

# Documentation: Simple Vision Control of 3<sup>rd</sup> Arm

Changsong Li

Advisor: Professor Guy Hoffman

## 1. Introduction:

We are building a smart robotic 3<sup>rd</sup> arm that attaches to your elbow. The purpose of the whole project is to develop and integrate sensing and control for this robotic arm using depth sensors, IMUs and ROS. In fall 2016 semester, Rain Zhou mainly focused on the hardware design like building the arm, plus the voice control; Sara Gregg mainly focused on IMU sensing; I mainly focused on vision control; Vighnesh Vatsal is the leader, who helped in every part and collected the sensing or vision data to control the arm actions.



(Shooting in our robotic lab, use speech recognition to control the arm grab the cup)

As for vision control, the general goal is controlling the 3<sup>rd</sup> arm to react intelligently corresponding to users' gestures or even eyeball movements. One scenario is that the 3D camera could detect the arm wearer are trying to approach an object, then we send commands to the 3<sup>rd</sup> arm automatically to control it move to the object and grab it. Well, this documentation is only about a simple vision control I implemented, you could follow it as a start.

## 2. Configurations(System Environment) and Prerequisites

### 2.1 Environment:

Ubuntu Trusty 14.04 LTS

ROS indigo

### 2.2 Prerequisites

- a. Get a computer with Ubuntu Trusty 14.04 installed. Actually the lab computer already has the proper system installed. You could type command “lsb\_release -a” to check it. Well, I don’t recommend to use virtual machine because I used it (VMware and Parallel) but meet a lot of driver problems for Kinect. You could try it.

- b. Follow the tutorials on [wiki.ros.org/](http://wiki.ros.org/) (ROS Wiki) to install ROS indigo

- c. Learn all the things in the beginner level on ROS Wiki. It would be easier for the future work to get a good understanding of the concepts workspace, package, topic, publisher, subscriber etc. Note don’t just read the code and concepts in the book. Try to write it by yourself and run it!

Go through fast for chapter 1 to 4 of book *ROS By Example Volume 1*(R. Patrick Goebel). Then following the instructions on chapter 5 to install all the required package first. In fact, for vision part, you don’t need to install so many packages, but you may try other parts in the future. So it’s better to install all of them.

In chapter 5, it tells you to clone the rbx1 package from GitHub. Don’t do it.

Clone the modified code in my GitHub repository by typing

“git clone <https://github.com/lcs2011cs/rbx1.git>”

(Note: I only modified code in rbx1\_vision package)

- d. Read the chapter 10, Robot Vision in the book. Read the codes carefully, try to run it using Kinect 1(in fact the lab Kinect is Kinect for windows, a version between 1 and 2) and understand the whole structures. Get a roughly idea of the algorithm used in the code. Then you are done for the preparations.

(Note here I recommend to use the driver freenect rather than openni. Because openni sometimes has problems for sensing depth data for Kinect for windows).

### 3. Description of What I did

#### 3.1 Configured the drivers for Kinect 1 and Kinect 2 in ROS.



(The left is Kinect 1, or Kinect for windows. The right is Kinect 2)

Kinect 2 senses depth data with higher accuracy, and gives RGB and depth image with higher resolution (1920\*1080, 512\*424 compared to 640\*480, 320\*240 in Kinect 1 and Kinect for windows).

For Kinect2, we need to install the driver `iai_kinect2` (you could find the code for it in GitHub by Google it). It has its own viewer and bridge codes compared while people has already developed a lot of packages for Kinect 1. It's hard to use but if you could manage Kinect 1 very well, you could try Kinect 2, as the logic of viewing image and transferring ROS image data to format used in OpenCV is very similar.

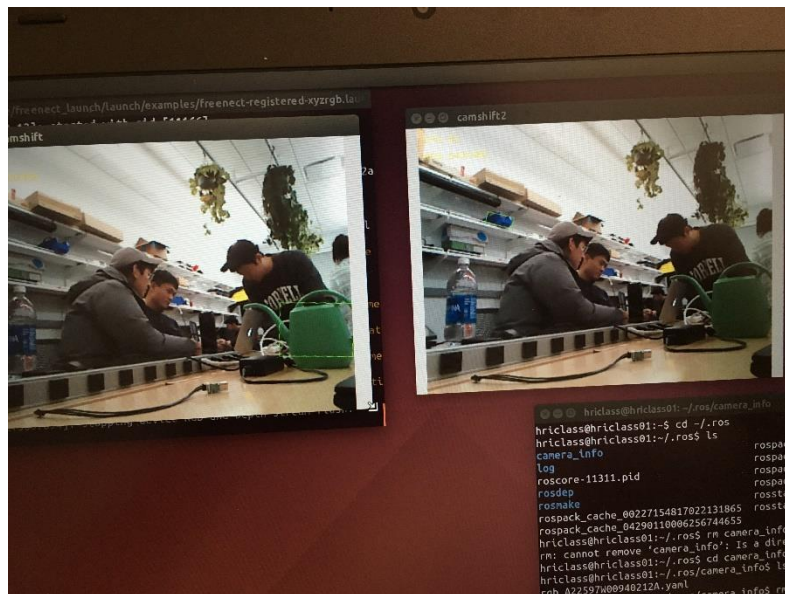
#### 3.2 Modified the algorithms for tracking color blob.

If you go through the chapter 10 carefully, you should already have a good understanding of the camshift algorithm to track color blob on 2D image. Well, I still used the same algorithm. The differences here compared with the code in the text book is the info published. You could see the files `camshift.py` and `camshift2.py` for details (There are comments in the code). The first file is used to track object and the second is used for tracking 3<sup>rd</sup> arm. The contours for the object I used here is a rotated rectangle, just the same with the examples shown in the text book.

#### 3.3 Wrote codes to track the 3D vector from 3<sup>rd</sup> arm to object and the moving direction of 3<sup>rd</sup> arm.

See code `pos_calculator.py` for details. The code first subscribed the info published by `camshift` and `camshift2` nodes. After getting the rotated rectangle (the contours of the object we are tracking) info, we do the following steps to estimate the 3D-positions of the object and 3<sup>rd</sup> arm.

- a. Scale the rotated rectangle by a factor like 0.8. Note not all the pixels in the rectangle belong to the object as the object may have any kinds of shapes. Since we only want to calculate the depth values of pixels belonging to object, we ignore the pixels in the margin to reduce the noise.

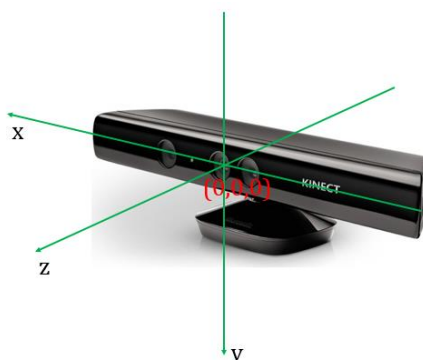


E.g. in the above figure, we are tracking the green sprinkling can. The contour is the green rectangle. We could see that the margin pixels within the rectangle is not belonging to the can

- b. Calculate the average depth value (z-axis) of all the pixels within the scaled rectangle. Then get the deviations of the depth value of every pixel from this mean value. Note if the deviation is very big for some pixels, then it probably indicates that these pixels are not on the object. So we should eliminate these pixels and recalculate the average depth. Do it until we think all the pixels participating in calculating the average depth are belonging to the object. You could set a threshold to stop the iterative steps. This is also used for reducing noise.
- c. After a and b, we get the 3-D position of the object. The same for 3<sup>rd</sup> arm.

Then we can get the vector (unit is meter) pointing from 3<sup>rd</sup> arm to object. Also it's not hard to calculate the moving directions of the 3<sup>rd</sup> arm as we calculate the 3D position of the 3<sup>rd</sup> arm with the rate of publishing data. We just used a variable to store the previous position, then after a time period, used the current position minus the previous one. Finally publish this data, which will be used by Vighnesh.

Note the 3D coordinates we calculate are based on the camera coordinate system, which is shown as following.

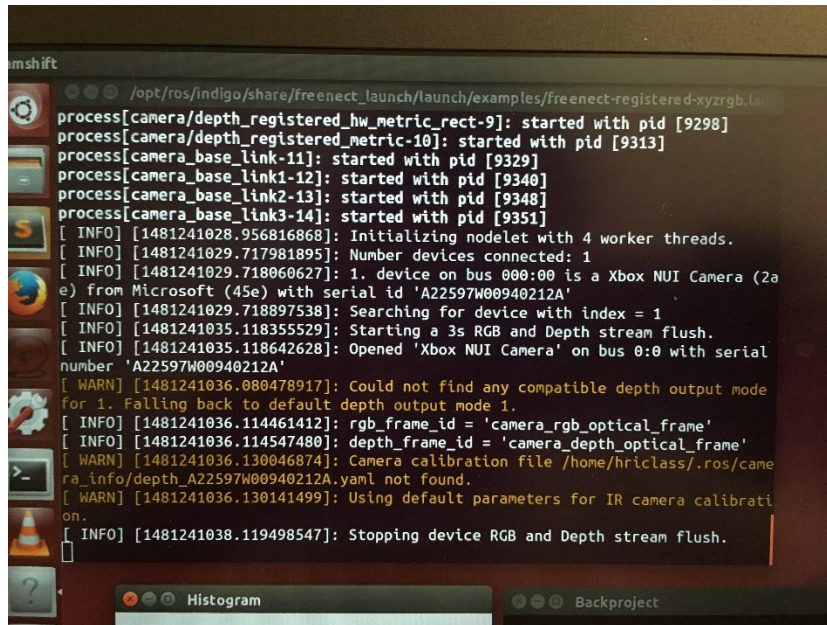




#### 4. Instructions of Running the Current Version

Make sure you have finished all the environment configurations. Then

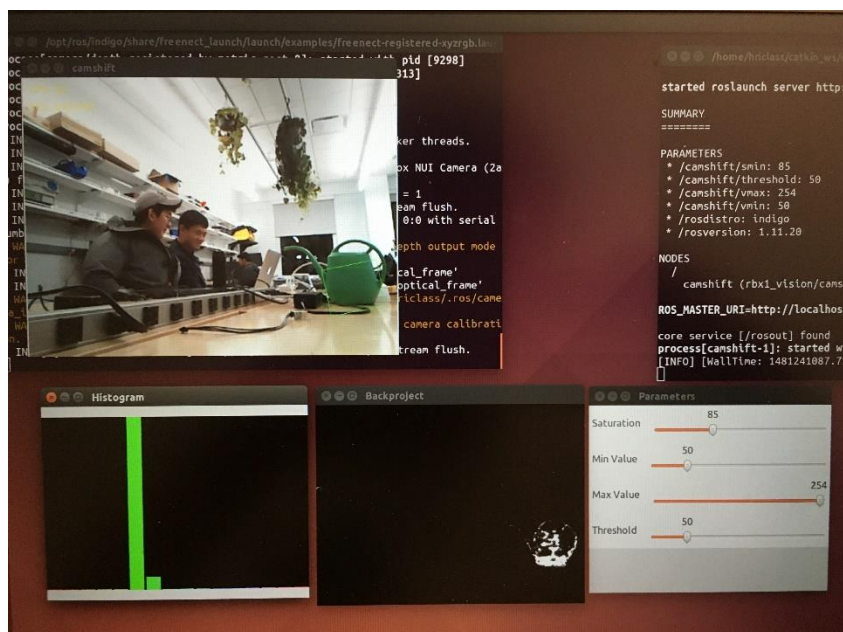
- Run the freenect driver. Command: `roslaunch freenect_launch freenect-registered-xyzrgb.launch`. The result should be like the following figure:



```
process[camera/depth_registered_hw_metric_rect-9]: started with pid [9298]
process[camera/depth_registered_metric-10]: started with pid [9313]
process[camera_base_link-11]: started with pid [9329]
process[camera_base_link1-12]: started with pid [9340]
process[camera_base_link2-13]: started with pid [9348]
process[camera_base_link3-14]: started with pid [9351]
[ INFO] [1481241028.956816868]: Initializing nodelet with 4 worker threads.
[ INFO] [1481241029.717981895]: Number devices connected: 1
[ INFO] [1481241029.718060627]: 1. device on bus 000:00 is a Xbox NUI Camera (2a
e) from Microsoft (45e) with serial id 'A22597W00940212A'
[ INFO] [1481241029.718897538]: Searching for device with index = 1
[ INFO] [1481241035.118355529]: Starting a 3s RGB and Depth stream flush.
[ INFO] [1481241035.118642628]: Opened 'Xbox NUI Camera' on bus 0:0 with serial
number 'A22597W00940212A'
[ WARN] [1481241036.080478917]: Could not find any compatible depth output mode
for 1. Falling back to default depth output mode 1.
[ INFO] [1481241036.114461412]: rgb_frame_id = 'camera_rgb_optical_frame'
[ INFO] [1481241036.114547480]: depth_frame_id = 'camera_depth_optical_frame'
[ WARN] [1481241036.130046874]: Camera calibration file /home/hriclass/.ros/came
ra_info/depth_A22597W00940212A.yaml not found.
[ WARN] [1481241036.130141499]: Using default parameters for IR camera calibrati
on.
[ INFO] [1481241038.119498547]: Stopping device RGB and Depth stream flush.
```

Note currently no calibration files are provided for RGB camera and IR (depth) camera. Actually if you want to do calibration, you only need to care about RGB camera as the registered-xyzrgb already do the transformation between RGB camera and depth camera. For calibration, follow the instructions on ROS Wiki [http://wiki.ros.org/camera\\_calibration/Tutorials/MonocularCalibration](http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration) [http://wiki.ros.org/openni\\_launch/Tutorials/IntrinsicCalibration](http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration). It's very simple. Then you can get your calibrated file uploaded.

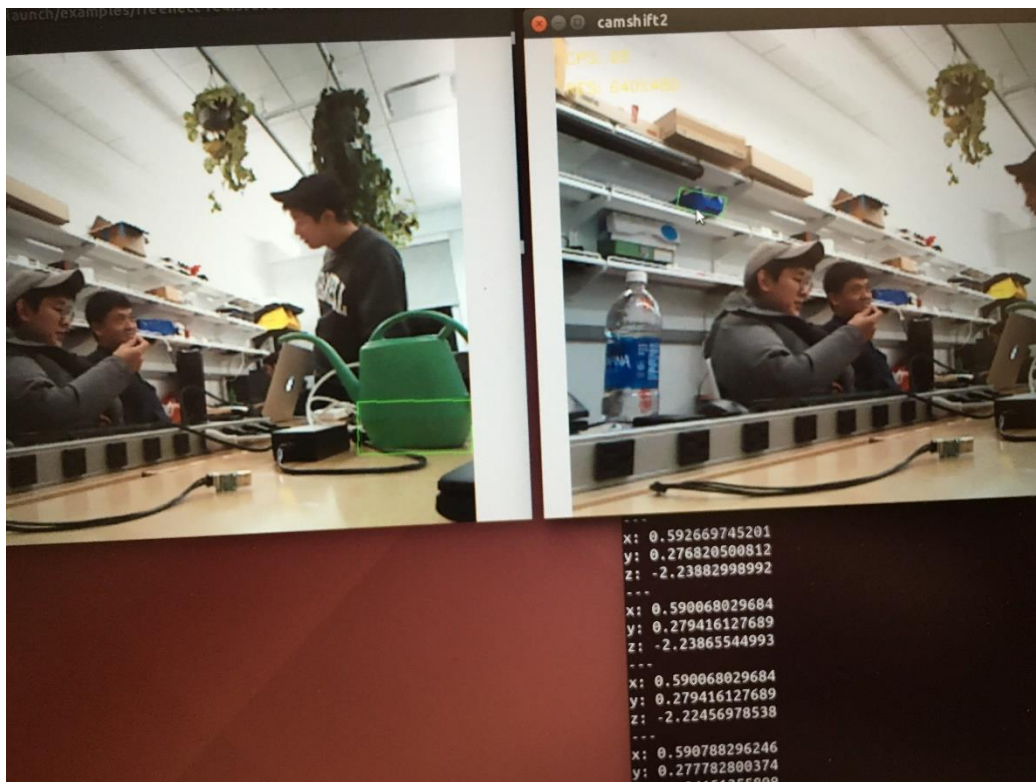
- Run the camshift node to track a color blob (object)



Command: `roslaunch rbx1_vision camshift.launch`

You should see the result similar to the figure above after using your mouse selecting the object you want to track. Here we track a green sprinkling can. The Histogram window shows the color info of the object. The Parameters window allows you to adjust parameters for camshift algorithm. The Backproject window shows the probability of each pixel having the same color info shown in Histogram window. White means 1 and black means 0.

- c. Run `camshift2` node to track 3<sup>rd</sup> arm. Do the same thing in b.
- d. Run the position calculation node. Command: `roslaunch rbx1_vision pos_calculator.launch`. Then if everything is correct, the node will publish the 3D vector pointing from 3<sup>rd</sup> arm to the object on topic `/direction`.



Here the green sprinkling can in the left image represents the object. The blue box in the right image represents the 3<sup>rd</sup> arm. And in the command line you could echo the info published on `/direction`. As shown in the above figure, the vector from the blue “3<sup>rd</sup> arm” to the green object is (0.59, 0.27, -2.22). Note the vector values are calculated based on camera coordinate system shown in 3.3

## 5. Recommended Future Work

### 5.1 Finish the recommended example demo

The demo is: First we have an object, which is a color blob (note it's better to use some light color which is easy to be distinguished from other things in the image). Also attach some color blob on the 3<sup>rd</sup> arm's hand. Run camshift to track the object and 3<sup>rd</sup> arm. Then move the 3<sup>rd</sup> arm slowly to approach the object will lead the control system to automatically control to 3<sup>rd</sup> arm to reach the object and grab it for the wearer.

### 5.2 Use different tracking algorithm

We could see there are limitations of the tracking algorithm now. You could try different algorithms, like just using feature points tracking to track 3<sup>rd</sup> arm. Or you could even use some 3D object tracking algorithm, which you can learn in the book *ROS By Example Volume 2*.

### 5.3 Try Kinect 2

As we said before, Kinect 2 has higher accuracy data. The main challenge for using it is to writing a data converting scripts, which should be very similar to `ros2opencv2.py` in `rbx1_vision` package.

**TAG:** It's really a goof lab with so many interesting topics about Robotics. The Professor and students here are nice and patient. You could learn a lot of things not only on the robotics fields, also on how to do your researches and projects better. Hope you could enjoy your time in the lab and do some big things!