ISLAMIC UNIVERSITY OF TECHNOLOGY

Extracting Landmarks with 3D Information for Simultaneous Localization and mapping(SLAM) using Kinect

By:

Gazi Md. Hasnat Zahan (094403)

A.F.M. Zunaid (094424)

Supervised by:

Md. Kamrul Hasan Ph.D

Assistant Professor

Department of Computer Science and Engineering

Co-Supervised by:

Mr. Hasan Mahmud

Assistant Professor

Department of Computer Science and Engineering

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

Academic Year: 2012-2013

Department of Computer Science and Engineering

Islamic University of Technology.

A Subsidiary Organ of the Organization of Islamic Cooperation.

Dhaka, Bangladesh.

November 5, 2013

Declaration of Authorship

We, Gazi Md. Hasnat Zahan & A.F.M. Zunaid, declare that this thesis titled,'Extracting Landmarks with 3D Information for Simultaneous Localization and mapping(SLAM) using Kinect' and the work presented in it are our own. We confirm that:

- This work was done wholly while in candidature for a Bachelor degree at this University.
- Where any part of this thesis has not been submitted previously for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.

Submitted By:

Gazi Md. Hasnat Zahan (094403)

A.F.M. Zunaid (094424)

Extracting Landmarks with 3D Information for Simultaneous Localization and mapping(SLAM) using Kinect

Approved By:

____

Prof. Dr. M.A. Mottalib
Head of the Department,
Department of Computer Science and Engineering,
Islamic University of Technology.

____

Md. Kamrul Hasan Ph.D
Thesis Supervisor,
Assistant Professor,
Department of Computer Science and Engineering,
Islamic University of Technology.

____

Mr. Hasan Mahmud
Thesis Co-Supervisor,
Assistant Professor,
Department of Computer Science and Engineering,
Islamic University of Technology.

Abstract:

SLAM is one of the most widely researched subfields of robotics. An intuitive understanding of the SLAM process can be conveyed though a hypothetical example. Consider a simple mobile robot: a set of wheels connected to a motor and a camera, complete with actuators-physical devices for controlling the speed and direction of the unit. Now imagine the robot being used remotely by an operator to map inaccessible places. The actuators allow the robot to move around, and the camera provides enough visual information for the operator to understand where surrounding objects are and how the robot is oriented in reference to them. What the human operator is doing is an example of SLAM (Simultaneous Localization and Mapping). Determining the location of objects in the environment is an instance of mapping, and establishing the robot position with respect to these objects is an example of localization. The SLAM subfield of robotics attempts to provide a way for robots to do SLAM autonomously. A solution to the SLAM problem would allow a robot to make maps without any human assistance What so ever. In this paper we proposed and implement a new technique to increase the efficiency of SLAM using Kinect sensor.

Acknowledgements

Special thanks to:

Prof. Dr. M.A. Mottalib

Head of the Department

Department of Computer Science and Engineering

Special thanks to:

Md. Kamrul Hasan Ph.D

Thesis Supervisor

Assistant Professor

Department of Computer Science and Engineering

Special thanks to:

Mr. Hasan Mahmud

Thesis Co-Supervisor

Assistant Professor

Department of Computer Science and Engineering

Contents

List of Figures

Dedicated to our parents....

Chapter 1

Introduction

1.1 Background

SLAM is one of the most widely researched subfields of robotics. An intuitive understanding of the SLAM process can be conveyed though a hypothetical example. Consider a simple mobile robot: a set of wheels connected to a motor and a camera, complete with actuators-physical devices for controlling the speed and direction of the unit. Now imagine the robot being used remotely by an operator to map inaccessible places. The actuators allow the robot to move around, and the camera provides enough visual information for the operator to understand where surrounding objects are and how the robot is oriented in reference to them. What the human operator is doing is an example of SLAM (Simultaneous Localization and Mapping). Determining the location of objects in the environment is an instance of mapping, and establishing the robot position with respect to these objects is an example of localization. The SLAM subfield of robotics attempts to provide a way for robots to do SLAM autonomously. A solution to the SLAM problem would allow a robot to make maps without any human assistance What so ever.

1.2 Application

If a solution to the SLAM problem could be found, it would open the door to innumerable mapping possibilities where human assistance is cumbersome or impossible. Maps could be made in areas which are dangerous or inaccessible to humans such as deep-sea environments or unstable structures. A solution to

SLAM would obviate outside localization methods like GPS or man-made beacons. It would make robot navigation possible in places like damaged (or undamaged) space stations and other planets. Even in locations where GPS or beacons are available, a solution to the SLAM problem would be invaluable. GPS is currently only accurate to within about one half of a 6 meter, which is often more than enough to be the difference between successful mapping and getting stuck. Placing man-made beacons is expensive in terms of time and money; in situations where beacons are an option, simply mapping by hand is almost always more practical.

1.3 History

The genesis of the probabilistic SLAM problem occurred at the 1986 IEEE Robotics and Automation Conference held in San Francisco. This was a time when probabilistic methods were only just beginning to be introduced into both robotics and AI. A number of researchers had been looking at applying estimation-theoretic methods to mapping and localization problems; these included Peter Cheeseman, Jim Crowley, and Hugh Durrant-Whyte. Over the course of the conference many paper table cloths and napkins were filled with long discussions about consistent mapping. Along the way, Raja Chatila, Oliver Faugeras, Randal Smith and others also made useful contributions to the conversation. The result of this conversation was a recognition that consistent probabilistic mapping was a fundamental problem in robotics with major conceptual and computational issues that needed to be addressed. Over the next few years a number of key papers were produced. Work by Smith and Cheesman [3] and Durrant-Whyte [3] established a statistical basis for describing relationships between landmarks and manipulating geometric uncertainty. A key element of this work was to show that there must be a high degree of correlation between estimates of the location of different landmarks in a map and that indeed these correlations would grow with successive observations. At the same time Ayache and Faugeras [1] were undertaking early work in visual navigation, Crowley [9] and Chatila and Laumond [6] in sonar-based navigation of mobile robots using Kalman filter-type algorithms. These two strands of research had much in common and resulted soon after in the landmark paper by Smith, Self and Cheeseman [3]. This paper showed that as a mobile robot moves through an unknown environment taking relative observations of landmarks, the estimates of these landmarks are all necessarily correlated with each other because of the

common error in estimated vehicle location [1]. The implication of this was profound: A consistent full solution to the combined localization and mapping problem would require a joint state composed of the vehicle pose and every landmark position, to be updated following each landmark observation. In turn, this would require the estimator to employ a huge state vector (of order the number of landmarks maintained in the map) with computation scaling as the square of the number of landmarks.
Crucially, this work did not look at the convergence properties of the map or its steady-state behavior. Indeed, it was widely assumed at the time that the estimated map errors would not converge and would instead exhibit a random walk behavior with unbounded error growth. Thus, given the computational complexity of the mapping problem and without knowledge of the convergence behavior of the map, researchers instead focused on a series of approximations to the consistent mapping problem solution which assumed or even forced the correlations between landmarks to be minimized or eliminated so reducing the full filter to a series of decoupled landmark to vehicle filters ([1], [2] for example). Also for these reasons, theoretical work on the combined localization and mapping problem came to a temporary halt, with work often focused on either mapping or localization as separate problems. The conceptual break-through came with the realization that the combined mapping and localization problem, once formulated as a single estimation problem, was actually convergent. Most importantly, it was recognized that the correlations between landmarks, that most researchers had tried to minimize, were actually the critical part of the problem and that, on the contrary, the more these correlations grew, the better the solution. The structure of the SLAM problem, the convergence result and the coining of the acronym 'SLAM' was first presented in a mobile robotics survey paper presented at the 1995 International Symposium on Robotics Research [3]. The essential theory on convergence and many of the initial results were developed by Csorba [11], [10]. Several groups already working on mapping and localization, notably at MIT [12], Zaragoza [5], [4], the ACFR at Sydney [2], [5] and others [7], [13], began working in earnest on SLAM1 applications in indoor, outdoor and sub-sea environments. At this time, work focused on improving computational efficiency and addressing issues in data association or 'loop closure'. The 1999 International Symposium on Robotics Research (ISRR'99) [23] was an important meeting point where the first SLAM session was held and where a degree of convergence between the Kalman-filter based SLAM methods and the probabilistic localization and mapping methods introduced by Thrun [6] was achieved. The 2000 IEEE ICRA Workshop on

SLAM attracted fifteen researchers and focused on issues such as algorithmic complexity, data association and implementation challenges. The following SLAM workshop at the 2002 ICRA attracted 150 researchers with a broad range of interests and applications. The 2002 SLAM summer school hosted by Henrik Christiansen at KTH in Stockholm attracted all the 1 Also called Concurrent Mapping and Localization (CML) at this time. key researchers together with some 50 PhD students from around the world and was a tremendous success in building the field. Interest in SLAM has grown exponentially in recent years, and workshops continue to be held at both ICRA and IROS. The SLAM summer school ran in 2004 in Tolouse and will run at Oxford in 2006.

1.4 Motivation

Situations arise where it is desirable to have exploratory robots navigate environments that may be previously uncharted, too dangerous for living beings, or for which GPS data is unavailable. There are many ways to program robotic navigation, some better than others. Common obstacle avoidance algorithms, such as wall-following or Vector Field Histogram (VFH) algorithms, concentrate on immediate navigation but do not attempt to learn more about the environment. One algorithm that looks to tackle this problem is the Simultaneous Localization and Mapping (SLAM) algorithm. SLAM is a probabilistic method in which a robot maps an environment while simultaneously localizing itself within the map. Once a map has been generated, navigation becomes much easier. One of the biggest hurdles in robotics is being able to accurately analyze sensor data. Some mapping robots look to track odometry data as a means to determine location, and to create a map. However odometry data is imperfect. The fact that there is always some degree of sensor error needs to be considered. As time elapses, sensor error compounds, drastically distorting any map that has been created. Figure 1 shows an example of odometry based mapping. In this case, error from the odometery quickly accumulates and while any individual error may be small, the sum quickly becomes significant.Alternatively, some robots have access to GPS. While GPS is convenient, it is oftentimes not available, or not accurate enough for use indoors or in conned environments. The solution to this problem is Simultaneous Localization and Mapping. The SLAM algorithm creates a map of the environment by statistically merging odometry data with sensor measurements of the environment. This forms a feedback loop by which a robot can make a much more accurate estimate of the state of its environment as well as the robot's location within that environment.

Chapter 2

Literature Review

2.1 The SLAM Problem

One of the major developments in SLAM research was proof that errors in feature location acquired by agents are correlated with one another. The correlation exists because an error in localization will have a universal effect on the perceived location of all features (we will discuss this in more detail in the chapter on the Kalman Filter). Understanding and utilizing the relationship among errors in feature locations and robot pose is at the core of SLAM research; it is the major motivation behind solving localization and mapping concurrently. This chapter will now consider localization and mapping each individually to develop a better understanding the SLAM problem and further illuminate the motivation behind finding a simultaneous solution.
Works by Maybeck, Durant-Whyte, Nebot, and Csorba all helped inspire this portion of my thesis, but the development of SLAM problem as it appears here is (as far as I know) unique. It represents a precursor to the works of the authors mentioned above, and is written to provide motivation for various steps taken in their respective works. Few specific citations are made, but this section would not have been possible without understanding gained from these authors. In particular, many of the specifics of the localization problem and the mapping problem presented here are elaborate on diagrams created by Durant-Whyte.
SLAM is a process by which a mobile robot can build a map of an environment and at the same time use this map to deduce it's location. In SLAM both the trajectory of the platform and the location of all landmarks are estimated on-line without the need for any a priori knowledge of location.

Consider a mobile robot moving through an environment taking relative observations of a number of unknown landmarks using a sensor located on the robot as shown in Figure 1. At a time instant k, the following quantities are defined:
Xk : The state vector describing the location and orientation of the vehicle.
Uk: The control vector, applied at time k1 to drive the vehicle to a state xk at time k.
Mi: A vector describing the location of the ith landmark whose true location is assumed time invariant.
Zik: An observation taken from the vehicle of the location of the ith landmark at time k. When there are multiple landmark observations at any one time or when the specific landmark is not relevant to the discussion, the observation will be written simply as Zk.
In addition, the following sets are also defined:
X0:k = x0, x1, , xk = X0:k-1, xk : The history of vehicle locations.
U0:k = u1, u2, uk = U0:k-1 , uk : The history of control inputs.
m = m1,m2, ,mn : The set of all landmarks.
Z0:k = z1, z2, , zk = Z0:k-1, zk : The set of all landmark observations.

2.2 The Probabilistic SLAM

In probabilistic form, the Simultaneous Localization and Map Building (SLAM) problem requires that the probability distribution:
P(xk, m — Z0:k, U0:k, x0)
be computed for all times k. This probability distribution describes the joint posterior density of the landmark locations and vehicle state (at time k) given the recorded observations and control inputs up to and including time k together with the initial state of the vehicle. In general, a recursive solution to the SLAM problem is desirable. Starting with an estimate for the distribution P(xk-1, m — Z0:k-1, U0:k-1) at time k-1, the joint posterior, following a control uk and observation zk, is computed using Bayes Theorem. This computation requires that a state transition model and an observation model are defined describing the effect of the control input and observation respectively.
The observation model describes the probability of making an observation zk when the vehicle location and landmark locations are known, and is generally described in the form
P(zk — xk,m).

It is reasonable to assume that once the vehicle location and map are defined, observations are conditionally independent given the map and the current vehicle state.
The motion model for the vehicle can be described in terms of a probability distribution on state transitions in the form
P(xk — xk-1, uk)
That is, the state transition is assumed to be a Markov process in which the next state xk depends only on the immediately proceeding state xk-1 and the applied control uk, and is independent of both the observations and the map.
The SLAM algorithm is now implemented in a standard two-step recursive (sequential) prediction (time-update) correction (measurement-update) form:
Time-update:
P(xk,m — Z0:k-1,U0:k, x0)
=P(xk — xk-1, uk) P(xk-1,m— Z0:k-1, U0:k-1, x0) dxk-1 ——————- (4)
Measurement Update:
P(xk,m — Z0:k,U0:k, x0) = P(zk — xk, m) P(xk, m — Z0:k-1,U0:k, x0)/ P(zk — Z0:k-1,U0:k) ———-(5)
Equations 4 and 5 provide a recursive procedure for calculating the joint posterior P(xk,m — Z0:k,U0:k, x0) for the robot state xk and map m at a time k based on all observations Z0:k and all control inputs U0:k up to and including time k. The recursion is a function of a vehicle model P(xk — xk-1,uk) and an observation model P(zk — xk , m). It is worth noting that the map building problem may be formulated as computing the conditional density P(m — X0:k,Z0:k,U0:k). This assumes that the location of the vehicle xk is known (or at least deterministic) at all times, subject to knowledge of initial location. A map m is then constructed by fusing observations from different locations. Conversely, the localization problem may be formulated as computing the probability distribution P(xk — Z0:k,U0:k,m). This assumes that the landmark locations are known with certainty and the objective is to compute an estimate of vehicle location with respect to these landmarks.

2.3 Steps of SLAM Implementation

The simultaneous localization and mapping (SLAM) problem asks if it is possible for a mobile a robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. In

SLAM both the trajectory of the platform and location of all landmarks are estimated without the need for a priori knowledge of the location [17].
Robotic mapping suffers from basically 2 kinds of errors. One is the measurement error and another is the process error. The solution is a probabilistic approach called SLAM that makes use of a kalman Filter. The kalman Filter is one of the mostly used filtering algorithms that frequently used in Robotics. The job of kalman filter is to reduce errors in the measurement and update of the robot state. The Kalman Filter works according to the Predict Update schema as shown in figure 1
There are mainly 5 steps in SLAM approach. These are:

1. State Estimation
2. Landmark Extraction
3. Data Association
4. State Update
5. Landmark Update

State Estimation: At any given time step the robot knows the actions that it will apply to its motors in the next time step. It also knows how those actions will change the robot position in its environment. Therefore, in the estimation step, the robot uses this information to predict where it will be in the next step.
Landmark Extraction: A landmark is any part of the environment that has some distinguishable features. After making a prediction for the next time step the robot carries some actions on its motors and moves to the time step for which it has made a prediction. The robot then take a sensor measurement of the environment and extracts any distinguishable landmarks. These landmarks must be such that they can be correctly recognized again at a future time step. Any error in landmark extraction can have catastrophic effect on the estimation of the map.
Data Association: After extracting all recognizable landmarks in its environment, the robot must compare each observed landmarks to the previously observed landmarks. The newly extracted landmarks are then associated with the previous landmarks. The data that are associated are the location of the extracted landmarks, their size, in which angle they are placed with the robot etc. The data association step is one of the most challenging step of SLAM as it incorporates with complexity algorithms. The robot matches up the landmarks

that correspond and compares the observed landmark locations to a prediction of where the landmarks had would have been had the state estimation step been completely accurate. This creates an error factor representing this difference, and is known as the innovation error.
State Update: The robot uses the innovation error to modify the prediction of its state. This is done by multiplying the innovation error by a statistically optimal factor called the kalman gain, and adding it to the prediction state. The robot has now formed its best estimate of its state for that time step.
Landmark Update: The robot then takes any new landmarks that it has observed and adds them to the state so that they may be re-observed for the later time step.

2.4 Kalman Filter

The Kalman Filter (or KF) was developed by R.E. Kalman, whose prominent paper on the subject was published in 1960 (Welch and Bishop, [2006]). The KF potential in specific localization and mapping problems was well known shortly after Kalman published his paper in 1960. However, the KF was not generally used to represent an entire world state (and was therefore not used as a complete solution to the SLAM problem) until 1988 (Castellanos, Neira, Tardos, [2004]). The Kalman Filter is an algorithm which processes data and estimates variable values (Maybeck, [1979]). In a SLAM context, the variable values to be estimated will consist of the robot pose, and feature locations-i.e. the world state. The data to be processed may include vehicle location, actuator input, sensor readings, and motion sensors of the mobile robot. In other words, the Kalman Filter can utilize all available data to simultaneously estimate robot pose and generate a feature map. Under certain conditions, the estimates made by the Kalman Filter are very good; in fact, they are in a sense ”optimal”. Welch and Bishop explain that the state estimates provided by the Kalman Filter will use any available information to minimize the mean of the squared error of the estimates with regard to the available information (Welch and Bishop, [2006]). Another way of putting this is that the error in the state estimates made by the Kalman Filter are minimized statistically (Maybeck, [1979]).

2.5 Kalman Filter and SLAM

There are many reasons for why KF techniques are such a popular choice for SLAM researchers. As was mentioned previously, the KF is capable of incorporating all available measurements (Maybeck, [1979]) and providing a least squares estimate of a process state. When modeled carefully, this is precisely what SLAM algorithms attempt to do. Furthermore, as we saw in The Slam Problem, SLAM would be trivial if robot sensors were noise free; handling noise in the best way possible is the essential purpose of the KF, (Maybeck, [1979]). Finally, the KF works well in practice and so researchers continue to use it (Durant-Whyte, [2002]).
In order to implement the Kalman Filter in the SLAM process, we have to describe the system. Therefore, it will be assumed that the robot moves in a linear fashion, and that the system can be described by the following equations:
xk+1 = Fkxk + Gkuk + vk ————–(1)
yk = Hkxk + wk ——————–(2)
Equation (1) is the state equation and indicates that the robot state at the next time step is a linear function of the current state, xk, the input vector, uk, and an error term, vk.
Equation (2) is the output equation and indicates that the measurement vector taken from the sensors is a linear function of the robot state, xk, and an error term, wk
The variables are as follows (Assuming n landmarks have been observed):
xk : Robot state. This is a (3 + 2n) 1 vector containing the x, y estimate of the robot's position as well as the estimated angle, ?, that the robot is pointed. The state also contains the estimated x, y coordinates of every previously observed landmark in the environment.
uk : System input. This is a 31 vector containing the predicted change in robot x, y, ? to the next time step.
yk : System measurement. This is a 2n1 vector containing observed, sensor measurements of the x, y coordinates for each landmark with respect to the robot's coordinate frame.
vk : Process noise. This is a (3 + 2n) 1 vector containing the error in the system e.g. slipping wheels. The error is assumed to be Gaussian with zero mean.
wk : Measurement noise. This is a 2n 1 vector containing the error in the sensor measurements. The error is assumed to be Gaussian with zero mean.

Vk : Covariance matrix of the process noise vector. This is a (3 + 2n) (3 + 2n) matrix.
Wk : Covariance matrix of the measurement noise vector. This is a 2n 2n matrix.
Fk : State Transition Matrix. This matrix maps the previous state to the next state. For the purposes of this project, this will just be the (3+2n)(3+2n) identity matrix since we want there to be a direct mapping.
Gk : Input Transition Matrix. This matrix maps the input vector to the next state. For the purposes of this project, this will just be a (3 + 2n) 3 matrix where the top 3 3 matrix is the 5 identity matrix. This is the case since we want the del x,del y, from the input vector to get mapped to only the x, y, ? elements of the state vector.
Hk : Measurement Transition Matrix. This matrix maps the state vector to the measurement vector.
For the purposes of this project, this will just be a 2n(3+2n) matrix consisting of a Hi submatrix for each observed landmark. Each Hi corresponds to the ith landmark and is arranged with enough 0's in each row so thatthe 2 2 identity matrix lines up with the ith landmark in the state vector. This arrangement is required so that Hk takes the same form as the measurement vector.

Chapter 3

Problem

In the recent papers based on EKF SLAM using Kinect, the researchers cited some of the limitations of the current algorithm that included -
1. fixing the translation bug in the EKF part
2. incorporating a line landmark detection system
3. adding more sophisticated landmark acossiation algorithm
4. producing a 3D map from the 2D slices of the data
5. improving efficiency throughout the process
6. handling degenerate cases where camera faces only planar surfaces
7. removing inconsistency between RGB-BA mapping and 8 point RANSAC by emphasizing on the image homography, building more robust loop closure logic etc [14][20].

Among these shortcomings, the one that we have considered to work in is 3D mapping. The previous approaches of SLAM didn't have the resource to map a 3D environment until the advent of visual SLAM. After that, the problem was to implement it in an efficient way as the previous visual SLAM implementations relied on monocular or stereo vision cameras who couldn't give any depth value from its readings that could be used to map the 3D environment. So, after kinect was introduced, there arose a chance to map 3D environments using the depth value and incorporate the height information in the map.

Chapter 4

Proposed Methodology

4.1 Our Approach

To overcome this problem, we will take the depth value and the angle information from the kinect sensor and then retrieve the height information of the landmarks by geometric calculation. This information will be added to every landmark in the map making it 3D.In present algorithm, they used 2D kalman filter where only the (x,y) position of the landmarks were incorporated. We will try to incorporate a Z-value i.e. the height information of the landmarks in the kalman filter that will give us the correlation between the robot position and the 3D landmarks in place of 2D. This height information will help in the mobile robot navigation where a robot will be able to determine the feasibility of navigating itself over the landmarks by judging its height information. We constructed an architecture of our proposed methodology in the light of the SLAM architecture. The architecture is shown below :

4.2 Why Kinect

The Xbox Kinect is a vision sensor made by Microsoft for use with the Xbox 360 gaming console. The Kinect is revolutionary in that it is capable of 3D image sensing. From Figure 3 below, it can be seen that the middle sensor is an RGB camera, and the other two are used in 3D depth sensing. The Kinect works by projecting a large number of IR dots around the area in front of the device. An example of this can be seen below in Figure 4. The depth device to the left in Figure 3 projects the IR dots around the room. The sensor to the right then

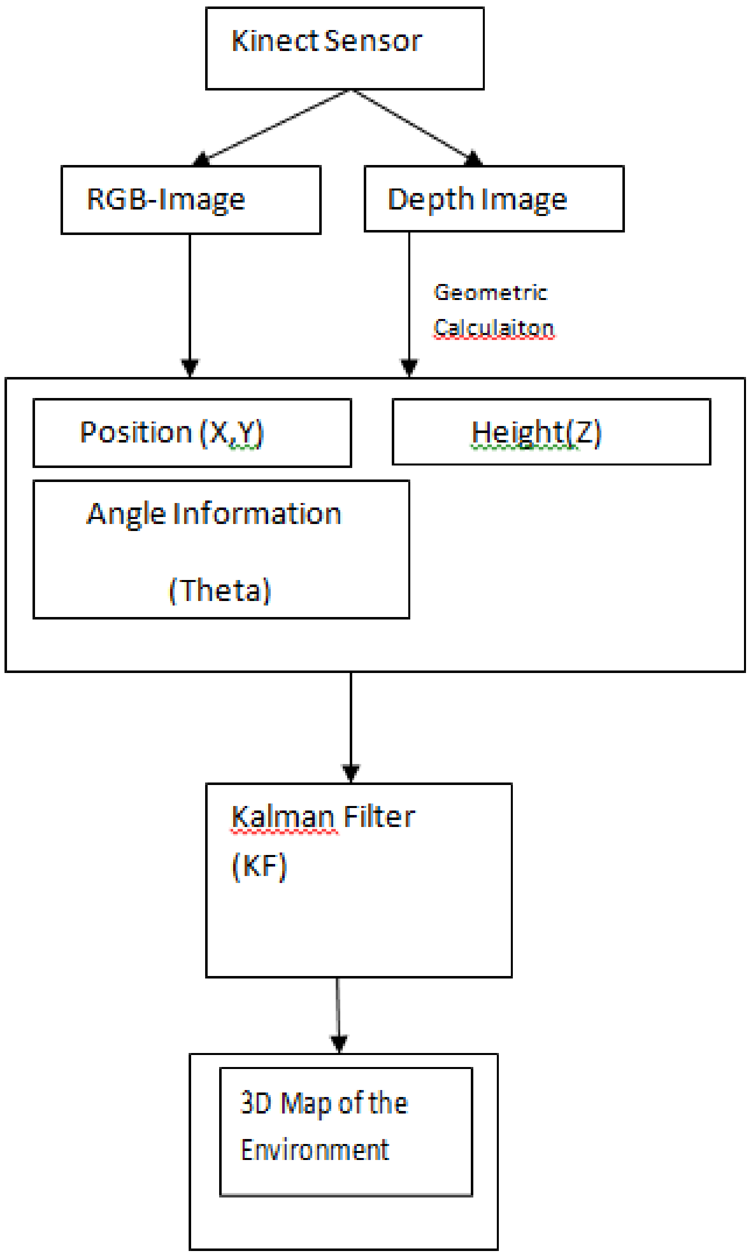


Figure 4.1: Architecture of our proposed SLAM

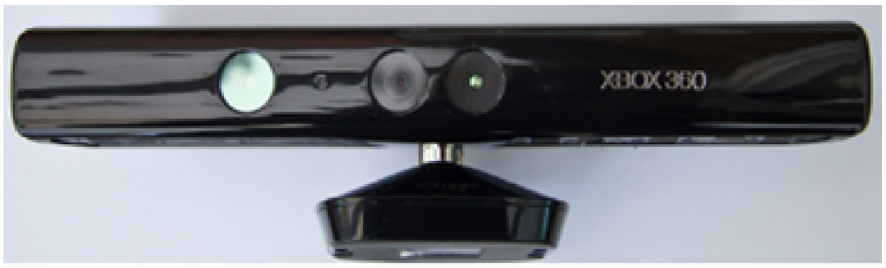


Figure 4.2: Kinect

observes the dot pattern. The Kinect is able to calculate the depth to every point in sight based on the dispersion pattern. For example, dots that are close together indicate objects that are closer, while dots that are more spread out indicate objects that are farther away. [13] Kinect has 8-bit RGB VGA resolution (640*480 Pixels) video camera. The Kinect sensor bar also contains two cameras, a special infrared light source, and four microphones. It also contains a stack of signal processing hardware that is able to make sense of all the data that the cameras, infrared light, and microphones can generate. By combining the output from these sensors, a program can track and recognize objects in front of it, determine the direction of sound signals, and isolate them from background noise.

4.3 Implmentation

The hardware for this project was chosen to have the necessary components to implement a SLAM algorithm. Therefore, the robot needed the following:
(1.) A base capable of supporting the rest of the hardware. The base had to be capable of receiving and executing motor commands, as well as generating and sending odometry data.
(2.) A vision sensor capable of generating RGB images of the environment in real time. This was necessary in order to identify and track landmarks in the environment.
(3.) A distance sensor capable of determining the distances to each landmark in the environment.
(4.) A processor capable of interfacing with the aforementioned hardware and containing the processing power to run a SLAM algorithm in real time.
For selecting a base , we chose a robot that will give us the necessary information. The information contains generating and sending motor commands as well as odometry data. Odometry data is normally used to calculate distance

covered by the robot. Odometry data is measured using a wheel encoder. Using the circumference of the wheel encoder can calculate the rotation of wheels and from that the distance is measured. For our thesis we select the iRobot create which is a simple to implement and also cheap. It also performs well and gives us all the necessary information.
The next important phase is to generate RGB images from the environment in real time. This is necessary in order to identify and track landmarks in the environment. For generating the RGB images we use XBOX 360 Kinect sensor. The outstanding feature of kinect is that it not only gives us the image but also gives us the depth information of each and every pixel. This is a revolutionary device and no other device ever gave that kind of information. We use the depth information to calculate the height of an particular object which is known as landmarks in SLAM. We found that calculating the height and incorporating it with the existing 2d KF SLAM algorithm we can give more information about the map to the robot. We know that map building is main purpose of the SLAM and building more efficient map gives us more accurate result to the SLAM. So the main contribution to our implementation of this thesis is to calculate the height of the landmarks using Kinect and incorporate this information with the existing kalman filter based SLAM algorithm and see what is the results.

4.4 Kinds of Landmarks we used in our implementation

For simplicity purpose and to avoid complexity we take very simple shaped landmarks. We choose simple shaped rectangle, triangle and circle for our thesis. The idea behind choosing these simple landmarks are that we assume that the environment is a planar surface and the objects surrounding the robot are very simple shaped that gives us more accurate result. Though the real world is surrounded with so many complex objects with so many complex combinations for better result and for creating the map more efficiently we assume that the environment is pretty simple to explore for the robot.
From the above three shapes we can easily understand how simple our landmarks are. We use these three shaped objects in our thesis implementation. We take several images where all possible combinations of these objects are considered. For example, first we take an image of only a single object that stands in front of kinect camera. Then we take another image of circle and rectangle shaped objects where they are not staying in parallel rather they are standing with two

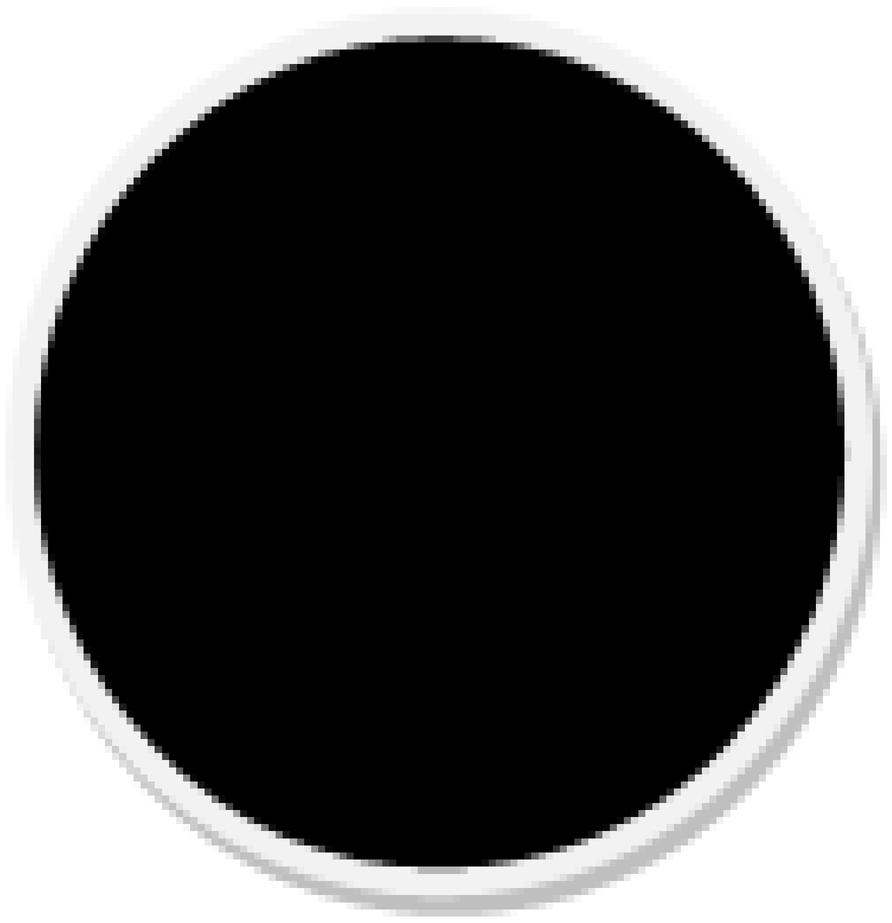


Figure 4.3: Circle

different positions in front of the camera. Then again we capture an image considering all the landmarks together.

4.5 Image Capturing Procedure

For capturing the image we made a GUI interface using Microsoft Visual Studio 2010.
The images captured using this GUI are color images and depth images. Some of the images are shown below:

4.6 Obtaining the Heights

When only the landmarks are visible in the images, we pass three arguments to the gHeight() function to calculate the height of the landmark. We calculate the

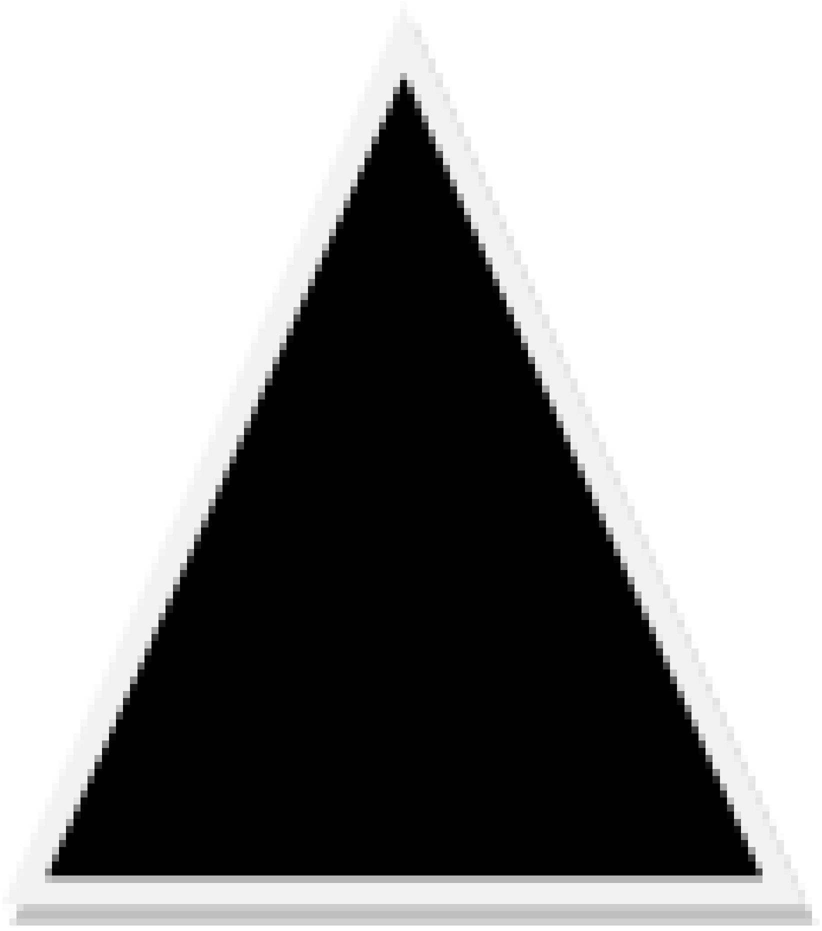


Figure 4.4: Triangle

top most pixel of the landmarks and also the bottom most pixel of the landmarks using how much pixels are covered by the landmarks in the total image and how much viewing angle is covered by the landmarks of the total image. After getting these two pixel locations we can get the depth value of the corresponding pixels. So this two depth measurement and the angle between this two line is passed as an argument to the gHeight() function. We build an algorithm the will take this tree values and calculate the corresponding height of the landmarks. The algorithm looks like this:

```
function gheight(d1,d2,pd)
ds=min(d1,d2);
db=max(d1,d2);
theta=((pd*47)/480);
db1=ds*cos(deg2rad(theta));
db2=db-db1;
dp=sqrt(ds power(2)-db1 power(2);
```

FIGURE 4.5: Rectangle

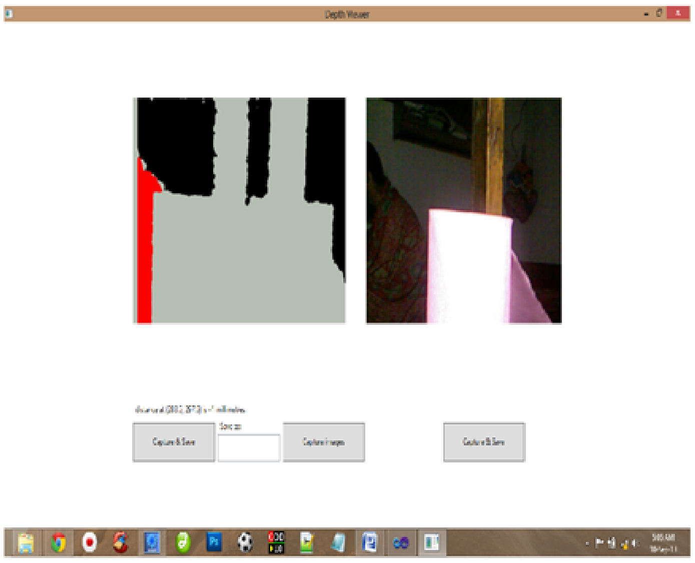


FIGURE 4.6: Graphical user interface made in visual studio

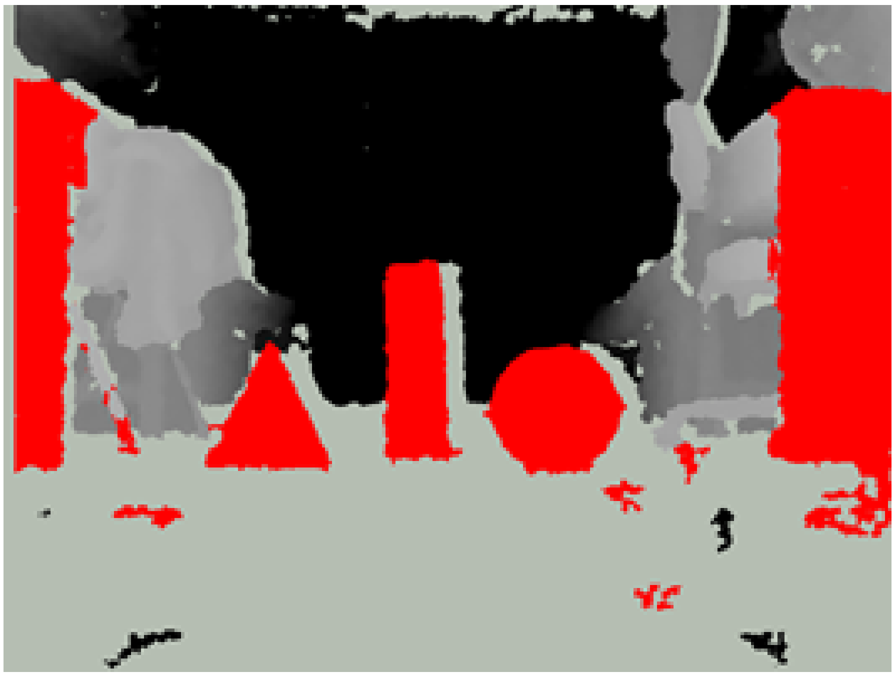


Figure 4.7: 1

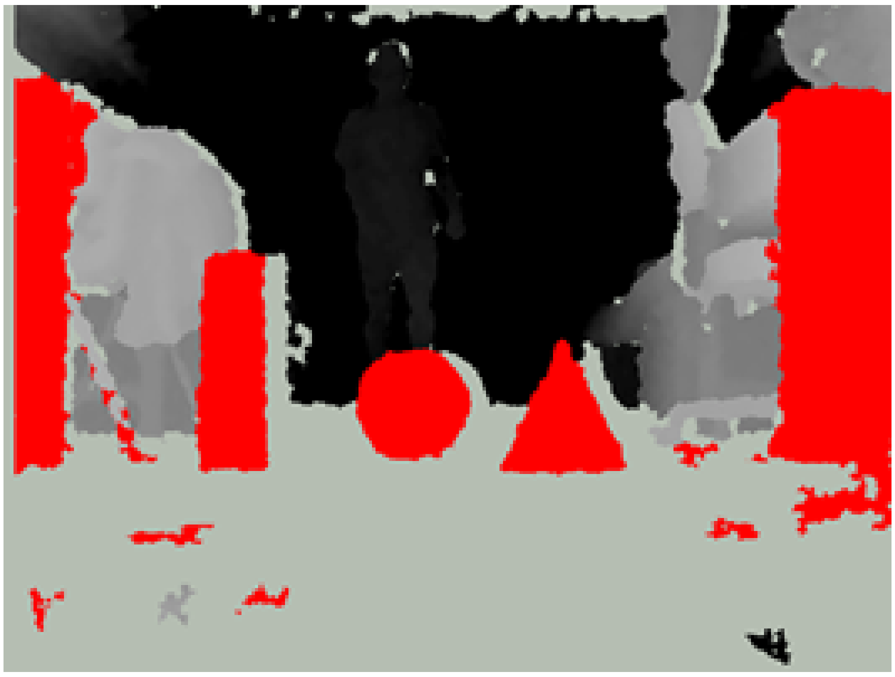


Figure 4.8: 2

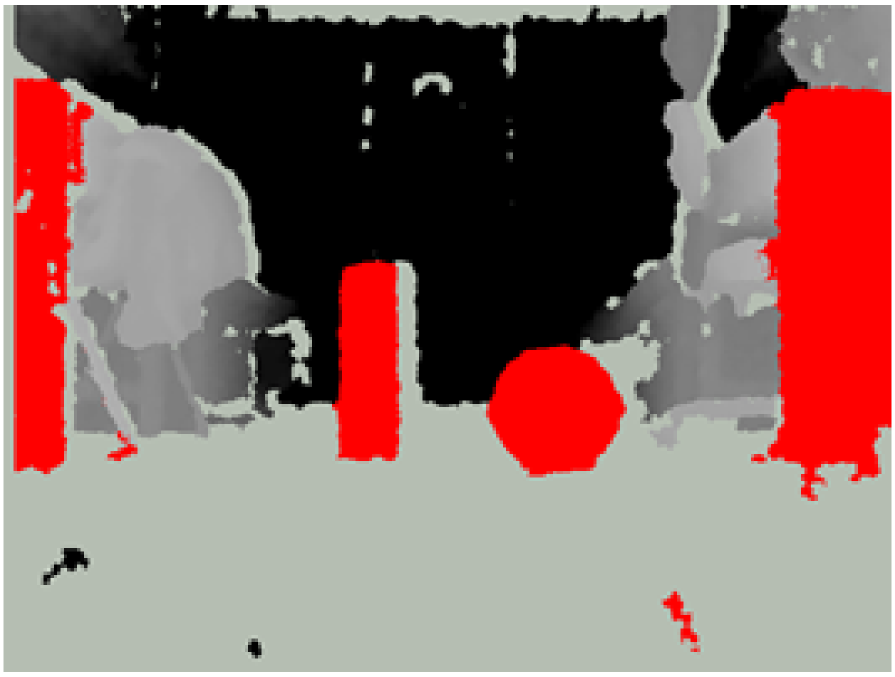


Figure 4.9: 3

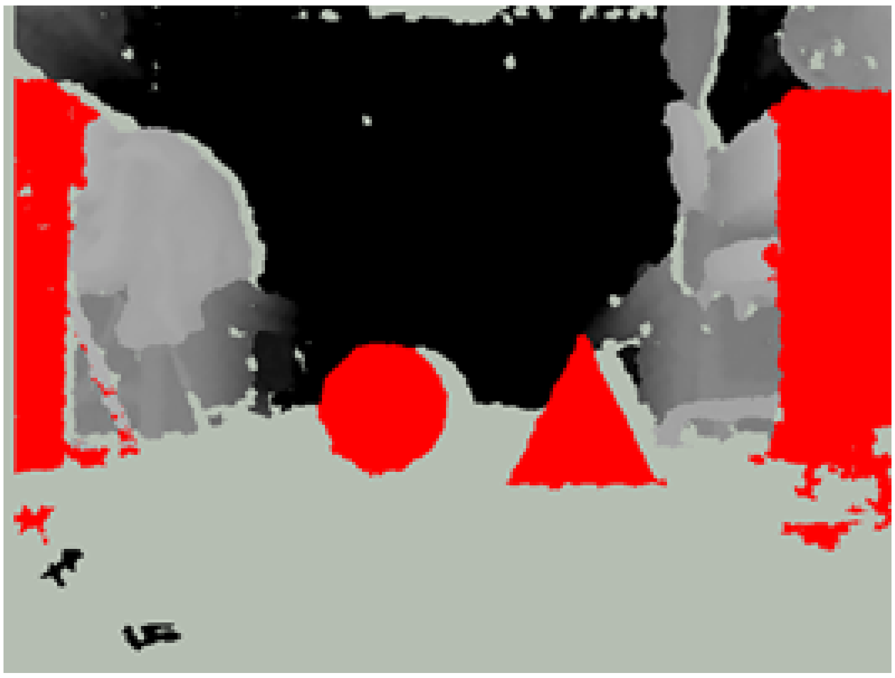


Figure 4.10: 4

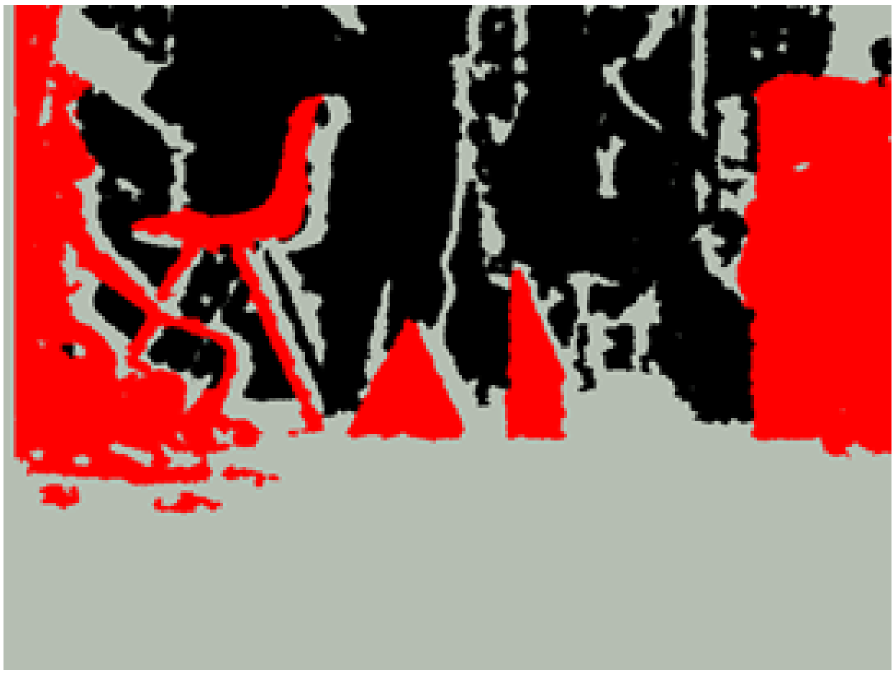


Figure 4.11: 5

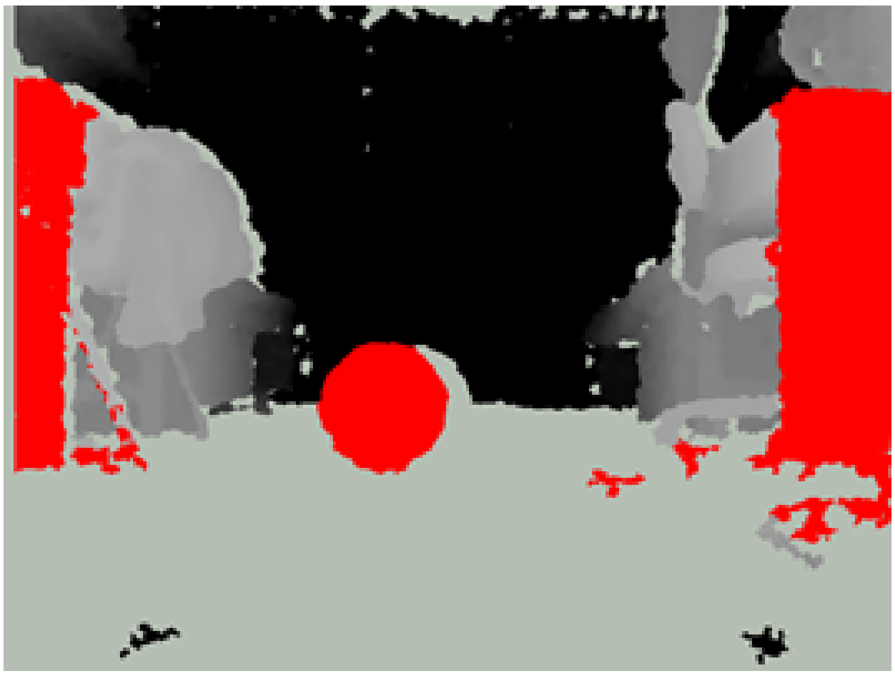


Figure 4.12: 6

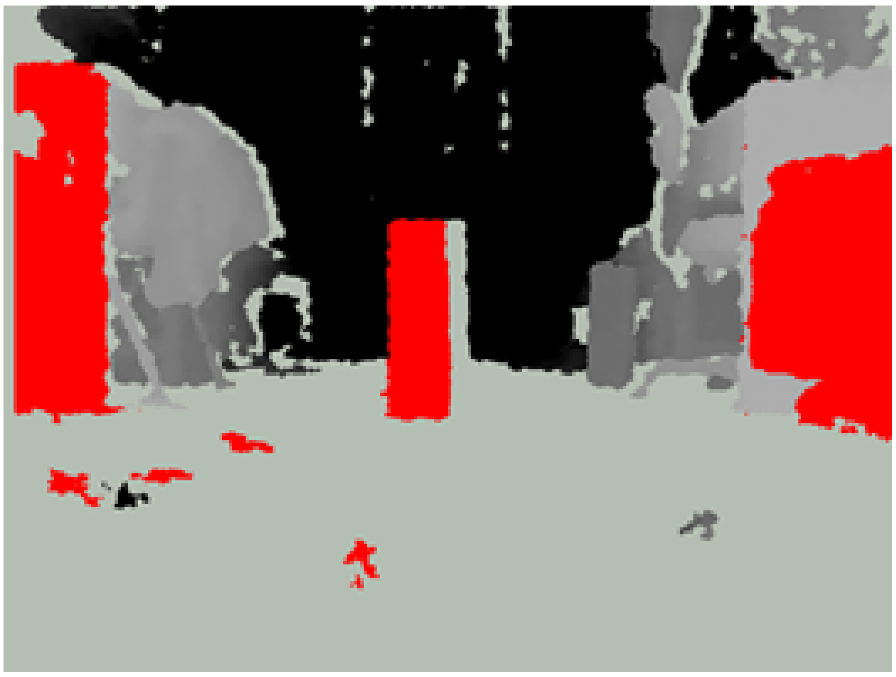


FIGURE 4.13: 7

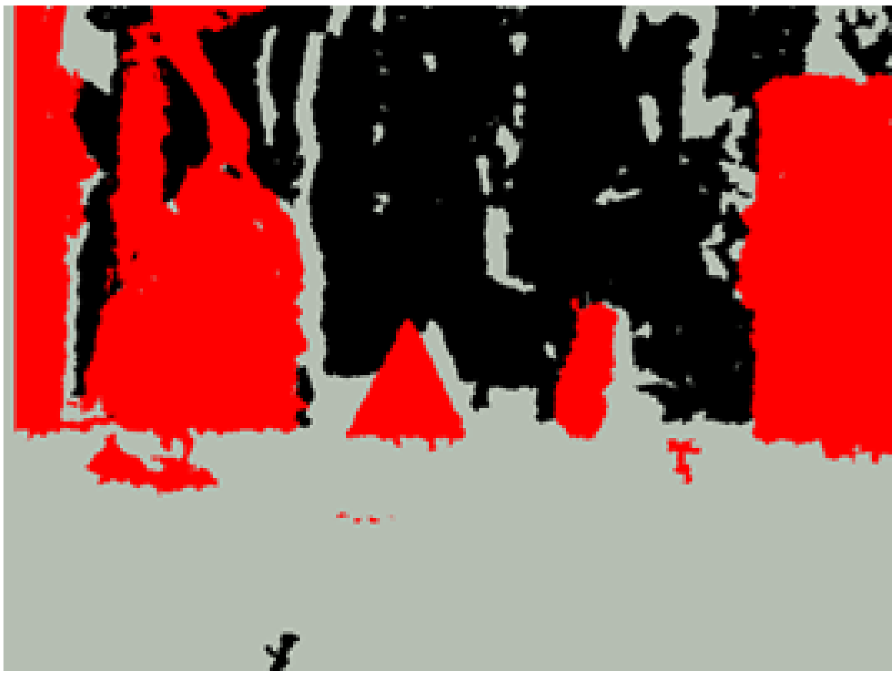


FIGURE 4.14: 8

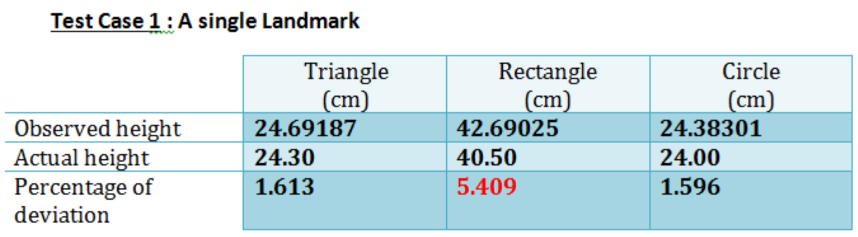
h=sqrt(db2 power(2)+dp power(2)

Chapter 5

Evaluation and Future Work

5.1 Evaluation

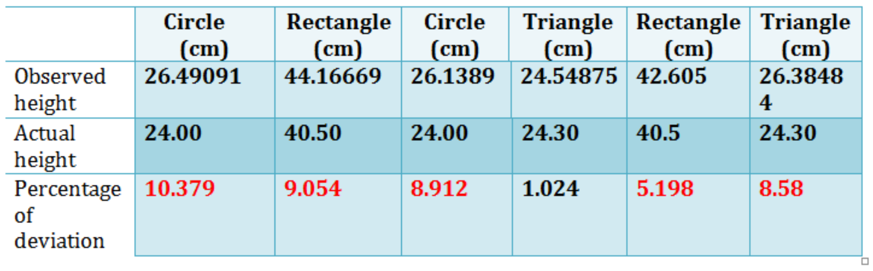
From the result table we can see that the percentage of deviation we have observed is very less. In each test cases the result is quite satisfactory. The observed height is very close to the actual height and the difference between them is very negligible in case of robot navigation. This small will not create any



Test Case 1 : A single Landmark

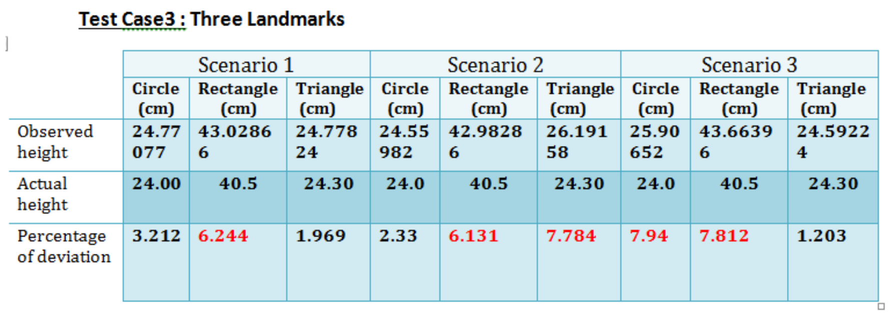
	Triangle (cm)	Rectangle (cm)	Circle (cm)
Observed height	**24.69187**	**42.69025**	**24.38301**
Actual height	**24.30**	**40.50**	**24.00**
Percentage of deviation	**1.613**	**5.409**	**1.596**

FIGURE 5.1: result table with single landmark



	Circle (cm)	**Rectangle (cm)**	**Circle (cm)**	**Triangle (cm)**	**Rectangle (cm)**	**Triangle (cm)**
Observed height	**26.49091**	**44.16669**	**26.1389**	**24.54875**	**42.605**	**26.3848 4**
Actual height	**24.00**	**40.50**	**24.00**	**24.30**	**40.5**	**24.30**
Percentage of deviation	**10.379**	**9.054**	**8.912**	**1.024**	**5.198**	**8.58**

FIGURE 5.2: result table with 2 landmarks



Test Case3 : Three Landmarks

	Scenario 1			Scenario 2			Scenario 3		
	Circle (cm)	Rectangle (cm)	Triangle (cm)	Circle (cm)	Rectangle (cm)	Triangle (cm)	Circle (cm)	Rectangle (cm)	Triangle (cm)
Observed height	24.77077	43.02866	24.77824	24.55982	42.98286	26.19158	25.90652	43.66396	24.59224
Actual height	24.00	40.5	24.30	24.0	40.5	24.30	24.0	40.5	24.30
Percentage of deviation	3.212	6.244	1.969	2.33	6.131	7.784	7.94	7.812	1.203

FIGURE 5.3: result table with 3 landmarks

kind of problem while developing the final map of the environment. Though in our experiment we have a used 3 different types of simple planar shapes it can also measure all kinds of simple shapes.

5.2 Future Work

From the problem statement we have given previously we have shown a method to make a 3-dimensional mapping environment by incorporating the height information of the landmarks. This information can help the robot while navigating specially in a situation that it needs to decide that whether it will roll over on the landmark or it will just simply avoid it. It depends on the height information of the landmarks. If the landmarks are well small in size then robot can easily identify this and do its job accordingly. So our proposed methodology shows hao to detect an object using the very versatile and efficient device like Kinect. We hope that our algorithm can be implemented accordingly and we can make the 3D map with the least amount of error. In this whole paper we basically focus on the basics of SLAM algorithm and how kalman filters are using in this algorithm. So in the near future we will try to implement the whole project using a real life device like kinect.

Chapter 6

Conclusion

We began our discussion in the first chapter by establishing what the SLAM problem is on an intuitive level, and by discussing what a solution to the SLAM problem would accomplish. In short, the SLAM problem is tough to solve or even understand fully, but the closer we get to solving it the more we open the door to a truly autonomous robot. To better understand the SLAM problem, we first broke the SLAM problem down into the component problems (localization and mapping on their own), and we discovered each could be solved without too much trouble. The problem is, in order to localize we need to start with an error free map, and to map we need to be able to localize. This is what motivated us to consider the first solution to the SLAM problem, the Kalman Filter.
In the Kalman Filter we found a solution that could take advantage of all available data statistically minimize the total squared error in our state prediction. The KF accomplishes this task for us by maintaining a state prediction and a covariance matrix containing a confidence value for every binary relationship in the environment. We noticed the KF specified how each observation could be used to update all feature predictions, and all covariance matrix values. Also, we noticed that the state prediction and covariance matrix had a fixed size independent of the number of iterations of the filter. Consequently, the KF allows us to map for as long as we want without increasing the time to run each iteration. It is safe to say that if there are approximately 200 features we wish to map, a Kalman Filter will be the correct method to use. It should also be noted that the Kalman Filter theory was not simply accepted at face value. It was rigorously proved, thus allowing for strong conceptual understanding. Also, the Kalman Filter was applied to SLAM in a somewhat different manner than it had been by former Trinity projects. A similar project

was accomplished in 2010 as discussed in [1]. Many concepts were learned and borrowed from this project. However, the program was written slightly differently, and one particular fault in the previous project was addressed. The previous project required that all of the 25 landmarks be in sight at all time. The methods described in this project allow for landmarks to go unobserved and not disturb the estimates of re observed landmarks. Nor do the estimates of the re observed landmarks affect the non re observed landmarks.
our proposed methodology shows how to detect an object using the very versatile and efficient device like Kinect. We hope that our algorithm can be implemented accordingly and we can make the 3D map with the least amount of error. In this whole paper we basically focus on the basics of SLAM algorithm and how kalman filters are using in this algorithm. So in the near future we will try to implement the whole project using a real life device like kinect.

Chapter 7

Simulation Code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using Microsoft.Kinect;
using System.IO;
namespace WpfApplication1
{
    public partial class MainWindow : Window
    {

        int someValue = 0;
        short[] rawDepthData;
        const float MaxDepthDistance = 4095; // max value returned
        const float MinDepthDistance = 850; // min value returned
        const float MaxDepthDistanceOffset = MaxDepthDistance - MinDepthDistance;
        KinectSensor _sensor;
        BitmapSource image;
        BitmapSource imgFrame;
        public MainWindow()
        {
            InitializeComponent();
        }

        private void image1_ImageFailed(object sender, ExceptionRoutedEventArgs e)
        {

        }
```

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    if (KinectSensor.KinectSensors.Count > 0)
    {
        _sensor = KinectSensor.KinectSensors[0];

        if (_sensor.Status == KinectStatus.Connected)
        {
            _sensor.DepthStream.Enable();
            _sensor.ColorStream.Enable();
            _sensor.AllFramesReady += new EventHandler<AllFramesReadyEventArgs>(_sensor_
            _sensor.Start();

        }
    }
}

void _sensor_AllFramesReady(object sender, AllFramesReadyEventArgs e)
{
    using (DepthImageFrame depthFrame = e.OpenDepthImageFrame())
    {
        if (depthFrame == null)
        {
            return;
        }

        byte[] pixels = GenerateColoredBytes(depthFrame);
        int stride = depthFrame.Width * 4;

        image = BitmapSource.Create(depthFrame.Width, depthFrame.Height,
            96, 96, PixelFormats.Bgr32, null, pixels, stride);
        image1.Source = image;

    }

    using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
    {
        if (colorFrame == null)
        {
            return;
        }

        byte[] pixels = new byte[colorFrame.PixelDataLength];
        colorFrame.CopyPixelDataTo(pixels);

        int stride = colorFrame.Width * 4;
        imgFrame = BitmapSource.Create(colorFrame.Width, colorFrame.Height,
            96, 96, PixelFormats.Bgr32, null, pixels, stride);
        image2.Source = imgFrame;
    }
}

void StopKinect(KinectSensor sensor)
{
    if (sensor != null)
```

```
            {
                sensor.Stop();
                sensor.AudioSource.Stop();
            }
        }

        private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
        {
            StopKinect(_sensor);
        }

        private byte[] GenerateColoredBytes(DepthImageFrame depthFrame)
        {

            //get the raw data from kinect with the depth for every pixel
            rawDepthData = new short[depthFrame.PixelDataLength];
            someValue = depthFrame.PixelDataLength;
            depthFrame.CopyPixelDataTo(rawDepthData);

            //use depthFrame to create the image to display on-screen
            //depthFrame contains color information for all pixels in image
            //Height x Width x 4 (Red, Green, Blue, empty byte)
            Byte[] pixels = new byte[depthFrame.Height * depthFrame.Width * 4];

            //Bgr32  - Blue, Green, Red, empty byte
            //Bgra32 - Blue, Green, Red, transparency
            //You must set transparency for Bgra as .NET defaults a byte to 0 = fully transparen

            //hardcoded locations to Blue, Green, Red (BGR) index positions
            const int BlueIndex = 0;
            const int GreenIndex = 1;
            const int RedIndex = 2;

            someValue = rawDepthData.Length;
            //loop through all distances
            //pick a RGB color based on distance
            for (int depthIndex = 0, colorIndex = 0;
                depthIndex < rawDepthData.Length && colorIndex < pixels.Length;
                depthIndex++, colorIndex += 4)
            {
                //gets the depth value
                int depth = rawDepthData[depthIndex] >> DepthImageFrame.PlayerIndexBitmaskWidth;


                if (depth < MinDepthDistance && depth !=-1)
                {
                    pixels[colorIndex + BlueIndex] = 178;
                    pixels[colorIndex + GreenIndex] = 190;
                    pixels[colorIndex + RedIndex] = 181;
                }
                if (depth > MaxDepthDistance)
                {
                    pixels[colorIndex + BlueIndex] = 178;
                    pixels[colorIndex + GreenIndex] = 190;
                    pixels[colorIndex + RedIndex] = 181;
                }
                if (depth == -1)
                {
                    pixels[colorIndex + BlueIndex] = 178;
                    pixels[colorIndex + GreenIndex] = 190;
                    pixels[colorIndex + RedIndex] = 181;
                }

                if (depth >MinDepthDistance && depth< MinDepthDistance+1000)
                {
                    pixels[colorIndex + BlueIndex] = 0;
                    pixels[colorIndex + GreenIndex] = 0;
                    pixels[colorIndex + RedIndex] = 255;
                }


            }
            return pixels;
        }

       private void image1_MouseMove(object sender, MouseEventArgs e)
        {
            double i = e.GetPosition(image1).X;
            double j = e.GetPosition(image1).Y;
            if (i > 640 || j > 480 || i<0 || j<0)
                return;
            label1.Content = String.Format("distance at ({0}, {1}) is {2} millimetres.",
                i, j, rawDepthData[640 * (int)Math.Floor(j) +
                (int)Math.Floor(i) - 1] >> DepthImageFrame.PlayerIndexBitmaskWidth);
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            using(TextWriter tw = new StreamWriter(textBox1.Text))
            {
                foreach(short val in rawDepthData)
                {
                    tw.WriteLine(val >> DepthImageFrame.PlayerIndexBitmaskWidth);

                }
                tw.Flush();
                tw.Close();
            }
            FileStream stream = new FileStream(textBox1.Text + "_depth" + ".png", FileMode.Creat
            BitmapEncoder encoder = new PngBitmapEncoder();
            encoder.Frames.Add(BitmapFrame.Create(image));
            encoder.Save(stream);
        }

        private void button2_Click(object sender, RoutedEventArgs e)
        {
            FileStream stream = new FileStream(textBox1.Text + "_color" + ".png", FileMode.Creat
            BitmapEncoder encoder = new PngBitmapEncoder();
            encoder.Frames.Add(BitmapFrame.Create(imgFrame));
            encoder.Save(stream);
        }

        private void button3_Click(object sender, RoutedEventArgs e)
        {

            /*TimerCallback callBack = new TimerCallback(OnTick);
            m_Timer = new Timer(callBack, null, 0, 2500);
            /*timer = new Timer(1000);
            timer.Elapsed += new ElapsedEventHandler(timer_Elapsed);
            timer.Enabled = true;
            timer.Start();*/
            }
   }
}
```

Chapter 8

Reference

- R. C. Smith and P. Cheeseman. On the representation and estimation of Spatial uncertainty. Int. J. of Robotics Research, 1986.
- H. Durrant-Whyte, Tim Bailey. Simultaneous Localisation and Mapping(SLAM):Part I The Essential Algorithms.
- S. B. Willimas, G. Dissanauake, H. Durrant-Whyte. An Efficient Approach to the Simultaneous Localisation and Mapping Problem. Australia, 2006.
- J.A. Castellanos, J.M.M. Montiel, J. Neira, and J.D. Tar-dos. Sensor in?uence in the performance of simultaneous mo-bile robot localization and map building. In P. Corke and J. Trevelyan, editors, Experimental Robotics IV, pages 287-296. Springer-Verlag, 2000.
- M.W.M.G. Dissanayake, P. Newman, H.F. Durrant-Whyte, S. Clark, and M. Csorba. An experimental and theoreticalinvestigation into simultaneous localisation and map building.Experimental Robotics IV, pages 265-274, 2000.
- H.J.S. Feder, J.J. Leonard, and C.M. Smith. Adaptive mo-bile robot navigation and mapping. International Journal of Robotics Research, Special Issue on Field and Service Robotics, 18(7):650-668, 1999
- J.J. Leonard and H.J.S. Feder. A computationally e?cient method for large-scale concurrent mapping and localization. In Proc. Ninth International Symposium on Robotics Research, pages 169-176. International Foundation of Robotics Research, 1999

- P. Newman. On The Structure and Solution of the Simultaneous Localisation and Map Building Problem. PhD thesis, University of Sydney, Australian Centre for Field Robotics, 1999.

- S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. Machine Learning and Autonomous Robots (joint issue), 1998.

- S.B. Williams, G. Dissanayake, and H.F. Durrant-Whyte. To-wards terrain-aided navigation for underwater robotics. Ad-vanced Robotics, 15(5):533-550, 2001.

- M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges.

- G. Dissanayake, S. Huang, Z. Wang, R. Ranashinghe. A Review of Recent Developments in Simultaneous Localization and Mapping. Sri Lanka,2011.

- M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem.

- G. Hu, S. HuANG, l. Zhao, A. Alempijevic, G. Dissanayake. A Robust RGB-D SLAM Algorithm. Int. Con. On Intelligent Robotics and Systems, 2012.

- A. T. Norton, A. R. McCook. Implementation of Simultaneous Localisation and mapping algorithm in an Autonomous Robot.2012

- J. E. Guivant and E. M. Nebot, ”Optimization of the simultaneous localization and map building (SLAM) algorithm for real time implemen-tation,” IEEE Transactions on Robotics and Automation, 17(3):242-257, 2001.