Determining the Viability of the Kinect as a Research Tool

by

Matthew S. Roscoe, Paul G. Plöger,

Kenneth B. Kent and Rainer Herpers

TR12-215, **January 23 2012**

Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
Email: fcs@unb.ca
http://www.cs.unb.ca

Table of Contents

List of Figures, Tables & Equations

List of Figures

List of Tables

List of Equations

Acronyms

Acronym	Definition
3D	Three-Dimensional
ARRSAC	Adaptive Real-Time Random Sample and Consensus
BRSU	Bonn-Rhein-Seig University
CPU	Central Processing Unit
CUDA	Compute Unified Design Architecture
GFLOPS	Giga-Floating Point Operations Per Second
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
IR	Infra-Red
LEL	Least-Entropy Like
NUI	Natural User Interface
PCL	Point Cloud Library
RANSAC	Random Sample and Consensus
RGB	Red-Green-Blue
RGBD	Red, Green, Blue, Depth
ROS	Robot Operating System
SLAM	Simultaneous Location And Mapping
STL	Standard Template Library (C++)

Abstract

This project investigated the viability of using the Microsoft Kinect in order to obtain reliable Red-Green-Blue-Depth (RGBD) information. This explored the usability of the Kinect in a variety of environments as well as its ability to detect different classes of materials and objects. This was facilitated through the implementation of Random Sample and Consensus (RANSAC) based algorithms and highly parallelized workflows in order to provide time sensitive results. We found that the Kinect provides detailed and reliable information in a time sensitive manner. Furthermore, the project results recommend usability and operational parameters for the use of the Kinect as a scientific research tool.

1.0 Introduction

In the field of robotics a common problem is attempting to understand the world or environment in which the robot is operating. This is a common issue, as robots do not have an "intuitive" sense about its environment. Environment segmentation is a technique that is used to allow for the isolation of different parts of an environment for identification or interaction.

Beyond having to deal with perceiving an environment that a robot is embedded within, there is the problem of having to do so in a reasonable amount of time. This means that there is a real-time constraint placed on this system. With service robots operating in ever changing environments the system must be able to perceive these changes fast enough to ensure that the robot has enough time to appropriately react to the change.

The need to perceive an environment and to do so in a time sensitive manner is demonstrated in the RoboCup Service Robotics Challenge [**1**]. This challenge requires robots to perform various tasks autonomously, such as fetching a drink from the kitchen. While this task sounds simple in nature it requires not only knowing what a kitchen and sink are; but how to find them and how to recognize that it has found them. While performing this task the robot must also be aware of people passing in front of it, or a new obstacle that may appear, creating a hazard for the robot or the people working around the system.

This paper will look at the ability to discover planar surfaces in an environment, as most objects within the environment we are concerned with are located on planes. The ability to locate planes provides the system with a searchable area within the environment. The focus on finding planes will allow the evaluation of the feasibility of several possible improvements to environment segmentation.

This project will have a scope that is limited to looking at the RANSAC algorithm, the Microsoft Kinect as well as parallel technologies in order to improve upon environment segmentation.

2.0 Environment Segmentation

2.1 What is Segmentation?

For this project we define segmentation as the ability to partition information within data sets, images, etc. It allows the distinction of a select portion or item within a scene. From there tasks such as classification can be performed.

In robotics it's not necessarily the entire scene that is important but rather all of the small pieces that make up the environment that the robot is working in. A robot will rarely interact with the whole environment but rather, will chose parts of the environment to interact with or perform given tasks within.

In order to interact with these segments of the environment or objects within them, the robot must be able to find them. This is currently done though various techniques of segmentation, which will be discussed in the upcoming sections.

2.2 Why Segment Environments?

Environment segmentation is a crucial ability within robotics and computer vision. The problem is currently that the technology is lacking in order to perform the task in a real time manner. This paper will investigate several upcoming changes that can allow for new approaches to this problem. These approaches are not only algorithmic in nature but in the information that can be gathered by performing segmentation.

Environment segmentation provides us with the ability to find "objects" within an environment. Once we are able to find objects we can then progress to determine what these objects are and how to interact with them. While this is something that comes very instinctively to humans, this is a complex task for computers to perform. This is the case due to the large quantity of information about the environment that needs to be processed in order to segment the environment in a meaningful manner. The amount of information being processed leads to slow algorithms, which make environment segmentation a difficult task to perform in a time sensitive manner.

It becomes even more complex when dealing with three-dimensional information. The added dimension not only increases the amount of data present but also how this data is processed. While a third dimension does add additional complexity to the problem it is also the feature that makes environment segmentation valuable to robotics.

2.3 Requirements on Environment Segmentation

In order for environment segmentation to be a valuable ability it must meet certain requirements. These requirements are based on the running time of the solution as well as the reliability of the solution.

The proposed solution must be able to perform the environment segmentation in real-time, which will be defined as an operational running time of 30Hz or 33ms per calculation. This requirement exists as service robots are operating within environments where what exists in the environment and its properties can change rapidly. If the solution is unable to perceive these changes quickly enough to react to them the solution is not sufficient, as well as not safe.

The solution must also be reliable when being used. This means that planes must be detected as often as possible with as small of a false positive rate as possible. The unreliable detection of planes would lead to the solution providing invalid information. Providing invalid information could go on to create safety issues if it is used for path planning or robotics arm movement decisions.

3.0 3D Environment Data Acquisition

3.1 Required Information

In order to perform environment segmentation we must first have an appropriate representation of the environment. This project employed the use of Willow Garage's Point Cloud Library (PCL) [**2**] to serve this purpose.

The point cloud library is a C++ template based library which focuses on the storage of 3D information within a data structure called a Point Cloud. PCL also provides users with many algorithms, which allow operations to be performed on said point clouds. PCL is also starting to provide users with the ability to take advantage of GPGPU algorithms within it's libraries.

PCL is used in this project for data storage due to its broad use within the Robot Operating System (ROS) and its use in the field of robotics. Rather than focusing on the version of PCL included within ROS, the new stand-alone library will be the version employed. This is due to the fact that the stand-alone version is heading towards implementing parallelization in the libraries, which is something that must be leveraged for this project.

Point Clouds allow for the storage of information such as the X,Y and Z coordinates of the points in the cloud. We can also store information about the size of a point, its color and other pieces of information.

3.2 Devices for gathering 3-Dimensional Environment Data

Computer vision started out using photo's and images that at the time were captured using conventional two-dimensional methodologies. In the last few years the capture of three-dimensional information has been slowly becoming more commonplace, especially for use in robotics. As such there has been a slow evolution of capturing 3D information.

Introduced around the 1950's the stereo camera has become one of the more common methods for capturing 3D information. These systems function by taking two images from slightly different angles and then merging them together [**3**]. This is done in order to replicate human binocular vision. Stereo depth vision is obtained by taking two images, which have been taken from two cameras on the same plane separated by a known distance. When you find a similar point on both images by knowing the distance between the cameras and the angles between them you can extract the location of this point in 3D space.

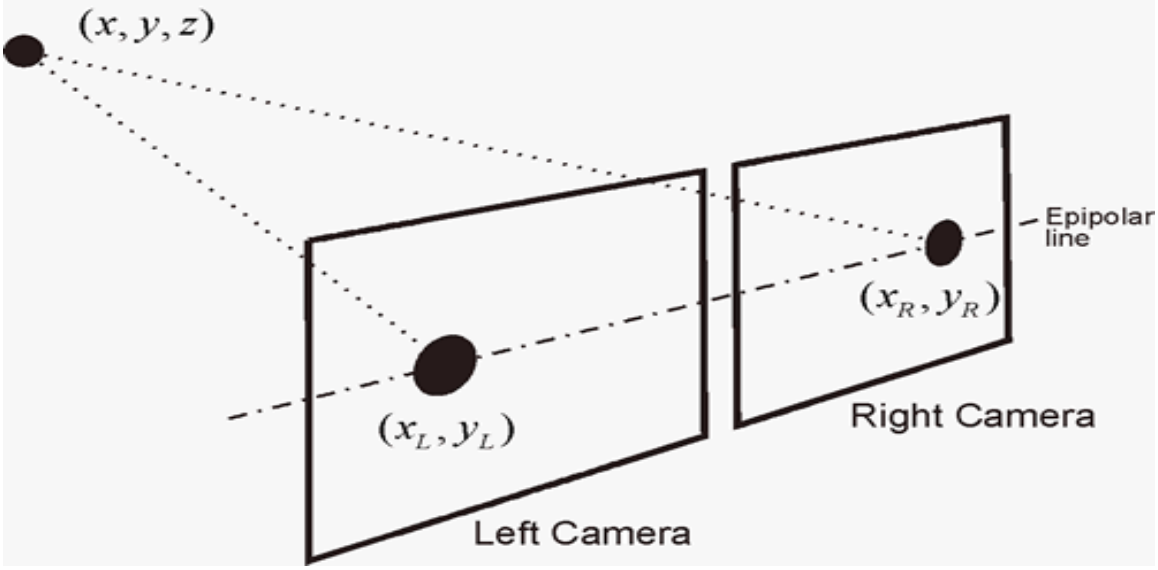


Figure 1 - Basic Stereo Vision Setup [4]

More recently 3D scanning lasers or 3D scanners are becoming more common due to their steadily improving capabilities and performance. These cameras operate in a multitude of manners each of which employ the emission of a signal normally through a laser and capturing and processing that signal [**3**]. While these methodologies produce more precise data as noted by Bernardini et al:

> The current state of the art allows the acquisition of a large class of objects, but requires expert operators and time consuming procedures for all but the simplest cases [**3**].

There is now a new entrant into the field of 3D information gathering and that is in the form of the Microsoft Kinect [**5**]. The Kinect is a new three-dimensional data capture device that combines both RGB and Depth into a single data source, and it does so at a fraction of the cost of the other systems.

3.3 Why use the Kinect?

This project will focus on the newest entrant to the 3D data capture field, the Microsoft Kinect. The other sensors are well understood and so are their operational limitations. The functionality of the Kinect will be examined and compared to the stated abilities as

published by Microsoft [5]. The system will be evaluated on its applicability in robotics as a replacement for current technologies. The secondary motivation for the focus on the Kinect is that it has a low procurement cost which could provide those abilities to a greater audience.

3.4 The Microsoft Kinect

The Microsoft Kinect is a senor suite based around the PrimeSense design, which allows it to provide depth, RGB, infrared and audio information to an end user of the product. Although originally designed as a controller for the Microsoft XBOX 360, it is fast becoming a well-known 3D data capture device [6]. This has occurred due to how easy it is to use in a multitude of fields, as well as its very low cost compared to the amount of information that you can obtain from it.

3.4.1 Kinect Usability

This section will cover the usability of the Microsoft Kinect within service robotics and the predefined usable range for the Kinect. Microsoft has stated that the Kinect can see a usable range from the front of the sensor from zero meters up to a maximum usable distance of five meters [5].

The range of view of the Kinect is a 57° horizontal field of view with a 43° vertical field of view. The motorized tilt of the sensor allows for ±28° of movement in the vertical axis [5].

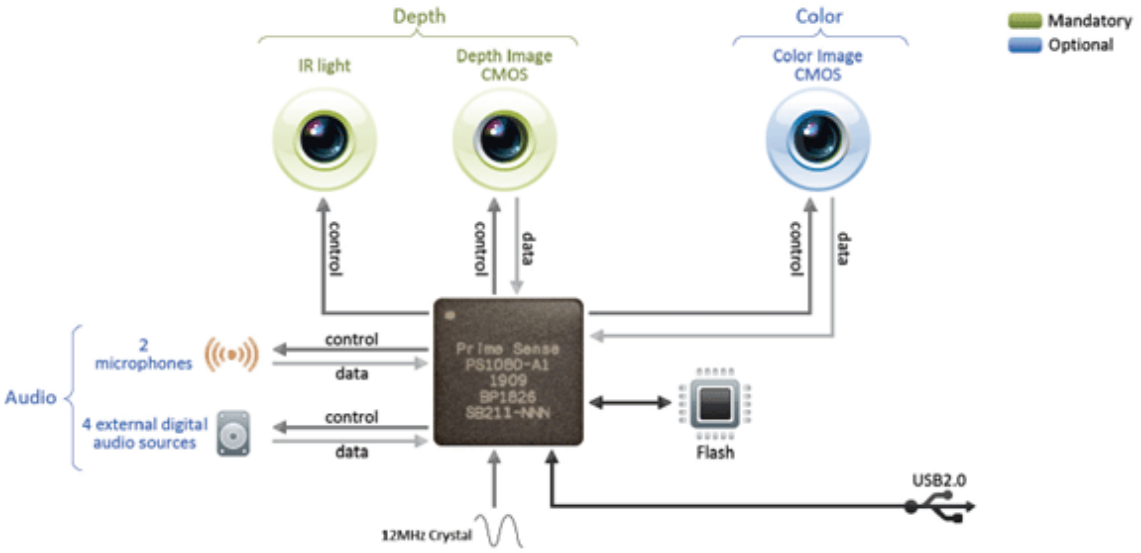


Figure 2 - Kinect Hardware Setup [7]

Within these fields of view they are resolved to a 640x480 pixel image that is delivered to the recipient. The data is in fact captured at 1280x1024 but the algorithm operating within the

Kinect compresses the data in order to allow for transmission at 30fps [**5**]. It is also possible to capture information from the Kinect using the OpenNI [**8**] backend libraries. These libraries allow for the collection of the full 1280x1024 pixel image but at a slowed frame rate of only 10Hz [**9**].

The maximum resolution of the Kinect is to resolve 1mm to a single pixel starting at a range of 0.8m. The depth image is mapped to a 320x240 pixel image further reducing the allowable resolution and capping how precise the Kinect is. This 320x240 pixel image is then mapped to the provided 640x480 pixel RGB/IR image before passing to the end user [**5**].

3.4.2 Added Depth

An important property of the Microsoft Kinect is that it allows for the user of the system to obtain a depth map of the area in question. It was possible before to capture depth information but it was costly and typically came from laser depth sensors. The Microsoft Kinect allows for the user to collect such information in a very easy to use manner.

The Kinect is theorized[1] to determine depth by projecting an infrared pattern, which its IR cameras are able to see [**10**]. It then compares the captured image to reference images looking for the amount of distortion present between the dots.

Figure 3 - Microsoft Kinect IR Speckle Pattern [11]

This process is known as structured light depth capture and has been widely used within 3D depth scanners mainly employing the use of lasers. While the details vary the basics of how

[1] Microsoft & PrimeSense will not release any of the technical documentation on how exactly the sensor system actually obtains its distance information thus how it works may only be guessed at based on observations of how it works.

structured light works is the same. An emitter, normally a laser, or in the case of the Kinect an IR emitter will emit a pattern onto a surface.

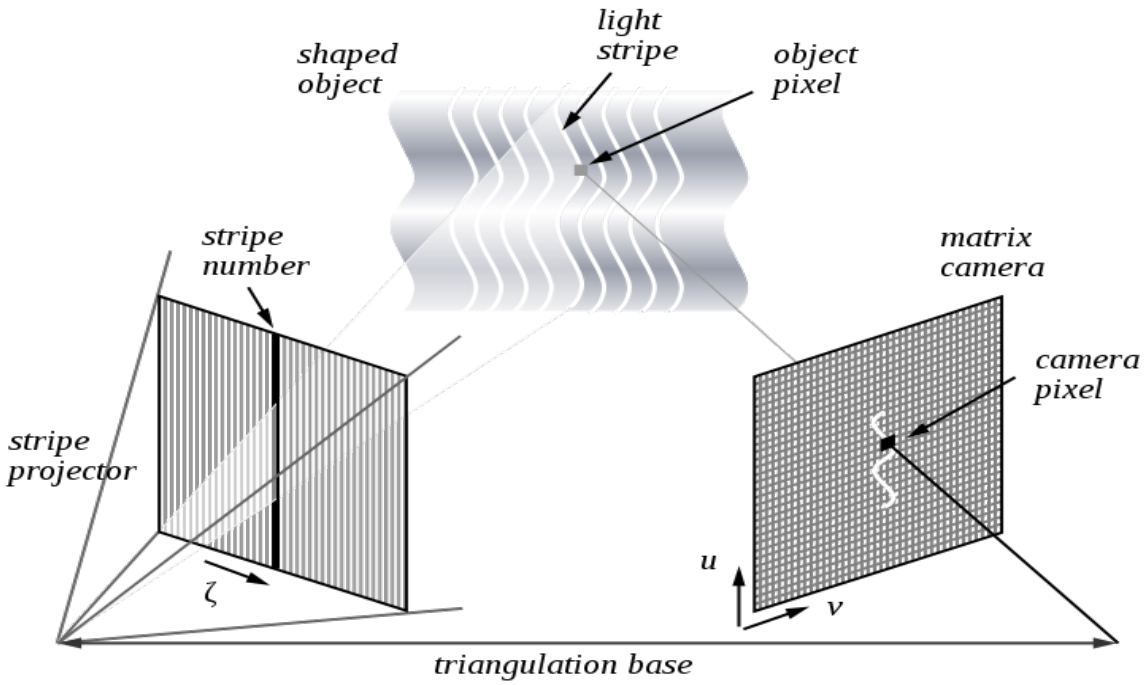


Figure 4 - Structured Light Capture Methodology [12]

A camera will capture the result of this pattern on the surface in question. Based on the amount of distortion a variety of algorithms can provide information on everything from the location of the surface to the details of the surface depending on the prevision of the emitter and the resolution of the capturing camera [**13**].

3.4.3 Drawbacks

While the Kinect is a cost efficient and readily available alternative to the standard 3D data capture devices, it does come with a few drawbacks. These drawbacks affect both the performance and usability of the Microsoft Kinect within the service robotics industry.

One of the major drawbacks to the Microsoft Kinect is the resolution of the IR sensor used to capture the aforementioned depth information. This sensor operates at a resolution of 320x240 pixels [**5**], which is then mapped to a 640x480 pixel RGB image. This means that each RGB pixel does not have a unique depth value. This leads to a "neighborhood" of pixels in an RGB image being assigned a non-unique depth value, which may or may not truly represent the distance of that pixel from the camera.

The other drawback to the Kinect is the resolution that is being provided. A 640x480 pixel does not provide a large amount of information in a single glance; especially when that image represents a small subset of the information actually present in the object. The lack of refinement or precision limits what information that can be obtained from these images in

the future. The ideal solution would allow for operation in a setting where the most amount of information about an environment can be obtained from the provided data capture.

4.0 Algorithms for Processing Environments

4.1 Methods used for Segmentation

There is a set of well-studied approaches for dealing with segmentation. They range from techniques, which search for a common quality between partitions, to model fitting.

Clustering is the technique of grouping items together based on a certain quality. In the case of images this can be color, intensity, or other qualities of the image. With RGB---D images we could also cluster based on the depth value of a pixel. While this may seem like a good way to approach environment segmentation it fails to deal with objects where the color or intensity change but where this happens within a single object.

This paper will focus on the model fitting type of algorithms. This is due to the fact that a mathematical model of that object can represent what we are searching for within the environments. With this model methodology in mind this paper will focus on RANSAC, which allows for hypothesis to be create about what might fit that model and to test them finding the best fit in that situation.

4.2 Why RANSAC?

This paper will focus on the RANSAC algorithm due to its performance metrics. This is not a reference to its run time but rather to its ability to determine positive correlations within data sets with very high levels of contamination or outliers [**14**].

This feature of RANSAC is extremely important for environment segmentation due to how much data is actually gathered in order to perform such a task. For example: a service robot is searching a kitchen for a desired cup. This cup will make up a very small percentage of what the robot is seeing. If the algorithm cannot handle situations where the desired data makes up a very small portion of the total data set the robot would never be able to discover the cup.

4.3 RANSAC

One well-studied and successfully applied parameter estimation algorithm was first proposed in 1981 and was called Random Sample Consensus (RANSAC) [**15**]. This algorithm works very well in noisy data sets as it can deduce planes in environments where the points that make up the plane make up far less than 50% of the entire data set. While this is very desirable especially with cameras that may produce very noisy data streams; it comes with runtime drawbacks. As noted in [**16**] [**17**] the RANSAC algorithm has no problem finding planes, but the hypothesis testing this algorithm performs requires a great deal of time.

The RANSAC algorithm operates by proposing a number of hypotheses. These hypotheses are predications of inliers to a plane (or any provided model) and from there, points are taken around the hypothesized inliers and tested to see if they fit in the predicted plane. This is done in an iterative manner, with each hypothesis fully tested before moving onto the next hypothesis.

The concept of hypothesis prediction is used to detect planes in that the algorithm hypothesize various models which may represent a plane within the provided data. It will then process the environment information checking to see if enough points conform to the proposed model to declare that model a planar surface.

```
input:
    data - a set of observations
    model - a model that can be fitted to data
    n - the minimum number of data required to fit the model
    k - the number of iterations performed by the algorithm
    t - a threshold value for determining when a datum fits a model
    d - the number of close data values required to assert that a model fits well to data
output:
    best_model - model parameters which best fit the data (or nil if no good model is found)
    best_consensus_set - data point from which this model has been estimated
    best_error - the error of this model relative to the data

iterations := 0
best_model := nil
best_consensus_set := nil
best_error := infinity
while iterations < k
    maybe_inliers := n randomly selected values from data
    maybe_model := model parameters fitted to maybe_inliers
    consensus_set := maybe_inliers

    for every point in data not in maybe_inliers
        if point fits maybe_model with an error smaller than t
            add point to consensus_set

    if the number of elements in consensus_set is > d
        (this implies that we may have found a good model,
        now test how good it is)
        this_model := model parameters fitted to all points in consensus_set
        this_error := a measure of how well better_model fits these points
        if this_error < best_error
            (we have found a model which is better than any of the previous ones,
            keep it until a better one is found)
            best_model := this_model
            best_consensus_set := consensus_set
            best_error := this_error

    increment iterations

return best_model, best_consensus_set, best_error
```

Equation 1 - Standard RANSAC (pseudo code)

4.4 RANSAC Variants

Raguram, Frahm and Pollefeys in their 2008 paper covered a large amount of the then current RANSAC variants. This covered the algorithms that attempted to speed up RANSAC through discarding bad hypothesis early on in the evaluation process. This included the $T_{d,d}$ Test by Matas and Chum [**18**], Capels expansion on the $T_{d,d}$ Test better known as the Bail-Out Test [**19**] all attempted to take advantage of discarding bad hypothesis

early in order to speed up computation. While this did improve performance it was not a great enough effect for real-time time constraints.

The other conventional direction that researchers followed was to attempt to improve the hypothesis generation that is required when running RANSAC. Raguram, Frahm and Pollefeys also cover several of these algorithms as well. The PROSAC [**20**], MLESAC [**21**] and Lo-RANSAC [**22**] all provided faster than RANSAC running times, but as presented their times still fell short of running in 33ms or less [**14**].

4.4.1 Least-Entropy-Like (LEL)

The LEL algorithm is another variant on the traditional RANSAC algorithm. While this algorithm does show the ability to quickly detect planes within an environment with very little error rate, it does still have the drawback on its runtime requirements. While markedly much faster than RANSAC, it is still running with a minimum runtime of six seconds according to the information provided [**17**].

4.4.2 Real-Time Plane Segmentation

Most recently Dirk Holz has presented a new and novel algorithm based on RANSAC that operates in a Real-Time time constraint. While this algorithm does hold up well in terms of finding planes and false positives and does so in a real time environment of about 30Hz. The problem with this algorithm is that in order to operate at 30Hz it processes at a resolution of QQVGA [**16**] where the data coming from the Microsoft Kinect comes in at a VGA resolution [**23**].

While the lower resolution allowed this process to detect objects and obstacles within the real-time constraint, it did so by thinning out the environment. While finding planes may not require the full 300,000 points with an eye on expanding beyond planes the proposed solution needs to be able to find more complex shapes in large environments. If these objects are small the more points we can process the more likely that we can correctly identify these objects in the environment.

4.4.3 Adaptive Real-Time Random Sample Consensus (ARRSAC)

In 2008 Rahul Raguram, Jan-Michael Frahm and Marc proposed ARRSAC [**14**]. While this algorithm is mainly designed for use in two-dimensional environments, it should have no problem being ported from a 2D world into a 3D environment. The discovered improvements of anywhere from 11x faster than RANSAC to 305x faster than RANSAC [**14**] if they hold in three-dimensions would allow for an algorithm which could easily live within a real-time constraint of operating at speeds of 30Hz or higher.

4.5 Why use Standard RANSAC?

This paper looks at the standard implementation of RANSAC due to upcoming advancements in the parallelization of this algorithm within the Point Cloud Library. With run times that are well within the real-time constraint, we can focus on the robustness and usefulness of this algorithm in association with the Kinect.

This project will use the PCL parallel implementation of RANSAC found within the development branch of PCL. This implementation of RANSAC is a parallelized version of the standard RANSAC algorithm that was presented above.

The focus on the parallel version will allow the demonstration of RANSAC as a viable candidate to operate within a real-time constraint. The RANSAC variants presented were all proven by their authors to operate at faster than RANSAC times. With this in mind proving that standard RANSAC in a parallel implementation can satisfy our requirements, a parallel implementation of any of the above algorithms would also satisfy the requirements for this system and more.

5 Improving Environment Processing

5.1 Parallel Computing

Computing today is most commonly performed on Central Processing Units (CPU's), which operate in a serial manner. The common exception to this rule is video games where the graphics are processed on the Graphics Processing Unit (GPU). GPU's, unlike their CPU counterparts, operate in a parallel manner taking multiple commands simultaneously.

This is an extremely useful tool in two methods; GPU's are designed for vector and matrix mathematics and are very quick at these calculations. Secondly where we can run multiple processes simultaneously, we can take advantage of having more power to compute with. There have been arguments about how much faster GPU's are compared to CPU's with NVIDIA stating 100x and Intel only claiming they are 14x faster [**24**].

5.2 NVIDIA Compute Unified Device Architecture (CUDA)

NVIDIA is one of the world's largest producers of GPU's and specializes in parallel processing architectures. For as long as GPU's have been produced programmers have been trying to take advantage of their abilities. The problem with GPU's is that in the beginning they were designed for very mathematical operations on data that was provided in forms of matrices or vectors.

There has been a recent push by both Apple & AMD and NVIDIA towards GPGPU's or General Purpose Graphics Processing Units. Apple & AMD have been pushing OpenCL an API that sits on top of all video cards to allow for general programs to run on any card that supports this standard [**25**]. NVIDA has been pushing CUDA which works only with NVIDIA CUDA [**26**] enabled cards allowing for more control over the API and what one

can do with it. We chose to examine CUDA vs. OpenCL due to the noted better speeds and abilities associated with CUDA [**27**].

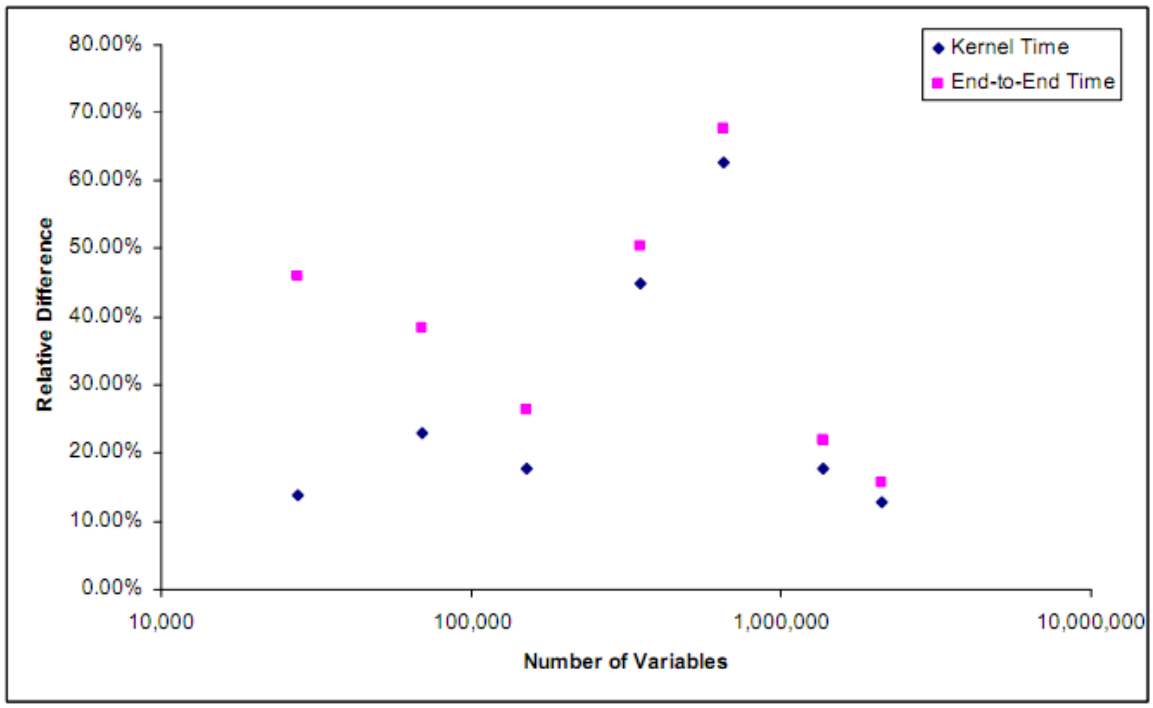


Figure 5 - Relative running times between CUDA and OpenCL [28]

As noted by Karimi et al, there is a notable running time difference between CUDA and OpenCL. In the case of their evaluation they found that CUDA was on average 13% to 63% faster than OpenCL in kernel execution times [**28**]. When end---to---end runtimes for CUDA and OpenCL were examined by Karimi et al, it was discovered that CUDA was between 16% and 67% faster than OpenCL [**28**].

Recently CUDA has leveled the playing field between CPU and GPU programming. Most programmers with a little training can now tackle what was once a specialty field. CUDA 4.0 also introduces the use of the Thrust library[2] providing programmers with access to the well-known functions within Standard Template Library (STL) that are optimized to run on parallel platforms such as those provided by NVIDIA [**29**].

[2] The Thrust Library is only available for NVIDA devices at the time of writing.

5.2.1 Speed

The main reason for diving into the world of GPGPU programming is the requirement for speed within robotics. As can be seen in Table 1 - CPU vs. GPU Computation Times", the CPU computation time for finding a single plane within a processed[3] point cloud averages 50ms. With a real-time restriction of finding n planes within 33ms CPU like speeds do not suffice, especially when we consider we could be searching for up to 10 planes.

GPU's have over the past 8 years started to differentiate themselves from CPU's in how quickly they can process data. While CPU's have been advancing at a steady pace, GPU's have been advancing in leaps and bounds, pushing the boundaries of what one can compute.

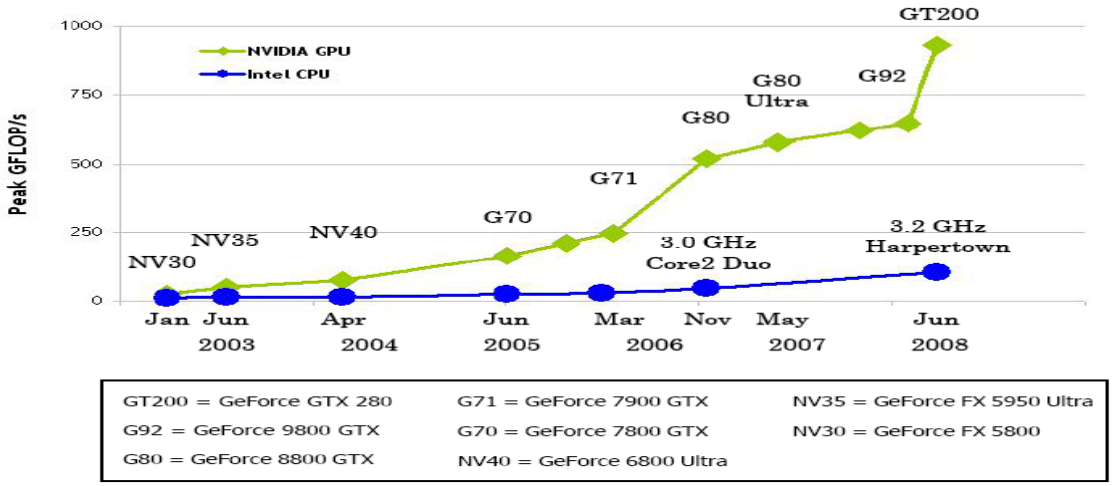


Figure 6 - CPU vs. GPU Computing Speeds [30]

The above graph depicts the amount of computation power gain in both CPU's and GPU's in a period from Jan 2003 until June 2008. While CPU's gained an improvement of nearly 100 GFLOPS GPU's improved at an astonishing rate of an increase that is greater than 900 GFLOPS. This impressive increase in speed has come with very little, if any increase in cost from one generation to another. The ability to harness these impressive speeds would lead to being able to perform environment segmentation in real time with computational cycles to spare.

5.2.2 Usage

One of the biggest advancements in the most recent version of CUDA has been the increase of usability of GPU's. Version 4.0 of CUDA brings with it the support of the Thrust Library, which brings the basic functionalities of STL to parallel architectures [**29**].

[3] Processed Point Clouds refer to point clouds who have had their field of vision restricted and statically down sampled the data to a point where we can use it. The average reduction would be 300000 points to 15000 points.

This is a major advancement for the usage of GPGPU's as they are no longer niche products that a select number of people can program. More importantly than just being available to a select group, they are becoming more and more general. This is allowing more and more code to be ported from CPU to GPU code and to take advantages of the abilities that the GPU can provide.

5.3 Parallel PCL

Parallel PCL is an upgrade to the standard point cloud library that allows for it to take advantage of NVIDIA's CUDA technology. The combination of PCL and parallel architectures allow for a great advancement in the speed and stability of the programs running on the PCL framework.

Task	CPU Computing Time (ms)	GPU Computing Time (ms)	Improvement
Normal Vector Computation (Simple)	288.7700	0.0968	2983.16
Normal Vector Computation (Complex)	597.4058	0.0984	6071.20
All Vector Computation (Simple)	299.1001	53.8019	5.56
All Vector Computation (Complex)	608.0731	43.7823	13.89
Single Plane Segmentation	50 – 80	3– 7	13.00

Table 1 - CPU vs. GPU Computation Times

As demonstrated in the table above there is marked improvement between CPU and GPU performance within the PCL libraries. Besides the speed improvements between CPU and GPU performances, is the scaling within the tasks performed. The environments used for these speed tests are demonstrated below.

Figure 7 - Environments used for Speed Testing

The above environments were used for the calculation of CPU and GPU times. The reasons for these scenes are that the empty scene provides lots of flat surfaces making normal predication fairly simple. The more cluttered scene has lots of curves and odd shapes making

normal estimation more difficult. This in theory increasing the amount of work that must be performed to find the solution.

6.0 Experimental Process & Design

For the experiments the parallel implementation of the standard RANSAC algorithm will be used. This implementation will come from the point cloud libraries development branch. There will be two sets of tests, which will look at several variables in two different environments. The first is the BRSU RoboCup lab and an outdoor environment on the BRSU campus.

For each test the running time of the RANSAC algorithm will be compared in both CPU and GPU running times. For experiments where recording the running time does not make sense, images of the results will be compiled in order to illustrate the effect that was examined.

6.1 Experimental Variables

When evaluating the viability of environment segmentation there are many possible things to look at but we will focus on four key points: The range, resolution and the ability for the Kinect to deal with reflective and transparent objects. We will also look at RANSAC's ability to discover planes within varying environments as well as its running times in both CPU and GPU modes of operation.

6.1.1 Range

This project will evaluate the range of useful information that can be gathered. This is not directly related to how far the Kinect can see as this is well documented. It has also been documented that there is a usable range for various activities, particularly those based around people detection.

This project will examine and determine the range in which the Kinect system can provide useful and reliable information to the user. This will be performed with the concept of providing information to service robots and thus a high level of performance and accuracy will be required of this system.

6.1.2 Resolution

This project also explores the Kinect's ability to properly resolve objects within the environment it is operating. For this project planes will be the objects examined, with more complex object types to follow in future work.

6.1.3 Reflective/Transparent Objects

We will look at the ability to use the sensor in combination with a robot in order to identify or interact with transparent or reflective objects/surfaces. The reasoning for looking at these types of objects is because these properties are common in household objects. Mirrors, windows, glasses all contain reflective and/or transparent properties. We must know how the Kinect deals with these objects in order for us to continue to interact with these objects.

If the Kinect is not able to detect objects such as glasses or windows this could prove to be a rather large deficiency in the sensor and must be examined. For these experiments the Kinect's ability to deal with the following objects will be looked at:

1. A glass cup.
2. A glass window.
3. A mirror.
4. A plane wrapped in a reflective material (aluminum wrap).

6.2 Object Segmentation

This requirement is not directly tested as it is required for all of the other tests. Segmentation will be used to deal only with the objects that we want at a given time. Any evaluation of segmentation will fall into one of the other categories.

We are not interested in the ability to "find" an object, but rather under what conditions we can identify objects. The ability to "find" an object lies within the RANSAC algorithm and how it is used. This can be improved or degraded at the behest of the user and thus can change. We will use the standard parallel RANSAC algorithm [**31**].

6.3 Test Constants

6.3.1 What we are looking for

For the testing of the Kinect and the parallel processing of the data we will stick to one of the simpler models for RANSAC, which is to find a planar surface. The environments that we look at will be devoid of any objects that are not planes.

We will also limit the number of discoverable planes to simply find the plane that we are interested in. This allows us to focus on the objects in question and the Kinect's reaction to the different properties of said object.

6.3.2 Environment

For the tests that are undertaken, we will attempt to keep the environment for the segmentations as similar as possible. This will be easier to obtain during tests, which are undertaken indoors. Any tests that are performed outside will be subject to natural environment changes.

6.4 Experimental Setup

6.4.1 Test Machine Specifications

The tests that have been used for the evaluation of the Microsoft Kinect were performed on a NVIDIA Tesla Workstation. The NVIDIA Tesla Workstation is running an Intel i7 950 at 3.06GHz with 12GB of RAM and an NVIDIA Tesla 2070 with 6GB of VRAM.

All tests are run on a fresh install of Ubuntu 10.10 with only the packages required for the testing installed. This includes ROS, PCL, CUDA, and any other dependencies that these packages may require.

6.4.2 Experiment #1 – Indoor Use

The indoor portion will focus on the RoboCup lab at BRSU. This environment has been specifically designed in order to allow for the development of a service robot to participate in the RoboCup challenge. This environment is ideal as it is designed to emulate the every day environment that environment segmentation must be performed in.

The experiment is set up so that the objects are known to be within the viewable distance of the Kinect. They will be observed using the PCL programs in order to find them and to determine how long it takes to perform the computations. The programs that this project used from the PCL libraries were the "Kinect_planes" and the "Kinect_normals" These programs allowed for the computation of normal or the entire process of plane computation. The programs also allow for execution to be hosted on the CPU or GPU allowing for comparison of running times.

Measurements will also be made in a more qualitative manner in order to best determine the usability of the Kinect in a given situation. These measurements are clearly explained as to their nature and the results that they provide.

6.4.3 Experiment #2 – Outdoor Use

The outdoor experiment will contain all of the same tests as the indoor test except that it will be performed in an outside environment. These tests are performed to look at the feasibility of using the Kinect in an outdoor setting. This is important as service robotics become more and more functional; eventually their tasks will branch beyond in house activities to include outdoor activities as well.

7.0 Results

7.1 Experimental Results

While a cost effective device, the lower cost does produce some drawbacks for the Kinect. In the following section the general usability of the Kinect and the drawbacks of using the Kinect will be discussed. Solutions to the drawbacks will also be discussed.

7.1.1 Object Detection

The Microsoft Kinect in combination with the RANSAC algorithm proved very apt at discovering as many planes as it was allowed to detect within an environment. Much like with previous experiments, the more planes that we detect the slower the performance.

The easiest way to improve the performance of the solution is to limit how you search. Limiting the viewing distance is one of the key ways to do this. Limiting viewing distance not only improves the running time of the solution, but will also increase the reliability of object detection.

The reliability of object detection increases with a smaller field of view as the precision of the Kinect increases within a certain range. Within this range objects are reliably mapped in such a manner that we can discover many attributes about the objects such as size, color, etc.

7.1.2 Resolution

One of the biggest questions regarding the Microsoft Kinect is how precise is the sensor. This is a very large question especially when it is vying to replace the more standard laser scanners.

Microsoft states that the usable range of the Microsoft Kinect is from 1.2 meters up to around 3.5 meters. While this may be the case, Microsoft is only performing one task, which is to recognize the human form. This task is more based on the accuracy of the Kinect which as recorded by the ROS team are fairly high with errors ranging from 1mm up to 1cm [**31**].

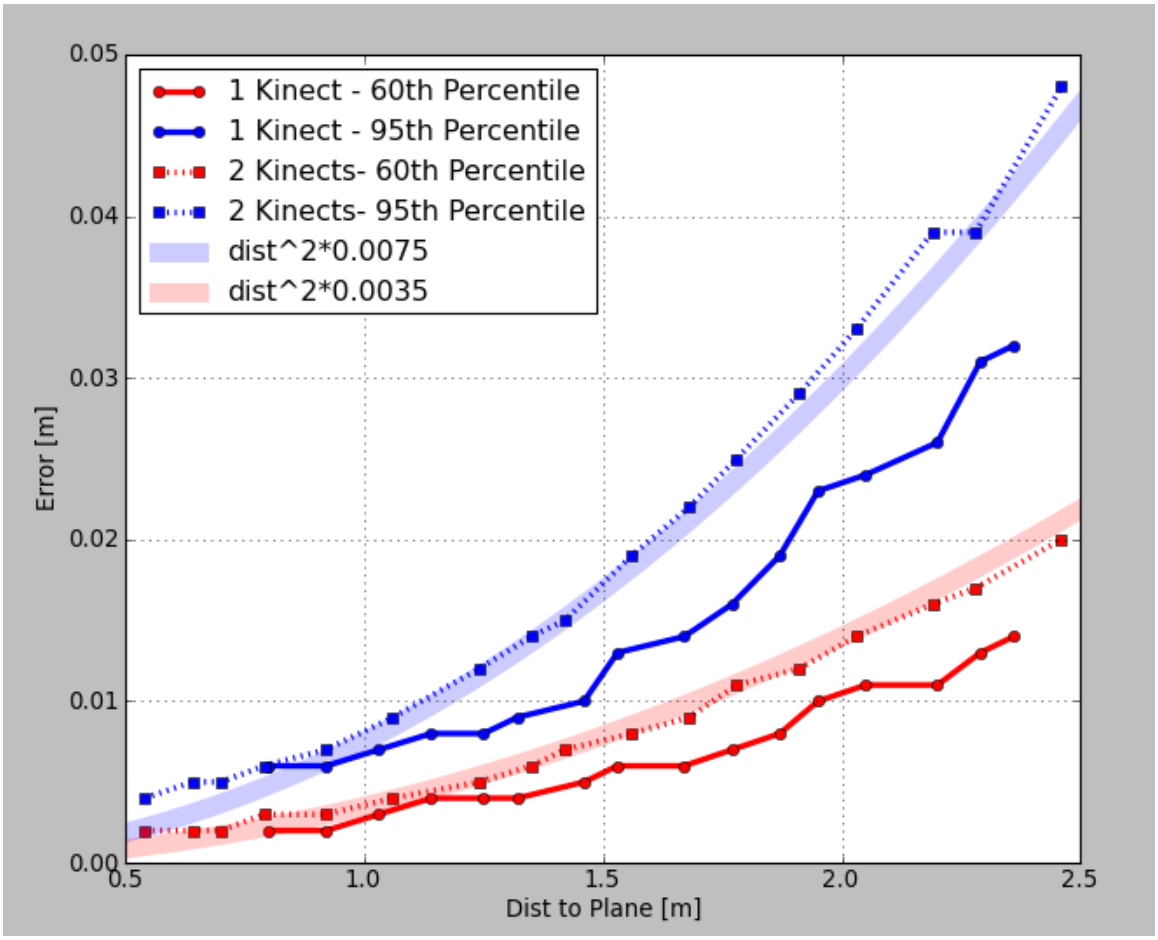


Figure 8 - Kinect Error Ranges [31]

The functionality we are concerned about regarding environment segmentation is the precision of the sensor. That is we can trust the Kinect to return values that are as close to the real "object" as possible. The precision of the Kinect is also the constraining property that details the usable range.

The robot operating system group has found values similar to Microsoft's, stating that the Kinect has a precision of about 0.01m being rendered to 1 pixel starting at a value of about 0.08m [**5**]. From here the precision declines at a fairly predictable manner with a maximum error or discrepancy of 5cm at a distance of roughly 2.5m [**31**] from the sensor itself. This limits the Kinect's functional range to roughly 0.5m to a maximum of 2.5m. This depends on the types of activities being performed, but for large enough objects, 0.03m differences at 2.5m is an acceptable error.

7.1.3 Range Usability

The Kinect solution is useful in most indoor environments and proved useful outdoors but with some drawbacks. Within indoor environments, the Kinect performed reliably detecting the planes that we required it to detect nearly 100% of the time. The same occurred in the

outdoor environments with the biggest drawback being the environment's effect on the performance of the solution.

Due to how RANSAC functions the more data points that are given the longer it takes to run. This is significant in the outdoor environment where there are more individual objects making up the environment. This can be dealt with in two different ways; first you can further reduce the range to block out non-important objects. Secondly you can reduce the density of the provided point clouds to reduce the resolution of the objects and keep processing at the same rate.

This attribute is also important due to the real time constraint that must be covered by the solution. If we cannot process an environment within a usable amount of time, the solution is not a useful one. In an environment that is changing so rapidly, the ability to perceive this change is very important to the safe operation of the system.

One of the major issues with the Kinect is that if we do not limit how far it can see (the range), it will attempt to render everything that it can find. In doing so it allows for users to see a great deal of information but none of the information beyond the 2.5m range is usable when considering distance information.

While the Kinect provides more data, it can only do so in a usable fashion within a fairly small range of the device. If distance about far away objects is required laser scanners and radar are still the preferred solution.

7.1.4 Reflective / Transparent Objects.

When dealing with transparent objects, they provided several pitfalls. The first and most obvious being that they are see through. Much like the human eye, the Kinect sees through the clear surface to what is behind it. Where the Kinect is theorized to use an IR pattern in order to determine distance, this leads to problems depending on the object in question.

With roughly flat surfaces such as windows or glass panes the Kinect sees through them to the objects or surfaces sitting in the background. While this can be very useful, this provides the user with more information than if the Kinect were to detect the object as a solid object. This feature rules out the direct ability to use the Kinect to determine features such as windows or glass surfaces. However, the rest of the information provided could allow for an implicit assumption that there is such an object present.

Figure 9 - "Void" Induced by Clear Glass Cup

When dealing with smaller transparent objects like a clear glass cup other problems occur. As demonstrated above, there is a "void" where the cup is located. This is most likely due to the Kinect itself unable to make any sense of the information that is coming back. Partial distance values can be found if there is an object directly behind the transparent object in question. These values though are not in anyway reliable however and should not be used.

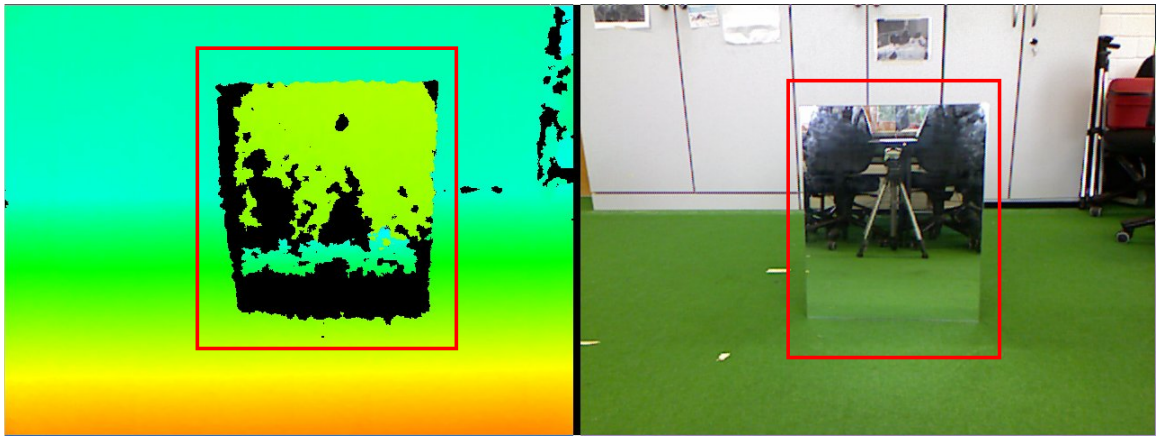


Figure 10 - Depth Capture of a Mirror

When working with reflective objects such as a mirror, depth values are returned but with a great deal of noise as can be seen above. Beyond inducing a large amount of noise, reflective surfaces still provide depth values, but these values are not trust worthy. As can be seen in Figure 10, the ground progresses from a dark yellow to a turquoise the further from the sensor you move. Within the mirror all ranges of this scale are captured in a single surface that is in reality almost uniform in its distance from the Kinect. Like transparent objects we can infer information about the plane in question, but the use of the depth values provided are not suggested.

7.1.5 Indoor vs. Outdoor Usage

One of the major pitfalls is that it preforms fairly poorly in outdoor conditions. This is largely due to the amount of IR information that is present and interferes with the projected pattern from the Kinect.

The most noticeable difference is the distance that the Kinect is able to see. As opposed to an indoor setting in an outdoor one the Kinect's field of view is very restricted. As can be seen below the range of usable information is greatly reduced. The boxes that are visible on the right are placed at one-meter intervals with the 2-4th boxes visible in this shot.

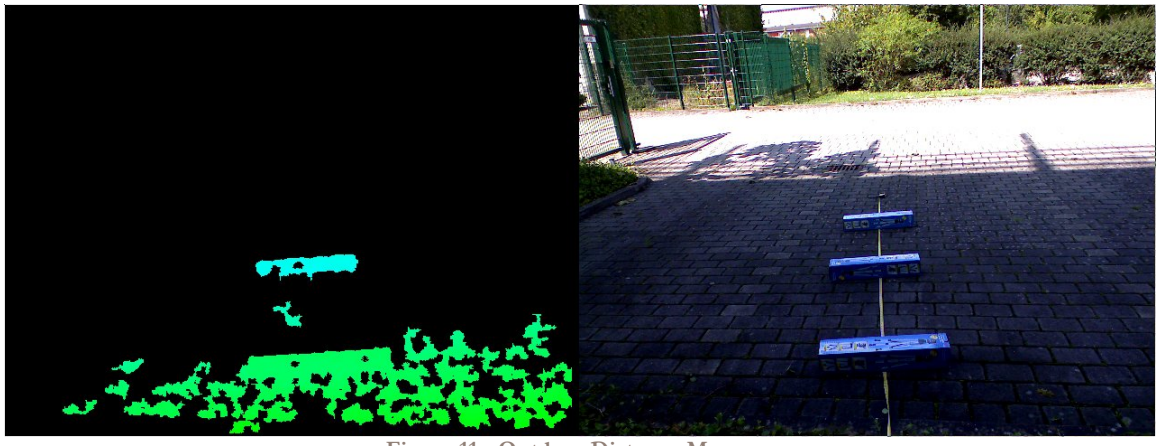


Figure 11 - Outdoor Distance Measure

The Kinect successfully captures the information up to and around the two-meter mark but beside the box at the three-meter mark no more information is captured. This greatly stands out in comparison to indoors usage where information can be captures for very distant objects. The usability of this information is not high but it can still be captured.

Figure 12 - Partial Capture

As mentioned above even when objects are within the range of the Kinect there is interference from the natural environment that ruins data captures. This primarily comes in the form of natural light, which contains large amounts of IR information. As can easily be seen in Figure 13, the IR sensor is over saturated with information not allowing for the Kinect to make sense of the scene.

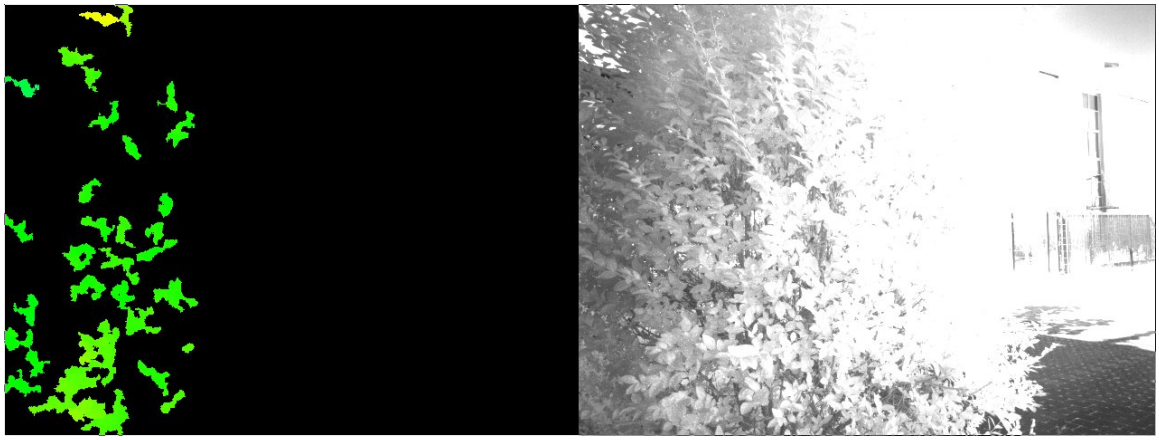


Figure 13 - Outdoor IR Information

With the large amount of interference demonstrated even in a relatively shade covered area the reliability of the Kinect in outdoor circumstances is questionable. While some of the structures can be identified in outdoor environments, most of the information being returned is not useful. This leads to the conclusion that at the current time the Kinect must remain a tool for indoor usage.

7.2 Pitfalls

During the experiments there were a few deficiencies addressed by the performance capabilities of the Kinect used in the experiment. While these are very important points to state, there were a few more intrinsic deficiencies that should be given some coverage.

7.2.1 Noise

The Microsoft Kinect creates a lot of noise when looking at images. As denoted in Figure 14, highlighted in red, we can find some of the areas with the most amount of noise present. This type of noise is typically found around edges that are not within the range of usability of the Kinect. It is where it cannot properly determine where a point belongs or a point is represented as a large point on the mapping.

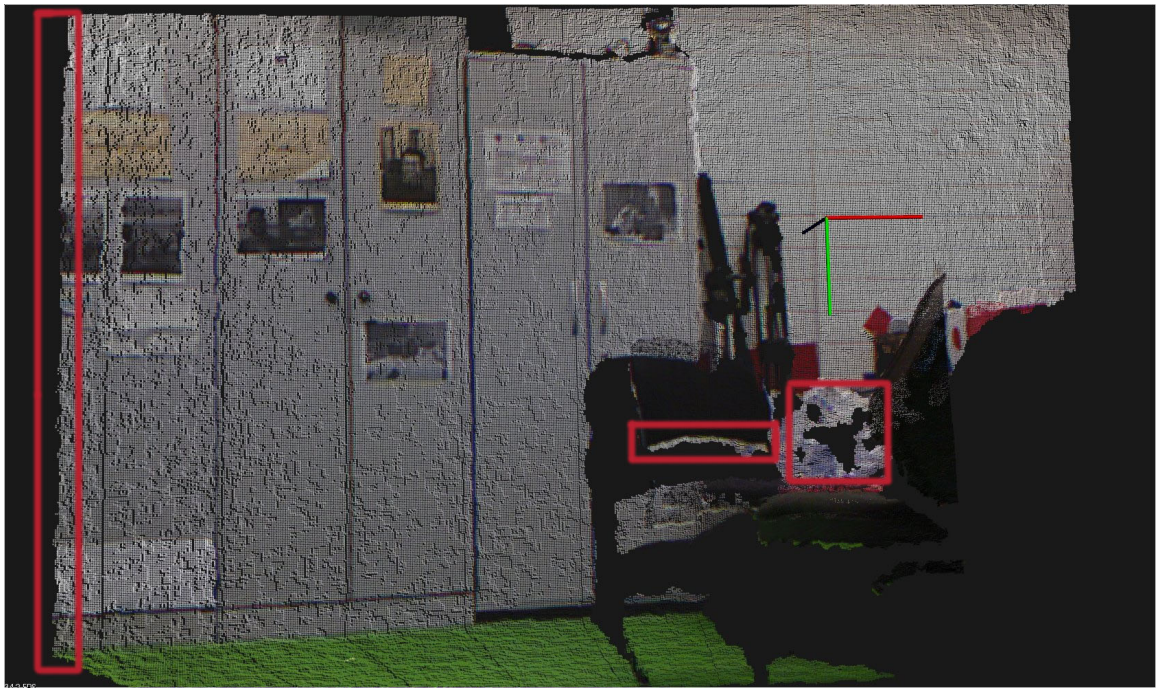


Figure 14 - Noise within Kinect Data Capture

One of the more notable forms of noise is visible in the texture of the cupboards and the floor of figure 14. This is noise from the Kinect's inability to determine if a surface is indeed flat after a certain distance. This can lead to problems when looking for surfaces or objects on a surface that may be parallel to the object in question. The closer the surface is to the sensor the less pronounced this effect, however it is always present.

7.2.2 Shadowing

One of the most notable problems that arose when using the Kinect is the "shadowing" effect. This is when an object in the foreground is blocking information about the information behind the object in question. This is not only a problem in that important information can be hidden from us, but also the shadow continues to extend outward from the face of the Sensor.

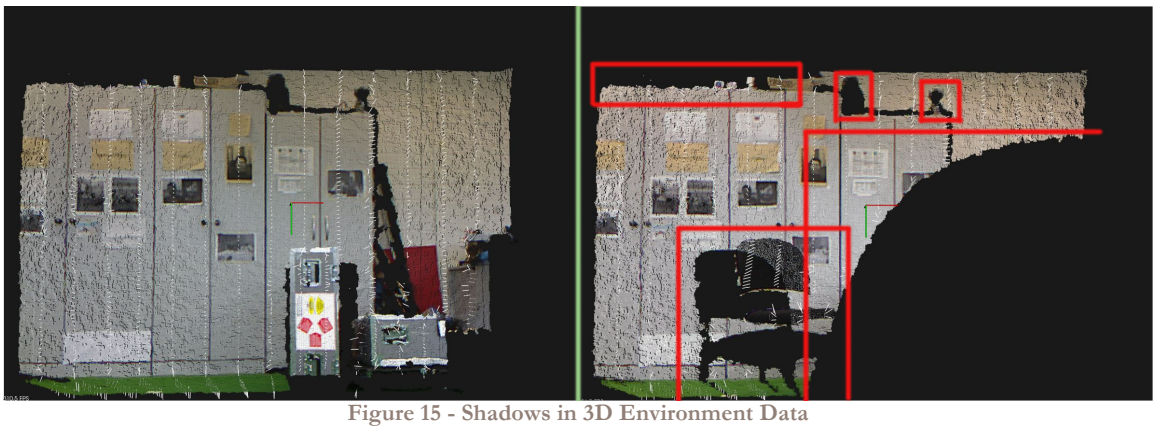


Figure 15 - Shadows in 3D Environment Data

As demonstrated in Figure 15, in red, we note that the amount of data lost is not equal to the objects blocking our view, but also to the projection of them onto any surfaces beyond. While this is a major draw back, it is not a new one, as this problem also exists with laser scanners and standard cameras.

We can counteract the effects of this by building up the environment in that we can keep an 'instant' image that we use for fast computation, but have a full 'map' stored so that we continually add objects in order to provide a full field of reference for the system.

7.3 Parallel vs. Serial Processing

When looking at parallel versus serial computation in regards to its application within RANSAC and environment segmentation parallel implementations win out over serial in many different aspects. The first and most important is that the parallel implementation is the only one where we can work with a full point cloud and still operate within our real-time constraint. This being such a vital requirement of this solution means that parallelized RANSAC is the only option at the time of writing.

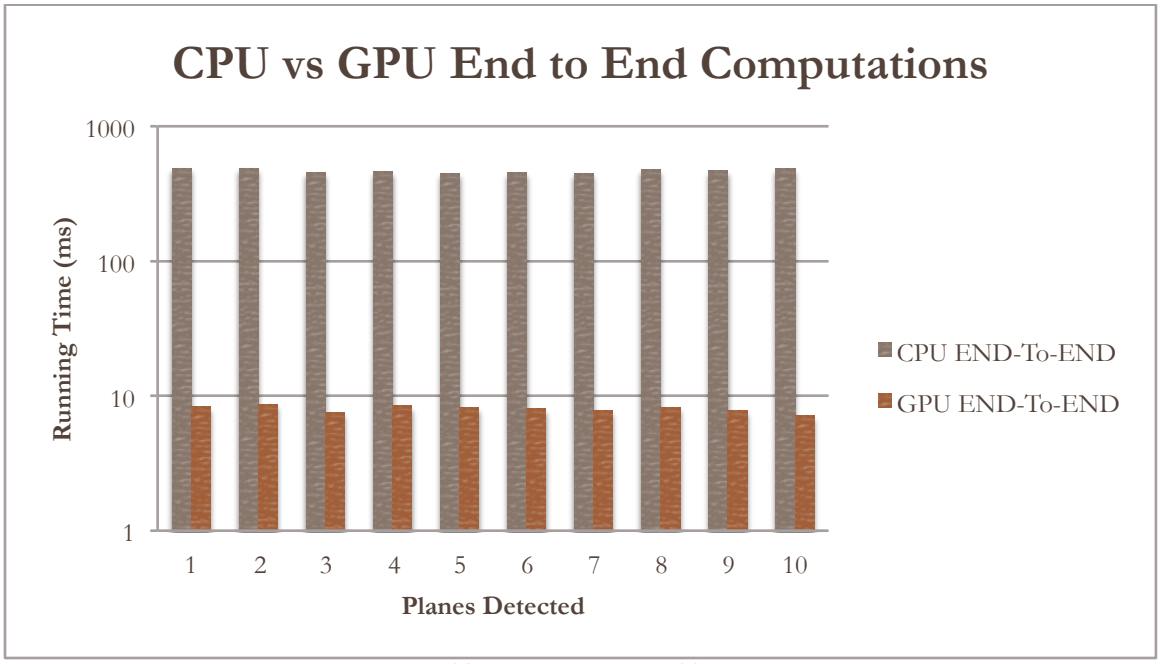


Figure 16 - End-To-End Computation Time (summation)

As can be seen in Figure 16, there is a very large difference in the amount of time required computing planes on solely the CPU or using a combination of the CPU & GPU. While this may look telling when the processes are more closely examined, done by timing the running times of various portions of the program(s). It is possible to see that the majority of the speed improvements come from one process. This is the calculation of the normal within the field of view of the Kinect. This is where the bulk of the difference is located and the

reason why the process is much faster on GPU's versus CPU's. To see the two steps separated to their run times refer to Appendix 3.

8.0 Future Work

While this project was able to examine the usability and properties of the Kinect sensor system a comparison of this to other tools that perform similar tasks would have been useful.

A detailed comparison to the recently release Microsoft KinectFusion would also be desirable to see what it implemented and how its process may be able to further enhance this methodology for environment segmentation.

9.0 Conclusions

Environment segmentation is an up and coming tool within the field of mobile and service robotics. Until this point, most of the devices required to do this costs thousands of euros and even at that price came with pretty large drawbacks. The Microsoft Kinect is a device that brings a large amount of power to a much wider audience. As long as users respect its abilities and deficiencies, the Kinect is a very powerful device that puts very advanced information in the hands of an unprecedented number of researchers.

In terms of Environment Segmentation there are still a few drawbacks that have to be overcome on multiple fronts. First, that full resolution processing is not yet viable nor is it recommended as discussed. Secondly, that the solution cannot simply rely on a GPU. As discussed there are times when a GPU is highly inefficient. There is also the issue that the Kinect is not a suitable tool for outdoor use in terms of depth information gathering.

All of this stated, the Kinect in combination with parallel PCL allows a new host of techniques and abilities to a large spread audience. If the Kinect is used within the ranges and condition for which it has been determined to be useful it stands to provide a useful set of tools to the mobile and service robotics fields.

Bibliography

[1] RoboCup @ Home. (2011, May) RoboCup@Home. [Online]. http://www.ais.uni-bonn.de/~holz/rulebook.pdf

[2] Steve Cousins and Radu Bogdan Rusu, "3D is here: Point Cloud Library (PCL)," *Proceedings of the IEEE Conference on Robotics and Automation (ICRA)*, 2011.

[3] Fausto Bernardini Holly Rushmeier, "The 3D Model Acquisition Pipeline," *Computer Graphics Forum,* vol. 21, no. 2, pp. 149-172, 2002.

[4] V.O. Roda G. Calin, "Real-Time Disparity Map Extraction In A Dual Head Stereo Vision System," *Latin America Applied Research*, vol. 37, pp. 21-24, 2007.

[5] Microsoft Research, *Kinect for Windows SDK (beta) Programming Guide*.: Microsoft Press, 2011, vol. 1.0b.

[6] Dickson Costa Diogo Costa, José Carlos Cavalcanti, "A Cambrian Explosion of Robotic Life," *CS Canada Management Science and Engineering*, vol. 5, no. 1, pp. 98-105, 2011.

[7] PrimeSense. PrimeSense Reference Design. [Online]. http://www.primesense.com/?p=514

[8] OpenNI. OpenNI User Guide. [Online]. http://www.openni.org/images/stories/pdf/OpenNI_UserGuide_v3.pdf

[9] Nicolas Burrus. (2011, July) Kinect RGBDemo v0.6.0. [Online]. http://nicolas.burrus.name/index.php/Research/KinectRgbDemoV6?from=Research.KinectRgbDemoV3#tocLink13

[10] Andreas Reichinger. (2011, Apr.) Kinect Pattern Uncovered. [Online]. http://azttm.wordpress.com/2011/04/03/

[11] "bbzippo". Kinect in infrared. [Online]. http://bbzippo.wordpress.com/2010/11/28/kinect-in-infrared/

[12] "Lightfarmer". (2008, May) Wikimedia Commons. [Online]. http://commons.wikimedia.org/wiki/File:1-stripesx7.svg

[13] Yvon Voisin, David Fofi Tadeusz Sliwa, "A Comparative Survey on Invisible Structured Light," *Society of Photographic Instrumentation Engineers 5303*, vol. 90, 2004.

[14] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys, "A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus," *Computer Vision - ECCV 2008*, vol. 5303, pp. 500 - 513, 2008.

[15] Martin A Fishcler and Robert C Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381 - 395, 1981.

[16] Dirk Holz, "Real-Time Plane Segmentation using RGB-D Cameras," University of Bonn,.

[17] Cosimo Distante Giovanni Indiveri, "Robust 3D Plane Estimation for Autonomous Vehicle Applications," Istituto Nazionale di Ottica CNR & Dipartimento Ingegneria Innovazione, Universita del Salento, 2011.

[18] J Matas and O Chum, "Randomized RANSAC with Td,d test," *Image and Vision Computing*, vol. 22, no. 10, pp. 837 - 842, 2004.

[19] D Capel, "An Effective Bail-out Test for RANSAC Consensus Scoring," in *Procedings from Britsh Machine Vision Conference*, 2005, pp. 629 - 638.

[20] Ondrej Chum and Jiri Matas, "Matching with PROSAC - Progressive Sample Consensus," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.

[21] Ben Tordoff and David W Murray, "Guided Sampling and Consensus for Motion Estimation," in *European Conference on Computer Vision*, vol. 2350, 2002, pp. 82 - 96.

[22] Ondrej Chum, Jiri Matas, and Josef Kittler, "Locally Optimized RANSAC," in *Deutsche Arbeitsgemeinschaft fur Mustererkennung*, 2003, pp. 236 - 243.

[23] Open Kinect. [Online]. http://openkinect.org/wiki/Protocol_Documentation

[24] W Victor Lee et al., "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," *ISCA '10 Proceedings of the 37th annual international symposium on Computer architecture*, pp. 451-460, June 2010.

[25] David Gohara, Guochun Shi John E. Stone, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, pp. 66-72, June 2010.

[26] NVIDIA. (2007) NVIDIA CUDA Computer Unified Device Architecture Programming Guide.

[27] Neil G. Dickson, Firas Hamz Kamran Karimi, "A Performance Comparison of CUDA and OpenCL," *eprint arXiv:1005.2581*, p. 12, May 2010.

[28] Firas Hamz Kamran Karimi Neil G. Dickson, "A Performance Comparison of CUDA and OpenCL," *eprint arXiv:1005.2581*, May 2010.

[29] NVIDIA. (2011, Feb.) NVIDA. [Online]. http://pressroom.nvidia.com/easyir/customrel.do?easyirid=A0D622CE9F579F09&version=live&prid=726171&releasejsp=release_157&xhtml=true

[30] Dr Zaius. (2009, Aug.) Confessions of a Speed Junkie (Overview). [Online]. http://gpgpu-computing.blogspot.com/2009/08/cage-match-cpu-vs-gpu.html

[31] Point Cloud Library. (2011, June) [Online]. http://pointclouds.org/

[32] Robot Operating System. (2011, June) Kinect Accuracy. [Online]. http://www.ros.org/wiki/openni_kinect/kinect_accuracy

[33] L. Falkenhagen, "3D object-based depth estimation from stereoscopic image sequences," *International Workshop Stereoscopic and 3D Imaging*, pp. 81-86, 1995.

Appendix 1 – Transparent Objects.

Figure 17 - Distance for clear glass cup.

Figure 18 - Partial Identification of a Clear Glass Cup

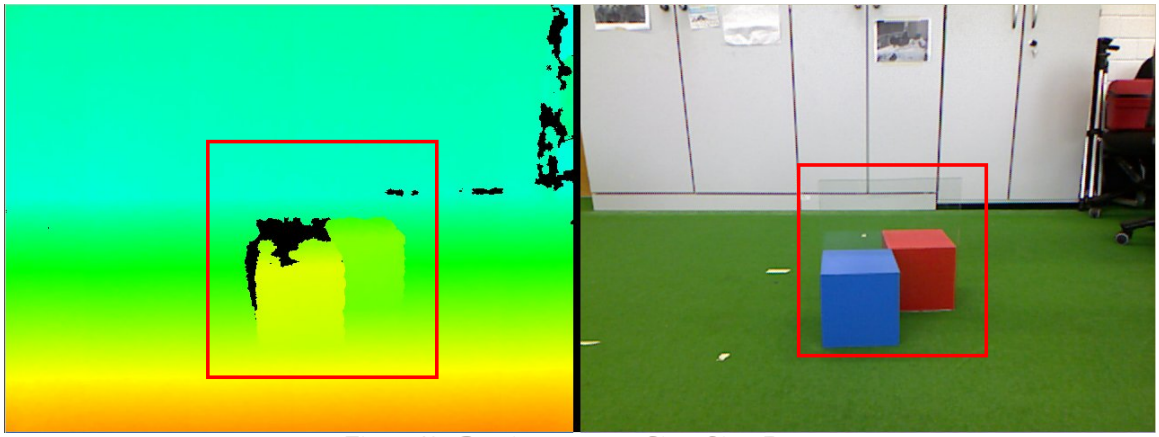


Figure 19 - Depth capture on Clear Glass Pane

Appendix 2 – Outdoor Data Capture

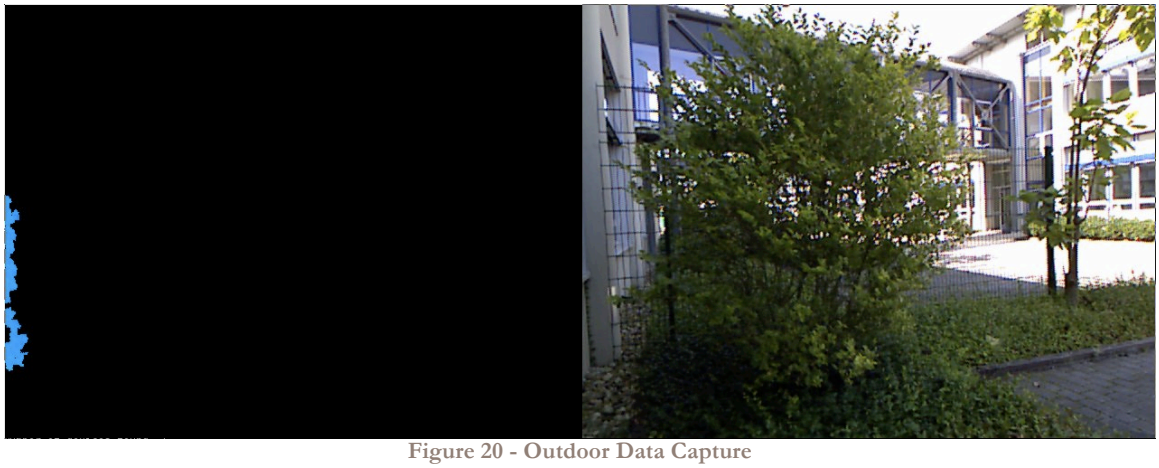


Figure 20 - Outdoor Data Capture

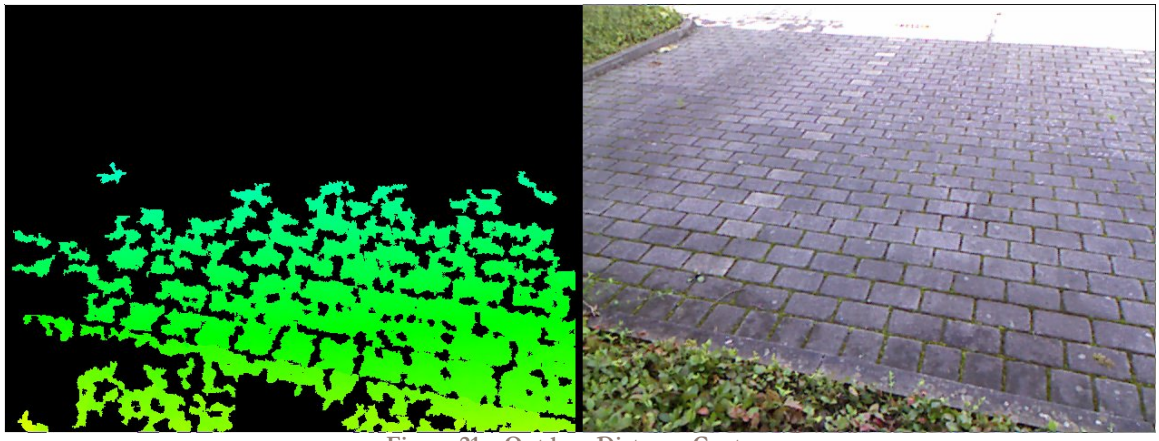


Figure 21 – Outdoor Distance Capture

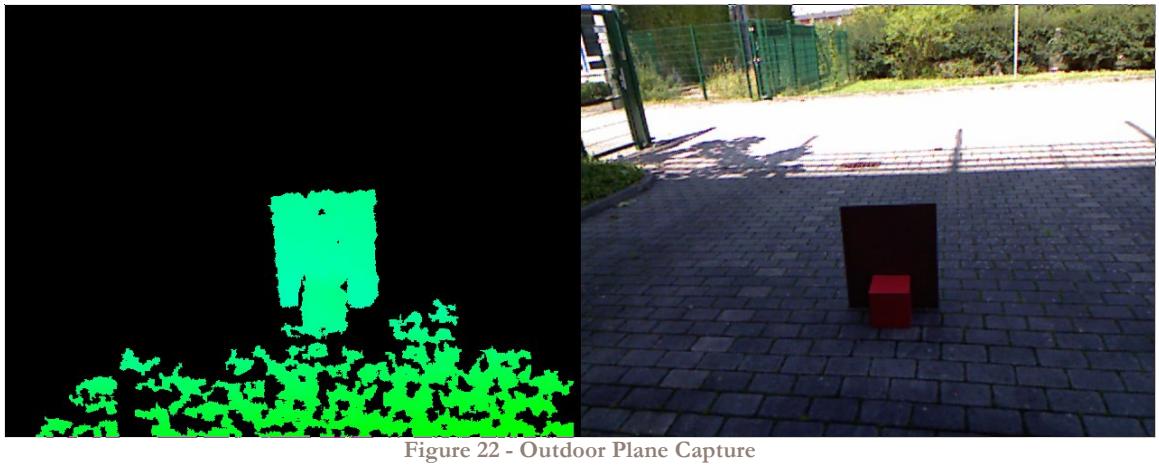


Figure 22 - Outdoor Plane Capture

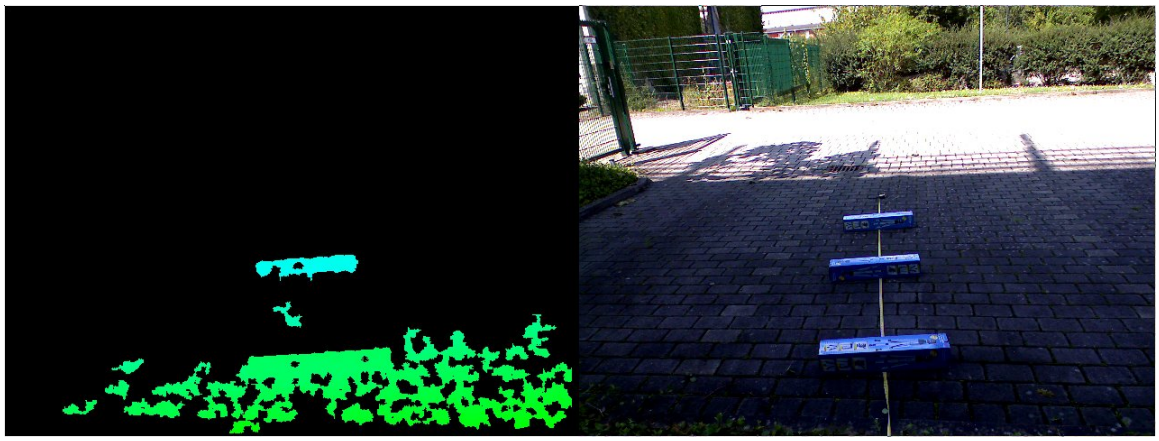


Figure 23 - Outdoor Distance Markers

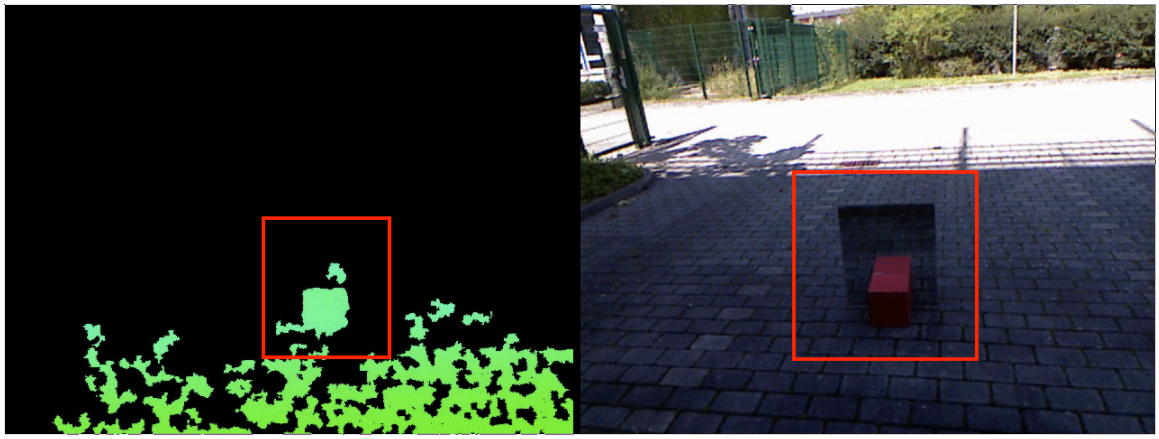


Figure 24 - Outdoor Mirror

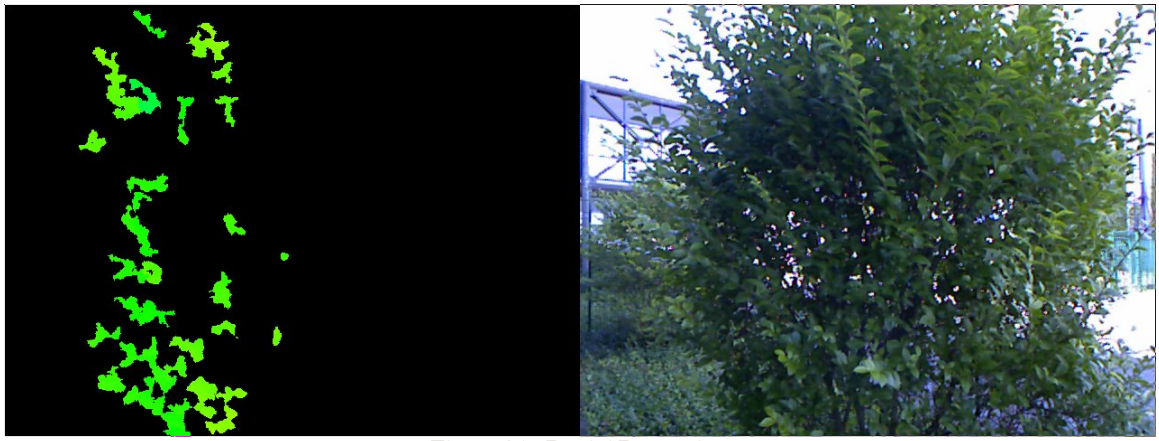


Figure 25 - Partial Processing

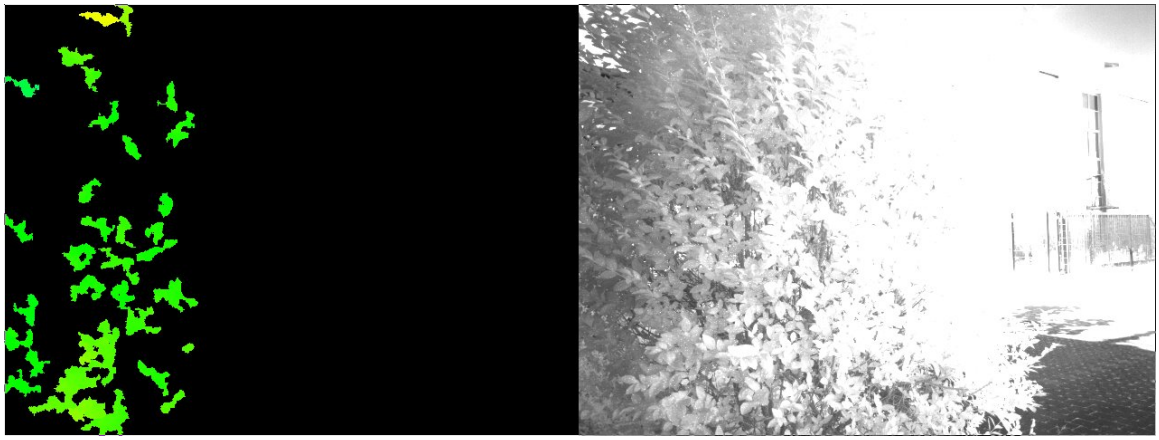
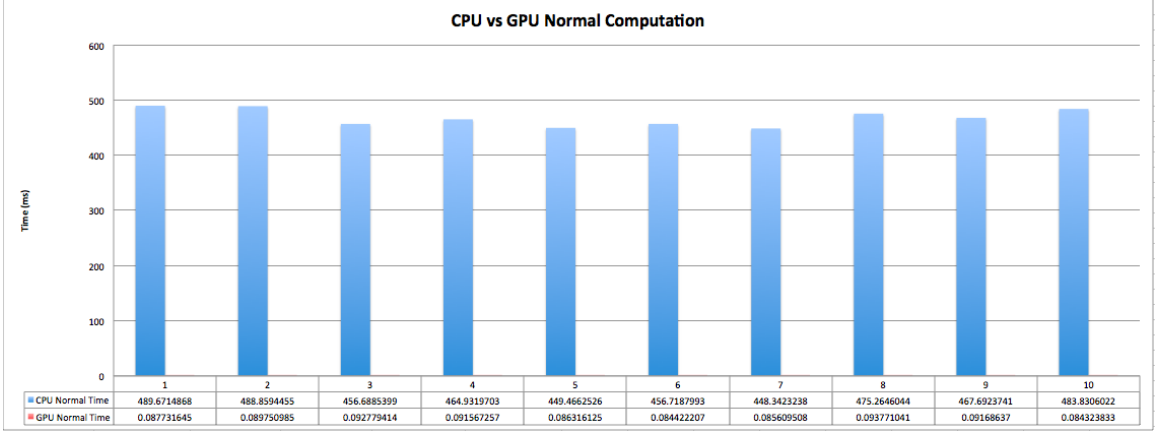


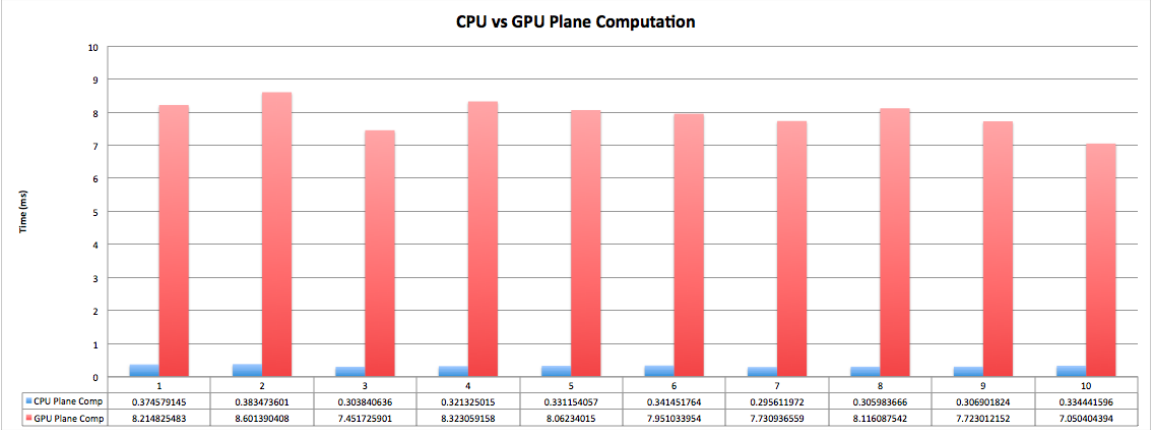
Figure 26 - Outdoor IR Interference

Appendix 3 – CPU vs. GPU Detailed Time Charts



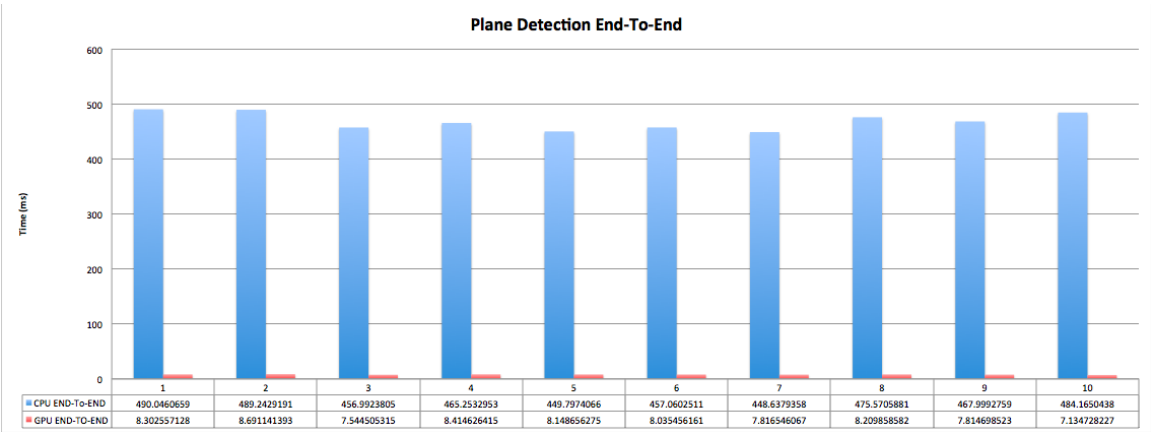
	1	2	3	4	5	6	7	8	9	10
CPU Normal Time	489.6714868	488.8594455	456.6885399	464.9319703	449.4662526	456.7187993	448.3423238	475.2646044	467.6923741	483.8306022
GPU Normal Time	0.087731645	0.089750985	0.092779414	0.091567257	0.086316125	0.084422207	0.085609508	0.093771041	0.09168637	0.084323833

Figure 27 - Normal Estimation Computation Times



	1	2	3	4	5	6	7	8	9	10
CPU Plane Comp	0.374579145	0.383473601	0.303840636	0.321325015	0.331154057	0.341451764	0.295611972	0.305983666	0.306901824	0.334441596
GPU Plane Comp	8.214825483	8.601390408	7.451725901	8.323059158	8.06234015	7.951033954	7.730936559	8.116087542	7.723012152	7.050404394

Figure 28 - Plane Estimation Computation Times



	1	2	3	4	5	6	7	8	9	10
CPU END-To-END	490.0460659	489.2429191	456.9923805	465.2532953	449.7974066	457.0602511	448.6379358	475.5705881	467.9992759	484.1650438
GPU END-TO-END	8.302557128	8.691141393	7.544505315	8.414626415	8.148656275	8.035456161	7.816546067	8.209858582	7.814698523	7.134728227

Figure 29 - End-To-End Computation Time