A report on an implementation of RANSAC for sphere detection to derive shape labels from depth data

Martin Brown
Stanford University
450 Serra Mall, Stanford, CA 94305
martinbr@stanford.edu

Abstract

This project aims to implement a non-learning-based shape recognition routine for spheres of user-defined radii. In particular, I implement a RANSAC-based routine on the depth channel of an RGB-D camera. Afterward, I use the RANSAC sphere detector to re-project the detected shapes back into the RGB image plane as labeled images patches that can be used for downstream processing, such as training a classifier in a supervised manner. The ultimate intent of this project was to explore the implementation of a RANSAC detector and how one might utilize non-learning shape detectors to bootstrap the creation of learning-based shape detectors in a self-supervised manner. There are various benefits to this approach. Specific to this project, we could use the labeled patches to train an RGB classifier in domains where infrared depth channel alone may be inadequate or non-existent, including at very close or far distances, in environments with bright lighting, on shapes composed of Lambertian materials, or on images without a depth channel. The RGB detector could also be used as an additional heuristic to better target the RANSAC detector, reducing computational complexity of the latter.

1. Introduction

The inspiration for this project originates from a mixture of ideas encountered during the lectures of CS 231A on representation learning, in combination with a set of interesting papers on shape analysis I encountered while researching potential project ideas.

1.1. Understandable Visual Representation learning

While there are certainly philosophical arguments as to whether aspects of artificial intelligence or meta-cognition are inherently beyond the limits of conscious human understanding, this project proceeds under the intuition that there are aspects of visual cognition that rely on a hierarchical symbolic representation of complex scenes or shapes. Furthermore this project conjectures that one of the lowest levels of symbolic representation would be simple platonic geometric solids such as spheres, cylinders, and cuboids. Figure 1 highlights some inspiration for this conjecture.

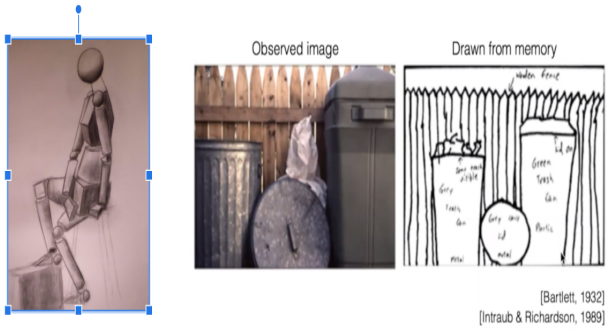

Figure 1. Left: A human figure rendered in the technique of ideal solids. Right: A recreation of an image from human memory

1.1.1 Learning-Based Methods

Modern learning-based methods for representation learning, particularly models using large neural networks composed of thousands or millions of parameters, suffer from the problem of understandability at an intuitive or 'common-sense' level. There has been work in the area of guiding the learning of the latent-space representation in

a way that is more understandable to humans [1], however the vector-based latent representation of most models has inherent limits of expressiveness due to it's fixed size. Furthermore, the data requirements for training a de-novo model are often prohibitive, although there are various techniques for working around this limitation. [4]

1.1.2 Models from First Principles

Conversely, the methods articulated in [2] and [3] start from a foundation of human-defined shapes for detection in human-created structures, and then deriving a graph-based scene description, similar to computer graphics, as the 'latent-space' representation of the visual data. This has a naturally hierarchical and extensible structure, but is limited by the need for a user to designate and define the catalog of desired shapes. There is also the need for various heuristics to whittle down the sample space of points to sample, otherwise the procedure would be computationally infeasible for general point clouds.

2. Problem statement

2.1. Data Set

The data set is composed of a series of RGB-D frames taken of a table strewn with various man-made objects that are approximately cylinders, cuboids, and spheres, including: Batteries of various types (AA, AAA, D, 9V), Children's blocks, Children's balls, and a few other children's toys in the form of these platonic shapes.

The objects are all capable of being grasped by a human with one hand. The frames are taken with an Intel Realsense D435i stereo depth camera. The objects are arranged in a mixed configuration and then photographed 10 times from various angles.

3. Technical Approach

3.1. Software Libraries

3.1.1 Open3D

I relying on the Open3D library [5] for the underlying computational geometry methods needed to go from the camera frames to point clouds and vertex normals. In particular, I utilize the following data structures:

Octree

KD-Tree

PointCloud

TriangleMesh

As well as various helper methods for cleaning and preprocessing the data. I also use the built-in RANSAC plane estimation tool as a first pass to sparsify the scene, which greatly improves the sphere extraction performance.

3.1.2 numpy

Numpy is used for mathematical computation including manipulation of the point cloud and vertex normal data for application of the RANSAC-based shape-fitting procedure, as well as re-projecting the detected spheres back to the image plane for patch labeling

3.2. RANSAC Shape Fitting

The approach outlined in [2] describes a seemingly straightforward method of determining the membership of points in the shape class based on the vertex normals of the sampled points within the point cloud. Cylinders, cuboids, and spheres, are each specified by 3 vertex normals to fit their respective shape models, and the shapes are then tested against the point cloud for inliers/outliers. However the devil is in the details, and there are numerous heuristics and thresholds that can be tweaked to optimize the accuracy and performance of the algorithm, many of which I discuss below.

3.2.1 Point-Cloud Hyper-parameters: Max Depth, Down-Sampling and Surface Normal Estimation

When converting from an RGB-D frame to a point cloud, one must decide on the maximum depth to truncate points beyond. This is a function of the camera accuracy and the scene at hand, and will influence the ultimate size of the point cloud. I chose 3.0 feet for my max depth since my image scene was a table-top collection of items and the camera's optimal depth range contains that depth.

Down-sampling was done with a voxel size of 0.01 based on manual experimentation. Qualitatively it appeared to capture the necessary surface detail for my spheres of interest (balls of radius 1.5 inches and above) yielding 20 surface points per sphere. However, it would likely be too sparse of a sampling for reliable detection of smaller objects such as a toy marble. For practical applications this is one of, if not the most significant hyper-parameter that can be tuned, assuming prior knowledge of the sizes of shapes being detected.

Surface normal estimation was done using Open3D's KDTreeSearchParamHybrid method with radius=0.1 and max nearest neighbors=30. This was a default recommendation from the documentation and I did not have time to investigate the implementation of this method or explore tweaking the hyper-parameters.

Also note that one has no guarantee on the orientation of the surface normals produced. To be sure the normals are in the intuitive 'outward' orientation for surfaces, we must explicitly orient them relative to the camera. This is performed by the orient$_n ormals_t owards_c amera_l ocation method in Open3D.$

3.2.2 Octree for Sampling and KD-Tree for Inlier Finding

[2] Describes a method of using an octree for sampling locally at different scales. In particular, they describe a sampling strategy that continuously updates the sampling distribution across octree levels based on prior successful shape matchings at that level. In the name of simplicity and expediency, I opted for a simplified form of sampling whereby after picking an initial point uniformly at random, I then sample the subsequent point from the leaf node indices of the initial point in the octree, i.e. the bottom-most level of the octree. The spheres I was attempting to detect were small relative to the scene and I don't believe sampling at larger level scales would have given any benefit for this particular application. My octree was initialized with max depth of 4 which yielded roughly 20 neighboring points per sampled leaf node.

A KD-Tree was used to quickly find the validation set in the point cloud for a given candidate sphere center and radius. The specific implementation was Open3D's KDTreeFlann.

3.2.3 Fitting the Sphere

Note that unlike [2], in my implementation I fit the spheres with two points and their surface normals, rather than three, again for simplicity and speed of execution.

The actual sphere fitting process is as follows. Given two points $p1, p2$ and their associated surface normals $n1, n2$ sampled via the octree-based method outlined above, find s, t which best satisfy the linear system of equations

$$\begin{bmatrix} n1 & -n2 \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = p2 - p1$$

(This is finding the intersection point, if it exists of the lines p1 + s*n1 and p2 + t*n2)

There are now several checks we perform with the least squares solution of this equation:

We check if the error is small (represented in my code by hyper-parameter $intersection_d if f_t olerance$). This tells us if the candidate lines even intersect.

We check that the candidate sphere normals are pointing 'outward', which requires that values for s and t in the intersection equation above are both negative. Note that this heuristic is possible because I'm only interested in actual spherical objects. If I was interested in negative space, such as a spherical depression, this heuristic would be discarded.

We calculate a center estimate based on the arithmetic mean of the estimated intersection point for each line, and similarly, a radius estimate. At this point we check that the distance from the center estimate of $p1$ and $p2$ falls within the radius$_d if f_t olerance hyper - parameter.$

We check the sphere radius is in the desired range. This is possible since we're only looking for balls of a particular size (a particular toy ball shape).

We now start checking the inlier coverage. We take the point cloud points within a requested search radius from the estimated sphere center. This is another hyper-parameter, hard-coded to $1.1 * radius_e stimate$. We need some slack in the radius estimate to account for noise in the depth channel.

We check for a minimum level of validation support i.e. whether there are sufficient points to have a useful inlier test. This is another hyper-parameter, set to 16 in the code.

This was chosen to be similar to the 20 points per toy ball observed in the down-sampled point cloud. Intuitively, this should track closely with the down-sampling density of the point cloud since this determines the approximate number of points the object of interest would reside in.

We check the validation set is within the radius estimate multiplied by a tolerance value. This is similar to the search radius when considering points on the exterior of the candidate sphere, but is also useful for ruling out candidate spheres with many internal points. The tolerance multiplier is another hyper-parameter, set to 0.05.

We check the angle tolerance of the validation normals compared to the projected surface normals.

Finally, we rule the candidate accepted if the inlier ratio exceeds 0.5, with 0.5 as yet another hyper-parameter.

During the sampling procedure we store all candidates, including duplicates and near-duplicates, whereas [2] removes the explained points from the point cloud once a candidate shape is accepted. This was done to get a better sense of the behavior of the implementation when tweaking the hyper-parameters. In the final re-projection of candidate spheres back to sample patches (described below), we perform a nearest-neighbor de-duping scheme.

3.2.4 Re-projection of Candidate Spheres to Image Patches

The re-projection works as follows:

For each candidate, check that the center of the sphere is not within a certain threshold of the centers of already seen spheres (this is the de-dupe step mentioned above).

For an unseen sphere candidate, create a triangle mesh with the given center and radius and convert to a sampled point cloud.

Using the original camera intrinsics to project the point cloud points to image plane coordinates.

Filter the projected image plane coordinates based on the original RGB frame dimensions.

Take the min and max of the row and column values of the projected image points and use them to select a square patch in the RGB frame.

4. Results

4.1. Removing large planes is critical

We struggled with getting acceptable results while the main table plane was part of the point cloud set. Part of the problem is the gradually undulating shape of the blanket the objects were placed on to reduce specularity, which let to many false positives for spheres above and below the table plane. Reducing the down-sampling may have helped distinguish the spheres further and allowed tighter hyper-parameters, at the cost of reduced performance. You can see the clear improvement after the plane is RANSAC'ed out. While there are still false positives, they are less numerous, and due to objects which actually have some spherical symmetry such as cylinders and cubes.

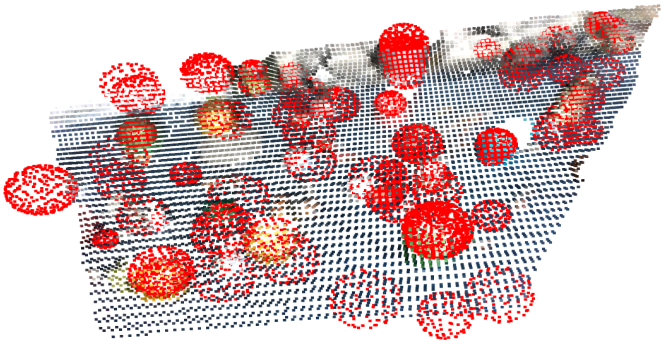

Figure 2. Sphere detection without plane segmentation

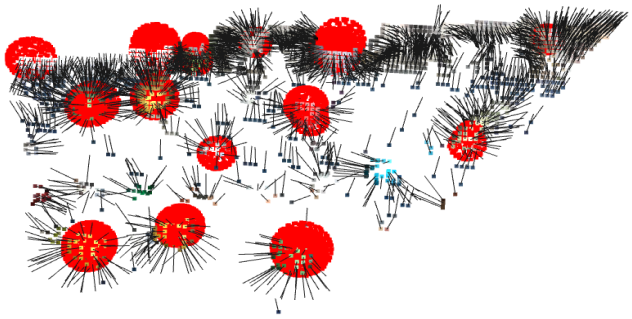

Figure 3. Sphere detection after segmenting out plane using Open3D RANSAC Plane Segmentation

4.2. Re-projecting detected spheres for labeling RGB patches

Reprojecting the sphere detections gives us at long last our labeled RGB sphere images, albeit with inaccuracies related

to incorrect shape detections of objects with spherical symmetry. There is also some over-cropping of the spheres due to the plane segmentation being a little too aggressive in some areas. This result shows the ability in principal to self-supervise RGB shape detectors, although more hyper-parameter tuning would be required to achieve reasonable accuracy.

Figure 4. Detected spheres from depth00003.png. Note that the bottom 3 figures are two cylinders and a cube respectively.

This project has given the author new appreciation for supervised learning with clean, labeled data.

4.3. Future Directions

I bit off a little more than I could chew with my proposal but I'm pleased with the prototype, showing it's mostly doing the right thing. For future development I would like to quantify the accuracy more rigorously across all the images and in different image scenes. I would also like to implement some of the additional shapes described in [2] and explore how one should go about optimally partitioning the point cloud based on the confidence of the various shape RANSAC detectors. Finally, I would continue to push the self-supervision angle of using the RANSAC labeling to train a supervised RGB classifier in tandem and in concert with the RANSAC detector. For example, the RGB classifier could be used to help tune the numerous RANSAC detector hyper-parameters.

References

[1] P. Esser, R. Rombach, and B. Ommer. A disentangling invertible interpretation network for explaining latent representations. *CVPR*.

[2] R. Schnabel, R. Wahl, and R. Klein. Efficient ransac for point-cloud shape detection. 2007.

[3] R. Schnabel, R. Wessel, R. Wahl, and R. Klein. Shape recognition in 3d point-clouds. 2008.

[4] S. J. P. Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 2009.

[5] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.