Generating and Learning from 3D Models of Objects through Interactions

Kathryn Baldauf, Aylish Wrench, Kiana Alcala
{$kathrynbaldauf, aylish.wrench, kiana.alcala$}@utexas.edu

Abstract

In this paper we implement an approach that will allow for the creation of 3D models. Being able to autonomously build or recognize a 3D model of an object would assist in human-robot interaction and developmental robotics. Currently the robot is only able to construct a view of an object from one angle, which limits the robots capabilities. Manipulation is necessary when trying to observe and classify an object. Therefore, in order to create the model, we have the robotic arm pick up the object, rotate it, and save multiple point clouds that will eventually be aligned to create the 3D model. We integrate filtering techniques such as downsampling, and point cloud alignment methods such as Sample Consensus Initial Alignment (SAC-IA) and Iterative Closest Point (ICP). Although the ultimate goal is to have a full model of an object, the focus of this paper is implementing and evaluating methods of alignment as they specifically relate to the segway base robots of the Building Wide Intelligence (BWI) lab at the University of Texas at Austin. In the future we would also like to store information about the object so that the robot will eventually be able to detect similar or previously viewed objects.

Introduction

The ability to observe and recognize objects is useful for a robot as it performs actions and navigates through different environments. As of right now, the Building Wide Intelligence (BWI) segway robot with the Kinova Mico arm is only able to view an object from one angle. The robot cannot see an object from all sides without moving either the object or itself, which limits the robots usable information about the object. Information that could help in the perception and detection of an object would be missing. If a 3D model of the object was available to the robot, it could use that model to fill in the missing information. Therefore, implementing code that will allow the robot to construct 3D models of objects will advance the abilities of the current BWI system.

Related Work

The main goal for this project was to get the robot to use motor actions to create a 3D model of an object. There have been many research efforts that attempt an object model construction as well as a variety of available methods for approaching this. The basis of this project stems from a curiosity of how humans interact with objects and how we could implement this behavior on the robot and its arm. In order to have the robot accurately represent interaction with objects, we first need to understand how children learn through object experimentation.

The theory that children learn by interacting with objects is well known in the field of psychology. Jean Piaget (1952) discusses children learning through sensorimotor activities in his book, *The Origins of Intelligence in Children*. Piaget divides a childs development from birth to the age of two into six sub-stages. According to Piaget, a child begins to interact with objects in the environment in the third sub-stage which occurs sometime after the child reaches four months old. In this sub-stage, the child performs an action with their body, usually accidentally, which has a visible effect on its environment. The child not only notices this change, but then tries to recreate it.

A child's interest in objects in its environments continues into the next sub-stage, which begins around eight or nine months. However, instead of reacting to changes in its environment, the child actively begins experimenting with objects. By performing different actions on an object such as moving or shaking, the child gains a better understanding of the objects properties and abilities. Piagets observations demonstrate the importance of sensorimotor activities in human development. We believe that incorporating both visual and motor functions to gain information about an object could prove to be effective in the development of robotic intelligence. By picking up an object, the robot can "learn" more than it would solely with visual data. Combining visual and motor data would produce a view of areas of the object that were obscured previously. More importantly, the robot would be able to accomplish all of this with less assistance from humans. After gaining a better understanding of how children learn through the observation and manipulation of objects, we formulated a methodology that allows the robotic arm to mimic the same actions.

One research paper by Krainin et al. (2011) presents various other research methods involved in manipulator and object-tracking for in-hand 3D modeling. The paper discusses visual recognition strategies, object modeling tech-

niques, and alignment processes. The authors of the paper also outline a process to successfully detect and segment out point cloud data corresponding to the arm's hand, which is a necessity when constructing an in-hand 3D model. The information discussed in the paper allowed us to decide on the best procedure for creating 3D models with our own robot.

Dorai et al. (1998) focus on registering and integrating multiple views of objects from range data in order to construct geometric and descriptive 3D models of objects. Although the researchers use a laser range scanner in order to obtain their data, their goals and much of the technical approach are similar to the ones we chose to implement in our project. The paper mentions Iterative Closest Point (ICP), an algorithm which our work implements and evaluates. The researchers also applied multiple preprocessing steps to remove the noise in the input images. The steps included the removal of isolated pixels and the use of a median filtering technique. Although we do not use this specific filtering technique, the work of the authors inspired us to implement filtering methods that we felt would best suit the robotic arm, such as a color filter and downsampling.

Another example of 3D model construction using filtering techniques to simplify challenges, in this case manipulator removal, is in the In-Hand Scanner for Small Objects (Slzle, M., PCL Developers). While this method requires a human to manipulate the object, the challenge of removing point cloud data corresponding to the manipulator is the same. They require that the object be of a different color from the skin tone of the human manipulating the object or that the human wear a glove in order to implement the use of a color filter. This color filter determines a threshold for use in distinguishing between the object and the hand. We initially implement a similar filter and evaluate its usefulness in the Evaluation section of this paper.

The paper that was the most influential in our project is work done by Krainin, Curless, and Fox (2011). They created a system which allows a robot to create a 3D surface model by moving an object into its view. In their system, the robot uses a volumetric information driven Next Best View algorithm to determine how it must pick up and manipulate the object to achieve the view that will give it the most information. Once the robot is finished with this view, it places the object back on the table. The robot cleans up the generated image by removing any points located on or near its hand. It then repeats the next best view algorithm to determine how it should re-grasp and move the object. This allows the robot to make the model more complete by cleaning up holes created by its manipulator during the initial grasp or by allowing it to see areas which it could not in the previous view. This work represents much of our end goal.

Methodology

The basic overview of the 3D model creation process is as follows. The robot grasps an object and lifts it into view of the stationary camera. The Xtion camera retrieves point cloud data of the full scene: the table, the hand, and the object. The object is segmented from the rest of the scene and saved as a separate point cloud. Once this point cloud is saved, the robot rotates its end joint, or hand, 20 degrees and takes a second point cloud image. This process is repeated until the object has been rotated a full 360 degrees. All of the saved point clouds are then combined to create a single point cloud which shows a complete view of the object. This process has been implemented as a ROS action. Unlike ROS services, ROS actions can be interrupted and intermediate results can be returned. This allows the user to obtain a partial model if an error occurs during the process. The source code for the 3D model action can be found in the make_model_as.cpp file found in the model_3d_object package of our code base.

Lifting Object into View

The first step in our process of creating a 3D model is to have the robot pick up the object and verify that the object has been lifted. To do this, the Make Model action uses existing code from the segbot_arm_perception and segbot_arm_manipulation packages in the BWI code base (Utexas-bwi/segbot_arm). In order to ensure a successful grasp, we assume the desired object is the only item on the table.

Once the robot has confirmed that the object has been successfully grasped and lifted, it moves its arm into the position shown in Figure 1. Through experimentation, we found this to be the optimal position for the arm as it allows the Xtion camera to see as much of the object as possible. Since the robot only has a single, stationary camera, it is impossible to get a view which shows the entire object. To solve this, the robot rotates its hand 20 degrees until it has rotated a full 360 degrees, saving a point cloud after each rotation. This produces a list of point clouds which together depict every angle of our targeted object.

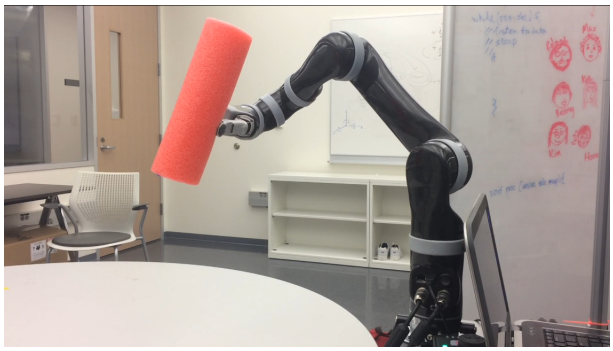

Figure 1: Scene of the object in the robot's hand

Object Detection

As previously mentioned, the initial point cloud taken by the Xtion camera contains everything in the cameras view. This includes the object, the Kinova arm, and the table. Before the point cloud can be used to form a correct 3D model, everything other than the object itself should be removed. This is the task of the Object Detection service we developed. To accomplish this goal, the service uses a variety of functions from the Point Cloud Library (PCL) (Rusu and

Cousins 2011). The service request takes in the center of the hand. This point is then used in computing the upper limit of a z filter. The z filter is implemented as a PCL PassThrough filter, described in section 3.D of Miknis et al. (2015). The filter removes anything from the point cloud other than that which lies between the camera and approximately five inches past the Kinova arm. After applying the z filter, the resulting point cloud is downsampled to remove noise. This has the added bonus of reducing the cloud size, making future computations simpler. For downsampling, we used PCLs VoxelGrid filter, described in section 3.C of Miknis et al. (2015). An unfortunate side effect of downsampling is that invalid, or NaN, values can be introduced into the point cloud. Therefore, the point cloud data is passed through a simple PCL function to remove the NaN values before proceeding on to the next step.

At this point, the resulting point cloud still contains the robots hand in addition to the object. Our initial approach to eliminate the hand data was to use a basic color filter. The color filter removes any points in the point cloud with RGB values corresponding to grey or black. In order for the color filter to work successfully, we assume that the Kinova arm is the only grey/black colored object in the scene. As a result, this color filter is not effective for manipulation of objects that are a similar color to the arm. As discussed in our results, this color filter proved to be fairly ineffective even for non-grey/black objects. Therefore, in the final version of our Object Detection service, the color filter has been disabled.

Once all filters have been applied to the original point cloud, the final cloud should now represent data for just the object (and possibly the robots hand). This cloud is converted from a PCL format to a ROS message format and returned as the result of the service. The cloud is then published to rViz for debugging purposes.

Alignment and Registration

Each cloud returned from the Object Detection service is saved in a vector. When the robot has completed a full rotation of its hand, saving point clouds every 20 degrees, the next step is to combine the point clouds into a single image. The process of combining, also known as registering, occurs in another ROS service: the Align Service. One of the advantages of implementing the alignment/combination process as a service is that it can be used independently of the Make Model action. For example, for testing purposes we created a program to save a series of point clouds from the Object Detection service. We could then use a separate program to read in these files and combine them. Both of these programs can be found in our GitHub repository.

In the Alignment Service, the first point cloud in the vector is designated as the original point cloud. This point cloud is used as the base of the model when combining the following point clouds. Before a point cloud can be added to the base, it must first be aligned. Alignment is the process of rotating and transforming one point cloud so that points which correspond with those of another point cloud overlap. There are many different existing alignment algorithms. In our work, we implemented both Sample Consensus-Initial Alignment (SAC-IA) and Iterative Closest Point.

The SAC-IA implementation used by PCL is based on the algorithm described by Rusa, Blodow, and Beetz (2009). To use SAC-IA, the features, in our case FPFH features, for a source and target cloud must be computed. A number of points from the target cloud are then selected. For each of these points, a histogram is created from its features. From there, a list of points from the source cloud that have similar histograms is generated. One of these points is chosen at random to correspond to the target point. Once corresponding points have been determined for each of the target points, a transformation for the target point cloud is calculated. Due to the randomness involved in the algorithm, SAC-IA does not usually produce exact results. However, the algorithm functions best for clouds which are a substantial euclidean distance or rotation apart. SAC-IA is therefore suitable to create an initial alignment but another algorithm must be combined with it to achieve the most accurate results.

To improve the alignment results from using SAC-IA, we implemented the ICP alignment from PCL. There are many variants of the ICP algorithm but the basic algorithm, as described by Rusinkiewicz and Levoy (2001), is as follows. An initial guess for the transformation between source and target point clouds is generated. A subset of points in one or both of the point clouds is chosen. These points are then matched to points in the other cloud. The pairs are then weighted in some manner. Based on previously chosen criteria, certain pairs are rejected. An error metric is then assigned based on the remaining pairs. A new transformation is created which attempts to minimize the error metric. This process is repeated until specified criteria is met. In the case of the PCL implementation, the process ends when any of the following is met: a max number of iterations is reached, the difference between the last transformation and the current transformation is less than a given threshold, or the sum of Euclidean squared errors is less than a given threshold (Rusu and Dixon 2016). Since features for the clouds do not need to be computed, ICP is faster than SAC-IA. For clouds that are closer together, ICP also produces more accurate results than SAC-IA. In our work, we use ICP to fine-tune the results from the SAC-IA algorithm in hopes of producing the most accurate results possible.

After applying both alignments, the transformed cloud and target cloud are combined to become the new ”original” model for the next round. We continue the process for each of the point clouds in the vector. When all point clouds have been added together, the resulting 3D model is transformed into a ROS message that can be returned to the Make Model action where it is published for viewing or debugging in rViz.

Evaluation

For each action or service created, a corresponding program was written to validate the functions and in some cases, results. The agile grasping, lift verification, and tabletop perception service were not evaluated for correctness as they are pre-tested by their original creators. Additionally, we created two files for saving individual point clouds, one for objects on the table and one for objects in the hand of the robot, to speed up testing. These files did not require validation as

they are reproductions of previous code or documentation within the PCL library.

Object Detection Service

Within this ROS service, we tested the distance of the z PassThrough filter, the use of the VoxelGrid filter, and lastly the color filter. As previously mentioned, the z filter is given a range such that any point cloud data outside of this range is removed. In order to evaluate our use of this, we first place a simple object in the arms hand in the view of the camera. We do not hardcode a set value for the full z filter as this would significantly limit the use of the service as the arm would not be able to be moved to other positions. However, we do hardcode the padding around the arm's location in which to expect the object's full cloud. This follows from the knowledge that the arm has limited capabilities when interacting with heavier objects and the assumption that it will mainly be used with objects typically hand held by a human, such as a box of food or a spray bottle. This value is set as 0.125 meters, which is approximately five inches. After experimenting with values, we determined that this is an appropriate amount given our preconditions, as any smaller value would significantly reduce the number of objects available to be interacted with and any larger value could present issues by allowing more of the arm or other objects to be present in the results.

Table 1: Effects of Downsampling

Number of Clouds Aligned	Downsampled	Not Downsampled
Four Clouds	12.02s	14.45s
Three Clouds	4s	5.05s

The downsampling method we implemented, as previously mentioned, is the PCL VoxelGrid filter. We set the leaf size to be approximately 0.005 meters as this allows for a good amount of information to be used for feature detection and alignment while still reducing the size of the point cloud significantly. Since this code is pre-existing, we instead evaluated the usefulness of including it. In Table 1, we show the average time of three trials of the completed alignment process using point clouds of objects on the table with no color filtering. From this data we conclude that not only does downsampling decrease computation time, but also the addition of more point clouds negatively affects the computation time in a significant way.

Since using a color filter to eliminate certain objects can both limit the scope of objects that can be interacted with and introduce holes in the resulting cloud, we decided not to implement this in our final code. However, we tested the results as there is potential to combine this functionality with extended code in the future to produce better results. Figures 2 and 3 show the comparison between a point cloud where the color filter was not applied and one where the filter was

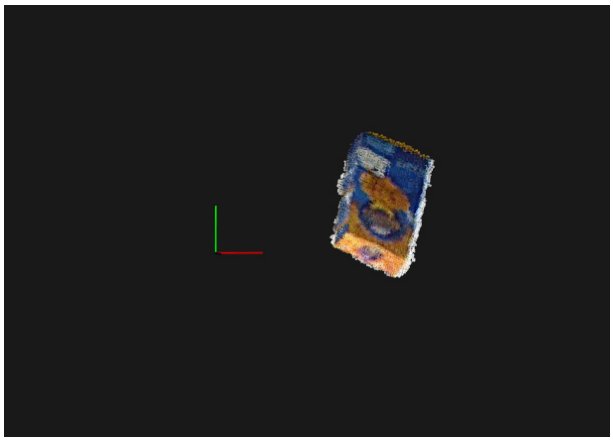

Figure 2: Point cloud data for a box with no color filtering

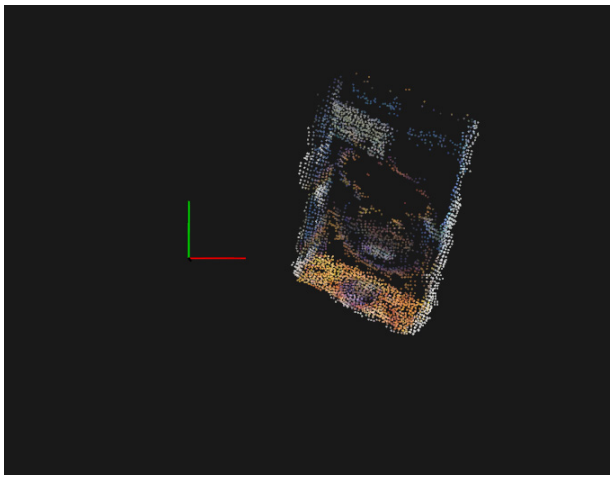

Figure 3: Point cloud data for a box with the color filtering

applied. Figure 4 shows the actual box. By comparing these figures, one can see that many unintentional points were subtracted from the point cloud in Figure 3. This is due partly to the nature of the color filter, as it will subtract any point that is outside of the given RGB range with no knowledge of the surrounding points or the overall object. Additionally, the Xtion camera introduces noise and false reflections that can cause points to appear differently and thus be filtered out. Figure 5 shows an example of the full alignment process with the arm data filtered out using the color filter. As shown, the resultant clouds still include some of the arm point cloud data. While this leftover data could potentially be removed using other methods, one can see that the majority of the object's information was also segmented out, and thus an incomplete model would always be produced.

Tables 2 and 3 show the fitness scores of two objects on the table aligned with the ICP algorithm alone. Fitness scores closer to zero indicate a perceived better alignment. Subsequently, the fitness score seemed to improve when the color filter was applied. As discussed above for Figures 2 and 3, Figure 3 has significant portions of important

Figure 4: Real image of the box

information about the object missing despite having a higher fitness score. This result can be attributed to the fact that the use of the color filter eliminates crucial point data, thus increasing the chance of a false alignment.

Table 2: ICP Alone, Object On the Table, No Color Filter			
Object	Avg. 1st Pair Fitness Score	Avg. 2nd Pair Fitness Score	Avg. 3rd Pair Fitness Score
Box	8.533e-05	4.050e-04	3.110e-04
Crayon	6.224e-05	2.490e-05	1.333e-05

Table 3: ICP Alone, Object On the Table, with Color Filter			
Object	Avg. 1st Pair Fitness Score	Avg. 2nd Pair Fitness Score	Avg. 3rd Pair Fitness Score
Box	7.595e-05	7.5066e-05	2.325e-04
Crayon	1.139e-04	1.912e-04	4.699e-05

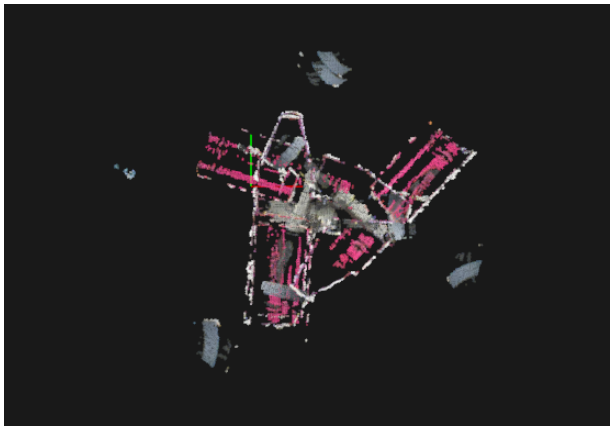

Figure 5: Alignment in hand using the color filter

Alignment Service

Within the alignment service, we tested the use of ICP, SAC-IA, and the corresponding parameters for each. SAC-IA is well known to be appropriate for large distances or turns. In our current implementation of model construction, this is not the case. However as discussed in detail in the future work section, we still implemented and tested it appropriately as it will be applicable in extended work. We compare the results using ICP alone as well as using SAC-IA followed by the ICP. Appropriate parameters were tested and altered if necessary.

In order to test the functionality of the ICP algorithm, we first fed in two point clouds, with the second one having some small rotational difference from the first. The first cloud was fed in as the ”InputSource” and the latter as the ”InputTarget”. Upon completion of the alignment algorithm, we combined the original cloud with the resulting cloud returned from the align function call of ICP. While our convergence scores were relatively good, the resulting model seemed to look as if the clouds had not been transformed at all. Upon further investigation, we discovered this was a result of changed documentation that led to confusion for this algorithm in the PCL library. When calling the align function in ICP, the resultant cloud is the transformation of the InputSource cloud onto the InputTarget cloud. Thus, instead of adding the original cloud with the resultant cloud, the resultant cloud and target cloud should be combined and stored for the next round of alignment. After fixing this, the ICP algorithm performed as expected and converged for each alignment. As shown earlier in Table 2 and 3, the ICP fitness score was relatively low both with and without color filtering when the object was on the table. Additionally, we determined that the algorithm will converge in under fifty iterations given point cloud data from our situation, as the data will always be of the same object with small rotations and translations. Other changes in parameters did not greatly impact the results.

Table 4: Testing Minimum Sample Distance	
Min Sample Distance	Resulting Fitness Score
0.001	3.722e-04
0.005	3.722e-04
0.01	3.722e-04
0.35	2.578e-03
0.40	2.670e-03
0.45	1.739e-04
0.5	1.876e-04
0.75	9.349e-04
1	4.007e-04

The SAC-IA presented many challenges to implement and test. Similar to the ICP algorithm, the resultant cloud contained the transformed InputSource. However, unlike the ICP algorithm, certain parameters greatly impacted the algorithm's fitness score and ability to converge. Table 4 shows the results of our experimentation with the Minimum Sample Distance parameter of the SAC-IA algorithm. This parameter determines the smallest distance between the points selected in the target cloud. We initially implemented the algorithm using a minimum sample distance of 0.005 meters, the size of our VoxelGrid leaf size. After some experimentation, as shown in Table 4, we discovered that a much larger value, one closer to 0.45, produced better fitness scores at face value. However, this is a result of the sample points being too far away and thus reducing how strict the algorithm behaves. Other parameters were found to have little or no effect on the resulting fitness score and correctness of alignment.

Table 5: SAC-IA and ICP, Object in Hand, No Color Filter				
Object	Avg. 1st Pair SAC-IA Fitness	Avg. 1st Pair ICP/Final Fitness Score	Avg. 2nd Pair SAC-IA Fitness Score	Avg. 2nd Pair ICP/Final Fitness Score
Box	4.728e-03	4.795e-03	6.852e-03	1.679e-02
Crayon	3.192e-03	2.501e-03	2.628e-03	2.754e-03

Table 6: SAC-IA and ICP, Object On the Table, No Color Filter				
Object	Avg. 1st Pair SAC-IA Fitness	Avg. 1st Pair ICP/Final Fitness Score	Avg. 2nd Pair SAC-IA Fitness Score	Avg. 2nd Pair ICP/Final Fitness Score
Box	1.721e-04	1.837e-04	3.382e-04	5.861e-04
Crayon	2.211e-04	1.881e-04	4.077e-04	4.341e-04

We then evaluated using both SAC-IA and ICP alignment on the resultant model. Table 6 shows the results of alignment after SAC-IA followed by ICP with an object on the table and no color filtering. Compared to the results of Table 2 with just the ICP alone, there is a noticeable deterioration in fitness score when the SAC-IA algorithm is called first. We attribute this to the random nature previously discussed of the SAC-IA algorithm when used with clouds that have such small difference in rotation or translation. Figure 6 shows an example resultant model aligned using this method on a crayon shaped object on the table and Figure 7 shows an image of the actual object. Comparing the two, it is easy to see that the resultant cloud has one image of the crayon combined with a flipped image, creating an image with two "crayon tops". This is an example of an issue that can arise using the SAC-IA algorithm on models that have little change as the model ended up being completely wrong. This could also be a result of using an object of this nature, with minimal feature differences around the circumference. However, this issue occurred even on more rigid objects, such as the box shown in Figure 8.

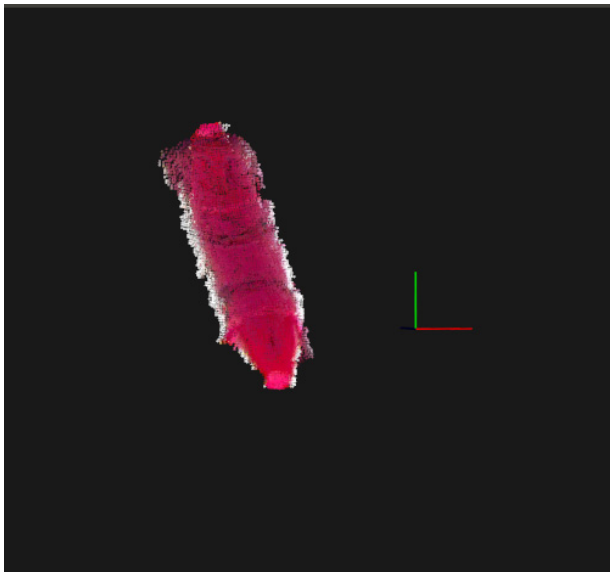

Figure 6: Point cloud aligned using SAC-IA followed by ICP

Table 5 shows results of the same method but in the arm's hand. The fitness scores as compared to other alignments mentioned in this paper are particularly bad. This is due to the fact that the arm has not been eliminated from the point cloud of the object and thus is skewing the information and causing false or bad alignments. The resulting cloud computed in the arm's hand does not reflect the actual object at all. We discuss how we intend to handle this problem in the future works section of this paper.

Figure 7: Image of the crayon

Further, we found that after each alignment and registration of clouds, the fitness score diminishes regardless of the method of alignment. This can be seen in Tables 2, 3, 5, and 6. We hypothesize that this is a result of slight errors when combining clouds after each alignment. If the features and normals do not match up entirely, the partial point cloud's data will be slightly disrupted when combined. As this process continues, the reliability of the cloud lessens along with the fitness score and correctness of the model. An example of this is seen in Figure 8, where the box sides aligned back to back instead of as intended. We outline our plan to handle this issue in the future work section of this paper.

Future Work

The most limiting challenge we faced within this project was determining a method to remove point cloud data corresponding to the arm before alignment. As we discussed, without this removal the model construction was greatly hin-

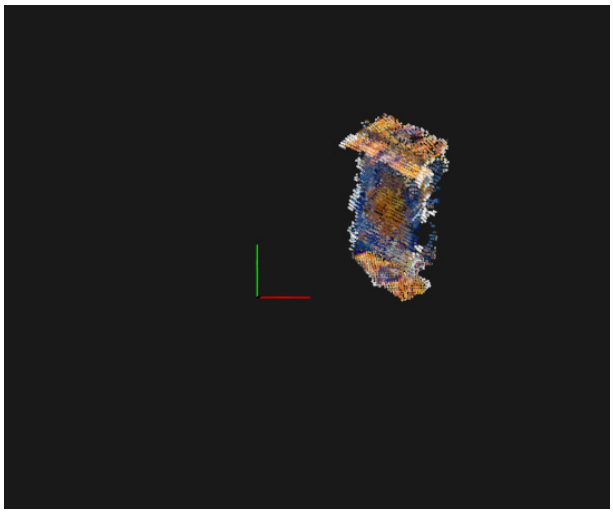

Figure 8: SAC-IA and ICP alignment of the box

dered. However, basic filtering techniques, such as the color filtering, proved to not be useful in our situation. A simple approach to eliminating this issue would be creating a virtual, predetermined sized box around the location of the manipulator and subtracting any points within this region from the perceived region of interest. Another potential path for this is in the previously mentioned related work by Krainin et al (2011). In the paper, the authors use a Kalman filter, which uses statistical measurements over time to determine estimates of unknown variables, to track and distinguish between the object and the manipulator. They combine this with a variation of the ICP algorithm to gradually create an updated model of both the object and the hand as it moves in the environment. We hope to research and implement these techniques to evaluate the increase in alignment and usefulness as it relates to our situation.

We additionally plan to develop a way to fill in holes created around areas of manipulator subtraction. In order to do this, we will follow a similar approach as mentioned previously the related work of Krainin, Curless, and Fox (2011). Similar to their work, we plan to replace the object on the table and repeat our final alignment process. This repositioning of the object could create issues further down the road in the case that the table is crowded, causing confusion in grasping the correct object. We hope, with the use of a partial model, to track the object even when it is not in the arm's hand to reduce the impact of the hole filling process.

To address the issue of long computation times and to better use the SAC-IA algorithm, we hope to combine our work with a next best view, or similar, algorithm. A next best view algorithm would be beneficial in this scenario as it would allow for the minimal amount of angles, and thus point cloud data, of the object to be used in the alignment process. While this would significantly reduce overall computation time, it would require the use of the SAC-IA algorithm as the turns would no longer be a set 20 degrees and could differ greatly between each view.

Further, in creating features for use in the various align-

ment algorithms, we used FPFH features. These features only take into account the curvature and geometry of an object without color information. However, color patterns and changes are an important characteristic of objects. We plan to adjust our final alignment process to use features that contain both geometry and color.

Once a successful model has been created, we hope to use the model construction process as well as the model itself for various applications down the road. This could include object recognition and machine learning applications. By providing the robots in the BWI lab a database of 3D objects that are commonly viewed or interacted with in the environment, object recognition can then be implemented to allow the robot to potentially identify predetermined targets, such as a bag of chips or a package, to retrieve. Following this, using a machine learning algorithm, we hope to train the robot to determine the best way to manipulate objects within this database. For example, train the robot to grasp and manipulate a water bottle in a different way than that of grasping a fragile bag of chips.

References

Piaget, J. 1952. *The Origins of Intelligence in Children (5th ed., Vol. 8)* New York: International Universities Press.

Krainin, M., Henry, P., Ren, X., Fox, D. 2011. Manipulator and Object Tracking for In-Hand 3D Object Modeling. *International Journal of Robotics Research*, 30(11), 1311-1327.

Dorai, C., Wang, G., Jain, A. K., Mercer, C. 1998. Registration and integration of multiple object views for 3D model construction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1), 83-89.

Krainin, M., Curless, B., Fox, D. 2011. Autonomous generation of complete 3D object models using next best view manipulation planning. *Robotics and Automation (ICRA), 2011 IEEE International Conference. IEEE.*, 5031-5037.

Utexas-bwi/segbot_arm. n.d.. Retrieved December 12, 2016, from *https://github.com/utexas-bwi/segbot_arm/*

Rusu, R. B., Cousins, S. 2011. 3d is here: Point cloud library (pcl). *In Robotics and Automation (ICRA), 2011 IEEE International Conference. IEEE.*, 1-4.

Miknis, M., Davies, R., Plassmann, P., Ware, A. 2015. Near real-time point cloud processing using the PCL. *International Conference on Systems, Signals and Image Processing (IWSSIP). IEEE.*, 153-156.

Rusu, R. B., Blodow, N., Beetz, M. 2009. Fast Point Feature Histograms (FPFH) for 3D registration. *IEEE International Conference on Robotics and Automation*, 3212-3217.

Rusinkiewicz, S., Levoy, M. 2001. Efficient variants of the ICP algorithm. *In Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, 145-152.

Rusu, R. B., Dixon, M. 2016. Point Cloud Library (PCL): Pcl::IterativeClosestPoint Class Template Reference. Retrieved December 12, 2016, from *http://docs.pointclouds.org/trunk/classpcl_1_1_iterative_closest_point.html*

Slzle, M., PCL Developers. In-hand Scanner for Small Objects. Retrieved December 13, 2016, from *http://pointclouds.org/documentation/tutorials/in_hand_scanner.php*

Acknowledgments

We would like to thank Peter Stone, Jivko Sinapov, and the Freshman Research Initiative Program at the University of Texas at Austin for giving us the resources and guidance necessary to pursue this project.

Resources

Link to our presentation: `https://goo.gl/dVzg1d`

Link to video footage: `https://youtu.be/eetU4vX8_F4`

Link to our github repository: `https://github.com/katiewasnothere/model_3d_object`