Kinect v2 extrinsic calibration with Baxter robot

Introduction

In this short report, a detailed process of calibration between Kinect v2 sensor and Baxter robot is introduced. The propose of this process is to connect the robot and sensor to expend the robot vision ability, particularly the depth sensor of Kinect v2. In detail, we merge the Kinect origin frame into Baxter tf tree as a local coordinate.

Process Detail

Preliminary
Before doing this calibration. The Kinect v2 sensor should finished driver installation and intrinsic calibration. I would recommend the 'IAI Kinect2' driver. The driver can be used in the Linux ubuntu 14.04 environment with ros indigo. A visual fiducial system is also useful for the further step The AprilTags is recommended for this calibration. There is also a ros package can be easily used.

Kinect mount
The first issue is the position and device to fix the Kinect sensor. A reasonable position for sensor related to Baxter robot is vital for further project. The general idea is to maximize the kinect visible range inside the Baxter workspace. However, if you have some further thoughts of final robot arm movement scope, it is definitely a good idea to consider them.

In detail, after surfing the Internet and with my own thought, there are four candidates to choose. These four choices can be divided into two groups mounting on the robot and face to face with Baxter. The second group is easy to understand that you mount Kinect on a tripod and put it opposite to the Baxter. This candidate can have the best view especially for the movement relate to a table because there is no interference with robot arm. However, this method can not guarantee a static Kinect position. You have to recalibrate if the position is moved and because of the separation it is hard for you the detect whether the position is changed or not.

The other group is to mount Kinect on the Baxter robot with customized 3D printed parts. The first choice is on top of the head. Notice that there is a fixed screw mount inside the hole in the head's cap above the LED rings. You could use that as a fixed mounting point for a 3D printed bracket. Another more aggressive method is to replace the default head with a customized one. From the Internet I even seen a mount with a step motor, so the view scope of kinect can be controlled to move. However, for the angle movable method, the calibration is kind of annoy. The second choice is to sit on top of the screen. This one is the easiest method because you only need to print a U bend link fit with the Baxter screen and use it to connect Baxter and Kinect. However, the occlusion issue is very serious. The third one the mount on the front rail. We choose this candidate because it can guarantee a static position also an acceptable vision range. A two-part mount is also designed.

Procrustes analysis
Notice that before doing Procrustes analysis, you have to decide a mount choice and keep the Kinect stable. As we talked before the core of this calibration is to merge the Kinect coordinate into Baxter coordinate system, so the Kinect frame is not the origin anymore but a local frame inside the Baxter tf tree.

Procrustes analysis determines a linear transformation of the points in one matrix (kinect) to best

conform them to the points in another matrix (Baxter). I use the Matlab 'procrustes' function, which return the translation component and the Orthogonal rotation and reflection component. I would recommend you read the Matlab Documentation of Procrustes first. After you read it, you may be confused how to find the input data, so now our ApirlTags comes to the stage. Basically, the input data is two data sets one for Kinect, one for Baxter and they both represent one mass-point set translation with different coordinate system. ApirlTags is used to represent this point set inside the Kinect view precisely and stably. The pose from Kinect can be found from the ApirlTags ros package topic. For Baxter, we also want to get the tag position but it is trickier compared with the Kinect. We start from the Baxter topic 'endpoint_pose'. This topic returns the end position pose of the Baxter gripper. We use tape to attach a tag on the end position of the gripper and claim that data from the 'endpoint_pose' is the tag pose related to Baxter.

Understand the meaning of the input data, now we need to know to record them and how many data do we need. For the first question, I put the tag vertical to the table (parallel to the Z and Y coordinates of Baxter). I did this because the Procrustes only need the translation information, so I try to keep the rotation distortion away. From the second question, I record 60 points and these points are distributed among the Kinect view uniformly (consider the depth). Theoretically you can do the recording process by hand controlling the Baxter arm and manipulate the data by hand too. I create a simple code to control Baxter arm in workspace with keyboard while keep the arm 'vertically' using Python reference to the Baxter example code. You need to input an initial translation in Baxter coordinate first and this code can let the arm move in six directions in Cartesian coordinate related to input location. For data pro-precess, I do it by hand considering the data amount is not that big but it is definitely a good exercise to manipulate it though code.

With all input data, go through the Matlab function and you will get the translation and rotation of Kinect origin related to the Baxter coordinate. The final step is to 'input' it to Baxter. There are two ways to do that. The first one directly modifies Baxter .urdf file and add this information as a child of the ‘base’ of Baxter inside the file. I try to do this way but it fails because looks like Baxter does not recognize my modification. The second way is to use tf package to add a frame. You can just use 'roslaunch' to call 'static_transform_publisher' inside the tf package and it works like a charm. You can also create a ros node to add as extra frame to tf and this method is more fit for a movable frame.

Result check

Generally speaking, the result is very precise especially for the translation part and for rotation there is an argument order mismatch issue of rotation representation between Matlab and ros, but with a hand modify it is easy to solve.

There is an output argument from Matlab Procrustes function to check the sum of squared errors and this could be the first check. You can also transform the tag pose from Kinect to Baxter though the generated Procrustes result and compare it with the 'endpoint_pose' to check the accuracy.