and dynamic filtering, it is the best option of all methods explored. Furthermore, with the OpenCV function small images, which are much faster to process, provide usefull data unlike the bitwise comparison which does not generate an accurate output at smaller image sizes. These attributes make the output of the function much more usable when trying to implement the SLAM algorithm, and thus, are preferred over functions which are simply faster. In addition, as the image resolution is decreased, the processing time significantly decreases. Even at these lower resolutions of images, the output disparity is still clear enough to discern different objects and to be processed for use in the SLAM algorithm. Table 5 shows the processing times for different image sizes for the OpenCV process (Figure 5).

Table 5: OpenCV data. The results are averaged over 10 trials.

Test	Disparities	Average Time	Standard Deviation
Image Size: 640 x 480	70	52,072 ms	1201 ms
Image Size: 320 x 240	35	5,470 ms	213 ms
Image Size: 160 x 120	17	627 ms	85 ms

Based off of the spacing between the cameras as well as their field of view, an equation was formed to calculate the distance to a detected object. Equation 1 shows the equation used to calculate the distance to a detected object. By inputting the disparity of the object in pixels, the equation will show the distance in centimeters to the object. This equation was tuned using the minimum image size of 160x120 pixels and a maximum disparity of 17 pixels. The resolution of the model is shown in Equation 2 which is the derivative of Equation 1. On average, the resolution for objects in the middle of the robots field of view will be approximately 10 centimeters.

$$\rho = 284 * e^{-0.077*\delta} \tag{1}$$

$$\rho = 21.868 * 0.9259^{\delta} \tag{2}$$

SLAM

While localization is the most important aspect of robotic navigation, it would not be possible without a map. The generation of the robot's map is simple. Once the location of the robot is found, simply take the disparity map generated by the two cameras and project the image into the 3D map. Then the map is updated to reflect the new data from the cameras. However, this process is the most simplistic approach and it does not account for errors or initialization. If the map has an error, the algorithm should have a mechanism for correcting this error. This is done in the mapping algorithm. When the robot identifies an obstacle in front of it, it increases the confidence in the map to indicate there is an obstacle while simultaneously decreasing the probability there are any obstacles between the robot and the observed obstacle. By recording the confidence of an obstacle instead of recording whether there is an obstacle or not, the map can account for a dynamic environment, as well as allow erroneous measurements to be recorded and removed without causing any problems to the

navigation and operation.

Figure 7 show the SLAM algorithm being used to map out a room. The robot was manually turned 45 degrees for each iteration of the algorithm. After each turn, the robot ran the OpenCV algorithm to generate a 3D image which was added to the map. As the robot rotated 360 degrees, it added to the map to generate an initial map for the room. After completing this procedure, the robot will be able to navigate the room to get different perspectives of the room to further build the map.

A limiting factor, along with processing power to complete the required calculations, was computer memory. The data storage for the large maps consumed more then 500 Megabytes of data, pushing the computer over its memory limit. To solve this problem, the robot's internal map was restricted to $100\times100\times30$ blocks. However, by creating multiple maps with the same dimensions and storing them in memory, the map size limit could be increased[11].

The most vital part of autonomous navigation is the process of localization. A particle filter was implemented in order to determine the robot's location within a map. The particle filter takes the imaging data and the inaccurate wheel data and outputs an estimate of the robot's location. By updating the particle filter frequently (at least once every second), the robot is able to narrow down on its actual location and account for any inaccuracies in the motion data. However, due to processing limitations, this frequency had to be reduced to once every two to three seconds. The basic particle filter algorithm has been modified to fit our application. First, the less confident particles are redistributed around the more confident particles. By redistributing the particles, the particles will begin to cluster faster than if the particles were simply randomly distributed. However, a small portion of the particles are always randomly distributed so a false positive match on the robot's location can be reversed. Lastly, the number of particles should vary depending on the confidence of the robot's location. This will allow improved performance once the robot's location has been found, and it will create a more smooth operation for the robot's navigation. After mapping out the room in Figure 7, the robot's localization was reset so it did not know

Figure 7: Real life mapping using stereoscopic imaging. The images show the left and right cameras and their combined disparity map. Robot manually rotated around fixed point to scan room and generate initial map. The last image is a top-down image of the robot's internal map. Filled pixels represent blocks (compiled over all height layers of the map) that are filled. The green and blue arrows represent the robot's actual location and the robot's perceived location respectively.

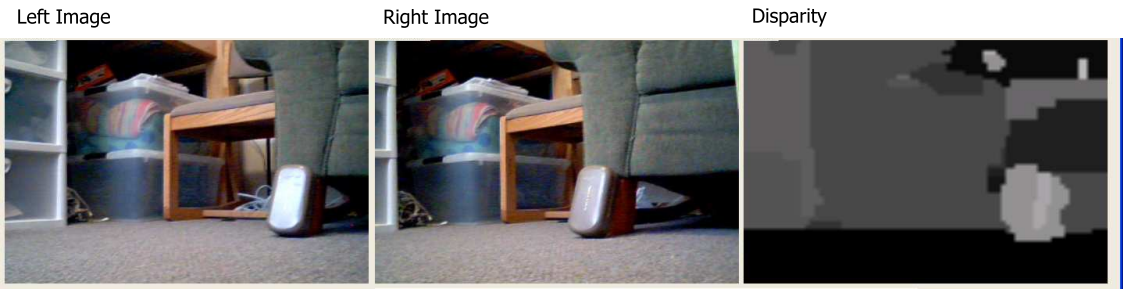

Figure 8: The view from the robots new location after the map was generated (in Figure 7) and the localization was reset. The process of re-localization can be seen in Figure 9 where the robot acquires its location with only the map data.

where it was. It was then turned to a 35 degree angle from its initial location. The view from its new orientation can be seen in Figure 8. Figures 9 show the particle filter finding the robots true location with only the map generated in the previous test.

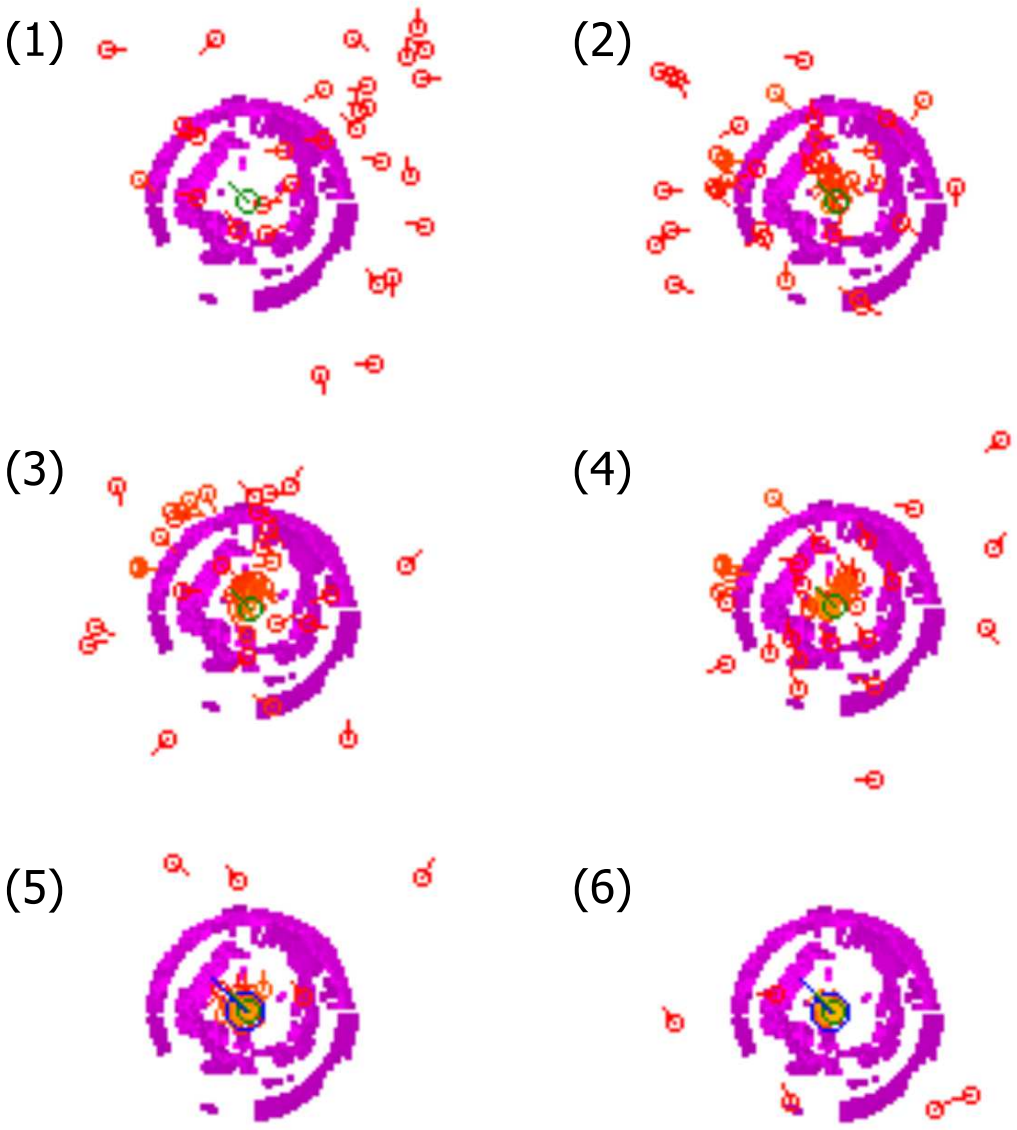

Figure 9: Localization on the robot from the 360 degree map. This diagram shows selected iterations from the initial state of the robot to when it found its location. The green circle and line show the robot and its orientation. The red-yellow circles and arrows show the particles and their orientation. Red depicts low confidence while yellow shows hight confidence. The shades in between red and yellow show partial confidence.

Discussion

The results of this project illustrate that robotic mapping through Stereoscopic Vision is both possible and has provided many benefits over the current methods for mapping. Not only is the robot able to see in horizontal resolution, stereoscopic vision allows the robot to see vertically as well. Extending the range of vision beyond a plane into a 3rd dimension allows the robot to map more area at once. Furthermore, the robot itself is inexpensive to construct when compared to other available sensors. These factors make stereoscopic vision a viable option for autonomous robotic mapping.

Stereoscopic imaging provided a convenient way to generate distances using off-the-shelf webcams. However, the systems output was limited by the environment. Because the cameras were designed to view objects at a fixed distance in constant and well-lit environments, any variation in the lighting intensity or viewing distance caused the cameras to measure incorrect distances while they adjusted. Furthermore, the cameras are only able to detect objects with contrast, provided the lighting is just right. If the lighting is too bright then the glare from any reflective surfaces prevents the cameras from detecting features, and if there is insufficient lighting, then the robot is unable to see anything. By switching the sensors to infrared sensors in future research, the robot may be able to see in more diverse lighting situations. Finally, if the robot were to encounter a glass obstacle, it would be unable to see it, as light passes through glass. This issue would be resolved by combining ultrasonic sensors to supplement the vision sensors[12]. Combining multiple sensor inputs which measure similar aspects of a system, also known as sensor fusion, allows for an improved accuracy

over what each sensor would provide individually. By combining different vision sensors that have distinct applications, a wider range of operating conditions will provide good results.

The limiting factor on the quality of mapping in this thesis was the processing time. Because both the stereoscopy and the SLAM algorithms were processing intensive programs, the inputs had to be downsized to allow frame rates of under 2 seconds. The ideal frame rate would be around 100 to 200 millisecond frames, thus providing 5 to 10 frames per second; however, with the current equipment, processing that quickly was not possible. With high-end equipment, off-site processing, or faster algorithms, the output of the mapping could be improved significantly. Faster processing power would allow for higher resolution image stereoscopy, more disparity shift tests, smaller mapping discretization, and a larger number of particles in the particle filter. Additionally, by retaining all camera frames between particle filter iterations rather than discarding them, the data could be probabilistically combined to generate more accurate images. By combining data, the particle filter would have access to more accurate data, making up for the slow refresh rate of the filter[13]. All of these factors would allow the robot to map the surroundings with more precision, higher accuracy, and with less chance of erroneous readings. Furthermore, more processing power would allow the use of high definition cameras which would significantly improve distance measurements. High definition cameras would provide over 6 times the current camera resolution (HD provides over 2 mega pixels of resolution, while the webcams used provided around 0.3 mega pixels).

This project focused on the strengths of stereoscopic vision. However, if this technology were coupled with high precision laser rangefinders, accurate wheel systems, and GPS systems, the robot would be able to map out its area of operations with very little error. Introducing new wheel systems would allow the robot to determine its movement using odometry. The robot's wheel movement sensors could then be combined with GPS data to determine its orientation and coarse position in a map to narrow the scope of localization. High precision laser rangefinders could then augment the camera systems to create very

accurate keypoints for localization and mapping.

By upgrading the equipment, the robot would be able to generate maps that are 100 times more precise (with discretizations down to 1mm blocks) and processing times which would allow the robot to move at a faster rate. In addition, off site processing would allow multiple robots to operate simultaneously using the same map. Using cooperative robotic mapping, dozens of robots could simultaneously map large areas ranging from large offices to complex cave systems. Rapid mapping of areas would be vital to search and rescue operations, earthquake rescue operations, and many military reconnaissance missions.

References

[1] W. Welling, Photography in America.

[2] Nasa Stereo Project: www.nasa.gov/stereo/.

[3] Darpa Grand Challenge: www.darpa.mil/GRANDCHALLENGE/.

[4] FindStereoCorrespondence API : http://www.emgu.com/wiki/files/2.0.0.0/html/d20a4c1e-a0df-b1af-9659-b83249f472bc.htm.

[5] S. Se, D. Lowe, and J. Little, IEEE Transactions on Robotics **21.3**, 364 (2005).

[6] EMGU disribution of OpenCV: http://www.emgu.com.

[7] B. Ristic, S. Arulampalam, and N. Gordon, *Beyond the Kalman filter: particle filters for tracking applications* (PUBLISHER, ADDRESS, 2004).

[8] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, (2000).

[9] D. Fox, International Journal of Robotics Research (2003).

[10] D. Fox, W. Burgard, and S. Thrun, Journal of Artificial Intelligence Research **11**, 391 (1999).

[11] C. Estrada, J. Neira, and J. Tardos, IEEE Transactions on Robotics **21.4**, 588 .

[12] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, International Journal of Robotics Research. (2000).

[13] C. Kwok, D. Fox, and M. Meila, Proceedings of the IEEE : Special Issue on Sequential State Estimation. .