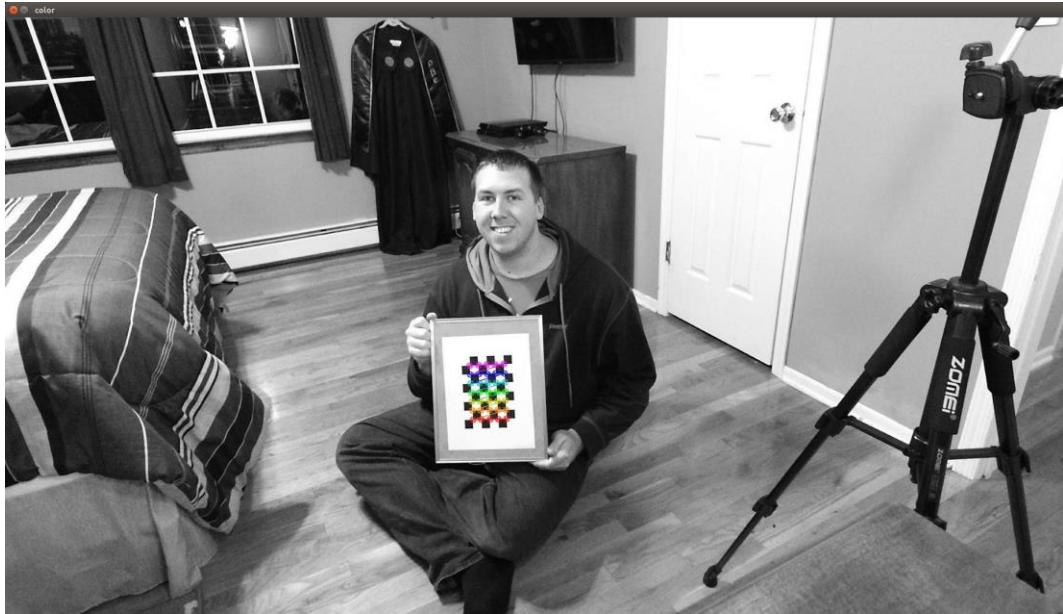


# Using Xbox Kinect V2 Within Robot Operating System for Outdoors Obstacle Detection

Written by  
Robert Crimmins (ECE/RBE)

Advised by  
Professor Alexander Wyglinski (ECE/RBE)



May 2018 – December 2018

*A Directed Research Submitted to the Faculty of Worcester Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of Master of Science.*

*This report represents the work of WPI graduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see <https://www.wpi.edu/academics/graduate>.*

## Table of Contents

Cover Page	
Table of Contents	
List of Figures	
List of Tables	
Chapter 1: Introduction	4
Chapter 2: Setting Up the Environment	6
Chapter 3: Calibrating the Kinect and Establishing a Bridge to ROS	10
Chapter 4: Performance and Results	19
Chapter 5: Conclusion	25
Reference URLs	26
Appendices	31
Appendix A: calib_color.yaml	31
Appendix B: calib_depth.yaml	32
Appendix C: calib_ir.yaml	33
Appendix D: calib_pose.yaml	34

## List of Figures

- Figure 1: RGB Camera and IR Sensor Orientation on Kinect Mainboard
- Figure 2: Pre-Calibration Photo – RGB and IR Sensor are Not Aligned
- Figure 3: RGB Camera Calibration Example
- Figure 4: IR Sensor Calibration Example
- Figure 5: Physical Setup of Kinect and Calibration Photo
- Figure 6: RGB Camera Calibration Sample Set
- Figure 7: IR Sensor Calibration Sample Set
- Figure 8: Distortion Coefficients Within the YAML Files
- Figure 9: Post-Calibration Photo – RGB and IR Sensors Aligned
- Figure 10: Outdoors Testing Layout
- Figure 11: Height Comparison to Curb
- Figure 12: Protonect Test Software Confirms Kinect Configuration
- Figure 13: RViz Showing Cans vis Point Cloud ROSTopic
- Figure 14: 3D Isometric View of Color Mapped Point Cloud
- Figure 15: 3D Top View of Color Mapped Point Cloud
- Figure 16: 3D Front View of Color Mapped Point Cloud
- Figure 17: 2D Front View of Color Mapped Point Cloud
- Figure 18: 2D Front View of Grayscale Point Cloud
- Figure 19: 2D Front View of Infrared Raw Image
- Figure 20: Protonect Viewer – Box 2 Feet Away from Kinect
- Figure 21: Protonect Viewer – Box 7 Feet Away from Kinect

## List of Tables

- Table 1: Comparison between the Kinect V1 and Kinect V2:

## Chapter 1: Introduction

The purpose of this directed research is to replicate the works of Javier Hernandez-Aceituno, Rafael Arnan, Jonay Toledo, and Leopoldo Acosta with obstacle detection outdoors. The goal is to integrate cost-effective off-the-shelf hardware to assist autonomous vehicles in navigation, mapping, and planning. Offerings such as the Microsoft Xbox Kinect were developed with advanced sensors at competitive prices for consumers. These devices feature wide-angle time-of-flight cameras, active infrared sensors, and high-resolution RGB cameras. This sensor suite aligns perfectly with the needs for low speed autonomous vehicles.

This research will be utilized on a drive-by-wire architecture golf cart being developed at the Wireless Innovation Laboratory at Worcester Polytechnic Institute. The hardware focused on in this paper is the Microsoft Xbox Kinect V2. Its predecessor, the Kinect V1, was released in November 2010 and has been widely adopted by the robotics and computer vision communities. Many hardware improvements have been made to the second-generation device as seen in the table below:

Table 1: Comparison between the Kinect V1 and Kinect V2:

Feature	Kinect for Windows 1	Kinect for Windows 2
Color Camera	640 x 480 @30 fps	1920 x 1080 @30 fps
Depth Camera	320 x 240	512 x 424
Max Depth Distance	~4.5 M	~4.5 M
Min Depth Distance	40 cm in near mode	50 cm
Horizontal Field of View	57 degrees	70 degrees
Vertical Field of View	43 degrees	60 degrees
Tilt Motor	yes	no
Skeleton Joints Defined	20 joints	26 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0

Although there are many pros associated with the second-generation Kinect, there are some limitations of utilizing this sensor. The Kinect V2 was not widely adopted with its public debut in February 2012 and was considered by Microsoft's intended audience, the gaming community, as a market failure. As of October 25, 2017, the Xbox Kinect V2 was officially discontinued and production ceased. Furthermore, the USB 3.0 / AC adapter necessary to use this sensor was

discontinued in January 2018. There are very few third-party alternatives available now offering the required adapter which makes utilization and further development more prohibitive. Despite these hurdles, the sensor's performance should still be researched for low speed autonomous vehicles.

The results we are attempting to replicate are from the University of La Laguna in Spain. Their team focused on classifying smooth, tilted, and navigable surfaces. They proved that the Kinect is superior to other technologies such as ultrasonic sensors or stereoscopic cameras in performance. The only downside was mitigating external light radiation which they combatted by keeping the sensor close to ground level. They claimed at low heights the sensor could differentiate between its own reflected infrared beams from external light radiation. For this research we will look at the following: setting up the environment, calibrating the cameras, establishing a bridge between raw data and Robot Operating System (ROS), and assessing the performance of the sensor.

## Chapter 2: Setting Up the Environment

I faced many challenges in setting up the software environment and I want to express my rationale for my implementation. Initially, I wanted to use the Xbox Kinect through an Ubuntu virtual machine through Window's Hyper-V. After some research, Hyper-V does not support Enhanced Sessions, which allows direct passthrough of USB devices to an Ubuntu virtual machine. Although there were some alternatives around this, support was minimal, and functionality was not guaranteed due to generic devices for virtualized hardware.

I decided to go with a bare metal installation on a standalone computer to minimize some technical issues. Due to the increasing difficulty of finding cost-effective hardware for the Kinect V2, software development has also slowed down. I opted to use Ubuntu 14.04 LTS as others have reported stable performance on this build. I installed Robot Operating System (ROS) for Ubuntu Indigo from the following link:

<http://wiki.ros.org/indigo/Installation/Ubuntu>

using the following commands:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release
-sc) main" > /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --
recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

sudo apt-get update

sudo apt-get install ros-indigo-desktop-full

apt-cache search ros-indigo
```

I then initialized rosdep using the following commands:

```
sudo rosdep init

rosdep update
```

I made a small quality-of-life fix so my ROS environment variables are automatically added to bash every time a new terminal was opened:

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

We then set up the ROS environment by ensuring the `ROS_ROOT` and `ROS_PACKAGE_PATH` is properly set using the following command:

```
printenv | grep ROS
```

We then set up our our `.sh` files using the following command:

```
source /opt/ros/indigo/setup.bash
```

We then install Catkin, which is a low-level build system macro for ROS, from the following site:

<http://wiki.ros.org/catkin>

using the following commands:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
echo $ROS_PACKAGE_PATH
```

The following should be returned in terminal:

```
/home/<youruser>/catkin_ws/src:/opt/ros/indigo/stacks
```

I then installed `libfreenect2`, which is an open source driver for the Kinect V2 from the following link:

<https://github.com/OpenKinect/libfreenect2>

by using the following commands:

```
git clone https://github.com/OpenKinect/libfreenect2.git
cd libfreenect2
sudo apt-get install libfreenect
cd depends; ./download_debs_trusty.sh
sudo apt-get install build-essential cmake pkg-config
```

I then installed `libusb`, which is a C library that provides generic access to USB devices, using the following commands:

```
sudo dpkg -i debs/libusb*.deb
```

I then built my directory using the following commands:

```
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
make
make install
```

I then set up my udev rules to allow for device access using the following commands:

```
sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/
```

I encountered an issue here where my Kinect hardware carried the “Xbox” branding, intended for the game console, as opposed to the “Kinect for Windows” product Microsoft also produced. I had the Xbox branded unit due to easy accessibility. To fix this, we can augment our hardware IDs into the rules file we just created. This was recommended from the following post:

<https://github.com/OpenKinect/libfreenect2/issues/193>

To look for the hardware ID’s, I used the following commands:

```
$usb-devices
$dmesg | grep "045e"
```

This is where I encountered my next issue. My device was not found in the list. I came across a post with some suggestions:

<https://askubuntu.com/questions/886588/kinect-2-on-ubuntu-16-04-device-not-listed-in-lsusb>

Per their recommendation, I reinstalled my dependencies and confirmed it was a USB 3.0 issue. A user on the ROS forums recommended installing Ubuntu 14.04.2 because it has a newer kernel version and better USB 3.0 support:

<https://answers.ros.org/question/207047/kinect-v2-no-devices-connected-ive-tried-everything/>



This was easily done with the following command:

```
sudo apt-get install linux-generic-lts-wily
```

I repeated the `$usb-devices` process using the `$dmesg` command above and took note of my Kinect USB device information. I searched for “045e” as the Vendor ID, which does not change regardless of the branding on the camera. Below is the information for my device:

```
VendorID: 045e
```

```
ProductID: 02c4
```

I then navigated to the following directory:

```
/lib/udev/rules.d
```

I modified the following rules files based on the recommendation above:

```
rules.d
```

```
40-ros-indigo-libfreenect.rules
```

```
40-libfreenect0.2.rules
```

I copied the formatting generated from the rules commands above and added my ProductID (02c4) as a valid USB device to connect to. If you do not complete this step, the software later will not detect any hardware is attached to the system.

Some of the files above are read-only or otherwise protected due to the directories that they are in. You can circumvent these obstacles by using the following:

```
sudo nano <directory> / <filename>
```

`sudo` elevates permissions, and `nano` is an editor built into terminal which will not face issues when saving.

We can finally test to see if `libfreenect2` is properly installed by navigating to the build directory and entering in the following command:

```
./bin/Protonect
```

Once that is confirmed working, we can move onto installing our bridge to ROS in Chapter 3.

### Chapter 3: Calibrating the Kinect and Establishing a Bridge to ROS

We will now install `kinect2_bridge`, which will transfer the raw data from the camera to ROS via topics. These tools are available at the following repository:

[https://github.com/code-iai/iai\\_kinect2](https://github.com/code-iai/iai_kinect2)

We then run the following commands to clone the repository contents to our Catkin workspace we made earlier:

```
cd ~/catkin_ws/src/  
git clone https://github.com/code-iai/iai_kinect2.git  
cd iai_kinect2  
rosdep install -r --from-paths .  
cd ~/catkin_ws  
catkin_make -DCMAKE_BUILD_TYPE="Release"
```

Before running `kinect2_bridge`, we must run ROS core in another terminal window by using the command:

```
roscore
```

Once running, we can launch `kinect2_bridge` with the following command:

```
roslaunch kinect2_bridge kinect2_bridge.launch
```

When browsing through the terminal window, take note of the Kinect Serial Number, which we will use later to calibrate it.

We can then see what `kinect2_bridge` would pass to our ROS applications by running the following command:

```
roslaunch kinect2_viewer kinect2_viewer kinect2 sd cloud
```

The first thing you will notice is that the cameras are not calibrated to each other. This is because the cameras are physically offset from one another as seen in the picture below:

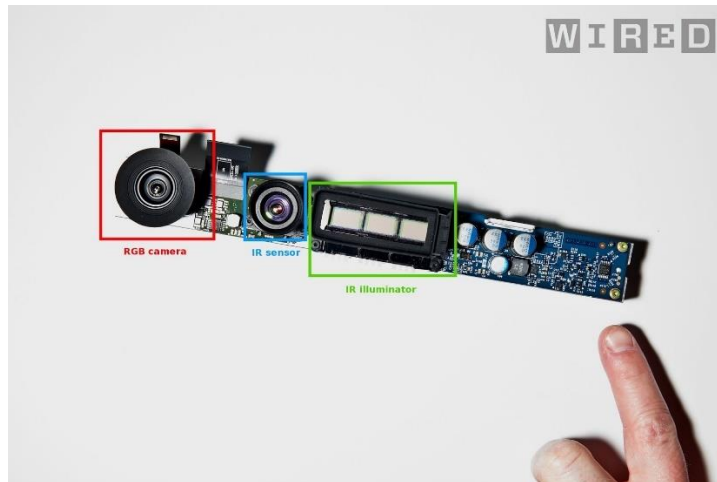


Figure 1: RGB Camera and IR Sensor Orientation on Kinect Mainboard

This offset will cause the IR data to not overlap with the RGB data. This will cause problems later when detecting obstacles because the sensor isn't portraying where objects actually are in real space.

Below is a photo of my camera before calibration. The IR Sensor point cloud data is laid on top of the RGB sensor data. Notice the corners of the box are not aligned:



Figure 2: Pre-Calibration Photo – RGB and IR Sensor are Not Aligned

We can correct this error by running a calibration process found here:

[https://github.com/code-iai/iai\\_kinect2/tree/master/kinect2\\_calibration#calibrating-the-kinect-one](https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration#calibrating-the-kinect-one)

For my calibration, I used the chess5x7x0.03.pdf pattern. It is very important you print this file without any scaling otherwise the cameras will not be properly calibrated. Using a digital caliper ensure the points of the squares are 3 cm apart diagonally. After securing the calibration image to a rigid board on a tripod, and holding the Kinect camera fixed on a second tripod we can begin calibration with the following process:

First, we limit the frame rate of the `kinect2_bridge` using the following command:

```
roslaunch kinect2_bridge kinect2_bridge _fps_limit:=2
```

Then we create a directory for the calibration data files using the following command:

```
mkdir ~/kinect_cal_data; cd ~/kinect_cal_data
```

Then we must record screenshots with the calibration photo in different orientations ideally covering the entire viewing window at different angles. I took approximately forty photos for each of the four steps, 160 total.

First, we must calibrate the RGB camera with the record command:

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 record  
color
```

The calibration process looks for changes in color horizontally and vertically in repetitious patterns. We provided the calibration script with the pattern we are using. It is now aware we will have 6x8 squares arranged in a 5" x 7" checkerboard pattern with 3 cm diagonal spacing. Considering these are given known values, it can map the number of pixels to a distance to correct any image distortion. Below are some examples:



Figure 3: RGB Camera Calibration Example

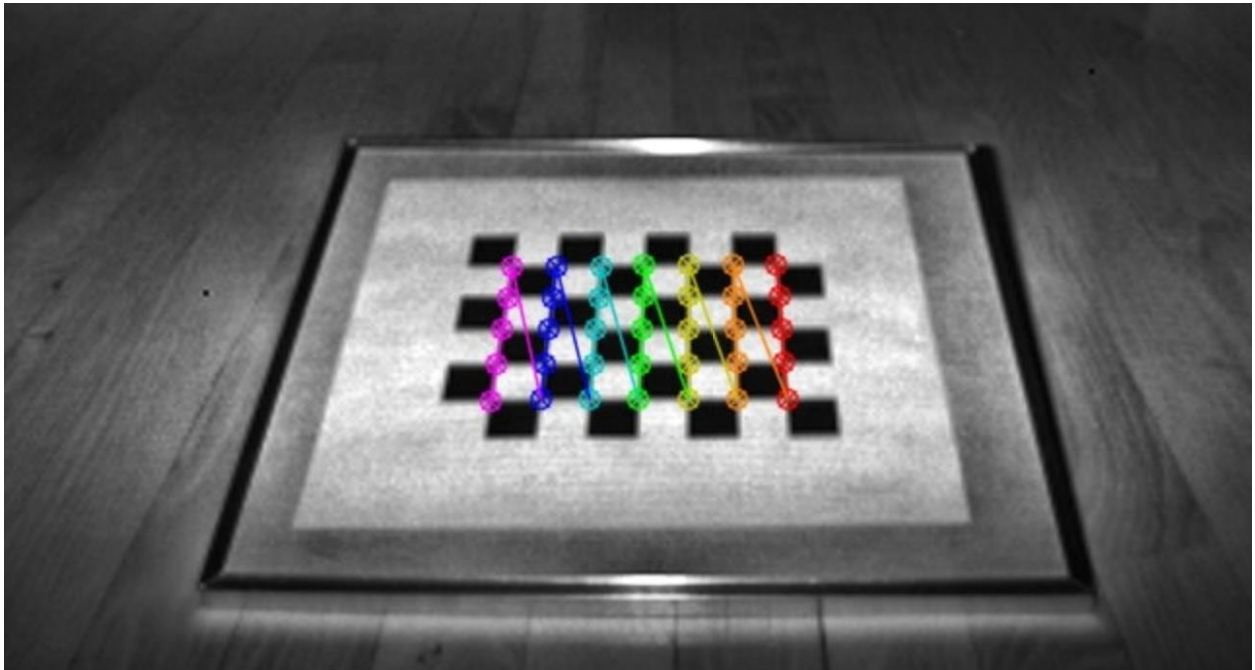


Figure 4: IR Sensor Calibration Example

After a decent sample set is captured, we must calibrate our intrinsics using the data set using the following command:

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 calibrate color
```

We repeat this process for the remaining three cycles:

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 record ir
```

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 calibrate ir
```

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 record sync
```

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 calibrate sync
```

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 record depth
```

```
roslaunch kinect2_calibration kinect2_calibration chess5x7x0.03 calibrate depth
```

This was the set up for my calibration photos:



Figure 5: Physical Setup of Kinect and Calibration Photo

I moved around the checkerboard calibration photo to fill the entire field of view for each one of the cycles (Color, IR, Sync, Depth). Below are some of my calibration photos.

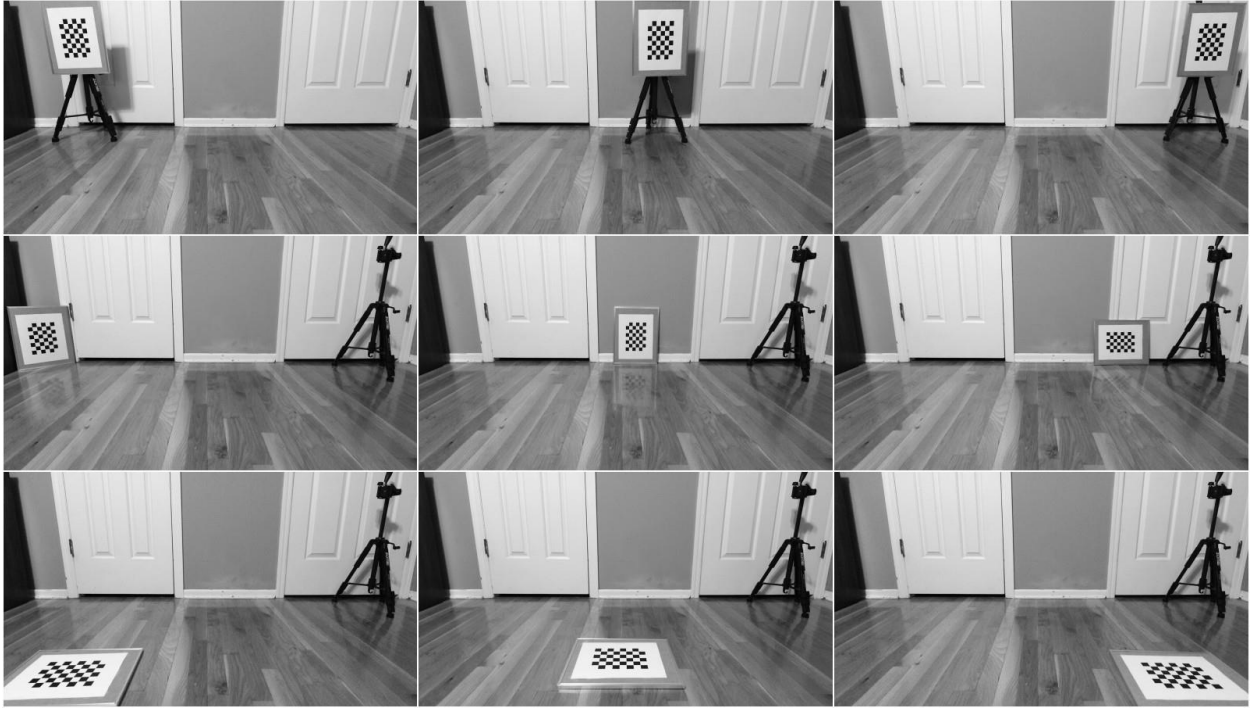


Figure 6: RGB Camera Calibration Sample Set

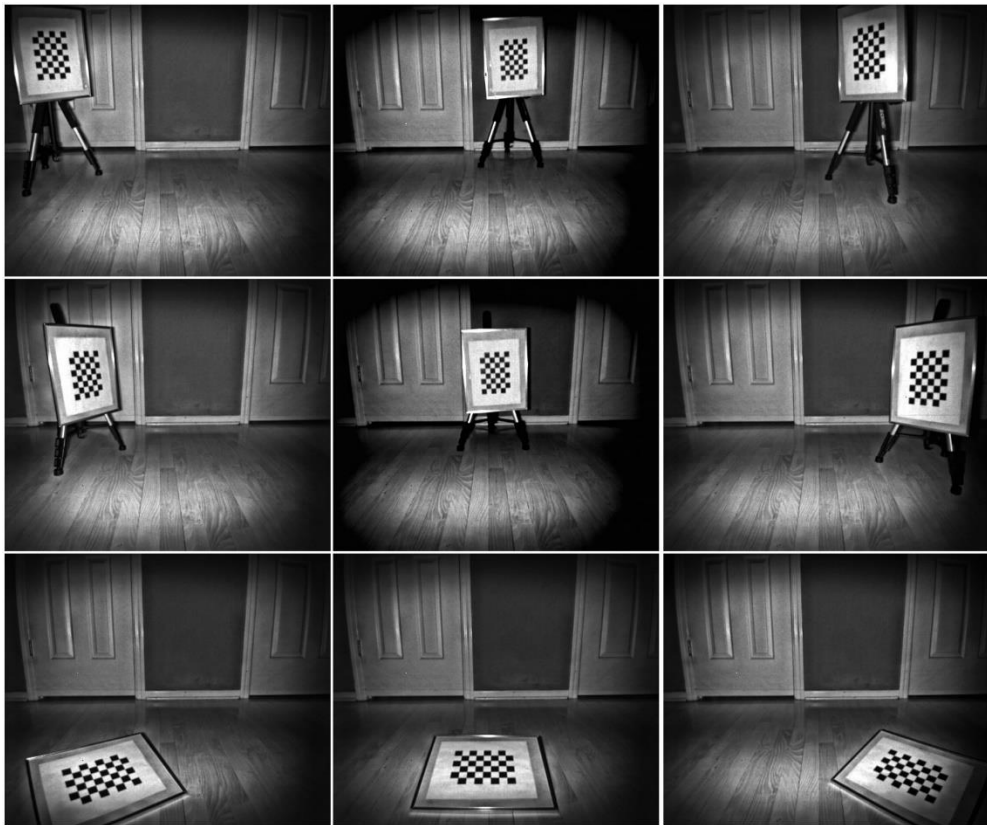


Figure 7: IR Sensor Calibration Sample Set

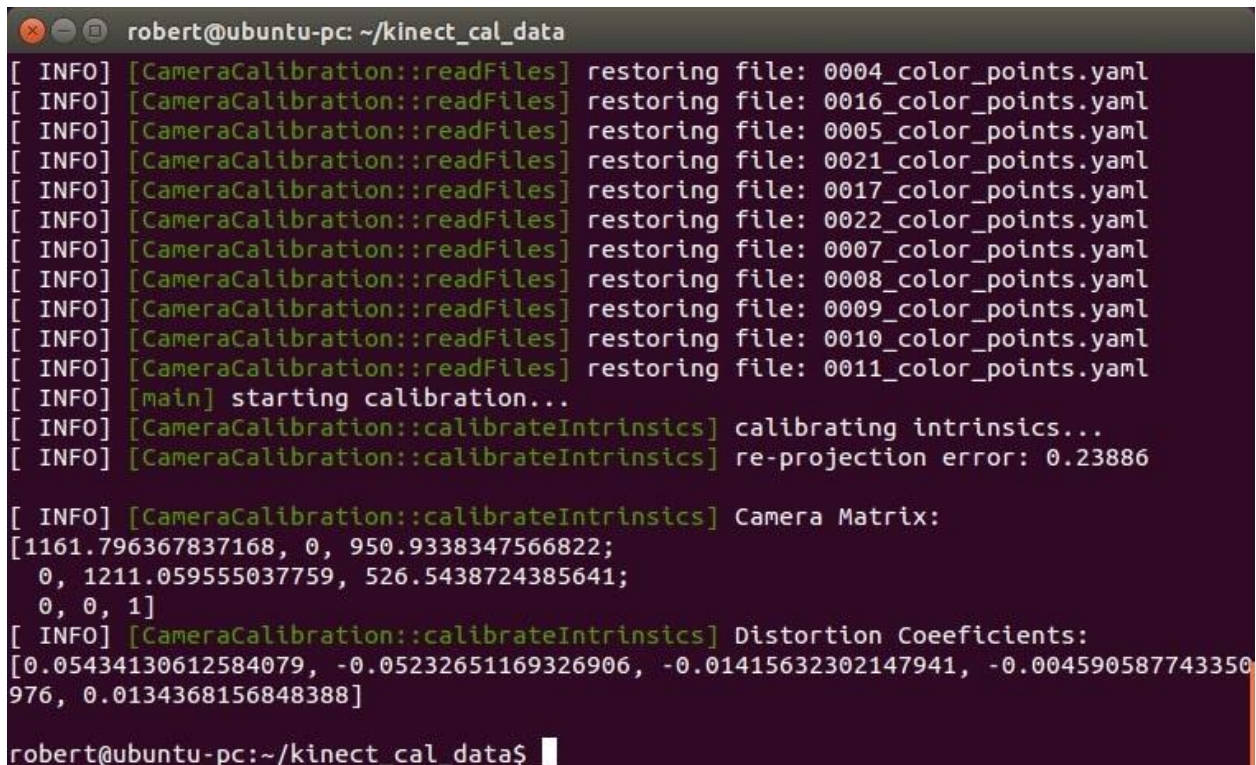
Once completed, we take our serial number captured earlier, which for my case was 021177340347 and we create a folder with that number as its name:

```
roscd kinect2_bridge/data; mkdir 021177340347
```

We now copy the following generated calibration files from the `kinect_cal_data` directory and paste them in our Catkin workspace `kinect2_bridge/data/021177340347` directory:

```
calib_color.yaml
calib_depth.yaml
calib_ir.yaml
calib_pose.yaml
```

Here is a screenshot showing the distortion coefficients within these calibration yaml files:

A terminal window titled 'robert@ubuntu-pc: ~/kinect\_cal\_data' showing the output of a calibration process. The output consists of several lines of log messages. It starts with a series of 'restoring file' messages for various color points (0004 to 0011). This is followed by 'starting calibration...' and 'calibrating intrinsics...'. A key line shows 're-projection error: 0.23886'. Below this, the 'Camera Matrix' is displayed as a 3x3 matrix of floating-point numbers. Finally, the 'Distortion Coefficients' are listed as a 1x5 array of floating-point numbers. The terminal ends with a prompt 'robert@ubuntu-pc:~/kinect\_cal\_data\$' and a cursor.

```
robert@ubuntu-pc: ~/kinect_cal_data
[ INFO] [CameraCalibration::readFiles] restoring file: 0004_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0016_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0005_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0021_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0017_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0022_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0007_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0008_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0009_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0010_color_points.yaml
[ INFO] [CameraCalibration::readFiles] restoring file: 0011_color_points.yaml
[ INFO] [main] starting calibration...
[ INFO] [CameraCalibration::calibrateIntrinsics] calibrating intrinsics...
[ INFO] [CameraCalibration::calibrateIntrinsics] re-projection error: 0.23886

[ INFO] [CameraCalibration::calibrateIntrinsics] Camera Matrix:
[1161.796367837168, 0, 950.9338347566822;
 0, 1211.059555037759, 526.5438724385641;
 0, 0, 1]
[ INFO] [CameraCalibration::calibrateIntrinsics] Distortion Coefficients:
[0.05434130612584079, -0.05232651169326906, -0.01415632302147941, -0.004590587743350
976, 0.0134368156848388]
robert@ubuntu-pc:~/kinect_cal_data$
```

Figure 8: Distortion Coefficients Within the YAML Files

Then we restart our `kinect2_bridge`, which will now use our calibration parameters, by running the following command:

```
roslaunch kinect2_bridge kinect2_bridge.launch
```



Below you will see the point cloud is now in alignment with the RGB camera. You can see the slight distortion in IR overlay to account for the physical separation between the sensors:



Figure 9: Post-Calibration Photo – RGB and IR Sensors Aligned

If your results do not look like my calibration photos, you will need to repeat the calibration process with a larger data set. My first attempt with 12 photos per cycle (48 total) was not enough. I had to expand my data set to 40 photos per cycle (160 total) to have decent results. The author of the `kinect2_calibration` repository states data sets over 100 photos per cycle (400 total) will have very accurate calibration as seen by his data charts and resulting photos:

[https://github.com/code-iai/iai\\_kinect2/tree/master/kinect2\\_calibration#calibrating-the-kinect-one](https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration#calibrating-the-kinect-one)

Once the calibration is finished, we now use the `kinect2_bridge` to publish data to ROS topics, which we can use in applications such as RViz. We use the following command:

```
roslaunch kinect2_bridge kinect2_bridge.launch publish_tf:=true
```

Then in another window, we run RViz with the following command:

```
roslaunch rviz rviz
```

We then change our Frame to `kinect2_link`. Then we can import models of our prototype vehicle, import models of the Kinect camera and its physical location in 3D Space. Then we can add features to our camera via the topics tab. Now we can assess the performance of our calibrated Kinect camera within ROS.

## Chapter 4: Results

I then moved my testing outdoors. I followed University of La Laguna's recommendation by placing the Kinect ~ 12" off the ground, tilted slightly downward. Here is a picture of my test apparatus:



Figure 10: Outdoors Testing Layout

I used a tape measure to mark one-foot increments. I placed soda cans every 12 inches apart from the Kinect. There are 6 cans with the furthest being 7 feet away from the Kinect. Soda cans are similar in height to curbs which make them great for testing, see below:



Figure 11: Height Comparison to Curb

I loaded the Protonect test software to ensure the Kinect was properly set up outdoors. Note Protonect doesn't use the calibration files we took earlier as it is a configuration verification tool.

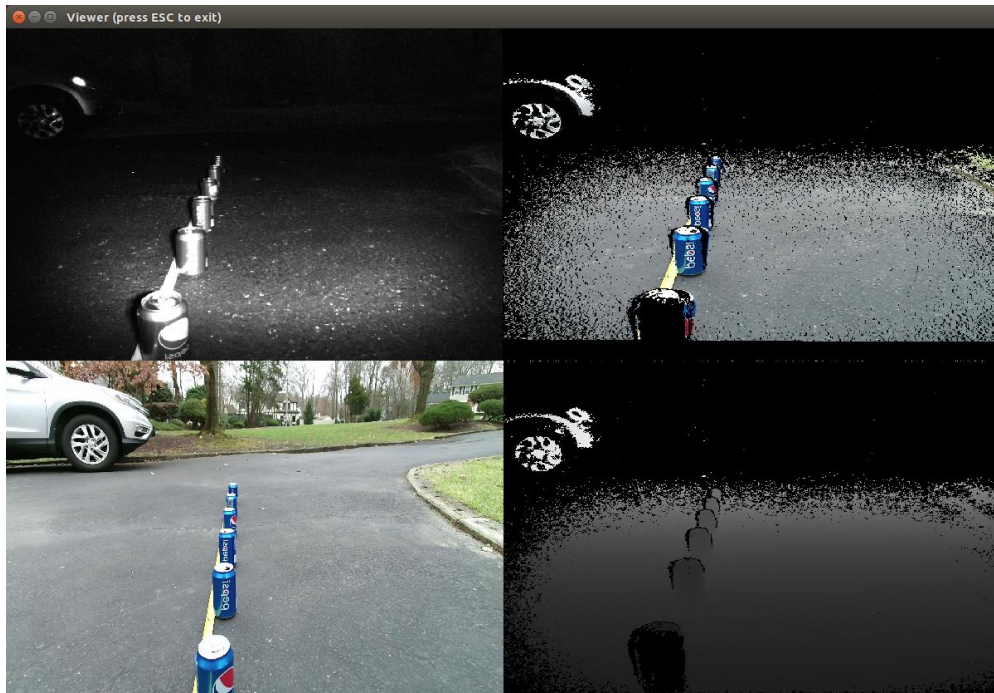


Figure 12: Protonect Test Software Confirms Kinect Configuration

I then ran RViz and added the Kinect Point Cloud and RGB ROSTopics:

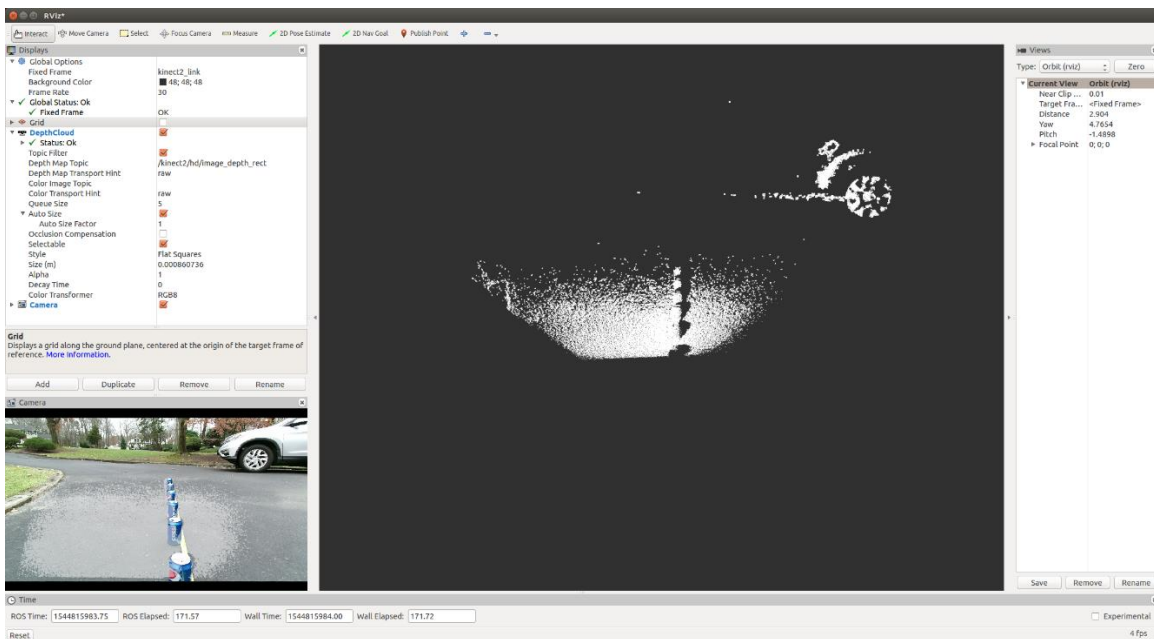


Figure 13: RViz Showing Cans vis Point Cloud ROSTopic

If we map colors to distance on the point cloud, we can get a better idea of what the Kinect sees:

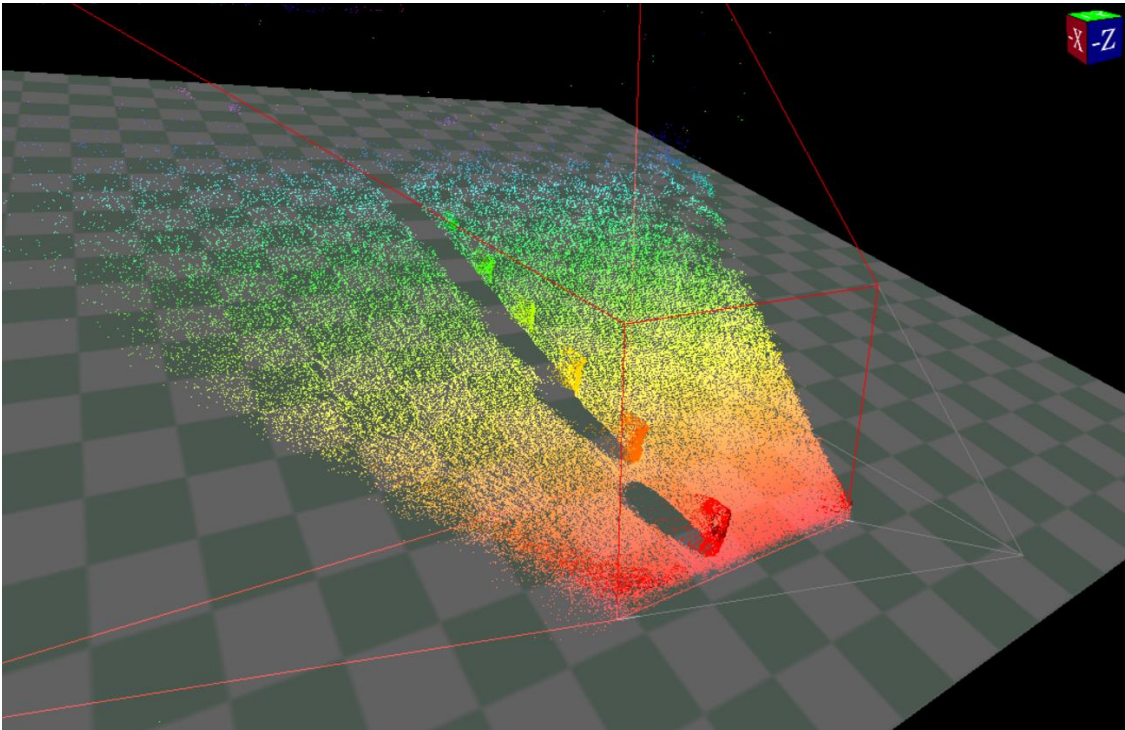


Figure 14: 3D Isometric View of Color Mapped Point Cloud

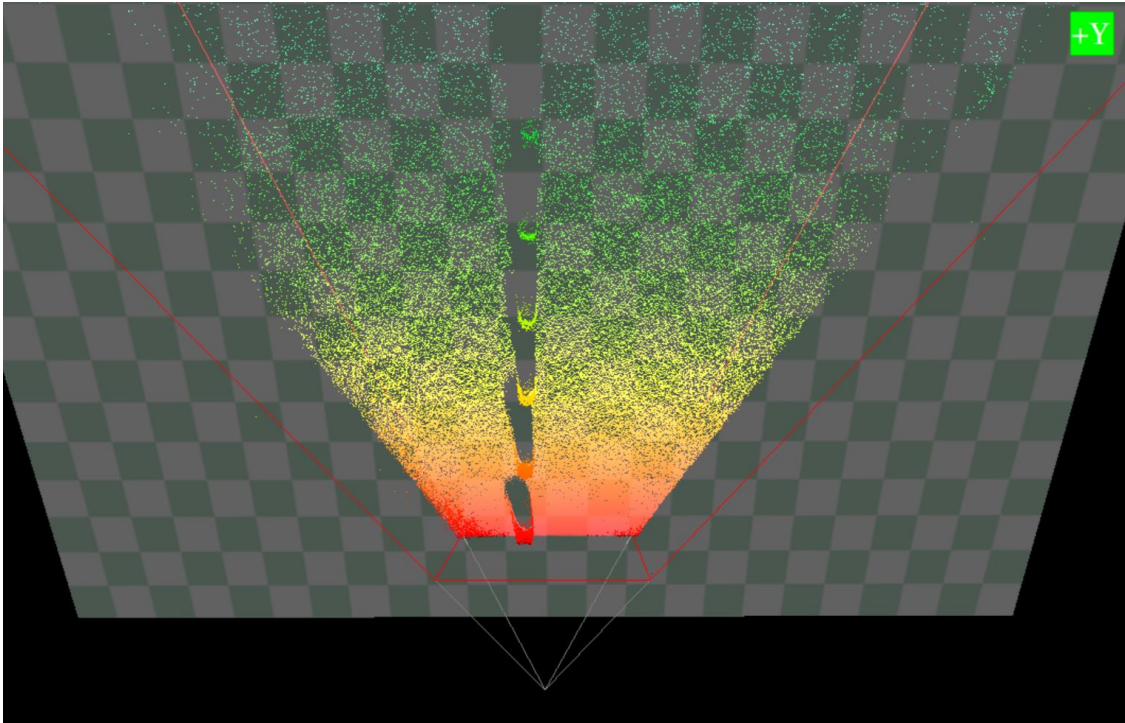


Figure 15: 3D Top View of Color Mapped Point Cloud

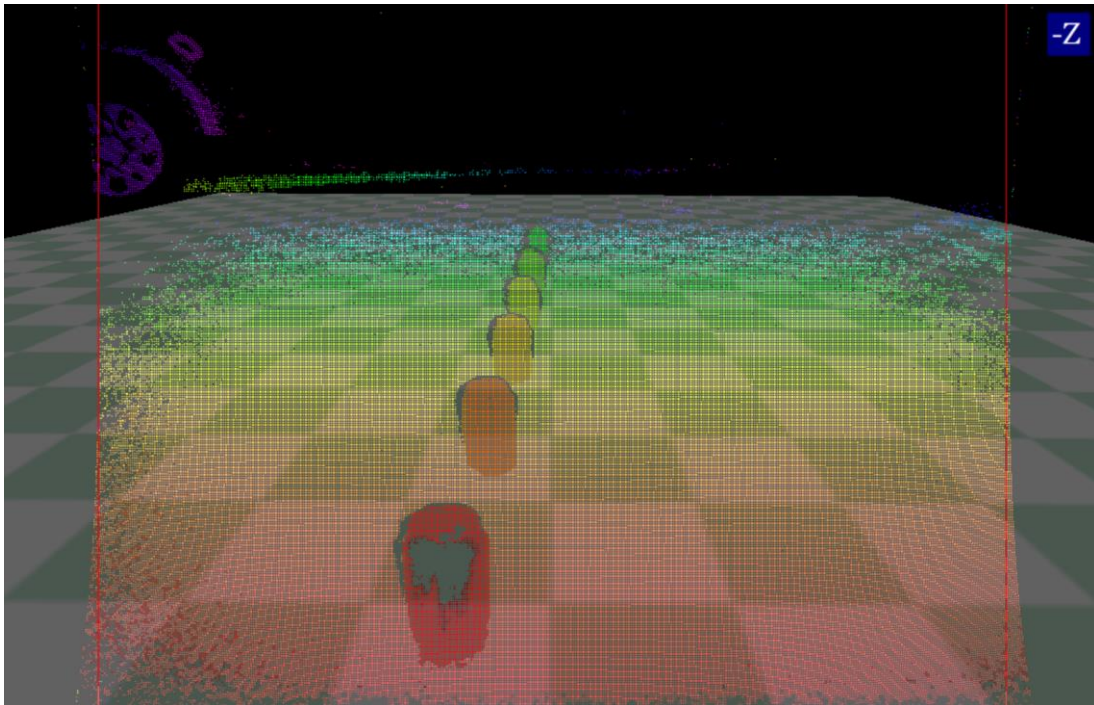


Figure 16: 3D Front View of Color Mapped Point Cloud

We can switch to 2D as well in both Color Depth and Grayscale:

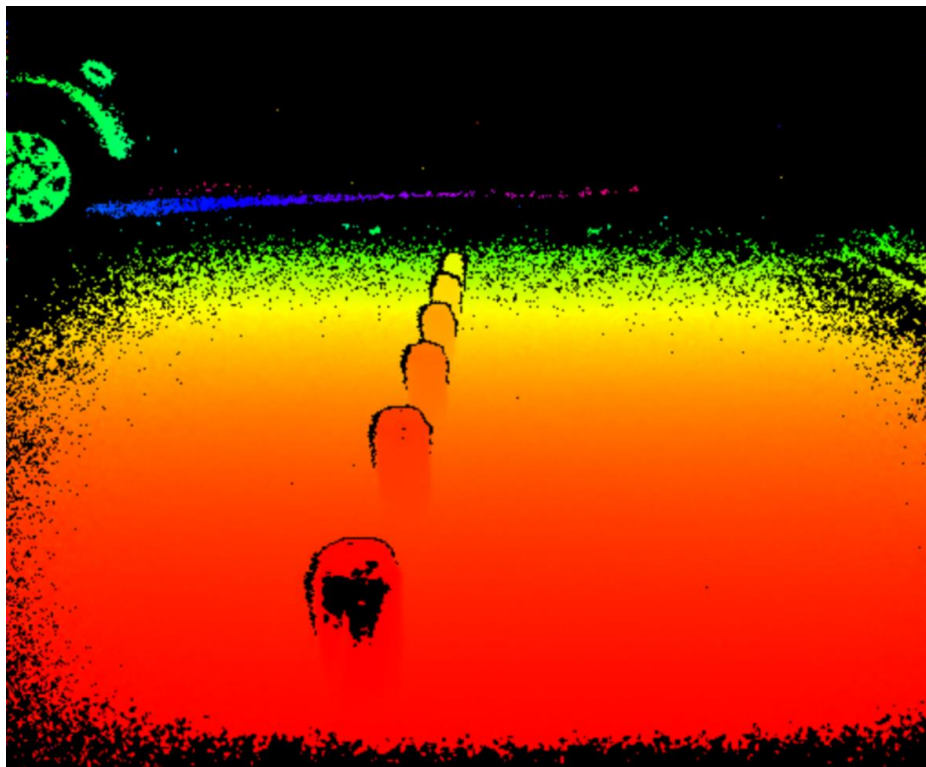


Figure 17: 2D Front View of Color Mapped Point Cloud

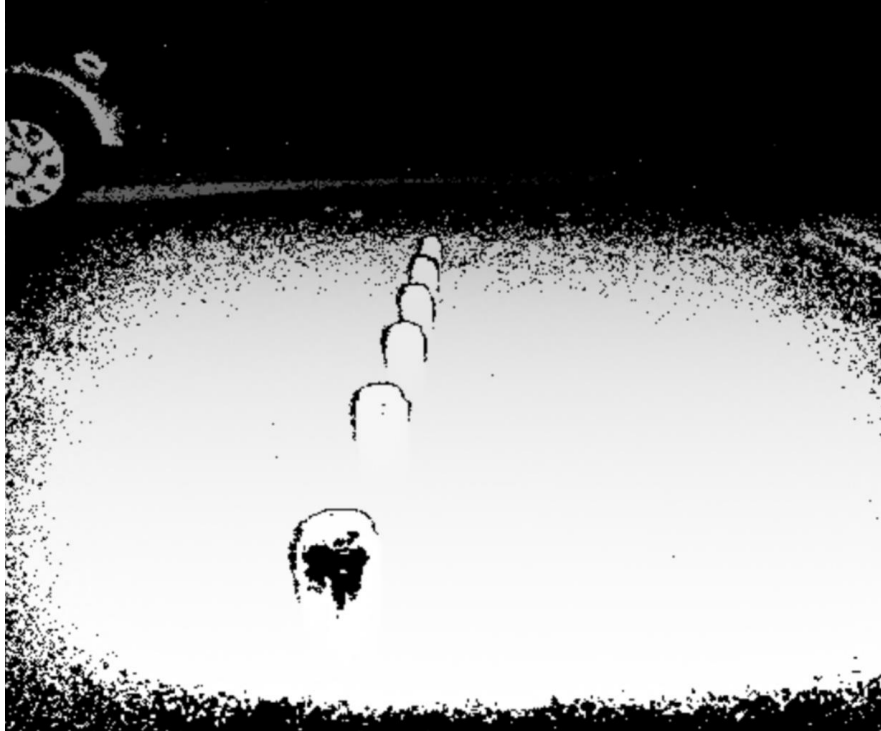


Figure 18: 2D Front View of Grayscale Point Cloud

Below we have the Infrared Sensor Data without Point Clouds:



Figure 19: 2D Front View of Infrared Raw Image

Here are some additional photos I took using Protonect's viewer where we can see multiple views simultaneously:

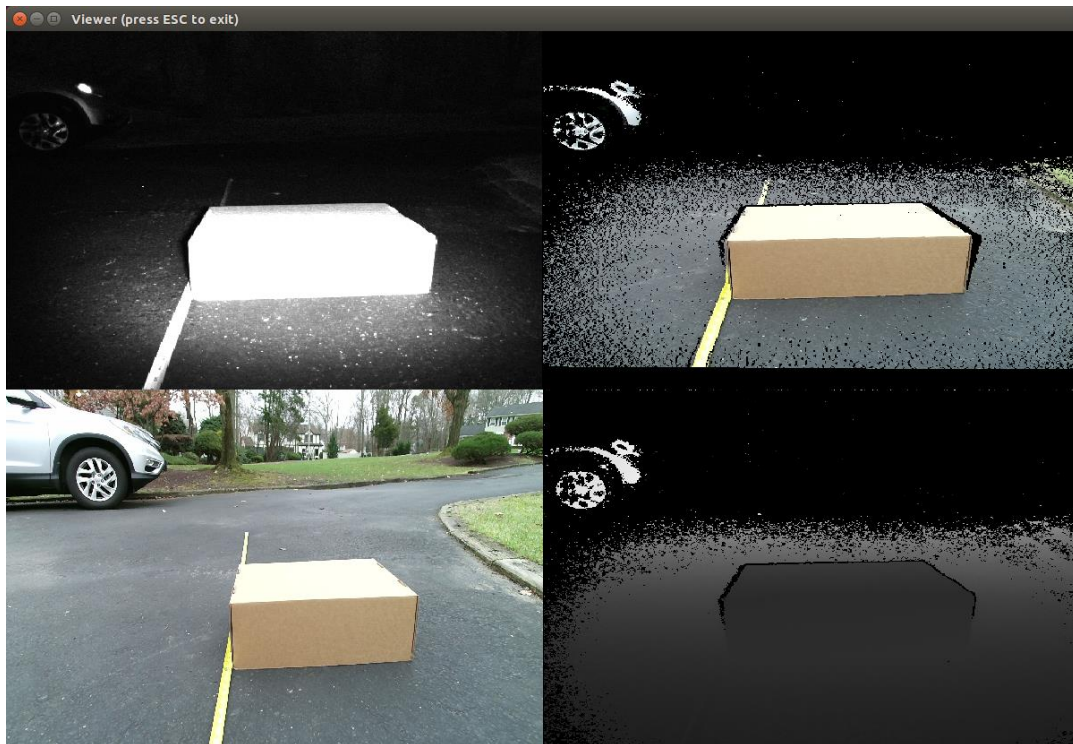


Figure 20: Protonect Viewer – Box 2 Feet Away from Kinect

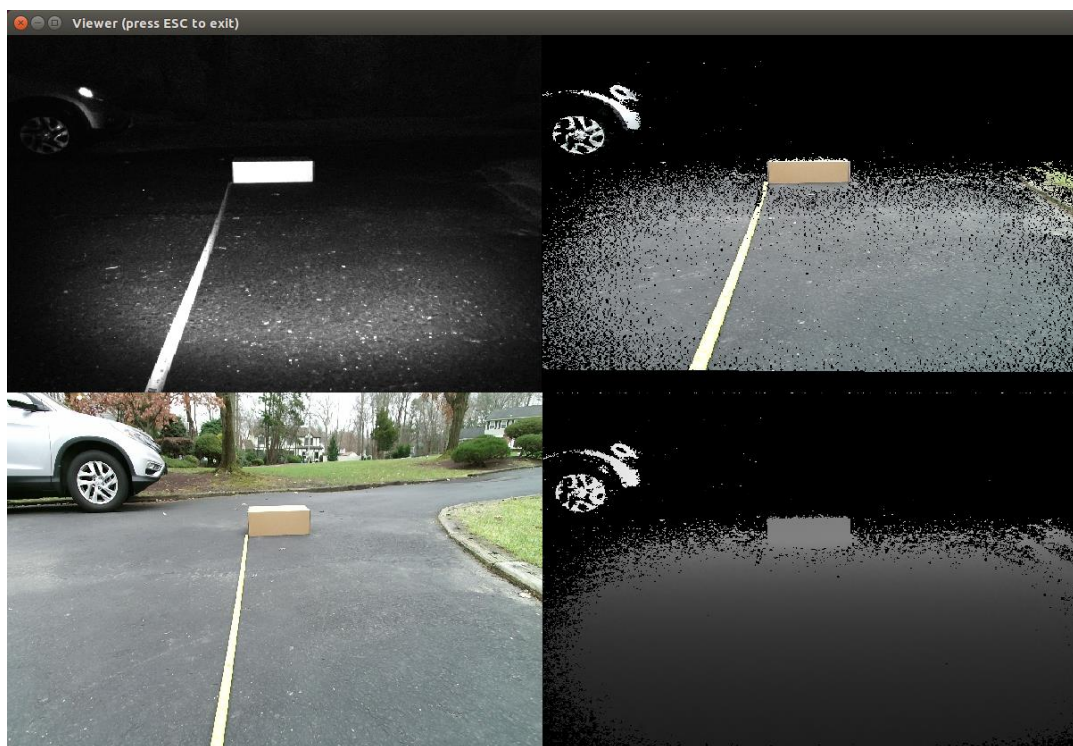


Figure 21: Protonect Viewer – Box 7 Feet Away from Kinect



## **Chapter 5: Conclusion**

After following the recommendations of the University of La Laguna students, we were able to achieve similar results. By placing the Xbox Kinect camera low to the ground, the Kinect is able to distinguish the difference between its own infrared signals and external light radiation interference.

For future research, if we conduct a larger calibration set, we can have higher accuracy similar to those of the Kinect2\_calibration repository author. Once fixed to a golf cart, or other autonomous vehicle, and ground clearances are known, obstacle height can become a more useful metric. In the Verdino project, they took the normal lines of the surfaces and determined if they were navigable by the golf cart. This was useful in mobile path planning when attempting to navigate on slopes. These features that are normally considered obstacles, combined with their software implementation, can be seen as navigable surfaces to the robot.

From this research, we can see that the Xbox Kinect V2 Sensor is a very capable and cost-effective solution that can be purposed for computer vision applications. Although with its increasing age, discontinuation of parts, and diminishing software support online, it may not be the best solution to implement compared to others offered on the market. There has been an increasing emergence of RealSense cameras being used in ROS for robotics applications. RealSense technologies have similar hardware such as depth sensors, RGB sensors, and infrared projectors. They are capable of performing depth, position, and orientation tracking as well.

With the emergence of these cost-effective hardware components we are seeing engineers repurpose these sensors into new applications. Throughout this research project, I have seen lightweight manufacturing robots, conveyor belt applications, 3D scanning, and many other unique uses of these sensors. With software such as Robot Operating System, we are creating smarter robots that utilize sensor fusion to more accurately sense and manipulate its environment. With further development, these different applications will improve quality of life in so many different industries.

## Reference URLs:

[https://github.com/code-iai/iai\\_kinect2](https://github.com/code-iai/iai_kinect2)

Tools for using Kinect One (Kinect V2) in ROS

[https://github.com/code-iai/iai\\_kinect2/tree/master/kinect2\\_calibration](https://github.com/code-iai/iai_kinect2/tree/master/kinect2_calibration)

OpenCV tool for calibrating two cameras to each other

[https://github.com/code-iai/iai\\_kinect2/issues/181](https://github.com/code-iai/iai_kinect2/issues/181)

Kinect V2 calibration – No distance sample data for depth calibration fix

[https://github.com/code-iai/iai\\_kinect2/issues/9](https://github.com/code-iai/iai_kinect2/issues/9)

Fix for dependencies with libfreenect2

[https://github.com/code-iai/iai\\_kinect2/issues/332](https://github.com/code-iai/iai_kinect2/issues/332)

OpenCL registration fix for kinect2\_bridge

Instructions for building make files for Kinect2\_bridge and modifying ~/.bashrc

[https://github.com/code-iai/iai\\_kinect2/issues/217](https://github.com/code-iai/iai_kinect2/issues/217)

Compiling workspace to fix roslaunch kinect2\_bridge launch files

[https://github.com/code-iai/iai\\_kinect2/issues/153](https://github.com/code-iai/iai_kinect2/issues/153)

How to enable publishing for other applications to use kinect2\_bridge data

[https://github.com/code-iai/iai\\_kinect2/issues/360](https://github.com/code-iai/iai_kinect2/issues/360)

How to test publishing is working using rostopics

<https://github.com/OpenKinect/libfreenect2>

OpenKinect / libfreenect2 – Open source drivers for Kinect V2

<https://github.com/OpenKinect/libfreenect2/wiki/Troubleshooting>

OpenKinect / libfreenect2 – Troubleshooting Wiki

[https://github.com/OpenPTrack/open\\_ptrack/issues/19](https://github.com/OpenPTrack/open_ptrack/issues/19)

Explanation for modifying udev rules workaround for Kinect detection by modifying USB Vendor/Product IDs.

<https://groups.google.com/forum/#!topic/openkinect/qdCWdXGxIog>

Using dmesg to find Vendor / Product ID necessary for augmenting udev rules

<https://answers.ros.org/question/210242/useing-rviz-from-kinect-v2-xbox-one/>

Kinect2\_bridge ROS Topics for RVIZ

[https://www.choitek.com/uploads/5/0/8/4/50842795/ros\\_kinect.pdf](https://www.choitek.com/uploads/5/0/8/4/50842795/ros_kinect.pdf)

Tutorial for vision, mapping, and localization with the Kinect V1 in ROS

<https://github.com/OpenKinect/libfreenect2#debianubuntu-1404>

Open source drivers for the Kinect for Windows V2

<http://wiki.ros.org/indigo/Installation/Ubuntu>

ROS Indigo Installation for Ubuntu 14.04.1

[http://wiki.wpi.edu/robotics/ROS\\_Commands](http://wiki.wpi.edu/robotics/ROS_Commands)

Common ROS Commands

[http://wiki.wpi.edu/robotics/ROS\\_File\\_Types](http://wiki.wpi.edu/robotics/ROS_File_Types)

ROS File Types

[http://wiki.wpi.edu/robotics/ROS\\_Terms](http://wiki.wpi.edu/robotics/ROS_Terms)

Explanation of common ROS Terms

<http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>

Tutorial for configuring a ROS Environment

<https://answers.ros.org/question/308566/apt-get-to-install-ros-package-with-specific-version/>

Installing older ROS packages with specific versions

<https://answers.ros.org/question/240235/how-to-install-packages-from-github/>

Installing github packages for use in ROS

[http://wiki.ros.org/catkin#Installing\\_catkin](http://wiki.ros.org/catkin#Installing_catkin)

Installing Catkin within Ubuntu

[https://subscription.packtpub.com/book/hardware\\_and\\_creative/9781782175193/1/ch01lv1sec1/creating-a-catkin-workspace](https://subscription.packtpub.com/book/hardware_and_creative/9781782175193/1/ch01lv1sec1/creating-a-catkin-workspace)

Instructions for creating Catkin workspaces

[http://wiki.ros.org/catkin/Tutorials/create\\_a\\_workspace](http://wiki.ros.org/catkin/Tutorials/create_a_workspace)

Additional instructions for creating Catkin workspaces

[http://wiki.ros.org/catkin/Tutorials/using\\_a\\_workspace](http://wiki.ros.org/catkin/Tutorials/using_a_workspace)

Instructions for using a Catkin workspace

[http://wiki.ros.org/catkin/commands/catkin\\_make](http://wiki.ros.org/catkin/commands/catkin_make)

Catkin make command instructions

<http://wiki.ros.org/ROS/Tutorials/CreatingPackage>

Creating packages within Catkin

<http://wiki.ros.org/catkin/workspaces>

Building packages within Catkin

<https://stackoverflow.com/questions/48434373/getting-kinect-v2-to-work-with-ubuntu-16-04-and-ros-kinetic>

Explanation for Catkin / Libfreenect2 integration

[http://wiki.ros.org/catkin\\_or\\_rosbuild](http://wiki.ros.org/catkin_or_rosbuild)

Comparison for Catkin versus rosbuild (Limitations Explanation)

[http://wiki.ros.org/catkin/migrating\\_from\\_rosbuild](http://wiki.ros.org/catkin/migrating_from_rosbuild)

Explanation for migrating away from rosbuild to catkin

<https://cgold.readthedocs.io/en/latest/first-step/installation.html>

CMake instructions / tips

<https://github.com/ethz-asl/kinect2-ros>

Tool for using Kinect One V2 in ROS

[http://wiki.ros.org/depth\\_image\\_proc](http://wiki.ros.org/depth_image_proc)

ROS built-in tool for processing Depth Images (references OpenNI)

[Not recommended - use only for example / reference]

<https://answers.ros.org/question/210242/useing-rviz-from-kinect-v2-xbox-one/>

Recommendation for calibrating camera before using within RViz

<https://answers.ros.org/question/242286/how-to-find-the-urdf-of-kinect-v2/>

URDF Explanation for RViz / Gazebo Applications

[https://github.com/code-](https://github.com/code-iai/iai_robots/blob/master/iai_kinect2_description/urdf/kinect2.urdf.xacro)

[iai/iai\\_robots/blob/master/iai\\_kinect2\\_description/urdf/kinect2.urdf.xacro](https://github.com/code-iai/iai_robots/blob/master/iai_kinect2_description/urdf/kinect2.urdf.xacro)

URDF Location for RViz / Gazebo Applications

[http://wiki.ros.org/rtabmap\\_ros/Tutorials/HandHeldMapping](http://wiki.ros.org/rtabmap_ros/Tutorials/HandHeldMapping)

RTABMAP – Mapping and Localization using OpenNI

[Not recommended - use only for example / reference]

[http://wiki.ros.org/openni\\_launch](http://wiki.ros.org/openni_launch)

OpenNI Wiki – Used for converting depth/RGB/IR to point clouds and depth/disparity images.

<https://answers.ros.org/question/235440/test-kinectno-devices-connected-waiting-for-devices-to-be-connected/>

Explaining why OpenNI is not recommended and suggesting freenect\_stack (which is no longer recommended either)

[http://wiki.ros.org/rgbd\\_launch](http://wiki.ros.org/rgbd_launch)

RGBD\_Launch – Launch files for RGB-D devices such as the Kinect in ROS

<https://answers.ros.org/question/143496/roslaunch-is-neither-a-launch-file-in-package-nor-is-a-launch-file-name/>

Explanation for roslaunch launch file packages / names.  
Reference for source setup / bash files.

[http://wiki.ros.org/freenect\\_launch](http://wiki.ros.org/freenect_launch)

Tutorial for freenect\_launch [Not recommended - use only for example / reference]

[https://github.com/ros-drivers/freenect\\_stack/issues](https://github.com/ros-drivers/freenect_stack/issues)

Freenect\_stack Issues [Not recommended - use only for example / reference]

<https://askubuntu.com/questions/886588/kinect-2-on-ubuntu-16-04-device-not-listed-in-lsusb>

Explanation for installing dependencies, navigating lsusb, building libfreenect2

<https://answers.ros.org/question/207047/kinect-v2-no-devices-connected-ive-tried-everything/>

Recommendation to update from 14.04 to 14.04.1 for USB 3 recognition in Linux kernel to fix Kinect2 USB controller problems

<https://www.youtube.com/watch?v=cjC5tXpVXzE>

Tutorial for Kinect V1 in ROS

<https://www.youtube.com/watch?v=YZwlt2msvpl>

Tutorial for Kinect V2 in ROS including RViz and rtabmap [Dead Source URL]

<https://www.youtube.com/watch?v=URhu-fAUWWQ>

Tutorial for installing libfreenect2 on Nvidia Jetson TK1

<https://www.youtube.com/watch?v=MToCOoCJVGs>

ROS (14.04) RViz Pose Example using Kinect V2 and rtabmap

<https://www.youtube.com/watch?v=zUeZnfQoBOo>

KinectV2 Demo on Robot Indoors

[https://www.youtube.com/watch?v=\\_qiLAWp7AqQ](https://www.youtube.com/watch?v=_qiLAWp7AqQ)

Kinect Navigation / Mapping / Planning example with loop closures.

[https://www.researchgate.net/publication/283326333\\_An\\_Indoor\\_Obstacle\\_Detection\\_System\\_Using\\_Depth\\_Information\\_and\\_Region\\_Growth](https://www.researchgate.net/publication/283326333_An_Indoor_Obstacle_Detection_System_Using_Depth_Information_and_Region_Growth)

Indoor Obstacle Detection System Using Depth Information and Region Growth

<https://angel.co/projects/182175-verdino>

Verdino Project Page by Nestor Morales Hernandez [Author]

[https://www.reddit.com/r/linux4noobs/comments/7vz150/remote\\_desktop\\_to\\_ubuntu/](https://www.reddit.com/r/linux4noobs/comments/7vz150/remote_desktop_to_ubuntu/)

Quality of Life optimization suggestion for Remote Desktop on Ubuntu

<https://symless.com/synergy>

Quality of Life – share keyboard / mouse across bare metal machines

<https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/learn-more/use-local-resources-on-hyper-v-virtual-machine-with-vmconnect>

Limitations of local resources on Ubuntu virtual machines

[Not recommended - use only for example / reference]

<https://xpra.org/>

Remote Desktop application for Ubuntu [Not recommended - use only for example / reference]

<https://www.howtoforge.com/how-to-install-x2goserver-on-ubuntu-14.04-as-vnc-alternative>

Remote Desktop application for Ubuntu [Not recommended - use only for example / reference]

# Appendix A

## calib\_color.yaml

```
%YAML:1.0
cameraMatrix: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 1.1617963678371680e+03, 0., 9.5093383475668224e+02, 0.,
         1.2110595550377591e+03, 5.2654387243856411e+02, 0., 0., 1. ]
distortionCoefficients: !!opencv-matrix
  rows: 1
  cols: 5
  dt: d
  data: [ 5.4341306125840788e-02, -5.2326511693269059e-02,
         -1.4156323021479405e-02, -4.5905877433509764e-03,
         1.3436815684838805e-02 ]
rotation: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 1., 0., 0., 0., 1., 0., 0., 0., 1. ]
projection: !!opencv-matrix
  rows: 4
  cols: 4
  dt: d
  data: [ 1.1617963678371680e+03, 0., 9.5093383475668224e+02, 0., 0.,
         1.2110595550377591e+03, 5.2654387243856411e+02, 0., 0., 0., 1.,
         0., 0., 0., 0., 1. ]
```

# Appendix B

## calib\_depth.yaml

```
%YAML:1.0  
depthShift: -1.4106589553147419e+01
```



# Appendix C

## calib\_ir.yaml

```
%YAML:1.0
cameraMatrix: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 3.8952059999562010e+02, 0., 2.5673175566817639e+02, 0.,
         4.0711960284607386e+02, 2.1195820796180755e+02, 0., 0., 1. ]
distortionCoefficients: !!opencv-matrix
  rows: 1
  cols: 5
  dt: d
  data: [ 1.2745334821987084e-01, -3.3936384531433733e-01,
         -5.5842671244626932e-03, -2.3722223949202751e-03,
         1.0172808241454298e-01 ]
rotation: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 1., 0., 0., 0., 1., 0., 0., 0., 1. ]
projection: !!opencv-matrix
  rows: 4
  cols: 4
  dt: d
  data: [ 3.8952059999562010e+02, 0., 2.5673175566817639e+02, 0., 0.,
         4.0711960284607386e+02, 2.1195820796180755e+02, 0., 0., 0., 1.,
         0., 0., 0., 0., 1. ]
```

# Appendix D

## calib\_pose.yaml

```
%YAML:1.0
rotation: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 9.9995022488478558e-01, -8.2249935015495275e-03,
          5.6477636960330958e-03, 8.1240660106269812e-03,
          9.9981092762976043e-01, 1.7666594000347129e-02,
          -5.7920034808118664e-03, -1.7619831838516332e-02,
          9.9982798231578851e-01 ]
translation: !!opencv-matrix
  rows: 3
  cols: 1
  dt: d
  data: [ -6.4840180891407320e-02, -7.2503333571428330e-03,
          3.2835706161128547e-02 ]
essential: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ -2.2476548831694171e-04, -3.2701748181810132e-02,
          -7.8291812610523301e-03, 3.2458517206651723e-02,
          -1.4125465534796490e-03, 6.5014475542836958e-02,
          6.7232065612618510e-03, -6.4887555349465983e-02,
          -1.1045569811989500e-03 ]
fundamental: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ -3.7834409191198661e-07, -5.2666773679931942e-05,
          6.1268997240825317e-03, 5.2414387906345340e-05,
          -2.1823933439006541e-06, 2.7900396704276943e-02,
          -1.4090570396367210e-02, -7.0179015355884777e-02, 1. ]
```