Mobile and Intelligent Robotics

Kinect Calibration using JACO
Final Project Report

February 2018

The group:

- **Group 8**

  Marcos Pires, no. 73162
  Nuno Miguel Silva, no. 72708

- **Our repository:**

  `http://code.ua.pt//projects/rmi1718-g08/repository`

Table of Contents

1 Introduction

This project consists in calibrating a Kinect camera, making use of a robotic arm, more specifically Kinova's JACO.

The calibration procedure has the objective to find the Kinect camera's position comparatively to the JACO's base link.

This calibration purpose is for maintaining the quality of measurements as well as to ensure the proper working of the Kinetic Camera.

In this report it will be succinctly explained all the steps made to achieve this goal.

2 Preparation and rosbag

In order to begin our work, a rosbag was recorded with the Kinect and the JACO. The rosbag is a file containing important data about what the sensors acquired in the time interval in which the bag was recorded. The rosbag can be used to simulate the JACO-Kinect environment anywhere.
The rosbag used in our simulations had two important topics:

- `/tf`
  - The transform frames used to know the JACO's pose.
  - This included the position and orientation of the arm's joints.

- `camera/depth_registered/points`
  - The point-cloud data including the points in the point-cloud and the RGB image captured by the Kinect.
  - Each pixel in the video sequence captured is a point in the cloud and it has the same color as the pixel.
  - The depth camera gives to this pixel a distance value to the camera.

It was added to this environment a yellow paper to estimate the table's position and it was also placed colored bands on the JACO arm so that the joints could be easily identified. The rviz program is a ROS tool used to visualize data captured from sensors. This application will be used in this project to visualize the pointcloud and the tf.

The pointcloud in **Fig.1** appears like this in the beginning because the camera position by default is in the origin with no rotation.

3 Video Processing

In an early approach it was extracted a video from the pointcloud color information to work with the obtained frames. Using OpenCV it was created an algorithm for image processing that consisted in some steps to extract the colored bands and the paper from the extracted video.

The HSV representation resembles a cylindrical geometry. Because if this, the Hue value is easily filtered applying an interval for the cylinder angle.

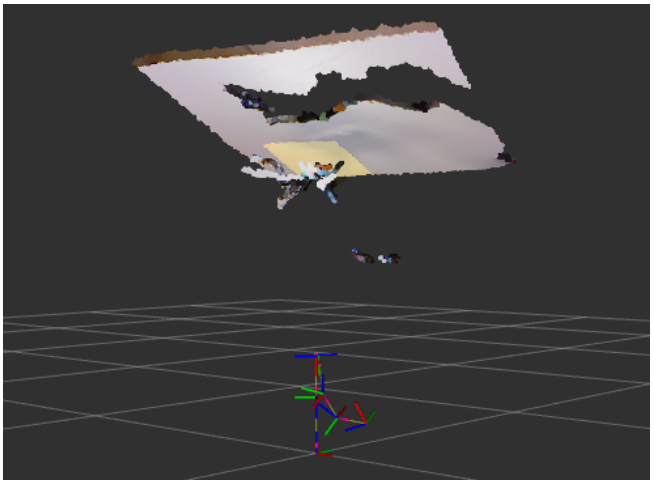


Fig. 1. First rviz visualization of the pointcloud and the tf

First several HSV filters were applied, five in total: four for the different colored bands and one for the colored paper (different values with different calibrations for the filter were used to obtain the proper results). This gave us five binary images with the location of one object in the frame.

After obtaining the binary images it was noticed that there was a considerable amount of noise. To fix this a median filter was applied to smooth the edges of the binary *blobs*, then a filter that computes the connectivity of the surrounding pixels (8-way connectivity) was applied where pixels that had at least one neighbour with a different color were ignored. After this the biggest *blob* is extracted and identified as the desird object. Lastly a final median filter is applied to again smooth the edges.

In **Fig.2** is presented an image where the obtained binary images are merged with the original frame, leaving only visible the colored bands and paper.

Fig. 2. Extraction of colored bands and paper from extracted video frame

4 Accessing the Pointcloud

The next step is to access the data contained in the pointcloud. To obtain the pointcloud data, a ROS subscriber was added to access data in the pointcloud topic `/camera/depth_registered/points`.

The pointcloud messages are of type *PointCloud2*. The data structure of this messages contain fields in which the most important and used for our work were the **header** with time of sensor data acquisition, and the coordinate frame ID, **width** and **height** of the 2D structure of the point cloud, **point_step** and **row_step** describe the size of each point and each row of point in the **data** array containing information about the position of the point and their respective color.

The **data** array, in the end, was the most used because it is where the most useful information is stored. Each point in the pointcloud is a 32-byte piece of data. The first 12 bytes describe the XYZ position of the point in segments of 4 bytes being each one a float variable. The color information of the point was stored starting at byte 16 with a 4-byte long segment of data in the BGRA format. All other position in the **data** array are set to zero.

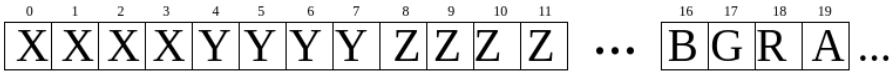


Fig. 3. Information structure in the data array

From time to time, it is possible to find the sequence `{0,0,192,127}` in the data array. Converting this a float variable gives a NaN. This happens because some objects were too close to the depth camera and it could not get the distance. In the pointcloud this points are not shown.

Fig. 4. Points missing in the pointcloud

In **Fig. 4**, to the right, it can be seen that some points respectively in the JACO are missing due to the robotic arm being too close to the depth camera.

5 Determining the rotation angles

The next step is to find the rotation angles to set the pointcloud straight. To achieve this, it was used the yellow paper in the table. Using the yellow paper it is possible to find a plane similar to the table. The goal is to apply a rotation that sets the points of this plane to the same height.

5.1 Filtering the yellow paper

The first step for this procedure is to filter the yellow paper in the pointcloud. Having a subset of the pointcloud with only the paper enables the possibility to determine 3 points of the paper, and thus setting a plane. Using the procedures of the Video Processing section, the RGB values of each point were converted into HSV format so that the colors are easier to filter. Setting intervals in the HSV filter it is possible to acquire a pointcloud similar to the **Fig. 5**.

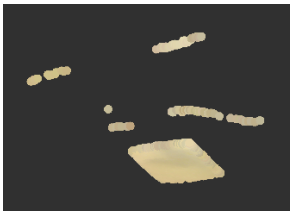


Fig. 5. Paper in pointcloud with noise

Some noise can be seen in the pointcloud, because not all yellow points in the pointcloud belong to the paper. Although, this points can be filtered using a simple rule: if the yellow point's neighbours are not yellow, this point is excluded from the pointcloud. The results can be seen in **Fig. 6**

Fig. 6. Paper in pointcloud without noise

5.2 Finding the paper points

Having the pointcloud of the yellow paper, is it possible to find the edges of the paper and so determine an estimate plane of the paper and the table. Assuming that the paper is placed on top of a flat and leveled table, to establish a plane it is only needed 3 points $p1, p2$, and $p3$. To find this points, the pointcloud is iterated: $p1$ is the point with lowest x value, $p2$ is the point with highest y value and $p3$ is the point with highest x value. This process is repeated 20 times for 20 reading and then a mean value is calculated.

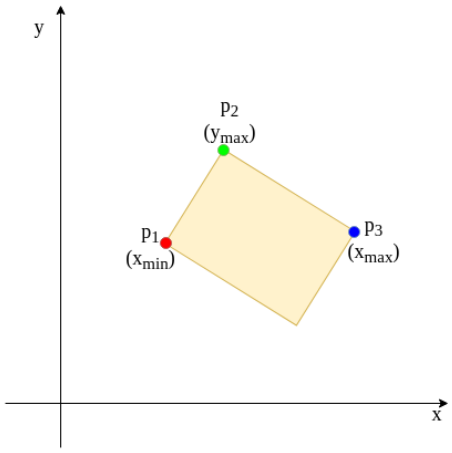


Fig. 7. The three points in the paper plane

However after implementing this algorithm, the results obtained were a bit different. Because of the noise in the pointcloud, it was obtained 3 points, one on each edge of the paper. This points can also be worked on since they are 3 points in the paper plane. This 3 points are published so they can be visualized in rviz in the topic `/output/filtered_cloud/paperpoints`.

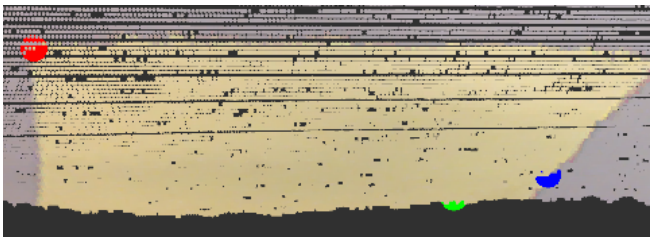


Fig. 8. Rviz visualization of the acquired points

5.3 Calculate the rotation transform

Finally, the angle of rotation can be calculated using the previous 3 points. Three rotation angles are to be found: **yaw**, **pitch** and **roll**. The **yaw** refer to the vertical axis, the **pitch** to the lateral axis, and the **roll** to the longitudinal axis.

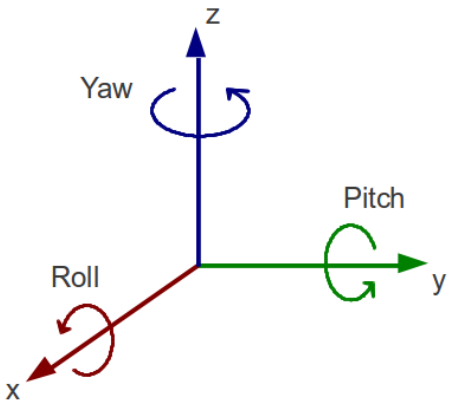


Fig. 9. Rotation axis

To calculate the **yaw** it was applied the following equation:

$$d = \sqrt{(p3_x - p1_x)^2 + (p3_y - p1_y)^2}$$
$$\text{x} = \text{abs}((\text{p1}_z - p3_z)^2)$$
$$\text{yaw} = \sin^{-1}(\tfrac{x}{d})$$

Where d is the distance and x is the height difference between p_1 and p_3. To calculate the **roll** the following equation is applied:

$$d = \sqrt{(p2_x - p1_x)^2 + (p2_y - p1_y)^2}$$
$$\text{x} = \text{abs}((\text{p1}_z - p2_z)^2)$$
$$\text{roll} = \sin^{-1}(\tfrac{x}{d})$$

Where d is the distance and x is the height difference between p_1 and p_2. After applying this two rotations, we already have the paper plane with all points

practically at the same height. The next step is to make sure that this plane is not upside-down.

5.4 Filtering the green joint

To detect if the pointcloud is upside-down after the previous rotations, it is used the green joint (later used for translation) of the robotic arm. The first step to find this joint in the pointcloud is to filter the points and create a subset of the pointcloud with only the green joint.
To make this happen, the procedure applied is the same as in the yellow paper but this time for the green HSV values of the joint. After filtering the points with the HSV interval, the noise is reduced using the same rule as before: if the green point's neighbours are not green, this point is excluded from the pointcloud.

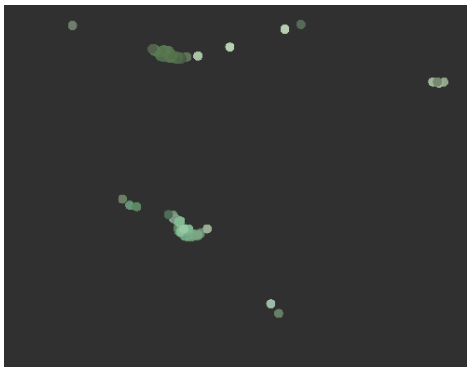


Fig. 10. Unfiltered green points in the pointcloud

Having found the green points, only one of this points is going to be used and it is the one nearest to the camera (and possibly one of the farthest from the paper). Knowing the position of each point, is it possible to calculate the distance to the camera from them. The chosen point will be the one with smaller calculated distance. This point is published so it can be visualized in rviz in the topic `/output/filtered_cloud/greenpoints`.
The **pitch** is now calculated using a simple algorithm. If the green point in below the yellow point, then the pointcloud is upside-down and the **pitch** is π (180°), otherwise is 0. **Fig. 11** depicts the result from the rotation.

6 Translation

With the correct rotation of the Kinect sensor what is now missing is the correct positioning of the Kinect, relative to the arm. For this it was used the green joint point as a reference.

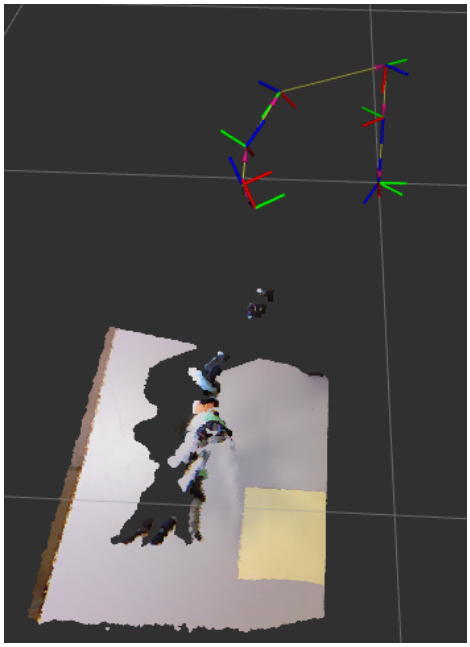


Fig. 11. Original pointcloud result after applying rotation

At this point only the **yaw**, **pitch** and **roll** values were obtained but they are not yet used. To apply translation it is needed, firstly, to apply rotation. The green point is selected to be rotated. The updated green point is obtained with the following equations:

$$
\begin{aligned}
g'_y &= g_y \cdot cos(yaw) - g_x \cdot sin(yaw) \\
g'_x &= g_x \cdot cos(yaw) + g_y \cdot sin(yaw) \\
g''_x &= g'_x \cdot cos(pitch) - g_z \cdot sin(pitch) \\
g'_z &= g_z \cdot cos(pitch) + g'_x \cdot sin(pitch) \\
g''_y &= g'_y \cdot cos(roll) - g'_z \cdot sin(roll) \\
g''_z &= g'_z \cdot cos(roll) + g'_y \cdot sin(roll)
\end{aligned}
$$

Where g''_x , g''_y and g''_z are the new green point coordinates.

The next step is to obtain the **tf** values. For this procedure, a transform is created between the base link of the robot arm and the joint with the green color (link 6). This transform is created using the method `lookupTransform` that receives the base link and joint link frames and outputs a transform from the base to the joint. With this transform, the exact coordinates in space where the green joint is can be obtain. The next step is to obtain the translation vector. This vector is obtained simply with the following equation:

$$(t_x, t_y, t_z) = (tf_x - g''_x, tf_y - g''_y, tf_z - g''_z)$$

Where (t_x, t_y, t_z) is the translation vector, (tf_x, tf_y, tf_z) are the coordinates of the transform between the base and the joint, and (g''_x, g''_y, g''_z) are the actual green points. This calculation is done 20 times for 20 readings and the final values used are the average of all results.

7 Applying the transform

The final step of the calibration procedure is to write the values to a file. An example of the file goes as following:

```
   x         y         z        yaw       pitch     roll
0.378954 0.168794 0.839875 0.00363137 3.14159 0.542009
```

Fig. 12. File example with final values

Opening a terminal, it can be used the `static_transform_publisher` to test this values.

```
mikael@kerfeldt:tpfinal$ rosrun tf static_transform_publisher 0.378954 0.168794 0.839875
0.00363137 3.14159 0.542009 m1n6s200_link_base camera_rgb_optical_frame 100
```

Fig. 13. Terminal command to test the transform

The command `rosrun` runs ROS executables. The tf package of ROS includes this program that publishes our transform to the **/tf** topic. The arguments for this program are the values acquired from the calibration, the JACO's base link frame, the camera's frame and the refresh rate.
After running this command, the Rviz will show the following:

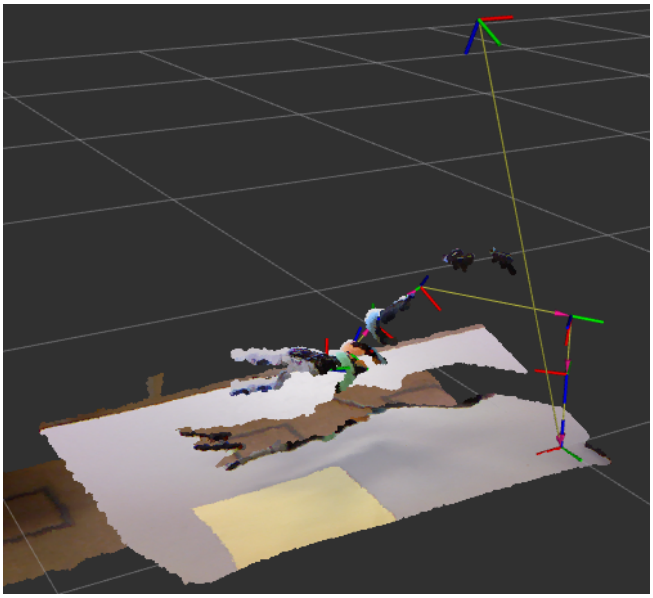


Fig. 14. Final result with JACO transforms overlapping the pointcloud

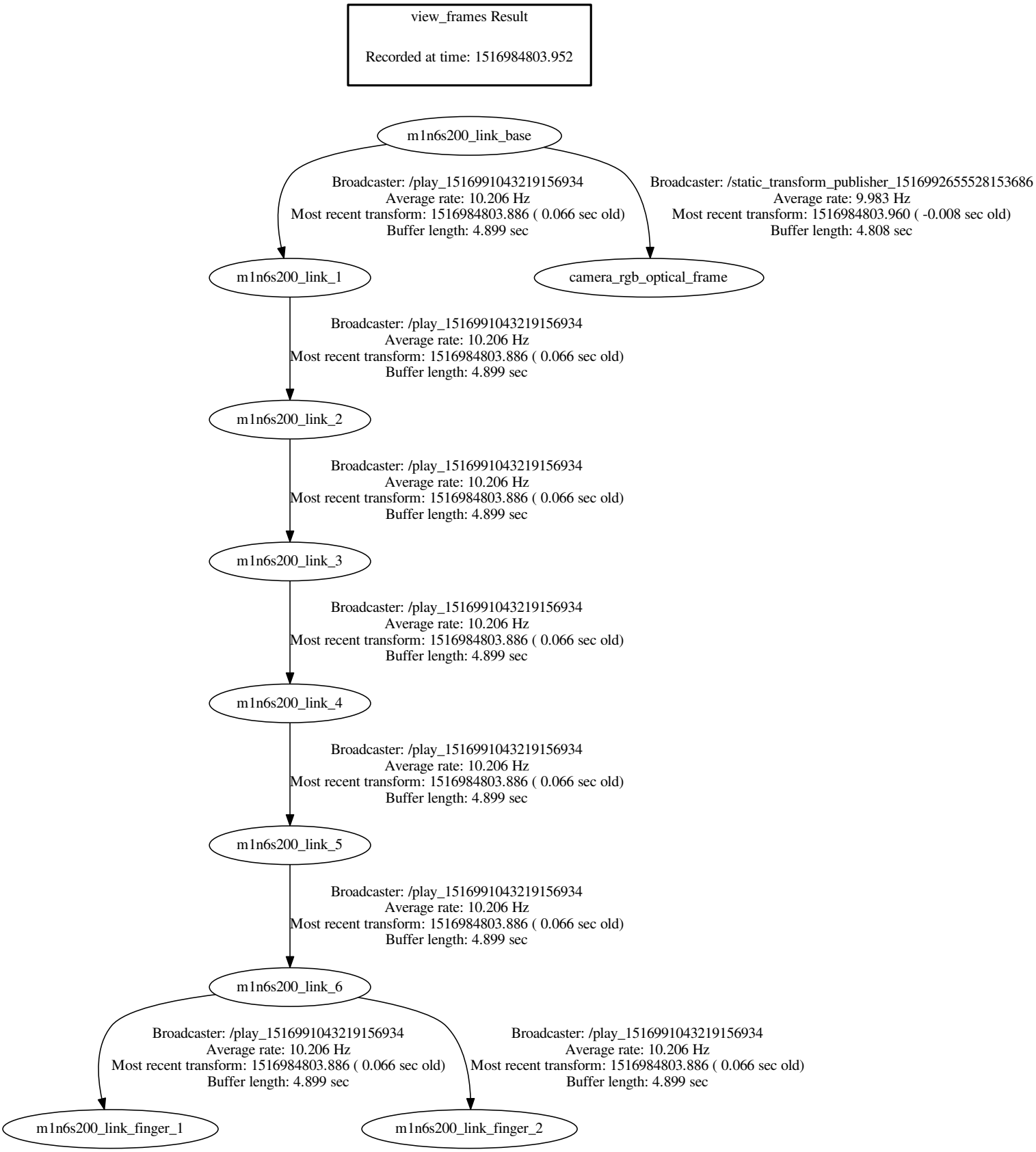
8 Results & Conclusion

Our results are a close to accurate estimate of the Kinect sensor's position and orientation relative to the robotic arm. It is "close to accurate" because errors are always present, either in delays between the recordings of the sensor and the arm feedback, or in miscalculations due to the angle of the green joint.

The main issues in the development process of this project were based on finding strategies to work around the problems we had, for example understanding the data structures of the pointcloud and tf messages, and finding a way to obtain the exact values for each calculation. In general, and how these problems were resolved, having worked around these somewhat unfamiliar obstacles helped our growth as future engineers.

References

1. OpenCV library
   https://docs.opencv.org/
2. PointCloud2 sensor messages
   http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html
3. PointCloud2 ROS documentation
   http://docs.ros.org/kinetic/api/pcl_ros/html/annotated.html
4. tf ROS documentation
   http://docs.ros.org/kinetic/api/tf/html/c++/annotated.html


view_frames Result
Recorded at time: 1516984803.952
m1n6s200_link_base
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
Broadcaster: /static_transform_publisher_1516992655528153686
Average rate: 9.983 Hz
Most recent transform: 1516984803.960 (-0.008 sec old)
Buffer length: 4.808 sec
m1n6s200_link_1
camera_rgb_optical_frame
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
m1n6s200_link_2
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
m1n6s200_link_3
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
m1n6s200_link_4
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
m1n6s200_link_5
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
m1n6s200_link_6
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
Broadcaster: /play_1516991043219156934
Average rate: 10.206 Hz
Most recent transform: 1516984803.886 (0.066 sec old)
Buffer length: 4.899 sec
m1n6s200_link_finger_1
m1n6s200_link_finger_2