Automated Three-Dimensional Object Scanning

Arnav Jain, Ashay Lokhande, and Brahma Pavse

Computer Science Department

The University of Texas at Austin

arnav_jain@utexas.edu, ashaylok@utexas.edu, brahmasp@utexas.edu

Abstract

This paper addresses the idea of optimizing a procedure through which a robot with a camera could obtain the most visual data while exerting the least energy. This procedure does not just measure the most efficient viewset for energy cost, but also ensures a minimum threshold for successfully obtaining meaningful data. In this paper, we identify a few key aspects of visualization that underlie this functionality. The actual procedure is identified by simulating robotic vision on Point Cloud Dataset files and estimating the cost for this vision using real-life energy cost metrics. The Point Cloud Dataset files are analyzed by finding the accuracy of different camera viewpoints relative to the object's center. Each viewpoint is considered independently, and then viewpoints are strung together to create view sets. These view sets represent real-life instances in which all of an object cannot be viewed from a single perspective. We present the results of using this procedure, its success within our simulation, and future implementations of this procedure.

Introduction[1]

The Building Wide Intelligence project Segway robots are designed to autonomously roam the Gates-Dell Complex main building. In providing many different services, the robots are required to be adept at:

- Autonomously navigating and localizing in the diverse areas of the Gates-Dell Complex.
- Understanding the surroundings in order to better service humans and navigate the building.
- Understanding how to interact with objects while navigating or giving virtual tours.

Although the current robots are able to navigate different floors by detecting obstacles and following maps, they are not good at interacting with their surroundings. As robots that tour the building, they should be able to interact with objects and have knowledge of the objects that surround them. Currently, these Segway robots have a Segway base, many sensors, and a camera that allow them to interact with their surroundings through navigation of different environments. With a built-in arm, one of the robots can correctly ascertain other traits of objects. However, the Segway robots lack the ability to view an object and construct a three-dimensional rendering of that object. This rendering requires multiple different views to be stitched together, posing the question of how to best obtain these different viewpoints. The best algorithm to obtain the best views (the greatest percentage of the object viewed) can be approached in a number of ways [Faugeras, 1993].

Figure 1: Segway robots with Segway bases and cameras. Other sensors located in the base are used to help the robot navigate the Gates-Dell Complex.

Related Works

Cost minimization algorithms all generally require a cost algorithm which takes into account different metrics

[1] Thanks to Jivko Sinapov for mentoring us, and to the mentors for providing guidance and maintaining the robots.

[Elegbede, Chu, Adjallah, Yalaoui, 2003]. These metrics have to be specific to the use case. In this case, that means minimizing cost relative to viewing the object. Overall, in object recognition and reconstruction, there are many different methods utilized to best reconstruct the object into a three-dimensional model.

Military Robots

One such example of object reconstruction is military applications of robotics. In terms of national security, unmanned military vehicles are the future for warfare. With regards to national defense, many war situations are found in hazardous zones, where resources are scarce and the terrain is unforgiving. Being unmanned, the robot has to navigate the terrain autonomously. Even robots are remote-controlled, such as the majority of drones, require some fall-back mechanism through which they could readjust for the terrain [Shaker, Wise, 1987].

Figure 2: Unmanned, military scouting robot that has an arm and a camera to navigate its surroundings.

Furthermore, in an emergency situation, the robot needs to have the ability to correctly and efficiently identify objects around it and determine how to use the objects. Especially with a lack of resources, many unmanned robots could be utilized to scout out rough terrains and determine strategic resources in the surroundings. Similarly, with regards to unmanned naval and space-based robots, it is pertinent for robots to have this visual awareness of their surroundings. Figure 2 above shows how most of these robots have cameras and arms to interact with and understand their environment. However, since most of these military robots are small, they have small batteries and need to efficiently consume that energy to be effective. The robot in Figure 2 follows energy usage minimization principles to be effective as an unmanned vehicle, especially in hazardous regions [Bhat, Meenakshi, 2016].

Normal Aligned Radial Feature (NARF)

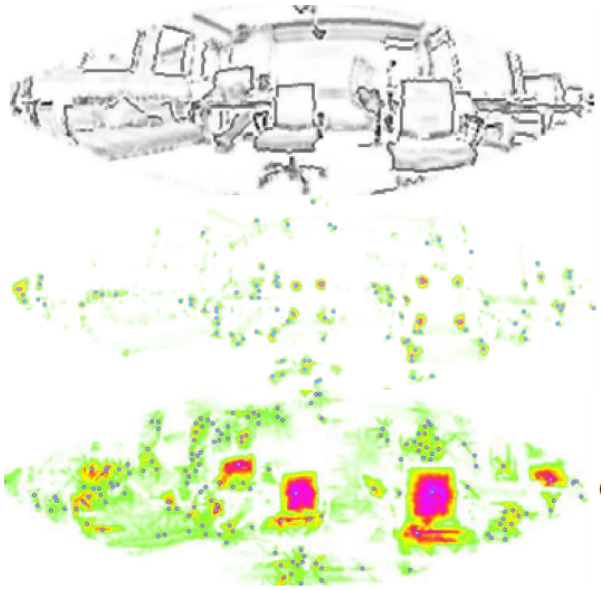
A relevant methodology for filtering images and checking for similarities in images can be found in the article: “Point feature extraction on 3D range scans taking into account object boundaries” by Steder, Rusu, Konolige, and Burgard.

The focus of this paper is to present a keypoint extraction method on three-dimensional point cloud data for both object recognition and pose identification. The authors' goal was to find the similarities between two different images in order to understand which parts of the images overlap. This allows one to determine what has already been scanned and help distinguish between new data from data that the robot has already taken into account. In the experiment, the paper discusses using single range scans, as obtained with three-dimensional laser-range finders, in which the data is incomplete and dependent on a viewpoint. In the paper they present a normal aligned radial feature (NARF), "a novel interest point extraction method together with a feature descriptor for points in three-dimensional range data". The NARF relies on detecting the borders of an object and having objects placed in locations where the surfaces are stable, as seen in Figure 3 [Steder, Rusu, Konolige, Burgard, 2011] .

The interest point detection relies on three ideas to accurately detect interest points in a three-dimensional image. It must take borders and surface structure into account, select points that are reliably detectable from different angles, and points that are in positions that provide stable areas for normal estimation. The goals for the development of the NARF Descriptor was to identify the difference in occupied and free space, to make the descriptor robust in handling different interest point positions, and enable them to extract a unique local coordinate frame at a single point. Once this procedure and its calculations were explained, the paper introduces two different experiments that test object matching and stability in recognizing interest points from different distances and angles [Steder, Rusu, Konolige, Burgard, 2011].

The above methodology, combined with some metric for threshold energy or cost factor, is an efficient and optimized manner of determining the ideal set that can be potentially used to reconstruct the object in question. This is a potential area of expansion and adaptation that can be made to enhance the efficiency and ensure further accuracy of results [Steder, Rusu, Konolige, Burgard, 2011].

Figure 3: An example of filtering and removing overlap of an image using the NARF algorithm.



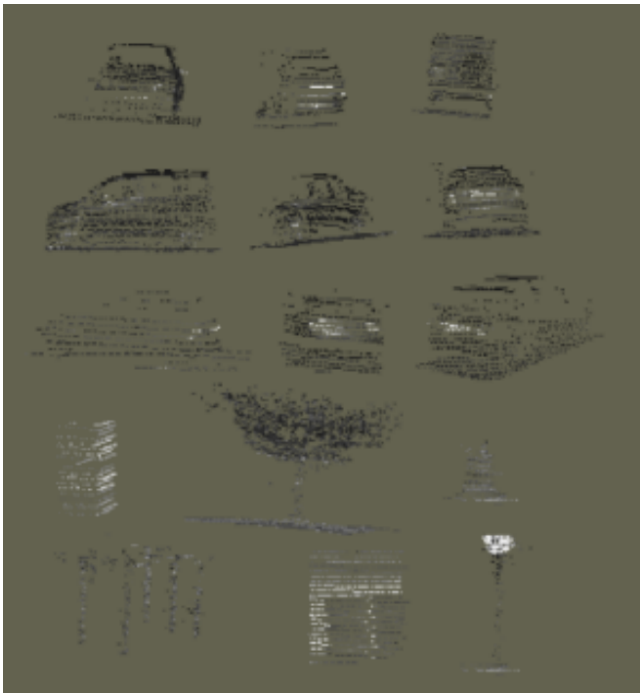
Segmentation and Analysis of Point Clouds

Another relevant paper describes how to approach large objects which can range in 10^5 measurements in terms of the number of points and how to accurately stitch together images. Since robots need to take multiple views to accurately get a full three-dimensional rendering of any given object, it is important to consider this method of stitching together point clouds. This article is called "Real-time object classification in three-dimensional point clouds using point feature histograms" by Himmelsbach, Leuttel and Wuensche.

In this paper, the motivation behind the given task was to assist a robot to navigate urban traffic as well as off-road environments. In order to achieve this goal, the authors combined two-dimensional and three-dimensional image processing techniques. Two-dimensional data was used for segmentation of point clouds into objects and three-dimensional data was used as raw point clouds to classify objects. The two-dimensional data was stitched together using information of relative position with regards to the entire point cloud [Himmelsbach, Leuttel, Wuensche, 2009].

Beyond utilizing both forms of image processing, the paper goes over fast object feature extraction, a unique perspective to consider with military robots and object recognition. Since most fast objects are hard to capture, the techniques used to recognize and map them can be helpful for robots to quickly obtain information about their environment without having to analyze the object fully. The fast objects are captured by histograms over point features. This paper is unique in its ability to implement a way to incorporate two-dimensional and three-dimensional methodologies to solve the problem. For future purposes, this paper helps account for very large point clouds [Himmelsbach, Leuttel, Wuensche, 2009].

Figure 4: Some hand-labeled examples of point clouds used for training a vehicle classifier. Positive examples (top 3 rows) and negative examples (bottom 2 rows).



For other implementations of imaging techniques, the paper discusses vehicle classification theories that can be used to classify specific vehicles. By using data sets of vehicles to train a vehicle classifier, the researchers hoped to train the visualization of vehicles to correctly identify the vehicle recognized. Figure 4 above gives examples of data points within a potential training dataset. This classification is especially relevant for military robots that need accurate classification algorithms to translate point cloud datasets into clumps of points that define objects. Visualization of objects lays the framework for object classification, upon which robots can eventually take specific actions. The training works by rewarding correct classifications and negating incorrect classifications. In this

manner, the classifier generalizes rules for classifying vehicles or other objects [Himmelsbach, Leuttel, Wuensche, 2009].

Methodology

Overall our goal was to determine a general trend in the creation of the most optimal set of views that performs an accurate three-dimensional scan of an object. The purpose of our project may be derived from the possibility of applying these findings to three-dimensional scans of unknown objects. Our project largely deals with simulations in rViz with objects that we already have full three-dimensional scans of. This allows us to test our findings with the data we used to derive our results from.

In an effort to make our work as compatible and expandable as possible, we divided our solution into four key C++ files. These four portions were each made to complete a single overarching task that could be combined with the other portions of the project while still being easily adaptable.

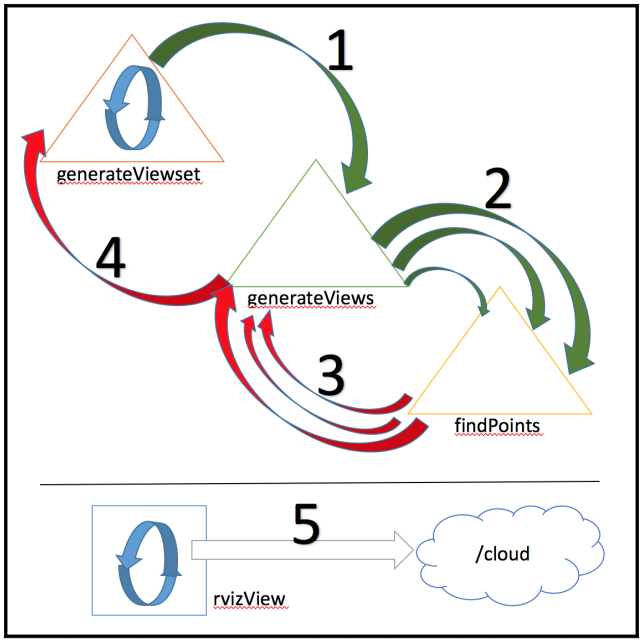
The four sections of code we have include two separate nodes that must run simultaneously. The first node is rvizView which we use to publish the point cloud dataset we need as a topic for the rest of the project to use. The second process includes three separate C++ files. This process contains our findPoints, generateViews, and generateViewset files. Together these files take the data published in rvizView, create viewpoints around the object, determine how much of the object can be seen from these views. Finally, generateViewset determines the most efficient collection of views in a collection we call a viewset..

Rviz Viewer

The first process that our project completes was developed as the rvizView.cpp file. In this ROS node, we convert a specified Point Cloud Data (.pcd) file into a message of type sensor_msgs::PointCloud2. This resulting message is then published to a topic we titled "/cloud" in order to be referenced by other parts of our project as well as to display on the rViz simulator. Besides publishing the dataset to a topic, we could have passed this PointCloud2 object into the functions we will be using it in. However, the main purpose behind setting up the topic was to make it easier for future work to send isolated object data in the form of a topic containing PointCloud2 data. It was also easier to visualize the entire object in rViz by publishing the .pcd to the "/cloud" topic [Kammerl, Woodall, 2016].

The second process that completes the remainder of our project is comprised of the three remaining files. Only one of these files, generateViewset, contains a call back and runs as a ROS node. The other two files contain code for function calls which are used in generateViewset. See Figure 5 below for a visual depiction of how these files interact.

Figure 5: A diagram displaying the behaviors and interactions between portions of our project. The blue arrows are used to display which parts of our code contain callbacks and run as ROS nodes. The green arrows are used to display function calls. The red arrows display return statements. Finally the gray arrows indicate that a file is publishing information. 1: generateViewset calls generateViews once in order to create the sets of views around the object. 2: generateViews calls findPoints once for each view created. 3: findPoints returns information about that view in relation to the object. 4: generateViews sends each view and the information obtained in step three back to generateViewset. 5: rvizView publishes the point cloud dataset to the "/cloud" topic.



Generating Viewsets

This process beings with generateViewset. Though this function beings the process, it is also where it ends. It requires both generateViews and findPoints to feed it the data it requires to determine the most efficient collection of views. Thus, we will describe the actual data being used by this file after discussing the files overall goal and function. The process begins with the assumption that generateViews and findPoints have correctly fed the information to generateViewset in the form of a two-dimensional vector. In this two-dimensional vector, each element in the innermost vector contains a struct with the Pose for the view, a filtered cloud containing only points that can be seen from that view, a boolean map which indicates which points of the original object can be seen, and a double that indicates the percent of the object seen from that view.

Before searching through the views, we create five initial viewset_object objects that are initialized with the starting view of the object at five different angles. This portion of the code is easily expandable and adaptable to incorporate an assortment of camera angle positions. The viewset_object contains a vector of the views in this viewset, a boolean map that is a combination of all the views in the viewset, and a double holding the total percentage of points that can be seen, based on the combined boolean map. These viewset_objects will be filled with the best combination of views based on that starting position. Then these viewset_objects can be compared to determine the most efficient set out of these optimized sets.

The procedure we used to determine the most efficient set of views primarily focuses on the boolean map associated with each view. Each view is compared to the current boolean map of the viewset_object to determine which view would add the most information to the current set of views. This process occurs indefinitely until the viewset_object's total percentage viewed is equal to or greater than a defined threshold value. This threshold value is the degree of accuracy desired for the three-dimensional scan.

Generating Views

The file generateViews.cpp defines a function that is called by generateViewset to create views, which we represent as Pose objects, around the object. These views are created in a manner that best fits the Building Wide Intelligence project Segway robots in the Gates Dell Complex at The University of Texas at Austin. These robots currently operate with cameras at a fixed height. However, it may still change its orientation. Due to the fact that the camera may not move its relative position on the robot, we create the views at a fixed height. The height is stored as a class constant and can be easily adapted for different heights or even robots that may change the height of their camera views. To give the best combination of angle and position for different camera views, we chose to create the views in the form of a circle around the object. If we were to expand our code to work with robots that may change the height of their cameras, such as quadcopters, we would generate views as a hemisphere around the object. Regardless, we create the positions of the camera views such that each is ten degrees around the the circle from the next closest view. The circle's size is determined by multiplying the length or width of the object, whichever is larger, by 1.5. This value is used as the radius of the circle that we create the views on. The orientation of each camera is adjusted so that it is always looking at the center of the object. For each view created, generateViews makes a call to findPoints. The information from calling findPoints on each view is returned as a struct. The generateViews function calls returns a two-dimensional vector filled with this findPoints data. The elements of the outer vector contain a vector that corresponds to a different view position. Each element of this vector contains the data for each separate view at the position. The difference between the views within the outer vector is each views orientation. The two-dimensional vector is returned to generateViewset at the end of generateViews.

Finding Visible Points

The final component of our project is the findPoints.cpp file. The file consists of a primary method called findPoints, which is called from the above generateViews.cpp file. When this method is called, a PointCloud of the object in question and a given viewpoint are passed in as arguments to the method. At a high level, the method traverses the points of the PointCloud and generates a filtered cloud based on our filtering algorithm. In addition, the method calculates the percentage of points viewed from that viewpoint and generates a boolean structure containing data on which of the points in the point cloud are visible. All this data is returned to generateViews through a C++ struct.

Our filtering algorithm is vital for simulating robotic vision within rViz. The algorithm involves traversing the points of the PointCloud. For each point, the method computes the slope between that point and the given viewpoint (passed as the argument). After the slope is calculated, the point is added to the map with the slope as a key and the point as the corresponding value. For now, this point is considered visible; thus, in the boolean structure, this point is entered as the key and its corresponding value as 1 (to designate true). As the method traverses the cloud, it may find a point that has the same slope as a point that is already in the map. If this is the case, the method must find the point closer to the viewpoint. Using Eucledian distance formulas, the method determines which point is closer and accordingly update the map. The closer point is considered visible since it blocks the view of the farther point. In the case that the new point is relevant (closer) and the old point is not visible, the method updates the boolean structure and marks the old point as 0 (false - not viewed) and the new point as 1 (true - viewed). Ideally, at the end of this traversal, we have a map of unique slopes to the closest viewed points for a given viewpoint.

Given a camera view, findPoints can determine all the points that a specific camera viewpoint can see. All these points are added to the PointCloud object. This object is published in generateViewset if it is one of the most optimal views. The final struct returned consists of: a geometry::Pose object of the camera view, a PointCloud object of the filtered cloud, a double of the percentage of the object viewed, and the boolean map of visible and hidden points.

Results

Once we had unit tested our code to ensure it was accurately producing the intended data, we applied our algorithm and code on different PCD data sets. As discussed earlier, running the code on PCD files enables us to check our working model in simulation. Further results involve applying the algorithm on the robot itself.

The general process by which we conduct experiments and collect data is consistent for all PCD data files. Our rvizView.cpp node reads a PCD file of the object we wish to test (spray bottle, cup, office etc) and passes this object as a PointCloud through our algorithm. At the conclusion of the program's runtime, our algorithm outputs the following relevant data:

- The number of points in the PCD.
- The percentage of the object viewed in the optimal viewset.
- The number of views in the optimal viewset.
- The cost metric calculated for the robot to reach each view.

Table 1: Data collected from a few PCD files of varying sizes and with different percentage viewed thresholds.

Item	No. of points	Percent Viewed	No. of Views	Energy Cost (J)	Distance Cost
Cup	2500	97.24	2	85.40876	3.317
Cup	2500	99.7	3	130.9583	5.086
Cup	2500	99.88	4	131.8596	5.121
Spray Bottle	2512	98.22	2	105.2095	4.086
Spray Bottle	2512	99.92	3	110.5910	4.295
Spray Bottle	2512	100	4	139.3782	5.413
Hand	5080	96.98	2	131.0356	5.089
Hand	5080	99.53	3	140.0992	5.441
Hand	5080	100	5	152.8191	5.935
Office	307000	32.6	1	0	0
Office	307000	100	2	206.5311	8.021

From the the data on the left, we are able to extract more useful information such as how does the cost vary as we increase the size of our object. This is a relevant metric that is required to eliminate wasteful work and optimize robot visualization in military applications. Collecting data from many object files, we graphed the relationship.

Figure 6: Graph that represents the Energy Cost (in Joules) vs the Size of the Object.

As we can see from above, as expected, the energy increases as the size of the object increases. It is interesting to note that while we expected a positive linear trend, the relationship is not actually linear. Instead, it appears to be logarithmic. This is confirmed via a best-fit curve. However, given our limited data size, this relationship may just be coincidental.

In addition to the numerical figures acquired, below are pictures of the the rViz simulations.

Figure 7: Robot visualizing the cup with 2 views and a 97.24% accuracy.

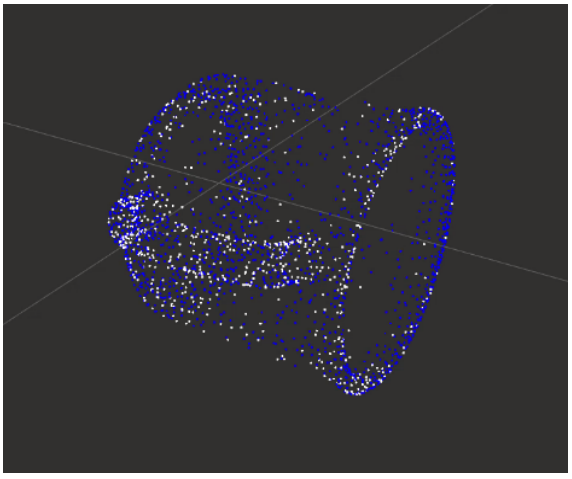


Figure 8: Robot visualizing the hand gesture with 2 views and a 96.28% accuracy.

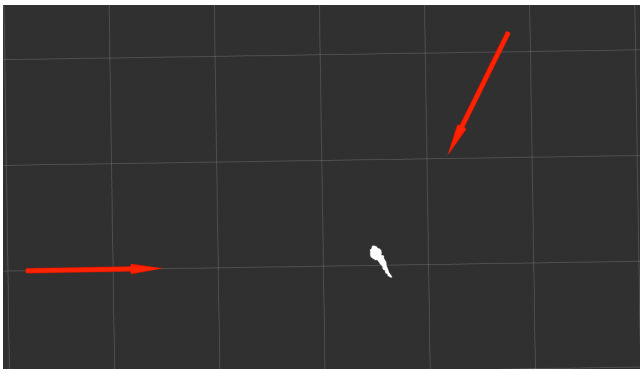
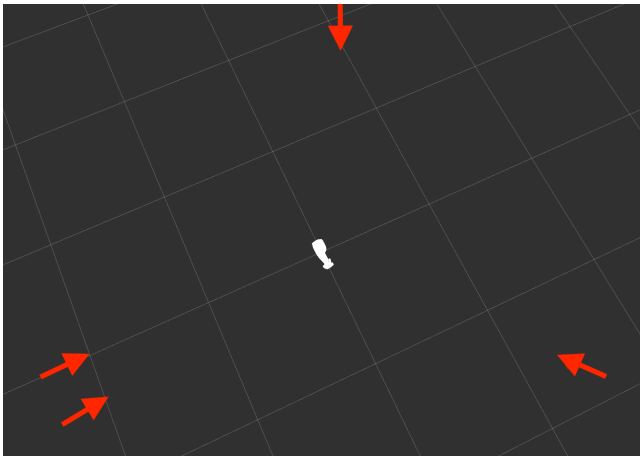


Figure 9: Robot visualizing the spray bottle with 4 views and a 100% accuracy. Note: Although our algorithm outputted a 100% accuracy, it is much more likely that the robot only sees about 99% or greater. This error could be due to comparison inaccuracies or a lack of precision within certain calculations in the algorithm.



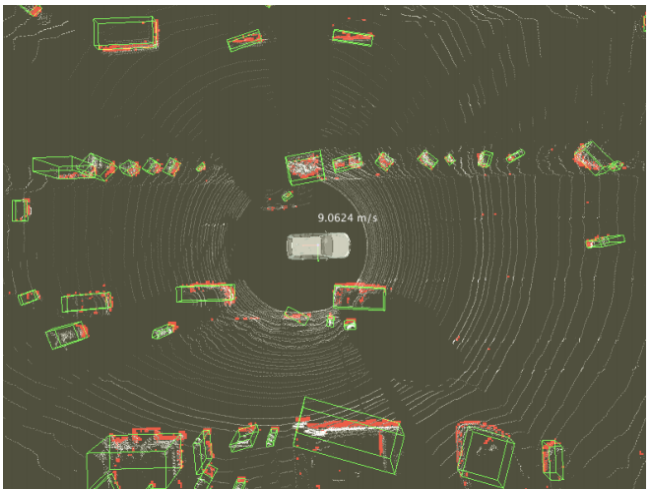
From the images above, it is reasonable to conclude that the views considered optimal by our algorithm comply with our common sense. However, it is intriguing to see how each view adds a specific percentage of extra information to the rendering. More importantly, some views do not add significant amounts of information. As seen with the hand above, the robot could see 96.98% of the hand with two views, but required two more views to just another 0.46% of the hand.

In general, we found that two views will give a robot a decent perception of most objects. We define a decent perception of objects as a visual accuracy of at least 85%. This accuracy can be fine for object identification; however, for object recognition (a potential future application), the robot needs more complex visual data. For example, with the hand gesture, the robot could probably tell that there is a specific gesture with just three views. But for the robot to understand emotion in that hand gesture, it needs the extra two views. In the end, this algorithm shows us that object recognition requires more information about the nature of the object and the amount of complexity necessary for identification. Since more views involves energy consumption, this information is necessary to achieve the most cost-efficient data above a certain minimum.

Future Work

Object classification is the intended application of this research. Classification can be broken down into two parts: isolating key points of interest and then correctly identifying each object and its significance.

Figure 10: Three-dimensional point cloud with objects identified through segmentation techniques, represented by 3D bounding boxes (green).



Isolating Key Points

Feature extraction within dense point cloud datasets is important for robots to quickly notice surroundings. As in the aforementioned implementation of military robots, scouting robots used by Defense Advanced Research Projects Agency need to quickly place points together into a single point cloud that constitutes each object. Feature extraction is done by utilizing two-dimensional data sets to measure differences in key image properties. A difference in texture, background, or other feature could result in a change from one object to the next. These segmentation techniques can help a robot identify different objects within a point cloud, as seen in Figure 10. When considered cumulatively, the point clouds are translated into smaller, condensed images which are local to that specific object [Himmelsbach, Leuttel, Wuensche, 2009].

This form of feature extraction is similar to many library functions that utilize image analysis to identify people and humans. With these images, it is easier for a robot to correctly identify the object, its nature, its properties, and its significance [Ziafati, 2016].

Identifying the Object and Significance

Object identification requires taking three-dimensional data from the Point Cloud Dataset related to the single object and running it through an object classifier. For our research, the next step would be to train a classifier that could identify certain objects. For example, for a military robot, the classifier would be trained on datasets of common military weapons, resources, and key points of interest. On the other hand, the Segway robots in the Building Wide Intelligence lab would benefit from being able to recognize common objects found within the Gates-Dell Complex, such as white boards, chairs, tables and other robots. This form of object recognition would help facilitate a truly seamless human-like tour experience for the Segway robots.

The training data set we would test it against would require examples of positive data and negative data. As we found in the "Point feature extraction on 3D range scans taking into account object boundaries" paper by Steder, Rusu, Konolige, and Burgard, these positive data sets would reflect correct examples of the object and negative data would reflect incorrect examples of the object. By giving the classifier examples of correct data and incorrect data, it would eventually utilize heuristics and commonalities spotted between the correct data. In addition, it would avoid commonalities between the incorrect data. In this manner, the classifier would allow us to identify important objects for the object to eventually act upon and interact with the environment [Aldoma *et al.*, 2012].

Acting on Objects

Upon identifying the item, the robot should be able to act on the object. Although the current Building Wide Intelligence Segway bots have some ability to manipulate objects and characterize objects from these interactions, these interactions can be improved through visual feedback. Ideally, a robot could see an object, recognize it, and then know how to use that object. For example, the robot could see a squishy ball, recognize it, and know that it can squish the ball. This function would require training the robot through reinforcement learning. The robot would be allowed to interact with the item in any ways and good interactions would be rewarded while bad interactions will be discouraged. In this manner, the robot would lean towards the favorable interactions, and over time learn how to interact with a specific object. Upon training with many different objects and then linking those actions to the images, the robot should be able to interact with its surroundings. For the Building Wide Intelligence robots, specifically, this could mean pushing the button to call an elevator or moving a chair out of the way.

This sequence of actions is very important for allowing the robot to interact with its surroundings. In the military application explored above, the robot would be able to quickly scout the environment and act upon certain actionable objects. However, for portable robots, it is important to minimize the cost of these actions.

Robot Movement for Different Size Objects

In minimizing costs, the robot needs to accurately estimate the cost for a considered course of action. In the case of different sizes of objects, this may require moving the camera, using the arm to move the object around, or driving around the object. If the object is a wall, the camera has to move. If the object is a table, the robot has to move. If the object is a small stuffed animal, the robot has to move the object with the arm. Each of these requires different actions that use different amounts of energy. In the cost estimation we use for our data in this paper, we assume that the robot moves around the object. However, that may not always be the most optimal means of visualizing the object. Using the arm takes way less energy than moving a Segway base [Heinzmann, Taylor, 2007]. As such, we would include accurate estimation of cost for different modes of interaction [Mohammed, Schmidt, Wang, Gao, 2014].

This accurate estimation would allow us to generalize our algorithm for finding the most optimal set of views for all forms of environmental interaction. This is the next step in more accurately identifying what is indeed the most optimal set of viewpoints.

Conclusion

Through this paper we explore determining the most efficient set of camera positions for a mobile robot to create a complete three-dimensional scan of an object. We explore the development of a general procedure for scanning an unknown object through our testing in simulation. We take fully developed three-dimensional scans and visualize, through rViz, how much of the object our procedure would capture as well as the cost of what we determine is the best set of those views. Through rviz we may also visualize the views in order to make generalizations. The biggest challenge we faced was adding the factor of angles to our visualization of the objects. What our biggest challenge in simulation was that the PCD files that we found had a lot of space between points. This space produced false results for our method of determining whether or not a point is can be seen from a view. The problem arose when the point was on the other side of the object from the view, but could still appear to be viewable because there was nothing blocking the view from the point. Another point of difficulty that we encountered was factoring in the angle between the point and the camera view. This was to account for the range of angles that the camera could see from a given orientation.

Overall, we were able to complete our goals while only sacrificing a few features we wished to incorporate.

Our project was entirely centered around simulation with key features made to be easily adaptable for implementation with the Building Wide Intelligence Segway robots at The University of Texas at Austin. We were able to generate views around an object, collect the required data, and determine the most effective set of these views. We were able to add the factor of cost as well. We were not able to go as in-depth with the cost function as we would have liked. This is because we did not factor in whether or not the object would be picked up by the robot or remain where it was found. Overall this project can be used as a stepping stone for further derivations of trends and functions.

References

Aldoma, A., Marton, Z., Tombari, F., Wohlkinger, W., Potthast, C., Zeisl, B., . . . Vincze, M. (2012, September 10). Three-Dimensional Object Recognition and 6 DoF Pose Estimation. *IEEE Robotics & Automation Magazine*, 80-90.

Bhat, S., & Meenakshi, M. (2014). *Vision Based Robotic System for Military Applications* (Rep.). IEEE.

Elegbede, A., Chu, C., Adjallah, K., & Yalaoui, F. (2003). *Reliability allocation through cost minimization* (1st ed., Vol. 52, IEEE Transactions on Reliability, pp. 106-111, Rep.). IEEE.

Faugeras, O. (1993). *Three-dimensional Computer Vision: A Geometric Viewpoint*. Boston, MA: Massachusetts Institute of Technology.

Heinzmann, J. D., & Taylor, B. M. (2007). *The Role of the Segway Personal Transporter (PT) in Emissions Reduction and Energy Efficiency* (pp. 4-5, Rep.). Segway.

Himmelsbach, M., Luettel, T., & Wuencshe, H. (2009). *Real-time object classification in 3D point clouds using point feature histograms* (Intelligent Robots and Systems (IROS), Rep.). IEEE.

Mohammed, A., Schmidt, B., Wang, L., & Gao, L. (2014). *Minimizing Energy Consumption for Robot Arm Movement* (pp. 400-405, Rep.). Elsevier.

Shaker, S., & Wise, A. (1987). *War without men. Robots on the future battlefield*. Elmsford, NY: Pergamon Books.

Steder, B., Rusu, R. B., Konolige, K., & Burgard, W. (2011). *Point Feature Extraction on 3D Range Scans Taking into Account Objesct Boundaries* (Robotics and Automation (ICRA), Rep.). IEEE.

Ziafati, P. (2016, February 12). Face_recognition.