Detail Design Specification

Electric-Eye

Nichelous Herndon
Adis Kovacevic
Zack Davis
Juan Sifuentes
Rommel Alonzo
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>5</td>
</tr>
<tr>
<td>List of Tables</td>
<td>6</td>
</tr>
<tr>
<td>Document Revision History</td>
<td>7</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>8</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>8</td>
</tr>
<tr>
<td>1.2 Product Concept</td>
<td>8</td>
</tr>
<tr>
<td>1.3 Project Scope</td>
<td>9</td>
</tr>
<tr>
<td>2. Architecture Overview</td>
<td>10</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Module Decomposition</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Module Descriptions</td>
<td>17</td>
</tr>
<tr>
<td>3. I/O Interface Layer</td>
<td>19</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td>19</td>
</tr>
<tr>
<td>3.2 Grabber</td>
<td>20</td>
</tr>
<tr>
<td>3.3 Interpreter</td>
<td>21</td>
</tr>
<tr>
<td>4. Data Processing Layer</td>
<td>22</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Processing</td>
<td>24</td>
</tr>
<tr>
<td>4.3 Initialization</td>
<td>25</td>
</tr>
<tr>
<td>4.4 Save File</td>
<td>26</td>
</tr>
<tr>
<td>4.5 Control Camera</td>
<td>27</td>
</tr>
<tr>
<td>4.6 Mapping</td>
<td>28</td>
</tr>
</tbody>
</table>
4.7 Request Map
4.8 Localize

5. Communication Layer

5.1 Overview
5.2 Pack Matrix Module
5.3 Unpack Message
5.4 Filter

6. Quality Assurance

6.1 Overview
6.2 Module and Unit Test
6.3 Component Test
6.4 Integration Test
6.5 System Verification Test
6.6 Test Cases

7. Requirements Traceability

8. Acceptance Plan

8.1 Packaging and Installation
8.2 Acceptance Testing
8.3 Acceptance Criteria

9. Hardware

9.1 Overview
9.2 ZOTAC D2700ITXS Motherboard
9.3 ZOTAC ZT-61102-10M Graphics Card
9.4 Mushkin Enhanced 60GB mSATA Internal SSD
9.5 Crucial 4GB DDR3 SO-DIMM RAM Kit
9.6 Asus XTION PRO RGB and Depth Sensor
9.7 M4-ATX-HV 250W Intelligent DC PSU
9.8 SYBA USB Numeric Keypad
9.9 Case

10. Glossary
10.1 Robot Operating System (ROS)
10.2 Point Cloud Library (PCL)
10.3 Transformation Matrix (6DOF)
10.4 RGB-D Camera
10.5 OpenNI
10.6 Point Cloud
10.7 Point Cloud Frame
10.8 Mapping
10.9 Localization
10.10 ROS Node
List of Figures

Figure 1-1: Product Concept .......................................................... 8
Figure 2-1: Architecture Overview .................................................. 10
Figure 2-2: Data Flow Between Modules ......................................... 13
Figure 3-1: I/O Interface Layer Subsystems and Modules ...................... 19
Figure 4-1: Data Processing Layer Subsystems and Modules .................. 22
Figure 5-1: Communication Layer Subsystems and Modules .................. 32
Figure 9-1: ZOTAC D2700IITXS Motherboard ..................................... 45
Figure 9-2: ZOTAC ZT-61102-10M Graphics Card ............................... 47
Figure 9-3: Mushkin Enhanced 60GB mSATA Internal SSD .................... 48
Figure 9-4: Crucial 4GB DDR3 SO-DIMM RAM Kit .............................. 49
Figure 9-5: Asus XTION PRO RGB and Depth Sensor .......................... 50
Figure 9-6: M4-ATX-HV 250W Intelligent DC PSU ............................. 51
Figure 9-7: SYBA USB Numeric Keypad .......................................... 52
Figure 9-8: Case Components ....................................................... 53
List of Tables

Table 2-1: Data Element Descriptions ........................................................................................................15
Table 2-2: Producer-Consumer Relationship ..............................................................................................16
Table 3-1: I/O Interface Layer Dataflow Descriptions ..............................................................................19
Table 4-1: Data Processing Layer Dataflow Descriptions ........................................................................23
Table 5-1: Communication Layer Dataflow Descriptions ........................................................................32
Table 7-1: Requirements Traceability Matrix .............................................................................................42
Table 8-1: Acceptance Criteria ..................................................................................................................44
### Document Revision History

<table>
<thead>
<tr>
<th>Revision Number</th>
<th>Revision Date</th>
<th>Description</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>02/11/2013</td>
<td>Composing Initial</td>
<td>Starting the DDS Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Draft</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>02/16/2013</td>
<td>Composing Initial</td>
<td>Compiling all completed individual sections and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Draft</td>
<td>updating pictures</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>02/18/2013</td>
<td>Finalizing First</td>
<td>Integrating all remaining individual sections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Draft</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>02/20/2013</td>
<td>Finalizing First</td>
<td>Making suggested changes by our peer review team and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Draft</td>
<td>other team members</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>03/07/2013</td>
<td>Baseline</td>
<td>Making changes to the DDS final draft</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Overview

The Detail Design Specification will cover the architectural design of the Electric-Eye and include details of the product concept, scope, and key requirements.

1.2 Product Concept

The Electric-Eye is a module that will first scan and map a room using the point cloud library in 3D, then perform localization when mounted onto a robot. Users of this product will be able to integrate this product into their robots that use the robot operating system (ROS) and use the features provided by the module. This will provide an easy to use plug and play system which robot designers can easily integrate into their projects.
1.3 Project Scope

The Electric-Eye will be limited to a standard sized room. It should be an easy to use system where the module shall assist the robot by giving it precise and formatted data. The key requirements will represent the most important functionalities that the Electric-Eye needs to make it successful.

The Electric-Eye must be able to scan an average size indoor room, map a world model based on the frames stored, and localize a robot based on its location in the world map. All of these operations must be processed within a reasonable time, and it is only allowed to be in one mode at a given period. The product must also be ROS compatible to the extent that it can send the transformation matrix and receive simple standard commands from ROS. What the robot does with the information provided by the product is up to the robot designer.

For packaging requirements, the Electric-Eye has to be physically mountable to a robot (which depends on the user). It must also have a single case housing to provide minimal interference with the robot. Since safety is a great concern, the Electric-Eye must have no sharp edges and be able to prevent fires from overheating and power surges.

Other requirements that must be noted are that the Electric-Eye has to be tested to ensure functionality and be able to operate under small vibration such as a robot operating on rugged terrain. Since it will draw power from a robot, it must not draw more than 150 watts of power, and be able to have an on board storage of at least 8 GB. To finalize the product, we will include documentation in American English, and will not provide further maintenance after May 2013.
2. Architecture Overview

Figure 2-1: Architecture Overview
2.1 Overview

This section will provide a description of the architectural layers of the Electric-Eye. The Electric-Eye will have 3 main layers to its architecture. The layers are: I/O Interface, Data Processing, and Communication. Figure 2-1 shows the overall structure and subsystems that compose the Electric-Eye’s architecture. Below is a brief description of each of the layers.

2.1.1 I/O Interface Layer

**Description:** This layer will communicate with the camera, convert camera input to point cloud format, and interpret the switch's position. Any information coming in from the camera or mode switch will be interpreted in this layer and sent to the Data Processing Layer.

**Services Provided:**
- Sends point cloud data to the Data Processing Layer
- Sends commands to the camera
- Sends switch mode to Data Processing Layer

**Services Expected:**
- Raw data from the camera
- Raw data from the switch
- Camera command from the Data Processing Layer

2.1.2 Data Processing Layer

**Description:** This layer will perform read/write operations to the on-board storage component along with mapping and localization operations. All major processing will be handled in this layer, specifically the world map and transformation matrix.

**Services Provided:**
- Commands to the I/O Interface Layer for camera control
- Transformation matrix to the Communication Layer

**Services Expected:**
- Switch mode from the I/O Interface Layer
- Point cloud frame from the I/O Interface Layer
- ROS sleep command from the Communication Layer
2.1.3 Communication Layer

Description: This layer will provide the necessary communication between the module and ROS. In this layer, messages will be packed into ROS format and transmitted to ROS. This layer will also receive commands from ROS, when a message is received it will be unpacked, interpreted, and sent to the Data Processing Layer.

Services Provided:

- Packs a transformation matrix and sends it to ROS
- Unpacks and interprets ROS messages and sends them to the Data Processing Layer

Services Expected:

- A transformation matrix is expected from the Data Processing Layer
- Commands from ROS
2.2 Module Decomposition

Figure 2-2: Data Flow Between Modules
<table>
<thead>
<tr>
<th>Layer</th>
<th>Data Element</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O Interface</td>
<td>I1</td>
<td>Sends a function call: turnoff(), frameStartRequest(), or frameStopRequest() to the camera driver</td>
</tr>
<tr>
<td>I/O Interface</td>
<td>I2</td>
<td>Receives a PointCloud Frame from camera</td>
</tr>
<tr>
<td>I/O Interface</td>
<td>I3</td>
<td>Numeric Keypad input as an integer</td>
</tr>
<tr>
<td>I/O Interface</td>
<td>I4</td>
<td>Sends PointCloud Frame to the Data Processing Layer</td>
</tr>
<tr>
<td>I/O Interface</td>
<td>I5</td>
<td>Integer representing current module state</td>
</tr>
<tr>
<td>Data Processing</td>
<td>D1</td>
<td>Integer representing current module state</td>
</tr>
<tr>
<td>Data Processing</td>
<td>D2</td>
<td>Constructor function call from Controller to Store Frames subsystem</td>
</tr>
</tbody>
</table>
| Data Processing      | D3           | Function call that stores PointCloud frame into SSD  
    pcl::io::savePCDFileASCII ("filename.pcd", saved_cloud);                                                                                     |
| Data Processing      | D4           | Constructor function call from Store Frames to the Camera Handler Subsystem                                                                       |
| Data Processing      | D5           | Function call from Control camera module.getFrame() or  
    Control camera module.sleep()                                                                                                               |
| Data Processing      | D6           | PointCloud Frame                                                                                                                                                                           |
| Data Processing      | D7           | Constructor function call from Controller to Mapping subsystem                                                                                 |
| Data Processing      | D8           | Requests individual PCD Files and Saves world map  
    PointCloud  
    pcl::io::savePCDFileASCII ("filename.pcd", WorldMap);                                                                                     |
| Data Processing      | D9           | Receives requested PCD file  
    pcl::io::loadPCDFileASCII<pcl::PointXYZ> ("filename.pcd", *PointCloudFrame);                                                                     |
<table>
<thead>
<tr>
<th>Category</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Processing</td>
<td>D10</td>
<td>Constructor function call from controller to Localization subsystem</td>
</tr>
</tbody>
</table>
| Data Processing   | D11  | Requesting the world map from on-board storage `pcl::io::loadPCDFileASCII<pcl::PointXYZ>`
|                   |      | ("filename.pcd", *PointCloudFrame);                                       |
| Data Processing   | D12  | Receiving the world map from on-board storage                              |
| Data Processing   | D13  | Passes World Map PCD File to Localization module                            |
| Data Processing   | D14  | Constructor function call from Store Frames to the Camera Handler Subsystem  |
| Data Processing   | D15  | PointCloud Frame                                                            |
| Data Processing   | D16  | Calculated Transformation Matrix data structure `struct Transformation Matrix{
|                   |      | float x,y,z,roll,pitch,yaw}`                                               |
| Communication     | C1   | Transform Message data structure which contains the Vector3 translation and Quaternion rotation objects |
| Communication     | C2   | ROS topic that contains an integer value                                    |
| Communication     | C3   | Integer value that was contained in the ROS topic                           |
| Communication     | C4   | Filtered Integer value that was contained in the ROS topic                  |

Table 2-1: Data Element Descriptions
<table>
<thead>
<tr>
<th>Producer-Consumer Relationship</th>
<th>Consumer Subsystem</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I/O Interface</td>
<td>Data Processing</td>
</tr>
<tr>
<td></td>
<td>Camera</td>
<td>Data Storage</td>
</tr>
<tr>
<td></td>
<td>Grabber</td>
<td>Data Processing</td>
</tr>
<tr>
<td></td>
<td>Switch</td>
<td>Initialization</td>
</tr>
<tr>
<td></td>
<td>Interpreter</td>
<td>Request Map</td>
</tr>
<tr>
<td></td>
<td>Save File</td>
<td>Pack Matrix</td>
</tr>
<tr>
<td></td>
<td>Data Storage</td>
<td>Filter Module</td>
</tr>
<tr>
<td></td>
<td>Control Camera</td>
<td>Unpack Message</td>
</tr>
<tr>
<td></td>
<td>Processing</td>
<td>ROS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Camera</th>
<th>-</th>
<th>I2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grabber</td>
<td>I1</td>
<td>-</td>
</tr>
<tr>
<td>Switch</td>
<td>-</td>
<td>I3</td>
</tr>
<tr>
<td>Interpreter</td>
<td>-</td>
<td>I5</td>
</tr>
<tr>
<td>Save File</td>
<td>-</td>
<td>D3</td>
</tr>
<tr>
<td>Data Storage</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Control Camera</td>
<td>D5</td>
<td>D6</td>
</tr>
<tr>
<td>Processing</td>
<td>-</td>
<td>D1</td>
</tr>
<tr>
<td>Initialization</td>
<td>D2</td>
<td>-</td>
</tr>
<tr>
<td>Mapping</td>
<td>D8</td>
<td>-</td>
</tr>
<tr>
<td>Request Map</td>
<td>D11</td>
<td>-</td>
</tr>
<tr>
<td>Localize</td>
<td>D14</td>
<td>-</td>
</tr>
<tr>
<td>Pack Matrix</td>
<td>-</td>
<td>C1</td>
</tr>
<tr>
<td>Filter Module</td>
<td>C4</td>
<td>-</td>
</tr>
<tr>
<td>Unpack Message</td>
<td>C3</td>
<td>-</td>
</tr>
<tr>
<td>ROS</td>
<td>C2</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2-2: Producer-Consumer Relationship
2.3 Module Descriptions

2.3.1 Grabber Module
The Grabber Module will turn on the RGB-D camera and start reading in Point Cloud Data.

2.3.2 Interpreter Module
The Interpreter Module will receive a signal from the switch and turn the signal into valid data for the system to use.

2.3.3 Processing Module
The Processing Module will receive the data regarding switch position and determine if the switch is in a valid position.

2.3.4 Initialization Module
The Initialization Module will initialize the proper subsystems based on the switch’s position.

2.3.5 Save File Module
The Save File Module will receive Point Cloud Data and save the data to a .pcd file onto on-board storage.

2.3.6 Control Camera Module
The Control Camera Module will signal the RGB-D camera and tell it to start reading in Point Cloud Data.

2.3.7 Mapping Module
The Mapping module will access on-board storage and perform mapping algorithms to the saved .pcd files. The final world map will be stored back onto on-board storage.

2.3.8 Request Map Module
The Request Map Module will access on-board storage and load the world map into memory.

2.3.9 Localize Module
The Localize Module will receive the world map that has been loaded into memory and will also read in Point Cloud Data from the camera. Using this data, the module will calculate an accurate transformation matrix.

2.3.10 Pack Matrix Module
The Pack Matrix Module will take an array and pack it into a Transform Message and send it to ROS.
2.3.11 Unpack Message Module

The Unpack Message Module will receive a ROS topic and unpack the message.

2.3.12 Filter Module

The Filter Module will filter out all unnecessary data that comes from the Unpack Message Module. After filtering is complete, the command will be sent to the Data Processing Layer.
3. I/O Interface Layer

![Diagram of I/O Interface Layer Subsystems and Modules]

### Table 3-1: I/O Interface Layer Dataflow Descriptions

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>Sends a function call: <code>turnoff()</code>, <code>frameStartRequest()</code>, or <code>frameStopRequest()</code> to the camera driver</td>
</tr>
<tr>
<td>I2</td>
<td>Receives a PointCloud Frame from camera</td>
</tr>
<tr>
<td>I3</td>
<td>Numeric Keypad input as an integer</td>
</tr>
<tr>
<td>I4</td>
<td>Sends PointCloud Frame to the Data Processing Layer</td>
</tr>
<tr>
<td>I5</td>
<td>Integer representing current module state</td>
</tr>
<tr>
<td>D5</td>
<td>Function call from Control camera module.getFrame() or Control camera module.sleep()</td>
</tr>
</tbody>
</table>

#### 3.1 Overview

The I/O interface layer is responsible for converting information from hardware components into a usable format or into data the system can use. Hardware components include an OpenNI compatible RGB-D camera as well as a Numeric USB keypad.
3.2 Grabber

3.2.1 Overview

The Grabber is responsible for interfacing between the RGB-D camera and the rest of the Electric-Eye system.

3.2.2 Interface

The RGB-D camera interfaces with the system by using a USB connection as well as preinstalled OpenNI drivers. This module will also interface with the Control Camera module that is located in the Data Processing Layer.

3.2.3 External Dependencies

RGB-D camera

3.2.4 Internal Dependencies

Point Cloud Library

#include <pcl/io/openni_grabber.h>

3.2.5 Process / Psuedocode

```cpp
pcl::Grabber* interface = new pcl::OpenNIGrabber();

FunctionPointer f = &callbackFunction;

//Needed for grabbing a frame from RGB-D camera
interface.registerCallback(f);

//Begin receiving point clouds
interface.start();

while(!terminate){
    pause(1);
}

//stop grabber
interface.stop();
```
3.3 Interpreter

3.3.1 Overview

The interpreter provides an interface between our USB Numeric Keypad and the rest of the Electric Eye system.

3.3.2 Interface

This module will interface with the external numeric keypad via USB and with the Processing Module in the Data Processing Layer.

3.3.3 External Dependencies

USB Numeric Keypad

3.3.4 Internal Dependencies

None

3.3.5 Process / Psuedocode

```cpp
#include <iostream>

using namespace std;

... ...
... int selection;

cin >> selection;
... ...
... 
```
4. Data Processing Layer

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I4</td>
<td>Sends PointCloud Frame to the Data Processing Layer</td>
</tr>
<tr>
<td>I5</td>
<td>Integer representing current module state</td>
</tr>
<tr>
<td>D1</td>
<td>Integer representing current module state</td>
</tr>
<tr>
<td>D2</td>
<td>Constructor function call from Controller to Store Frames subsystem</td>
</tr>
<tr>
<td>D3</td>
<td>Function call that stores PointCloud frame into SSD ( \text{pcl::io::savePCDFileASCII} ) ( \text{(&quot;filename.pcd&quot;, saved_cloud);} )</td>
</tr>
<tr>
<td>D4</td>
<td>Constructor function call from Store Frames to the Camera Handler Subsystem</td>
</tr>
<tr>
<td>D5</td>
<td>Function call from Control camera module.getFrame() or Control camera module.sleep()</td>
</tr>
<tr>
<td>D6</td>
<td>PointCloud Frame</td>
</tr>
</tbody>
</table>
### Table 4-1: Data Processing Layer Dataflow Descriptions

<table>
<thead>
<tr>
<th>D7</th>
<th>Constructor function call from Controller to Mapping subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>D8</td>
<td>Requests individual PCD Files and Saves world map PointCloud</td>
</tr>
<tr>
<td></td>
<td><code>pcl::io::savePCDFileASCII (&quot;filename.pcd&quot;, WorldMap);</code></td>
</tr>
<tr>
<td>D9</td>
<td>Receives requested PCD file</td>
</tr>
<tr>
<td></td>
<td>`pcl::io::loadPCDFileASCII<a href="">pcl::PointXYZ</a> (&quot;filename.pcd&quot;,</td>
</tr>
<tr>
<td></td>
<td>*PointCloudFrame);`</td>
</tr>
<tr>
<td>D10</td>
<td>Constructor function call from controller to Localization subsystem</td>
</tr>
<tr>
<td>D11</td>
<td>Requesting the world map from on-board storage</td>
</tr>
<tr>
<td></td>
<td>`pcl::io::loadPCDFileASCII<a href="">pcl::PointXYZ</a> (&quot;filename.pcd&quot;,</td>
</tr>
<tr>
<td></td>
<td>*PointCloudFrame);`</td>
</tr>
<tr>
<td>D12</td>
<td>Receiving the world map from on-board storage</td>
</tr>
<tr>
<td>D13</td>
<td>Passes World Map PCD File to Localization module</td>
</tr>
<tr>
<td>D14</td>
<td>Constructor function call from Store Frames to the Camera Handler Subsystem</td>
</tr>
<tr>
<td>D15</td>
<td>PointCloud Frame</td>
</tr>
<tr>
<td>D16</td>
<td>Calculated Transformation Matrix data structure</td>
</tr>
<tr>
<td></td>
<td><code>struct Transformation Matrix{</code></td>
</tr>
<tr>
<td></td>
<td><code>float x,y,z,roll,pitch,yaw)</code></td>
</tr>
<tr>
<td>C4</td>
<td>Receiving a sleep command invoked as an integer.</td>
</tr>
</tbody>
</table>

#### 4.1 Overview

The Data Processing Layer is responsible for sorting and processing any data that has been passed from an RGB-D camera or has been stored in the systems internal storage. The DPL will perform three unique operations on PointCloud frames. Operations will be presented as classes in actual implementation and only one of these classes will be initialized at any one time. Operations performed will include Storing PointCloud frames, mapping, and localization operations. These operations will be known as processing subsystems.
4.2 Processing

4.2.1 Overview

The Processing module subsystem will have overall control of the Data Processing Layer.

4.2.2 Interface

This module interfaces with the Interpreter Module so that it can receive input data from the numeric keypad. The Initialization Module and Filter Module will also interface with this module.

4.2.3 External Dependencies

None

4.2.4 Internal Dependencies

Interpreter Module

4.2.5 Process / Psuedocode

```
if(Numeric number has been entered) selection = get Number from input layer.
if(CommLayer.sleep() has been invoked) selection = 3;

//Invoke initialization module
```
# 4.3 Initialization

## 4.3.1 Overview

This module will determine which of the 3 processing subsystems will be initialized.

## 4.3.2 Interface

This module will interface with the Processing Module so that it can determine which subsystems to initialize. The Save File, Mapping, and Localization Module will also interface with this module so that they can be initialized.

## 4.3.3 External Dependencies

None

## 4.3.4 Internal Dependencies

Processing Module

## 4.3.5 Process / Pseudocode

```c
switch(selection){
    case 0:
        //Initialize StoreFrames subsystem class
        //Begin reading and storing frames from RGB-D camera
        break;
    case 1:
        //Initialize Mapping subsystem class
        //Begin reading frames and performing ICP algorithm
        break;
    case 2:
        //Initialize Localization subsystem class
        //Begin reading frames from camera and perform localization ops.
        break;
    case 3:
        //Sleep command has been invoked. Disconnect camera and
        //discontinue any localizing operations
    default:
        //Filter out the number. Not a current mode or option.
```
4.4 Save File

4.4.1 Overview

Reads in PointCloud frame from Camera handler subsystem and stores frame into PCD file.

4.4.2 Interface

This module will interface with the SSD via mSATA. The Initialization and Control Camera Module will also interface with this module.

4.4.3 External Dependencies

None

4.4.4 Internal Dependencies

Initialization and Control Camera Modules are both internal dependencies for this module.

4.4.5 Process / Psuedocode

```cpp
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/io/openni_grabber.h>

Camera module = new Cameramodule();
pcl::PointCloud<pcl::PointXYZ> saved_cloud;
int counter = 0;

//Checks to see if a termination signal has been sent or if our resources haven't been overloaded
while(!terminate && resourceCheck()){
    //Read pointCloud from Camera Controller
    module.readFrame(*saved_cloud);
    //Save the pointcloud
    pcl::io::savePCDFileASCII ("frame_" + counter +".pcd", saved_cloud);
    //Increase the counter
    counter++;
}

createFile("numberOfFrames.txt", counter);
```
4.5 Control Camera

4.5.1 Overview

Control Camera module provides access to RGB-D camera, returns current RGB-D frame from to the module that request it. If sleep commands has been invokes, module will invoke a disconnect function.

4.5.2 Interface

This module will interface with the Grabber module so that it can receive PointCloud frames from the camera. Save File and Localization Modules will utilize this module to read in frames from the camera.

4.5.3 External Dependencies

None

4.5.4 Internal Dependencies

Grabber Module

4.5.5 Process / Pseudocode

```java
Function control_camera_module(PointCloud pointer, Boolean sleepCommand){
    If(sleepCommand==TRUE){
        Grabber.disconnect();
        Return;
    }
    Grabber.getFrame(PointCloud pointer);
}
```
4.6 Mapping

4.6.1 Overview
Reads PCD files from SSD and build a world map using the Iterative Closest Point Algorithm.

4.6.2 Interface
This module will interface with the on-board storage via mSATA for reading stored frames. The Initialization Module will interface with this module to ensure that it is initialized.

4.6.3 External Dependencies
None

4.6.4 Internal Dependencies
Initialization Module and on-board storage

4.6.5 Process / Psuedocode
int numberOfFrames;
int counter = 0;
ICPMapping mapping = new ICPMapping();
pcl::PointCloud<pcl::PointXYZ> saved_cloud;
pcl::PointCloud<pcl::PointXYZ> worldmap_cloud;

// Reads in number of Files
readFile("numberOfFrames.txt", &numberOfFrames);

// Sets parameters for ICP
mapping.setParameters(parameters);

// Sets initial cloud to begin mapping
readPointCloudFrame(*worldmap_cloud);
map.setWorldMap(*worldmap_cloud);

while(counter <= numberOfFrames){
    // Reads in new PointCloud frame
    readPointCloudFrame(*saved_cloud, counter);
    map.loadNextPointCloud(*saved_cloud);

    // Aligns the new pointCloud with the World Map
    map.align();

    // Increase counter
    counter++;
}

savePointCloudFrame("worldmap_cloud.pcd", *worldmap_cloud);
4.7 Request Map

4.7.1 Overview
The Request Map module loads in the world map PCD files from SSD into a PontCloud pointer. Passes sleep command to localization module.

4.7.2 Interface
This module will interface with the on-board storage via mSATA. The localization and Initialization Modules will also interface with this module.

4.7.3 External Dependencies
None

4.7.4 Internal Dependencies
Initialization Module and on-board storage

4.7.5 Process / Psuedocode

<table>
<thead>
<tr>
<th>Gets world map from SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passes world map onto PointCloud pointer</td>
</tr>
</tbody>
</table>
4.8 Localize

4.8.1 Overview

The Localize module gets the current RGB-D camera frame and, using the world map, builds a transformation matrix. If sleep command is invoked, localization operations are halted, and sleep command is passed to the Control Camera module.

4.8.2 Interface

This module will interface with the Control Camera module so that it can read in frames, Request Map Module so that it can load the created world map into memory, and the Pack Matrix Module so that it can send the calculated transformation matrix to the Communication Layer.

4.8.3 External Dependencies

None

4.8.4 Internal Dependencies

Request Map module

4.8.5 Process / Pseudocode

```cpp
#include <iostream>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include <pcl/registration/icp.h>

int main (int argc, char** argv)
{
    pcl::PointCloud<pcl::PointXYZ>::Ptr world_map (new pcl::PointCloud<pcl::PointXYZ>);
    pcl::PointCloud<pcl::PointXYZ>::Ptr cloud_out (new pcl::PointCloud<pcl::PointXYZ>);

    pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp;
    icp.setInputCloud(world_map);
    icp.setInputTarget(cloud_in);
    pcl::PointCloud<pcl::PointXYZ> Final;
    icp.align(Final);
    icp.getFinalTransformation();

    return (0);
}
```
5. Communication Layer

Table 5-1: Communication Layer Dataflow Descriptions

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D16</td>
<td>Calculated Transformation Matrix data structure</td>
</tr>
<tr>
<td></td>
<td><code>struct Transformation Matrix{</code></td>
</tr>
<tr>
<td></td>
<td><code>float x,y,z,roll,pitch,yaw</code></td>
</tr>
<tr>
<td>C1</td>
<td>Transform Message data structure which contains the Vector3 translation and Quaternion rotation objects</td>
</tr>
<tr>
<td>C2</td>
<td>ROS topic that contains an integer value</td>
</tr>
<tr>
<td>C3</td>
<td>Integer value that was contained in the ROS topic</td>
</tr>
<tr>
<td>C4</td>
<td>Filtered Integer value that was contained in the ROS topic</td>
</tr>
</tbody>
</table>

Figure 5-1: Communication Layer Subsystems and Modules
5.1 Overview

The Communication Layer sends and receives ROS messages. This layer serves as the link between the Electric-Eye and ROS. When a transformation matrix is sent to this layer, it is packed into a ROS message and transmitted. ROS will send commands to this layer to send the module into a low power state, these messages will be unpacked and interpreted so that the module responds the way that ROS requests.

5.2 Pack Matrix Module

5.2.1 Overview

The Pack Matrix module will take Transformation Matrix data structure and pack it into a Transform Message and send it to ROS.

5.2.2 Interface

Pack Matrix will listen for incoming data from the Data Processing Layer. After receiving this data it will pack a Transform Message and send the message to ROS. This layer will interface with the external Robot Operating System by publishing a ROS topic to a ROS Subscriber Node.

5.2.3 External Dependencies

ROS

5.2.4 Internal Dependencies

This module is dependent on receiving a Transformation Matrix data structure from the Localize Module.
### 5.2.5 Process / Pseudocode

```c
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>

void Callback(float transMatrix[]){
    static tf::TransformBroadcaster br;
    tf::Transform transform;
    transform.setOrigin( tf::Vector3(transMatrix[0], transMatrix[1], transMatrix[2]) );
    transform.setRotation( tf::Quaternion(transMatrix[3], transMatrix[4], transMatrix[5]) );
    br.sendTransform(tf::StampedTransform(transform, ros::Time::now(), "world", 
                                           "Transformation Matrix"));
}

int main(int argc, char** argv){

    ros::NodeHandle node;
    ros::Subscriber sub = node.subscribe("TransformationMatrix", 10, &Callback);

    ros::spin();
    return 0;
}
```
5.3 Unpack Message

5.3.1 Overview
The Unpack Message module will receive a ROS topic and unpack the message.

5.3.2 Interface
Unpack Message will listen for incoming data from a ROS publisher node that it is subscribed to. When a ROS topic is received by this module, it will unpack the message and send the unpacked data to the Filter module.

5.3.3 External Dependencies
This module is dependent on a ROS topic which comes from the external ROS that is operating the robot.

5.3.4 Internal Dependencies
None

5.3.5 Process / Pseudocode

```c++
#include "ros/ros.h"
#include "std_msgs/String.h"

void Callback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "listener");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("chatter", 1000, Callback);
    ros::spin();
    return 0;
}
```
5.4 Filter

5.4.1 Overview

The Filter module will filter out all unnecessary data that comes from ROS.

5.4.2 Interface

The Filter module will receive an integer value from the Unpack Message module and compare the value to a list of supported commands. If a command is supported then the module will send the command to the Processing Module in the Data Processing Layer, if a command is not supported then the module will disregard the data.

5.4.3 External Dependencies

None

5.4.4 Internal Dependencies

This module is depends on the Unpack Message module to send an integer value.

5.4.5 Process / Pseudocode

```c
int main(int argc)
{
    filter(argc)
}

//cmdList[0] = the integer value for sleep
//cmdList[1] = the integer value for wake
bool filter(int command)
{
    if(command == cmdList[0] or command == cmdList[1])
        return true;
    else
        return false;
}
```
6. Quality Assurance

6.1 Overview

The Electric-Eye will undergo a series of tests that will verify that each module, unit, and component meet the requirements specified in the System Requirements Specification document. There will also be further test cases to ensure that the integration of these and the system as a whole will be verified.

6.2 Module and Unit Test

6.2.1 I/O Interface Layer

6.2.1.1 RGB-D CAMERA INTERPRETER

The subsystem shall be tested to ensure that it can read information from an OpenNI compatible RGB-D camera and convert it into a usable format.

6.2.1.2 SIGNAL INTERPRETER

The subsystem shall be tested to ensure that it can take in an analogue signal from a switch and convert it into the corresponding digital signal.

6.2.2 Data Processing Layer

6.2.2.1 CONTROLLER

The subsystem shall be tested to ensure that it sends the appropriate control signals to the other subsystems and receive control state signal.

6.2.2.2 STORE FRAMES

The subsystem shall be tested to ensure that it can store depth frames into the module’s internal storage.

6.2.2.3 CAMERA HANDLER

The subsystem shall be tested to ensure that it sends appropriate camera control commands and receive frames.

6.2.2.4 LOCALIZATION

The subsystem shall be tested to ensure that it can compute the current transformation matrix based on the current frames provided and the world map.
6.2.2.5  MAPPING
The subsystem shall be tested to ensure that it can receive frames from the internal storage, concatenate them into a world map, and store the map back into storage.

6.2.3  Communication Layer
6.2.3.1  PACK TRANSFORMATION MATRIX
The subsystem shall be tested to ensure that it parses the transformation matrix into a ROS compatible message and send it out of the module.

6.2.3.2  COMMAND INTERPRETER
The subsystem shall be tested to ensure that it filters incoming messages for appropriate commands, parses and unpacks acceptable ROS messages, and forward them to the Data Processing Layer’s Controller subsystem.

6.3  Component Test
6.3.1  RGB-D Camera
The component shall be tested to ensure that it will read in

6.3.2  Number Pad
The component shall be tested to ensure that it can receive the specific mode command from the user and forward that command to the Electric-Eye.

6.3.3  Case
The component shall be tested to ensure that it encases the Electric-Eye completely and has no sharp edges.

6.3.4  Computer
The component shall be tested to ensure that it will be the appropriate environment to hold and execute all appropriate software and hardware components of the Electric-Eye.

6.3.5  Power Supply
The component shall be tested to ensure that it will be able to draw power from the robot and supply it to the Electric-Eye.
6.4 Integration Test

The Electric-Eye will run through a series of tests to make sure that the integration of the layers are acceptable by confirming the actions below are completed successfully.

6.4.1 I/O Interface Layer

- Receives raw data from the RGB-D camera
- Receives raw data from the switch
- Sends commands to the camera
- Sends PCL data to the Data Processing layer
- Sends formatted switch mode to the Data Processing layer

6.4.2 Data Processing Layer

- Receives switch mode command from the I/O Interface layer
- Receives PCL frame from the I/O Interface layer
- Receives ROS command
- Sends camera commands to the I/O Interface layer
- Sends transformation matrix to the Communication layer

6.4.3 Communication Layer

- Receives transformation matrix from the Data Processing layer
- Receives command requests from ROS
- Sends the interpreted and appropriate ROS command

6.5 System Verification Test

The Electric-Eye will go through a System Verification Test to confirm that all high level requirements in the System Requirements Specification document are satisfied. Further details about these requirements can be seen in the section 8.3.

6.6 Test Cases

6.6.1 Electric-Eye is off and switch is changed to scan mode

The Electric-Eye will be turned on and change to scan mode. It will start reading in frames and store them into the on-board storage.

6.6.2 Electric-Eye is in off mode and switch is changed to map mode

The Electric-Eye will be turned on and change to map mode. If there are stored frames, it will create the map and store them back into storage.
6.6.3 Electric-Eye is in off mode and switch is changed to localize mode
The Electric-Eye will be turned on and change to localize mode. If there is any world map in storage, it will begin reading frames and produce the transformation matrix.

6.6.4 Electric-Eye is in off mode and receives a ROS sleep command
The Electric-Eye will ignore this command and stay in off mode.

6.6.5 Electric-Eye is in scan mode and is turned off
The Electric-Eye will finish storing frames into storage and is turn off.

6.6.6 Electric-Eye is in scan mode and switch is changed to map mode
The Electric-Eye will finish storing frames into storage and switch into map mode. If there are stored frames, it will create the map and store them back into storage.

6.6.7 Electric-Eye is in scan mode and switch is changed to localize mode
The Electric-Eye will finish storing frames into storage and changes to localize mode. If there is any world map in storage, it will begin reading frames and produce the transformation matrix.

6.6.8 Electric-Eye is in scan mode and receives a ROS sleep command
The Electric-Eye will finish storing frames into storage and turn the camera to sleep.

6.6.9 Electric-Eye is in map mode and is turned off
The Electric-Eye will finish creating and storing the world map and turn off.

6.6.10 Electric-Eye is in map mode and switch is changed to scan mode
The Electric-Eye will finish creating and storing the world map and switch to scan mode. It will start reading in frames and store them into the on-board storage.

6.6.11 Electric-Eye is in map mode and switch is changed to localize mode
The Electric-Eye will finish creating and storing the world map and switch to localize mode. If there is any world map in storage, it will begin reading frames and produce the transformation matrix.

6.6.12 Electric-Eye is in map mode and receives a ROS sleep command
The Electric-Eye will finish creating and storing the world map and put the camera to sleep.

6.6.13 Electric-Eye is in localize mode and is turned off
The Electric-Eye will finish sending the transformation matrix and turn off.
6.6.14 **Electric-Eye is in localize mode and switch is changed to scan mode**

The Electric-Eye will finish sending the transformation matrix and switch to scan mode. It will start reading in frames and store them into the on-board storage.

6.6.15 **Electric-Eye is in localize mode and switch is changed to map mode**

The Electric-Eye will finish sending the transformation matrix and switch to map mode. If there are stored frames, it will create the map and store them back into storage.

6.6.16 **Electric-Eye is in localize mode and receives a ROS sleep command**

The Electric-Eye will finish sending the transformation matrix and put the camera to sleep.

6.6.17 **Electric-Eye is in any mode and switch is changed to an unknown input**

The Electric-Eye will continue its normal process and ignore the input.

6.6.18 **Electric-Eye is in any mode and it receives an unknown ROS input.**

The Electric-Eye will continue its normal process and ignore the input.
### 7. Requirements Traceability

#### Electric-Eye Modules

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Grabber</th>
<th>Interpreter</th>
<th>Processing</th>
<th>Initialization</th>
<th>Save File</th>
<th>Control Camera</th>
<th>Mapping</th>
<th>Request Map</th>
<th>Localize</th>
<th>Pack Matrix</th>
<th>Unpack Message</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 ROS Compatible</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3.2 Mapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3.3 Localization</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3.7 RGB-D Camera</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.8 On-board Storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>8.3 Mode Switch</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 7-1: Requirements Traceability Matrix*
8. Acceptance Plan

8.1 Packaging and Installation

The Electric-Eye shall have the following physical components installed and ready in the final deliverable.

- RGB-D Camera
- Number Pad
- Case
- Computer
- Graphics Card
- Power Supply

8.2 Acceptance Testing

The Electric-Eye shall go through a series of tests to confirm that each layer, subsystem, and component has met the necessary requirements. This method will verify that our product will meet all acceptable criteria. The process will be described in more detail in the system test plan.

8.3 Acceptance Criteria

The following criteria must be met in order for the Electric-Eye to be considered acceptable.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Robot Operating System (ROS) Compatible</td>
<td>The module must communicate with the ROS and send the robot the transformation matrix data via Ethernet.</td>
</tr>
<tr>
<td>3.2 Mapping Operations</td>
<td>Using locally stored frames, the Electric-Eye will build a world map of the surrounding environment</td>
</tr>
<tr>
<td>3.3 Localization Operations</td>
<td>Using the built world map the Electric-Eye must be able to calculate where it is located inside of the map and which direction it is facing or tilting. The data calculated will be a 6DOF</td>
</tr>
<tr>
<td>3.4 Operate Under Vibrations</td>
<td>The module shall be able to operate under an environment where vibrations may occur, such as when the robot travels through rugged terrain</td>
</tr>
<tr>
<td>3.5 Power Requirement</td>
<td>The Electric-Eye should connect to a robot with a 2-terminal connection.</td>
</tr>
<tr>
<td>3.6</td>
<td>Robot Commands</td>
</tr>
<tr>
<td>3.7</td>
<td>RGB-D Camera</td>
</tr>
<tr>
<td>3.8</td>
<td>On-board Storage</td>
</tr>
<tr>
<td>4.1</td>
<td>Single Case Housing</td>
</tr>
<tr>
<td>4.3</td>
<td>Physical Mounting</td>
</tr>
<tr>
<td>5.1</td>
<td>Response Time</td>
</tr>
<tr>
<td>5.2</td>
<td>Processing Time</td>
</tr>
<tr>
<td>5.3</td>
<td>Operating Temperature</td>
</tr>
<tr>
<td>6.1</td>
<td>Heat Safety</td>
</tr>
<tr>
<td>6.2</td>
<td>Power Surge Protection</td>
</tr>
<tr>
<td>6.3</td>
<td>No Sharp Edges</td>
</tr>
<tr>
<td>7.1</td>
<td>Code Documentation</td>
</tr>
<tr>
<td>7.2</td>
<td>Testing</td>
</tr>
<tr>
<td>7.3</td>
<td>Maintenance</td>
</tr>
<tr>
<td>8.1</td>
<td>Documentation in American English</td>
</tr>
<tr>
<td>8.2</td>
<td>Cost Effective</td>
</tr>
<tr>
<td>8.3</td>
<td>Mode Switch</td>
</tr>
</tbody>
</table>

**Table 8-1: Acceptance Criteria**
9. Hardware

9.1 Overview

The Electric-Eye uses various components to complete its designed objective. In this section we describe the hardware elements that make up the system. Components shall be described by stating their purpose, specifications and interfaces.

9.2 ZOTAC D2700ITXS Motherboard

9.2.1 Purpose

The ZOTAC motherboard will be the foundation of our hardware system. It provides an foundation for all other hardware components to be attached onto. Included with this motherboard is an onboard Intel Atom 2.13 GHz Dual Core CPU which will allow us to run our code and OS.
9.2.2 Specifications

The ZOTAC motherboard has the following features:

- 2.13 GHz Intel Processor
- 1x mSATA port
- 1x PCI Express port
- 2x 204-pin DDR3 DIMM slots
  - Up to 4 GB ram
- 6x USB Interface ports

9.2.3 Interface

The ZOTAC motherboard provides an interface for our NVIDIA graphics card, SSD hard drive, 2x 2GB DDR3 RAM, USB Numeric Keypad, and our ASUS PRO RGB and Depth Sensor. As well as providing an interface, it also interfaces with our M4-ATX-HV PSU Power Supply Unit in order to fulfill its power requirements.
9.3 ZOTAC ZT-61102-10M Graphics Card

9.3.1 Purpose
The main purpose of the ZOTAC ZT-61102-10M Graphics Card is to provide the Electric-Eye system with enough resources to run all PointCloud operations within a reasonable amount of time.

9.3.2 Specification
The ZOTAC ZT-61102-10M Graphics Card includes a GeForce GTX650 Ti GPU built using NVIDIA's new Kepler Architecture which is essential for allowing certain PCL processes to run in parallel.

9.3.3 Interface
Interfaces with the ZOTAC motherboard using the PCI Express 3.0 x16 slot.
9.4 Mushkin Enhanced 60GB mSATA Internal SSD

9.4.1 Purpose
The Mushkin SSD allows the Electric-Eye to provide permanent storage to important system dependencies such as PCL, Lubuntu and ROS.

9.4.2 Specification
The Mushkin SSD interfaces using a mSATA 3.0 port. This means that we can read and write up to 505 MB/s.

9.4.3 Interface
The ZOTAC motherboard provides a mSATA interface onto which the Muskin SSD can be mounted on.
9.5 Crucial 4GB DDR3 SO-DIMM RAM Kit

9.5.1 Purpose
The Crucial RAM provides short term storage for any data that the system is using.

9.5.2 Specification
4 GB of DDR3 RAM.

9.5.3 Interface
Interfaces to the ZOTAC motherboard through its 2x204 pin SO-DIMM slots.
9.6 Asus XTION PRO RGB and Depth Sensor

9.6.1 Purpose

The purpose of this Asus RGB-D camera is to read color and depth information of its current environment.

9.6.2 Specification

The Asus camera provides a 1280x1024 color and depth frame at a rate of 30fps.

9.6.3 Interface

Interfaces to the ZOTAC motherboard through a Universal Serial Bus port or USB.
9.7 M4-ATX-HV 250W Intelligent DC PSU

9.7.1 Purpose
The PSU acts as a ATX PSU by converting power from a car, boat, or RV battery into power usable by most general-purpose motherboards.

9.7.2 Specification
Designed to provide power and control the motherboards ON/OFF switch based on the ignition status, M4-ATX-HV provides a wide input voltage (6-34V) and fits onto any motherboard with a 20-24 pin ATX connector.

9.7.3 Interface
The M4 connects to a 2 terminal power supply unit located on the mounted robot and an ATX connector located on the motherboard.
9.8 SYBA USB Numeric Keypad

9.8.1 Purpose

The numeric keypad takes in user input to change between the different modes of the Electric-Eye. Modes include: Mapping, Localizing, and Scanning.

9.8.2 Specification

19 key USB Numeric Keypad

9.8.3 Interface

Interfaces to the ZOTAC motherboard through a Universal Serial Bus port or USB.
9.9 Case

1.1.1 Purpose
The case for the Electric Eye provides housing and protection for the hardware components and a way to mount the Electric Eye to a robot.

1.1.2 Specification
The case will be about 7.5 in X 7.5 in X 7.5 in size. It will be made from cut acrylic panels and 4 7.5 inch bars of AL 6061 8020 extrusions.

1.1.3 Interface
Houses all other components.
10. Glossary

10.1 Robot Operating System (ROS)

The Robot Operating System, or ROS, is an operating system that can be installed on top of Linux or Microsoft Windows. This operating system provides libraries and tools to help software developers create robot applications. Some of the features that ROS offers are hardware abstraction, device drivers, message-passing and more. More information about ROS can be found at: www.ros.org

10.2 Point Cloud Library (PCL)

Point Cloud Library, or PCL, is a large scale open project for 2D/3D image and point cloud processing. PCL contains many algorithms that are useful for 3D image processing such as noise filtering. PCL is open source software that is free for commercial and research use. More information about PCL can be found at: www.pointclouds.org

10.3 Transformation Matrix (6DOF)

The transformation matrix, or 6DOF, is a term used throughout the document. The transformation matrix is a set of data that will be sent to ROS from the Electric-Eye which will include the pitch, roll, yaw, x, y, and z data of the current location that the robot is located in respect to the camera.

10.4 RGB-D Camera

A special camera that reads frames like a normal camera but can represent the depth of the image using color. (Red means near while blue means far). Thus this type of camera can be very useful in helping robot navigate. More information can be found at: http://www.hizook.com/blog/2010/03/28/low-cost-depth-cameras-aka-ranging-cameras-or-rgb-d-cameras-emerge-2010

10.5 OpenNI

OpenNI is an open source software development kit used for the development of 3D sensing libraries and applications. More information can be found at: http://www.openni.org/
10.6 Point Cloud

Point Cloud is a set of vertices in 3D space that include x, y, and z coordinates. It is used by 3D sensing cameras to capture and store 3D images. More information can be found at: http://en.wikipedia.org/wiki/Point_cloud

10.7 Point Cloud Frame

A camera screen shot that holds point cloud data of an area.

10.8 Mapping

Mapping is the process of collecting data of one’s environment and using it to construct a type of map that can be used for localization. More information can be found at: http://en.wikipedia.org/wiki/Robot_localization

10.9 Localization

Localization is the process of determining one’s position and direction in a given environment. More information can be found at: http://en.wikipedia.org/wiki/Navigation

10.10 ROS Node

A node is processes that are used to help control robot functionality by communicating with other nodes. More information can be found at: http://www.ros.org/wiki/Nodes