Object Identification in Dynamic Environments

Andrew Chou, Elliott Jin, Wendy Mu

December 10, 2010

1 Introduction

The overall problem our project tackles is to have a robot be able to recognize objects that have been introduced or moved around in an environment. More specifically, given a room before and after new objects have been introduced, we would like to be able to recognize how many new objects there are, whether they should be picked up or not, navigate to where those that should be picked up are, and pick them up.

1.1 Applications

The main application for this would be to have a robot be able to clean a room, by having it first scan to see what the clean room should look like, and then go back after some period of time when the room has been changed, and clean up the room. Other practical applications include setting up a room or searching for objects in a crowded environment.

2 Methods and Data

To collect and test data, we used the Personal Robot 2 (PR2), which has the capability to take laser scans of its environment. We used the PR2 to take repeated scans of a room, before and after we introduce new objects to the environment. The laser scans returned 3D point clouds that represented the objects and layout of the environment. Additionally, we preprocessed the 3D point cloud files to make the format compatible with the rest of our processing pipeline. The PR2 uses the Robot Operating System (ROS), so the code for our project is all in a form that can be run on the PR2. The duration of each scan was 100 seconds, giving on the order of 680,000 points in each point cloud, which was generally dense enough for our algorithms.

3 Point Cloud Alignment

In practical applications, the robot would take the before and after room scans at two different times, so it would be difficult to ensure that the robot will be in the exact same position and orientation for both scans. As a result, the 3D point clouds from the before and after scans will not even have constant elements aligned, such as a wall or big table. In order to align the before and after scans of the room, we used the Iterative Closest Point (ICP) algorithm. We used a pre-existing implementation of the ICP algorithm that ROS already contained. This was the an iterative approach using the Levenberg-Marquardt algorithm, which is used to minimize the non-linear least-squares error. Because the robot at the point would be in the perspective of the second scan, we transform the old point cloud to be aligned with the new point cloud.

4 Background Subtraction

To find the differences between the two scans, we removed all the points in the second scan that were less than some small fixed distance away from any point

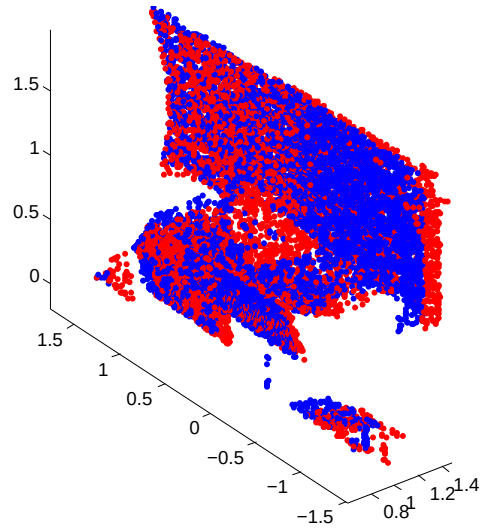

Figure 1: Before aligning before (red) and after (blue) scans

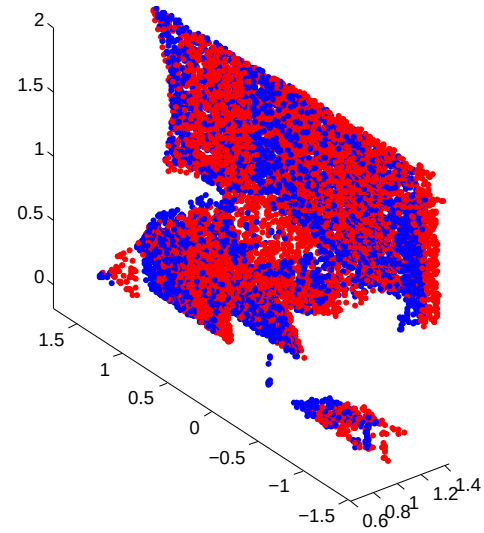

Figure 2: After aligning before (red) to after (blue) scan

in the ICP-aligned first scan. After background subtraction, we are left with a point cloud that is a subset of the points in the second scan, representing the objects that have been introduced into or moved in the environment. In particular, we only compute the difference one way so that objects that have been moved are treated the same as objects that have been introduced, and objects that have been removed are ignored. This makes sense because practically, the robot should only deal with what is in the room at the time of the second scan.

```
removeBackground(PointCloud before, PointCloud after)
begin
  for i := 1 to after.size step 1 do
    for j := 1 to before.size step 1 do
      if dist(after[i], before[j]) < threshold)
        continue;
      fi
      after.remove(i);
    od
  od
end
```

This code implemented directly turned out to be slower than we would prefer given the size of the point clouds, so we have also implemented some optimizations by taking advantage of knowing the ranges of the x, y, and z- coordinates.

We empirically determined a suitable value for the threshold, which we set to be 0.04 meters, or 4 centimeters.

5 Identifying Object Clusters and Location

5.1 Clustering

In the point cloud resulting from background subtraction, we have only the points that correspond to new or moved objects in the second scan. If more than one object has changed, we would like to be able to know exactly how many objects there are.

Here, we assume that none of the objects are touching; otherwise from only a 3D point cloud it would be difficult to tell whether two touching objects were one oddly-shaped object or actually two. Assuming objects are not touching, figuring out which points correspond to different objects becomes a clustering problem.

In order to determine how many new objects were introduced, we used a variation of single-linked clustering. Two points belong to the same cluster if there is a path between them consisting of intermediate points such that no distance between intermediate points is greater than some fixed minimum distance. We used an agglomerative method in which we iterate through the list of points not yet in a cluster, and add them if they are less than the fixed minimum distance away from any point in the current cluster that we are building. If no points are less than this distance away, then we move onto building a new cluster. Using this algorithm, we can determine how many clusters (objects) there are and which points correspond to each object.

The algorithm we used is as follows:

```
find_neighbor(PointCloud cloud, int[] labels,
  int position, int curr_label)
begin
   for i := position + 1 to cloud.size step 1 do
       if (labels[i] = 0 ∧
        dist(cloud[position], cloud[i]) < threshold)
          labels[i] = curr_label;
          find_neighbors(cloud, labels, i, curr_label);
       fi
   od
end


find_clusters(PointCloud cloud, int[] labels)
begin
   curr_label := 0;
   for i := 1 to cloud.size step 1 do
       if labels[i] = 0
          num_labels := num_labels + 1;
          labels[i] = num_labels;
          find_neighbors(cloud, labels, i, num_labels);
       fi
   od
   return curr_label;
end
```

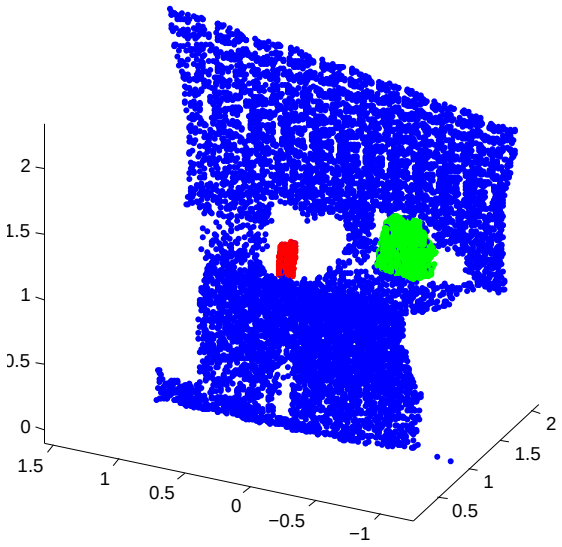

Figure 3: After background subtraction and clustering: This figure shows the new scan with objects introduced (blue), and the two clusters that were recognized by our clustering algorithm after the background was subtracted, a juice carton (red) and a backpack (green)

The method find_clusters gives each point a label corresponding to the object it belongs to. The method returns the total number of clusters that we found in the point cloud, representing the number of changed objects between the first and second scans.

To remove noise that could show up as clusters, we only considered point clusters that had more than 400 points, which we empirically determined would be a good cutoff given the minimum size of the set of objects we were considering and the amount of time we spent scanning the room (how dense the point cloud scan was). We also empirically determined a suitable value for the threshold, which we set to be 0.03 meters, or 3 centimeters.

5.2 Location

To represent the location of each object, we use the center of mass of the object based on the points in the cluster that correspond to the object.

6 Object Classification

It is possible that new objects will be introduced into the room that we may not want the robot to pick up, such as furniture or other large objects that it cannot physically pick up. Thus, we additionally implemented binary classification on point cloud objects to help the PR2 decide which objects should be picked up. We would like to pick up objects that are "trash." To pick features for this classification problem, we thought about properties about objects that distinguish trash from non-trash. For instance, a smaller object is more likely to be trash than a much larger object, etc.

6.1 Features

The main features we chose to examine were related to the principal axes of the object, or the lenghts of an appropriately rotated bounding box. The ten features we calculated were:

1. Volume of the bounding box
2. Area of each of the three distinct face of the bounding box
3. Lenth of each of the three sides of the bounding box
4. The ratio of each of the three pairs of the bounding box

We implemented batch gradient descent, because our training set would just be the set of clusters representing objects that we obtained from the robot, and this is a small enough set that non-stochastic gradient descent would not be too slow. Since we had relatively few data points, we implemented leave-one-out cross validation so that we could potentially verify that we won't have overfitting based on our ratio of features to number of data points.

7 Integration with the PR2

The code for our project uses ROS (Robot Operating System) [1] that runs on the PR2, so all of this process can be done on the robot. We have pipelined the process so that, when already given data from the before and after scans of the room, the robot can automatically align the point scans, subtract out the background, pick out the clusters that are objects, navigate to the object, and point its head towards the object.

In ROS, each step must be implemented as ROS node, which is an executable that can communicate with other ROS nodes, and in particular, the PR2. We have created the following ROS nodes:

1. Point Cloud Listener with ICP: Takes a scan of the new room and aligns the old scan with the new scan.
2. Background Remover: Subtracts out common elements between the two aligned scans.
3. Cluster Finder: Classifies the remaining points into different clusters while discarding noise.
4. Classifier: Classifies whether the robot should attempt to pick up the object or not.
5. Navigation Server: Sends navigation goals to the PR2 to navigate to objects in its environment and point its head towards the object.

8 Conclusion

This paper shows the fundamental steps needed to consider having a robot clean a room. There are many challenges to this process that can be addressed using concepts in machine learning. As this paper has shown, among them are aligning

scans taken from different perspectives, finding differences between rooms, clustering objects, classification, and navigation. Other challenges in the future may include aligning scans from vastly different perspectives (as opposed to from mostly the same position), more advanced object classification, and learning how to pick up different objects.

9 Future Work

Although we have created and tested all the individual ROS nodes mentioned above, we have not fully tested the integrated nodes together as a complete pipeline. In the future, we would like to integrate all these nodes so that the robot can automatically do all of these steps without human intervention.

We would like to further optimize some parts of the process in terms of speed, especially the background removal and clustering algorithms.

Additionally, we would like to gather much more data so that we could have more training examples to test each part of our pipeline more rigorously, but especially to test the object classification code that we have written.

We would like to add grasping capabilities at the end of our pipeline; that is, given that it is in front of an object that it needs to pick up, have the PR2 actually pick up the object from the environment.

Finally, we would like to implement further improvements to object classification. For instance, in addition to being able to only classify "trash" and "non-trash," it would be helpful if the robot could determine what kind of object it is so that after picking it up from the environment, it could put it back in the correct place.

10 Acknowledgement

We would like to thank Ellen Klingbeil and Morgan Quigley for their help on working with ROS and the PR2, as well as Professor Ng for his suggestion to work on the project of having a robot clean a room.

11 References

[1] *ROS Wiki*, `http://www.ros.org`.