Sensing Objects and Manipulating them Using Baxter

Abhilash Narayana(an472), Chaitanya Mitash(cm1074), Anagha Desai(ad1014)

June 29, 2017

This report provides details about the overall approach used for sensing cylindrical objects and manipulating them using Baxter.

1. Getting the Image stream:
   Steps followed to get the image stream from Baxter’s left hand camera:
   (a) We subscribed to the /cameras/left_hand_camera/image topic and defined a callback function that is called for each image received.
   (b) In the callback function, a globally defined image variable is updated to reflect the new image received.

2. Invoking Object Detection and Placing Objects in the Simulator:
   (a) The simulator subscribes to the topics ”/ready_to_plan” and ”/planning_plans” which are advertised by the planner.
   (b) In the callback functions for both these topics, the following tasks are performed:
      i. get the tf transform between ”/base” and ”/left_hand_camera_axis” - this was implemented using a TF listener.
      ii. The Object Detection code is then called by passing the globally defined and constantly updated image.
      iii. After the object detection is complete, the randmoize_position() function is called which places the recognized objects in the simulator according the color, position and orientation as returned by the Object detection code.
      iv. The planner can start planning once the simulator is updated and this can be communicated to the planner by publishing on the topic ”manipulation_request”

3. Object Detection:
   (a) Image Segmentation
      i. We first converted the image to gray scale and used blur to smooth the image.
      ii. Implemented the automatic thresholding method to calculate the threshold value.
      iii. The results from automatic thresholding was not able to perform proper segmentation as it was also detecting large sections of background which were relatively bright, as objects – so we went with a manual constant threshold which works in the room conditions.
iv. Next we find contours in the image by using the `findContours()` function of OpenCV and we are only interested in the external contours for each object, so we pass as a parameter, `RETR_EXTERNAL`.

v. We filtered out the small contours in the image which could not possibly be the object of interested based on the \textit{CONTOUR_SIZE_THRESHOLD} set by observations.

(b) Finding Centroids of the Detected Objects

i. We use the `moments()` function of OpenCV to find the moments for each contour.

ii. We used the formula in the textbook to calculate the centroid. This is in the image coordinate system.

(c) Finding Orientations of the Objects

i. We use the OpenCV function `fitEllipse()` over each contour to find the min ellipse covering the objects.

ii. We decide whether the object is standing or sleeping on the table, based on the difference between the width and height of the objects. \textit{SLEEPING_OBJECT_THRESHOLD} is set by observations.

iii. For sleeping objects, we use the orientation of the major axis of the ellipse thus obtained from the min ellipse.

(d) Finding color

i. We first convert the image to \textit{HSV} format and for each contours, we pass the vector of points in the contour to the function \textit{findColor()}.

ii. In this function, we count the number of points in range of each of the four colors, and return the color which has maximum number of points in its range.

iii. We also update the probability with which the object is detected for a particular color.

4. Positioning Objects in the Simulator:

(a) Iterate through the object list returned by the simulator.

(b) Retrieve the color of object, by reading the material color attribute.

(c) Find the next object of the color with the highest probability, and place it at the desired position and orientation.

(d) In case some objects are not placed, we find in the list of detected objects which are not placed, the ones which might be yellow as yellow is the one which has least chance of detection.

5. Transformations:

(a) First we find the intrinsic camera parameters from the topic CameraInfo by extracting \(f_x, f_y, c_x, c_y\) from the \(P\) matrix which are the focal length/pixel size and principal point coordinates.

(b) We calculate the position of centroid in camera frame by using these parameters, converting them to \((\text{row}, \text{column})\) format. However, multiplication by the depth is held until we are placing the object in simulator.

(c) The depth of object is retrieved for standing objects as the difference in heights of table and camera(retrieved from tf) - the height of centroid when object is standing(0.07) and for sleeping the centroid is at height(0.02). This is based on the dimensions of the cylindrical objects provided.
(d) To convert the position to global frame we use the tf information available.

(e) The height of the objects in global frame is fixed a little above the table height to get plans. The height was fixed by observations.

(f) The orientation of the standing objects was fixed as 0,0,0,1.

(g) For sleeping objects we first rotated the objects about the x-axis by 90 degrees to make it sleep. And then about z-axis by angle obtained in the detection. This was in the camera frame. We then composed the transformation between camera and global frame.

(h) As the orientation of the camera w.r.t base was not perfectly vertically facing down(it had 8 degrees shift) we calculated the overall approximate transform and used it.

6. Decision making for Picking Objects:

   (a) We read the input from a file "color.txt" indicating color and number of objects of that color we are allowed to pick and maintain a count of these.

   (b) We find the next standing object which is closest to the robot along the x-axis and is allowed by the input file. Assumption is that standing objects are easy to plan for.

   (c) If no such objects exist we go for the sleeping objects.