Online 3D Object Mesh Pose Recognition: Midterm Report

Mike Hagenow (mhagenow 9079441078) and Kevin Welsh (kpwelsh 9081472293)

April 1, 2020

Abstract

Many of the existing methods for 3D object pose recognition from point clouds have deficits that make them not ideal for a aviation manufacturing environment. In this work, we propose extensions to existing methods for 3D object recognition. Our original plan was to create a method we called Face-Based-Features RANSAC where we create mesh correspondences between points and faces from a mesh, rather than sampling the face of the mesh and doing point to point correspondences as is typical of existing methods. We have successfully implemented and tested this algorithm. Fundamental limits in performance have led us to pursue a new method which utilizes a sparse medial axis representation and an informed variation of Iterative Closest Point (ICP) to fit arbitrary meshes with a significantly reduced runtime. So far, this method has been successful in recognizing and fitting our test meshes. Looking forward, we are excited to improve the performance and robustness to error and occlusions and believe we are on track to developing a new method that can be used by the community in arbitrary 3D mesh recognition.

1 Overview

1.0.1 Current State

To this date, we have made substantial progress both in terms of implementing and understanding existing mesh recognition methods as well as implementing and testing our own novel algorithms. We have developed a method that uses face-based features and a RANSAC kernel to recognize arbitrary meshes. While this method can find geometries given sufficient time, our use case for online recognition demands a faster solution. To address this, we have developed a prototype of a new method which uses medial axis matching and a variant of Iterative Closest Point (ICP) to determine geometry in a more deterministic amount of time. Initial results show promise in recognizing meshes from a point cloud. The additional sections provide more substantial detail about each of the methods explored thus far, including results and technical details. An updated timeline and our proposed comparison appear at the end of this document.

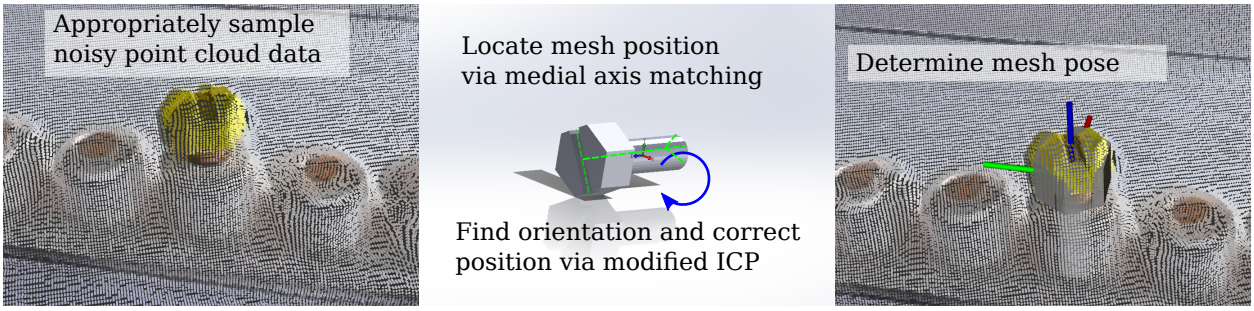

Figure 1: General problem statement to fit an arbitrary mesh to noisy point cloud data. Our method uses the medial axis as a geometric feature for correspondence and uses a modified version of Iterative Closest Point (ICP) for orientation and refinement.

1.0.2 Difficulties

The main difficulty of the project stems from the unknown of pursuing a novel algorithm for mesh pose recognition. Otherwise, while implementing existing methods for comparison and insight, we have faced some struggle with missing detail in technical papers as well some instances where formulas were actually incorrect in published literature. Luckily, we were able to fix problems and develop our own ideas for missing parts of the algorithms. Our only other issue has been in implementing comparison methods. For our arbitrary geometric mesh comparison method, we were able to clone a library based on the state of the art published literature, however, it rarely finds meshes even in simple point cloud scenes. We will further investigate and possibly contact the authors prior to our final comparison.

1.0.3 Next Steps

Based on the promising results so far, we believe for the final project deadline, we will be able to deliver a python package for our 3D pose recognition method. So far, most of our testing has been on simulated scenes and not images taken from real point clouds. We plan to add more robustness to deal with the expected noise and occlusions that arise from moving to real data, as well as improve scalabiliy and performance as required for a realistic scene.

2 Details of Progress

2.1 Efficient RANSAC

To explore fitting geometry from point cloud data, we developed a form of Efficient RANSAC to better understand the limitations and strengths of the method. We have developed a prototype that is capable of fitting two of the primitive geometries: spheres and cylinders

2.1.1 Implementation Details

We have provided an end-to-end python implementation for a modified version of the Efficient RANSAC method described in [5]. This is a module that takes in a point cloud and will plot the point cloud using matplotlib along with all of the identified primitive geometry. Figure 2 shows one example of the results. In our testing, we used point clouds with 5000 sampled points. RANSAC offers probabalistic guarantees to determine how likely a particular geometry can be identified based on how many points need to be identified and what percentage of the overall scene is represented by an object. The probability of selecting the correct points for a shape parameterized by k points, where the shape consists of n of the total N samples in the point cloud can be described as:

$$P(n) = (\frac{n}{N})^k \tag{1}$$

As described in [5], this can be rearranged to determine the number of iterations, T, that are required to detect shapes with a certain probability, p_t:

$$T \geq \frac{ln(1-p_t)}{ln(1-P(n)} \tag{2}$$

Based on the probabilistic model and empirical testing with our models, we set our system to run 1000 iterations. A different set of meshes or different-sized point cloud would require a different choice. For each iteration, a hypothesis is generated. In order to validate hypotheses, we count inliers and normal-vector consistency, described in more detail in Section 2.1.2. To decide whether a hypothesis should be accepted, we apply a threshold for consistent points that is based on the surface area of the candidate shape (i.e., an approximation for how many sampled points might be in the cloud). Finally, we only keep shapes if there is not already a shape at approximately the same center point. While the original report removes inliers to improve subsequent performance (by lowering N), we chose a more simple method that accomplishes the same result.

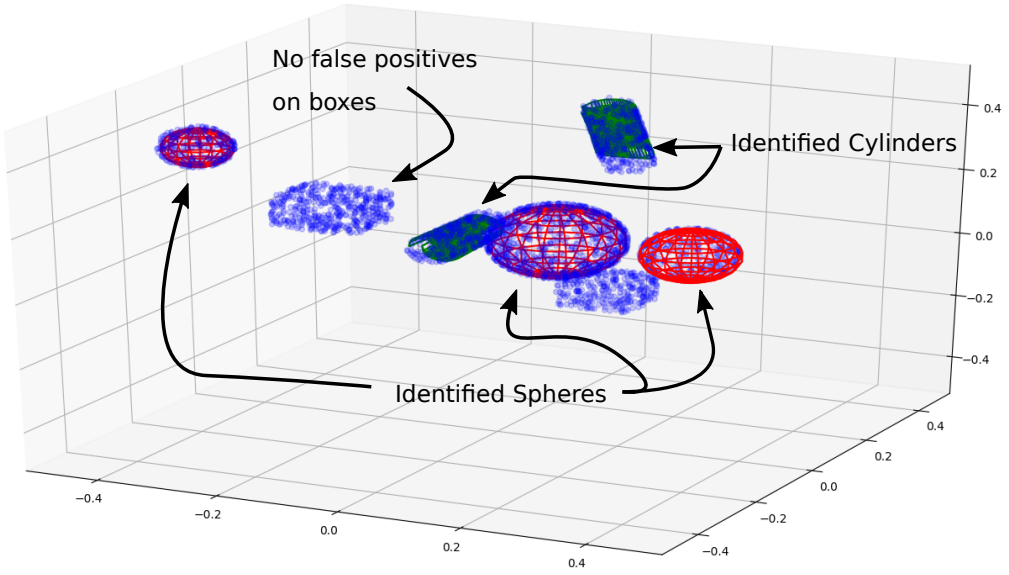

Figure 2: Our implementation of the Efficient RANSAC algorithm successfully identifies spheres and cylinders present in the scene. These results are generated from 1000 iterations. Despite other random objects in the scene, our scoring algorithm does not incorrectly classify the boxes. Note: the cylinder equation does not provide bounds for the cylinder length. The bounds are instead estimated from the inliers.

2.1.2 Lessons Learned

By creating a prototype of the efficient RANSAC platform, we were able to develop an intuition for the strengths and weaknesses of a RANSAC-based method. From our testing, we were able to learn two main lessons that helped to inform our novel algorithms.

Geometric relationships are often skewed by practical measurement error: The theory of the primitive recognition is correct (i.e., one can develop relationships to recognize geometric primitives using simple geometric relationships involving sampled points and their respective normal directions). However, we find that in practice, often small errors in the positions and particularly in the estimated normal directions can lead to requiring many sets of samples on the geometry before a reasonable model can be estimated. In other words, these simple geometric relationships are not robust to practical measurement noise. We present a case study for the spherical model as an example.

The spherical model is fully defined with two points and two corresponding normal directions. Since the normals are tangent to the surface of the sphere, the center of the sphere can be recovered by looking at the intersection of the lines created from the points and their normals. To add a small amount of robustness, the midpoint of the shortest distance between these lines is taken as the center which still allows the center to be found in the case when the normals do not properly intersect. The geometric relationship is shown in Figure 3. These geometric relationships are described by the following:

$$x_1, y_1 = \arg\min_{x_1, y_1} \|(x_1\mathbf{n}_1 + \mathbf{p}_1) - (x_2\mathbf{n}_2 + \mathbf{p}_2)\| \tag{3}$$

$$\mathbf{c} = ((x_1\mathbf{n}_1 + \mathbf{p}_1) + (x_2\mathbf{n}_2 + \mathbf{p}_2))/2 \tag{4}$$

$$r = (\|\mathbf{p}_1 - \mathbf{c}\| + \|\mathbf{p}_2 - \mathbf{c}\|)/2 \tag{5}$$

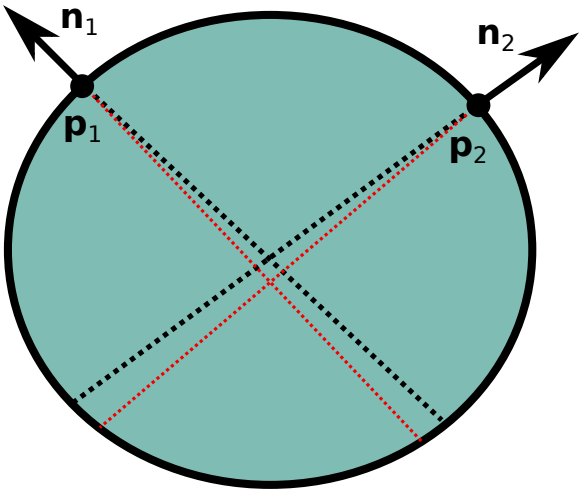

Figure 3: 2D visualization of how the sphere center point is determined based on two points and their normals. Red dotted lines show that for a few degrees of error in the normal, the resulting center can become inaccurate.

To evaluate, we test a point cloud of 1000 points which is sampled from an perfect sphere model (i.e., all points should be perfectly on the sphere). The normals are estimated from nearest neighbors as is common for a point cloud approach. When we run 1000 iterations of RANSAC, only 35 percent properly recover the sphere. This problem is further exacerbated when the two points are close by on the sphere leading to similar normal directions. To simulate this, we ran an evaluation where we select the second random point from a small neighborhood of the first point. In this case, the RANSAC only identifies the correct sphere 5 percent of the time. There is a trade off between robustness of pose estimation and likelihood of the sampled set of points all belonging to the same object. Our takeaway is that to keep the solution tractable, it is often necessary to limit the combinatorial nature of problems. As such, we require a robust refinement step that can function in spite of potentially inaccurate poses.

Scoring methods for a candidate shape: In the efficient RANSAC method, three main criteria are used for scoring: a calculation of consistent points within a small tolerance of the hypothesis mesh face, the normal of these consistent points compared to the normal of the mesh, and a calculation of the largest connected component of consistent points. The first two conditions, leading to a consistent set of points P_ψ for a hypothesis ψ, can be summarized as:

$$P_\psi = \{p | p \in P \wedge d(\psi, p) < \epsilon \wedge \arccos(|n(p)n(\psi, p)|) < \alpha\} \tag{6}$$

While we did not use a connected components calculation in our efficient RANSAC algorithm due to a lack of detail in the original reference, we were inspired to use a similar concept later on in our medial axis matching algorithm. For scoring in our novel algorithm, we use the idea of mesh inliers (i.e., points close to the mesh face), but also extend to include the concept of outliers, which we consider as points that would be within the mesh geometry. The resulting total score that we use to accept or reject a hypothesis pose is a weighted sum of both inliers and outliers.

2.2 Face-Based Features RANSAC

The first algorithm that we developed uses a RANSAC-type method to try and fit an arbitrary geometry mesh. While other previous methods have extended RANSAC methods to arbitrary geometry, our key differentiation was to use face-based correspondences instead of sampling mesh faces and doing point-to-point correspondences with the point cloud. Additional details about the method, features, and results are found below.

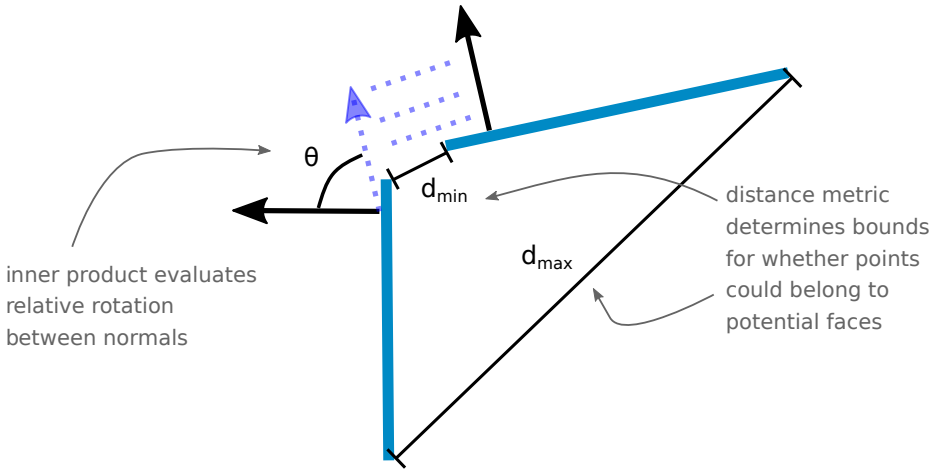

Figure 4: 2D Visualization exemplifies how features are computed for two mesh faces. The position of the two normal vectors exemplifies why a distance bound is needed for points on a face.

2.2.1 Implementation Details

We provide an end-to-end implementation in python. The system takes a PLY mesh and a JSON point cloud and identifies where the mesh occurs within the scene.

To be able to determine a the pose of a mesh from a face correspondence, we exploit two fundamental relationships between a set of faces and the pose (origin and rotation) of the mesh. Given $\vec{f_i}$ and $\vec{n}_{fi}$, a vector from the origin to a point on the face and the face normal in mesh space, along with $\vec{p_i}$ and $\vec{n}_i$, a point on the corresponding face in the scene and the scene normal, the following relations hold:

$$\left[\vec{n}_1|\vec{n}_2|\vec{n}_3\right] = R\left[\vec{n}_{f1}|\vec{n}_{f2}|\vec{n}_{f3}\right] \tag{7}$$

$$\left[\begin{array}{c}\vec{n}_1\\\hline\vec{n}_2\\\hline\vec{n}_3\end{array}\right]\vec{o} = \left[\begin{array}{c}\vec{p}_1\cdot\vec{n}_1 - \vec{f_1}\cdot\vec{n}_{f1}\\ \vec{p}_2\cdot\vec{n}_2 - \vec{f_2}\cdot\vec{n}_{f2}\\ \vec{p}_3\cdot\vec{n}_3 - \vec{f_3}\cdot\vec{n}_{f3}\end{array}\right] \tag{8}$$

These relations allow us to determine the position and orientation of a mesh in the scene, provided we have corresponded three points with three separate faces on the mesh. This is in contrast to the methodology common to existing algorithms, which is to first sample points from the mesh and attempt to correspond points to points. Our method works by trying to match triples of points with triples of faces. This is done by computing features of both that can be compared. We use a total of 6 features to compute potential face correspondences. The first three are the inner products of normals. Since each point on the point cloud has a corresponding normal taken from nearest neighbors, this gives a metric that is agnostic to where a point lies on a given face. By using the inner product of normals, we functionally get a metric of rotation between points. Second, we compute a distance metric. In other methods, this would represent the distance between the sampled mesh points. Since we want to compare between the mesh face and a point in the scene, we instead compute a distance bound. This is equivalent to finding what sets of three faces could represent these three points and their positions, given that the points could be anywhere on the face. Naturally, these leads to fairly conservative bounds, but avoids the issue of sampling the mesh and relying on points on the mesh face accurately corresponding to points in the scene. Figure 4 provides a visualization for these features for a set of meshes.

In order to store this information, we leverage an existing implementation of an R-tree which allows us to structure our mesh features as a 6-dimensional hyper-rectangle. For any given set of three points, we can then compute the 6 dimensional feature vector, and quickly find all of the mesh features that contain the point. Then, we evaluate the matches to see whether any yields a pose hypothesis that is consistent with the point cloud (e.g., scoring algorithm).

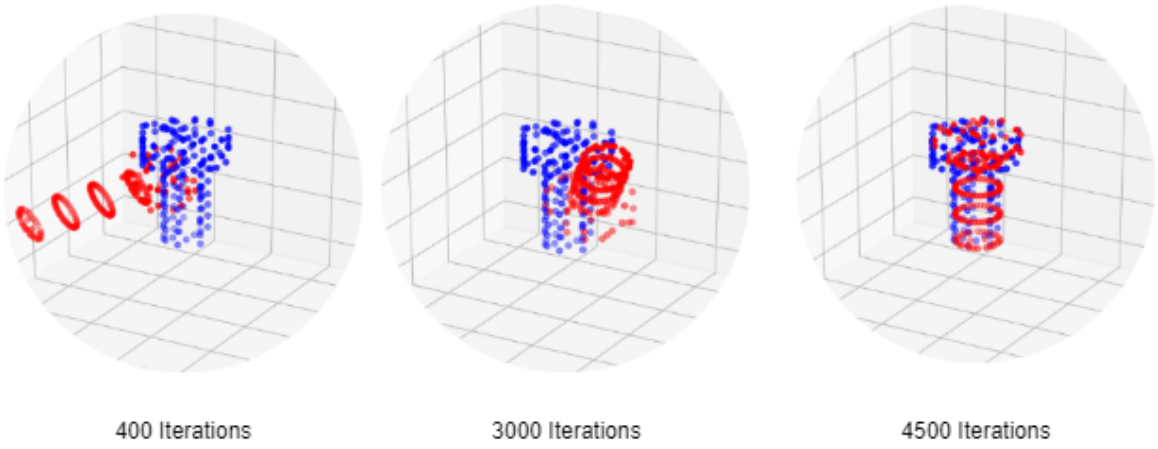

Figure 5: Best identified mesh pose in read after 400, 3000 and 4500 iterations operating on the scene cloud in blue. This example was taken from a simulated cloud without additional noise. While this algorithm can find a good pose in the presence of noise, it takes substantially more iterations.

2.2.2 Results and Limitations

We tested our algorithm on a variety of simulated point cloud scenes, some of which included noise. We focused our testing on locating a simple screw model in a scene with 0-1 other objects. Even in cases of realistic noise (10% of model scale), the algorithm is successfully able to locate the screw pose.

The main limitation is the performance. Figure 5 demonstrates that our algorithm can identify the screw model, but often takes many RANSAC iterations to properly find the model, even in a relatively sparse scene. As a reminder, one of our goals is to locate scene objects quickly to be used in an online interface in manufacturing environments. While our features represent technically sound relations that help to identify point-face correspondences, there can still be several thousand face combination matches for given set of three points. As a result, the performance succumbs to the combinatorial nature of the problem. In other algorithms, such as Papazov et al. [4], additional constraints are added, such as sampling points from a known radius, however, we thought such heuristics diverged from our idea of developing a more general purpose algorithm.

In summary, we were successful in our goal of implementing a face-based-feature variant of RANSAC for pose estimation, but limited performance has led us to design a different, more promising algorithm described below.

2.3 Current Algorithm

The most important thing that was learned from implementing a version of Face-Based RANSAC is that the combinatorial problem of corresponding points to faces is far too inefficient. With this in mind, we decided to avoid developing scene-model correspondences over large sets of points. As a result, our Medial Axis Matching algorithm is focused on deriving a very small set (<100) of key points from the scene, as well as a very small set (<10) of key points from the model.

To achieve this goal, our algorithm leverages the Medial Axis Transform (MAT) of a given mesh. Recently, there has been evidence that the medial axis of shape plays a role in human shape recognition and cognition[3]. Similarly built upon this intuition, MAT-Net attempts to leverage the medial axis for 3D pose recognition using a neural network[2]. In contrast to the more popular neural network approaches, ours takes inspiration from Boluk et. al.[1], where a mesh is reduced to a set of critical points of the medial axis. We iterate on this idea by applying it to 3D environments, and extend the concept of the MAT to partially observed objects.

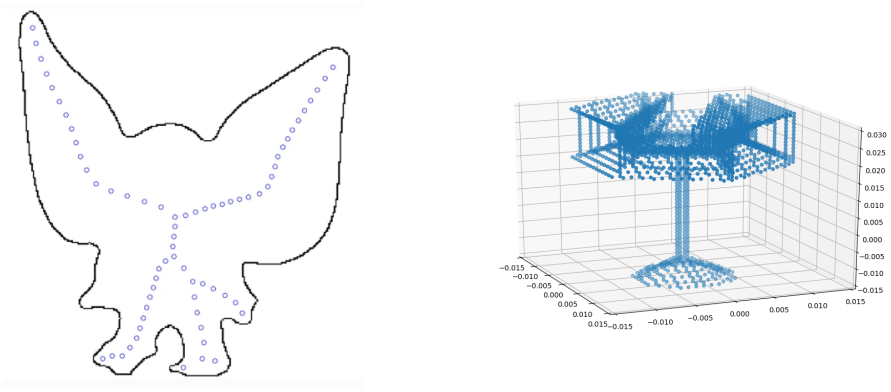

Figure 6: Discretized representation of the medial axis. a) A 2d representation (figure adapated from Boluk et. al. [1]). b) A discretized medial axis generated from our toy screw model.

2.3.1 Implementation Details

The algorithm consists of several steps. First, a set of hypotheses are generated by corresponding a sparse form of the medial axis between the mesh and the point cloud. These give candidate positions for the meshes in the scene. Using modified ICP, the orientation of the mesh is recovered and the overall pose is refined based on the point cloud. Finally, hypothesizes are validated via a scoring method similar to existing algorithms. The following sections provide more detail for each step of the method.

Medial Axis Generation: The medial axis of a shape is the set of all points that have more than one closest point on the shape. The first step of our algorithm is to generate this set for both the reference model and the scene point cloud. To do this, we draw on the algorithms developed by Siddiqi et al.[6]. We use an Octree as our dynamic spatial resolution primitive, where we choose to expand node of the tree if we think it may contain a segment of the medial axis. This is determined by first computing the local distance field (i.e., the minimum distance to the mesh/point cloud), and looking at the net flux of the gradient through that Octree node. Siddiqi et al. shows that points on the medial axis manifest a negative flux (a source of the vector field). Similarly, it can be shown that an Octree node contains a medial axis point only if it contains a point on the surface of the mesh/point cloud, or it has a negative net flux itself. This allows us to discretely resolve the medial axis to arbitrary precision.

Critical Point Detection: At first glance, identifying a distcretized medial axis has merely transformed the problem from matching two point clouds sampled from the surface to matching two point clouds sampled from the interior. However, there is a key difference: we are now able to shrink the point clouds appropriately down to a handful of keypoints. This is done via a novel, three-part erosion scheme. The first step is to transform the point cloud into a set of connected graphs by connecting all adjacent points. This forms the basis of a noisy medial axis graph. The second step is to remove the outliers and noise, by successively removing all nodes of the graph that have fewer than 8 neighbors. Then, the last step is to filter each component based on size, and continue removing the least connected nodes until there is only one of each component remaining. These remaining points are classified as "key points". For the purpose of key point correspondence, we do the same procedure to the mesh we are searching for. Finally, we can find potential scene-model correspondences by matching the key points based on their distance to the surface. This typically results in several hypothesis model positions in the scene.

Iterative Closest Point: Once we have identified a set of hypothesis positions, we need to find the optimal orientation and correct any error in the position of the hypothesis mesh. To do so, we developed an implementation of the Iterative Closest Point (ICP) algorithm, which allows us to both refine the positions and estimate the optimal orientation. The basic ICP algorithm works by computing the minimum distance between a point in the scene with each point on the mesh. From these sets of corresponding points, an

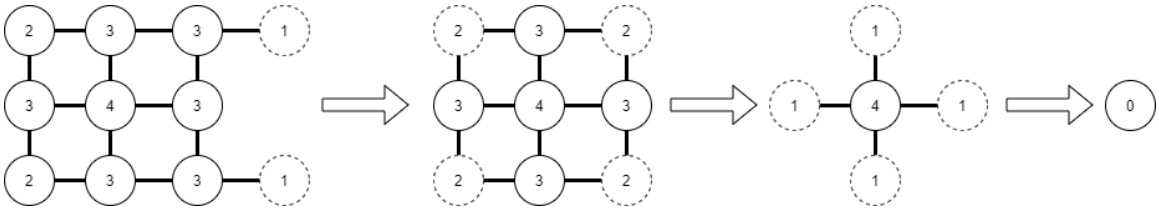

Figure 7: Example of pruning the graph points of a medial axis. At each step, the least connected nodes are eroded away until there is small graph that represents highly connected components.

optimal rotation and translation can be determined using least squares (e.g., SVD). This process is performed iteratively until convergence or a max number of iterations. There are a few design choices in implementing the algorithm. The first is the maximum distance to consider for potential match points. We determine this based on the radius of the mesh. Second, the points can be given individual weights as part of the least squares. We have experimented with distance-based weighting as well as weighting based on the angle between normal vectors as seen in Equations 9-10.

$$w_i = 1 - \frac{|d_i|}{d_{max}} \tag{9}$$

$$w_i = |\mathbf{n}_{closest,i} \cdot \mathbf{n}_{mesh,i}| \tag{10}$$

where d_i is the closest distance for an individual mesh face point, d_{max} is the max distance considered for ICP, and $n_{closest}$ and n_{mesh} are the normals for the closest point cloud point and mesh respectively. Empirically in our testing so far, we note similar results regardless of the weighting function.

The ICP algorithm is typically used to refine an initial pose estimate (i.e., position and orientation). In our algorithm, the medial axis matching does not give an estimate of orientation. As the ICP algorithm is sensitive to local minima in the least squares fitting, we added orientation random-restarts to help converge on the proper pose.

Looking forward, there are still two main areas to improve with respect to the ICP algorithm. The first is to add robustness with respect to occlusions. We believe that proper restarts and exploration of the search space can help to improve issues. Additionally, currently the points that are used from the mesh represent the center of each of the faces, which does not necessarily align with the points on the point cloud. To make these point correspondences more compatible with our face-based correspondence mentality, we are interested in exploring weighting functions that account for the bound of faces (as described earlier).

Hypothesis Validation: Finally, once we have a set of refined pose hypotheses, we merely need to accept or reject them. To note here, because of the way we generated key points, there are very few hypotheses that intersect with each other. Our working assumption thus far has been that a hypothesis that fits the surrounding cloud well will not have any other conflicting hypotheses. As a result, the validation stage is a straightforward procedure inspired by our RANSAC beginnings: we simply count the number of "inliers" (points that lie on the surface of the proposed model instance) and "outliers" (points that lie within the proposed model instance) and accept or reject based on a binary thresholding of those values.

2.3.2 Results

We have completed some preliminary testing of our algorithm. So far, we have only focused on recognizing the screw model as one prototypical example that fits within our use case. In simple situations, the algorithm can successfully identify the screw. Some examples of the recognition are provided in Figure 8. As per our original plan, we are still working on iterating and making our algorithm more robust. Specifically, we plan to start testing additional meshes and occluded scenes promptly. The sample results indicate that the ICP rotation can still fail in certain cases and recognition in noisy occluded scenes can make it difficult to precisely locate the mesh.

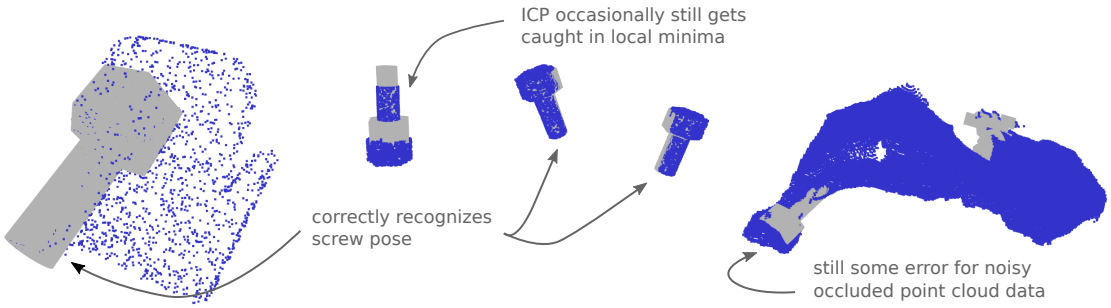

Figure 8: Example early results from the medial axis algorithm. Left: the algorithm correctly finds a screw next to a model glove. Middle: the algorithm correctly identifies the pose of 2 screws. The third gets caught in a local minimum. Right: a real point cloud is evaluated. The screw is located and correctly rotated, but still has some error due to occlusion and noise (Note: validation is turned off in this example which is why other false-positives appear).

3 Proposed Evaluation and Timeline

3.1 Timeline

To evaluate our final method, we propose to provide evidence that it can identify arbitrary meshes (e.g., 10 different meshes) in a variety of generated and real point cloud scenes. We will report both successes and limitations in order to properly identify where this method can be helpful for other uses. We will also provide a comparison to three methods: our implementation of efficient RANSAC, an existing RANSAC-based arbitrary mesh recognition, and a neural net approach: PointNet++.

3.1.1 Efficient RANSAC

We will run regression testing for a variety of perfect and noise-injected scenes to see how often efficient ransac and our medial axis method can identify random combinations and poses of spheres and cylinders within the scene.

3.1.2 ObjRecRANSAC

ObjRecRANSAC is an implementation of the geometry-based arbitrary mesh recognition method proposed in Papazov er al. [4]. Since our intention is to use our algorithm for a manufacturing environment, we will compare 10 test manufacturing scenes as well as one actual point cloud (we can't get anymore now that we are locked out of the building). Each scene will have some sort of standard industrial element (e.g., screw, bolt, tool) in it and we will directly compare the performance (i.e., runtime) and a confusion matrix of the results of each method.

3.1.3 PointNet++

We also desire to compare to a neural net approach. To avoid retraining, we will use our method for comparing 5 of the objects that the neural net can recognize. Similar to above, we will report performance results as well as a confusion matrix of accuracy for each method.

3.2 Timeline

Figure 9 shows our original project proposal timeline for reference. We have successfully implemented our version of Efficient RANSAC and our version of FBF-RANSAC as desired by the midterm report. In addition to the original plan, we have also developed our new Medial-Axis based method which we will use for final

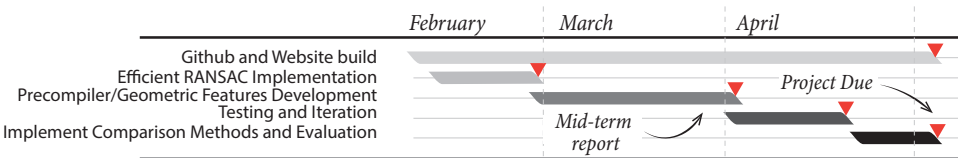

Figure 9: Original project timeline.

comparison. We are on track to deliver our final method with 3 comparisons to existing methods by the project due date.

References

[1] Demirci M.F. Boluk, A. Object recognition based on critical nodes. *Pattern Anal Applic*, 22:147–163, 2019. doi: 10.1007/s10044-018-00777-w.

[2] Jianwei Hu, Bin Wang, Lihui Qian, Yiling Pan, Xiaohu Guo, Lingjie Liu, and Wenping Wang. Mat-net: Medial axis transform network for 3d object recognition. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 774–781. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/109. URL `https://doi.org/10.24963/ijcai.2019/109`.

[3] Vladislav Ayzenberg; Yunxiao Chen; Sami R. Yousif; Stella F. Lourenco. Skeletal representations of shape in human vision: Evidence for a pruned medial axis model. *Journal of Vision*, 19, 06 2019. doi: 10.1167/19.6.6.

[4] Chavdar Papazov, Sami Haddadin, Sven Parusel, Kai Krieger, and Darius Burschka. Rigid 3d geometry matching for grasping of known objects in cluttered scenes. *The International Journal of Robotics Research*, 31(4):538–553, 2012. doi: 10.1177/0278364911436019. URL `https://doi.org/10.1177/0278364911436019`.

[5] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. *Comput. Graph. Forum*, 26:214–226, 06 2007. doi: 10.1111/j.1467-8659.2007.01016.x.

[6] Bouix S. Tannenbaum A. et al. Siddiqi, K. Hamilton-jacobi skeletons. *International Journal of Computer Vision*, 48:215–231, 2002. doi: 10.1023/A:1016376116653.