Environment Mapping in Larger Spaces

Andrew Ciambrone

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Science & Application

Denis Gračanin, Chair
Christopher L. North
Jeffrey Todd Ogle

December 14, 2016
Blacksburg, Virginia

Keywords: Sensor Fusion, Environment Mapping, Image Processing, Computer Vision

Copyright 2016, Andrew J. Ciambrone
Environment Mapping in Larger Spaces

Andrew J. Ciambrone

ABSTRACT

Spatial mapping or environment mapping is the process of exploring a real world environment and creating its digital representation. To create convincing mixed reality programs, an environment mapping device must be able to detect a user’s position and map the user’s environment. Currently available commercial spatial mapping devices mostly use infrared camera to obtain a depth map which is effective only for short to medium distances (3-4 meters). This work describes an extension to the existing environment mapping devices and techniques to enable mapping of larger architectural environments using a combination of a camera, Inertial Measurement Unit (IMU), and Light Detection and Ranging (LIDAR) devices supported by sensor fusion and computer vision techniques. There are three main parts to the proposed system. The first part is data collection and data fusion using embedded hardware, the second part is data processing (segmentation) and the third part is creating a geometry mesh of the environment. The developed system was evaluated against its ability to determine the dimension of the room and of objects within the room. This low cost system can significantly expand the mapping range of the existing mixed reality devices such as Microsoft HoloLens device.
Environment Mapping in Larger Spaces

Andrew J. Ciambrone

GENERAL AUDIENCE ABSTRACT

Mixed reality is the mixing of computer generated graphics and real world objects together to create an augmented view of the space. Environmental mapping, the process of creating a digital representation of an environment, is used in mixed reality applications so that its virtual objects can logically interact with the physical environment. Most of the current approaches to this problem work only for short to medium distances. This work describes an extension to the existing devices and techniques to enable mapping of larger architectural spaces. The developed system was evaluated against its ability to determine the dimension of the room and of objects within the room. With certain conditions the system was able to evaluate the dimensions of a room with an error less than twenty percent and is capable of determining the dimensions of objects with an error less than five percent in adequate conditions. This low cost system can significantly expand the mapping range of the existing mixed reality devices such as the Microsoft HoloLens device, allowing for more diverse mixed reality applications to be developed and used.
## Contents

1 Introduction 1

   1.1 Proposed Approach .................................................. 3

   1.2 Contributions ........................................................... 5

2 Literature Review 6

   2.1 Image Segmentation ..................................................... 11

3 Approach 13

   3.1 Data Collection ......................................................... 15

      3.1.1 Implementation ................................................... 16

   3.2 Processing ............................................................... 16

      3.2.1 Image Segmentation .............................................. 17

      3.2.2 Preprocessing ..................................................... 17
3.2.3 Filter selection ............................................ 21
3.2.4 Segmentation Algorithm Choice .......................... 22
3.2.5 Region Growing ........................................... 24
3.2.6 Plane Creation ............................................ 31

4 Evaluation .................................................. 34

4.1 Simulation .................................................. 35
4.2 Experiment .................................................. 36
  4.2.1 Room Dimensions ...................................... 36
  4.2.2 Dimensions of an Object .............................. 39

5 Conclusion ............................................... 48

5.1 Discussion ............................................... 48
5.2 Lessons Learned .......................................... 50
  5.2.1 OpenCV ............................................... 51
  5.2.2 RTIMULib ............................................ 52
5.3 Future Work ............................................. 53

Bibliography ............................................... 54
List of Figures

2.1 LAB Color space model [6]. .............................................. 9

3.1 An outline of the system workflow. ................................. 14

3.2 An outline of the data collection workflow. ......................... 15

3.3 Segmentation with and without filtering. ....................... 18

3.4 Segmentation with too much filtering. ............................ 19

3.5 Comparison of the three filters .................................. 21

3.6 An example of image segmentation. .............................. 24

3.7 An example of region growing. .................................. 26

3.8 An example of region merging. .................................. 28

3.9 Calculating the boundary points ................................. 33

4.1 Simulation segmentation. ....................................... 35
4.2 LIDAR data (top view). ................................................. 39
4.3 System data (top view). ................................................. 40
4.4 LIDAR data (perspective view). ......................................... 41
4.5 System data (perspective view). ......................................... 43
4.6 Object dimension evaluation setup. ....................................... 44
4.7 Object dimension evaluation. ............................................ 45
4.8 Error of object dimension estimate as a function of distance. ......... 46
4.9 Error of object dimension estimate as a function of angle. ............. 47
5.1 A stationary IMU inaccurately showing changes rotation (green line) .... 50
## List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Velodyne LIDAR puck specifications [4]</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Dimensions of the room.</td>
<td>38</td>
</tr>
<tr>
<td>4.3</td>
<td>Dimensions of the white board</td>
<td>42</td>
</tr>
<tr>
<td>A.1</td>
<td>Raspberry PI specification</td>
<td>58</td>
</tr>
<tr>
<td>A.2</td>
<td>Camera specification[22]</td>
<td>59</td>
</tr>
<tr>
<td>A.3</td>
<td>Lidar specification [3]</td>
<td>59</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Spatial mapping or environment mapping is the process of exploring a real world environment and creating a digital representation of it. This process is used in various application domains, including mixed reality. Current mixed reality commercial devices, such as Microsoft HoloLens and Google Project Tango devices, have built-in support for environment mapping. However, the approaches that these devices take are effective for short to medium ranges (3-4 meters). Most built environments contain large spaces that are not suitable for these devices thus limiting the applicability. The described research focuses on augmenting the environment mapping capabilities of the current mixed reality devices.

Real time environment mapping is an important component of many systems and is a well-researched concept. Over the last few decades many different approaches have been developed to solve the problem. In general, there are two main categories: real time and off-line
approaches. Off-line or batch approaches tend to process all information simultaneously in a single batch. The real time approaches processes the information as it comes in and produce the result in real time.

The real time approaches have to map the environment on the fly without any knowledge of future states. Real time systems are also limited to the amount of time the system has to process the data is limited by the amount of time it takes for the next frame of information to come in.

Offline approaches have more data at their disposal and are allowed to take more time to process that data. This gives offline approaches an advantage over real time approaches, which lets produce higher quality three dimensional geometries typically. Real time approaches, on the other hand, usually create more rudimentary meshes and are more prone to error. However, real time approaches are more appropriate for interactive mixed reality. Thus system being described in this work is considered a real time approach.

Mixed reality devices tend to be lightweight mobile devices. Their processing power, memory and other resources are constrained. Limited memory and limited CPU power mean that decisions must be made on how much data can be collected. Collecting too much data could introduce delay causing a latency between what the user sees and what the systems outputs. One solution to this problem would be to collect the sensor data on the device and process it on a more powerful machine by sending it over the network. However, this would add a bottleneck to the system due to network constraints.
Typical real time mapping strategies focus on processing high density point clouds. However due to the limitations of a mixed reality system, generating such a high density point cloud is unrealistic due to limitations in Light Detection and Ranging (LIDAR) technologies and computer vision techniques. Creating a dense point cloud is often achieved by using expensive image processing techniques such as “structure from motion” technique or by using a bulky and expensive LIDAR systems. The approach described in this work is limited to working with sparse point clouds.

1.1 Proposed Approach

Overall, there are three main components to the system. The first part focuses on data collection and data fusion. The second part processes the collected data and finally, the third component formulates and outputs a 3D mesh.

There are three types of data to be collected and processed by this system: images, range measurements, and the devices specific acceleration, gyro and magnetic data. This is achieved by using a combination of a camera, a LIDAR device, and an inertial measurement unit (IMU). The quality of the data and the rate of collection depends on the quality of the device. While having more data samples would create a better result, the system must fit within the constraints of a wearable device. The biggest constraint would be determining the type of LIDAR. Most LIDAR devices are big and bulky and are not portable. The most promising type of LIDAR would be a singular range finder device such as the Garmin LIDAR
Once the data is collected and bundled together by the time it was captured the system can then start processing the data. The first step is to segment the image into meaningful chunks. The purpose of the segmentation is to group pixels of the image into sections so that when the LIDAR does hit that portion of the image we can interpolate entire regions instead of just points of the image. There are various strategies for segmenting an image. Examples of popular techniques include using edge detection, convolutional neural networks, or region-growing.

Given the segmented image the next step in the processing phase is to create the actual mesh from the images. Since the image is segmented into parts the system assumes that the segments lie the same plane of orientation. To determine the regions orientation at least three LIDAR points must be captured on that particular region.

With a regions orientation it is possible to determine the size and the bounds of the region in the real world. Calculating the bounds of the region in the world space can be done by finding the intersection between a ray that coming from the camera and the calculated plane in camera space.

The system will be evaluated on its accuracy of representing the environment and how accurately it can estimate the dimensions of the space its located in. Evaluations can also be done by comparing this system to an existing system that cannot be used in mixed reality applications such as high density LIDAR systems.
1.2 Contributions

This work describes a method of environment mapping that is relative to the system’s localized space using only commodity off the shelf components. The goals of this work is to create a system that could augment the range of existing mixed reality technologies. The work presented here achieves these goals by describing a work flow and its implementation that provides the dimensions of the room, given if the user performs a full rotation, and a method for capturing the size of objects at reasonable distances.

According to the results from the evaluation the system can determine the dimensions of a room with an error of less then 20 percent. The results also show that the system is capable of determining the dimensions of objects with an error less then 5 percent adequate conditions.
Chapter 2

Literature Review

The first version of the Microsoft Kinect created its depth map by using an infrared laser light and two infrared cameras [18]. By sending out a speckled pattern of infrared light out into a room the Kinect is able to calculate depth by looking at the deformities in the light in a process called structured light. This process is also combined with two other computer vision techniques called depth from focus and depth from stereo.

Depth from focus is the idea that items far away are more blurry than those that are closer. Therefore if a part of infrared pattern more blurry then another portion of the pattern its determined to be further away.

The other computer vision principle being used by the Kinect is depth from stereo or parallax. By having two perspectives the Kinect can calculate the differences in the infrared light which allows it to calculate the distance.
The Kinect also improves the accuracy by using an astigmatic lens on its infrared cameras. An astigmatic lens is a lens with different focal length in the $x$ and $y$. The difference in focal lengths causes the infrared light from the laser to be in an ellipsoid shape. The direction of the ellipsoid shows the orientation at the point on the surface.

The second version of the Microsoft Kinect which is what the HoloLens depth finding sensor is believed to be based off uses a different approach [14]. Instead of structured light, version two of the Kinect is time of flight based.

What Microsoft has done with the Kinect and the HoloLens is impressive and has transformed the a great deal of the of the computer vision community. However, using infrared light to determine distance only works in short ranges [19].

LIDAR devices also principle of time of flight by determining a distance of a object by sending a pulse of light and the timing the duration before the light beam is reflected back [12]. Due to the known constant speed of light, the distance can be calculated by dividing the total amount of time by two and multiplying it by the speed of light. With the orientation and position the LIDAR device then a position of the measured object can be calculated.

Mapping an environment can be done in three different ways, geometrically, radiometrically and semantically [26]. The geometric mapping spatially maps the real world, the radiometric mapping maps the color of the world, and the semantic mapping creates an understanding of the environment. Geometric mapping usually uses sensors such as LIDAR or range finder cameras while radiometric mapping typically uses cameras but could also use thermal imag-
Semantic mapping tries to create understanding in the scene such as object detection or understanding where objects are in reference of each other. Recently this area of mapping has been advancing with the use of deep learning [26].

For cloud point collection there are two main types of methods, passive and active methods. Passive methods rely on reasonable lighting and typically use imagery for their methods of point collection. Active methods manipulate the scene to gather more information. Microsoft HoloLens and Kinect are good examples of mapping devices that use active methods to determine distances by using an infrared laser [14]. An example of a passive method would be the “structure from motion” technique [26].

Feature extraction is the idea of locating point of interest from an image. Examples of the features in an 2D image are: pixel properties, textures, and shapes within an image.

Texture features are defined as the spatial placement of the intensity values in an image. For example, a checkerboard would give a different texture compared to a spherical contour. Looking at the texture features of an image would be useful for determining regions of an image [26].

Shape features can be detected by using contour filters such as the Canny, Sobel, Roberts, and Prewitt operator [26]. All of these operators are useful in determining the contours of an image. To get more then just the contours, a Hough transform could be used to gather primitive geometric shapes such as circles or ellipse.

The $L^*A^*B$ color space is defined by International Commission on Illumination or CIE is a
color space with three components $L$ for luminance and two color channels, $a$ and $b$ [6]. The $a$ component represents the color ranges from green to red and the $b$ component represents the color ranges from blue to yellow as in figure 2.1.

![LAB Color space model](image.png)

**Figure 2.1: LAB Color space model [6].**

Because of the way the color space is designed distances between colors can accurately measured by using calculating their Euclidean distance. The L*A*B color space also has the added benefit of decreasing the importance of light when determining distances between colors. In the L*A*B color space luminance can only have values between 0 – 100 while the $a$ and $b$ components can have values between $-127 and 127$.

The simultaneous localization and mapping (SLAM) problem is one of incrementally building a consistent map of the environment and simultaneously determining its location within the map [9]. Work done in this area is closely related to the work done for the mixed reality
environment mapping. The main difference being that these strategies are typically used with robotics or drones where the system’s position could be controlled.

Environment mapping for mixed reality applications on the other hand have little control over the user. The system cannot force the user to look a certain direction or to move in a certain way it must estimate the environment based on what information it has.

Sensor fusion [7, 8] is based on intelligent integration of data derived from a collection of disparate sensor so that the resulting information provide more accurate information or information that cannot be derived from individual sensors (e.g., orientation, position).

The Pinhole Camera is the relationship between the coordinates of a 3D point and its projection onto the image plane [24]. This gives the ability to calculate a point on the image frame into a point in 3D space. Equations 2.1 and 2.2 are for calculating real world points on the camera image [24].

\begin{align*}
    u &= m_u \frac{f_X}{Z} + m_u t_u \\
    v &= m_v \frac{f_Y}{Z} + m_v t_v
\end{align*}

where \((u, v)\) is defined as the coordinate of the pixel in the image plane, \(f_X\) and \(f_Y\) is the focal length in the \(X\) and \(Y\) direction respectfully, \(t_u, t_v\) are the translation of the camera plane, and \(m_u\) and \(m_v\) are the image plane width and height of the in number of pixels.
2.1 Image Segmentation

Image segmentation is the process of dividing an image into discrete parts. There are many different groups of algorithms that could be used to segment an image. The most common include clustering, edge detection, and machine learning techniques such as neural networks.

Superpixel segmentation [23] is a method of image segmentation. One of the ideas introduced is the classification of good versus bad segmentation. They define similarity of pixels in two ways: intra-region similarity and inter-region dissimilarity. Intra-region similarity is defined as the elements in the region being similar in brightness, similar in texture, and have low contour energy. While inter-region dissimilarity is defined as the dissimilarity of these properties but between regions.

Texture similarity is computed by performing texton analysis [23]. This approach defines brightness similarity as a distance between the histograms between regions. The contour energy is computed by the orientation energy at the pixel level. The inter-region contour energy is the summation of the contour energy at the pixel level. The curvilinear continuity is defined as 2.3.

\[ C(S) = \sum_{(q, q') \in J(S)} \log P_{\text{tangent}}(\alpha(q, q')) \]  \hspace{1cm} (2.3)

The summation of the normalized features are what is used to compare between regions.

Seeded Region Growing [5] is a image segmentation method that is quite often used in
computer vision. The algorithm starts by picking an set of $N$ seeds. Then for each iteration each of the seeds gain one additional pixel to its region. This is done by looking at the pixels that border the regions.

The pixel that is added to the region is determined by the least difference from the region. If a pixel is approached by two regions then the algorithm considers that to be a boundary pixel [15]. The success of the algorithm relies on the selection of initial pixel seeds. An automated way of initial pixel seeds would be to use the converging squares algorithm [20].
Chapter 3

Approach

Mapping large architectural (indoor) spaces presents challenges due to limited range of currently available mixed-reality devices. It can take some time for the user to navigate the whole area and complete space mapping. While user interactions do take place in the immediate surrounding of the user, having some information of the more distant parts of the environment (without the need for the user to go there) could provide better context for user interactions. For example, mapping objects further away (such as walls and high ceilings) enables construction of the model of the surrounding architectural space.

That can be achieved by augmenting the current mixed-reality devices with additional longer-range sensor technologies. However, these additions must be light and mobile and safe for the user to use. Due to these constraints main difficult design decisions must be made to produce fast but not necessarily highly accurate results. However, since this those results...
describe areas of space more distant form the user, even less accurate results are sufficient. The moment the user moves closer, the near-range mapping hardware is used to improve the accuracy.

Figure 3.1: An outline of the system workflow.

Figure 3.1 shows the overall approach. The main processing loop is based on the camera input. Each camera frame is annotated by LIDAR and IMU data within a specified time distance from the frame capture time. The fused data is then used to determine region in space and construct the surrounding geometry.
3.1 Data Collection

Figure 3.2 shows the data collection workflow. In total five types of data are being collected and used: the specific force or acceleration of the system, the angular rate, the magnetic field, image data, and distance data. This is achieved by using a combination of a camera, a LIDAR device, and an inertial measurement unit or IMU.

To reduce the natural error of the IMU, sensor fusion is performed on the data. Since time performance is essential, a simplified version of Kalman filter, called RTQF, from the RTIMULib library is used [17]. The sampling rate for these components are 80 samples per second for the IMU sensor and 100 samples per second for the LIDAR sensor. The camera is capable of capturing up to 30 frames per second. However, due to the time it takes to
process each frame the effective camera frame rate is reduced to approximately 5-6 frames per second.

### 3.1.1 Implementation

Because data collection is critical, three threads are dedicated to the process of collecting the data points. Two threads are used to collect the data from the sensors. One thread is dedicated to collecting the IMU data and the other is dedicated to collecting from the LIDAR. The third thread is the thread dedicated to synchronizing the data. This thread then takes the synced data and pushes it onto an Asynchronous queue. This data is later be used collected by another thread that collects and processes the image data.

### 3.2 Processing

In this phase we are processing the data that is collected from the sensors and preparing it to be converted into geometry meshes. This is done in three main steps:

1. Image segmentation
2. Image annotation
3. Plane creation
3.2.1 Image Segmentation

The success of this system relies heavily on the segmentation image. A segmentation algorithm takes in image and partitions it into discrete parts. The main goal of any image segmentation is to simplify the image by grouping pixels together by corresponding surfaces or objects.

In this system the goals of the segmentation is to separate the image into surfaces and to get the bounds of the region. The system needs a segmentation algorithm that maintains consistency of the generated regions, be as close to real time as possible, and add a bias to creating larger regions.

In indoor spaces there are a lot of variations in the image due to surface textures (e.g., a rough brick) and lighting differences. Overhead lights and windows can create areas of intense light differences. Consequently, creating a good segmentation is difficult. To overcome these issues, images must be preprocessed before segmentation.

3.2.2 Preprocessing

One major flaw with many image segmentation approaches is that they are often prone to noise in the image so steps must be taken to reduce the noise before the segmentation can be ran on the image. This can be done by blurring or smoothing the image.

Blurring an image reduces the amount of noise and details in an image. Figure 3.3 left shows
a segmentation without filtering. The black regions are segments below the minimum filter size.

Figure 3.3: Segmentation with and without filtering.

A filtered image is preferred compared to the unfiltered image because the noise and tiny details in a image prevent segmenting surfaces into larger segments. For example, when segmenting a brick wall the system would perform better if it created only one segment for the wall instead of having a segment for each brick. However, image blurring does have a cost. Blurring processes each pixel in an image which does add a computation required for each frame.

It is also important to realize the amount of blurring that takes place. Too much blurring and the image will lose critical details such as the edges between surfaces. For example, Figure 3.4 shows a segmentation on an image with too much filtering. In the figure the edges between the walls were lost causing the segments to bleed over onto other surfaces.

Popular methods of image smoothing include Gaussian, median, and bilateral filter.
Figure 3.4: Segmentation with too much filtering.

**Gaussian Filter**

The Gaussian filter works by dividing the image into boxes where each pixel is transformed by the function:

\[
g(x, y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{d^2}{2\sigma^2}}
\]  

(3.1)

where \( d \) is defined as the distance between the pixel at \((x, y)\) and the pixel at the center of the box and \( \sigma \) is the standard deviation of the section [25].

**Median Filter**

The median filter is similar to the Gaussian filter by breaking the image into sections and performing processing on each section independently. However, it doesn’t average the entire section. Instead it calculates the median values in the rows and columns of that section and
sets the pixels to that value. Because it’s not averaging the median filter tends to preserve edges better compared to the Gaussian filter. However it does require more computation because the values of the image must be sorted in order to calculate the median.

**Bilateral Filter**

The bilateral filter is another filtering technique that is based on the Gaussian filter but retains the edges like the median filter. The bilateral filter manages to retain edges better then the Gaussian filter by looking at variation of the intensities [21]. Equation 3.2 shows the formula for the bilateral filter:

\[
BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|)G_{\sigma_r}(I_p - I_q)I_q
\]  

(3.2)

where \( W_p \) is the normalization factor defined as:

\[
W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|)G_{\sigma_r}(I_p - I_q)
\]  

(3.3)

and where \( G_{\sigma_s} \) is the spatial Gaussian that decreases the influence of the distant pixels. \( G_{\sigma_r} \) is a range Gaussian that decreases the influence of the values different from \( I_p \). \( \sigma_s \) and \( \sigma_r \) are parameters that determine how much each of the Gaussian filters can weight each pixel.
3.2.3 Filter selection

The median filter is used because it preserves the edges better than the Gaussian filter while at the same time removing noise in it. Although for this application it was found that while the bilateral filter does perform better than the median filter in retaining edges, it also takes much longer to process compared to the median filter.

Differential lighting in an image is something that can’t be filtered out. However, approaches can be taken to reduce the influence that light has on the segmentation. One approach is
changing the color space of the image to one where lightness is a separate component. $RGB$ is the typical format that frames from a camera come in however a color translation to a different color space such as the $L \ast A \ast B$ is rather simple [10]. In this implementation the color space transformation is done using the OpenCV library.

The $L\ast A\ast B$ color space also has the added benefit of reducing the significance of the lighting component. In the $L\ast A\ast B$ color space luminance can only have values between $0 - 100$ while the $a$ and $b$ components can have values between $-127$ and $127$ [6]. Calculating the differences of colors in the $L \ast A \ast B$ color space can be done by using the delta-E method.

### 3.2.4 Segmentation Algorithm Choice

While the filtering and changing the color space does help with the differential lighting, it still holds a major influence in the image segmentation choice. Many of the image segmentation processes use a process called global thresholding. That works well for images that are consistent in lighting such as microscopic images but does not work well in an indoor space with overhead lighting. Therefore, the segmentation algorithm has to use adaptive threshold or some other method to compensate for the light differences.

Finally, the amount of time needed for the segmentation was a major decision point in choosing a optimal segmentation algorithm. Currently, a great deal of research is ongoing in using convolutional neural network (CNN) to segment an image. For example, it is possible to train an CNN for room estimation and create an Autocad layout file [13]. Obviously
given enough time and resources the image segmentation could be vastly improved. The selected image segmentation algorithm (Algorithm 1) is a simplified unseeded region growing algorithm (Figure 3.6).

Algorithm 1 Image Segmentation

```plaintext
procedure REGION GROWING(Img, MinRegionSize)

Regions ← ∅
RegionMap ← [][] (all zeros)
RegionCounter = 1
Seeds ← ∅
CurrentRegion ← NULL

while Pixels left to process do
    if Seeds.length = 0 then
        if CurrentRegion != NULL then
            Regions ← Regions ∪ {CurrentRegion}

        newSeed ∈ RegionMap ≠ 0
        CurrentSeeds ← newSeed
        CurrentRegion ← New Region based on newSeed

    repeat
        CurrentSeed ← seed ∈ Seeds
        for all Neighbor ∈ 8 or 4 neighbor near CurrentSeed do
            if \( \text{Distance}(\text{CurrentSeed.Color, Neighbor.Color}) < \text{Threshold} \) then
                CurrentSeeds ← CurrentSeeds ∪ {Neighbor}
                CurrentRegion ← CurrentRegion ∪ {Neighbor}
        until Seeds.length = 0

    Return Region Merging(Regions, MinRegionSize)
```

The region growing was selected because it creates consistent segments and follows the guidelines of the parameters such as minimum segment size. Another reason why region growing was selected over other methods was because unlike many of image segmentation
algorithms region growing is a local method. It is only concerned with certain portions of
the image during the segmentation which works well with the differences in lighting typically
found in architectural spaces.

![Image of segmented image](image.png)

Figure 3.6: An example of image segmentation.

### 3.2.5 Region Growing

The region growing algorithm can be separated into three main parts. The first part is the
seed selection.
Seed selection can be crucial to seeded region growing algorithms because those algorithms typically only have a number of regions they can grow. The algorithm above is considered unseeded because no initial seeds are set through the parameters. There is also no real hard limit to the number of regions except for the number of pixels the image. Since there is no limit to the number of seeds, the selection of initial seeds for a region has little impact and can be negated by region merging. However, a heuristic that uses a peaks method in [5] is a good method of seed selection. This method looks for the points of highest variation in the image.

Region Growing After an initial seed is selected the algorithm the region growing portion begins. Figure 3.7 shows that the center seed looks at its neighbors and compares their color to its own color.

Popular methods for defining what a neighbor are the 8-connected or the 4-connected. In this system it was found that the 4-connected method has a slight advantage in creating an accurate segmentation. However, the advantage is not enough to justify the additional iterations the 4-connected neighborhood does create.

If the color difference or distance between current region seed and the pixel neighbor isn’t too large, the pixel neighbor will be added to the current region being built and added to the list of pixels to process. This process continues until there are no more seeds left. If a region runs out of seeds to process and there are still pixels left in the image left unlabeled, then another region is created on an unlabeled pixel.
Region Merging  Once every pixel in the image has been labeled with a region, the region merging portion of the algorithm begins (Figure 3.8). Region merging is used because the image contains artifacts from the camera sensor that were not filtered out by the median filter. For every region that is smaller than the minimum desired size, the region looks at its neighboring regions and determines if it can merge with one of those. There are two conditions to decide if a region will be merged with another. The first condition is if the difference between the average color of the region and its neighboring region is within the threshold. This condition prevents a bias from occurring with the order of the pixel processing. The second condition is when another region encompasses a region they are merged. This means that small regions that are surrounded by the same larger region will be merged with that region.

![Figure 3.7: An example of region growing.](image)
Algorithm 2 Merge Regions

\begin{algorithm}
\SetKwProg{RegionMerging}{procedure}{RegionMerging\newline(Regions, MinRegionSize)}{}
\textbf{RegionMerging}(Regions, MinRegionSize) \{
\begin{algorithmic}
\ForAll {Region ∈ Regions}
\If {Region.size < MinRegionSize}
\If {All Neighboring Regions are the same}
\State NeighboringRegion = NeighboringRegion ∪ Region
\EndIf
\ForAll {NeighRegion}
\If {Distance(Region.AvgColor, NeighRegion.AvgColor) < Threshold}
\State Region = Region ∪ NeighboringRegion
\EndIf
\EndFor
\EndIf
\EndFor
\EndIf
\Return Regions
\end{algorithmic}
\end{algorithm}

The segments created by the region growing algorithm has several properties.

1. All the pixels in the frame must belong to a region.

2. The points in the pixel must be connected at some point meaning no two isolated groups of pixels can be considered parts of the same region.

3. All regions are disjoint. There are no overlaps.

Selection of Data Points

While it would be best if all of the data points are used, only those data points around the time of frame capture can be used to create independent planes. This constraint verifies that the initial position of the LIDAR data points were collected are around the same position.
that the frame was collected. The time range value is determined based on the assumption that the user does not move faster than one meter a second or roughly 3.28 feet a second.

**LIDAR Frame Mapping**

With the frame segmented and the relevant data selected, the next step is to find where on the frame the LIDAR points lie. This is achieved by placing the frame into camera space. Where the camera space is defined as the camera being at position (0, 0, 0) with the camera and the LIDAR facing in the $y$ positive position. Each data point has the following attributes: *Time*, *Distance*, *Yaw*, and *Pitch*.

The conversion for these attributes is described in Equations 3.4–3.7. Let $sD1$ and $sD2$ be
the data points closest to the time that the frame was captured.

\[ \lambda = \frac{t - sD1.Time}{sD2.Time - sD1.Time} \quad (3.4) \]

\[ \text{Distance} = (1 - \lambda) \times sD1.Distance + \lambda \times sD2.Distance \quad (3.5) \]

\[ \text{Yaw} = (1 - \lambda) \times sD1.Yaw + \lambda \times sD2.Yaw \quad (3.6) \]

\[ \text{Pitch} = (1 - \lambda) \times sD1.Pitch + \lambda \times sD2.Pitch \quad (3.7) \]

Next the width and height of the image need to be calculated in the camera space (Equations 3.8-3.9).

\[ \text{FrameWidth} = \text{Distance} \times \tan\left(\frac{hFOV}{2}\right) \quad (3.8) \]

\[ \text{FrameHeight} = \text{Distance} \times \tan\left(\frac{vFOV}{2}\right) \quad (3.9) \]

where \( hFOV \) and \( vFOV \) are the horizontal and vertical field of view of the camera, respectively.

Then for each data point \( D \) we transform its attributes (Equations 3.10–3.14).
$Y_{Cam} = D.Yaw - \overline{Yaw}$ \hfill (3.10)

$P_{Cam} = D.Pitch - \overline{Pitch}$ \hfill (3.11)

$x_{Cam} = \text{Distance} \times \tan(Y_{Cam})$ \hfill (3.12)

$z_{Cam} = (\text{Distance} \times \csc(Y_{Cam}) \times \tan(P_{Cam})) - \delta$ \hfill (3.13)

$y_{Cam} = \text{Distance} - \epsilon$ \hfill (3.14)

where $\delta$ is defined as the distance between the camera and LIDAR and $\epsilon$ is defined as the distance offset for the LIDAR.

Finally, to calculate the pixel coordinates of the LIDAR points in the frame for the data point $D$ (Equations 3.15–3.16).

$x_{Pixel} = ImageWidth \times \frac{FrameWidth + x_{Cam}}{2 \times FrameWidth}$ \hfill (3.15)

$z_{Pixel} = ImageHeight \times \frac{FrameHeight + y_{Cam}}{2 \times FrameHeight}$ \hfill (3.16)
where $ImageWidth$ and $ImageHeight$ are the width and the height of the frame in pixels, respectively.

### 3.2.6 Plane Creation

Since there is little depth data, the system needs to extrapolate the depth’s of other parts of the environment. This is done by assuming that the regions captured during the segmentation process lie on the same orientation plane. With the orientation of a region its bounds and size can be determined. Which allows the system to extrapolate additional depth points of the region. To achieve this the LIDAR points need to be grouped by the region which they fall into.

For each region with three or more points, a check is done to verify if the points are well suited to create the plane. If the three points are overlapping or are co-linear (Equation 3.17), creating a plane is not possible.

In cases where there are more then three points a search must be performed that chooses the 3 points that form the largest possible triangle. By creating the largest possible triangle the amount of error in the plane creation and boundary extrapolation is reduced.

\[
(p_2.y - p_1.y) \times (p_3.x - p_2.x) - (p_3.y - p_2.y) \times (p_2.x - p_1.x) \tag{3.17}
\]
Mesh Creation

Once the system has determined that the region does contain three points that are optimal for creating a plane, the system can then proceed to calculating the position of the bounds of the region in camera space. The current implementation only calculates for eight boundary points. However, this could be easily extended to more points. In order to correctly determine the boundary point locations we need to also calculate an plane for the region. This region plane \( rp \) will hold the correct orientation for the region (Equation 3.18).

\[
(x_r, y_r, z_r) = \left( \frac{rp\text{.distance}}{D} x_c, \frac{rp\text{.distance}}{D} y_c, \frac{rp\text{.distance}}{D} z_c \right)
\]  

(3.18)

Where \( D \) is defined as:

\[
D = \sqrt{y_c^2 + x_c^2 + z_c^2}
\]  

(3.19)

In Equations 3.18 and 3.19 let \( x_c, y_c, \) and \( z_c \) be LIDAR points on the camera plane.

Figure 3.9 shows how the camera space positions of the region bounding points are determined. For each boundary point on the region (which is determined during the image segmentation process) a ray is created with its origin set at \((0, 0, 0)\) pointing to the boundary point on the image plane. The intersection between this ray and the region plane is the position of the boundary points in camera space.

All the boundary points and LIDAR points are saved and used are used to represent the mesh of that region.
Figure 3.9: Calculating the boundary points
Chapter 4

Evaluation

The described system was evaluated in two ways: in its ability to predict its environments dimensions from the points in the created mesh and by its ability to estimate the dimensions of an object in the environment.

Table 4.1: Velodyne LIDAR puck specifications [4].

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal FOV</td>
<td>360 deg</td>
</tr>
<tr>
<td>vertical FOV</td>
<td>30 deg</td>
</tr>
<tr>
<td>Accuracy</td>
<td>+/- 3cm</td>
</tr>
<tr>
<td>Number of Points</td>
<td>300,000</td>
</tr>
<tr>
<td>Range</td>
<td>100 meters</td>
</tr>
</tbody>
</table>
4.1 Simulation

Before the actual system was implemented, many of the algorithms and techniques were tested out in a simulated environment. While the evaluations performed in the simulation are not accurate a representation of a real-world environment, they provide some context about how accurate the system could be given ideal conditions.

The simulation was developed inside the Unity Game Engine [1]. The camera’s properties such as its field of view and resolution were set to be representative of the actual hardware used in the system’s implementation. The scene was filled with different shapes of different sizes and different colors.

![Simulation segmentation.](image)

Figure 4.1: Simulation segmentation.

Figure 4.1 the results for the simulation’s are shown. The segmentation algorithm can split the scene by surfaces.

Since the IMU and LIDAR were simulated, they added little error to the sensor readings. Resulting in very accurate results.
4.2 Experiment

4.2.1 Room Dimensions

When evaluating the system for its ability to detect the dimensions of the room, the system’s data was compared against the data from the LIDAR puck and against a ground truth which was determined from measurements taken from both a laser tape measure and a measuring tape.

Evaluation Setup

The evaluation was conducted in a room where Microsoft HoloLens device would be unable to detect the walls if placed in the center of the room due to the size of the room. To provide a comparison between modern state of the art versus our implementation the system was compared against the Velodyne LIDAR Puck or VPL-16. Table 4.1 shows the specifications of the LIDAR Puck who’s specification can be found in . The LIDAR Puck was calibrated and setup according to the instructions provided by the manufacturer of the puck.

Both implemented system and the LIDAR Puck were placed in the center of the room at approximately the same height. The implemented system was placed on top of a tripod where it was rotated around the room completing a full rotation capturing different parts of the room.
Evaluation Results

The results from the Room Dimension evaluation can be found in Table 4.2.

The dimensions of the room were calculated by finding the maximum and minimum values points in each of the spatial dimensions. However, due to the error in the gyroscope yaw value some of the extrapolated region boundary points may have caused some of the data points to be erroneous. To account for these erroneous data points a median value was calculated at each of the directions where meshes were created. However, the Table 4.2 contains the calculated dimension of both data sets.

The percent error was calculated by using Equation 4.1

\[ \text{PercentError} = \frac{\|\text{GroundTruth} - \text{Measured}\|}{\|\text{GroundTruth}\|} \] (4.1)

The percent error in the $X$ dimension of the developed system for all data points is 42.8% however when accounting for the erroneous points the percent error in the $X$ dimension is 1.5%. For the compared system the percent error in the $X$ dimension is 26.5%. The error in the compared system was high due to the windows in the room.

The percent error found in the $Y$ dimension of the developed system for all data points 30.1%. However when accounting for the erroneous points the percent error in the $Y$ dimension is 10.3%. For the compared system the percent error in the $Y$ dimension is 0.2%.

The percent error in the $Z$ dimension of the developed system for all data points is 176.02% however when accounting for the erroneous points the percent error in the $Z$ dimension is
Table 4.2: Dimensions of the room.

<table>
<thead>
<tr>
<th>Metric</th>
<th>X (cm)</th>
<th>Y (cm)</th>
<th>Z (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground Truth</td>
<td>1097.28</td>
<td>871.22</td>
<td>273.05</td>
</tr>
<tr>
<td>LIDAR Puck</td>
<td>1388.11</td>
<td>873.75</td>
<td>289.98</td>
</tr>
<tr>
<td>Developed System (All Points)</td>
<td>1567.01</td>
<td>1133.04</td>
<td>482.54</td>
</tr>
<tr>
<td>Developed System (Median)</td>
<td>1080.80</td>
<td>960.72</td>
<td>176.02</td>
</tr>
</tbody>
</table>

35.5%. For the compared system the percent error in the Z dimension is 6.2%.

**Discussion of the Results**

Due to the error in the gyroscope’s yaw an absolute orientation of the room was not going to accurately represent the direction that the developed system was pointed at. An estimate of the system’s orientation was later applied to the data points collected.

Frames were captured at eight different orientations evenly distributed around the Z axis. This estimate may have introduced errors into the estimate of the room dimensions. Another factor that introduced error in the Z dimension was that no data was collected of the floor and ceiling. That is evident in the high error in approximation of the room's height.

Figure 4.5 shows that the captured region extended the rather high in the collection. That means the segmentation was performed poorly causing a leak between the walls and ceiling.

Another important observation to note is of the error in the X direction for the LIDAR puck.
The LIDAR puck was calibrated and measurements were taken multiple times following the direction of the manufacturer. However, the error in the $X$ dimension remained.

![LIDAR data (top view)](image)

Figure 4.2: LIDAR data (top view).

### 4.2.2 Dimensions of an Object

An important part of environment mapping is accurately representing the dimensions of objects in a room. In this part of the evaluation the system ability to perform this task was evaluated. The dimensions of the object that the system was trying to measure a white board hanging on a wall.
Figure 4.3: System data (top view).

Evaluation Setup

The distance and angle between the object and the system affected the accuracy of the system’s estimate of the object’s dimensions. In order to test the system’s ability to accurately estimate the dimension of an object the system was placed facing that object at various distances and angles about the room.
Evaluation Results

Table 4.3 shows the changes in accuracy from the different distances and angles taken around the room.

Figures 4.8 and 4.9 show the percent error (calculated using Equation 4.1) of the dimensions based on the distance and angle of the implemented system.

Discussion of Results

There are a few things that cause the error in calculating the error in an object's dimensions. The first is accuracy of the segmentation. If the segmentation is too large or too small the
Table 4.3: Dimensions of the white board

<table>
<thead>
<tr>
<th>Metric</th>
<th>Width (cm)</th>
<th>Height (cm)</th>
<th>Angle° Taken</th>
<th>Distance (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground Truth</td>
<td>169.22</td>
<td>117.48</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Reading 1</td>
<td>167.63</td>
<td>109.20</td>
<td>0</td>
<td>182.88</td>
</tr>
<tr>
<td>Reading 2</td>
<td>164.14</td>
<td>117.58</td>
<td>0</td>
<td>271.78</td>
</tr>
<tr>
<td>Reading 3</td>
<td>157.85</td>
<td>106.10</td>
<td>0</td>
<td>383.54</td>
</tr>
<tr>
<td>Reading 4</td>
<td>168.79</td>
<td>127.73</td>
<td>0</td>
<td>510.54</td>
</tr>
<tr>
<td>Reading 5</td>
<td>147.53</td>
<td>100.62</td>
<td>0</td>
<td>645.16</td>
</tr>
<tr>
<td>Reading 6</td>
<td>141.24</td>
<td>114.12</td>
<td>25</td>
<td>353.06</td>
</tr>
<tr>
<td>Reading 7</td>
<td>136.00</td>
<td>124.57</td>
<td>50</td>
<td>347.98</td>
</tr>
<tr>
<td>Reading 8</td>
<td>94.10</td>
<td>126.81</td>
<td>60</td>
<td>342.90</td>
</tr>
<tr>
<td>Reading 9</td>
<td>102.31</td>
<td>127.18</td>
<td>50</td>
<td>490.22</td>
</tr>
</tbody>
</table>
predicted boundaries of the object are going to be different than what they truly are. The effect of the segmentation error has increases as the distance increases because the further the system is from the object the more spatial impact each pixel has.

However, the largest factor in the amount of error in the calculation is the error in the gyroscope, particularly the value of the yaw. Looking at Figure 4.9 it is apparent that the amount of error in the object’s width is much higher than the error in the height. Conversely looking at Figure 4.8 when the system is parallel to the white board, the error of the width and height are roughly around the same area.

Figure 4.5: System data (perspective view).
Figure 4.6: Object dimension evaluation setup.
Figure 4.7: Object dimension evaluation.
Figure 4.8: Error of object dimension estimate as a function of distance.
Figure 4.9: Error of object dimension estimate as a function of angle.
Chapter 5

Conclusion

This thesis set out to develop a system that could be used to augment the ranging abilities of mixed reality devices. To achieve this, a method of environment mapping was developed that is relative to the system’s localized space using only commodity off the shelf components. We describe methods which provide mixed reality devices with the geometry of a larger surrounding space (walls, ceiling) and a method for capturing the size of objects within the environment.

5.1 Discussion

While the system has been developed, implemented and tested, many improvements are possible. Because of the limitations of the hardware and how the system is being used, a great deal of inaccuracies are to be expected. For example, the success of the system is
highly dependent on the user of the system. If the user of the system doesn’t provide a good amount of rotational range then many of the LIDAR points may overlap or be co-linear meaning planes can’t be constructed and meshes can not be generated.

Also, the projection of the LIDAR points on the image frame may not be accurate if the user moves with faster speed than the maximum recommended speed. The current implementation does not work well with close curved surfaces. Because of the way the segmentation works, curved surfaces are often grouped together as one flat object. However, because this system mainly focuses on those far away this does not play a major role. Estimating the position of the user using an IMU alone is also prone to error due to the calculations involved [27].

The success of the current implementation of the system is limited by the error and noise in the IMU sensor. The error in the yaw direction limited the ability of the system to accurately determine the system’s orientation which lead to inaccuracies in much data. Figure 5.1 shows the extent of the inaccuracies of the IMU. The region plane in the figure was captured while the system was stationary. This shouldn’t have happened because all of the LIDAR points should coincided with the same point of the image. However, due to the excessive noise in the sensor readings the system projected the angles of the LIDAR points inaccurately leading to the system creating an erroneous region plane. The orientation planes are widely inaccurate leading to the creation of meshes that are extremely large. The next generation of IMU will reduce this problem.
Figure 5.1: A stationary IMU inaccurately showing changes rotation (green line)

5.2 Lessons Learned

The major lesson learned involves the trade-offs while developing a system. Mobility is extremely important to mixed reality devices which means computation and energy resources are limited. Many of the sensors that are used can have high resolution and high sampling rates. However, due to constraints of the hardware many of these had to be reduced. For instance, the Raspberry PI 3 used to develop this system cannot easily read 30 frames per second and write each frame to memory because of the limited write speed.

Eventually this became an lesson of balance, i.e., quality versus time to compute. The
resolution of the camera could be reduced to improve processing speed however that would also reduce the accuracy of the system.

The LIDAR system could read distances at 500 samples a second. However this would introduce a lot of noise in the system. This increase in data collection rates the system would also have to perform some sort of error correction. This is not necessary considering that the human head doesn’t move that much in 1/500 of a second.

Choices also had to be made in with the IMU as well. The IMU had multiple settings for data resolution. Higher data resolution also created more noise in the data. The higher resolution also limited the range of movement that the user could perform during each second.

5.2.1 OpenCV

OpenCV is a open source computer vision library that provided many great functions [11]. It was used for color space conversion and for its median filter. The most useful aspect however was its data structure libraries. OpenCV support for the hardware used is not complete since most of the OpenCV GPU calls use CUDA which is not an option on Raspberry PI.

It is important to note that OpenCV does provide image segmentation implementations such as the watershed image segmentation. The watershed image segmentation algorithm is a form of region growing that looks at the intensities of the image as a topological map and fills the map according to the gradients of the intensities. There was a few problems with using this implementation. The first is that watershed algorithms usually run on gray scale
values instead of all color channels. The algorithm also requires that markers or points of initial flow to be picked. The success of this algorithm relies on this selection of these points.

There do exist some algorithms that pick these points but they don’t always provide accurate results and most examples using the watershed algorithm show the user picking the initial points.

5.2.2 RTIMULib

The real-time IMU library provided many of the tools required to accurately read from the IMU. The library provided sensor fusion techniques such as the Kalman filter and it also provided its own lightweight implementation of the Kalman filter called RTQF [17]. The library also provided tools for providing Euler angles and quaternions.

While the library was very useful, it did have some problems. The most relevant and persistent problem with the library was how it stored its IMU data. The RTIMULib stores its read data on an queue that can contain up to 512 data points. Once this limit is reached, all the data points in the queue are erased. To prevent this from occurring data has to be pulled off at the same rate as the data is being collected from the IMU. This of course is a trade-off that had to be made. In order to prevent IMU data from being lost a decision was made to dedicate a entire thread to its collection from the queue. However, even then the queue is occasionally overfilled because the thread is waiting on the Async queue to be free or because of system interrupts.
5.3 Future Work

The developed system is an augmentation of current 3D environment mapping techniques for Microsoft Hololens and Google Tango. Its aim is to extend the usage of these systems by allowing the user to use these devices in larger spaces and supporting more versatile mixed reality applications.

We would also like to experiment with using different hardware configurations such as using more than one LIDAR or using a real-time operating system.

One of the limitations with this approach was the spread of LIDAR points. If the user doesn’t move much or moves too linearly many of the LIDAR points couldn’t be used to create a plane due to them being co-linear. An improvement that could be made on this system would be to add additional LIDAR sensors and point them at different angles thus increasing the number and spread of cloud points. Providing a visual feedback to the user in terms of space coverage will inform user where to move in order to improve the cloud point quality and enhance environment mapping.

IMU can be used for position tracking, at least for a short period of time [16, 27] Instead of running everything on the Raspberry PI, running the segmentation on a remote device and performing the mesh creation and data collection on the Raspberry PI could be implemented.
Bibliography


Appendix A

Hardware Implementation

Because the target systems that this work is trying to augment are lightweight, the developed system itself needed to be lightweight. There couldn’t be any moving parts and the power consumption of the system had to be low. The main controller for this system is the Raspberry PI 3 (table A.1). In table A.2 the specification for the camera used in the developed system. For the lidar used the Garmin lidar lite 3 [3] was used its specification can be found in table A.3. The IMU used by the developed system was the MPU9250 and its specification can be found in table A.4.

<table>
<thead>
<tr>
<th>Table A.1: Raspberry PI specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Clock Speed</td>
</tr>
<tr>
<td>Number of Cores</td>
</tr>
<tr>
<td>RAM</td>
</tr>
</tbody>
</table>
Table A.2: Camera specification [22]

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Still Resolution</td>
<td>8 Megapixels</td>
</tr>
<tr>
<td>Focal Length</td>
<td>3.60 +/- 0.01 mm</td>
</tr>
<tr>
<td>Horizontal Field of View</td>
<td>53.50 +/- 0.13 degrees</td>
</tr>
<tr>
<td>Vertical Field of View</td>
<td>41.41 +/- 0.11 degrees</td>
</tr>
<tr>
<td>Sensor Resolution</td>
<td>3280 x 2464 pixels</td>
</tr>
</tbody>
</table>

Table A.3: Lidar specification [3]

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>40 Meters</td>
</tr>
<tr>
<td>Accuracy</td>
<td>+/- 2.5 cm at distances greater than 1 meter</td>
</tr>
<tr>
<td>Resolution</td>
<td>1 cm</td>
</tr>
<tr>
<td>Frequency</td>
<td>500 Hz</td>
</tr>
<tr>
<td>Interface</td>
<td>I2C</td>
</tr>
</tbody>
</table>

Table A.4: IMU specification [2]

<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>+/- 2g</td>
</tr>
<tr>
<td>Gyroscopes</td>
<td>+/- 250 degrees</td>
</tr>
<tr>
<td>Interface</td>
<td>I2C</td>
</tr>
</tbody>
</table>