Point Cloud Segmentation for Mobile Robot Manipulation

by

Charlotte Zhu

B.S., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2016

Certified by . . . .
Prof. Tomás Lozano-Pérez
Thesis Supervisor

Accepted by . . . .
Dr. Christopher Terman
Chairman, Masters of Engineering Thesis Committee

Point Cloud Segmentation for Mobile Robot Manipulation

by

Charlotte Zhu

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we develop a system for estimating a belief state for a scene over multiple observations of the scene. Given as input a sequence of observed RGB-D point clouds of a scene, a list of known objects in the scene and their pose distributions as a prior, and a black-box object detector, our system outputs a belief state of what is believed to be in the scene. This belief state consists of the states of known objects, walls, the floor, and "stuff" in the scene based on the observed point clouds. The system first segments the observed point clouds and then incrementally updates the belief state with each segmented point cloud.

Thesis Supervisor: Prof. Tomás Lozano-Pérez

Acknowledgments

Special thanks to my thesis supervisor Tomás Lozano-Pérez for putting up with me for so long, answering all my emails and questions, and helping me whenever I got stuck. I'd also like to thank Jared Glover and Lawson Wong for their mentorship and guidance, my friends for all the laughs and distractions, and of course my family for their neverending love and support. I would have never made it without them.

Contents

1 Introduction **13**

2 Related Work **15**

2.1 Object Detection . . . 15

2.2 Manipulation Planning . . . 16

2.3 Building a Semantic Belief State . . . 17

3 Semantic Segmentation **19**

3.1 Objects . . . 20

3.1.1 Object Detection . . . 20

3.1.2 Region Growing . . . 20

3.2 Floor and Wall Planes . . . 21

3.3 Stuff . . . 23

3.4 Results . . . 24

3.4.1 Scene 1 . . . 24

3.4.2 Scene 2 . . . 25

3.4.3 Scene 3 . . . 26

3.4.4 Scene 4 . . . 26

4 Building a Semantic Belief State **29**

4.1 Objects . . . 29

4.2 Floor Plane and Wall Segments . . . 31

4.3 Stuff . . . 32

4.4 Results . . . 33

4.4.1 Scene 1 . . . 34

4.4.2 Scene 2 36
4.4.3 Scene 3 38
4.4.4 Scene 4 41

5 Conclusions and Future Work **45**

List of Figures

1-1 Flowchart of our system. 14

2-1 BPA single object detection pipeline [6]. 16

3-1 Pipeline for point cloud semantic segmentation. 20
3-2 Scene 1: Wall and floor 24
3-3 Scene 2: Box on table. Robot knows box and table are in scene. 25
3-4 Scene 2: Box on table. Robot knows table is in scene, does not know box is in scene. 26
3-5 Scene 3: Box and Cascade bottle on table. Robot knows box and table are in scene, does not know Cascade bottle is in scene 27
3-6 Scene 4: Box on table with wall. Robot knows box and table are in scene . . 28

4-1 Belief state representation. 30
4-2 Scene 1: Box on table. 35
4-3 Belief state after multiple observations of scene 1. 35
4-4 Scene 2: Box on table near glass wall. 37
4-5 Belief state after multiple observations of scene 2. The purple points correspond to the pillars on the glass wall, the green corresponds to points on the table that were slightly outside of the detected table pose, and the yellow corresponds to the cable hanging down to the left of the table. 37
4-6 Scene 3: Table with Cascade bottle. 39

4-7 Belief state after multiple observations of scene 3. For (a)-(d), the purple points correspond to the original Cascade bottle, the yellow corresponds to the moved Cascade bottle, and the green corresponds to points on the table that were slightly outside of the detected table pose. For (e)-(f), the purple corresponds to the original Cascade bottle and the green corresponds to the moved Cascade bottle. 40

4-8 Scene 4: Table with box and Cascade bottle. 43

4-9 Belief state after multiple observations of scene 4. The purple points correspond to the Cascade bottle, the yellow corresponds to points on the table that were slightly outside of the detected table pose, and the green corresponds to points on the ground that were not associated with the floor plane. 44

List of Tables

2.1 A comparison of precision and recall for object detection algorithms [6]. . . . 16

List of Algorithms

1 Region Growing Psuedocode 21
2 Plane Finding Psuedocode 22
3 Clustering Psuedocode 23
4 Update Object Psuedocode 31
5 Update Walls Psuedocode 32
6 Update Stuff Psuedocode 34

Chapter 1

Introduction

The introduction of robotics into human-centric indoor environments requires robots to be able to autonomously move around the environment and interact with objects and their surroundings. But in order to autonomously navigate through its surroundings, a robot needs to know which places are free to move into and what obstacles it might collide with. In order to manipulate an object, a robot must be able to find and identify the object in its surroundings. All of these point to the need for a robot to be able to first semantically reason about and understand its spatial surroundings to some degree.

However, it can be difficult to identify all objects and obstacles in a scene accurately and consistently. Home environments can be more challenging than industrial environments because they usually involve a wide variety of objects, a high degree of clutter, and are generally dynamic due to the involvement of other agents (e.g. humans adding/moving/removing objects). Furthermore, object recognition systems are not yet perfect and tend to perform poorly in cluttered environments where there is potentially lots of occlusion.

Therefore, even if the robot starts with prior knowledge of the objects in the scene, there may still be objects or areas that it is unable to identify in its environment. The unknown areas could be a known object that simply couldn't be recognized by the detection system or perhaps a new, unknown object that was introduced into the scene at a later time. In the face of such uncertainty, it is necessary for the robot to use state estimation to maintain a representation of the state and its uncertainty of its surroundings.

In this thesis, we present a system that keeps track of not only the states of known objects across multiple observations of a scene, but also the states of unknown objects or points that the robot sees but cannot be identified, colloquially referred to as "stuff". Keeping track of "stuff" is important because it is a way of modeling and keeping track of the robot's uncertainty of its surroundings. It can also help the robot build a better semantic world model and thus help improve task and motion planning.

"Stuff" represents areas that the robot should avoid and/or look at again for additional observations, perhaps from a different viewpoint or angle. For example, it would be good if the robot could avoid collisions with the unknown objects while interacting with the known objects. It could also be that the unknown object is actually an object that the robot is looking for, but the object detector wasn't able to recognize it due it being partially obscured or being viewed from a strange angle or simply due to noise. In this case, the robot would want to look at the object again, perhaps from a different angle.

More specifically, in this thesis, we develop a system that takes as input a sequence of observed RGB-D point clouds of a scene, a list of known objects in the scene and their pose distributions as a prior, and a black-box object detector, and then outputs the belief state for what the robot believes is in the scene, represented as the states of known objects, walls, the floor, and "stuff," based on the observed point clouds (Figure 1-1).

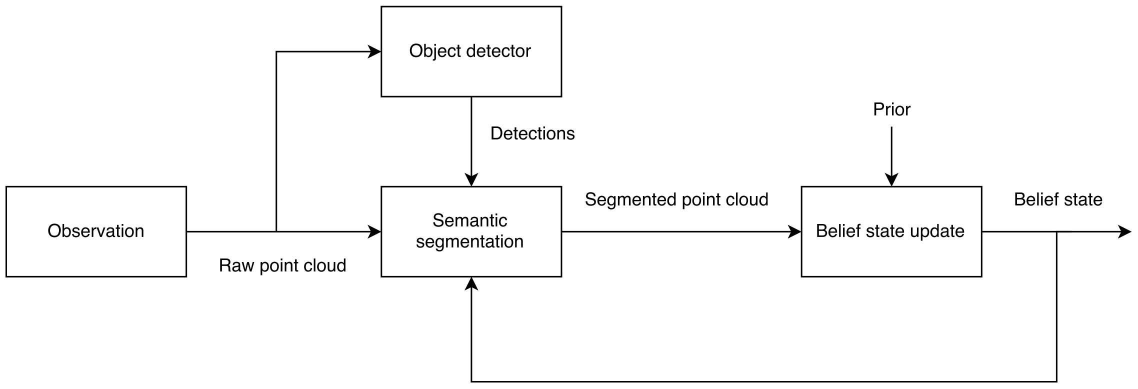


Figure 1-1: Flowchart of our system.

Chapter 2

Related Work

2.1 Object Detection

Object detection has long been a fundamental problem in robotics. But a lot of progress has been made in recent years thanks to advances in 3-D cameras and technology. The primary goal is to recognize a known object instance and estimate its 3-D position and orientation in space. Given an observed scene, usually represented as a RGB-D point cloud, and an object model, object detection systems try to align and match the model to the scene by minimizing an alignment cost function between corresponding points in the point cloud and the object model.

This alignment cost function can be specified in terms of sum-of-squared distances between nearest-neighbor points on the point cloud and the object model, and iterative algorithms like ICP (Iterative Closest Point) are guaranteed to reach a local minimum of the cost function [2]. However, ICP can be very slow because it attempts to find dense correspondences between the two point sets at each iteration. ICP is also highly susceptible to outliers. This can be problematic with in highly cluttered scenes since it then requires the system to properly segment the objects in the scene with little room for errors.

Instead, many modern alignment algorithms instead attempt to use sparse point sets and match only points computed at unique keypoints in the scene. These keypoint features include both position information and orientation information, such as 3D point positions, surface normals, curvature directions, edges, spin images, SIFT features, and FPFH features, etc. [7] [14] [17].

Our robot perceives the world through a Microsoft Kinect sensor, and the object detection algorithm we use is known as the Bingham Procrustean Alignment (BPA) algorithm [6]. BPA builds upon the object detection methods given in Tang et al. and Aldoma et al. and uses the Bingham distribution to model uncertainty on 3-D rotations [18] [1]. It provides a posterior distribution over the space of possible alignments of the object model given the point cloud data that also includes information from keypoint feature positions.

A brief overview of the BPA system's single object detection pipeline is shown in Figure 2-1. BPA was tested on two Kinect-based data sets, one from Aldoma et. al [1] referred to as Kinect and the other devised by the BPA authors referred to as Clutter. BPA's performance on these data sets is compared to that of ICP and the system used in Aldoma et. al in Table 2.1.

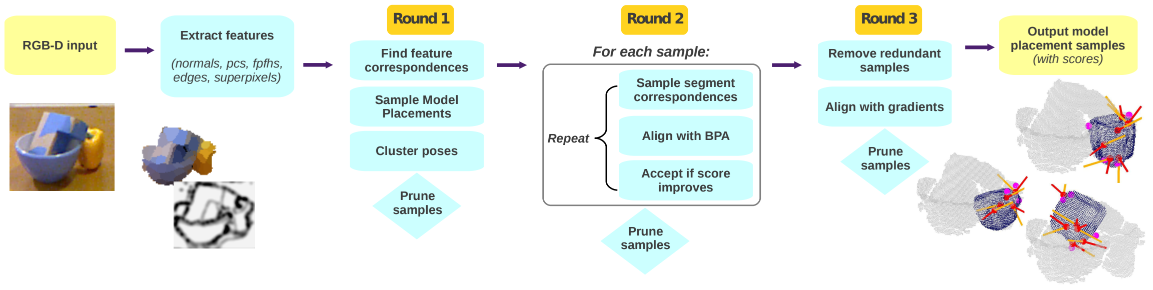


Figure 2-1: BPA single object detection pipeline [6].

	BPA		*ICP*		*Aldoma et. al [1]*	
	precision	recall	precision	recall	precision	recall
Kinect	89.4	**86.4**	71.8	71.0	**90.9**	79.5
Clutter	**83.8**	**73.3**	73.8	63.3	82.9	64.2

Table 2.1: A comparison of precision and recall for object detection algorithms [6].

2.2 Manipulation Planning

Traditional motion planning algorithms attempt to find paths between fully specified geometric configurations, such as a robot moving from an initial position to a target position without colliding with any obstacles [10]. However, they cannot address problems in which the configuration space of interest is not just that of the robot. For example, if the goal is to clean the kitchen, a motion planner could plan how to get to the table but not decide that the table needs to be cleaned.

On the other hand, task planners can reason over very large state spaces to produce sequences of abstract actions to reach a goal [15]. However, they usually do not take into account the geometric or kinematic constraints necessary to complete those actions. Returning to the previous example, if the goal is to clean the kitchen, a task planner could decide that the robot needs to go to the closet to get a mop regardless of the obstacles in between the kitchen and the closet.

HPN attempts to integrate both task planning and motion planning. It is a hierarchical approach to solving manipulation planning problems, which performs a temporal decomposition by planning operations at multiple levels of abstraction. This ensures that problems to be addressed by the planner always have a reasonably short horizon, making planning feasible [8]. The implementation of HPN we are using plans in belief space. Instead of maintaining a world state, it instead maintains a belief state and updates this belief state based on an observation resulting from executing the action in the world [9].

2.3 Building a Semantic Belief State

As a robot explores its environment, it is necessary to combine all the data it collects into a rich 3-D map of its environment. Most indoor environments also contain sources of semantic information such as objects that the robot can recognize. Thus, it is also desirable for the robot to be able semantically segment its map of the environment so it can reason about its surroundings at a higher level.

We want to be able to segment a map and then be able to update the map and the segmentation as the robot collects new data in real time. If the new data overlaps with the past data, then they should be merged together. This introduces two problems: recognizing when two point clouds correspond to the same place in an environment and then merging their segmentations together.

Finman et al. incrementally build a segmentation with new data that is added to an existing segmentation [5]. It uses the Felzenszwalb algorithm to segment the new data, and then tries to find border points of the new data in the old segments and re-segment the new data and the borders [3]. Another algorithm by Finman et al. uses objects for place recognition by detecting objects in new 3-D data, connecting them into a graph, and then comparing the graph to the full object graph based on previous data to see if there is a match [4].

We borrow some elements of both these algorithms by using detected objects as keypoints in matching new point clouds with our current map of the world and incrementally adding in new data to our existing segmentation.

Chapter 3

Semantic Segmentation

In order to update the robot's belief state of known objects, walls, the floor, and "stuff" given a new observation, we first need to be able to segment the robot's observed point cloud into these four categories.

In this chapter we present our algorithm for segmenting a point cloud. We take as input a RGB-D point cloud, the robot's current belief state, and a black-box object detector and output a segmentation comprised of known objects, walls, the floor, and "stuff." As previously mentioned, stuff represents areas that the robot cannot identify, perhaps an unknown object in the scene or an object that wasn't recognized by the object detector.

The robot's belief state is described in more detail later in Chapter 4, but it contains a list of known objects in the scene and their pose distributions based on what the robot has previously observed so far. These objects are a subset of the set of object models that the object detector can recognize.

We try to look for these objects in the current point cloud using the object detector. But if a detected pose returned by the object detector is too unlikely given its pose distribution, the detection is discarded. Then we find planes in the point cloud using RANSAC and try to identify them as either the floor or walls. With the remaining points, we use a clustering algorithm to find unidentified clusters of points of significant size as stuff. This pipeline is shown in Figure 3-1.

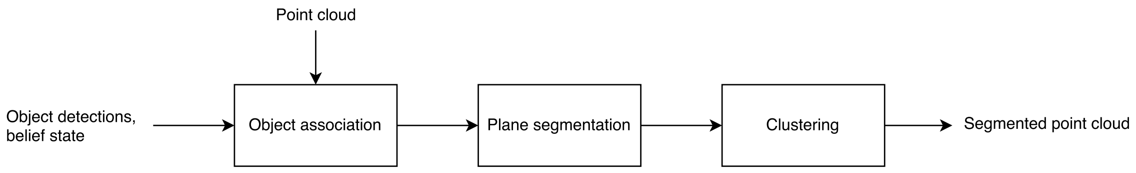


Figure 3-1: Pipeline for point cloud semantic segmentation.

3.1 Objects

3.1.1 Object Detection

Given the objects and their pose distributions in the robot's belief state, we want to try to find these objects in the current observed point cloud. For each object, we use an object detector to find a pose for that object type in the point cloud, and then we see how likely the detected object pose is given the pose distribution. If it is too unlikely given the pose distribution, we ignore it as a match and consider it as if the we could not find that object in the point cloud.

We use the Bingham Procrustean Alignment (BPA) algorithm as our black-box object detector [6]. It takes in a list of object models to look for in the point cloud and uses a multiple object pose estimation (MOPE) algorithm to find the poses of the object instances found in the point cloud [16]. It returns the set of poses for the set of object models with the highest score.

Given the detections returned by the object detector, we score how likely a detected object pose is given the pose distribution. If it is too unlikely, then we do not consider it a match. For example, if the robot believes there is a box on the left of the table, but the object detector detects a box on the right of the table, we consider it too unlikely and ignore the detected pose. Otherwise, we consider it a match.

3.1.2 Region Growing

The black-box object detector only returns object poses, not the points in the point cloud that the objects correspond to. In order to associate points to objects, for each detected object, we transform the mesh of the object to the detected pose and check for points that fall within the mesh. Then, we use a region growing algorithm (Algorithm 1) on these points to pick up other points that correspond to the object that may have not been aligned properly with the mesh.

We use the Point Cloud Library (PCL) normal estimation method to estimate the surface normals for all the points in the point cloud [11]. Then for each point p in the object region, we look at all points within a radius of r of p and test if the angle between their surface normals and the surface normal of p is below a certain threshold $\theta_{threshold}$. If it is, we add it to the region.

We used a radius of r of 0.03 meters and a $\theta_{threshold}$ of 0.3 radians. We also loosely bound the region by the object mesh's bounding box so it doesn't grow to be much larger than the object should be. This can happen in certain cases where the boundaries between objects are ambiguous.

We filter all of the points inside these object regions out of the point cloud for the next steps in the segmentation process.

Algorithm 1 Region Growing Psuedocode

```
procedure REGION–GROWING(pointcloud, region)
    estimate normals for points in pointcloud
    initialize queue $Q$ with points in region
    while $Q$ not empty do
        $p1 \leftarrow Q$.pop()
        neighbors $\leftarrow$ points in pointcloud within radius $r$ of $p1$
        for $p2$ in neighbors do
            $\theta \leftarrow$ angle between normals of $p1$ and $p2$
            if $\theta \leq \theta_{threshold}$ then
                region.append($p2$)
                $Q$.add($p2$)
            end if
        end for
    end while
    return region
end procedure
```

3.2 Floor and Wall Planes

After filtering out the points corresponding to the detected objects, we use a plane segmentation algorithm to find planes in the remaining points in the point cloud.

We used the PCL segmentation filter [13]. It uses the iterative RANSAC method for fitting a plane equation to the data points (Algorithm 2). We used a distance of 0.05 meters as a threshold for how close a point must be to the model in order to be considered an inlier.

If the returned plane contains more than a certain number of point inliers, we remove

Algorithm 2 Plane Finding Psuedocode

```
procedure FIND–PLANES(pointcloud)
    planes ← []
    while do
        find plane in pointcloud using RANSAC
        if plane inliers $\geq$ minimum plane size then
            planes.append(plane)
            filter out plane inliers from pointcloud
        else
            break
        end if
    end while
    return planes
end procedure
```

those points and try to find another plane in the remaining data points. We repeat this until no more planes can be found. We used a minimum plane size of 150 inliers.

To check if any of the planes correspond to the floor plane, we first assume that the floor lies perpendicular to the z axis around $z = 0$ meters. Then we check if any of the planes found match these characteristics within some threshold. We consider the plane to be perpendicular to the z axis if its normal vector is close to $(\mathbf{0}, \mathbf{0}, \mathbf{1})$. We allowed an absolute difference of $(\mathbf{0.5}, \mathbf{0.5}, \mathbf{0.1})$ between the plane's vector and a normal vector of $\mathbf{0}, \mathbf{0}, \mathbf{1}$ and a maximum floor height of $z = 0.1$ meters.

For walls, we assume that the walls will be mostly parallel to the z axis. If we find a plane that satisfies this, we then count how many of the points in the original point cloud are on one side of the plane and how many fall on the other side. If one of these totals is close enough to zero, indicating most of the points fall on only one side of the plane, we assume it is a wall.

We check whether the plane's normal vector $\mathbf{v}$ is mostly parallel to the z axis by thresholding the vector values. More specifically, we checked whether $|\mathbf{v}_3| \leq 0.1 \land (|\mathbf{v}_3| \geq 0.5 \lor |\mathbf{v}_3| \geq 0.5$. We allowed a maximum of 50 points to fall on one side of the plane in order for it to be considered a wall.

We filter all of the points corresponding to the floor and wall planes out of the point cloud for the next steps in the segmentation process. .

3.3 Stuff

Assuming the previous steps have all worked correctly, the remaining points in the point cloud should ideally be either objects that the object detector could not recognize or objects that were not in the robot's belief state, implying that the robot did not know about them beforehand. We use a clustering algorithm (Algorithm 3) to cluster the remaining points together and ignore clusters that are of insignificant size.

We used the PCL cluster extraction method to find clusters of points that contain at least a minimum number of points [12]. We used a minimum cluster size of 25 points, a radius r of 0.05 meters for finding neighbor points, and a $\theta_{threshold}$ of 0.3 radians.

Algorithm 3 Clustering Psuedocode

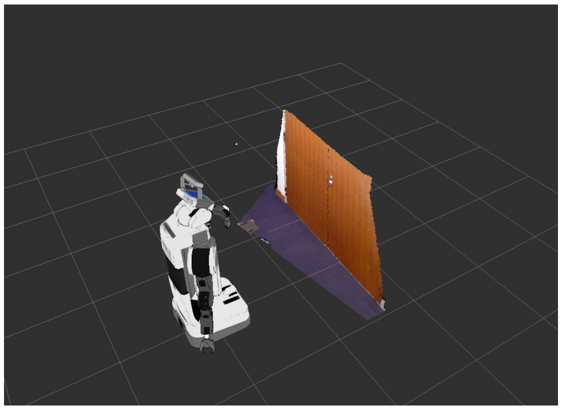
```
 1: procedure CLUSTERING(pointcloud)
 2:     $clusters \leftarrow []$
 3:     $processed \leftarrow []$
 4:     for point $p$ in $pointcloud$ do
 5:         if $p$ not in $processed$ then
 6:             initialize queue $Q$ to empty
 7:             $cluster \leftarrow []$
 8:             $Q$.add($p$)
 9:             while $Q$ not empty do
10:                 $q \leftarrow Q$.pop()
11:                 $cluster$.append($q$)
12:                 $neighbors \leftarrow$ points in $pointcloud$ within radius $r$ of $q$
13:                 for $candidate$ in $neighbors$ do
14:                     if $candidate$ not in $Q$ and $candidate$ not in $processed$ then
15:                         $\theta \leftarrow$ angle between normals of $candidate$ and $q$
16:                         if $\theta \leq \theta_{threshold}$ then
17:                             $Q$.add($candidate$)
18:                         end if
19:                     end if
20:                 end for
21:             end while
22:             $clusters$.append($cluster$)
23:             add all points in $cluster$ to $processed$
24:         end if
25:     end for
26:     return $clusters$
27: end procedure
```

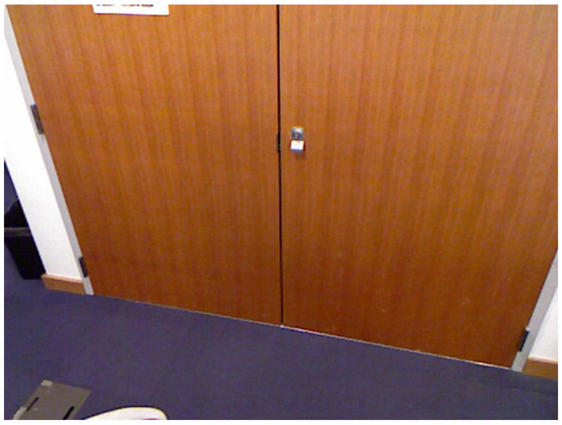
3.4 Results

In the following results, in each segmentation, the point cloud is drawn in cyan, and the segmented portions are then drawn on top of it. Red points correspond to the floor, and pink points correspond to walls. Purple, yellow, and green points correspond to "stuff."

3.4.1 Scene 1

Scene 1 shown in Figure 3-2 contains a wall and the floor. The robot did not know of any objects in the scene from its current belief state.


(a) Point cloud visualization


(b) Kinect RGB image

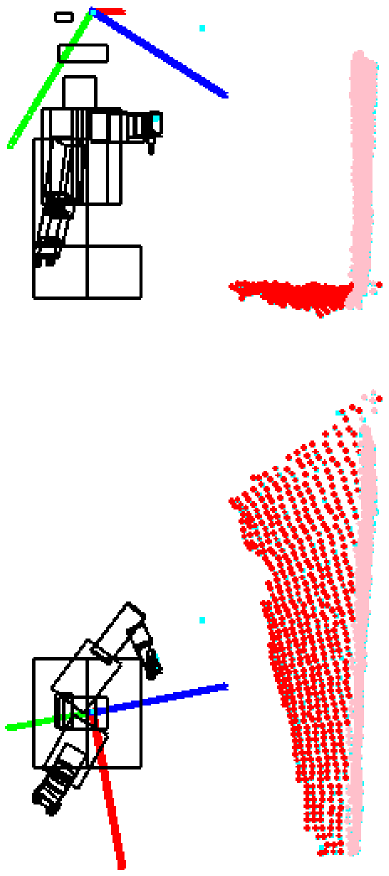
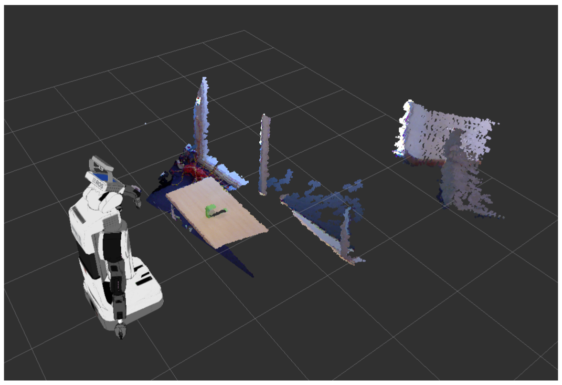

(c) Point cloud segmentation

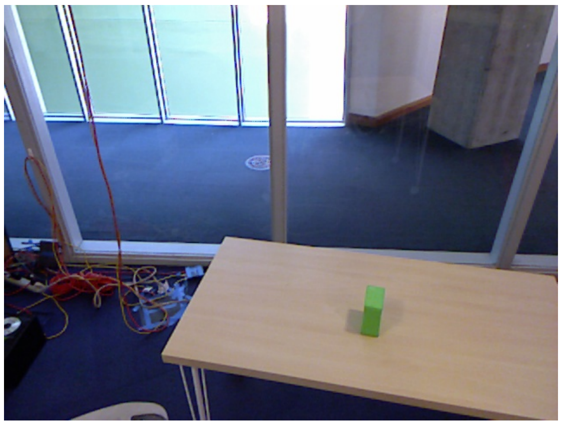
Figure 3-2: Scene 1: Wall and floor

3.4.2 Scene 2

Scene 2 contains a table with a green box on top. In Figure 3-3, the robot knew about the box and table objects from its current belief state. The detected box and table poses are outlined in magenta and blue respectively. The yellow and green points correspond to the pillar on the glass wall behind the table as seen in Figure 3-3 (b).

Scene 2 is also used in Figure 3-4. But this time, the robot knew about the table object from its current belief state, but it did not know about the box object. So in the segmentation, the box is now "stuff" since it was not in the robot's belief state as an object and is drawn in yellow. As before, the detected table pose is outlined in blue, and the green and purple points correspond to the pillar on the glass wall behind the table.


(a) Point cloud visualization


(b) Kinect RGB image

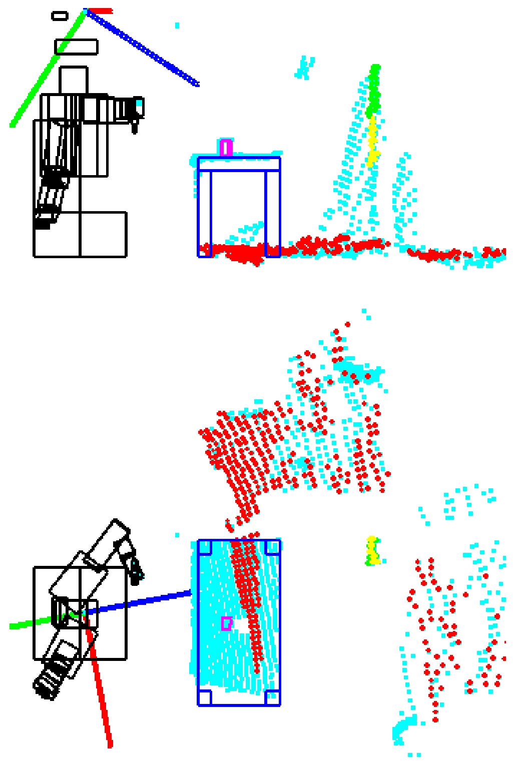
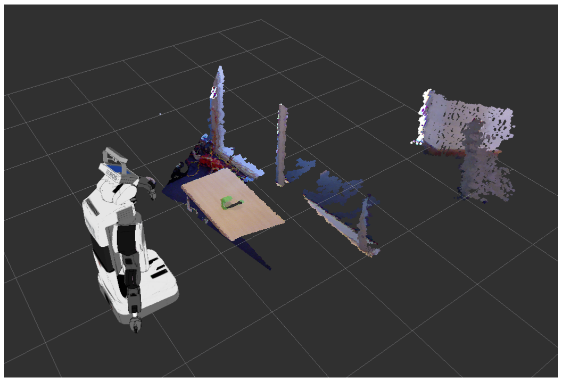
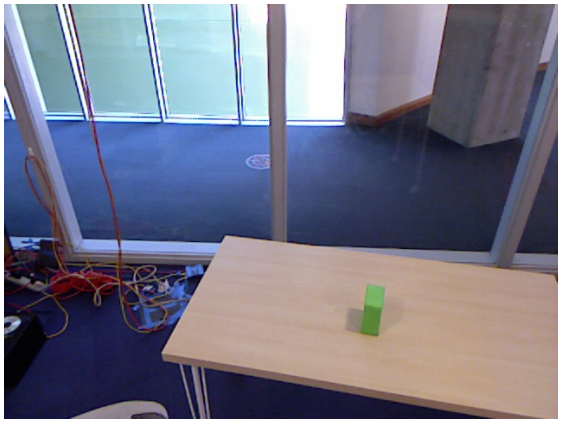

(c) Point cloud segmentation

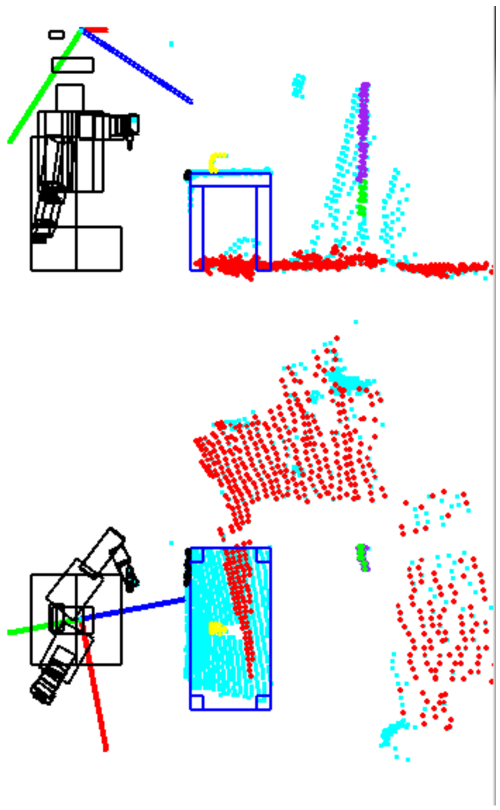
Figure 3-3: Scene 2: Box on table. Robot knows box and table are in scene.



(a) Point cloud visualization



(b) Kinect RGB image



(c) Point cloud segmentation

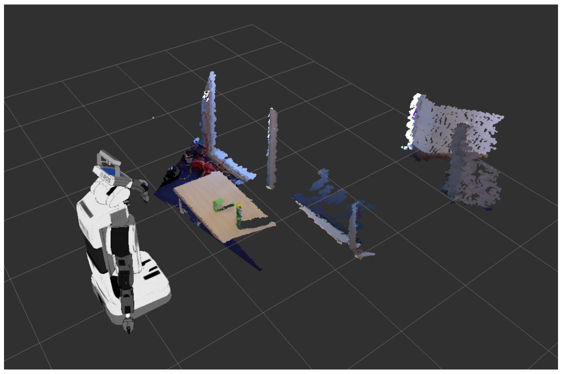
Figure 3-4: Scene 2: Box on table. Robot knows table is in scene, does not know box is in scene.

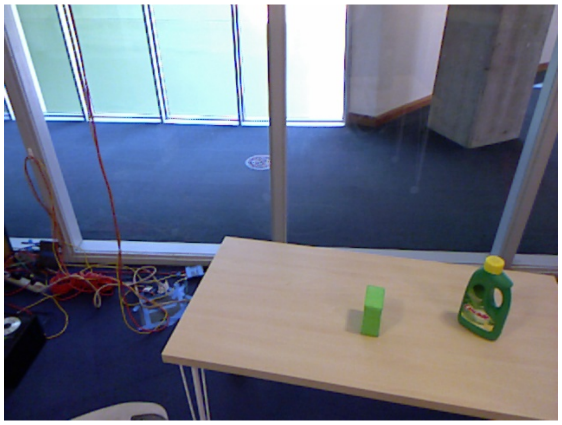
3.4.3 Scene 3

Scene 3 is similar to Scene 2, except now there is also a green Cascade bottle on the table as well as green box. In Figure 3-5, the robot knew about the box and table objects from its current belief state and detected both objects. It did not know about the Cascade bottle. The detected box and table poses are outlined in magenta and blue respectively. Here, the yellow points correspond to the unknown Cascade bottle, and the green points correspond to the pillar on the glass wall behind the table.

3.4.4 Scene 4

Scene 4 is similar to Scene 1, except now there is also a table with a green box on top of it in front of the wall. In Figure 3-6, the robot knew about the table object from its current belief state, but it did not know about the box. Here, the green points correspond to the


(a) Point cloud visualization


(b) Kinect RGB image

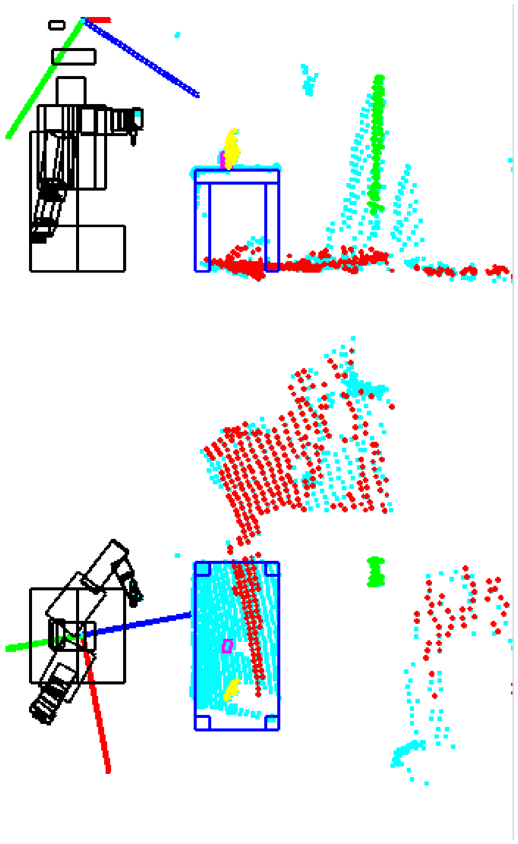
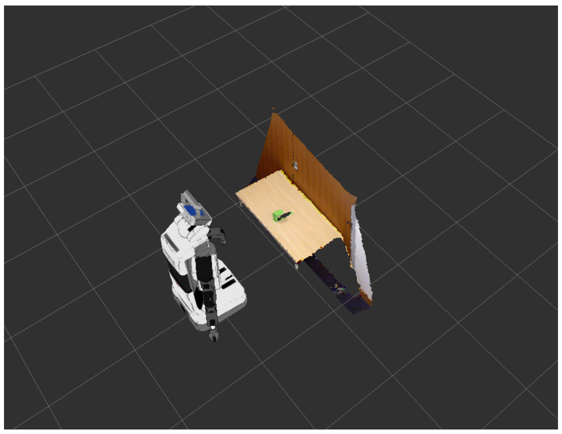
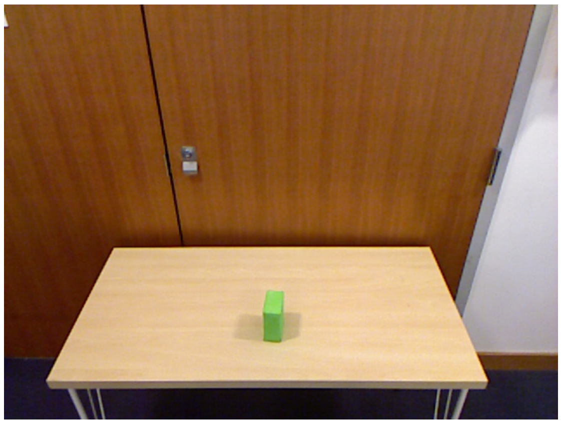

(c) Point cloud segmentation

Figure 3-5: Scene 3: Box and Cascade bottle on table. Robot knows box and table are in scene, does not know Cascade bottle is in scene

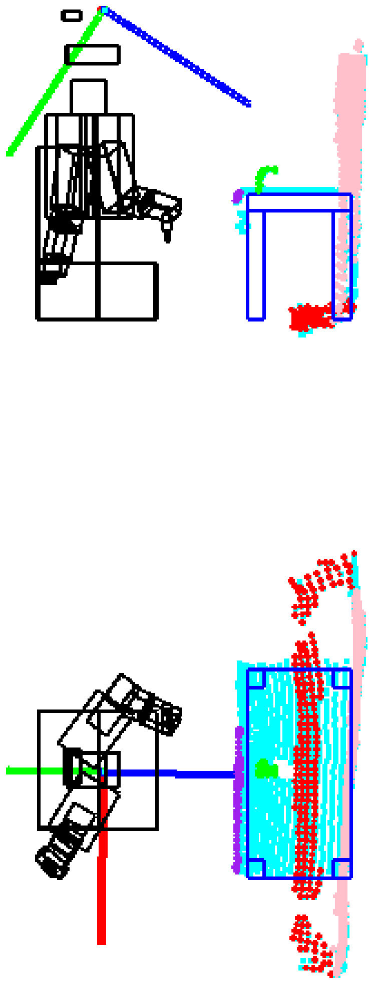
unknown box, and the purple points correspond to points that were not aligned properly with the detected table pose. Since we assume that the detections are correct, we do not correct for this. But it is perhaps useful for the robot to still have some way of knowing there is an obstacle there so it doesn't run into the table because it believes the table is a little farther back than it actually is in reality.



(a) Point cloud visualization



(b) Kinect RGB image



(c) Point cloud segmentation

Figure 3-6: Scene 4: Box on table with wall. Robot knows box and table are in scene

Chapter 4

Building a Semantic Belief State

Now that we have a way to segment a point cloud, in this chapter, we present our algorithm for merging multiple segmented point clouds together.

We take as input a sequence of segmented observed point clouds and a list of known objects in the scene and their pose distributions as a prior and output the belief state of what the robot believes is in the scene based on the segmented point clouds. This belief state is represented by the states of known objects, walls, the floor, and stuff.

We initialize our belief state with the given list of known objects in the scene and their pose distributions. Then we update it incrementally with each segmented point cloud. We assume that observations occur frequently enough so there is not a lot of a change in between sequential point clouds. If the robot moves a lot in between sequential point clouds, then it becomes much harder to register point clouds at adjacent time steps since the point clouds could be of completely different areas of the room.

Our belief state tracks the states of known objects, walls, floor, and stuff. Each of these is represented in a different way, which is detailed in each section below (Figure 4-1).

4.1 Objects

An object is represented as an object type, a pose distribution and the probability of the object being in the scene, which we refer to as the mode probability. Its object type corresponds to an object model for the object detector to use.

When the robot first starts, it is given an initial list of known objects and their pose distributions in the scene. This is used as a prior for the objects in the robot's initial belief

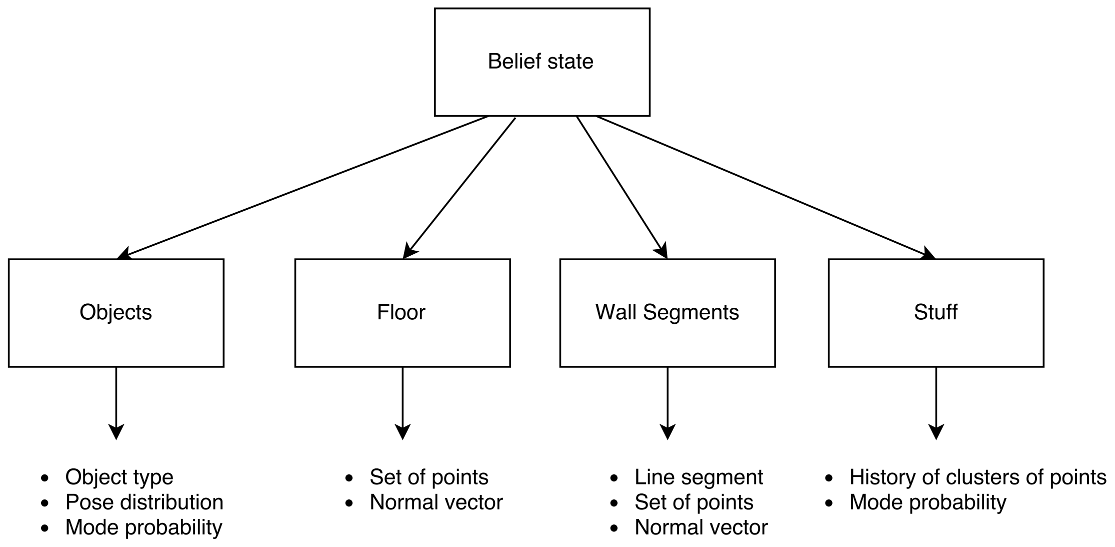


Figure 4-1: Belief state representation.

state of the world. We assume that this prior knowledge of the world is correct. That is, if the robot is initially told that there is an object of some type with a certain pose distribution in the world, then an object of that type with a pose in that pose distribution exists in the real world. We initialize the mode probability of all the objects to 0.99.

Then we update both the distribution and mode probability for each segmentation (Algorithm 4). If the segmentation contains an object pose for an object, we update the pose distribution for that object based on the detected object pose using a Kalman filter update. We also update the mode probability of the object given an observation using Bayes' formula (Equation 4.1). $P(X)$ is the previous mode probability, $P(Y)$ is the probability that we observed the object, and $P(X|Y)$ is the mode probability given that we observed the object. We used $P(Y|X) = 0.95$ and $P(Y|\overline{X}) = 0.05$ to model the probability that the observation was an error.

$$P(X|Y) = \frac{P(X) * P(Y|X)}{P(X) * P(Y|X) + P(\overline{X}) * P(Y|\overline{X})} \tag{4.1}$$

If the segmentation does not contain an object pose for an object, we do not update the pose distribution for that object. We update the mode probability of the object given a non-observation as before, except now $P(Y)$ is the probability that we did not observe the object. We used $P(Y|X) = 0.05$ and $P(Y|\overline{X}) = 0.95$ to model the probability that the non-observation was an error.

We continue updating the pose distributions and mode probabilities in the same way with all following segmented point clouds.

Algorithm 4 Update Object Psuedocode

```
1: procedure UPDATE–OBJECT(object, detected_pose)
2:     if detected_pose then
3:         update object's pose distribution
4:         update object's mode probability given observation
5:     else
6:         update object's mode probability given no observation
7:     end if
8: end procedure
```

4.2 Floor Plane and Wall Segments

The floor plane is represented as a set of points and a normal vector. We assume the floor cannot be moved, so there is no need to keep track of a mode probability for the floor in the scene. For each segmentation, we take the intersection of the set of points in the segmentation's floor plane and the set of points in our current belief state's floor plane.

We assume that wall observations are reliable so we also do not keep track of a mode probability or a pose distribution for walls. We represent a wall segment as the line segment produced from projecting the wall onto the floor plane. The wall can then be reconstructed by sweeping the line segment up along the z axis.

Unlike the floor, there can be multiple walls in a scene, so we keep track of a list of wall segments in our belief state. A wall plane is similarly represented as a set of points and a normal vector. For each segmentation, we check if any of the walls in the segmentation match the walls in our current belief state. We assume that walls cannot move, so if the two wall planes overlap enough and are coplanar, we consider them to both belong to the same wall.

If we find a match for the wall in the segmentation, we merge the two by updating the wall plane in our current belief state to contain the intersection of the two planes' points. We also update the normal vector to be the average of the two planes' normal vectors weighted by the ratio of the number of points in the plane to the total number of points in both planes. Otherwise, we add the wall in the segmentation to our current belief state.

We used a *threshold* of 0.1 and a *min_overlap_ratio* of 40%.

Algorithm 5 Update Walls Psuedocode

```
procedure UPDATE–WALLS(segmentation, state)
    for $wall1$ in $segmentation$ →walls do
        for $wall2$ in $state$ →walls do
            $\mathbf{v} = wall1$ →normal $\times$ $wall2$ →normal
            if $\|\mathbf{v}\| \leq threshold$ then
                if $wall1$ and $wall2$ overlap ratio $\geq min\_overlap\_ratio$ then
                    update $wall2$ to contain points in $wall1$
                end if
            end if
        end for
        if $wall1$ didn't match any walls in $state$ then
            add $wall1$ to $state$ →walls
        end if
    end for
end procedure
```

4.3 Stuff

"Stuff" represents unknown objects that the object detector could not recognize or objects that were not in the robot's current belief state. However, unlike the objects in the robot's current belief state, we cannot represent stuff with a pose distribution because we do not explicitly have a model for an unknown object in stuff since we do not necessarily know what object it is. Instead, we represent stuff as a history of clusters of points, as well as the probability of it being in the scene, or the mode probability.

We want to be able to ultimately combine all the observations of the unknown object in order to get a more complete representation of the unknown object. But we also want to keep all the observations separate from each other to make it easier to match the unknown object in point clouds at adjacent time steps. Therefore, we keep a history of the clusters of points associated with the unknown object.

Given a new segmentation, we try to match the stuff in the segmentation to the stuff in our current belief state. For the remainder of this section, we will refer to the former as "old stuff" and the latter as "new stuff" for convenience. An unknown object in old stuff is represented as a history of clusters of points, and an unknown object in new stuff is represented as a cluster of points. When comparing the two, for old stuff, we only consider the most recent cluster of points in the unknown object's history.

To find matches, we run ICP on the most recent cluster of points in the history of each unknown object in old stuff paired with every cluster of points for each unknown object in new stuff. Since ICP attempts to register one set of points (target) to another set of points (source) by transforming the source to match the target, we run it twice: once with old stuff as the target and new stuff as the source, and then again with new stuff as the target and old stuff as the source. For each target, we keep track of which source results in the lowest ICP score, which is calculated as the sum of squared distances from the source to the target.

In order to reduce the number of calls to ICP, we only consider pairs of old stuff and new stuff if they are relatively close to each other and their bounding boxes overlap. Since we assume that observations occur frequently, we assume that the robot has not moved a great deal in between sequential point clouds. We also assume that nothing in the scene will have moved or changed a great deal in between sequential point clouds. Therefore, something in the previous point cloud should still be relatively near the same area in the next point cloud.

We consider an unknown object in old stuff x and an unknown object in new stuff y to match if y gives the lowest ICP score out of all unknown objects in new stuff with x as a target, and if x gives the lowest ICP score out of unknown objects in old stuff with y as a target. Then we update x to include y in its history and update its mode probability given an observation as detailed in Section 4.1.

If we do not find a match for an unknown object of old stuff, we update its mode probability given a non-observation as detailed in Section 4.1. If the mode probability drops below a certain threshold, we remove it from our belief state. We used a minimum mode probability of 0.15.

If we do not find a match for an unknown object in new stuff, we add it to our current belief state.

4.4 Results

In the following results, a wall shown in the belief state visualizations is drawn using the points associated with it instead of sweeping its line segment along the z axis. The object pose drawn is the mean pose from that object's pose distribution, and stuff is drawn using the most recent cluster of points in its history.

In each belief state visualization, the most recently observed point cloud is drawn in cyan.

Algorithm 6 Update Stuff Psuedocode

```
1: procedure UPDATE–STUFF(old_stuff, new_stuff)
2:     $old\_to\_new \leftarrow$ for each element $x$ in $old\_stuff$, the element $y$ in $new\_stuff$ with
   the highest ICP($x$, $y$) score if bounding boxes for $x$ and $y$ overlap
3:     $new\_to\_old \leftarrow$ for each element $y$ in $new\_stuff$, the element $x$ in $old\_stuff$ with
   the highest ICP($y$, $x$) score if bounding boxes for $x$ and $y$ overlap
4:     for $x$ in $old\_stuff$ do
5:         $y = old\_to\_new[x]$
6:         $x' = new\_to\_old[y]$
7:         if $x == x'$ then
8:             add $y$ to $x$ $\rightarrow$history
9:             update $x$'s mode probability given observation
10:            remove $y$ from $new\_stuff$
11:        else
12:            update $x$'s mode probability given no observation
13:            if $x$'s mode probability $\leq min\_mode\_prob$ then
14:                remove $x$ from $old\_stuff$
15:            end if
16:        end if
17:    end for
18:
19:    // add all remaining clusters that were not matched
20:    for $y$ in $new\_stuff$ do
21:        add $y$ to $old\_stuff$
22:    end for
23: end procedure
```

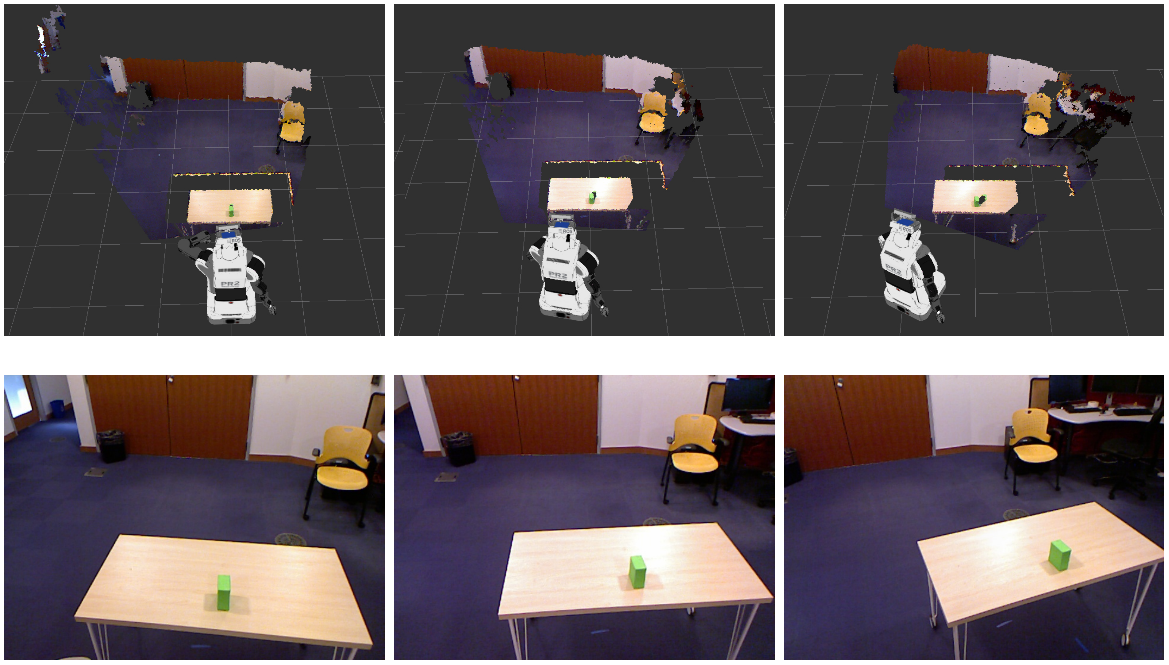
Red points correspond to the floor, pink points correspond to walls, and purple, green, and yellow points correspond to "stuff."

4.4.1 Scene 1

Scene 1 contains a table with a green box on top in the middle of the room. Figure 4-2 shows three observations taken of the scene. The robot first starts out at the middle of the table and then moves to the left side of the table. The robot initially starts with prior knowledge about the box and table objects in the scene.

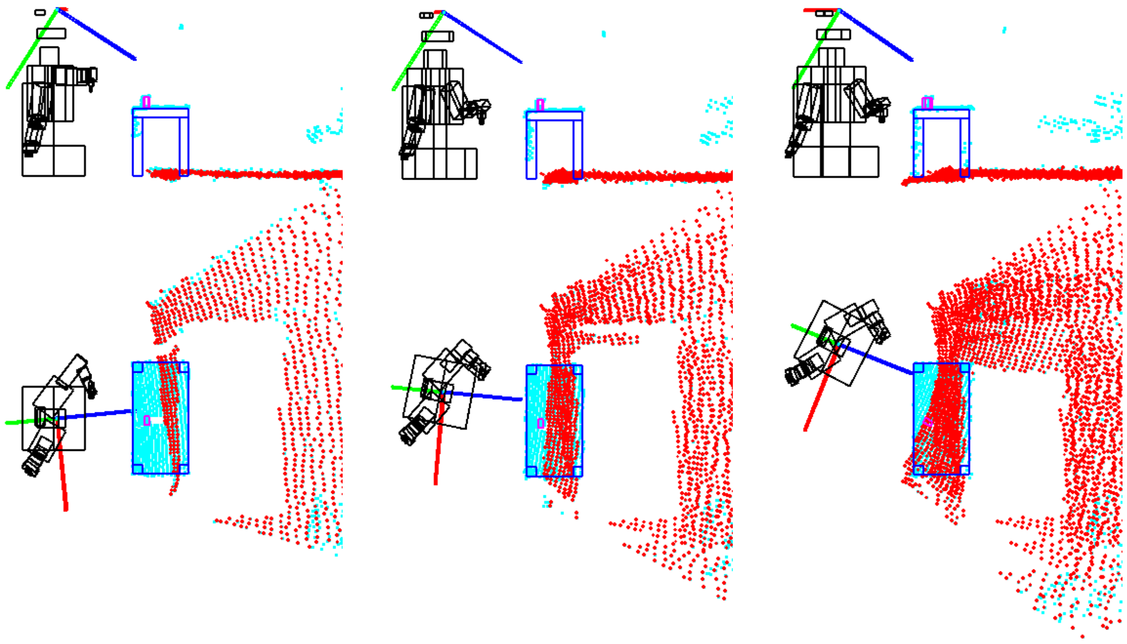
Figure 4-3 shows the robot's belief state after each of the three observations. The detected box and table poses are outlined in magenta and blue respectively.

As the robot moves and gets more observations, the number of floor points grows as more points are observed. The box and table are detected each time, so at the end of the third observation, the mode probability for both the box and table objects is 0.99. This indicates that the robot is very certain that those objects exist in the scene.



(a) Observation 1 (b) Observation 2 (c) Observation 3

Figure 4-2: Scene 1: Box on table.



(a) After observation 1 (b) After observation 2 (c) After observation 3

Figure 4-3: Belief state after multiple observations of scene 1.

4.4.2 Scene 2

Scene 2 again contains a table with a green box on top near a glass wall. Figure 4-8 shows four observations taken of the scene. The robot first starts out at the middle of the table, moves to the left side of the table, moves to the right side of the table, and then moves back to the middle of the table. The robot initially starts with prior knowledge about the box and table objects in the scene.

Figure 4-9 shows the robot's belief state after each of the four observations. As before, for each belief states, the most recently observed point cloud is drawn in cyan. The detected box and table poses are outlined in magenta and blue respectively. The red points correspond to the floor. Purple, green, and yellow points are "stuff."

As the robot moves and gets more observations, the number of floor points again grows as more points are observed. The box and table are detected each time, so at the end of the third observation, the mode probability for both the box and table objects is 0.99. This indicates that the robot is very certain that those objects exist in the scene.

Both clusters of purple points correspond to the pillars on the glass wall behind the table. The yellow points correspond to the cable hanging down by the left side of the table. The green points correspond to points on the table that weren't associated with the table during the point cloud segmentation step. This seems to be because the table pose is a little lower than the points.

The purple cluster corresponding to the pillar on the glass wall to the far right is only seen in the second point cloud. So after the next two observations, it's mode probability decreases as indicated by the lighter shades of purple in the last two belief states.

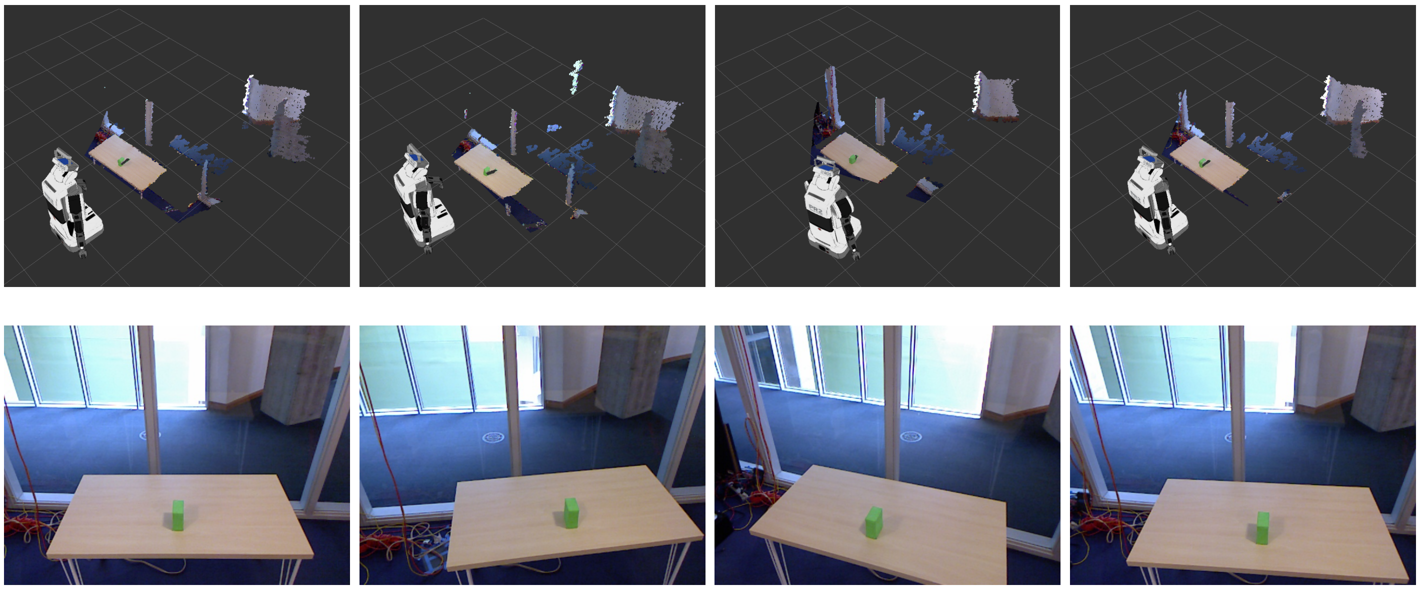

(a) Observation 1 (b) Observation 2 (c) Observation 3 (d) Observation 4

Figure 4-4: Scene 2: Box on table near glass wall.

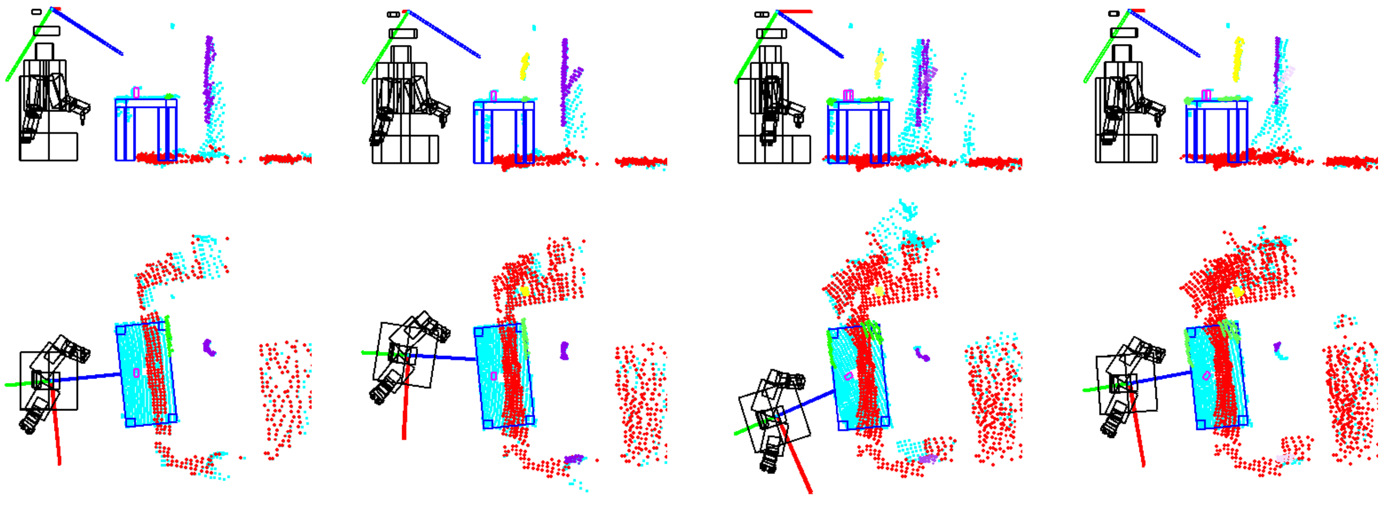

(a) After observation 1 (b) After observation 2 (c) After observation 3 (d) After observation 4

Figure 4-5: Belief state after multiple observations of scene 2. The purple points correspond to the pillars on the glass wall, the green corresponds to points on the table that were slightly outside of the detected table pose, and the yellow corresponds to the cable hanging down to the left of the table.

4.4.3 Scene 3

Scene 3 contains a table with a Cascade bottle on top. Figure 4-6 shows six observations taken of the scene. The Cascade bottle first starts out in the middle of the table, but it is later moved to the right side of the table. The robot first starts out at the middle of the table, moves towards the left of the table, moves towards the right of the table, and then moves back to the middle of the table. The robot initially starts with prior knowledge of the table object in the scene.

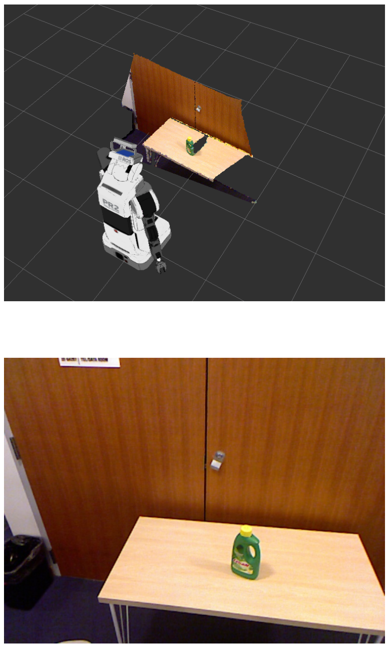
Figure 4-7 shows the robot's belief state after each of the six observations. The detected table pose is outlined in blue, and the red points correspond to the floor.

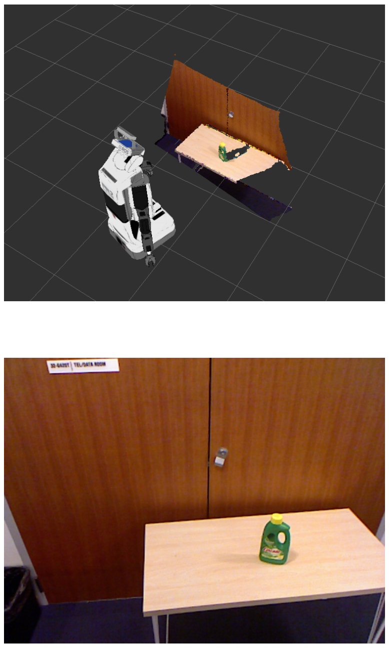
As the robot moves and gets more observations, the number of floor points again grows as more points are observed. The table is detected each time, so at the end of the last observation, the mode probability for the table object is 0.99, indicating that the robot is very certain that that object exists in the scene.

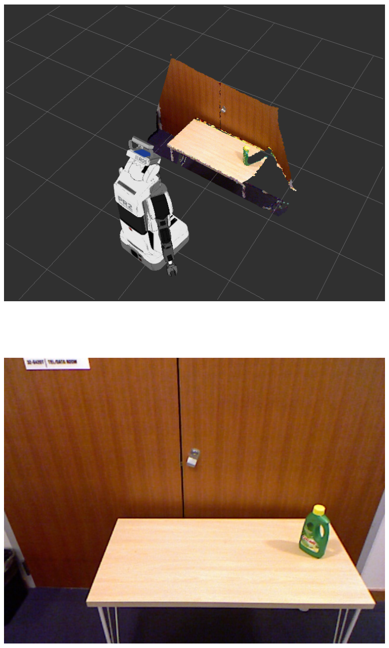
The purple cluster in the first two belief states corresponds to the Cascade bottle in the middle of the table. In between the second and third observations, the bottle is then moved to the right side of the table. This is the yellow cluster shown in the third belief state in Figure 4-7 (c). The purple cluster is not seen again in the later observations, so its mode probability continually decreases after each observation, indicated by the lighter shades of purple. Finally, its mode probability gets so low that it is removed from the belief state as seen in Figure 4-7 (f).

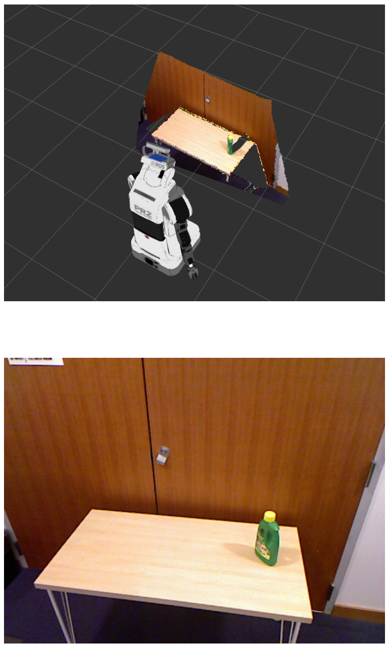
The robot continues to see the yellow cluster in the following observations so its mode probability is still 0.99 in the last belief state. The yellow cluster in the third and fourth belief states (Figure 4-7 (c) and (d)) is the same as the green cluster in the fifth and sixth belief states (Figure 4-7 (e) and (f)). The colors just got switched due to the way we visualize the belief state.

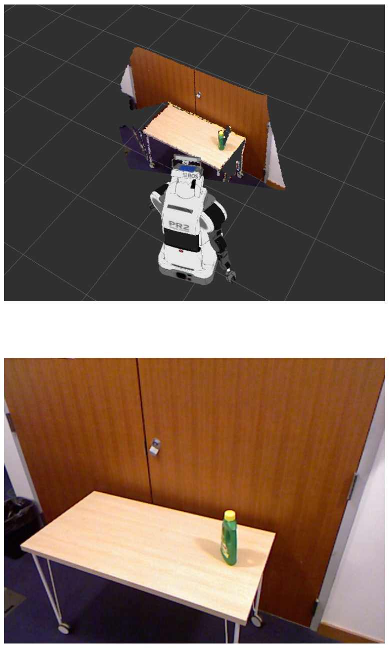
The green cluster in the second belief state (Figure 4-7 (b)) corresponds to points on the table that were not associated with the table object during the point cloud segmentation step. This seems to be because the table pose is slightly lower than those particular points. This does not happen for the later point clouds though, so this cluster also decreases in mode probability until it is removed from the belief state as well.


(a) Observation 1


(b) Observation 2


(c) Observation 3


(d) Observation 4


(e) Observation 5

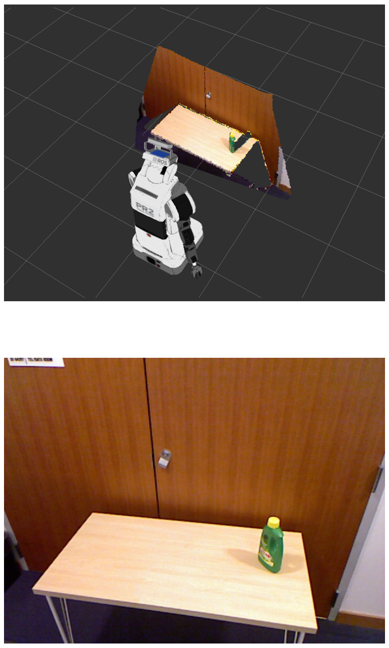
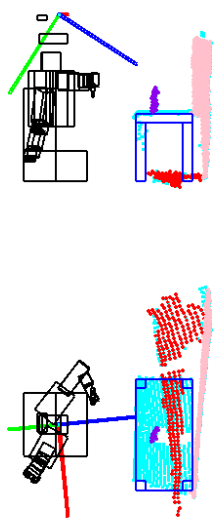
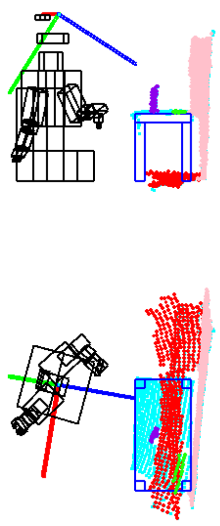
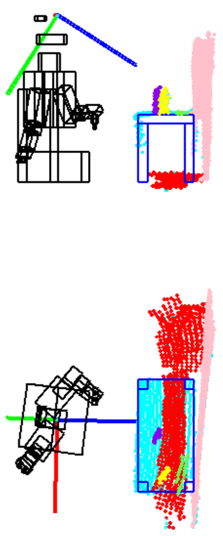
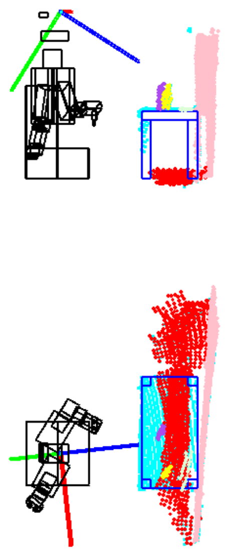

(f) Observation 6

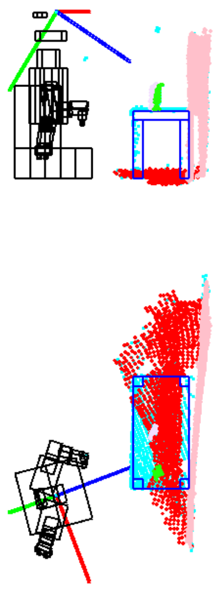
Figure 4-6: Scene 3: Table with Cascade bottle.


(a) After observation 1


(b) After observation 2


(c) After observation 3


(d) After observation 4


(e) After observation 5

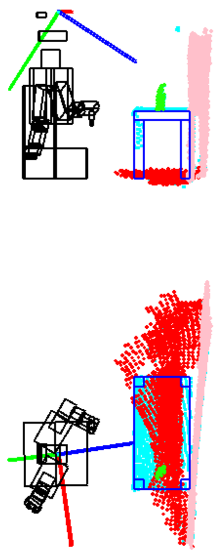

(f) After observation 6

Figure 4-7: Belief state after multiple observations of scene 3. For (a)-(d), the purple points correspond to the original Cascade bottle, the yellow corresponds to the moved Cascade bottle, and the green corresponds to points on the table that were slightly outside of the detected table pose. For (e)-(f), the purple corresponds to the original Cascade bottle and the green corresponds to the moved Cascade bottle.

4.4.4 Scene 4

Scene 4 contains a table with both a green box and a Cascade bottle on top. Figure 4-8 shows five observations taken of the scene. The robot first starts out at the middle of the table and then moves towards the right of the table. The robot initially starts with prior knowledge of the box and table objects in the scene.

Figure 4-9 shows the robot's belief state after each of the five observations. The detected box and table poses are outlined in magenta and blue respectively.

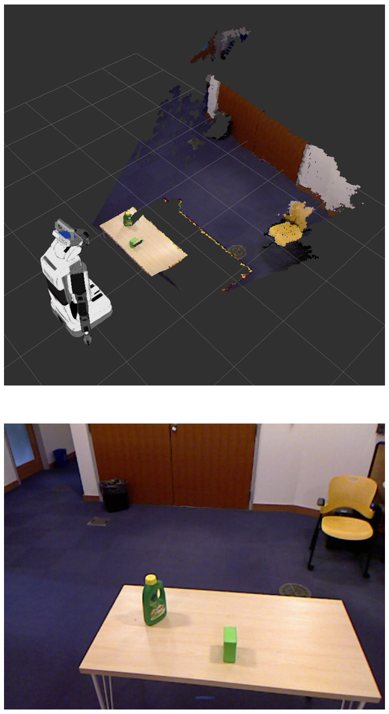
As the robot moves and gets more observations, the number of floor points grows as more points are observed. The box and table are detected each time, so at the end of the third observation, the mode probability for both the box and table objects is 0.99. This indicates that the robot is very certain that those objects exist in the scene.

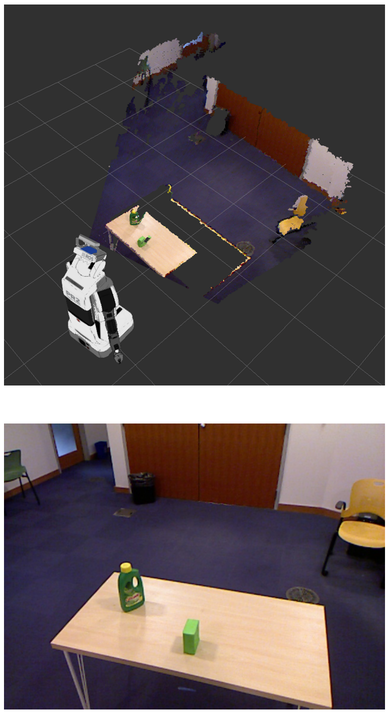
The purple cluster corresponds to the Cascade bottle. The robot observes it each time, and we match the clusters from adjacent time steps to each other, so the mode probability for the purple cluster at the end of the observations is 0.99. For some of the observations, some points from the table are clustered together with the Cascade bottle, so we would need to combine all the observations in the purple cluster's history in order to get a good representation of what it actually looks like.

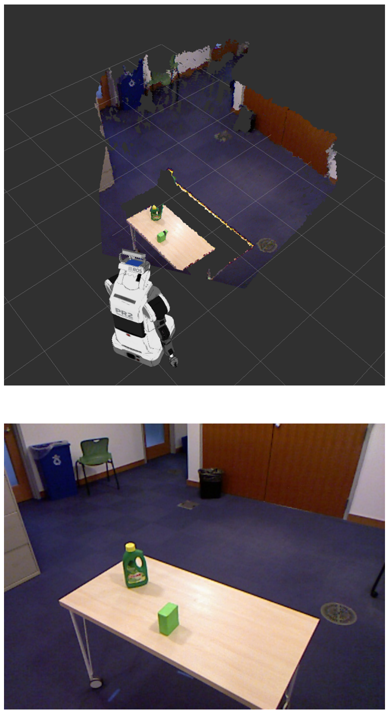
The yellow cluster along the side of the table that gets added to the third belief state (Figure 4-9 (c)) corresponds to points along that side of the table that were not aligned properly with the detected table pose during the segmentation step for that point cloud. This does not happen for the later point clouds though, so this cluster decreases in mode probability as more observations are made, indicated by the lighter shades of yellow. At the end of the observations, the yellow cluster has a mode probability of 0.21.

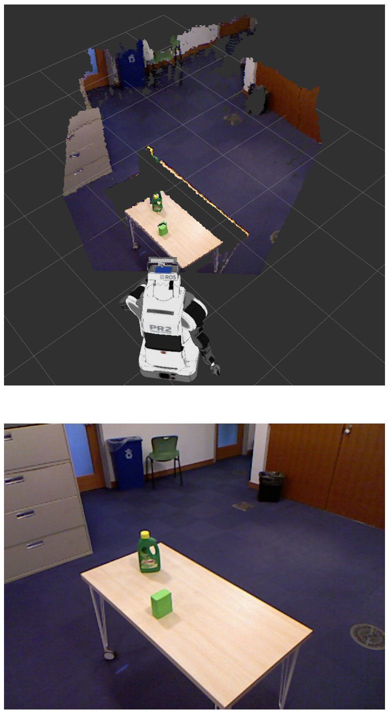
Similarly, the green cluster along the floor that gets added to the second belief state (Figure 4-9 (b)) corresponds to some points along the ground that were not added to the floor plane during the segmentation step for that point cloud. This does not happen for the later point clouds though, so this cluster also decreases in mode probability as more observations are made, indicated by the lighter shades of green, until it is removed from the belief state.

As the robot moves to the right side of the table, it is able to see a cabinet on the left side of the table. It interprets this as a wall when enough of it is visible, as shown in the last belief state in Figure 4-9 (e).


(a) Observation 1


(b) Observation 2


(c) Observation 3


(d) Observation 4

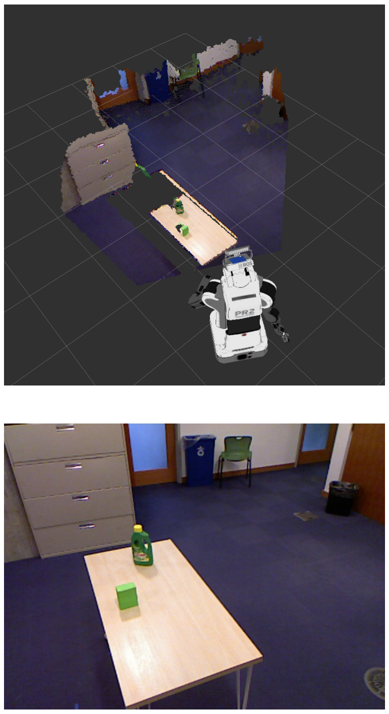
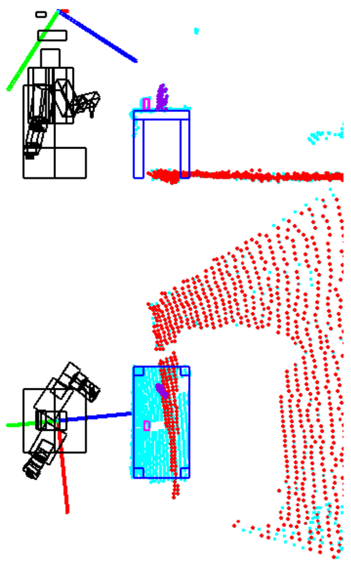
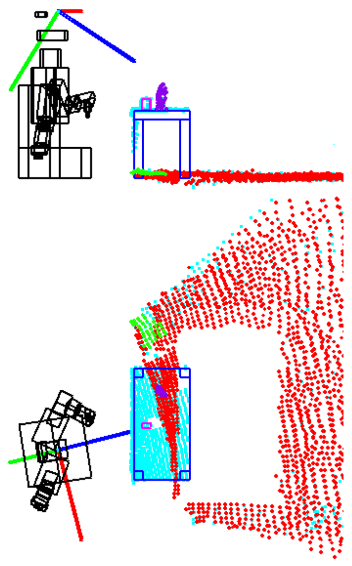
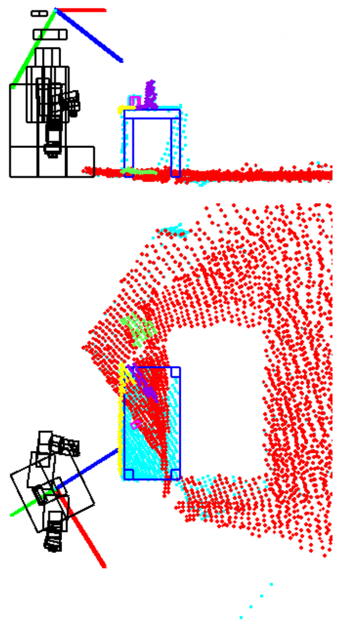
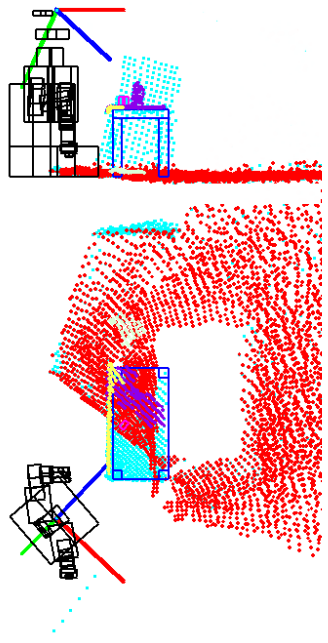

(e) Observation 5

Figure 4-8: Scene 4: Table with box and Cascade bottle.


(a) After observation 1


(b) After observation 2


(c) After observation 3


(d) After observation 4

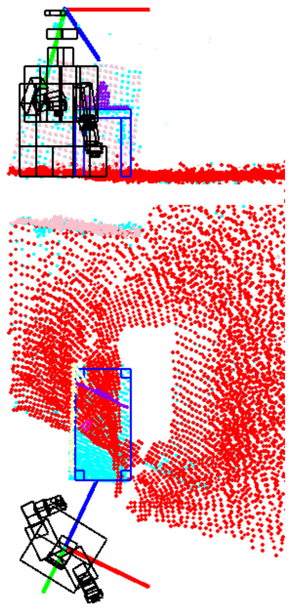

(e) After observation 5

Figure 4-9: Belief state after multiple observations of scene 4. The purple points correspond to the Cascade bottle, the yellow corresponds to points on the table that were slightly outside of the detected table pose, and the green corresponds to points on the ground that were not associated with the floor plane.

Chapter 5

Conclusions and Future Work

In this thesis, we presented a system that estimated a semantic belief state for a scene given multiple observations of the scene. We first segmented the observed point clouds and then incrementally updates the belief state with each segmented point cloud. Our main focus was to develop an end to end system, and our main goal for the future is to improve upon it.

There are a couple problems, some of which are already evident from the results shown in Chapters 3 and 4. When object detections are even slightly incorrect, this can throw the segmentation off and result in "stuff" being associated with parts of the object.

For example, this happens when the detected table pose is slightly misaligned and then the edge of the table is added to the belief state as stuff. This is a seemingly rather harmless mistake though, since it can be useful for the robot to know that there is something where the actual table edge is so it can avoid running into the edge. It is also often corrected by later observations and then removed from the belief state when its mode probability gets low enough.

Updating the mode probability every time an object or an instance of stuff is not seen could, however, be problematic. This means that the robot will essentially forget about or assume an object, known or unknown, does not exist if it does not see it for awhile. This can be bad in certain situations, such as if the robot continues to take observations while it is not moving and an object is not detected from that viewpoint. It may be better to instead try to predict whether an object should be visible in the point cloud given the robot's current position and only update its mode probability if it is visible.

Future work would involve making our system more robust to cases such as these and more reliable in detecting "stuff." Currently, our clustering algorithm is likely to fail in very cluttered scenes. It will also most likely not be able to identify very small items on a table, such as a pen, since we are using a minimum cluster size in order to avoid adding insignificantly small clusters to our belief state.

It could be useful to incorporate color data into the point cloud segmentation step. It may also be useful to eventually add "stuff" to the list of objects in our belief state if we can identify what the stuff is. We did not prioritize speed or runtime while developing our system so there may be ways to optimize it and be able to segment a point cloud and update our belief state more efficiently.

Bibliography

[1] A. Aldoma, F. Tombari, L. Di Stefano, and M. Vincze. A global hypotheses verification method for 3d object recognition. *IEEE International Conference on on Robotics and Automation*, 2013.

[2] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:239–256, 1992.

[3] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *Int. J. Comput. Vision*, 59(2), 2004.

[4] R. Finman, L. Paull, and J. J. Leonard. Toward Object-based Place Recognition in Dense RGB-D Maps. *ICRA workshop on visual place recognition in changing environments*, 2015.

[5] R. Finman, T. Whelan, M. Kaess, and J. J. Leonard. Efficient incremental map segmentation in dense RGB-D maps. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5488–5494, 2014.

[6] J. Glover and S. Popovic. Bingham Procrustean Alignment for Object Detection in Clutter. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[7] A. Johnson. *Spin-Images: A Representation for 3-D Surface Matching.* PhD thesis, August 1997.

[8] L. P. Kaelbling and T. Lozano-Perez. Hierarchical Task and Motion Planning in the Now. *IEEE Conference on Robotics and Automation*, 2011.

[9] L. P. Kaelbling and T. Lozano-Perez. Integrated Task and Motion Planning in the Now. *Technical Report 2012-018, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology*, 2012.

[10] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[11] Point Cloud Library. Estimating Surface Normals in a PointCloud.

[12] Point Cloud Library. Euclidean Cluster Extraction.

[13] Point Cloud Library. Plane model segmentation.

[14] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[15] P. Norvig and S. J. Russell. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2009.

[16] S. Popovic. *Parallelized Two-Stage Object Detection in Cluttered RGB-D Scenes.* PhD thesis, June 2013.

[17] R. B. Rusu, N. Blodow, and M. Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, May 12-17*, 2009.

[18] J. Tang, S. Miller, A. Singh, and P. Abbeel. A Textured Object Recognition Pipeline for Color and Depth Image Data. *In the proceedings of the International Conference on Robotics and Automation (ICRA)*, 2012.