Vision System Documentation

Nick Stanley (gstanley@andrew.cmu.edu)

13 August 2012

1 Background

In our "cage" the robot sits with a downward facing Kinect, its camera roughly perpendicular to the x-y plane. The position $(0, 0)$ for the world frame is the corner of the table (the left corner as you approach it in the room) with x increasing towards the right and y increasing towards the nearest wall, and z pointing upwards (which correspond to convention). The kinect system has a point cloud of 640×480 points.

The kinect frame is different from the world frame, so the `container_node` gets the transform and outputs `robot_points` at 0.5 Hz, which is the points in the world frame.

2 The Calibration System

The robot's calibration system is fairly straightforward:

1. The robot hand moves to a certain position in the optimal vision range.
2. The robot hand's attachment (a premade checkerboard) is analyzed by the Kinect's RGB camera.
3. The points from the corners of the RGB camera are correlated to points in the cloud.
4. Those points are paired with the location of the robotic arm in the world frame, known by encoders on the mechanism itself.
5. After gathering these points, they are loaded into Matlab and we use Horn's Absolute Positioning method (source here) to find the transform.
6. The transform is written to a rosparams file, which is then loaded into the server.
7. When you start the `container_node`, it reads from the server and applies that transform to the input cloud.

The syntax is simply: `roslaunch launch calibration.launch`

Be sure to have the system running in some form when launching `calibration`. When it is complete, it will say "[calibration] process has finished cleanly". In which case it's okay to just ctrl-C out of it.

3 The Camera System / The Container Node

The container node is simply started with `roslaunch launch container.launch`

It simply takes in a depth cloud from the Kinect and applies the transform loaded from `transform_params.yaml` to it. This takes a significant portion of time, since the depth cloud is broadcasted at a frequency of 30 Hz, and the output is almost exactly 0.5 Hz. The reduction should not be an issue for machine learning purposes in the near future, but other methods may exist to reduce the time taken:

1. Only transform a subset of the cloud. Identify which parts are very far from the table (i.e., the ground, for example) and ignore those points instead of applying the transform to them.
2. Only transform the points when they are needed. For example, when searching in a subset, transform the a subset that you will predict to correlate (based on the triviality of the transform).

4 Modifications

This is a list of things that may be done in the future to change the program. First, I want to state the main constraint that we have on the calibration sequence and container node: the Kinect must be facing downward, perpendicular to the table. It can be slightly off, but some of the math relies on this constraint.

4.1 Writing New Calibration Positions

Go into the file `src/get_points_in_both_frames.cpp` in the package `calibration`. There is a list of joint positions `float jointPositions[numPositions][numJoints]`. Use the tablet controller to manually move the robot to a certain position and record the joint pose. Then, put the set of poses into `jointPositions`, where each subarray is a position composed of six floats, one for each joint. Then, the robot will move to each in sequence during calibration.

5 Troubleshooting

A description of current bugs (and database for future bugs).

5.1 Container Node Cannot Get Parameters

It will read "Did not get tx" or ty or tz or qw, etc. This means that the file `transform_params.yaml` in the package `parameters` is not properly filled out, or that (generally) the parameters were not loaded into the parameter server. Make sure that you ran the launch file rather than simply running the executable (the latter does not automatically load the parameters into the server).

5.2 Calibration Infinite Loops

While trying to get the corners from a checkerboard, the robot might not see a complete picture. Use `image_view` to see what the Kinect is seeing, and check to make sure that 1) there is nothing blocking the line of sight and 2) there is sufficient lighting (but not an overabundance from the sun–the Kinect doesn't behave well with IR interference).

5.3 Calibration Failed and Wrote to File

Have no fear! If you are able, just rerun the calibration. If not, the old file is saved in the folder `old_transform_params` under the package `parameters`. Note that the name of the file is the date when it was moved in the folder and made obsolete, not when it was created. Most likely it will simply be the most recent one, so simply copy paste it to the file `transform_params.yaml` under the package `parameters`.

5.4 Calibration Didn't Get Transform

Check the output of MATLAB in the terminal where you ran the system. If it says that "absor" is an undefined function, and that "file 'absor.m'" was not found, check to make sure that the file is on the computer where the system is being run. If not, the file can be downloaded here as of 15 August 2012.